NAME

Sys::Virt::Domain − Represent & manage a libvirt guest domain

DESCRIPTION

The `Sys::Virt::Domain` module represents a guest domain managed by the virtual machine monitor.

METHODS

my `$id` = `$dom`−>**get_id()**

Returns an integer with a locally unique identifier for the domain.

my `$uuid` = `$dom`−>**get_uuid()**

Returns a 16 byte long string containing the raw globally unique identifier (UUID) for the domain.

my `$uuid` = `$dom`−>**get_uuid_string()**

Returns a printable string representation of the raw UUID, in the format 'XXXXXXXX−XXXX−XXXX−XXXX−XXXXXXXXXXXX'.

my `$name` = `$dom`−>**get_name()**

Returns a string with a locally unique name of the domain

my `$hostname` = `$dom`−>get_hostname($flags=0)

Returns a string representing the hostname of the guest. `$flags` can be zero or more of

Sys::Virt::Domain::GET_HOSTNAME_AGENT

Report the guest agent hostname

Sys::Virt::Domain::GET_HOSTNAME_LEASE

Report the DHCP lease hostname

my `$str` = `$dom`−>get_metadata($type, `$uri`, `$flags` =0)

Returns the metadata element of type `$type` associated with the domain. If `$type` is `Sys::Virt::Domain::METADATA_ELEMENT` then the `$uri` parameter specifies the XML namespace to retrieve, otherwise `$uri` should be `undef`. The optional `$flags` parameter defaults to zero.

`$dom`−>set_metadata($type, `$val`, `$key`, `$uri`, `$flags`=0)

Sets the metadata element of type `$type` to hold the value `$val`. If `$type` is `Sys::Virt::Domain::METADATA_ELEMENT` then the `$key` and `$uri` elements specify an XML namespace to use, otherwise they should both be `undef`. The optional `$flags` parameter defaults to zero.

`$dom`−>**is_active()**

Returns a true value if the domain is currently running

`$dom`−>**is_persistent()**

Returns a true value if the domain has a persistent configuration file defined

`$dom`−>**is_updated()**

Returns a true value if the domain is running and has a persistent configuration file defined that is out of date compared to the current live config.

my `$xml` = `$dom`−>get_xml_description($flags=0)

Returns an XML document containing a complete description of the domain's configuration. The optional `$flags` parameter controls generation of the XML document, defaulting to 0 if omitted. It can be one or more of the XML DUMP constants listed later in this document.

my `$type` = `$dom`−>**get_os_type()**

Returns a string containing the name of the OS type running within the domain.

`$dom`−>create($flags)

Start a domain whose configuration was previously defined using the `define_domain` method in Sys::Virt. The `$flags` parameter accepts one of the DOMAIN CREATION constants documented later, and defaults to 0 if omitted.

$dom->create_with_files($fds, $flags)
Start a domain whose configuration was previously defined using the define_domain method in Sys::Virt. The $fds parameter is an array of UNIX file descriptors which will be passed to the init process of the container. This is only supported with container based virtualization.The $flags parameter accepts one of the DOMAIN CREATION constants documented later, and defaults to 0 if omitted.

$dom->**undefine()**
Remove the configuration associated with a domain previously defined with the define_domain method in Sys::Virt. If the domain is running, you probably want to use the shutdown or destroy methods instead.

$dom->**suspend()**
Temporarily stop execution of the domain, allowing later continuation by calling the resume method.

$dom->**resume()**
Resume execution of a domain previously halted with the suspend method.

$dom->**pm_wakeup()**
Wakeup the guest from power management suspend state

$dom->pm_suspend_for_duration($target, $duration, $flags=0)
Tells the guest OS to enter the power management suspend state identified by $target. The $target parameter should be one of the NODE SUSPEND CONTANTS listed in Sys::Virt. The $duration specifies when the guest should automatically wakeup. The $flags parameter is optional and defaults to zero.

$dom->save($filename)
Take a snapshot of the domain's state and save the information to the file named in the $filename parameter. The domain can later be restored from this file with the restore_domain method on the Sys::Virt object.

$dom->managed_save($flags=0)
Take a snapshot of the domain's state and save the information to a managed save location. The domain will be automatically restored with this state when it is next started. The $flags parameter is unused and defaults to zero.

$bool = $dom->has_managed_save_image($flags=0)
Return a non-zero value if the domain has a managed save image that will be used at next start. The $flags parameter is unused and defaults to zero.

$dom->managed_save_remove($flags=0)
Remove the current managed save image, causing the guest to perform a full boot next time it is started. The $flags parameter is unused and defaults to zero.

$dom->managed_save_define_xml($xml, $flags=0)
Update the XML of the managed save image to $xml. The $flags parameter is unused and defaults to zero.

$xml = $dom->managed_save_get_xml_description($flags=0)
Get the XML in the managed save image. The $flags parameter accepts the following constants

Sys::Virt::Domain::SAVE_IMAGE_XML_SECURE
Include security sensitive information in the XML dump, such as passwords.

$dom->core_dump($filename[, $flags])
Trigger a core dump of the guest virtual machine, saving its memory image to $filename so it can be analysed by tools such as crash. The optional$flags flags parameter is currently unused and if omitted will default to 0.

$dom->core_dump_format($filename, $format, [, $flags])
Trigger a core dump of the guest virtual machine, saving its memory image to $filename so it can be analysed by tools such as crash. The$format parameter is one of the core dump format

constants. The optional`$flags` flags parameter is currently unused and if omitted will def ault to 0.

Sys::Virt::Domain::CORE_DUMP_FORMAT_RAW
The raw ELF format

Sys::Virt::Domain::CORE_DUMP_FORMAT_KDUMP_ZLIB
The zlib compressed ELF format

Sys::Virt::Domain::CORE_DUMP_FORMAT_KDUMP_SNAPPY
The snappy compressed ELF format

Sys::Virt::Domain::CORE_DUMP_FORMAT_KDUMP_LZO
The lzo compressed ELF format

Sys::Virt::Domain::CORE_DUMP_FORMAT_WIN_DMP
The Windows dump format

`$dom->`**destroy()**
Immediately poweroff the machine. This is equivalent to removing the power plug. The guest OS is given no time to cleanup / save state. For a clean poweroff sequence, use the `shutdown` method instead.

my `$info` = `$dom->`**get_info()**
Returns a hash reference summarising the execution state of the domain. The elements of the hash are as follows:

maxMem
The maximum memory allowed for this domain, in kilobytes

memory
The current memory allocated to the domain in kilobytes

cpuTime
The amount of CPU time used by the domain

nrVirtCpu
The current number of virtual CPUs enabled in the domain

state
The execution state of the machine, which will be one of the constants &Sys::Virt::Domain::STATE_*.

my ($state, `$reason`) = `$dom->`**get_state()**
Returns an array whose values specify the current state of the guest, and the reason for it being in that state. The`$state` v alues are the same as for the `get_info` API, and the `$reason` values come from:

Sys::Virt::Domain::STATE_CRASHED_UNKNOWN
It is not known why the domain has crashed

Sys::Virt::Domain::STATE_CRASHED_PANICKED
The domain has crashed due to a kernel panic

Sys::Virt::Domain::STATE_NOSTATE_UNKNOWN
It is not known why the domain has no state

Sys::Virt::Domain::STATE_PAUSED_DUMP
The guest is paused due to a core dump operation

Sys::Virt::Domain::STATE_PAUSED_FROM_SNAPSHOT
The guest is paused due to a snapshot

Sys::Virt::Domain::STATE_PAUSED_IOERROR
The guest is paused due to an I/O error

Sys::Virt::Domain::STATE_PAUSED_MIGRATION
The guest is paused due to migration

Sys::Virt::Domain::STATE_PAUSED_SAVE
The guest is paused due to a save operation

Sys::Virt::Domain::STATE_PAUSED_UNKNOWN
It is not known why the domain has paused

Sys::Virt::Domain::STATE_PAUSED_USER
The guest is paused at admin request

Sys::Virt::Domain::STATE_PAUSED_WATCHDOG
The guest is paused due to the watchdog

Sys::Virt::Domain::STATE_PAUSED_SHUTTING_DOWN
The guest is paused while domain shutdown takes place

Sys::Virt::Domain::STATE_PAUSED_SNAPSHOT
The guest is paused while a snapshot takes place

Sys::Virt::Domain::STATE_PAUSED_CRASHED
The guest is paused due to a kernel panic

Sys::Virt::Domain::STATE_PAUSED_STARTING_UP
The guest is paused as it is being started up.

Sys::Virt::Domain::STATE_PAUSED_POSTCOPY
The guest is paused as post-copy migration is taking place

Sys::Virt::Domain::STATE_PAUSED_POSTCOPY_FAILED
The guest is paused as post-copy migration failed

Sys::Virt::Domain::STATE_RUNNING_BOOTED
The guest is running after being booted

Sys::Virt::Domain::STATE_RUNNING_FROM_SNAPSHOT
The guest is running after restore from snapshot

Sys::Virt::Domain::STATE_RUNNING_MIGRATED
The guest is running after migration

Sys::Virt::Domain::STATE_RUNNING_MIGRATION_CANCELED
The guest is running after migration abort

Sys::Virt::Domain::STATE_RUNNING_RESTORED
The guest is running after restore from file

Sys::Virt::Domain::STATE_RUNNING_SAVE_CANCELED
The guest is running after save cancel

Sys::Virt::Domain::STATE_RUNNING_UNKNOWN
It is not known why the domain has started

Sys::Virt::Domain::STATE_RUNNING_UNPAUSED
The guest is running after a resume

Sys::Virt::Domain::STATE_RUNNING_WAKEUP
The guest is running after wakeup from power management suspend

Sys::Virt::Domain::STATE_RUNNING_CRASHED
The guest was restarted after crashing

Sys::Virt::Domain::STATE_RUNNING_POSTCOPY
The guest is running but post-copy is taking place

Sys::Virt::Domain::STATE_BLOCKED_UNKNOWN
The guest is blocked for an unknown reason

Sys::Virt::Domain::STATE_SHUTDOWN_UNKNOWN
It is not known why the domain has shutdown

Sys::Virt::Domain::STATE_SHUTDOWN_USER
The guest is shutdown due to admin request

Sys::Virt::Domain::STATE_SHUTOFF_CRASHED
The guest is shutoff after a crash

Sys::Virt::Domain::STATE_SHUTOFF_DESTROYED
The guest is shutoff after being destroyed

Sys::Virt::Domain::STATE_SHUTOFF_FAILED
The guest is shutoff due to a virtualization failure

Sys::Virt::Domain::STATE_SHUTOFF_FROM_SNAPSHOT
The guest is shutoff after a snapshot

Sys::Virt::Domain::STATE_SHUTOFF_MIGRATED
The guest is shutoff after migration

Sys::Virt::Domain::STATE_SHUTOFF_SAVED
The guest is shutoff after a save

Sys::Virt::Domain::STATE_SHUTOFF_SHUTDOWN
The guest is shutoff due to controlled shutdown

Sys::Virt::Domain::STATE_SHUTOFF_UNKNOWN
It is not known why the domain has shutoff

Sys::Virt::Domain::STATE_SHUTOFF_DAEMON
The daemon stopped the guest due to a failure

Sys::Virt::Domain::STATE_PMSUSPENDED_UNKNOWN
It is not known why the domain was suspended to RAM

Sys::Virt::Domain::STATE_PMSUSPENDED_DISK_UNKNOWN
It is not known why the domain was suspended to disk

my `$info` = `$dom`->get_control_info($flags=0)
Returns a hash reference providing information about the control channel. The returned keys in the hash are

`state`
One of the CONTROL INFO constants listed later

`details`
Currently unused, always 0.

`stateTime`
The elapsed time since the control channel entered the current state.

my `$time` = `$dom`->get_time($flags=0);
Get the current time of the guest, in seconds and nanoseconds. The `$flags` parameter is currently unused and defaults to zero. The return value is an array ref with two elements, the first contains the time in seconds, the second contains the remaining nanoseconds.

`$dom`->set_time($secs, `$nsecs`, `$flags`=0);
Set the current time of the guest, in seconds and nanoseconds. The `$flags` parameter accepts one of

`Sys::Virt::Domain::TIME_SYNC`
Re-sync domain time from domain's RTC.

$dom->set_user_password($username, $password, $flags=0);

Update the password for account $username to be $password. $password is the clear-text password string unless the PASSWORD_ENCRYPTED flag is set.

Sys::Virt::Domain::PASSWORD_ENCRYPTED

The $password is encrypted with the password scheme required by the guest OS.

$dom->rename($newname, $flags=0)

Change the name of an inactive guest to be $newname. The$flags parameter is currently unused and defaults to zero.

my @errs = $dom->get_disk_errors($flags=0)

Returns a list of all disk errors that have occurred on the backing store for the guest's virtual disks. The returned array elements are hash references, containing two keys

path

The path of the disk with an error

error

The error type

$dom->send_key($keycodeset, $holdtime, \@keycodes, $flags=0)

Sends a sequence of keycodes to the guest domain. The $keycodeset should be one of the constants listed later in the KEYCODE SET section. $holdtiem is the duration, in milliseconds, to keep the key pressed before releasing it and sending the next keycode. @keycodes is an array reference containing the list of keycodes to send to the guest. The elements in the array should be keycode values from the specified keycode set. $flags is currently unused.

my $info = $dom->get_block_info($dev, $flags=0)

Returns a hash reference summarising the disk usage of the host backing store for a guest block device. The $dev parameter should be the path to the backing store on the host. $flags is currently unused and defaults to 0 if omitted. The returned hash contains the following elements

capacity

Logical size in bytes of the block device backing image *

allocation

Highest allocated extent in bytes of the block device backing image

physical

Physical size in bytes of the container of the backing image

$dom->set_max_memory($mem)

Set the maximum memory for the domain to the value $mem. The value of the $mem parameter is specified in kilobytes.

$mem = $dom->**get_max_memory()**

Returns the current maximum memory allowed for this domain in kilobytes.

$dom->set_memory($mem, $flags)

Set the current memory for the domain to the value $mem. The value of the $mem parameter is specified in kilobytes. This must be less than, or equal to the domain's max memory limit. The $flags parameter can control whether the update affects the live guest, or inactive config, defaulting to modifying the current state.

$dom->set_memory_stats_period($period, $flags)

Set the period on which guests memory stats are refreshed, with $period being a value in seconds. The $flags parameter is currently unused.

$dom->**shutdown()**

Request that the guest OS perform a graceful shutdown and poweroff. This usually requires some form of cooperation from the guest operating system, such as responding to an ACPI signal, or a guest agent process. For an immediate, forceful poweroff, use the destroy method instead.

$dom->reboot([$flags])
Request that the guest OS perform a graceful shutdown and optionally restart. The optional $flags parameter is currently unused and if omitted defaults to zero.

$dom->reset([$flags])
Perform a hardware reset of the virtual machine. The guest OS is given no opportunity to shutdown gracefully. The optional $flags parameter is currently unused and if omitted defaults to zero.

$dom->**get_max_vcpus()**
Return the maximum number of vcpus that are configured for the domain

$dom->attach_device($xml[, $flags])
Hotplug a new device whose configuration is given by $xml, to the running guest. The optional <$flags> parameter defaults to 0, but can accept one of the device hotplug flags described later.

$dom->detach_device($xml[, $flags])
Hotunplug an existing device whose configuration is given by $xml, from the running guest. The optional <$flags> parameter defaults to 0, but can accept one of the device hotplug flags described later.

$dom->detach_device_alias($alias[, $flags])
Hotunplug an existing device which is identified by $alias. The optional <$flags> parameter defaults to 0, but can accept one of the device hotplug flags described later.

$dom->update_device($xml[, $flags])
Update the configuration of an existing device. The new configuration is given by $xml. The optional <$flags> parameter defaults to 0 but can accept one of the device hotplug flags described later.

$data = $dom->block_peek($path, $offset, $size[, $flags])
Peek into the guest disk $path, at byte $offset capturing $size bytes of data. The returned scalar may contain embedded NULLs. The optional $flags parameter is currently unused and if omitted defaults to zero.

$data = $dom->memory_peek($offset, $size[, $flags])
Peek into the guest memory at byte $offset virtual address, capturing $size bytes of memory. The return scalar may contain embedded NULLs. The optional $flags parameter is currently unused and if omitted defaults to zero.

$flag = $dom->**get_autostart()**;
Return a true value if the guest domain is configured to automatically start upon boot. Return false, otherwise

$dom->set_autostart($flag)
Set the state of the autostart flag, which determines whether the guest will automatically start upon boot of the host OS

$dom->set_vcpus($count, [$flags])
Set the number of virtual CPUs in the guest VM to $count. The optional$flags parameter can be used to control whether the setting changes the live config or inactive config.

$dom->set_vcpu($cpumap, $state, [$flags])
Set the state of the CPUs in $cpumap to $state. The $flags parameter defaults to zero if not present.

$count = $dom->get_vcpus([$flags])
Get the number of virtual CPUs in the guest VM. The optional$flags parameter can be used to control whether to query the setting of the live config or inactive config.

$dom->set_guest_vcpus($cpumap, $state, [$flags=0])
Set the online status of the guest OS CPUs. The $cpumap parameter describes the set of CPUs to modify (eg ''0-3,^1''). $state is either **1** to set the CPUs online, or **0** to set them offline. The $flags parameter is currently unused and defaults to 0.

`$info` `$dom`->get_guest_vcpus([$flags=0])

Query information about the guest OS CPUs. The returned data is a hash reference with the following keys.

vcpus

String containing bitmap representing CPU ids reported currently known to the guest.

online

String containing bitmap representing CPU ids that are currently online in the guest.

offlinable

String containing bitmap representing CPU ids that can be offlined in the guest.

The `$flags` parameter is currently unused and defaults to 0.

`$type` = `$dom`->**get_scheduler_type()**

Return the scheduler type for the guest domain

`$stats` = `$dom`->block_stats($path)

Fetch the current I/O statistics for the block device given by `$path`. The returned hash reference contains keys for

`rd_req`

Number of read requests

`rd_bytes`

Number of bytes read

`wr_req`

Number of write requests

`wr_bytes`

Number of bytes written

`errs`

Some kind of error count

my `$params` = `$dom`->get_scheduler_parameters($flags=0)

Return the set of scheduler tunable parameters for the guest, as a hash reference. The precise set of keys in the hash are specific to the hypervisor.

`$dom`->set_scheduler_parameters($params, `$flags`=0)

Update the set of scheduler tunable parameters. The value names for tunables vary, and can be discovered using the `get_scheduler_params` call

my `$params` = `$dom`->get_memory_parameters($flags=0)

Return a hash reference containing the set of memory tunable parameters for the guest. The keys in the hash are one of the constants MEMORY PARAMETERS described later. The `$flags` parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

`$dom`->set_memory_parameters($params, `$flags`=0)

Update the memory tunable parameters for the guest. The `$params` should be a hash reference whose keys are one of the MEMORY PARAMETERS constants. The `$flags` parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

my `$params` = `$dom`->get_blkio_parameters($flags=0)

Return a hash reference containing the set of blkio tunable parameters for the guest. The keys in the hash are one of the constants BLKIO PARAMETERS described later. The `$flags` parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

`$dom`->set_blkio_parameters($params, `$flags`=0)

Update the blkio tunable parameters for the guest. The `$params` should be a hash reference whose keys are one of the BLKIO PARAMETERS constants. The `$flags` parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$stats = $dom->get_block_iotune($disk, $flags=0)
Return a hash reference containing the set of blkio tunable parameters for the guest disk $disk. The keys in the hash are one of the constants BLOCK IOTUNE PARAMETERS described later.

$dom->set_block_iotune($disk, $params, $flags=0);
Update the blkio tunable parameters for the guest disk $disk. The $params should be a hash reference whose keys are one of the BLOCK IOTUNE PARAMETERS constants.

my $params = $dom->get_interface_parameters($intf, $flags=0)
Return a hash reference containing the set of interface tunable parameters for the guest. The keys in the hash are one of the constants INTERFACE PARAMETERS described later.

$dom->set_interface_parameters($intf, $params, $flags=0)
Update the interface tunable parameters for the guest. The $params should be a hash reference whose keys are one of the INTERFACE PARAMETERS constants.

my $params = $dom->get_numa_parameters($flags=0)
Return a hash reference containing the set of numa tunable parameters for the guest. The keys in the hash are one of the constants NUMA PARAMETERS described later. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$dom->set_numa_parameters($params, $flags=0)
Update the numa tunable parameters for the guest. The $params should be a hash reference whose keys are one of the NUMA PARAMETERS constants. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

my $params = $dom->get_perf_events($flags=0)
Return a hash reference containing the set of performance events that are available for the guest. The keys in the hash are one of the constants PERF EVENTS described later. The$flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$dom->set_perf_events($params, $flags=0)
Update the enabled state for performance events for the guest. The $params should be a hash reference whose keys are one of the PERF EVENTS constants. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$dom->block_resize($disk, $newsize, $flags=0)
Resize the disk $disk to have new size $newsize KB. If the disk is backed by a special image format, the actual resize is done by the hypervisor. If the disk is backed by a raw file, or block device, the resize must be done prior to invoking this API call, and it merely updates the hypervisor's view of the disk size. The following flags may be used

Sys::Virt::Domain::BLOCK_RESIZE_BYTES
Treat $newsize as if it were in bytes, rather than KB.

$dom->interface_stats($path)
Fetch the current I/O statistics for the block device given by $path. The returned hash containins keys for

rx_bytes
Total bytes received

rx_packets
Total packets received

rx_errs
Total packets received with errors

rx_drop
Total packets drop at reception

`tx_bytes`
Total bytes transmitted

`tx_packets`
Total packets transmitted

`tx_errs`
Total packets transmitted with errors

`tx_drop`
Total packets dropped at transmission.

`$dom`->memory_stats($flags=0)
Fetch the current memory statistics for the guest domain. The `$flags` parameter is currently unused and can be omitted. The returned hash containins keys for

`swap_in`
Data read from swap space

`swap_out`
Data written to swap space

`major_fault`
Page fault involving disk I/O

`minor_fault`
Page fault not involving disk I/O

`unused`
Memory not used by the system

`available`
Total memory seen by guest

`rss`
Resident set size. Size of memory resident in host RAM.

`$info` = `$dom`->**get_security_label()**
Fetch information about the security label assigned to the guest domain. The returned hash reference has two keys, `model` gives the name of the security model in effect (eg `selinux`), while `label` provides the name of the security label applied to the domain. This method only returns information about the first security label. To retrieve all labels, use `get_security_label_list`.

`@info` = `$dom`->**get_security_label_list()**
Fetches information about all security labels assigned to the guest domain. The elements in the returned array are all hash references, whose keys are as described for `get_security_label`.

`$ddom` = `$dom`->migrate(destcon, \%params, flags=0)
Migrate a domain to an alternative host. The `destcon` parameter should be a `Sys::Virt` connection to the remote target host. The `flags` parameter takes one or more of the `Sys::Virt::Domain::MIGRATE_XXX` constants described later in this document. The `%params` parameter is a hash reference used to set various parameters for the migration operation, with the following valid keys.

`Sys::Virt::Domain::MIGRATE_PARAM_URI`
The URI to use for initializing the domain migration. It takes a hypervisor specific format. The uri_transports element of the hypervisor capabilities XML includes details of the supported URI schemes. When omitted libvirt will auto-generate suitable default URI. It is typically only necessary to specify this URI if the destination host has multiple interfaces and a specific interface is required to transmit migration data.

`Sys::Virt::Domain::MIGRATE_PARAM_DEST_NAME`
The name to be used for the domain on the destination host. Omitting this parameter keeps the domain name the same. This field is only allowed to be used with hypervisors that support

domain renaming during migration.

`Sys::Virt::Domain::MIGRATE_PARAM_DEST_XML`
The new configuration to be used for the domain on the destination host. The configuration must include an identical set of virtual devices, to ensure a stable guest ABI across migration. Only parameters related to host side configuration can be changed in the XML. Hypervisors which support this field will forbid migration if the provided XML would cause a change in the guest ABI. This field cannot be used to rename the domain during migration (use VIR_MIGRATE_PARAM_DEST_NAME field for that purpose). Domain name in the destination XML must match the original domain name.

Omitting this parameter keeps the original domain configuration. Using this field with hypervisors that do not support changing domain configuration during migration will result in a failure.

`Sys::Virt::Domain::MIGRATE_PARAM_GRAPHICS_URI`
URI to use for migrating client's connection to domain's graphical console as VIR_TYPED_PARAM_STRING. If specified, the client will be asked to automatically reconnect using these parameters instead of the automatically computed ones. This can be useful if, e.g., the client does not have a direct access to the network virtualization hosts are connected to and needs to connect through a proxy. The URI is formed as follows:

```
protocol://hostname[:port]/[?parameters]
```

where protocol is either ''spice'' or ''vnc'' and parameters is a list of protocol specific parameters separated by '&'. Currently recognized parameters are ''tlsPort'' and ''tlsSubject''. For example,

```
spice://target.host.com:1234/?tlsPort=4567
```

`Sys::Virt::Domain::MIGRATE_PARAM_BANDWIDTH`
The maximum bandwidth (in MiB/s) that will be used for migration. If set to 0 or omitted, libvirt will choose a suitable default. Some hypervisors do not support this feature and will return an error if this field is used and is not 0.

`Sys::Virt::Domain::MIGRATE_PARAM_BANDWIDTH_POSTCOPY`
The maximum bandwidth (in MiB/s) that will be used for migration during post-copy phase. If set to 0 or omitted, libvirt will choose a suitable default. Some hypervisors do not support this feature and return an error if this field is used and is not 0.

`Sys::Virt::Domain::MIGRATE_PARAM_LISTEN_ADDRESS`
The address on which to listen for incoming migration connections. If omitted, libvirt will listen on the wildcard address (0.0.0.0 or ::). This default may be a security risk if guests, or other untrusted users have the ability to connect to the virtualization host, thus use of an explicit restricted listen address is recommended.

`Sys::Virt::Domain::MIGRATE_PARAM_DISK_PORT`
Port that destination server should use for incoming disks migration. Type is VIR_TYPED_PARAM_INT. If set to 0 or omitted, libvirt will choose a suitable default. At the moment this is only supported by the QEMU driver.

`Sys::Virt::Domain::MIGRATE_PARAM_MIGRATE_DISKS`
The list of disks to migrate when doing block storage migration. In contrast to other parameters whose values are plain strings, the parameter value should be an array reference, whose elements are in turn strings representing the disk target names.

`Sys::Virt::Domain::MIGRATE_PARAM_COMPRESSION`
The type of compression method use use, either `xbzrle` or `mt`.

`Sys::Virt::Domain::MIGRATE_PARAM_COMPRESSION_MT_THREADS`
The number of compression threads to use

`Sys::Virt::Domain::MIGRATE_PARAM_COMPRESSION_MT_DTHREADS`
The number of decompression threads

`Sys::Virt::Domain::MIGRATE_PARAM_COMPRESSION_MT_LEVEL`
The compression level from 0 (no compression) to 9 (maximum compression)

`Sys::Virt::Domain::MIGRATE_PARAM_COMPRESSION_XBZRLE_CACHE`
The size of the cache for xbzrle compression

`Sys::Virt::Domain::MIGRATE_PARAM_PERSIST_XML`
The alternative persistent XML config to copy

`Sys::Virt::Domain::MIGRATE_PARAM_AUTO_CONVERGE_INITIAL`
The initial percentage to throttle guest vCPUs

`Sys::Virt::Domain::MIGRATE_PARAM_AUTO_CONVERGE_INCREMENT`
The additional percentage step size to throttle guest vCPUs if progress is not made

`Sys::Virt::Domain::MIGRATE_PARAM_PARALLEL_CONNECTIONS`
The number of connections used during parallel migration.

`Sys::Virt::Domain::MIGRATE_PARAM_TLS_DESTINATION`
Override the destination host name used for TLS verification. Normally the TLS certificate from the destination host must match the host's name for TLS verification to succeed. When the certificate does not match the destination hostname and the expected cetificate's hostname is known, this parameter can be used to pass this expected hostname when starting the migration.

`Sys::Virt::Domain::MIGRATE_PARAM_DISKS_URI`
The URI to use for initializing the domain migration for storage. It takes a hypervisor specific format. The uri_transports element of the hypervisor capabilities XML includes details of the supported URI schemes. When omitted libvirt will auto-generate suitable default URI. It is typically only necessary to specify this URI if the destination host has multiple interfaces and a specific interface is required to transmit storage data.

`$ddom` = `$dom`->migrate(destcon, flags=0, dname=undef, uri=undef, bandwidth=0)
Migrate a domain to an alternative host. Use of positional parameters with `migrate` is deprecated in favour of passing a hash reference as described above.

`$ddom` = `$dom`->migrate2(destcon, dxml, flags, dname, uri, bandwidth)
Migrate a domain to an alternative host. This method is deprecated in favour of passing a hash ref to `migrate`.

`$ddom` = `$dom`->migrate_to_uri(desturi, \%params, flags=0)
Migrate a domain to an alternative host. The `desturi` parameter should be a valid libvirt connection URI for the remote target host. The `flags` parameter takes one or more of the `Sys::Virt::Domain::MIGRATE_XXX` constants described later in this document. The `%params` parameter is a hash reference used to set various parameters for the migration operation, with the same keys described for the `migrate` API.

`$dom`->migrate_to_uri(desturi, flags, dname, bandwidth)
Migrate a domain to an alternative host. Use of positional parameters with `migrate_to_uri` is deprecated in favour of passing a hash reference as described above.

`$dom`->migrate_to_uri2(dconnuri, miguri, dxml, flags, dname, bandwidth)
Migrate a domain to an alternative host. This method is deprecated in favour of passing a hash ref to `migrate_to_uri`.

`$dom`->migrate_set_max_downtime($downtime, `$flags`=0)
Set the maximum allowed downtime during migration of the guest. A longer downtime makes it more likely that migration will complete, at the cost of longer time blackout for the guest OS at the switch over point. The `downtime` parameter is measured in milliseconds. The `$flags` parameter is currently unused and defaults to zero.

$downtime = $dom->migrate_get_max_downtime($flags=0) Get the current value of the maximum downtime allowed during a migration of a guest. The returned <downtime> value is measured in milliseconds. The $flags parameter is currently unused and defaults to zero.

$dom->migrate_set_max_speed($bandwidth, $flags=0)

Set the maximum allowed bandwidth during migration of the guest. The bandwidth parameter is measured in MB/second. The $flags parameter takes zero or more of the constants:

$Sys::Virt::Domain::MIGRATE_MAX_SPEED_POSTCOPY

Set the post-copy speed instead of the pre-copy speed.

$bandwidth = $dom->migrate_get_max_speed($flags=0)

Get the maximum allowed bandwidth during migration fo the guest. The returned <bandwidth> value is measured in MB/second. The $flags parameter is accepts the same constants as migrate_set_max_speed.

$dom->migrate_set_compression_cache($cacheSize, $flags=0)

Set the maximum allowed compression cache size during migration of the guest. The cacheSize parameter is measured in bytes. The $flags parameter is currently unused and defaults to zero.

$cacheSize = $dom->migrate_get_compression_cache($flags=0)

Get the maximum allowed compression cache size during migration of the guest. The returned <bandwidth> value is measured in bytes. The $flags parameter is currently unused and defaults to zero.

$dom->migrate_start_post_copy($flags=0)

Switch the domain from pre-copy to post-copy mode. This requires that the original migrate command had the Sys::Virt::Domain::MIGRATE_POST_COPY flag specified.

$dom->inject_nmi($flags)

Trigger an NMI in the guest virtual machine. The $flags parameter is currently unused and defaults to 0.

$dom->open_console($st, $devname, $flags)

Open the text console for a serial, parallel or paravirt console device identified by $devname, connecting it to the stream $st. If $devname is undefined, the default console will be opened. $st must be a Sys::Virt::Stream object used for bi-directional communication with the console. $flags is currently unused, defaulting to 0.

$dom->open_channel($st, $devname, $flags)

Open the text console for a data channel device identified by $devname, connecting it to the stream $st. $st must be a Sys::Virt::Stream object used for bi-directional communication with the channel. $flags is currently unused, defaulting to 0.

$dom->open_graphics($idx, $fd, $flags)

Open the graphics console for a guest, identified by $idx, counting from 0. The $fd should be a file descriptor for an anoymous socket pair. The $flags argument should be one of the constants listed at the end of this document, and defaults to 0.

$fd = $dom->open_graphics_fd($idx, $flags)

Open the graphics console for a guest, identified by $idx, counting from 0. The $flags argument should be one of the constants listed at the end of this document, and defaults to 0. The return value will be a file descriptor connected to the console which must be closed when no longer needed. This method is preferred over open_graphics since it will work correctly under sVirt mandatory access control policies.

my $mimetype = $dom->screenshot($st, $screen, $flags)

Capture a screenshot of the virtual machine's monitor. The $screen parameter controls which monitor is captured when using a multi-head or multi-card configuration. $st must be a Sys::Virt::Stream object from which the data can be read. $flags is currently unused and defaults to 0. The mimetype of the screenshot is returned

@vcpuinfo = $dom->get_vcpu_info($flags=0)

Obtain information about the state of all virtual CPUs in a running guest domain. The returned list will have one element for each vCPU, where each elements contains a hash reference. The keys in the hash are, number the vCPU number, cpu the physical CPU on which the vCPU is currently scheduled, cpuTime the cumulative execution time of the vCPU, state the running state and affinity giving the allowed shedular placement. The value for affinity is a string representing a bitmask against physical CPUs, 8 cpus per character. To extract the bits use the unpack function with the b* template. NB The state, cpuTime, cpu values are only available if using $flags value of 0, and the domain is currently running; otherwise they will all be set to zero.

$dom->pin_vcpu($vcpu, $mask)

Pin the virtual CPU given by index $vcpu to physical CPUs given by $mask. The $mask is a string representing a bitmask against physical CPUs, 8 cpus per character.

$mask = $dom->**get_emulator_pin_info()**

Obtain information about the CPU affinity of the emulator process. The returned $mask is a bitstring against physical CPUs, 8 cpus per character. To extract the bits use the unpack function with the b* template.

$dom->pin_emulator($newmask, $flags=0)

Pin the emulator threads to the physical CPUs identified by the affinity in $newmask. The $newmask is a bitstring against the physical CPUa, 8 cpus per character. To create a suitable bitstring, use the vec function with a value of 1 for the BITS parameter.

@iothreadinfo = $dom->get_iothread_info($flags=0)

Obtain information about the state of all IOThreads in a running guest domain. The returned list will have one element for each IOThread, where each elements contains a hash reference. The keys in the hash are, number the IOThread number and affinity giving the allowed schedular placement. The value for affinity is a string representing a bitmask against physical CPUs, 8 cpus per character. To extract the bits use the unpack function with the b* template.

$dom->pin_iothread($iothread, $mask)

Pin the IOThread given by index $iothread to physical CPUs given by $mask. The $mask is a string representing a bitmask against physical CPUs, 8 cpus per character.

$dom->add_iothread($iothread, $flags=0)

Add a new IOThread by the $iothread value to the guest domain. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$dom->del_iothread($iothread, $flags=0)

Delete an existing IOThread by the $iothread value from the guest domain. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

$dom->set_iothread($iothread, $params, $flags=0)

Set parameters for the IOThread by the $iothread value on the guest domain. The $params parameter is a hash reference whose keys are the IOTHREAD STATS constants documented later. The $flags parameter accepts one or more the CONFIG OPTION constants documented later, and defaults to 0 if omitted.

my @stats = $dom->get_cpu_stats($startCpu, $numCpus, $flags=0)

Requests the guests host physical CPU usage statistics, starting from host CPU <$startCpu> counting up to $numCpus. If $startCpu is -1 and $numCpus is 1, then the utilization across all CPUs is returned. Returns an array of hash references, each element containing stats for one CPU.

my $info = $dom->**get_job_info()**

Returns a hash reference summarising the execution state of the background job. The elements of the hash are as follows:

type
The type of job, one of the JOB TYPE constants listed later in this document.

timeElapsed
The elapsed time in milliseconds

timeRemaining
The expected remaining time in milliseconds. Only set if the `type` is JOB_UNBOUNDED.

dataTotal
The total amount of data expected to be processed by the job, in bytes.

dataProcessed
The current amount of data processed by the job, in bytes.

dataRemaining
The expected amount of data remaining to be processed by the job, in bytes.

memTotal
The total amount of mem expected to be processed by the job, in bytes.

memProcessed
The current amount of mem processed by the job, in bytes.

memRemaining
The expected amount of mem remaining to be processed by the job, in bytes.

fileTotal
The total amount of file expected to be processed by the job, in bytes.

fileProcessed
The current amount of file processed by the job, in bytes.

fileRemaining
The expected amount of file remaining to be processed by the job, in bytes.

my ($type, `$stats`) = `$dom`->get_job_stats($flags=0)
Returns an array summarising the execution state of the background job. The `$type` value is one of the JOB TYPE constants listed later in this document. The `$stats` value is a hash reference, whose elements are one of the following constants.

type
The type of job, one of the JOB TYPE constants listed later in this document.

The `$flags` parameter defaults to zero and can take one of the following constants.

Sys::Virt::Domain::JOB_STATS_COMPLETED
Return the stats of the most recently completed job.

Sys::Virt::Domain::JOB_STATS_KEEP_COMPLETED
Don't clear the completed stats after reading them.

Sys::Virt::Domain::JOB_TIME_ELAPSED
The elapsed time in milliseconds

Sys::Virt::Domain::JOB_TIME_ELAPSED_NET
Time in milliseconds since the beginning of the migration job NOT including the time required to transfer control flow from the source host to the destination host.

Sys::Virt::Domain::JOB_TIME_REMAINING
The expected remaining time in milliseconds. Only set if the `type` is JOB_UNBOUNDED.

Sys::Virt::Domain::JOB_DATA_TOTAL
The total amount of data expected to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_DATA_PROCESSED
The current amount of data processed by the job, in bytes.

Sys::Virt::Domain::JOB_DATA_REMAINING
The expected amount of data remaining to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_MEMORY_TOTAL
The total amount of mem expected to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_MEMORY_PROCESSED
The current amount of mem processed by the job, in bytes.

Sys::Virt::Domain::JOB_MEMORY_REMAINING
The expected amount of mem remaining to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_MEMORY_CONSTANT
The number of pages filled with a constant byte which have been transferred

Sys::Virt::Domain::JOB_MEMORY_NORMAL
The number of pages transferred without any compression

Sys::Virt::Domain::JOB_MEMORY_NORMAL_BYTES
The number of bytes transferred without any compression

Sys::Virt::Domain::JOB_MEMORY_BPS
The bytes per second transferred

Sys::Virt::Domain::JOB_MEMORY_DIRTY_RATE
The number of memory pages dirtied per second

Sys::Virt::Domain::JOB_MEMORY_PAGE_SIZE
The memory page size in bytes

Sys::Virt::Domain::JOB_MEMORY_ITERATION
The total number of iterations over guest memory

Sys::Virt::Domain::JOB_MEMORY_POSTCOPY_REQS
The number of page requests received from the destination host during post-copy migration.

Sys::Virt::Domain::JOB_DISK_TOTAL
The total amount of file expected to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_DISK_PROCESSED
The current amount of file processed by the job, in bytes.

Sys::Virt::Domain::JOB_DISK_REMAINING
The expected amount of file remaining to be processed by the job, in bytes.

Sys::Virt::Domain::JOB_DISK_BPS
The bytes per second transferred

Sys::Virt::Domain::JOB_AUTO_CONVERGE_THROTTLE
The percentage by which vCPUs are currently throttled

Sys::Virt::Domain::JOB_COMPRESSION_CACHE
The size of the compression cache in bytes

Sys::Virt::Domain::JOB_COMPRESSION_BYTES
The number of compressed bytes transferred

Sys::Virt::Domain::JOB_COMPRESSION_PAGES
The number of compressed pages transferred

Sys::Virt::Domain::JOB_COMPRESSION_CACHE_MISSES
The number of changing pages not in compression cache

Sys::Virt::Domain::JOB_COMPRESSION_OVERFLOW
The number of changing pages in the compression cache but sent uncompressed since the compressed page was larger than the non-compressed page.

Sys::Virt::Domain::JOB_DOWNTIME
The number of milliseconds of downtime expected during migration switchover.

Sys::Virt::Domain::JOB_DOWNTIME_NET
Real measured downtime (ms) NOT including the time required to transfer control flow from the source host to the destination host.

Sys::Virt::Domain::JOB_SETUP_TIME
The number of milliseconds of time doing setup of the job

Sys::Virt::Domain::JOB_OPERATION
The type of operation associated with the job

Sys::Virt::Domain::JOB_SUCCESS
Whether the job was successfully completed.

Sys::Virt::Domain::JOB_DISK_TEMP_TOTAL
Possible total temporary disk space for the job in bytes

Sys::Virt::Domain::JOB_DISK_TEMP_USED
Current total temporary disk space for the job in bytes

Sys::Virt::Domain::JOB_ERRMSG
The error message from a failed job

The values for the Sys::Virt::Domain::JOB_OPERATION field are

Sys::Virt::Domain::JOB_OPERATION_UNKNOWN
No known job type

Sys::Virt::Domain::JOB_OPERATION_START
The guest is starting

Sys::Virt::Domain::JOB_OPERATION_SAVE
The guest is saving to disk

Sys::Virt::Domain::JOB_OPERATION_RESTORE
The guest is restoring from disk

Sys::Virt::Domain::JOB_OPERATION_MIGRATION_IN
The guest is migrating in from another host

Sys::Virt::Domain::JOB_OPERATION_MIGRATION_OUT
The guest is migrating out to another host

Sys::Virt::Domain::JOB_OPERATION_SNAPSHOT
The guest is saving a snapshot

Sys::Virt::Domain::JOB_OPERATION_SNAPSHOT_REVERT
The guest is reverting to a snapshot

Sys::Virt::Domain::JOB_OPERATION_DUMP
The guest is saving a crash dump

Sys::Virt::Domain::JOB_OPERATION_BACKUP
The guest is performing a block backup

`$dom->`**abort_job()**
Aborts the currently executing job

my `$info` = `$dom->`get_block_job_info($path, `$flags`=0)
Returns a hash reference summarising the execution state of the block job. The `$path` parameter should be the fully qualified path of the block device being changed. Valid `$flags` include:

Sys::Virt::Domain::BLOCK_JOB_INFO_BANDWIDTH_BYTES
Treat bandwidth value as bytes instead of MiB.

$dom->set_block_job_speed($path, $bandwidth, $flags=0)
Change the maximum I/O bandwidth used by the block job that is currently executing for $path. The $bandwidth argument is specified in MB/s. The $flags parameter can take the bitwise union of the values:

Sys::Virt::Domain::BLOCK_JOB_SPEED_BANDWIDTH_BYTES
The $bandwidth parameter value is measured in bytes/s instead of MB/s.

$dom->abort_block_job($path, $flags=0)
Abort the current job that is executing for the block device associated with $path

$dom->block_pull($path, $bandwidth, $flags=0)
Merge the backing files associated with $path into the top level file. The $bandwidth parameter specifies the maximum I/O rate to allow in MB/s. The $flags parameter can take the bitwise union of the values:

Sys::Virt::Domain::BLOCK_PULL_BANDWIDTH_BYTES
The $bandwidth parameter value is measured in bytes/s instead of MB/s.

$dom->block_rebase($path, $base, $bandwidth, $flags=0)
Switch the backing path associated with $path to instead use $base. The $bandwidth parameter specifies the maximum I/O rate to allow in MB/s. The $flags parameter can take the bitwise union of the values:

Sys::Virt::Domain::BLOCK_REBASE_BANDWIDTH_BYTES
The $bandwidth parameter value is measured in bytes/s instead of MB/s.

$dom->block_copy($path, $destxml, $params, $flags=0)
Copy contents of a disk image <$path> into the target volume described by $destxml which follows the schema of the <disk> element in the domain XML. The $params parameter is a hash of optional parameters to control the process

Sys::Virt::Domain::BLOCK_COPY_BANDWIDTH
The maximum bandwidth in bytes per second.

Sys::Virt::Domain::BLOCK_COPY_GRANULARITY
The granularity in bytes of the copy process

Sys::Virt::Domain::BLOCK_COPY_BUF_SIZE
The maximum amount of data in flight in bytes.

$dom->block_commit($path, $base, $top, $bandwidth, $flags=0)
Commit changes there were made to the temporary top level file $top. Takes all the differences between $top and $base and merge them into $base. The $bandwidth parameter specifies the maximum I/O rate to allow in MB/s. The $flags parameter can take the bitwise union of the values:

Sys::Virt::Domain::BLOCK_COMMIT_BANDWIDTH_BYTES
The $bandwidth parameter value is measured in bytes instead of MB/s.

$count = $dom->**num_of_snapshots()**
Return the number of saved snapshots of the domain

@names = $dom->**list_snapshot_names()**
List the names of all saved snapshots. The names can be used with the lookup_snapshot_by_name

@snapshots = $dom->**list_snapshots()**
Return a list of all snapshots currently known to the domain. The elements in the returned list are instances of the Sys::Virt::DomainSnapshot class. This method requires O(n) RPC calls, so the list_all_snapshots method is recommended as a more efficient alternative.

my @snapshots = $dom->list_all_snapshots($flags)
Return a list of all domain snapshots associated with this domain. The elements in the returned list are instances of the Sys::Virt::DomainSnapshot class. The $flags parameter can be used to filter the list of return domain snapshots.

my $snapshot = $dom->get_snapshot_by_name($name)
Return the domain snapshot with a name of $name. The returned object is an instance of the Sys::Virt::DomainSnapshot class.

$dom->**has_current_snapshot()**
Returns a true value if the domain has a currently active snapshot

$snapshot = $dom->**current_snapshot()**
Returns the currently active snapshot for the domain.

$snapshot = $dom->create_snapshot($xml[, $flags])
Create a new snapshot from the $xml. The $flags parameter accepts the **SNAPSHOT CREATION** constants listed in Sys::Virt::DomainSnapshots.

my @checkpoints = $dom->list_all_checkpoints($flags)
Return a list of all domain checkpoints associated with this domain. The elements in the returned list are instances of the Sys::Virt::DomainCheckpoint class. The $flags parameter can be used to filter the list of return domain checkpoints.

my $checkpoint = $dom->get_checkpoint_by_name($name)
Return the domain checkpoint with a name of $name. The returned object is an instance of the Sys::Virt::DomainCheckpoint class.

$checkpoint = $dom->create_checkpoint($xml[, $flags])
Create a new checkpoint from the $xml. The $flags parameter accepts the **CHECKPOINT CREATION** constants listed in Sys::Virt::DomainCheckpoints.

$dom->backup_begin($backupxml, $checkpointxml=undef, $flags=0);
Start a point-in-time backup job for the specified disks of a running domain. The $backupxml parameter describes the backup operation, including which disks to use. The optional $checkpointxml parameter can be used to create a checkpoint covering to the same point in time as the backup. The optional $flags parameter can be one of the following constants:

Sys::Virt::Domain::BACKUP_BEGIN_REUSE_EXTERNAL
Assume that the output/temporary files for the backup have been precreated by the caller with the correct size and format.

$xml = $dom->backup_get_xml_description($flags=0);
Get the XML description of the currently executing backup job. If there is no backup job then an error is raised.

$dom->fs_trim($mountPoint, $minimum, $flags=0);
Issue an FS_TRIM command to the device at $mountPoint to remove chunks of unused space that are at least $minimum bytes in length. $flags is currently unused and defaults to zero.

$dom->fs_freeze(\@mountPoints, $flags=0);
Freeze all the filesystems associated with the @mountPoints array reference. If <@mountPoints> is an empty list, then all filesystems will be frozen. $flags is currently unused and defaults to zero.

$dom->fs_thaw(\@mountPoints, $flags=0);
Thaw all the filesystems associated with the @mountPoints array reference. If <@mountPoints> is an empty list, then all filesystems will be thawed. $flags is currently unused and defaults to zero.

@fslist = $dom->get_fs_info($flags=0);
Obtain a list of all guest filesystems. The returned list will contain one element for each filesystem, whose value will be a hash reference with the following keys

name
The name of the guest device that is mounted

fstype
The filesystem type (eg 'ext4', 'fat', 'ntfs', etc)

mountpoint
The location in the filesystem tree of the mount

devalias
An array reference containing list of device aliases associated with the guest device. The device aliases correspond to disk target names in the guest XML configuration

`@nics` = `$dom`->get_interface_addresses($src, `$flags`=0);
Obtain a list of all guest network interfaces. The `$src` parameter is one of the constants

Sys::Virt::Domain::INTERFACE_ADDRESSES_SRC_LEASE
Extract the DHCP server lease information

Sys::Virt::Domain::INTERFACE_ADDRESSES_SRC_AGENT
Query the guest OS via an agent

Sys::Virt::Domain::INTERFACE_ADDRESSES_SRC_ARP
Extract from the local ARP tables

The returned list will contain one element for each interface. The values in the list will be a hash reference with the following keys

name
The name of the guest interface that is mounted

hwaddr
The hardware address, aka MAC, if available.

addrs
An array reference containing list of IP addresses associated with the guest NIC. Each element in the array is a further hash containing

addr
The IP address string

prefix
The IP address network prefix

type
The IP address type (IPv4 vs IPv6)

`$dom`->send_process_signal($pid, `$signum`, `$flags`=0);
Send the process `$pid` the signal `$signum`. The `$signum` value must be one of the constants listed later, not a POSIX or Linux signal value. `$flags` is currently unused and defaults to zero.

`$dom`->set_block_threshold($dev, `$threshold`, `$flags`=0);
Set the threshold level for delivering the EVENT_ID_BLOCK_THRESHOLD if the device or backing chain element described by `$dev` is written beyond the set `$threshold` level. The threshold level is unset once the event fires. The event might not be delivered at all if libvirtd was not running at the moment when the threshold was reached.

`$dom`->set_lifecycle_action($type, `$action`, `$flags`=0)
Changes the actions of lifecycle events for domain represented as <on_$type>$action</on_$type> in the domain XML.

`$info` = `$dom`->get_launch_security_info($flags=0)
Get information about the domain launch security policy. `$flags` is currently unused and defaults to zero. The returned hash may contain the following keys

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_MEASUREMENT
The AMD SEV launch measurement

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_API_MAJOR
The host SEV API major version

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_API_MINOR
The host SEV API minor version

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_BUILD_ID
The host SEV firmware build ID

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_POLICY
The guest SEV policy

`$dom`->set_launch_security_state(\%params, `$flags`=0)
Set information about the domain launch security state. `$flags` is currently unused and defaults to zero. The provided hash may contain the following keys

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_SECRET
The SEV secret string to inject

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_SECRET_HEADER
The SEV secret header string to inject

Sys::Virt::Domain::LAUNCH_SECURITY_SEV_SECRET_SET_ADDRESS
The address at which to inject the SEV secret. If omitted it can be automatically determined from the firmware

`$info` = `$dom`->get_guest_info($types, `$flags`=0)
Get information about the domain guest configuration. The `$types` parameter determines what pieces of information are returned and should be the bitwise or of the following constants:

Sys::Virt::Domain::GUEST_INFO_USERS
Active user information

Sys::Virt::Domain::GUEST_INFO_OS
Misc operating system information

Sys::Virt::Domain::GUEST_INFO_TIMEZONE
The guest timezone

Sys::Virt::Domain::GUEST_INFO_HOSTNAME
The guest hostname

Sys::Virt::Domain::GUEST_INFO_FILESYSTEM
Filesystem mount information

Sys::Virt::Domain::GUEST_INFO_DISKS
Block device information

Sys::Virt::Domain::GUEST_INFO_INTERFACES
Network interfaces information

`$flags` is currently unused and defaults to zero.

`$dom`->set_agent_response_timeout($timeout, `$flags`=0)
Set the amount of time to wait for the agent to respond to a command. `$timeout` is a positive integer representing the number of seconds to wait, or one of the constants:

Sys::Virt::Domain::AGENT_RESPONSE_TIMEOUT_BLOCK
Wait forever with no timeout

Sys::Virt::Domain::AGENT_RESPONSE_TIMEOUT_NOWAIT
Don’t want at all, return immediately

Sys::Virt::Domain::AGENT_RESPONSE_TIMEOUT_DEFAULT
Use the hypervisor driver's default timeout

The `$flags` parameter is currently unused and defaults to zero.

my `@keys` = `$dom`->authorized_ssh_keys_get($user, `$flags`=0)
Retrieve the list of authorized SSH keys for the user account `$user`. The`$flags` parameter is currently unused and defaults to zero.

`$dom`->authorized_ssh_keys_set($user, \@keys, `$flags`=0)
Update the list of authorized SSH keys for the user account `$user`. The`@keys` parameter should be an array reference containing the new keys, if any. The default behaviour is to set the authorized SSH keys to the exact set specified in `@keys`. This can be modified via the `$flags` parameter which takes the following constants

Sys::Virt::Domain::AUTHORIZED_SSH_KEYS_SET_APPEND
Append `@keys` to the current set of keys, rather than replacing the existing keys.

Sys::Virt::Domain::AUTHORIZED_SSH_KEYS_SET_REMOVE
Remove `@keys` from the current set of keys, rather than replacing the existing keys. It is not an error if the keys do not exist.

my `@msgs` = `$dom`->get_messages($flags=0)
Retrieve a list of messages associated with the domain. The optional `$flags` parameter can accept zero or more of

Sys::Virt::Domain::MESSAGE_DEPRECATION
Warnings about use of deprecated features

Sys::Virt::Domain::MESSAGE_TAINTING
Warnings about actions that have tainting the domain

`$dom`->start_dirty_rate_calc($dom, `$secs`, `$flags`=0)
Request calculation of the domain's memory dirty rate over the next `$secs` seconds. `$flags` is currently unused and defaults to zero.

CONSTANTS

A number of the APIs take a `flags` parameter. In most cases passing a value of zero will be satisfactory. Some APIs, however, accept named constants to alter their behaviour. This section documents the current known constants.

DOMAIN STATE

The domain state constants are useful in interpreting the `state` key in the hash returned by the `get_info` method.

Sys::Virt::Domain::STATE_NOSTATE
The domain is active, but is not running / blocked (eg idle)

Sys::Virt::Domain::STATE_RUNNING
The domain is active and running

Sys::Virt::Domain::STATE_BLOCKED
The domain is active, but execution is blocked

Sys::Virt::Domain::STATE_PAUSED
The domain is active, but execution has been paused

Sys::Virt::Domain::STATE_SHUTDOWN
The domain is active, but in the shutdown phase

Sys::Virt::Domain::STATE_SHUTOFF
The domain is inactive, and shut down.

Sys::Virt::Domain::STATE_CRASHED
The domain is inactive, and crashed.

Sys::Virt::Domain::STATE_PMSUSPENDED
The domain is active, but in power management suspend state

CONTROL INFO

The following constants can be used to determine what the guest domain control channel status is

Sys::Virt::Domain::CONTROL_ERROR
The control channel has a fatal error

Sys::Virt::Domain::CONTROL_OK
The control channel is ready for jobs

Sys::Virt::Domain::CONTROL_OCCUPIED
The control channel is busy

Sys::Virt::Domain::CONTROL_JOB
The control channel is busy with a job

If the status is `Sys::Virt::Domain::CONTROL_ERROR`, then one of the following constants describes the reason

Sys::Virt::Domain::CONTROL_ERROR_REASON_NONE
There is no reason for the error available

Sys::Virt::Domain::CONTROL_ERROR_REASON_UNKNOWN
The reason for the error is unknown

Sys::Virt::Domain::CONTROL_ERROR_REASON_INTERNAL
There was an internal error in libvirt

Sys::Virt::Domain::CONTROL_ERROR_REASON_MONITOR
There was an error speaking to the monitor

DOMAIN CREATION

The following constants can be used to control the behaviour of domain creation

Sys::Virt::Domain::START_PAUSED
Keep the guest vCPUs paused after starting the guest

Sys::Virt::Domain::START_AUTODESTROY
Automatically destroy the guest when the connection is closed (or fails)

Sys::Virt::Domain::START_BYPASS_CACHE
Do not use OS I/O cache if starting a domain with a saved state image

Sys::Virt::Domain::START_FORCE_BOOT
Boot the guest, even if there was a saved snapshot

Sys::Virt::Domain::START_VALIDATE
Validate the XML document against the XML schema

DOMAIN DEFINE

The following constants can be used to control the behaviour of domain define operations

Sys::Virt::Domain::DEFINE_VALIDATE
Validate the XML document against the XML schema

KEYCODE SETS

The following constants define the set of supported keycode sets

Sys::Virt::Domain::KEYCODE_SET_LINUX
The Linux event subsystem keycodes

Sys::Virt::Domain::KEYCODE_SET_XT
The original XT keycodes

Sys::Virt::Domain::KEYCODE_SET_ATSET1
The AT Set1 keycodes (aka XT)

Sys::Virt::Domain::KEYCODE_SET_ATSET2
The AT Set2 keycodes (aka AT)

Sys::Virt::Domain::KEYCODE_SET_ATSET3
The AT Set3 keycodes (aka PS2)

Sys::Virt::Domain::KEYCODE_SET_OSX
The OS-X keycodes

Sys::Virt::Domain::KEYCODE_SET_XT_KBD
The XT keycodes from the Linux Keyboard driver

Sys::Virt::Domain::KEYCODE_SET_USB
The USB HID keycode set

Sys::Virt::Domain::KEYCODE_SET_WIN32
The Windows keycode set

Sys::Virt::Domain::KEYCODE_SET_QNUM
The XT keycode set, with the extended scancodes using the high bit of the first byte, instead of the low bit of the second byte.

Sys::Virt::Domain::KEYCODE_SET_RFB
A deprecated alias for `Sys::Virt::Domain::KEYCODE_SET_QNUM`

MEMORY PEEK

The following constants can be used with the `memory_peek` method's flags parameter

Sys::Virt::Domain::MEMORY_VIRTUAL
Indicates that the offset is using virtual memory addressing.

Sys::Virt::Domain::MEMORY_PHYSICAL
Indicates that the offset is using physical memory addressing.

VCPU STATE

The following constants are useful when interpreting the virtual CPU run state

Sys::Virt::Domain::VCPU_OFFLINE
The virtual CPU is not online

Sys::Virt::Domain::VCPU_RUNNING
The virtual CPU is executing code

Sys::Virt::Domain::VCPU_BLOCKED
The virtual CPU is waiting to be scheduled

VCPU HOST PLACEMENT STATE

The following constants are useful when interpreting the virtual CPU host placement

Sys::Virt::Domain::VCPU_INFO_CPU_OFFLINE
The virtual CPU is not online

Sys::Virt::Domain::VCPU_INFO_CPU_UNAVAILABLE
The virtual CPU placement is not available from this hypervisor

OPEN GRAPHICS CONSTANTS

The following constants are used when opening a connection to the guest graphics server

Sys::Virt::Domain::OPEN_GRAPHICS_SKIPAUTH
Skip authentication of the client

OPEN CONSOLE CONSTANTS

The following constants are used when opening a connection to the guest console

Sys::Virt::Domain::OPEN_CONSOLE_FORCE
: Force opening of the console, disconnecting any other open session

Sys::Virt::Domain::OPEN_CONSOLE_SAFE
: Check if the console driver supports safe operations

OPEN CHANNEL CONSTANTS

The following constants are used when opening a connection to the guest channel

Sys::Virt::Domain::OPEN_CHANNEL_FORCE
: Force opening of the channel, disconnecting any other open session

XML DUMP OPTIONS

The following constants are used to control the information included in the XML configuration dump

Sys::Virt::Domain::XML_INACTIVE
: Report the persistent inactive configuration for the guest, even if it is currently running.

Sys::Virt::Domain::XML_SECURE
: Include security sensitive information in the XML dump, such as passwords.

Sys::Virt::Domain::XML_UPDATE_CPU
: Update the CPU model definition to match the current executing state.

Sys::Virt::Domain::XML_MIGRATABLE
: Update the XML to allow migration to older versions of libvirt

DEVICE HOTPLUG OPTIONS

The following constants are used to control device hotplug operations

Sys::Virt::Domain::DEVICE_MODIFY_CURRENT
: Modify the domain in its current state

Sys::Virt::Domain::DEVICE_MODIFY_LIVE
: Modify only the live state of the domain

Sys::Virt::Domain::DEVICE_MODIFY_CONFIG
: Modify only the persistent config of the domain

Sys::Virt::Domain::DEVICE_MODIFY_FORCE
: Force the device to be modified

MEMORY OPTIONS

The following constants are used to control memory change operations

Sys::Virt::Domain::MEM_CURRENT
: Modify the current state

Sys::Virt::Domain::MEM_LIVE
: Modify only the live state of the domain

Sys::Virt::Domain::MEM_CONFIG
: Modify only the persistent config of the domain

Sys::Virt::Domain::MEM_MAXIMUM
: Modify the maximum memory value

CONFIG OPTIONS

The following constants are used to control what configuration a domain update changes

Sys::Virt::Domain::AFFECT_CURRENT
: Modify the current state

Sys::Virt::Domain::AFFECT_LIVE
Modify only the live state of the domain

Sys::Virt::Domain::AFFECT_CONFIG
Modify only the persistent config of the domain

MIGRATE OPTIONS

The following constants are used to control how migration is performed

Sys::Virt::Domain::MIGRATE_LIVE
Migrate the guest without interrupting its execution on the source host.

Sys::Virt::Domain::MIGRATE_PEER2PEER
Manage the migration process over a direct peer−2−peer connection between the source and destination host libvirtd daemons.

Sys::Virt::Domain::MIGRATE_TUNNELLED
Tunnel the migration data over the libvirt daemon connection, rather than the native hypervisor data transport. Requires PEER2PEER flag to be set.

Sys::Virt::Domain::MIGRATE_PERSIST_DEST
Make the domain persistent on the destination host, defining its configuration file upon completion of migration.

Sys::Virt::Domain::MIGRATE_UNDEFINE_SOURCE
Remove the domain's persistent configuration after migration completes successfully.

Sys::Virt::Domain::MIGRATE_PAUSED
Do not re-start execution of the guest CPUs on the destination host after migration completes.

Sys::Virt::Domain::MIGRATE_NON_SHARED_DISK
Copy the complete contents of the disk images during migration

Sys::Virt::Domain::MIGRATE_NON_SHARED_INC
Copy the incrementally changed contents of the disk images during migration

Sys::Virt::Domain::MIGRATE_CHANGE_PROTECTION
Do not allow changes to the virtual domain configuration while migration is taking place. This option is automatically implied if doing a peer−2−peer migration.

Sys::Virt::Domain::MIGRATE_UNSAFE
Migrate even if the compatibility check indicates the migration will be unsafe to the guest.

Sys::Virt::Domain::MIGRATE_OFFLINE
Migrate the guest config if the guest is not currently running

Sys::Virt::Domain::MIGRATE_COMPRESSED
Enable compression of the migration data stream

Sys::Virt::Domain::MIGRATE_ABORT_ON_ERROR
Abort if an I/O error occurrs on the disk

Sys::Virt::Domain::MIGRATE_AUTO_CONVERGE
Force convergance of the migration operation by throttling guest runtime

Sys::Virt::Domain::MIGRATE_RDMA_PIN_ALL
Pin memory for RDMA transfer

Sys::Virt::Domain::MIGRATE_POSTCOPY
Enable support for post-copy migration

Sys::Virt::Domain::MIGRATE_TLS
Setting this flag will cause the migration to attempt to use the TLS environment configured by the hypervisor in order to perform the migration. If incorrectly configured on either source or destination, the migration will fail.

Sys::Virt::Domain::MIGRATE_PARALLEL
Send memory pages to the destination host through several network connections. See `Sys::Virt::Domain::MIGRATE_PARAM_PARALLEL_*` parameters for configuring the parallel migration.

Sys::Virt::Domain::MIGRATE_NON_SHARED_SYNCHRONOUS_WRITES
Force the guest writes which happen when copying disk images for non-shared storage migration to be synchronously written to the destination. This ensures the storage migration converges for VMs doing heavy I/O on fast local storage and slow mirror.

UNDEFINE CONSTANTS

The following constants can be used when undefining virtual domain configurations

Sys::Virt::Domain::UNDEFINE_MANAGED_SAVE
Also remove any managed save image when undefining the virtual domain

Sys::Virt::Domain::UNDEFINE_SNAPSHOTS_METADATA
Also remove any snapshot metadata when undefining the virtual domain.

Sys::Virt::Domain::UNDEFINE_NVRAM
Also remove any NVRAM state file when undefining the virtual domain.

Sys::Virt::Domain::UNDEFINE_KEEP_NVRAM
keep NVRAM state file when undefining the virtual domain.

Sys::Virt::Domain::UNDEFINE_CHECKPOINTS_METADATA
Also remove any checkpoint metadata when undefining the virtual domain.

JOB TYPES

The following constants describe the different background job types.

Sys::Virt::Domain::JOB_NONE
No job is active

Sys::Virt::Domain::JOB_BOUNDED
A job with a finite completion time is active

Sys::Virt::Domain::JOB_UNBOUNDED
A job with an unbounded completion time is active

Sys::Virt::Domain::JOB_COMPLETED
The job has finished, but isn't cleaned up

Sys::Virt::Domain::JOB_FAILED
The job has hit an error, but isn't cleaned up

Sys::Virt::Domain::JOB_CANCELLED
The job was aborted at user request, but isn't cleaned up

MEMORY PARAMETERS

The following constants are useful when getting/setting memory parameters for guests

Sys::Virt::Domain::MEMORY_HARD_LIMIT
The maximum memory the guest can use.

Sys::Virt::Domain::MEMORY_SOFT_LIMIT
The memory upper limit enforced during memory contention.

Sys::Virt::Domain::MEMORY_MIN_GUARANTEE
The minimum memory guaranteed to be reserved for the guest.

Sys::Virt::Domain::MEMORY_SWAP_HARD_LIMIT
The maximum swap the guest can use.

Sys::Virt::Domain::MEMORY_PARAM_UNLIMITED
The value of an unlimited memory parameter

BLKIO PARAMETERS

The following parameters control I/O tuning for the domain as a whole

Sys::Virt::Domain::BLKIO_WEIGHT
The I/O weight parameter

Sys::Virt::Domain::BLKIO_DEVICE_WEIGHT
The per-device I/O weight parameter

Sys::Virt::Domain::BLKIO_DEVICE_READ_BPS
The per-device I/O bytes read per second

Sys::Virt::Domain::BLKIO_DEVICE_READ_IOPS
The per-device I/O operations read per second

Sys::Virt::Domain::BLKIO_DEVICE_WRITE_BPS
The per-device I/O bytes write per second

Sys::Virt::Domain::BLKIO_DEVICE_WRITE_IOPS
The per-device I/O operations write per second

BLKIO TUNING PARAMETERS

The following parameters control I/O tuning for an individual guest disk.

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_BYTES_SEC
The total bytes processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_BYTES_SEC
The bytes read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_BYTES_SEC
The bytes written per second.

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_IOPS_SEC
The total I/O operations processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_IOPS_SEC
The I/O operations read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_IOPS_SEC
The I/O operations written per second.

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_BYTES_SEC_MAX
The maximum total bytes processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_BYTES_SEC_MAX
The maximum bytes read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_BYTES_SEC_MAX
The maximum bytes written per second.

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_IOPS_SEC_MAX
The maximum total I/O operations processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_IOPS_SEC_MAX
The maximum I/O operations read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_IOPS_SEC_MAX
The maximum I/O operations written per second.

Sys::Virt::Domain::BLOCK_IOTUNE_SIZE_IOPS_SEC
The maximum I/O operations per second

Sys::Virt::Domain::BLOCK_IOTUNE_GROUP_NAME
A string representing a group name to allow sharing of I/O throttling quota between multiple drives

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum total bytes processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum bytes read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum bytes written per second.

Sys::Virt::Domain::BLOCK_IOTUNE_TOTAL_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum total I/O operations processed per second.

Sys::Virt::Domain::BLOCK_IOTUNE_READ_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum I/O operations read per second.

Sys::Virt::Domain::BLOCK_IOTUNE_WRITE_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum I/O operations written per second.

SCHEDULER CONSTANTS

Sys::Virt::Domain::SCHEDULER_CAP
The VM cap tunable

Sys::Virt::Domain::SCHEDULER_CPU_SHARES
The CPU shares tunable

Sys::Virt::Domain::SCHEDULER_LIMIT
The VM limit tunable

Sys::Virt::Domain::SCHEDULER_RESERVATION
The VM reservation tunable

Sys::Virt::Domain::SCHEDULER_SHARES
The VM shares tunable

Sys::Virt::Domain::SCHEDULER_VCPU_PERIOD
The VCPU period tunable

Sys::Virt::Domain::SCHEDULER_VCPU_QUOTA
The VCPU quota tunable

Sys::Virt::Domain::SCHEDULER_GLOBAL_PERIOD
The VM global period tunable

Sys::Virt::Domain::SCHEDULER_GLOBAL_QUOTA
The VM global quota tunable

Sys::Virt::Domain::SCHEDULER_WEIGHT
The VM weight tunable

NUMA PARAMETERS

The following constants are useful when getting/setting the guest NUMA memory policy

Sys::Virt::Domain::NUMA_MODE
The NUMA policy mode

Sys::Virt::Domain::NUMA_NODESET
The NUMA nodeset mask

The following constants are useful when interpreting the `Sys::Virt::Domain::NUMA_MODE` parameter value

Sys::Virt::Domain::NUMATUNE_MEM_STRICT
Allocation is mandatory from the mask nodes

Sys::Virt::Domain::NUMATUNE_MEM_PREFERRED
Allocation is preferred from the masked nodes

Sys::Virt::Domain::NUMATUNE_MEM_INTERLEAVE
Allocation is interleaved across all masked nodes

Sys::Virt::Domain::NUMATUNE_MEM_RESTRICTIVE
Allocation is determined by the host using the masked nodes.

INTERFACE PARAMETERS

The following constants are useful when getting/setting the per network interface tunable parameters

Sys::Virt::Domain::BANDWIDTH_IN_AVERAGE
The average inbound bandwidth

Sys::Virt::Domain::BANDWIDTH_IN_PEAK
The peak inbound bandwidth

Sys::Virt::Domain::BANDWIDTH_IN_BURST
The burstable inbound bandwidth

Sys::Virt::Domain::BANDWIDTH_IN_FLOOR
The minimum inbound bandwidth

Sys::Virt::Domain::BANDWIDTH_OUT_AVERAGE
The average outbound bandwidth

Sys::Virt::Domain::BANDWIDTH_OUT_PEAK
The peak outbound bandwidth

Sys::Virt::Domain::BANDWIDTH_OUT_BURST
The burstable outbound bandwidth

PERF EVENTS

The following constants defined performance events which can be monitored for a guest

Sys::Virt::Domain::PERF_PARAM_CMT
The CMT event counter which can be used to measure the usage of cache (bytes) by applications running on the platform. It corresponds to the ''perf.cmt'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_MBML
The MBML event counter which can be used to monitor the amount of data (bytes/s) sent through the memory controller on the socket. It corresponds to the ''perf.mbml'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_MBMT
The MBMT event counter which can be used to monitor total system bandwidth (bytes/s) from one level of cache to another. It corresponds to the ''perf.mbmt'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_CACHE_MISSES
The cache_misses perf event counter which can be used to measure the count of cache misses by applications running on the platform. It corresponds to the ''perf.cache_misses'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_CACHE_REFERENCES
The cache_references perf event counter which can be used to measure the count of cache hits by applications running on the platform. It corresponds to the ''perf.cache_references'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_CPU_CYCLES
The cpu_cycles perf event counter which can be used to measure how many cpu cycles one instruction needs. It corresponds to the ''perf.cpu_cycles'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_INSTRUCTIONS
The instructions perf event counter which can be used to measure the count of instructions by applications running on the platform. It corresponds to the ''perf.instructions'' field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_BRANCH_INSTRUCTIONS
The branch_instructions perf event counter which can be used to measure the count of instructions by applications running on the platform. It corresponds to the ‘‘perf.branch_instructions’’ field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_BRANCH_MISSES
The branch_misses perf event which can be used to measure the count of branch misses by applications running on the platform. It corresponds to the ‘‘perf.branch_misses’’ field in the *Stats APIs.

Sys::Virt::Domain::PERF_PARAM_BUS_CYCLES The bus_cycles perf event counter which can be used to measure the count of bus cycles by applications running on the platform. It corresponds to the ‘‘perf.bus_cycles’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_STALLED_CYCLES_FRONTEND The stalled_cycles_frontend perf event counter which can be used to measure the count of stalled cpu cycles in the frontend of the instruction processor pipeline by applications running on the platform. It corresponds to the ‘‘perf.stalled_cycles_frontend’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_STALLED_CYCLES_BACKEND The stalled_cycles_backend perf event counter which can be used to measure the count of stalled cpu cycles in the backend of the instruction processor pipeline by application running on the platform. It corresponds to the ‘‘perf.stalled_cycles_backend’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_REF_CPU_CYCLES The ref_cpu_cycles perf event counter which can be used to measure the count of total cpu cycles not affected by CPU frequency scaling by applications running on the platform. It corresponds to the ‘‘perf.ref_cpu_cycles’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_CPU_CLOCK The cpu_clock perf event counter which can be used to measure the count of cpu clock time by applications running on the platform. It corresponds to the ‘‘perf.cpu_clock’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_TASK_CLOCK The task_clock perf event counter which can be used to measure the count of task clock time by applications running on the platform. It corresponds to the ‘‘perf.task_clock’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_PAGE_FAULTS The page_faults perf event counter which can be used to measure the count of page faults by applications running on the platform. It corresponds to the ‘‘perf.page_faults’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_CONTEXT_SWITCHES The context_switches perf event counter which can be used to measure the count of context switches by applications running on the platform. It corresponds to the ‘‘perf.context_switches’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_CPU_MIGRATIONS The cpu_migrations perf event counter which can be used to measure the count of cpu migrations by applications running on the platform. It corresponds to the ‘‘perf.cpu_migrations’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_PAGE_FAULTS_MIN The page_faults_min perf event counter which can be used to measure the count of minor page faults by applications running on the platform. It corresponds to the ‘‘perf.page_faults_min’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_PAGE_FAULTS_MAJ The page_faults_maj perf event counter which can be used to measure the count of major page faults by applications running on the platform. It corresponds to the ‘‘perf.page_faults_maj’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_ALIGNMENT_FAULTS The alignment_faults perf event counter which can be used to measure the count of alignment faults by applications running on the platform. It corresponds to the ‘‘perf.alignment_faults’’ field in the *Stats APIs.
Sys::Virt::Domain::PERF_PARAM_EMULATION_FAULTS The emulation_faults perf event counter which can be used to measure the count of emulation faults by applications running on the platform. It corresponds to the ‘‘perf.emulation_faults’’ field in the *Stats APIs.

IOTHREAD STATS

The following constants defined IOThread statistics which can be monitored for a guest

Sys::Virt::Domain::IOTHREAD_PARAM_POLL_MAX_NS
The maximum polling time that can be used by polling algorithm in ns. The polling time starts at 0 (zero) and is the time spent by the guest to process IOThread data before returning the CPU to the host. The polling time will be dynamically modified over time based on the poll_grow and poll_shrink parameters provided.

Sys::Virt::Domain::IOTHREAD_PARAM_POLL_GROW
This provides a value for the dynamic polling adjustment algorithm to use to grow its polling interval up to the poll_max_ns value.

Sys::Virt::Domain::IOTHREAD_PARAM_POLL_SHRINK
This provides a value for the dynamic polling adjustment algorithm to use to shrink its polling interval when the polling interval exceeds the poll_max_ns value.

VCPU FLAGS

The following constants are useful when getting/setting the VCPU count for a guest

Sys::Virt::Domain::VCPU_LIVE
Flag to request the live value

Sys::Virt::Domain::VCPU_CONFIG
Flag to request the persistent config value

Sys::Virt::Domain::VCPU_CURRENT
Flag to request the current config value

Sys::Virt::Domain::VCPU_MAXIMUM
Flag to request adjustment of the maximum vCPU value

Sys::Virt::Domain::VCPU_GUEST
Flag to request the guest VCPU mask

Sys::Virt::Domain::VCPU_HOTPLUGGABLE
Flag to make vcpus added hot(un)pluggable

STATE CHANGE EVENTS

The following constants allow domain state change events to be interpreted. The events contain both a state change, and a reason.

Sys::Virt::Domain::EVENT_DEFINED
Indicates that a persistent configuration has been defined for the domain.

Sys::Virt::Domain::EVENT_DEFINED_ADDED
The defined configuration is newly added

Sys::Virt::Domain::EVENT_DEFINED_UPDATED
The defined configuration is an update to an existing configuration

Sys::Virt::Domain::EVENT_DEFINED_RENAMED
The defined configuration is a rename of an existing configuration

Sys::Virt::Domain::EVENT_DEFINED_FROM_SNAPSHOT
The defined configuration was restored from a snapshot

Sys::Virt::Domain::EVENT_RESUMED
The domain has resumed execution

Sys::Virt::Domain::EVENT_RESUMED_MIGRATED
The domain resumed because migration has completed. This is emitted on the destination host.

Sys::Virt::Domain::EVENT_RESUMED_UNPAUSED
The domain resumed because the admin unpaused it.

Sys::Virt::Domain::EVENT_RESUMED_FROM_SNAPSHOT
The domain resumed because it was restored from a snapshot

Sys::Virt::Domain::EVENT_RESUMED_POSTCOPY
The domain resumed but post-copy is running in background

Sys::Virt::Domain::EVENT_STARTED
The domain has started running

Sys::Virt::Domain::EVENT_STARTED_BOOTED
The domain was booted from shutoff state

Sys::Virt::Domain::EVENT_STARTED_MIGRATED
The domain started due to an incoming migration

Sys::Virt::Domain::EVENT_STARTED_RESTORED
The domain was restored from saved state file

Sys::Virt::Domain::EVENT_STARTED_FROM_SNAPSHOT
The domain was restored from a snapshot

Sys::Virt::Domain::EVENT_STARTED_WAKEUP
The domain was woken up from suspend

Sys::Virt::Domain::EVENT_STOPPED
The domain has stopped running

Sys::Virt::Domain::EVENT_STOPPED_CRASHED
The domain stopped because guest operating system has crashed

Sys::Virt::Domain::EVENT_STOPPED_DESTROYED
The domain stopped because administrator issued a destroy command.

Sys::Virt::Domain::EVENT_STOPPED_FAILED
The domain stopped because of a fault in the host virtualization environment.

Sys::Virt::Domain::EVENT_STOPPED_MIGRATED
The domain stopped because it was migrated to another machine.

Sys::Virt::Domain::EVENT_STOPPED_SAVED
The domain was saved to a state file

Sys::Virt::Domain::EVENT_STOPPED_SHUTDOWN
The domain stopped due to graceful shutdown of the guest.

Sys::Virt::Domain::EVENT_STOPPED_FROM_SNAPSHOT
The domain was stopped due to a snapshot

Sys::Virt::Domain::EVENT_SHUTDOWN
The domain has shutdown but is not yet stopped

Sys::Virt::Domain::EVENT_SHUTDOWN_FINISHED
The domain finished shutting down

Sys::Virt::Domain::EVENT_SHUTDOWN_HOST
The domain shutdown due to host trigger

Sys::Virt::Domain::EVENT_SHUTDOWN_GUEST
The domain shutdown due to guest trigger

Sys::Virt::Domain::EVENT_SUSPENDED
The domain has stopped executing, but still exists

Sys::Virt::Domain::EVENT_SUSPENDED_MIGRATED
The domain has been suspended due to offline migration

Sys::Virt::Domain::EVENT_SUSPENDED_PAUSED
The domain has been suspended due to administrator pause request.

Sys::Virt::Domain::EVENT_SUSPENDED_IOERROR
The domain has been suspended due to a block device I/O error.

Sys::Virt::Domain::EVENT_SUSPENDED_FROM_SNAPSHOT
The domain has been suspended due to resume from snapshot

Sys::Virt::Domain::EVENT_SUSPENDED_WATCHDOG
The domain has been suspended due to the watchdog triggering

Sys::Virt::Domain::EVENT_SUSPENDED_RESTORED
The domain has been suspended due to restore from saved state

Sys::Virt::Domain::EVENT_SUSPENDED_API_ERROR
The domain has been suspended due to an API error

Sys::Virt::Domain::EVENT_SUSPENDED_POSTCOPY
The domain has been suspended for post-copy migration

Sys::Virt::Domain::EVENT_SUSPENDED_POSTCOPY_FAILED
The domain has been suspended due post-copy migration failing

Sys::Virt::Domain::EVENT_UNDEFINED
The persistent configuration has gone away

Sys::Virt::Domain::EVENT_UNDEFINED_REMOVED
The domain configuration has gone away due to it being removed by administrator.

Sys::Virt::Domain::EVENT_UNDEFINED_RENAMED
The undefined configuration is a rename of an existing configuration

Sys::Virt::Domain::EVENT_PMSUSPENDED
The domain has stopped running

Sys::Virt::Domain::EVENT_PMSUSPENDED_MEMORY
The domain has suspend to RAM.

Sys::Virt::Domain::EVENT_PMSUSPENDED_DISK
The domain has suspend to Disk.

Sys::Virt::Domain::EVENT_CRASHED
The domain has crashed

Sys::Virt::Domain::EVENT_CRASHED_PANICKED
The domain has crashed due to a kernel panic

Sys::Virt::Domain::EVENT_CRASHED_CRASHLOADED
The domain has crashed and reloaded itself

EVENT ID CONSTANTS

Sys::Virt::Domain::EVENT_ID_LIFECYCLE
Domain lifecycle events

Sys::Virt::Domain::EVENT_ID_REBOOT
Soft / warm reboot events

Sys::Virt::Domain::EVENT_ID_RTC_CHANGE
RTC clock adjustments

Sys::Virt::Domain::EVENT_ID_IO_ERROR
File IO errors, typically from disks

Sys::Virt::Domain::EVENT_ID_WATCHDOG
Watchdog device triggering

Sys::Virt::Domain::EVENT_ID_GRAPHICS
Graphics client connections.

Sys::Virt::Domain::EVENT_ID_IO_ERROR_REASON
File IO errors, typically from disks, with a root cause

Sys::Virt::Domain::EVENT_ID_CONTROL_ERROR
Errors from the virtualization control channel

Sys::Virt::Domain::EVENT_ID_BLOCK_JOB
Completion status of asynchronous block jobs, identified by source file name.

Sys::Virt::Domain::EVENT_ID_BLOCK_JOB_2
Completion status of asynchronous block jobs, identified by target device name.

Sys::Virt::Domain::EVENT_ID_DISK_CHANGE
Changes in disk media

Sys::Virt::Domain::EVENT_ID_TRAY_CHANGE
CDROM media tray state

Sys::Virt::Domain::EVENT_ID_PMSUSPEND
Power management initiated suspend to RAM

Sys::Virt::Domain::EVENT_ID_PMSUSPEND_DISK
Power management initiated suspend to Disk

Sys::Virt::Domain::EVENT_ID_PMWAKEUP
Power management initiated wakeup

Sys::Virt::Domain::EVENT_ID_BALLOON_CHANGE
Balloon target changes

Sys::Virt::Domain::EVENT_ID_DEVICE_ADDED
Asynchronous guest device addition

Sys::Virt::Domain::EVENT_ID_DEVICE_REMOVED
Asynchronous guest device removal

Sys::Virt::Domain::EVENT_ID_TUNABLE
Changes of any domain tuning parameters. The callback will be provided with a hash listing all changed parameters. The later DOMAIN TUNABLE constants can be useful when accessing the hash keys

Sys::Virt::Domain::EVENT_ID_AGENT_LIFECYCLE
Domain guest agent lifecycle events. The `state` parameter to the callback will match one of the constants

Sys::Virt::Domain::EVENT_AGENT_LIFECYCLE_STATE_CONNECTED
The agent is now connected

Sys::Virt::Domain::EVENT_AGENT_LIFECYCLE_STATE_DISCONNECTED
The agent is now disconnected

The second parameter, `reason`, matches one of the following constants

Sys::Virt::Domain::EVENT_ID_MIGRATION_ITERATION
Domain migration progress iteration. The `iteration` parameter to the callback will specify the number of iterations migration has made over guest RAM.

Sys::Virt::Domain::EVENT_AGENT_LIFECYCLE_REASON_UNKNOWN
The reason is unknown

Sys::Virt::Domain::EVENT_AGENT_LIFECYCLE_REASON_DOMAIN_STARTED
The domain was initially booted

Sys::Virt::Domain::EVENT_AGENT_LIFECYCLE_REASON_CHANNEL
The channel on a running guest changed state

Sys::Virt::Domain::EVENT_ID_JOB_COMPLETED
Domain background job completion notification. The callback provides a hash containing the job stats. The keyus in the hash are the same as those used with the `Sys::Virt::Domain::get_job_stats()` method.

Sys::Virt::Domain::EVENT_ID_DEVICE_REMOVAL_FAILED
Guest device removal has failed.

Sys::Virt::Domain::EVENT_ID_METADATA_CHANGE
The domain metadata has changed

Sys::Virt::Domain::EVENT_ID_BLOCK_THRESHOLD
The event occurs when the hypervisor detects that the given storage element was written beyond the point specified by threshold. The event is useful for thin-provisioned storage.

Sys::Virt::Domain::EVENT_ID_MEMORY_FAILURE
The event occurs when the hypervisor detects hardware memory corruption.

Sys::Virt::Domain::EVENT_ID_MEMORY_DEVICE_SIZE_CHANGE
The event occurs when the guest accepts a request to change the memory device size.

IO ERROR EVENT CONSTANTS

These constants describe what action was taken due to the IO error.

Sys::Virt::Domain::EVENT_IO_ERROR_NONE
No action was taken, the error was ignored & reported as success to guest

Sys::Virt::Domain::EVENT_IO_ERROR_PAUSE
The guest is paused since the error occurred

Sys::Virt::Domain::EVENT_IO_ERROR_REPORT
The error has been reported to the guest OS

WATCHDOG EVENT CONSTANTS

These constants describe what action was taken due to the watchdog firing

Sys::Virt::Domain::EVENT_WATCHDOG_NONE
No action was taken, the watchdog was ignored

Sys::Virt::Domain::EVENT_WATCHDOG_PAUSE
The guest is paused since the watchdog fired

Sys::Virt::Domain::EVENT_WATCHDOG_POWEROFF
The guest is powered off after the watchdog fired

Sys::Virt::Domain::EVENT_WATCHDOG_RESET
The guest is reset after the watchdog fired

Sys::Virt::Domain::EVENT_WATCHDOG_SHUTDOWN
The guest attempted to gracefully shutdown after the watchdog fired

Sys::Virt::Domain::EVENT_WATCHDOG_DEBUG
No action was taken, the watchdog was logged

Sys::Virt::Domain::EVENT_WATCHDOG_INJECTNMI
An NMI was injected into the guest after the watchdog fired

GRAPHICS EVENT PHASE CONSTANTS

These constants describe the phase of the graphics connection

Sys::Virt::Domain::EVENT_GRAPHICS_CONNECT
The initial client connection

Sys::Virt::Domain::EVENT_GRAPHICS_INITIALIZE
The client has been authenticated & the connection is running

Sys::Virt::Domain::EVENT_GRAPHICS_DISCONNECT
The client has disconnected

GRAPHICS EVENT ADDRESS CONSTANTS

These constants describe the format of the address

Sys::Virt::Domain::EVENT_GRAPHICS_ADDRESS_IPV4
An IPv4 address

Sys::Virt::Domain::EVENT_GRAPHICS_ADDRESS_IPV6
An IPv6 address

Sys::Virt::Domain::EVENT_GRAPHICS_ADDRESS_UNIX
An UNIX socket path address

DISK CHANGE EVENT CONSTANTS

These constants describe the reason for a disk change event

Sys::Virt::Domain::EVENT_DISK_CHANGE_MISSING_ON_START
The disk media was cleared, as its source was missing when attempting to start the guest

Sys::Virt::Domain::EVENT_DISK_DROP_MISSING_ON_START
The disk device was dropped, as its source was missing when attempting to start the guest

TRAY CHANGE CONSTANTS

These constants describe the reason for a tray change event

Sys::Virt::Domain::EVENT_TRAY_CHANGE_CLOSE
The tray was closed

Sys::Virt::Domain::EVENT_TRAY_CHANGE_OPEN
The tray was opened

DOMAIN BLOCK JOB TYPE CONSTANTS

The following constants identify the different types of domain block jobs

Sys::Virt::Domain::BLOCK_JOB_TYPE_UNKNOWN
An unknown block job type

Sys::Virt::Domain::BLOCK_JOB_TYPE_PULL
The block pull job type

Sys::Virt::Domain::BLOCK_JOB_TYPE_COPY
The block copy job type

Sys::Virt::Domain::BLOCK_JOB_TYPE_COMMIT
The block commit job type

Sys::Virt::Domain::BLOCK_JOB_TYPE_ACTIVE_COMMIT
The block active commit job type

Sys::Virt::Domain::BLOCK_JOB_TYPE_BACKUP
The block backup job type

DOMAIN BLOCK JOB COMPLETION CONSTANTS

The following constants can be used to determine the completion status of a block job

Sys::Virt::Domain::BLOCK_JOB_COMPLETED
A successfully completed block job

Sys::Virt::Domain::BLOCK_JOB_FAILED
An unsuccessful block job

Sys::Virt::Domain::BLOCK_JOB_CANCELED
A block job canceled byy the user

Sys::Virt::Domain::BLOCK_JOB_READY
A block job is running

DOMAIN BLOCK REBASE CONSTANTS

The following constants are useful when rebasing block devices

Sys::Virt::Domain::BLOCK_REBASE_SHALLOW
Limit copy to top of source backing chain

Sys::Virt::Domain::BLOCK_REBASE_REUSE_EXT
Reuse existing external file for copy

Sys::Virt::Domain::BLOCK_REBASE_COPY_RAW
Make destination file raw

Sys::Virt::Domain::BLOCK_REBASE_COPY
Start a copy job

Sys::Virt::Domain::BLOCK_REBASE_COPY_DEV
Treat destination as a block device instead of file

Sys::Virt::Domain::BLOCK_REBASE_RELATIVE
Keep backing chain referenced using relative names

DOMAIN BLOCK COPY CONSTANTS

The following constants are useful when copying block devices

Sys::Virt::Domain::BLOCK_COPY_SHALLOW
Limit copy to top of source backing chain

Sys::Virt::Domain::BLOCK_COPY_REUSE_EXT
Reuse existing external file for copy

Sys::Virt::Domain::BLOCK_COPY_TRANSIENT_JOB
Don't force usage of recoverable job for the copy operation

Sys::Virt::Domain::BLOCK_COPY_SYNCHRONOUS_WRITES
Force the copy job to synchronously propagate guest writes into the destination image, so that the copy is guaranteed to converge

DOMAIN BLOCK JOB ABORT CONSTANTS

The following constants are useful when aborting job copy jobs

Sys::Virt::Domain::BLOCK_JOB_ABORT_ASYNC
Request only, do not wait for completion

Sys::Virt::Domain::BLOCK_JOB_ABORT_PIVOT
Pivot to mirror when ending a copy job

DOMAIN BLOCK COMMIT JOB CONSTANTS

The following constants are useful with block commit job types

Sys::Virt::Domain::BLOCK_COMMIT_DELETE
Delete any files that are invalid after commit

Sys::Virt::Domain::BLOCK_COMMIT_SHALLOW
NULL base means next backing file, not whole chain

Sys::Virt::Domain::BLOCK_COMMIT_ACTIVE
Allow two phase commit when top is active layer

Sys::Virt::Domain::BLOCK_COMMIT_RELATIVE
Keep backing chain referenced using relative names

DOMAIN SAVE / RESTORE CONSTANTS

The following constants can be used when saving or restoring virtual machines

Sys::Virt::Domain::SAVE_BYPASS_CACHE
Do not use OS I/O cache when saving state.

Sys::Virt::Domain::SAVE_PAUSED
Mark the saved state as paused to prevent the guest CPUs starting upon restore.

Sys::Virt::Domain::SAVE_RUNNING
Mark the saved state as running to allow the guest CPUs to start upon restore.

DOMAIN CORE DUMP CONSTANTS

The following constants can be used when triggering domain core dumps

Sys::Virt::Domain::DUMP_LIVE
Do not pause execution while dumping the guest

Sys::Virt::Domain::DUMP_CRASH
Crash the guest after completing the core dump

Sys::Virt::Domain::DUMP_BYPASS_CACHE
Do not use OS I/O cache when writing core dump

Sys::Virt::Domain::DUMP_RESET
Reset the virtual machine after finishing the dump

Sys::Virt::Domain::DUMP_MEMORY_ONLY
Only include guest RAM in the dump, not the device state

DESTROY CONSTANTS

The following constants are useful when terminating guests using the `destroy` API.

Sys::Virt::Domain::DESTROY_DEFAULT
Destroy the guest using the default approach

Sys::Virt::Domain::DESTROY_GRACEFUL
Destroy the guest in a graceful manner

SHUTDOWN CONSTANTS

The following constants are useful when requesting that a guest terminate using the `shutdown` API

Sys::Virt::Domain::SHUTDOWN_DEFAULT
Shutdown using the hypervisor's default mechanism

Sys::Virt::Domain::SHUTDOWN_GUEST_AGENT
Shutdown by issuing a command to a guest agent

Sys::Virt::Domain::SHUTDOWN_ACPI_POWER_BTN
Shutdown by injecting an ACPI power button press

Sys::Virt::Domain::SHUTDOWN_INITCTL
Shutdown by talking to initctl (containers only)

Sys::Virt::Domain::SHUTDOWN_SIGNAL
Shutdown by sending SIGTERM to the init process

Sys::Virt::Domain::SHUTDOWN_PARAVIRT
Shutdown by issuing a paravirt power control command

REBOOT CONSTANTS

The following constants are useful when requesting that a guest terminate using the `reboot` API

Sys::Virt::Domain::REBOOT_DEFAULT
Reboot using the hypervisor's default mechanism

Sys::Virt::Domain::REBOOT_GUEST_AGENT
Reboot by issuing a command to a guest agent

Sys::Virt::Domain::REBOOT_ACPI_POWER_BTN
Reboot by injecting an ACPI power button press

Sys::Virt::Domain::REBOOT_INITCTL
Reboot by talking to initctl (containers only)

Sys::Virt::Domain::REBOOT_SIGNAL
Reboot by sending SIGHUP to the init process

Sys::Virt::Domain::REBOOT_PARAVIRT
Reboot by issuing a paravirt power control command

METADATA CONSTANTS

The following constants are useful when reading/writing metadata about a guest

Sys::Virt::Domain::METADATA_TITLE
The short human friendly title of the guest

Sys::Virt::Domain::METADATA_DESCRIPTION
The long free text description of the guest

Sys::Virt::Domain::METADATA_ELEMENT
The structured metadata elements for the guest

DISK ERROR CONSTANTS

The following constants are useful when interpreting disk error codes

Sys::Virt::Domain::DISK_ERROR_NONE
No error

Sys::Virt::Domain::DISK_ERROR_NO_SPACE
The host storage has run out of free space

Sys::Virt::Domain::DISK_ERROR_UNSPEC
An unspecified error has occurred.

MEMORY STATISTIC CONSTANTS

Sys::Virt::Domain::MEMORY_STAT_SWAP_IN
Swap in

Sys::Virt::Domain::MEMORY_STAT_SWAP_OUT
Swap out

Sys::Virt::Domain::MEMORY_STAT_MINOR_FAULT
Minor faults

Sys::Virt::Domain::MEMORY_STAT_MAJOR_FAULT
Major faults

Sys::Virt::Domain::MEMORY_STAT_RSS
Resident memory

Sys::Virt::Domain::MEMORY_STAT_UNUSED
Unused memory

Sys::Virt::Domain::MEMORY_STAT_AVAILABLE
Available memory

Sys::Virt::Domain::MEMORY_STAT_ACTUAL_BALLOON
Actual balloon limit

Sys::Virt::Domain::MEMORY_STAT_USABLE
Amount of usable memory

Sys::Virt::Domain::MEMORY_STAT_LAST_UPDATE
Time of last stats refresh from guest

Sys::Virt::Domain::MEMORY_STAT_DISK_CACHES
Disk cache size

Sys::Virt::Domain::MEMORY_STAT_HUGETLB_PGALLOC
The amount of successful huge page allocations

Sys::Virt::Domain::MEMORY_STAT_HUGETLB_PGFAIL
The amount of failed huge page allocations

DOMAIN LIST CONSTANTS

The following constants can be used when listing domains

Sys::Virt::Domain::LIST_ACTIVE
Only list domains that are currently active (running, or paused)

Sys::Virt::Domain::LIST_AUTOSTART
Only list domains that are set to automatically start on boot

Sys::Virt::Domain::LIST_HAS_SNAPSHOT
Only list domains that have a stored snapshot

Sys::Virt::Domain::LIST_INACTIVE
Only list domains that are currently inactive (shutoff, saved)

Sys::Virt::Domain::LIST_MANAGEDSAVE
Only list domains that have current managed save state

Sys::Virt::Domain::LIST_NO_AUTOSTART
Only list domains that are not set to automatically start on boto

Sys::Virt::Domain::LIST_NO_MANAGEDSAVE
Only list domains that do not have any managed save state

Sys::Virt::Domain::LIST_NO_SNAPSHOT
Only list domains that do not have a stored snapshot

Sys::Virt::Domain::LIST_OTHER
Only list domains that are not running, paused or shutoff

Sys::Virt::Domain::LIST_PAUSED
Only list domains that are paused

Sys::Virt::Domain::LIST_PERSISTENT
Only list domains which have a persistent config

Sys::Virt::Domain::LIST_RUNNING
Only list domains that are currently running

Sys::Virt::Domain::LIST_SHUTOFF
Only list domains that are currently shutoff

Sys::Virt::Domain::LIST_TRANSIENT
Only list domains that do not have a persistent config

Sys::Virt::Domain::LIST_HAS_CHECKPOINT
Only list domains that have a stored checkpoint

Sys::Virt::Domain::LIST_NO_CHECKPOINT
Only list domains that do not have a stored checkpoint

SEND KEY CONSTANTS

The following constants are to be used with the `send_key` API

Sys::Virt::Domain::SEND_KEY_MAX_KEYS
The maximum number of keys that can be sent in a single call to `send_key`

BLOCK STATS CONSTANTS

The following constants provide the names of well known block stats fields

Sys::Virt::Domain::BLOCK_STATS_ERRS
: The number of I/O errors

Sys::Virt::Domain::BLOCK_STATS_FLUSH_REQ
: The number of flush requests

Sys::Virt::Domain::BLOCK_STATS_FLUSH_TOTAL_TIMES
: The time spent processing flush requests

Sys::Virt::Domain::BLOCK_STATS_READ_BYTES
: The amount of data read

Sys::Virt::Domain::BLOCK_STATS_READ_REQ
: The number of read requests

Sys::Virt::Domain::BLOCK_STATS_READ_TOTAL_TIMES
: The time spent processing read requests

Sys::Virt::Domain::BLOCK_STATS_WRITE_BYTES
: The amount of data written

Sys::Virt::Domain::BLOCK_STATS_WRITE_REQ
: The number of write requests

Sys::Virt::Domain::BLOCK_STATS_WRITE_TOTAL_TIMES
: The time spent processing write requests

CPU STATS CONSTANTS

The following constants provide the names of well known cpu stats fields

Sys::Virt::Domain::CPU_STATS_CPUTIME
: The total CPU time, including both hypervisor and vCPU time.

Sys::Virt::Domain::CPU_STATS_USERTIME
: THe total time in kernel

Sys::Virt::Domain::CPU_STATS_SYSTEMTIME
: The total time in userspace

Sys::Virt::Domain::CPU_STATS_VCPUTIME
: The total vCPU time.

CPU STATS CONSTANTS

The following constants provide the names of well known schedular parameters

Sys::Virt::SCHEDULER_EMULATOR_PERIOD
: The duration of the time period for scheduling the emulator

Sys::Virt::SCHEDULER_EMULATOR_QUOTA
: The quota for the emulator in one schedular time period

Sys::Virt::SCHEDULER_IOTHREAD_PERIOD
: The duration of the time period for scheduling the iothread

Sys::Virt::SCHEDULER_IOTHREAD_QUOTA
: The quota for the iothread in one schedular time period

DOMAIN STATS FLAG CONSTANTS

The following constants are used as flags when requesting bulk domain stats from `Sys::Virt::get_all_domain_stats`.

Sys::Virt::Domain::GET_ALL_STATS_ACTIVE
: Include stats for active domains

Sys::Virt::Domain::GET_ALL_STATS_INACTIVE
Include stats for inactive domains

Sys::Virt::Domain::GET_ALL_STATS_OTHER
Include stats for other domains

Sys::Virt::Domain::GET_ALL_STATS_PAUSED
Include stats for paused domains

Sys::Virt::Domain::GET_ALL_STATS_PERSISTENT
Include stats for persistent domains

Sys::Virt::Domain::GET_ALL_STATS_RUNNING
Include stats for running domains

Sys::Virt::Domain::GET_ALL_STATS_SHUTOFF
Include stats for shutoff domains

Sys::Virt::Domain::GET_ALL_STATS_TRANSIENT
Include stats for transient domains

Sys::Virt::Domain::GET_ALL_STATS_ENFORCE_STATS
Require that all requested stats fields are returned

Sys::Virt::Domain::GET_ALL_STATS_BACKING
Get stats for image backing files too

Sys::Virt::Domain::GET_ALL_STATS_NOWAIT
Skip stats if they can't be acquired without waiting

DOMAIN STATS FIELD CONSTANTS

The following constants are used to control which fields are returned for stats queries.

Sys::Virt::Domain::STATS_BALLOON
Balloon statistics

Sys::Virt::Domain::STATS_BLOCK
Block device info

Sys::Virt::Domain::STATS_CPU_TOTAL
CPU usage info

Sys::Virt::Domain::STATS_INTERFACE
Network interface info

Sys::Virt::Domain::STATS_STATE
General lifecycle state

Sys::Virt::Domain::STATS_VCPU
Virtual CPU info

Sys::Virt::Domain::STATS_PERF
Performance event counter values

Sys::Virt::Domain::STATS_IOTHREAD
IOThread performance statistics values

Sys::Virt::Domain::STATS_MEMORY
Memory bandwidth statistics values

Sys::Virt::Domain::STATS_DIRTYRATE
Memory dirty rate statistics

PROCESS SIGNALS

The following constants provide the names of signals which can be sent to guest processes. They mostly correspond to POSIX signal names.

Sys::Virt::Domain::PROCESS_SIGNAL_NOP
SIGNOP

Sys::Virt::Domain::PROCESS_SIGNAL_HUP
SIGHUP

Sys::Virt::Domain::PROCESS_SIGNAL_INT
SIGINT

Sys::Virt::Domain::PROCESS_SIGNAL_QUIT
SIGQUIT

Sys::Virt::Domain::PROCESS_SIGNAL_ILL
SIGILL

Sys::Virt::Domain::PROCESS_SIGNAL_TRAP
SIGTRAP

Sys::Virt::Domain::PROCESS_SIGNAL_ABRT
SIGABRT

Sys::Virt::Domain::PROCESS_SIGNAL_BUS
SIGBUS

Sys::Virt::Domain::PROCESS_SIGNAL_FPE
SIGFPE

Sys::Virt::Domain::PROCESS_SIGNAL_KILL
SIGKILL

Sys::Virt::Domain::PROCESS_SIGNAL_USR1
SIGUSR1

Sys::Virt::Domain::PROCESS_SIGNAL_SEGV
SIGSEGV

Sys::Virt::Domain::PROCESS_SIGNAL_USR2
SIGUSR2

Sys::Virt::Domain::PROCESS_SIGNAL_PIPE
SIGPIPE

Sys::Virt::Domain::PROCESS_SIGNAL_ALRM
SIGALRM

Sys::Virt::Domain::PROCESS_SIGNAL_TERM
SIGTERM

Sys::Virt::Domain::PROCESS_SIGNAL_STKFLT
SIGSTKFLT

Sys::Virt::Domain::PROCESS_SIGNAL_CHLD
SIGCHLD

Sys::Virt::Domain::PROCESS_SIGNAL_CONT
SIGCONT

Sys::Virt::Domain::PROCESS_SIGNAL_STOP
SIGSTOP

Sys::Virt::Domain::PROCESS_SIGNAL_TSTP
SIGTSTP

Sys::Virt::Domain::PROCESS_SIGNAL_TTIN
SIGTTIN

Sys::Virt::Domain::PROCESS_SIGNAL_TTOU
SIGTTOU

Sys::Virt::Domain::PROCESS_SIGNAL_URG
SIGURG

Sys::Virt::Domain::PROCESS_SIGNAL_XCPU
SIGXCPU

Sys::Virt::Domain::PROCESS_SIGNAL_XFSZ
SIGXFSZ

Sys::Virt::Domain::PROCESS_SIGNAL_VTALRM
SIGVTALRM

Sys::Virt::Domain::PROCESS_SIGNAL_PROF
SIGPROF

Sys::Virt::Domain::PROCESS_SIGNAL_WINCH
SIGWINCH

Sys::Virt::Domain::PROCESS_SIGNAL_POLL
SIGPOLL

Sys::Virt::Domain::PROCESS_SIGNAL_PWR
SIGPWR

Sys::Virt::Domain::PROCESS_SIGNAL_SYS
SIGSYS

Sys::Virt::Domain::PROCESS_SIGNAL_RT0
SIGRT0

Sys::Virt::Domain::PROCESS_SIGNAL_RT1
SIGRT1

Sys::Virt::Domain::PROCESS_SIGNAL_RT2
SIGRT2

Sys::Virt::Domain::PROCESS_SIGNAL_RT3
SIGRT3

Sys::Virt::Domain::PROCESS_SIGNAL_RT4
SIGRT4

Sys::Virt::Domain::PROCESS_SIGNAL_RT5
SIGRT5

Sys::Virt::Domain::PROCESS_SIGNAL_RT6
SIGRT6

Sys::Virt::Domain::PROCESS_SIGNAL_RT7
SIGRT7

Sys::Virt::Domain::PROCESS_SIGNAL_RT8
SIGRT8

Sys::Virt::Domain::PROCESS_SIGNAL_RT9
SIGRT9

Sys::Virt::Domain::PROCESS_SIGNAL_RT10
SIGRT10

Sys::Virt::Domain::PROCESS_SIGNAL_RT11
SIGRT11

Sys::Virt::Domain::PROCESS_SIGNAL_RT12
SIGRT12

Sys::Virt::Domain::PROCESS_SIGNAL_RT13
SIGRT13

Sys::Virt::Domain::PROCESS_SIGNAL_RT14
SIGRT14

Sys::Virt::Domain::PROCESS_SIGNAL_RT15
SIGRT15

Sys::Virt::Domain::PROCESS_SIGNAL_RT16
SIGRT16

Sys::Virt::Domain::PROCESS_SIGNAL_RT17
SIGRT17

Sys::Virt::Domain::PROCESS_SIGNAL_RT18
SIGRT18

Sys::Virt::Domain::PROCESS_SIGNAL_RT19
SIGRT19

Sys::Virt::Domain::PROCESS_SIGNAL_RT20
SIGRT20

Sys::Virt::Domain::PROCESS_SIGNAL_RT21
SIGRT21

Sys::Virt::Domain::PROCESS_SIGNAL_RT22
SIGRT22

Sys::Virt::Domain::PROCESS_SIGNAL_RT23
SIGRT23

Sys::Virt::Domain::PROCESS_SIGNAL_RT24
SIGRT24

Sys::Virt::Domain::PROCESS_SIGNAL_RT25
SIGRT25

Sys::Virt::Domain::PROCESS_SIGNAL_RT26
SIGRT26

Sys::Virt::Domain::PROCESS_SIGNAL_RT27
SIGRT27

Sys::Virt::Domain::PROCESS_SIGNAL_RT28
SIGRT28

Sys::Virt::Domain::PROCESS_SIGNAL_RT29
SIGRT29

Sys::Virt::Domain::PROCESS_SIGNAL_RT30
SIGRT30

Sys::Virt::Domain::PROCESS_SIGNAL_RT31
SIGRT31

Sys::Virt::Domain::PROCESS_SIGNAL_RT32
SIGRT32

DOMAIN TUNABLE CONSTANTS

The following constants are useful when accessing domain tuning parameters in APIs and events

Sys::Virt::Domain::TUNABLE_CPU_CPU_SHARES
Proportional CPU weight

Sys::Virt::Domain::TUNABLE_CPU_EMULATORPIN
Emulator thread CPU pinning mask

Sys::Virt::Domain::TUNABLE_CPU_EMULATOR_PERIOD
Emulator thread CPU period

Sys::Virt::Domain::TUNABLE_CPU_EMULATOR_QUOTA
Emulator thread CPU quota

Sys::Virt::Domain::TUNABLE_CPU_IOTHREAD_PERIOD
Iothread thread CPU period

Sys::Virt::Domain::TUNABLE_CPU_IOTHREAD_QUOTA
Iothread thread CPU quota

Sys::Virt::Domain::TUNABLE_CPU_VCPUPIN
VCPU thread pinning mask

Sys::Virt::Domain::TUNABLE_CPU_VCPU_PERIOD
VCPU thread period

Sys::Virt::Domain::TUNABLE_CPU_VCPU_QUOTA
VCPU thread quota

Sys::Virt::Domain::TUNABLE_CPU_GLOBAL_PERIOD
VM global period

Sys::Virt::Domain::TUNABLE_CPU_GLOBAL_QUOTA
VM global quota

Sys::Virt::Domain::TUNABLE_BLKDEV_DISK
The name of guest disks

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_BYTES_SEC
Read throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_IOPS_SEC
Read throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_BYTES_SEC
Total throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_IOPS_SEC
Total throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_BYTES_SEC
Write throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_IOPS_SEC
Write throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_BYTES_SEC_MAX
Maximum read throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_IOPS_SEC_MAX
Maximum read throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_BYTES_SEC_MAX
Maximum total throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_IOPS_SEC_MAX
Maximum total throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_BYTES_SEC_MAX
Maximum write throughput in bytes per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_IOPS_SEC_MAX
Maximum write throughput in I/O operations per sec

Sys::Virt::Domain::TUNABLE_BLKDEV_SIZE_IOPS_SEC
The maximum I/O operations per second

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum total bytes processed per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum bytes read per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_BYTES_SEC_MAX_LENGTH
The duration in seconds allowed for maximum bytes written per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_TOTAL_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum total I/O operations processed per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_READ_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum I/O operations read per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_WRITE_IOPS_SEC_MAX_LENGTH
The duration in seconds allowed for maximum I/O operations written per second.

Sys::Virt::Domain::TUNABLE_BLKDEV_GROUP_NAME
The name of the blkdev group

Sys::Virt::Domain::TUNABLE_IOTHREADSPIN
The I/O threads pinning

DOMAIN LIFECYCLE CONSTANTS

The following constants are useful when setting action for lifecycle events.

Sys::Virt::Domain::LIFECYCLE_POWEROFF
The poweroff lifecycle event type

Sys::Virt::Domain::LIFECYCLE_REBOOT
The reboot lifecycle event type

Sys::Virt::Domain::LIFECYCLE_CRASH
The crash lifecycle event type

DOMAIN LIFECYCLE ACTION CONSTANTS

Sys::Virt::Domain::LIFECYCLE_ACTION_DESTROY
The destroy lifecycle action

Sys::Virt::Domain::LIFECYCLE_ACTION_RESTART
The restart lifecycle action

Sys::Virt::Domain::LIFECYCLE_ACTION_RESTART_RENAME
The restart-rename lifecycle action

Sys::Virt::Domain::LIFECYCLE_ACTION_PRESERVE
The preserve lifecycle action

Sys::Virt::Domain::LIFECYCLE_ACTION_COREDUMP_DESTROY
The coredump-destroy lifecycle action

Sys::Virt::Domain::LIFECYCLE_ACTION_COREDUMP_RESTART
The coredump-restart lifecycle action

MEMORY FAILURE ACTION CONSTANTS

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_ACTION_IGNORE
The failure could be ignored

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_ACTION_INJECT
An MCE was injected to the guest

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_ACTION_FATAL
The failure is non-recoverable and the hypervisor was not able to handle it

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_ACTION_RESET
The failure is non-recoverable and the guest was not able to handle it.

MEMORY FAILURE RECIPIENT CONSTANTS

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_RECIPIENT_HYPERVISOR
The memory failure was in hypervisor address space

Sys::Virt::Domain::EVENT_MEMORY_FAILURE_RECIPIENT_GUEST
The memory failure was in guest address space

MEMORY FAILURE FLAG CONSTANTS

Sys::Virt::Domain::MEMORY_FAILURE_ACTION_REQUIRED
Whether the flag is action-required or action-optional

Sys::Virt::Domain::MEMORY_FAILURE_RECURSIVE
The failure occurred while the previous fault was being handled.

MEMORY DIRTY RATE STATUS CONSTANTS

Sys::Virt::Domain::DIRTYRATE_UNSTARTED
The dirty rate is not being measured currently.

Sys::Virt::Domain::DIRTYRATE_MEASURING
The dity rate is in the process of being measured

Sys::Virt::Domain::DIRTYRATE_MEASURED
The dirty rate has been measured

AUTHORS

Daniel P. Berrange <berrange@redhat.com>

COPYRIGHT

LICENSE

SEE ALSO

Sys::Virt, Sys::Virt::Error, `http://libvirt.org`