NAME

qemu−doc − QEMU version 4.2.1 User Documentation

SYNOPSIS

qemu−system−x86_64 [*options*] [*disk_image*]

DESCRIPTION

The QEMU PC System emulator simulates the following peripherals:

- i440FX host PCI bridge and PIIX3 PCI to ISA bridge
- Cirrus CLGD 5446 PCI VGA card or dummy VGA card with Bochs VESA extensions (hardware level, including all non standard modes).
- PS/2 mouse and keyboard
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- PCI and ISA network adapters
- Serial ports
- IPMI BMC, either and internal or external one
- Creative SoundBlaster 16 sound card
- ENSONIQ AudioPCI ES1370 sound card
- Intel 82801AA AC97 Audio compatible sound card
- Intel HD Audio Controller and HDA codec
- Adlib (OPL2) − Yamaha YM3812 compatible chip
- Gravis Ultrasound GF1 sound card
- CS4231A compatible sound card
- PCI UHCI, OHCI, EHCI or XHCI USB controller and a virtual USB−1.1 hub.

SMP is supported with up to 255 CPUs.

QEMU uses the PC BIOS from the Seabios project and the Plex86/Bochs LGPL VGA BIOS.

QEMU uses YM3812 emulation by Tatsuyuki Satoh.

QEMU uses GUS emulation (GUSEMU32 <**http://www.deinmeister.de/gusemu/**>) by Tibor ‘‘TS’’ Schütz.

Note that, by default, GUS shares **IRQ** (7) with parallel ports and so QEMU must be told to not have parallel ports to have working GUS.

```
        qemu-system-x86_64 dos.img -soundhw gus -parallel none
```

Alternatively:

```
        qemu-system-x86_64 dos.img -device gus,irq=5
```

Or some other unclaimed IRQ.

CS4231A is the chip used in Windows Sound System and GUSMAX products

OPTIONS

disk_image is a raw hard disk image for IDE hard disk 0. Some targets do not need a disk image.

Standard options

−h Display help and exit

−version
Display version information and exit

−machine [type=]*name*[,prop=*value*[,...]]

Select the emulated machine by *name*. Use `-machine help` to list available machines.

For architectures which aim to support live migration compatibility across releases, each release will introduce a new versioned machine type. For example, the 2.8.0 release introduced machine types "pc−i440fx−2.8" and "pc−q35−2.8" for the x86_64/i686 architectures.

To allow live migration of guests from QEMU version 2.8.0, to QEMU version 2.9.0, the 2.9.0 version must support the "pc−i440fx−2.8" and "pc−q35−2.8" machines too. To allow users live migrating VMs to skip multiple intermediate releases when upgrading, new releases of QEMU will support machine types from many previous versions.

Supported machine properties are:

accel=*accels1*[:*accels2*[:...]]

This is used to enable an accelerator. Depending on the target architecture, kvm, xen, hax, hvf, whpx or tcg can be available. By default, tcg is used. If there is more than one accelerator specified, the next one is used if the previous one fails to initialize.

kernel_irqchip=on|off

Controls in-kernel irqchip support for the chosen accelerator when available.

gfx_passthru=on|off

Enables IGD GFX passthrough support for the chosen machine when available.

vmport=on|off|auto

Enables emulation of VMWare IO port, for vmmouse etc. auto says to select the value based on accel. For accel=xen the default is off otherwise the default is on.

kvm_shadow_mem=size

Defines the size of the KVM shadow MMU.

dump−guest−core=on|off

Include guest memory in a core dump. The default is on.

mem−merge=on|off

Enables or disables memory merge support. This feature, when supported by the host, deduplicates identical memory pages among VMs instances (enabled by default).

aes−key−wrap=on|off

Enables or disables AES key wrapping support on s390−ccw hosts. This feature controls whether AES wrapping keys will be created to allow execution of AES cryptographic functions. The default is on.

dea−key−wrap=on|off

Enables or disables DEA key wrapping support on s390−ccw hosts. This feature controls whether DEA wrapping keys will be created to allow execution of DEA cryptographic functions. The default is on.

nvdimm=on|off

Enables or disables NVDIMM support. The default is off.

enforce−config−section=on|off

If **enforce-config-section** is set to *on*, force migration code to send configuration section even if the machine-type sets the **migration.send−configuration** property to *off*. NOTE: this parameter is deprecated. Please use **−global migration.send−configuration**=*on*|*off* instead.

memory−encryption=

Memory encryption object to use. The default is none.

−cpu *model*

Select CPU model (`-cpu help` for list and additional feature selection)

–accel *name***[,prop=***value***[,...]]**

This is used to enable an accelerator. Depending on the target architecture, kvm, xen, hax, hvf, whpx or tcg can be available. By default, tcg is used. If there is more than one accelerator specified, the next one is used if the previous one fails to initialize.

thread=single|multi

Controls number of TCG threads. When the TCG is multi-threaded there will be one thread per vCPU therefor taking advantage of additional host cores. The default is to enable multi-threading where both the back-end and front-ends support it and no incompatible TCG features have been enabled (e.g. icount/replay).

–smp [cpus=]*n***[,cores=***cores***][,threads=***threads***][,dies=dies][,sockets=***sockets***][,maxcpus=***maxcpus***]**

Simulate an SMP system with *n* CPUs. On the PC target, up to 255 CPUs are supported. On Sparc32 target, Linux limits the number of usable CPUs to 4. For the PC target, the number of *cores* per die, the number of *threads* per cores, the number of *dies* per packages and the total number of *sockets* can be specified. Missing values will be computed. If any on the three values is given, the total number of CPUs *n* can be omitted. *maxcpus* specifies the maximum number of hotpluggable CPUs.

–numa node[,mem=*size***][,cpus=***firstcpu***[–***lastcpu***]][,nodeid=***node***]**
–numa node[,memdev=*id***][,cpus=***firstcpu***[–***lastcpu***]][,nodeid=***node***]**
–numa dist,src=*source***,dst=***destination***,val=***distance*
–numa cpu,node–id=*node***[,socket–id=***x***][,core–id=***y***][,thread–id=***z***]**

Define a NUMA node and assign RAM and VCPUs to it. Set the NUMA distance from a source node to a destination node.

Legacy VCPU assignment uses **cpus** option where *firstcpu* and *lastcpu* are CPU indexes. Each **cpus** option represent a contiguous range of CPU indexes (or a single VCPU if *lastcpu* is omitted). A non-contiguous set of VCPUs can be represented by providing multiple **cpus** options. If **cpus** is omitted on all nodes, VCPUs are automatically split between them.

For example, the following option assigns VCPUs 0, 1, 2 and 5 to a NUMA node:

```
        –numa node,cpus=0–2,cpus=5
```

cpu option is a new alternative to **cpus** option which uses **socket–id|core–id|thread–id** properties to assign CPU objects to a *node* using topology layout properties of CPU. The set of properties is machine specific, and depends on used machine type/**smp** options. It could be queried with **hotpluggable-cpus** monitor command. **node-id** property specifies *node* to which CPU object will be assigned, it's required for *node* to be declared with **node** option before it's used with **cpu** option.

For example:

```
        –M pc \
        –smp 1,sockets=2,maxcpus=2 \
        –numa node,nodeid=0 –numa node,nodeid=1 \
        –numa cpu,node–id=0,socket–id=0 –numa cpu,node–id=1,socket–id=1
```

mem assigns a given RAM amount to a node. **memdev** assigns RAM from a given memory backend device to a node. If **mem** and **memdev** are omitted in all nodes, RAM is split equally between them.

mem and **memdev** are mutually exclusive. Furthermore, if one node uses **memdev**, all of them have to use it.

source and *destination* are NUMA node IDs. *distance* is the NUMA distance from *source* to *destination*. The distance from a node to itself is always 10. If any pair of nodes is given a distance, then all pairs must be given distances. Although, when distances are only given in one direction for each pair of nodes, then the distances in the opposite directions are assumed to be the same. If, however, an asymmetrical pair of distances is given for even one node pair, then all node pairs must be provided distance values for both directions, even when they are symmetrical. When a node is unreachable from another node, set the pair's distance to 255.

Note that the **−numa** option doesn't allocate any of the specified resources, it just assigns existing resources to NUMA nodes. This means that one still has to use the **−m**, **−smp** options to allocate RAM and VCPUs respectively.

−add−fd fd=*fd***,set=***set***[,opaque=***opaque***]**

Add a file descriptor to an fd set. Valid options are:

fd=*fd*

This option defines the file descriptor of which a duplicate is added to fd set. The file descriptor cannot be stdin, stdout, or stderr.

set=*set*

This option defines the ID of the fd set to add the file descriptor to.

opaque=*opaque*

This option defines a free-form string that can be used to describe *fd*.

You can open an image using pre-opened file descriptors from an fd set:

```
qemu-system-x86_64 \
-add-fd fd=3,set=2,opaque="rdwr:/path/to/file" \
-add-fd fd=4,set=2,opaque="rdonly:/path/to/file" \
-drive file=/dev/fdset/2,index=0,media=disk
```

−set *group***.***id***.***arg***=***value*

Set parameter *arg* for item *id* of type *group*

−global *driver***.***prop***=***value*
−global driver=*driver***,property=***property***,value=***value*

Set default value of *driver*'s property *prop* to *value*, e.g.:

```
qemu-system-x86_64 -global ide-hd.physical_block_size=4096 disk-image.img
```

In particular, you can use this to set driver properties for devices which are created automatically by the machine model. To create a device which is not created automatically and set properties on it, use **−device**.

−global *driver.prop*=*value* is shorthand for −global driver=*driver*,property=*prop*,value=*value*. The longhand syntax works even when *driver* contains a dot.

−boot
[order=*drives***][,once=***drives***][,menu=on|off][,splash=***sp_name***][,splash−time=***sp_time***][,reboot−timeout=***rb_timeout***][,stric**

Specify boot order *drives* as a string of drive letters. Valid drive letters depend on the target architecture. The x86 PC uses: a, b (floppy 1 and 2), c (first hard disk), d (first CD-ROM), n−p (Etherboot from network adapter 1−4), hard disk boot is the default. To apply a particular boot order only on the first startup, specify it via **once**. Note that the **order** or **once** parameter should not be used together with the **bootindex** property of devices, since the firmware implementations normally do not support both at the same time.

Interactive boot menus/prompts can be enabled via **menu=on** as far as firmware/BIOS supports them. The default is non-interactive boot.

A splash picture could be passed to bios, enabling user to show it as logo, when option splash=*sp_name* is given and menu=on, If firmware/BIOS supports them. Currently Seabios for X86 system support it. limitation: The splash file could be a jpeg file or a BMP file in 24 BPP format(true color). The resolution should be supported by the SVGA mode, so the recommended is 320x240, 640x480, 800x640.

A timeout could be passed to bios, guest will pause for *rb_timeout* ms when boot failed, then reboot. If *rb_timeout* is '−1', guest will not reboot, qemu passes '−1' to bios by default. Currently Seabios for X86 system support it.

Do strict boot via **strict=on** as far as firmware/BIOS supports it. This only effects when boot priority is changed by bootindex options. The default is non-strict boot.

```
# try to boot from network first, then from hard disk
qemu-system-x86_64 -boot order=nc
# boot from CD-ROM first, switch back to default order after reboot
qemu-system-x86_64 -boot once=d
# boot with a splash picture for 5 seconds.
qemu-system-x86_64 -boot menu=on,splash=/root/boot.bmp,splash-time=5000
```

Note: The legacy format '−boot *drives*' is still supported but its use is discouraged as it may be removed from future versions.

−m [size=]*megs***[,slots=n,maxmem=size]**

Sets guest startup RAM size to *megs* megabytes. Default is 128 MiB. Optionally, a suffix of "M" or "G" can be used to signify a value in megabytes or gigabytes respectively. Optional pair *slots*, *maxmem* could be used to set amount of hotpluggable memory slots and maximum amount of memory. Note that *maxmem* must be aligned to the page size.

For example, the following command-line sets the guest startup RAM size to 1GB, creates 3 slots to hotplug additional memory and sets the maximum memory the guest can reach to 4GB:

```
qemu-system-x86_64 -m 1G,slots=3,maxmem=4G
```

If *slots* and *maxmem* are not specified, memory hotplug won't be enabled and the guest startup RAM will never increase.

−mem−path *path*

Allocate guest RAM from a temporarily created file in *path*.

−mem−prealloc

Preallocate memory when using −mem−path.

−k *language*

Use keyboard layout *language* (for example `fr` for French). This option is only needed where it is not easy to get raw PC keycodes (e.g. on Macs, with some X11 servers or with a VNC or curses display). You don't normally need to use it on PC/Linux or PC/Windows hosts.

The available layouts are:

```
ar  de-ch  es  fo     fr-ca  hu  ja  mk     no  pt-br  sv
da  en-gb  et  fr     fr-ch  is  lt  nl     pl  ru     th
de  en-us  fi  fr-be  hr     it  lv  nl-be  pt  sl     tr
```

The default is `en-us`.

−audio−help

Will show the −audiodev equivalent of the currently specified (deprecated) environment variables.

−audiodev [driver=]*driver***,id=***id***[,***prop***[=***value***][,...]]**

Adds a new audio backend *driver* identified by *id*. There are global and driver specific properties. Some values can be set differently for input and output, they're marked with `in|out.`. You can set the input's property with `in.`*`prop`* and the output's property with `out.`*`prop`*. For example:

```
-audiodev alsa,id=example,in.frequency=44110,out.frequency=8000
-audiodev alsa,id=example,out.channels=1 # leaves in.channels unspecified
```

NOTE: parameter validation is known to be incomplete, in many cases specifying an invalid option causes QEMU to print an error message and continue emulation without sound.

Valid global options are:

id=*identifier*

Identifies the audio backend.

timer−period=*period*

Sets the timer *period* used by the audio subsystem in microseconds. Default is 10000 (10 ms).

in|out.mixing−engine=on|off

Use QEMU's mixing engine to mix all streams inside QEMU and convert audio formats when not supported by the backend. When off, *fixed-settings* must be off too. Note that disabling this option means that the selected backend must support multiple streams and the audio formats used by the virtual cards, otherwise you'll get no sound. It's not recommended to disable this option unless you want to use 5.1 or 7.1 audio, as mixing engine only supports mono and stereo audio. Default is on.

in|out.fixed−settings=on|off

Use fixed settings for host audio. When off, it will change based on how the guest opens the sound card. In this case you must not specify *frequency*, *channels* or *format*. Default is on.

in|out.frequency=*frequency*

Specify the *frequency* to use when using *fixed-settings*. Default is 44100Hz.

in|out.channels=*channels*

Specify the number of *channels* to use when using *fixed-settings*. Default is 2 (stereo).

in|out.format=*format*

Specify the sample *format* to use when using *fixed-settings*. Valid values are: `s8`, `s16`, `s32`, `u8`, `u16`, `u32`. Default is `s16`.

in|out.voices=*voices*

Specify the number of *voices* to use. Default is 1.

in|out.buffer−length=*usecs*

Sets the size of the buffer in microseconds.

−audiodev none,id=*id***[,***prop***[=***value***][,...]]**

Creates a dummy backend that discards all outputs. This backend has no backend specific properties.

−audiodev alsa,id=*id***[,***prop***[=***value***][,...]]**

Creates backend using the ALSA. This backend is only available on Linux.

ALSA specific options are:

in|out.dev=*device*

Specify the ALSA *device* to use for input and/or output. Default is `default`.

in|out.period−length=*usecs*

Sets the period length in microseconds.

in|out.try−poll=on|off

Attempt to use poll mode with the device. Default is on.

threshold=*threshold*

Threshold (in microseconds) when playback starts. Default is 0.

−audiodev coreaudio,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend using Apple's Core Audio. This backend is only available on Mac OS and only supports playback.

Core Audio specific options are:

in|out.buffer−count=*count*

Sets the *count* of the buffers.

−audiodev dsound,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend using Microsoft's DirectSound. This backend is only available on Windows and only supports playback.

DirectSound specific options are:

latency=*usecs*

Add extra *usecs* microseconds latency to playback. Default is 10000 (10 ms).

−audiodev oss,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend using OSS. This backend is available on most Unix-like systems.

OSS specific options are:

in|out.dev=*device*

Specify the file name of the OSS *device* to use. Default is `/dev/dsp`.

in|out.buffer−count=*count*

Sets the *count* of the buffers.

in|out.try−poll=on|of

Attempt to use poll mode with the device. Default is on.

try−mmap=on|off

Try using memory mapped device access. Default is off.

exclusive=on|off

Open the device in exclusive mode (vmix won't work in this case). Default is off.

dsp−policy=*policy*

Sets the timing policy (between 0 and 10, where smaller number means smaller latency but higher CPU usage). Use −1 to use buffer sizes specified by `buffer` and `buffer−count`. This option is ignored if you do not have OSS 4. Default is 5.

−audiodev pa,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend using PulseAudio. This backend is available on most systems.

PulseAudio specific options are:

server=*server*

Sets the PulseAudio *server* to connect to.

in|out.name=*sink*

Use the specified source/sink for recording/playback.

in|out.latency=*usecs*

Desired latency in microseconds. The PulseAudio server will try to honor this value but actual latencies may be lower or higher.

−audiodev sdl,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend using SDL. This backend is available on most systems, but you should use your platform's native backend if possible. This backend has no backend specific properties.

−audiodev spice,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend that sends audio through SPICE. This backend requires `−spice` and automatically selected in that case, so usually you can ignore this option. This backend has no backend specific properties.

−audiodev wav,id=*id***[,***prop***[=***value***][,...]]**

Creates a backend that writes audio to a WAV file.

Backend specific options are:

path=*path*

Write recorded audio into the specified file. Default is `qemu.wav`.

−soundhw *card1***[,***card2***,...] or −soundhw all**

Enable audio and selected sound hardware. Use 'help' to print all available sound hardware. For example:

```
qemu-system-x86_64 -soundhw sb16,adlib disk.img
qemu-system-x86_64 -soundhw es1370 disk.img
qemu-system-x86_64 -soundhw ac97 disk.img
qemu-system-x86_64 -soundhw hda disk.img
qemu-system-x86_64 -soundhw all disk.img
qemu-system-x86_64 -soundhw help
```

Note that Linux's i810_audio OSS kernel (for AC97) module might require manually specifying clocking.

```
modprobe i810_audio clocking=48000
```

−device *driver***[,***prop***[=***value***][,...]]**

Add device *driver*. *prop*=*value* sets driver properties. Valid properties depend on the driver. To get help on possible drivers and properties, use `-device help` and `-device driver,help`.

Some drivers are:

−device ipmi−bmc−sim,id=*id***[,slave_addr=***val***][,sdrfile=***file***][,furareasize=***val***][,furdatafile=***file***][,guid=***uuid***]**

Add an IPMI BMC. This is a simulation of a hardware management interface processor that normally sits on a system. It provides a watchdog and the ability to reset and power control the system. You need to connect this to an IPMI interface to make it useful

The IPMI slave address to use for the BMC. The default is 0x20. This address is the BMC's address on the I2C network of management controllers. If you don't know what this means, it is safe to ignore it.

id=*id*

The BMC id for interfaces to use this device.

slave_addr=*val*

Define slave address to use for the BMC. The default is 0x20.

sdrfile=*file*

file containing raw Sensor Data Records (SDR) data. The default is none.

fruareasize=*val*

size of a Field Replaceable Unit (FRU) area. The default is 1024.

frudatafile=*file*

file containing raw Field Replaceable Unit (FRU) inventory data. The default is none.

guid=*uuid*

value for the GUID for the BMC, in standard UUID format. If this is set, get "Get GUID" command to the BMC will return it. Otherwise "Get GUID" will return an error.

−device ipmi−bmc−extern,id=*id***,chardev=***id***[,slave_addr=***val***]**

Add a connection to an external IPMI BMC simulator. Instead of locally emulating the BMC like the above item, instead connect to an external entity that provides the IPMI services.

A connection is made to an external BMC simulator. If you do this, it is strongly recommended that you use the "reconnect=" chardev option to reconnect to the simulator if the connection is lost. Note that if this is not used carefully, it can be a security issue, as the interface has the ability to send resets, NMIs, and power off the VM. It's best if QEMU makes a connection to an external simulator running on a secure port on localhost, so neither the simulator nor QEMU is exposed to any outside network.

See the "lanserv/README.vm" file in the OpenIPMI library for more details on the external interface.

−device isa−ipmi−kcs,bmc=*id***[,ioport=***val***][,irq=***val***]**

Add a KCS IPMI interafce on the ISA bus. This also adds a corresponding ACPI and SMBIOS entries, if appropriate.

bmc=*id*

The BMC to connect to, one of ipmi-bmc-sim or ipmi-bmc-extern above.

ioport=*val*

Define the I/O address of the interface. The default is 0xca0 for KCS.

irq=*val*

Define the interrupt to use. The default is 5. To disable interrupts, set this to 0.

−device isa−ipmi−bt,bmc=*id***[,ioport=***val***][,irq=***val***]**

Like the KCS interface, but defines a BT interface. The default port is 0xe4 and the default interrupt is 5.

−name *name*

Sets the *name* of the guest. This name will be displayed in the SDL window caption. The *name* will also be used for the VNC server. Also optionally set the top visible process name in Linux. Naming of individual threads can also be enabled on Linux to aid debugging.

−uuid *uuid*

Set system UUID.

Block device options

−fda *file*
−fdb *file*

Use *file* as floppy disk 0/1 image.

−hda *file*
−hdb *file*
−hdc *file*
−hdd *file*

Use *file* as hard disk 0, 1, 2 or 3 image.

−cdrom *file*

Use *file* as CD-ROM image (you cannot use **−hdc** and **−cdrom** at the same time). You can use the host CD-ROM by using */dev/cdrom* as filename.

−blockdev *option***[,***option***[,***option***[,...]]]**

Define a new block driver node. Some of the options apply to all block drivers, other options are only accepted for a specific block driver. See below for a list of generic options and options for the most common block drivers.

Options that expect a reference to another node (e.g. `file`) can be given in two ways. Either you specify the node name of an already existing node (file=*node-name*), or you define a new node inline, adding options for the referenced node after a dot (file.filename=*path*,file.aio=native).

A block driver node created with **−blockdev** can be used for a guest device by specifying its node name for the `drive` property in a **−device** argument that defines a block device.

Valid options for any block driver node:

`driver`

Specifies the block driver to use for the given node.

`node-name`

This defines the name of the block driver node by which it will be referenced later. The name must be unique, i.e. it must not match the name of a different block driver node, or (if you use **−drive** as well) the ID of a drive.

If no node name is specified, it is automatically generated. The generated node name is not intended to be predictable and changes between QEMU invocations. For the top level, an explicit node name must be specified.

`read-only`

Open the node read-only. Guest write attempts will fail.

Note that some block drivers support only read-only access, either generally or in certain configurations. In this case, the default value **read−only=off** does not work and the option must be specified explicitly.

`auto-read-only`

If **auto−read−only=on** is set, QEMU may fall back to read-only usage even when **read−only=off** is requested, or even switch between modes as needed, e.g. depending on whether the image file is writable or whether a writing user is attached to the node.

`force-share`

Override the image locking system of QEMU by forcing the node to utilize weaker shared access for permissions where it would normally request exclusive access. When there is the potential for multiple instances to have the same file open (whether this invocation of QEMU is the first or the second instance), both instances must permit shared access for the second instance to succeed at opening the file.

Enabling **force−share=on** requires **read−only=on**.

`cache.direct`

The host page cache can be avoided with **cache.direct=on**. This will attempt to do disk IO directly to the guest's memory. QEMU may still perform an internal copy of the data.

`cache.no-flush`

In case you don't care about data integrity over host failures, you can use **cache.no−flush=on**. This option tells QEMU that it never needs to write any data to the disk but can instead keep things in cache. If anything goes wrong, like your host losing power, the disk storage getting disconnected accidentally, etc. your image will most probably be rendered unusable.

`discard=`*`discard`*

discard is one of ''ignore'' (or ''off'') or ''unmap'' (or ''on'') and controls whether `discard` (also known as `trim` or `unmap`) requests are ignored or passed to the filesystem. Some machine types may not support discard requests.

`detect-zeroes=`*`detect-zeroes`*

detect-zeroes is ''off'', ''on'' or ''unmap'' and enables the automatic conversion of plain zero writes by the OS to driver specific optimized zero write commands. You may even choose ''unmap'' if *discard* is set to ''unmap'' to allow a zero write to be converted to an `unmap` operation.

Driver-specific options for `file`

This is the protocol-level block driver for accessing regular files.

`filename`

The path to the image file in the local filesystem

`aio`

Specifies the AIO backend (threads/native, default: threads)

`locking`

Specifies whether the image file is protected with Linux OFD / POSIX locks. The default is to use the Linux Open File Descriptor API if available, otherwise no lock is applied. (auto/on/off, default: auto)

Example:

```
        -blockdev driver=file,node-name=disk,filename=disk.img
```

Driver-specific options for `raw`

This is the image format block driver for raw images. It is usually stacked on top of a protocol level block driver such as `file`.

`file`

Reference to or definition of the data source block driver node (e.g. a `file` driver node)

Example 1:

```
-blockdev driver=file,node-name=disk_file,filename=disk.img
-blockdev driver=raw,node-name=disk,file=disk_file
```

Example 2:

```
-blockdev driver=raw,node-name=disk,file.driver=file,file.filename=di
```

Driver-specific options for `qcow2`

This is the image format block driver for qcow2 images. It is usually stacked on top of a protocol level block driver such as `file`.

`file`

Reference to or definition of the data source block driver node (e.g. a `file` driver node)

`backing`

Reference to or definition of the backing file block device (default is taken from the image file). It is allowed to pass `null` here in order to disable the default backing file.

`lazy-refcounts`

Whether to enable the lazy refcounts feature (on/off; default is taken from the image file)

`cache-size`

The maximum total size of the L2 table and refcount block caches in bytes (default: the sum of l2−cache−size and refcount-cache-size)

`l2-cache-size`

The maximum size of the L2 table cache in bytes (default: if cache-size is not specified − 32M on Linux platforms, and 8M on non-Linux platforms; otherwise, as large as possible within the cache-size, while permitting the requested or the minimal refcount cache size)

`refcount-cache-size`

The maximum size of the refcount block cache in bytes (default: 4 times the cluster size; or if cache-size is specified, the part of it which is not used for the L2 cache)

`cache-clean-interval`

Clean unused entries in the L2 and refcount caches. The interval is in seconds. The default value is 600 on supporting platforms, and 0 on other platforms. Setting it to 0 disables this feature.

`pass-discard-request`

Whether discard requests to the qcow2 device should be forwarded to the data source (on/off; default: on if discard=unmap is specified, off otherwise)

`pass-discard-snapshot`

Whether discard requests for the data source should be issued when a snapshot operation (e.g. deleting a snapshot) frees clusters in the qcow2 file (on/off; default: on)

`pass-discard-other`

Whether discard requests for the data source should be issued on other occasions where a cluster gets freed (on/off; default: off)

`overlap-check`

Which overlap checks to perform for writes to the image (none/constant/cached/all; default: cached). For details or finer granularity control refer to the QAPI documentation of `blockdev-add`.

Example 1:

```
-blockdev driver=file,node-name=my_file,filename=/tmp/disk.qcow2
-blockdev driver=qcow2,node-name=hda,file=my_file,overlap-check=none,c
```

Example 2:

```
-blockdev driver=qcow2,node-name=disk,file.driver=http,file.filename=h
```

Driver-specific options for other drivers

Please refer to the QAPI documentation of the `blockdev-add` QMP command.

–drive *option*[,*option*[,*option*[,...]]]

Define a new drive. This includes creating a block driver node (the backend) as well as a guest device, and is mostly a shortcut for defining the corresponding **–blockdev** and **–device** options.

–drive accepts all options that are accepted by **–blockdev**. In addition, it knows the following options:

file=*file*

This option defines which disk image to use with this drive. If the filename contains comma, you must double it (for instance, ‘‘file=my,,file’’ to use file ‘‘my,file’’).

Special files such as iSCSI devices can be specified using protocol specific URLs. See the section for ‘‘Device URL Syntax’’ for more information.

if=*interface*

This option defines on which type on interface the drive is connected. Available types are: ide, scsi, sd, mtd, floppy, pflash, virtio, none.

bus=*bus***,unit=***unit*

These options define where is connected the drive by defining the bus number and the unit id.

index=*index*

This option defines where is connected the drive by using an index in the list of available connectors of a given interface type.

media=*media*

This option defines the type of the media: disk or cdrom.

snapshot=*snapshot*

snapshot is ‘‘on’’ or ‘‘off’’ and controls snapshot mode for the given drive (see **–snapshot**).

cache=*cache*

cache is ‘‘none’’, ‘‘writeback’’, ‘‘unsafe’’, ‘‘directsync’’ or ‘‘writethrough’’ and controls how the host cache is used to access block data. This is a shortcut that sets the **cache.direct** and **cache.no–flush** options (as in **–blockdev**), and additionally **cache.writeback**, which provides a default for the **write-cache** option of block guest devices (as in **–device**). The modes correspond to the following settings:

	cache.writeback	cache.direct	cache.no-flush
writeback	on	off	off
none	on	on	off
writethrough	off	off	off
directsync	off	on	off
unsafe	on	off	on

The default mode is **cache=writeback**.

aio=*aio*

aio is ‘‘threads’’, or ‘‘native’’ and selects between pthread based disk I/O and native Linux AIO.

format=*format*

Specify which disk *format* will be used rather than detecting the format. Can be used to specify format=raw to avoid interpreting an untrusted format header.

werror=*action***,rerror=***action*

Specify which *action* to take on write and read errors. Valid actions are: ''ignore'' (ignore the error and try to continue), ''stop'' (pause QEMU), ''report'' (report the error to the guest), ''enospc'' (pause QEMU only if the host disk is full; report the error to the guest otherwise). The default setting is **werror=enospc** and **rerror=report**.

copy−on−read=*copy-on-read*

copy-on-read is ''on'' or ''off'' and enables whether to copy read backing file sectors into the image file.

bps=*b***,bps_rd=***r***,bps_wr=***w*

Specify bandwidth throttling limits in bytes per second, either for all request types or for reads or writes only. Small values can lead to timeouts or hangs inside the guest. A safe minimum for disks is 2 MB/s.

bps_max=*bm***,bps_rd_max=***rm***,bps_wr_max=***wm*

Specify bursts in bytes per second, either for all request types or for reads or writes only. Bursts allow the guest I/O to spike above the limit temporarily.

iops=*i***,iops_rd=***r***,iops_wr=***w*

Specify request rate limits in requests per second, either for all request types or for reads or writes only.

iops_max=*bm***,iops_rd_max=***rm***,iops_wr_max=***wm*

Specify bursts in requests per second, either for all request types or for reads or writes only. Bursts allow the guest I/O to spike above the limit temporarily.

iops_size=*is*

Let every *is* bytes of a request count as a new request for iops throttling purposes. Use this option to prevent guests from circumventing iops limits by sending fewer but larger requests.

group=*g*

Join a throttling quota group with given name *g*. All drives that are members of the same group are accounted for together. Use this option to prevent guests from circumventing throttling limits by using many small disks instead of a single larger disk.

By default, the **cache.writeback=on** mode is used. It will report data writes as completed as soon as the data is present in the host page cache. This is safe as long as your guest OS makes sure to correctly flush disk caches where needed. If your guest OS does not handle volatile disk write caches correctly and your host crashes or loses power, then the guest may experience data corruption.

For such guests, you should consider using **cache.writeback=off**. This means that the host page cache will be used to read and write data, but write notification will be sent to the guest only after QEMU has made sure to flush each write to the disk. Be aware that this has a major impact on performance.

When using the **−snapshot** option, unsafe caching is always used.

Copy-on-read avoids accessing the same backing file sectors repeatedly and is useful when the backing file is over a slow network. By default copy-on-read is off.

Instead of **−cdrom** you can use:

```
qemu−system−x86_64 −drive file=file,index=2,media=cdrom
```

Instead of **−hda**, **−hdb**, **−hdc**, **−hdd**, you can use:

```
qemu−system−x86_64 −drive file=file,index=0,media=disk
qemu−system−x86_64 −drive file=file,index=1,media=disk
qemu−system−x86_64 −drive file=file,index=2,media=disk
qemu−system−x86_64 −drive file=file,index=3,media=disk
```

You can open an image using pre-opened file descriptors from an fd set:

```
qemu-system-x86_64 \
-add-fd fd=3,set=2,opaque="rdwr:/path/to/file" \
-add-fd fd=4,set=2,opaque="rdonly:/path/to/file" \
-drive file=/dev/fdset/2,index=0,media=disk
```

You can connect a CDROM to the slave of ide0:

```
qemu-system-x86_64 -drive file=file,if=ide,index=1,media=cdrom
```

If you don't specify the ''file='' argument, you define an empty drive:

```
qemu-system-x86_64 -drive if=ide,index=1,media=cdrom
```

Instead of **−fda**, **−fdb**, you can use:

```
qemu-system-x86_64 -drive file=file,index=0,if=floppy
qemu-system-x86_64 -drive file=file,index=1,if=floppy
```

By default, *interface* is ''ide'' and *index* is automatically incremented:

```
qemu-system-x86_64 -drive file=a -drive file=b"
```

is interpreted like:

```
qemu-system-x86_64 -hda a -hdb b
```

−mtdblock *file*

Use *file* as on-board Flash memory image.

−sd *file*

Use *file* as SecureDigital card image.

−pflash *file*

Use *file* as a parallel flash image.

−snapshot

Write to temporary files instead of disk image files. In this case, the raw disk image you use is not written back. You can however force the write back by pressing **C−a s**.

−fsdev local,id=*id***,path=***path***,security_model=***security_model* **[,writeout=***writeout***][,readonly][,fmode=***fmode***][,dmode=***dmode***]**
[,throttling.*option***=***value***[,throttling.***option***=***value***[,...]]]**
−fsdev proxy,id=*id***,socket=***socket***[,writeout=***writeout***][,readonly]**
−fsdev proxy,id=*id***,sock_fd=***sock_fd***[,writeout=***writeout***][,readonly]**
−fsdev synth,id=*id***[,readonly]**

Define a new file system device. Valid options are:

local

Accesses to the filesystem are done by QEMU.

proxy

Accesses to the filesystem are done by **virtfs−proxy−helper** (1).

synth

Synthetic filesystem, only used by QTests.

id=*id*

Specifies identifier for this device.

path=*path*

Specifies the export path for the file system device. Files under this path will be available to the 9p client on the guest.

security_model=*security_model*

Specifies the security model to be used for this export path. Supported security models are ''passthrough'', ''mapped-xattr'', ''mapped-file'' and ''none''. In ''passthrough'' security model, files are stored using the same credentials as they are created on the guest. This requires QEMU to run as root. In ‘‘mapped-xattr’’ security model, some of the file attributes like uid, gid, mode bits and link target are stored as file attributes. For ‘‘mapped-file’’ these attributes are stored in the hidden .virtfs_metadata directory. Directories exported by this security model cannot interact with other unix tools. ‘‘none’’ security model is same as passthrough except the sever won’t report failures if it fails to set file attributes like ownership. Security model is mandatory only for local fsdriver. Other fsdrivers (like proxy) don’t take security model as a parameter.

writeout=*writeout*

This is an optional argument. The only supported value is ‘‘immediate’’. This means that host page cache will be used to read and write data but write notification will be sent to the guest only when the data has been reported as written by the storage subsystem.

readonly

Enables exporting 9p share as a readonly mount for guests. By default read-write access is given.

socket=*socket*

Enables proxy filesystem driver to use passed socket file for communicating with **virtfs−proxy−helper** (1).

sock_fd=*sock_fd*

Enables proxy filesystem driver to use passed socket descriptor for communicating with **virtfs−proxy−helper** (1). Usually a helper like libvirt will create socketpair and pass one of the fds as sock_fd.

fmode=*fmode*

Specifies the default mode for newly created files on the host. Works only with security models ‘‘mapped-xattr’’ and ‘‘mapped-file’’.

dmode=*dmode*

Specifies the default mode for newly created directories on the host. Works only with security models ‘‘mapped-xattr’’ and ‘‘mapped-file’’.

throttling.bps−total=*b***,throttling.bps−read=***r***,throttling.bps−write=***w*

Specify bandwidth throttling limits in bytes per second, either for all request types or for reads or writes only.

throttling.bps−total−max=*bm***,bps−read−max=***rm***,bps−write−max=***wm*

Specify bursts in bytes per second, either for all request types or for reads or writes only. Bursts allow the guest I/O to spike above the limit temporarily.

throttling.iops−total=*i***,throttling.iops−read=***r***, throttling.iops−write=***w*

Specify request rate limits in requests per second, either for all request types or for reads or writes only.

throttling.iops−total−max=*im***,throttling.iops−read−max=***irm***, throttling.iops−write−max=***iwm*

Specify bursts in requests per second, either for all request types or for reads or writes only. Bursts allow the guest I/O to spike above the limit temporarily.

throttling.iops−size=*is*

Let every *is* bytes of a request count as a new request for iops throttling purposes.

−fsdev option is used along with −device driver ‘‘virtio−9p−...’’.

−device virtio−9p−*type***,fsdev=***id***,mount_tag=***mount_tag*

Options for virtio−9p−... driver are:

type

Specifies the variant to be used. Supported values are ‘‘pci’’, ‘‘ccw’’ or ‘‘device’’, depending on the machine type.

fsdev=*id*

Specifies the id value specified along with −fsdev option.

mount_tag=*mount_tag*

Specifies the tag name to be used by the guest to mount this export point.

−virtfs local,path=*path***,mount_tag=***mount_tag* **,security_model=***security_model***[,writeout=***writeout***][,readonly] [,fmode=***fmode***][,dmode=***dmode***][,multidevs=***multidevs***]**
−virtfs proxy,socket=*socket***,mount_tag=***mount_tag* **[,writeout=***writeout***][,readonly]**
−virtfs proxy,sock_fd=*sock_fd***,mount_tag=***mount_tag* **[,writeout=***writeout***][,readonly]**
−virtfs synth,mount_tag=*mount_tag*

Define a new filesystem device and expose it to the guest using a virtio−9p−device. The general form of a Virtual File system pass-through options are:

local

Accesses to the filesystem are done by QEMU.

proxy

Accesses to the filesystem are done by **virtfs−proxy−helper** (1).

synth

Synthetic filesystem, only used by QTests.

id=*id*

Specifies identifier for the filesystem device

path=*path*

Specifies the export path for the file system device. Files under this path will be available to the 9p client on the guest.

security_model=*security_model*

Specifies the security model to be used for this export path. Supported security models are ''passthrough'', ''mapped-xattr'', ''mapped-file'' and ''none''. In ''passthrough'' security model, files are stored using the same credentials as they are created on the guest. This requires QEMU to run as root. In ''mapped-xattr'' security model, some of the file attributes like uid, gid, mode bits and link target are stored as file attributes. For ''mapped-file'' these attributes are stored in the hidden .virtfs_metadata directory. Directories exported by this security model cannot interact with other unix tools. ''none'' security model is same as passthrough except the sever won't report failures if it fails to set file attributes like ownership. Security model is mandatory only for local fsdriver. Other fsdrivers (like proxy) don't take security model as a parameter.

writeout=*writeout*

This is an optional argument. The only supported value is ''immediate''. This means that host page cache will be used to read and write data but write notification will be sent to the guest only when the data has been reported as written by the storage subsystem.

readonly

Enables exporting 9p share as a readonly mount for guests. By default read-write access is given.

socket=*socket*

Enables proxy filesystem driver to use passed socket file for communicating with **virtfs−proxy−helper** (1). Usually a helper like libvirt will create socketpair and pass one of the fds as sock_fd.

sock_fd

Enables proxy filesystem driver to use passed 'sock_fd' as the socket descriptor for interfacing with **virtfs−proxy−helper** (1).

fmode=*fmode*

Specifies the default mode for newly created files on the host. Works only with security models ''mapped-xattr'' and ''mapped-file''.

dmode=*dmode*

Specifies the default mode for newly created directories on the host. Works only with security models ''mapped-xattr'' and ''mapped-file''.

mount_tag=*mount_tag*

Specifies the tag name to be used by the guest to mount this export point.

multidevs=*multidevs*

Specifies how to deal with multiple devices being shared with a 9p export. Supported behaviours are either ''remap'', ''forbid'' or ''warn''. The latter is the default behaviour on which virtfs 9p expects only one device to be shared with the same export, and if more than one device is shared and accessed via the same 9p export then only a warning message is logged (once) by qemu on host side. In order to avoid file ID collisions on guest you should either create a separate virtfs export for each device to be shared with guests (recommended way) or you might use ''remap'' instead which allows you to share multiple devices with only one export instead, which is achieved by remapping the original inode numbers from host to guest in a way that would prevent such collisions. Remapping inodes in such use cases is required because the original device IDs from host are never passed and exposed on guest. Instead all files of an export shared with virtfs always share the same device id on guest. So two files with identical inode numbers but from actually different devices on host would otherwise cause a file ID collision and hence potential misbehaviours on guest. ''forbid'' on the other hand assumes like ''warn'' that only one device is shared by the same export, however it will not only log a warning message but also deny access to additional devices on guest. Note though that ''forbid'' does currently not block all possible file access operations (e.g. **readdir()** would still return entries from other devices).

−virtfs_synth

Create synthetic file system image. Note that this option is now deprecated. Please use `-fsdev synth` and `-device virtio-9p-...` instead.

−iscsi

Configure iSCSI session parameters.

USB options

−usb

Enable USB emulation on machine types with an on-board USB host controller (if not enabled by default). Note that on-board USB host controllers may not support USB 3.0. In this case **−device qemu-xhci** can be used instead on machines with PCI.

−usbdevice *devname*

Add the USB device *devname*. Note that this option is deprecated, please use `-device usb-...` instead.

mouse

Virtual Mouse. This will override the PS/2 mouse emulation when activated.

tablet

Pointer device that uses absolute coordinates (like a touchscreen). This means QEMU is able to report the mouse position without having to grab the mouse. Also overrides the PS/2 mouse emulation when activated.

braille

Braille device. This will use BrlAPI to display the braille output on a real or fake device.

Display options

−display *type*

Select type of display to use. This option is a replacement for the old style −sdl/−curses/... options. Valid values for *type* are

sdl Display video output via SDL (usually in a separate graphics window; see the SDL documentation for other possibilities).

curses

Display video output via curses. For graphics device models which support a text mode, QEMU can display this output using a curses/ncurses interface. Nothing is displayed when the graphics device is in graphical mode or if the graphics device does not support a text mode. Generally only the VGA device models support text mode. The font charset used by the guest can be specified with the `charset` option, for example `charset=CP850` for IBM CP850 encoding. The default is `CP437`.

none

Do not display video output. The guest will still see an emulated graphics card, but its output will not be displayed to the QEMU user. This option differs from the −nographic option in that it only affects what is done with video output; −nographic also changes the destination of the serial and parallel port data.

gtk Display video output in a GTK window. This interface provides drop-down menus and other UI elements to configure and control the VM during runtime.

vnc Start a VNC server on display <arg>

egl-headless

Offload all OpenGL operations to a local DRI device. For any graphical display, this display needs to be paired with either VNC or SPICE displays.

spice-app

Start QEMU as a Spice server and launch the default Spice client application. The Spice server will redirect the serial consoles and QEMU monitors. (Since 4.0)

−nographic

Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can totally disable graphical output so that QEMU is a simple command line application. The emulated serial port is redirected on the console and muxed with the monitor (unless redirected elsewhere explicitly). Therefore, you can still use QEMU to debug a Linux kernel with a serial console. Use **C−a h** for help on switching between the console and monitor.

−curses

Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, QEMU can display the VGA output when in text mode using a curses/ncurses interface. Nothing is displayed in graphical mode.

−alt−grab

Use Ctrl-Alt-Shift to grab mouse (instead of Ctrl-Alt). Note that this also affects the special keys (for fullscreen, monitor-mode switching, etc).

−ctrl−grab

Use Right-Ctrl to grab mouse (instead of Ctrl-Alt). Note that this also affects the special keys (for fullscreen, monitor-mode switching, etc).

−no−quit

Disable SDL window close capability.

−sdl

Enable SDL.

−spice *option***[,***option***[,...]]**

Enable the spice remote desktop protocol. Valid options are

port=<nr>

Set the TCP port spice is listening on for plaintext channels.

addr=<addr>
Set the IP address spice is listening on. Default is any address.

ipv4
ipv6
unix
Force using the specified IP version.

password=<secret>
Set the password you need to authenticate.

sasl
Require that the client use SASL to authenticate with the spice. The exact choice of authentication method used is controlled from the system / user's SASL configuration file for the 'qemu' service. This is typically found in /etc/sasl2/qemu.conf. If running QEMU as an unprivileged user, an environment variable SASL_CONF_PATH can be used to make it search alternate locations for the service config. While some SASL auth methods can also provide data encryption (eg GSSAPI), it is recommended that SASL always be combined with the 'tls' and 'x509' settings to enable use of SSL and server certificates. This ensures a data encryption preventing compromise of authentication credentials.

disable-ticketing
Allow client connects without authentication.

disable-copy-paste
Disable copy paste between the client and the guest.

disable-agent-file-xfer
Disable spice-vdagent based file-xfer between the client and the guest.

tls−port=<nr>
Set the TCP port spice is listening on for encrypted channels.

x509−dir=<dir>
Set the x509 file directory. Expects same filenames as −vnc `$display`,x509=$dir

x509−key−file=<file>
x509−key−password=<file>
x509−cert−file=<file>
x509−cacert−file=<file>
x509−dh−key−file=<file>
The x509 file names can also be configured individually.

tls−ciphers=<list>
Specify which ciphers to use.

tls−channel=[main|display|cursor|inputs|record|playback]
plaintext−channel=[main|display|cursor|inputs|record|playback]
Force specific channel to be used with or without TLS encryption. The options can be specified multiple times to configure multiple channels. The special name ''default'' can be used to set the default mode. For channels which are not explicitly forced into one mode the spice client is allowed to pick tls/plaintext as he pleases.

image−compression=[auto_glz|auto_lz|quic|glz|lz|off]
Configure image compression (lossless). Default is auto_glz.

jpeg−wan−compression=[auto|never|always]
zlib−glz−wan−compression=[auto|never|always]
Configure wan image compression (lossy for slow links). Default is auto.

streaming−video=[off|all|filter]
Configure video stream detection. Default is off.

agent−mouse=[on|off]
Enable/disable passing mouse events via vdagent. Default is on.

playback−compression=[on|off]
Enable/disable audio stream compression (using celt 0.5.1). Default is on.

seamless−migration=[on|off]
Enable/disable spice seamless migration. Default is off.

gl=[on|off]
Enable/disable OpenGL context. Default is off.

rendernode=<file>
DRM render node for OpenGL rendering. If not specified, it will pick the first available. (Since 2.9)

−portrait
Rotate graphical output 90 deg left (only PXA LCD).

−rotate *deg*
Rotate graphical output some deg left (only PXA LCD).

−vga *type*
Select type of VGA card to emulate. Valid values for *type* are

cirrus
Cirrus Logic GD5446 Video card. All Windows versions starting from Windows 95 should recognize and use this graphic card. For optimal performances, use 16 bit color depth in the guest and the host OS. (This card was the default before QEMU 2.2)

std Standard VGA card with Bochs VBE extensions. If your guest OS supports the VESA 2.0 VBE extensions (e.g. Windows XP) and if you want to use high resolution modes (>= 1280x1024x16) then you should use this option. (This card is the default since QEMU 2.2)

vmware
VMWare SVGA-II compatible adapter. Use it if you have sufficiently recent XFree86/XOrg server or Windows guest with a driver for this card.

qxl QXL paravirtual graphic card. It is VGA compatible (including VESA 2.0 VBE support). Works best with qxl guest drivers installed though. Recommended choice when using the spice protocol.

tcx (sun4m only) Sun TCX framebuffer. This is the default framebuffer for sun4m machines and offers both 8−bit and 24−bit colour depths at a fixed resolution of 1024x768.

cg3 (sun4m only) Sun cgthree framebuffer. This is a simple 8−bit framebuffer for sun4m machines available in both 1024x768 (OpenBIOS) and 1152x900 (OBP) resolutions aimed at people wishing to run older Solaris versions.

virtio
Virtio VGA card.

none
Disable VGA card.

−full−screen
Start in full screen.

−g *width***x***height***[x***depth***]**
Set the initial graphical resolution and depth (PPC, SPARC only).

−vnc *display***[,***option***[,***option***[,...]]]**
Normally, if QEMU is compiled with graphical window support, it displays output such as guest graphics, guest console, and the QEMU monitor in a window. With this option, you can have QEMU listen on VNC display *display* and redirect the VGA display over the VNC session. It is very useful to

enable the usb tablet device when using this option (option **−device usb-tablet**). When using the VNC display, you must use the **−k** parameter to set the keyboard layout if you are not using en-us. Valid syntax for the *display* is

to=*L*

With this option, QEMU will try next available VNC *display*s, until the number *L*, if the origianlly defined "−vnc *display*" is not available, e.g. port 5900+*display* is already used by another application. By default, to=0.

*host***:***d*

TCP connections will only be allowed from *host* on display *d*. By convention the TCP port is 5900+*d*. Optionally, *host* can be omitted in which case the server will accept connections from any host.

unix:*path*

Connections will be allowed over UNIX domain sockets where *path* is the location of a unix socket to listen for connections on.

none

VNC is initialized but not started. The monitor `change` command can be used to later start the VNC server.

Following the *display* value there may be one or more *option* flags separated by commas. Valid options are

reverse

Connect to a listening VNC client via a ''reverse'' connection. The client is specified by the *display*. For reverse network connections (*host*:*d*,`reverse`), the *d* argument is a TCP port number, not a display number.

websocket

Opens an additional TCP listening port dedicated to VNC Websocket connections. If a bare *websocket* option is given, the Websocket port is 5700+*display*. An alternative port can be specified with the syntax `websocket=`*port*.

If *host* is specified connections will only be allowed from this host. It is possible to control the websocket listen address independently, using the syntax `websocket=`*host*:*port*.

If no TLS credentials are provided, the websocket connection runs in unencrypted mode. If TLS credentials are provided, the websocket connection requires encrypted client connections.

password

Require that password based authentication is used for client connections.

The password must be set separately using the `set_password` command in the **pcsys_monitor**. The syntax to change your password is: `set_password <protocol> <password>` where <protocol> could be either ''vnc'' or ''spice''.

If you would like to change <protocol> password expiration, you should use `expire_password <protocol> <expiration-time>` where expiration time could be one of the following options: now, never, +seconds or UNIX time of expiration, e.g. +60 to make password expire in 60 seconds, or 1335196800 to make password expire on ''Mon Apr 23 12:00:00 EDT 2012'' (UNIX time for this date and time).

You can also use keywords ''now'' or ''never'' for the expiration time to allow <protocol> password to expire immediately or never expire.

tls−creds=*ID*

Provides the ID of a set of TLS credentials to use to secure the VNC server. They will apply to both the normal VNC server socket and the websocket socket (if enabled). Setting TLS credentials will cause the VNC server socket to enable the VeNCrypt auth mechanism. The credentials should have been previously created using the **−object tls-creds** argument.

tls−authz=*ID*

Provides the ID of the QAuthZ authorization object against which the client's x509 distinguished name will validated. This object is only resolved at time of use, so can be deleted and recreated on the fly while the VNC server is active. If missing, it will default to denying access.

sasl

Require that the client use SASL to authenticate with the VNC server. The exact choice of authentication method used is controlled from the system / user's SASL configuration file for the 'qemu' service. This is typically found in /etc/sasl2/qemu.conf. If running QEMU as an unprivileged user, an environment variable SASL_CONF_PATH can be used to make it search alternate locations for the service config. While some SASL auth methods can also provide data encryption (eg GSSAPI), it is recommended that SASL always be combined with the 'tls' and 'x509' settings to enable use of SSL and server certificates. This ensures a data encryption preventing compromise of authentication credentials. See the **vnc_security** section for details on using SASL authentication.

sasl−authz=*ID*

Provides the ID of the QAuthZ authorization object against which the client's SASL username will validated. This object is only resolved at time of use, so can be deleted and recreated on the fly while the VNC server is active. If missing, it will default to denying access.

acl Legacy method for enabling authorization of clients against the x509 distinguished name and SASL username. It results in the creation of two `authz−list` objects with IDs of `vnc.username` and `vnc.x509dname`. The rules for these objects must be configured with the HMP ACL commands.

This option is deprecated and should no longer be used. The new **sasl-authz** and **tls-authz** options are a replacement.

lossy

Enable lossy compression methods (gradient, JPEG, ...). If this option is set, VNC client may receive lossy framebuffer updates depending on its encoding settings. Enabling this option can save a lot of bandwidth at the expense of quality.

non-adaptive

Disable adaptive encodings. Adaptive encodings are enabled by default. An adaptive encoding will try to detect frequently updated screen regions, and send updates in these regions using a lossy encoding (like JPEG). This can be really helpful to save bandwidth when playing videos. Disabling adaptive encodings restores the original static behavior of encodings like Tight.

share=[allow−exclusive|force−shared|ignore]

Set display sharing policy. 'allow−exclusive' allows clients to ask for exclusive access. As suggested by the rfb spec this is implemented by dropping other connections. Connecting multiple clients in parallel requires all clients asking for a shared session (vncviewer: −shared switch). This is the default. 'force−shared' disables exclusive client access. Useful for shared desktop sessions, where you don't want someone forgetting specify −shared disconnect everybody else. 'ignore' completely ignores the shared flag and allows everybody connect unconditionally. Doesn't conform to the rfb spec but is traditional QEMU behavior.

key-delay-ms

Set keyboard delay, for key down and key up events, in milliseconds. Default is 10. Keyboards are low-bandwidth devices, so this slowdown can help the device and guest to keep up and not lose events in case events are arriving in bulk. Possible causes for the latter are flaky network connections, or scripts for automated testing.

audiodev=*audiodev*

Use the specified *audiodev* when the VNC client requests audio transmission. When not using an −audiodev argument, this option must be omitted, otherwise is must be present and specify a valid audiodev.

i386 target only

−win2k−hack

Use it when installing Windows 2000 to avoid a disk full bug. After Windows 2000 is installed, you no longer need this option (this option slows down the IDE transfers).

−no−fd−bootchk

Disable boot signature checking for floppy disks in BIOS. May be needed to boot from old floppy disks.

−no−acpi

Disable ACPI (Advanced Configuration and Power Interface) support. Use it if your guest OS complains about ACPI problems (PC target machine only).

−no−hpet

Disable HPET support.

−acpitable [sig=*str***][,rev=***n***][,oem_id=***str***][,oem_table_id=***str***][,oem_rev=***n***] [,asl_compiler_id=***str***][,asl_compiler_rev=***n***][,data=***file1***[:***file2***]...]**

Add ACPI table with specified header fields and context from specified files. For file=, take whole ACPI table from the specified files, including all ACPI headers (possible overridden by other options). For data=, only data portion of the table is used, all header information is specified in the command line. If a SLIC table is supplied to QEMU, then the SLIC's oem_id and oem_table_id fields will override the same in the RSDT and the FADT (a.k.a. FACP), in order to ensure the field matches required by the Microsoft SLIC spec and the ACPI spec.

−smbios file=*binary*

Load SMBIOS entry from binary file.

−smbios type=0[,vendor=*str***][,version=***str***][,date=***str***][,release=***%d.%d***][,uefi=on|off]**

Specify SMBIOS type 0 fields

−smbios type=1[,manufacturer=*str***][,product=***str***][,version=***str***][,serial=***str***][,uuid=***uuid***][,sku=***str***][,family=***str***]**

Specify SMBIOS type 1 fields

−smbios type=2[,manufacturer=*str***][,product=***str***][,version=***str***][,serial=***str***][,asset=***str***][,location=***str***]**

Specify SMBIOS type 2 fields

−smbios type=3[,manufacturer=*str***][,version=***str***][,serial=***str***][,asset=***str***][,sku=***str***]**

Specify SMBIOS type 3 fields

−smbios type=4[,sock_pfx=*str***][,manufacturer=***str***][,version=***str***][,serial=***str***][,asset=***str***][,part=***str***]**

Specify SMBIOS type 4 fields

−smbios type=17[,loc_pfx=*str***][,bank=***str***][,manufacturer=***str***][,serial=***str***][,asset=***str***][,part=***str***][,speed=***%d***]**

Specify SMBIOS type 17 fields

Network options

−nic [tap|bridge|user|l2tpv3|vde|netmap|vhost−user|socket][,...][,mac=macaddr][,model=mn]

This option is a shortcut for configuring both the on-board (default) guest NIC hardware and the host network backend in one go. The host backend options are the same as with the corresponding **−netdev** options below. The guest NIC model can be set with **model=***modelname*. Use **model=help** to list the available device types. The hardware MAC address can be set with **mac=***macaddr*.

The following two example do exactly the same, to show how **−nic** can be used to shorten the command line length (note that the e1000 is the default on i386, so the **model=e1000** parameter could even be omitted here, too):

```
qemu-system-x86_64 -netdev user,id=n1,ipv6=off -device e1000,netdev=n1,ma
qemu-system-x86_64 -nic user,ipv6=off,model=e1000,mac=52:54:98:76:54:32
```

−nic none

Indicate that no network devices should be configured. It is used to override the default configuration (default NIC with ‘‘user’’ host network backend) which is activated if no other networking options are provided.

−netdev user,id=*id***[,***option***][,***option***][,...]**

Configure user mode host network backend which requires no administrator privilege to run. Valid options are:

id=*id*

Assign symbolic name for use in monitor commands.

ipv4=on|off and ipv6=on|off

Specify that either IPv4 or IPv6 must be enabled. If neither is specified both protocols are enabled.

net=*addr***[/***mask***]**

Set IP network address the guest will see. Optionally specify the netmask, either in the form a.b.c.d or as number of valid top-most bits. Default is 10.0.2.0/24.

host=*addr*

Specify the guest-visible address of the host. Default is the 2nd IP in the guest network, i.e. x.x.x.2.

ipv6−net=*addr***[/***int***]**

Set IPv6 network address the guest will see (default is fec0::/64). The network prefix is given in the usual hexadecimal IPv6 address notation. The prefix size is optional, and is given as the number of valid top-most bits (default is 64).

ipv6−host=*addr*

Specify the guest-visible IPv6 address of the host. Default is the 2nd IPv6 in the guest network, i.e. xxxx::2.

restrict=on|off

If this option is enabled, the guest will be isolated, i.e. it will not be able to contact the host and no guest IP packets will be routed over the host to the outside. This option does not affect any explicitly set forwarding rules.

hostname=*name*

Specifies the client hostname reported by the built-in DHCP server.

dhcpstart=*addr*

Specify the first of the 16 IPs the built-in DHCP server can assign. Default is the 15th to 31st IP in the guest network, i.e. x.x.x.15 to x.x.x.31.

dns=*addr*

Specify the guest-visible address of the virtual nameserver. The address must be different from the host address. Default is the 3rd IP in the guest network, i.e. x.x.x.3.

ipv6−dns=*addr*

Specify the guest-visible address of the IPv6 virtual nameserver. The address must be different from the host address. Default is the 3rd IP in the guest network, i.e. xxxx::3.

dnssearch=*domain*

Provides an entry for the domain-search list sent by the built-in DHCP server. More than one domain suffix can be transmitted by specifying this option multiple times. If supported, this will cause the guest to automatically try to append the given domain suffix(es) in case a domain name can not be resolved.

Example:

```
        qemu-system-x86_64 -nic user,dnssearch=mgmt.example.org,dnssearch=exam
```

domainname=*domain*

Specifies the client domain name reported by the built-in DHCP server.

tftp=*dir*

When using the user mode network stack, activate a built-in TFTP server. The files in *dir* will be exposed as the root of a TFTP server. The TFTP client on the guest must be configured in binary mode (use the command `bin` of the Unix TFTP client).

tftp−server−name=*name*

In BOOTP reply, broadcast *name* as the "TFTP server name" (RFC2132 option 66). This can be used to advise the guest to load boot files or configurations from a different server than the host address.

bootfile=*file*

When using the user mode network stack, broadcast *file* as the BOOTP filename. In conjunction with **tftp**, this can be used to network boot a guest from a local directory.

Example (using pxelinux):

```
        qemu-system-x86_64 -hda linux.img -boot n -device e1000,netdev=n1 \
        -netdev user,id=n1,tftp=/path/to/tftp/files,bootfile=/pxelinux.0
```

smb=*dir***[,smbserver=***addr***]**

When using the user mode network stack, activate a built-in SMB server so that Windows OSes can access to the host files in *dir* transparently. The IP address of the SMB server can be set to *addr*. By default the 4th IP in the guest network is used, i.e. x.x.x.4.

In the guest Windows OS, the line:

```
        10.0.2.4 smbserver
```

must be added in the file *C:\WINDOWS\LMHOSTS* (for windows 9x/Me) or *C:\WINNT\SYSTEM32\DRIVERS\ETC\LMHOSTS* (Windows NT/2000).

Then *dir* can be accessed in *\\smbserver\qemu*.

Note that a SAMBA server must be installed on the host OS.

hostfwd=[tcp|udp]:[*hostaddr***]:***hostport***−[***guestaddr***]:***guestport*

Redirect incoming TCP or UDP connections to the host port *hostport* to the guest IP address *guestaddr* on guest port *guestport*. If *guestaddr* is not specified, its value is x.x.x.15 (default first address given by the built-in DHCP server). By specifying *hostaddr*, the rule can be bound to a specific host interface. If no connection type is set, TCP is used. This option can be given multiple times.

For example, to redirect host X11 connection from screen 1 to guest screen 0, use the following:

```
        # on the host
        qemu-system-x86_64 -nic user,hostfwd=tcp:127.0.0.1:6001-:6000
        # this host xterm should open in the guest X11 server
        xterm -display :1
```

To redirect telnet connections from host port 5555 to telnet port on the guest, use the following:

```
        # on the host
        qemu-system-x86_64 -nic user,hostfwd=tcp::5555-:23
        telnet localhost 5555
```

Then when you use on the host `telnet localhost 5555`, you connect to the guest telnet server.

guestfwd=[tcp]:*server***:***port***−***dev*

guestfwd=[tcp]:*server***:***port***−***cmd:command*

Forward guest TCP connections to the IP address *server* on port *port* to the character device *dev* or to a program executed by *cmd:command* which gets spawned for each connection. This option can be given multiple times.

You can either use a chardev directly and have that one used throughout QEMU's lifetime, like in the following example:

```
# open 10.10.1.1:4321 on bootup, connect 10.0.2.100:1234 to it wheneve
# the guest accesses it
qemu-system-x86_64 -nic user,guestfwd=tcp:10.0.2.100:1234-tcp:10.10.1
```

Or you can execute a command on every TCP connection established by the guest, so that QEMU behaves similar to an inetd process for that virtual server:

```
# call "netcat 10.10.1.1 4321" on every TCP connection to 10.0.2.100:
# and connect the TCP stream to its stdin/stdout
qemu-system-x86_64 -nic  'user,id=n1,guestfwd=tcp:10.0.2.100:1234-cmd
```

−netdev tap,id=*id***[,fd=***h***][,ifname=***name***][,script=***file***][,downscript=***dfile***][,br=***bridge***][,helper=***helper***]**

Configure a host TAP network backend with ID *id*.

Use the network script *file* to configure it and the network script *dfile* to deconfigure it. If *name* is not provided, the OS automatically provides one. The default network configure script is */etc/qemu−ifup* and the default network deconfigure script is */etc/qemu−ifdown*. Use **script=no** or **downscript=no** to disable script execution.

If running QEMU as an unprivileged user, use the network helper *helper* to configure the TAP interface and attach it to the bridge. The default network helper executable is */path/to/qemu−bridge−helper* and the default bridge device is *br0*.

fd=*h* can be used to specify the handle of an already opened host TAP interface.

Examples:

```
#launch a QEMU instance with the default network script
qemu-system-x86_64 linux.img -nic tap



#launch a QEMU instance with two NICs, each one connected
#to a TAP device
qemu-system-x86_64 linux.img \
-netdev tap,id=nd0,ifname=tap0 -device e1000,netdev=nd0 \
-netdev tap,id=nd1,ifname=tap1 -device rtl8139,netdev=nd1



#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge br0
qemu-system-x86_64 linux.img -device virtio-net-pci,netdev=n1 \
-netdev tap,id=n1,"helper=/path/to/qemu-bridge-helper"
```

−netdev bridge,id=*id***[,br=***bridge***][,helper=***helper***]**

Connect a host TAP network interface to a host bridge device.

Use the network helper *helper* to configure the TAP interface and attach it to the bridge. The default network helper executable is */path/to/qemu−bridge−helper* and the default bridge device is *br0*.

Examples:

```
#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge br0
qemu-system-x86_64 linux.img -netdev bridge,id=n1 -device virtio-net,netd



#launch a QEMU instance with the default network helper to
#connect a TAP device to bridge qemubr0
qemu-system-x86_64 linux.img -netdev bridge,br=qemubr0,id=n1 -device virt
```

−netdev socket,id=*id***[,fd=***h***][,listen=[***host***]:***port***][,connect=***host***:***port***]**

This host network backend can be used to connect the guest's network to another QEMU virtual machine using a TCP socket connection. If **listen** is specified, QEMU waits for incoming connections on *port* (*host* is optional). **connect** is used to connect to another QEMU instance using the **listen** option. **fd**=*h* specifies an already opened TCP socket.

Example:

```
# launch a first QEMU instance
qemu-system-x86_64 linux.img \
-device e1000,netdev=n1,mac=52:54:00:12:34:56 \
-netdev socket,id=n1,listen=:1234
# connect the network of this instance to the network of the first instan
qemu-system-x86_64 linux.img \
-device e1000,netdev=n2,mac=52:54:00:12:34:57 \
-netdev socket,id=n2,connect=127.0.0.1:1234
```

−netdev socket,id=*id***[,fd=***h***][,mcast=***maddr***:***port***[,localaddr=***addr***]]**

Configure a socket host network backend to share the guest's network traffic with another QEMU virtual machines using a UDP multicast socket, effectively making a bus for every QEMU with same multicast address *maddr* and *port*. NOTES:

1. Several QEMU can be running on different hosts and share same bus (assuming correct multicast setup for these hosts).
2. mcast support is compatible with User Mode Linux (argument **eth***N***=mcast**), see **<http://user−mode−linux.sf.net>**.
3. Use **fd=h** to specify an already opened UDP multicast socket.

Example:

```
# launch one QEMU instance
qemu-system-x86_64 linux.img \
-device e1000,netdev=n1,mac=52:54:00:12:34:56 \
-netdev socket,id=n1,mcast=230.0.0.1:1234
# launch another QEMU instance on same "bus"
qemu-system-x86_64 linux.img \
-device e1000,netdev=n2,mac=52:54:00:12:34:57 \
-netdev socket,id=n2,mcast=230.0.0.1:1234
# launch yet another QEMU instance on same "bus"
qemu-system-x86_64 linux.img \
-device e1000,netdev=n3,mac=52:54:00:12:34:58 \
-netdev socket,id=n3,mcast=230.0.0.1:1234
```

Example (User Mode Linux compat.):

```
# launch QEMU instance (note mcast address selected is UML's default)
qemu-system-x86_64 linux.img \
-device e1000,netdev=n1,mac=52:54:00:12:34:56 \
-netdev socket,id=n1,mcast=239.192.168.1:1102
# launch UML
/path/to/linux ubd0=/path/to/root_fs eth0=mcast
```

Example (send packets from host's 1.2.3.4):

```
qemu-system-x86_64 linux.img \
-device e1000,netdev=n1,mac=52:54:00:12:34:56 \
-netdev socket,id=n1,mcast=239.192.168.1:1102,localaddr=1.2.3.4
```

–netdev l2tpv3,id=*id***,src=***srcaddr***,dst=***dstaddr***[,srcport=***srcport***][,dstport=***dstport***],txsession=***txsession***[,rxsession=***rxsession***][,ipv6][**

Configure a L2TPv3 pseudowire host network backend. L2TPv3 (RFC3391) is a popular protocol to transport Ethernet (and other Layer 2) data frames between two systems. It is present in routers, firewalls and the Linux kernel (from version 3.3 onwards).

This transport allows a VM to communicate to another VM, router or firewall directly.

src=*srcaddr*
source address (mandatory)

dst=*dstaddr*
destination address (mandatory)

udp
select udp encapsulation (default is ip).

srcport=*srcport*
source udp port.

dstport=*dstport*
destination udp port.

ipv6
force v6, otherwise defaults to v4.

rxcookie=*rxcookie*
txcookie=*txcookie*
Cookies are a weak form of security in the l2tpv3 specification. Their function is mostly to prevent misconfiguration. By default they are 32 bit.

cookie64
Set cookie size to 64 bit instead of the default 32

counter=off
Force a 'cut–down' L2TPv3 with no counter as in draft–mkonstan–l2tpext–keyed–ipv6–tunnel–00

pincounter=on
Work around broken counter handling in peer. This may also help on networks which have packet reorder.

offset=*offset*
Add an extra offset between header and data

For example, to attach a VM running on host 4.3.2.1 via L2TPv3 to the bridge br-lan on the remote Linux host 1.2.3.4:

```
# Setup tunnel on linux host using raw ip as encapsulation
# on 1.2.3.4
ip l2tp add tunnel remote 4.3.2.1 local 1.2.3.4 tunnel_id 1 peer_tunnel_i
encap udp udp_sport 16384 udp_dport 16384
ip l2tp add session tunnel_id 1 name vmtunnel0 session_id \
0xFFFFFFFF peer_session_id 0xFFFFFFFF
ifconfig vmtunnel0 mtu 1500
ifconfig vmtunnel0 up
brctl addif br-lan vmtunnel0


# on 4.3.2.1
# launch QEMU instance - if your network has reorder or is very lossy add

qemu-system-x86_64 linux.img -device e1000,netdev=n1 \
-netdev l2tpv3,id=n1,src=4.2.3.1,dst=1.2.3.4,udp,srcport=16384,dstport=16
```

−netdev vde,id=*id***[,sock=***socketpath***][,port=***n***][,group=***groupname***][,mode=***octalmode***]**

Configure VDE backend to connect to PORT *n* of a vde switch running on host and listening for incoming connections on *socketpath*. Use GROUP *groupname* and MODE *octalmode* to change default ownership and permissions for communication port. This option is only available if QEMU has been compiled with vde support enabled.

Example:

```
# launch vde switch
vde_switch -F -sock /tmp/myswitch
# launch QEMU instance
qemu-system-x86_64 linux.img -nic vde,sock=/tmp/myswitch
```

−netdev vhost−user,chardev=*id***[,vhostforce=on|off][,queues=n]**

Establish a vhost-user netdev, backed by a chardev *id*. The chardev should be a unix domain socket backed one. The vhost-user uses a specifically defined protocol to pass vhost ioctl replacement messages to an application on the other end of the socket. On non-MSIX guests, the feature can be forced with *vhostforce*. Use 'queues=*n*' to specify the number of queues to be created for multiqueue vhost-user.

Example:

```
qemu -m 512 -object memory-backend-file,id=mem,size=512M,mem-path=/hugetl
-numa node,memdev=mem \
-chardev socket,id=chr0,path=/path/to/socket \
-netdev type=vhost-user,id=net0,chardev=chr0 \
-device virtio-net-pci,netdev=net0
```

−netdev hubport,id=*id***,hubid=***hubid***[,netdev=***nd***]**

Create a hub port on the emulated hub with ID *hubid*.

The hubport netdev lets you connect a NIC to a QEMU emulated hub instead of a single netdev. Alternatively, you can also connect the hubport to another netdev with ID *nd* by using the **netdev=***nd* option.

−net nic[,netdev=*nd***][,macaddr=***mac***][,model=***type***] [,name=***name***][,addr=***addr***][,vectors=***v***]**

Legacy option to configure or create an on-board (or machine default) Network Interface Card(NIC) and connect it either to the emulated hub with ID 0 (i.e. the default hub), or to the netdev *nd*. The NIC is an e1000 by default on the PC target. Optionally, the MAC address can be changed to *mac*, the device address set to *addr* (PCI cards only), and a *name* can be assigned for use in monitor commands. Optionally, for PCI cards, you can specify the number *v* of MSI-X vectors that the card should have; this option currently only affects virtio cards; set *v* = 0 to disable MSI-X. If no **−net** option is specified, a single NIC is created. QEMU can emulate several different models of network card. Use `-net nic,model=help` for a list of available devices for your target.

-net user|tap|bridge|socket|l2tpv3|vde[,...][,name=*name***]**

Configure a host network backend (with the options corresponding to the same **-netdev** option) and connect it to the emulated hub 0 (the default hub). Use *name* to specify the name of the hub port.

Character device options

The general form of a character device option is:

-chardev *backend***,id=***id***[,mux=on|off][,***options***]**

Backend is one of: **null**, **socket**, **udp**, **msmouse**, **vc**, **ringbuf**, **file**, **pipe**, **console**, **serial**, **pty**, **stdio**, **braille**, **tty**, **parallel**, **parport**, **spicevmc**, **spiceport**. The specific backend will determine the applicable options.

Use `-chardev help` to print all available chardev backend types.

All devices must have an id, which can be any string up to 127 characters long. It is used to uniquely identify this device in other command line directives.

A character device may be used in multiplexing mode by multiple front-ends. Specify **mux=on** to enable this mode. A multiplexer is a "1:N" device, and here the "1" end is your specified chardev backend, and the "N" end is the various parts of QEMU that can talk to a chardev. If you create a chardev with **id=myid** and **mux=on**, QEMU will create a multiplexer with your specified ID, and you can then configure multiple front ends to use that chardev ID for their input/output. Up to four different front ends can be connected to a single multiplexed chardev. (Without multiplexing enabled, a chardev can only be used by a single front end.) For instance you could use this to allow a single stdio chardev to be used by two serial ports and the QEMU monitor:

```
-chardev stdio,mux=on,id=char0 \
-mon chardev=char0,mode=readline \
-serial chardev:char0 \
-serial chardev:char0
```

You can have more than one multiplexer in a system configuration; for instance you could have a TCP port multiplexed between UART 0 and UART 1, and stdio multiplexed between the QEMU monitor and a parallel port:

```
-chardev stdio,mux=on,id=char0 \
-mon chardev=char0,mode=readline \
-parallel chardev:char0 \
-chardev tcp,...,mux=on,id=char1 \
-serial chardev:char1 \
-serial chardev:char1
```

When you're using a multiplexed character device, some escape sequences are interpreted in the input.

Note that some other command line options may implicitly create multiplexed character backends; for instance **-serial mon:stdio** creates a multiplexed stdio backend connected to the serial port and the QEMU monitor, and **-nographic** also multiplexes the console and the monitor to stdio.

There is currently no support for multiplexing in the other direction (where a single QEMU front end takes input and output from multiple chardevs).

Every backend supports the **logfile** option, which supplies the path to a file to record all data transmitted via the backend. The **logappend** option controls whether the log file will be truncated or appended to when opened.

The available backends are:

−chardev null,id=*id*

A void device. This device will not emit any data, and will drop any data it receives. The null backend does not take any options.

−chardev socket,id=*id***[,***TCP options* **or** *unix options***][,server][,nowait][,telnet][,websocket][,reconnect=***seconds***][,tls−creds=***id***][,tls−authz=***id***]**

Create a two-way stream socket, which can be either a TCP or a unix socket. A unix socket will be created if **path** is specified. Behaviour is undefined if TCP options are specified for a unix socket.

server specifies that the socket shall be a listening socket.

nowait specifies that QEMU should not block waiting for a client to connect to a listening socket.

telnet specifies that traffic on the socket should interpret telnet escape sequences.

websocket specifies that the socket uses WebSocket protocol for communication.

reconnect sets the timeout for reconnecting on non-server sockets when the remote end goes away. qemu will delay this many seconds and then attempt to reconnect. Zero disables reconnecting, and is the default.

tls-creds requests enablement of the TLS protocol for encryption, and specifies the id of the TLS credentials to use for the handshake. The credentials must be previously created with the **−object tls-creds** argument.

tls-auth provides the ID of the QAuthZ authorization object against which the client's x509 distinguished name will be validated. This object is only resolved at time of use, so can be deleted and recreated on the fly while the chardev server is active. If missing, it will default to denying access.

TCP and unix socket options are given below:

TCP options: port=*port***[,host=***host***][,to=***to***][,ipv4][,ipv6][,nodelay]**

host for a listening socket specifies the local address to be bound. For a connecting socket species the remote host to connect to. **host** is optional for listening sockets. If not specified it defaults to `0.0.0.0`.

port for a listening socket specifies the local port to be bound. For a connecting socket specifies the port on the remote host to connect to. **port** can be given as either a port number or a service name. **port** is required.

to is only relevant to listening sockets. If it is specified, and **port** cannot be bound, QEMU will attempt to bind to subsequent ports up to and including **to** until it succeeds. **to** must be specified as a port number.

ipv4 and **ipv6** specify that either IPv4 or IPv6 must be used. If neither is specified the socket may use either protocol.

nodelay disables the Nagle algorithm.

unix options: path=*path*

path specifies the local path of the unix socket. **path** is required.

−chardev udp,id=*id***[,host=***host***],port=***port***[,localaddr=***localaddr***][,localport=***localport***][,ipv4][,ipv6]**

Sends all traffic from the guest to a remote host over UDP.

host specifies the remote host to connect to. If not specified it defaults to `localhost`.

port specifies the port on the remote host to connect to. **port** is required.

localaddr specifies the local address to bind to. If not specified it defaults to `0.0.0.0`.

localport specifies the local port to bind to. If not specified any available local port will be used.

ipv4 and **ipv6** specify that either IPv4 or IPv6 must be used. If neither is specified the device may use either protocol.

−chardev msmouse,id=*id*

Forward QEMU's emulated msmouse events to the guest. **msmouse** does not take any options.

−chardev vc,id=*id***[[,width=***width***][,height=***height***]][[,cols=***cols***][,rows=***rows***]]**

Connect to a QEMU text console. **vc** may optionally be given a specific size.

width and **height** specify the width and height respectively of the console, in pixels.

cols and **rows** specify that the console be sized to fit a text console with the given dimensions.

−chardev ringbuf,id=*id***[,size=***size***]**

Create a ring buffer with fixed size **size**. *size* must be a power of two and defaults to `64K`.

−chardev file,id=*id***,path=***path*

Log all traffic received from the guest to a file.

path specifies the path of the file to be opened. This file will be created if it does not already exist, and overwritten if it does. **path** is required.

−chardev pipe,id=*id***,path=***path*

Create a two-way connection to the guest. The behaviour differs slightly between Windows hosts and other hosts:

On Windows, a single duplex pipe will be created at \\.*pipe****path***.

On other hosts, 2 pipes will be created called ***path***.*in* and ***path***.*out*. Data written to ***path***.*in* will be received by the guest. Data written by the guest can be read from ***path***.*out*. QEMU will not create these fifos, and requires them to be present.

path forms part of the pipe path as described above. **path** is required.

−chardev console,id=*id*

Send traffic from the guest to QEMU's standard output. **console** does not take any options.

console is only available on Windows hosts.

−chardev serial,id=*id***,path=path**

Send traffic from the guest to a serial device on the host.

On Unix hosts serial will actually accept any tty device, not only serial lines.

path specifies the name of the serial device to open.

−chardev pty,id=*id*

Create a new pseudo-terminal on the host and connect to it. **pty** does not take any options.

pty is not available on Windows hosts.

−chardev stdio,id=*id***[,signal=on|off]**

Connect to standard input and standard output of the QEMU process.

signal controls if signals are enabled on the terminal, that includes exiting QEMU with the key sequence **Control-c**. This option is enabled by default, use **signal=off** to disable it.

−chardev braille,id=*id*

Connect to a local BrlAPI server. **braille** does not take any options.

−chardev tty,id=*id***,path=***path*

tty is only available on Linux, Sun, FreeBSD, NetBSD, OpenBSD and DragonFlyBSD hosts. It is an alias for **serial**.

path specifies the path to the tty. **path** is required.

−chardev parallel,id=*id***,path=***path*
−chardev parport,id=*id***,path=***path*

parallel is only available on Linux, FreeBSD and DragonFlyBSD hosts.

Connect to a local parallel port.

path specifies the path to the parallel port device. **path** is required.

−chardev spicevmc,id=*id***,debug=***debug***,name=***name*

spicevmc is only available when spice support is built in.

debug debug level for spicevmc

name name of spice channel to connect to

Connect to a spice virtual machine channel, such as vdiport.

−chardev spiceport,id=*id***,debug=***debug***,name=***name*

spiceport is only available when spice support is built in.

debug debug level for spicevmc

name name of spice port to connect to

Connect to a spice port, allowing a Spice client to handle the traffic identified by a name (preferably a fqdn).

Bluetooth(R) options

−bt hci[...]

Defines the function of the corresponding Bluetooth HCI. −bt options are matched with the HCIs present in the chosen machine type. For example when emulating a machine with only one HCI built into it, only the first `-bt hci[...]` option is valid and defines the HCI's logic. The Transport Layer is decided by the machine type. Currently the machines `n800` and `n810` have one HCI and all other machines have none.

Note: This option and the whole bluetooth subsystem is considered as deprecated. If you still use it, please send a mail to **<qemu−devel@nongnu.org>** where you describe your usecase.

The following three types are recognized:

−bt hci,null

(default) The corresponding Bluetooth HCI assumes no internal logic and will not respond to any HCI commands or emit events.

−bt hci,host[:*id***]**

(`bluez` only) The corresponding HCI passes commands / events to / from the physical HCI identified by the name *id* (default: `hci0`) on the computer running QEMU. Only available on `bluez` capable systems like Linux.

−bt hci[,vlan=*n***]**

Add a virtual, standard HCI that will participate in the Bluetooth scatternet *n* (default `0`). Similarly to **−net** VLANs, devices inside a bluetooth network *n* can only communicate with other devices in the same network (scatternet).

−bt vhci[,vlan=*n***]**

(Linux-host only) Create a HCI in scatternet *n* (default 0) attached to the host bluetooth stack instead of to the emulated target. This allows the host and target machines to participate in a common scatternet and communicate. Requires the Linux `vhci` driver installed. Can be used as following:

```
qemu-system-x86_64 [...OPTIONS...] -bt hci,vlan=5 -bt vhci,vlan=5
```

−bt device:*dev***[,vlan=***n***]**

Emulate a bluetooth device *dev* and place it in network *n* (default `0`). QEMU can only emulate one type of bluetooth devices currently:

keyboard

Virtual wireless keyboard implementing the HIDP bluetooth profile.

TPM device options

The general form of a TPM device option is:

−tpmdev *backend***,id=***id***[,***options***]**

The specific backend type will determine the applicable options. The `−tpmdev` option creates the TPM backend and requires a `−device` option that specifies the TPM frontend interface model.

Use `−tpmdev help` to print all available TPM backend types.

The available backends are:

−tpmdev passthrough,id=*id***,path=***path***,cancel−path=***cancel-path*

(Linux-host only) Enable access to the host's TPM using the passthrough driver.

path specifies the path to the host's TPM device, i.e., on a Linux host this would be `/dev/tpm0`. **path** is optional and by default `/dev/tpm0` is used.

cancel-path specifies the path to the host TPM device's sysfs entry allowing for cancellation of an ongoing TPM command. **cancel-path** is optional and by default QEMU will search for the sysfs entry to use.

Some notes about using the host's TPM with the passthrough driver:

The TPM device accessed by the passthrough driver must not be used by any other application on the host.

Since the host's firmware (BIOS/UEFI) has already initialized the TPM, the VM's firmware (BIOS/UEFI) will not be able to initialize the TPM again and may therefore not show a TPM-specific menu that would otherwise allow the user to configure the TPM, e.g., allow the user to enable/disable or activate/deactivate the TPM. Further, if TPM ownership is released from within a VM then the host's TPM will get disabled and deactivated. To enable and activate the TPM again afterwards, the host has to be rebooted and the user is required to enter the firmware's menu to enable and activate the TPM. If the TPM is left disabled and/or deactivated most TPM commands will fail.

To create a passthrough TPM use the following two options:

```
−tpmdev passthrough,id=tpm0 −device tpm−tis,tpmdev=tpm0
```

Note that the `−tpmdev` id is `tpm0` and is referenced by `tpmdev=tpm0` in the device option.

−tpmdev emulator,id=*id***,chardev=***dev*

(Linux-host only) Enable access to a TPM emulator using Unix domain socket based chardev backend.

chardev specifies the unique ID of a character device backend that provides connection to the software TPM server.

To create a TPM emulator backend device with chardev socket backend:

```
−chardev socket,id=chrtpm,path=/tmp/swtpm−sock −tpmdev emulator,id=tpm0,c
```

Linux/Multiboot boot specific

When using these options, you can use a given Linux or Multiboot kernel without installing it in the disk image. It can be useful for easier testing of various kernels.

−kernel *bzImage*

Use *bzImage* as kernel image. The kernel can be either a Linux kernel or in multiboot format.

−append *cmdline*

Use *cmdline* as kernel command line

−initrd *file*

Use *file* as initial ram disk.

−initrd "*file1* **arg=foo,***file2***"**

This syntax is only available with multiboot.

Use *file1* and *file2* as modules and pass arg=foo as parameter to the first module.

–dtb *file*

Use *file* as a device tree binary (dtb) image and pass it to the kernel on boot.

Debug/Expert options

–fw_cfg [name=]*name***,file=***file*

Add named fw_cfg entry with contents from file *file*.

–fw_cfg [name=]*name***,string=***str*

Add named fw_cfg entry with contents from string *str*.

The terminating NUL character of the contents of *str* will not be included as part of the fw_cfg item data. To insert contents with embedded NUL characters, you have to use the *file* parameter.

The fw_cfg entries are passed by QEMU through to the guest.

Example:

```
-fw_cfg name=opt/com.mycompany/blob,file=./my_blob.bin
```

creates an fw_cfg entry named opt/com.mycompany/blob with contents from ./my_blob.bin.

–serial *dev*

Redirect the virtual serial port to host character device *dev*. The default device is `vc` in graphical mode and `stdio` in non graphical mode.

This option can be used several times to simulate up to 4 serial ports.

Use `-serial none` to disable all serial ports.

Available character devices are:

vc[:*W***x***H***]**

Virtual console. Optionally, a width and height can be given in pixel with

```
vc:800x600
```

It is also possible to specify width or height in characters:

```
vc:80Cx24C
```

pty [Linux only] Pseudo TTY (a new PTY is automatically allocated)

none

No device is allocated.

null

void device

chardev:*id*

Use a named character device defined with the `-chardev` option.

/dev/XXX

[Linux only] Use host tty, e.g. */dev/ttyS0*. The host serial port parameters are set according to the emulated ones.

/dev/parport*N*

[Linux only, parallel port only] Use host parallel port *N*. Currently SPP and EPP parallel port features can be used.

file:*filename*

Write output to *filename*. No character can be read.

stdio

[Unix only] standard input/output

pipe:*filename*
name pipe *filename*

COM*n*
[Windows only] Use host serial port *n*

udp:[*remote_host***]:***remote_port***[@[***src_ip***]:***src_port***]**
This implements UDP Net Console. When *remote_host* or *src_ip* are not specified they default to `0.0.0.0`. When not using a specified *src_port* a random port is automatically chosen.

If you just want a simple readonly console you can use `netcat` or `nc`, by starting QEMU with: `-serial udp::4555` and nc as: `nc -u -l -p 4555`. Any time QEMU writes something to that port it will appear in the netconsole session.

If you plan to send characters back via netconsole or you want to stop and start QEMU a lot of times, you should have QEMU use the same source port each time by using something like `-serial udp::4555@4556` to QEMU. Another approach is to use a patched version of netcat which can listen to a TCP port and send and receive characters via udp. If you have a patched version of netcat which activates telnet remote echo and single char transfer, then you can use the following options to set up a netcat redirector to allow telnet on port 5555 to access the QEMU port.

`QEMU Options:`
–serial udp::4555@4556

`netcat options:`
–u –P 4555 –L 0.0.0.0:4556 –t –p 5555 –I –T

`telnet options:`
localhost 5555

tcp:[*host***]:***port***[,***server***][,nowait][,nodelay][,reconnect=***seconds***]**
The TCP Net Console has two modes of operation. It can send the serial I/O to a location or wait for a connection from a location. By default the TCP Net Console is sent to *host* at the *port*. If you use the *server* option QEMU will wait for a client socket application to connect to the port before continuing, unless the `nowait` option was specified. The `nodelay` option disables the Nagle buffering algorithm. The `reconnect` option only applies if *noserver* is set, if the connection goes down it will attempt to reconnect at the given interval. If *host* is omitted, 0.0.0.0 is assumed. Only one TCP connection at a time is accepted. You can use `telnet` to connect to the corresponding character device.

`Example to send tcp console to 192.168.0.2 port 4444`
–serial tcp:192.168.0.2:4444

`Example to listen and wait on port 4444 for connection`
–serial tcp::4444,server

`Example to not wait and listen on ip 192.168.0.100 port 4444`
–serial tcp:192.168.0.100:4444,server,nowait

telnet:*host***:***port***[,server][,nowait][,nodelay]**
The telnet protocol is used instead of raw tcp sockets. The options work the same as if you had specified `-serial tcp`. The difference is that the port acts like a telnet server or client using telnet option negotiation. This will also allow you to send the MAGIC_SYSRQ sequence if you use a telnet that supports sending the break sequence. Typically in unix telnet you do it with Control–] and then type "send break" followed by pressing the enter key.

websocket:*host***:***port***,server[,nowait][,nodelay]**
The WebSocket protocol is used instead of raw tcp socket. The port acts as a WebSocket server. Client mode is not supported.

unix:*path***[,server][,nowait][,reconnect=***seconds***]**
A unix domain socket is used instead of a tcp socket. The option works the same as if you had specified `-serial tcp` except the unix domain socket *path* is used for connections.

mon:*dev_string*
This is a special option to allow the monitor to be multiplexed onto another serial port. The monitor is accessed with key sequence of **Control-a** and then pressing **c**. *dev_string* should be any one of the serial devices specified above. An example to multiplex the monitor onto a telnet server listening on port 4444 would be:

```
-serial mon:telnet::4444,server,nowait
```

When the monitor is multiplexed to stdio in this way, Ctrl+C will not terminate QEMU any more but will be passed to the guest instead.

braille
Braille device. This will use BrlAPI to display the braille output on a real or fake device.

msmouse
Three button serial mouse. Configure the guest to use Microsoft protocol.

–parallel *dev*
Redirect the virtual parallel port to host device *dev* (same devices as the serial port). On Linux hosts, */dev/parportN* can be used to use hardware devices connected on the corresponding host parallel port.

This option can be used several times to simulate up to 3 parallel ports.

Use `-parallel none` to disable all parallel ports.

–monitor *dev*
Redirect the monitor to host device *dev* (same devices as the serial port). The default device is `vc` in graphical mode and `stdio` in non graphical mode. Use `-monitor none` to disable the default monitor.

–qmp *dev*
Like –monitor but opens in 'control' mode.

–qmp–pretty *dev*
Like –qmp but uses pretty JSON formatting.

–mon [chardev=]name[,mode=readline|control][,pretty[=on|off]]
Setup monitor on chardev *name*. `pretty` turns on JSON pretty printing easing human reading and debugging.

–debugcon *dev*
Redirect the debug console to host device *dev* (same devices as the serial port). The debug console is an I/O port which is typically port 0xe9; writing to that I/O port sends output to this device. The default device is `vc` in graphical mode and `stdio` in non graphical mode.

–pidfile *file*
Store the QEMU process PID in *file*. It is useful if you launch QEMU from a script.

–singlestep
Run the emulation in single step mode.

––preconfig
Pause QEMU for interactive configuration before the machine is created, which allows querying and configuring properties that will affect machine initialization. Use QMP command 'x–exit–preconfig' to exit the preconfig state and move to the next state (i.e. run guest if –S isn't used or pause the second time if –S is used). This option is experimental.

–S Do not start CPU at startup (you must type 'c' in the monitor).

−realtime mlock=on|off

Run qemu with realtime features. mlocking qemu and guest memory can be enabled via **mlock=on** (enabled by default).

−overcommit mem−lock=on|off
−overcommit cpu−pm=on|off

Run qemu with hints about host resource overcommit. The default is to assume that host overcommits all resources.

Locking qemu and guest memory can be enabled via **mem−lock=on** (disabled by default). This works when host memory is not overcommitted and reduces the worst-case latency for guest. This is equivalent to **realtime**.

Guest ability to manage power state of host cpus (increasing latency for other processes on the same host cpu, but decreasing latency for guest) can be enabled via **cpu−pm=on** (disabled by default). This works best when host CPU is not overcommitted. When used, host estimates of CPU cycle and power utilization will be incorrect, not taking into account guest idle time.

−gdb *dev*

Wait for gdb connection on device *dev*. Typical connections will likely be TCP-based, but also UDP, pseudo TTY, or even stdio are reasonable use case. The latter is allowing to start QEMU from within gdb and establish the connection via a pipe:

```
(gdb) target remote | exec qemu-system-x86_64 -gdb stdio ...
```

−s Shorthand for −gdb tcp::1234, i.e. open a gdbserver on TCP port 1234.

−d *item1***[,...]**

Enable logging of specified items. Use '−d help' for a list of log items.

−D *logfile*

Output log in *logfile* instead of to stderr

−dfilter *range1***[,...]**

Filter debug output to that relevant to a range of target addresses. The filter spec can be either *start+size*, *start−size* or *start..end* where *start end* and *size* are the addresses and sizes required. For example:

```
-dfilter 0x8000..0x8fff,0xffffffc000080000+0x200,0xffffffc000060000-0x100
```

Will dump output for any code in the 0x1000 sized block starting at 0x8000 and the 0x200 sized block starting at 0xffffffc000080000 and another 0x1000 sized block starting at 0xffffffc00005f000.

−seed *number*

Force the guest to use a deterministic pseudo-random number generator, seeded with *number*. This does not affect crypto routines within the host.

−L *path*

Set the directory for the BIOS, VGA BIOS and keymaps.

To list all the data directories, use `-L help`.

−bios *file*

Set the filename for the BIOS.

−enable−kvm

Enable KVM full virtualization support. This option is only available if KVM support is enabled when compiling.

−xen−domid *id*

Specify xen guest domain *id* (XEN only).

−xen−attach

Attach to existing xen domain. libxl will use this when starting QEMU (XEN only). Restrict set of available xen operations to specified domain id (XEN only).

−no−reboot
Exit instead of rebooting.

−no−shutdown
Don't exit QEMU on guest shutdown, but instead only stop the emulation. This allows for instance switching to monitor to commit changes to the disk image.

−loadvm *file*
Start right away with a saved state (`loadvm` in monitor)

−daemonize
Daemonize the QEMU process after initialization. QEMU will not detach from standard IO until it is ready to receive connections on any of its devices. This option is a useful way for external programs to launch QEMU without having to cope with initialization race conditions.

−option−rom *file*
Load the contents of *file* as an option ROM. This option is useful to load things like EtherBoot.

−rtc [base=utc|localtime|*datetime***][,clock=host|rt|vm][,driftfix=none|slew]**
Specify **base** as `utc` or `localtime` to let the RTC start at the current UTC or local time, respectively. `localtime` is required for correct date in MS-DOS or Windows. To start at a specific point in time, provide *datetime* in the format `2006−06−17T16:01:21` or `2006−06−17`. The default base is UTC.

By default the RTC is driven by the host system time. This allows using of the RTC as accurate reference clock inside the guest, specifically if the host time is smoothly following an accurate external reference clock, e.g. via NTP. If you want to isolate the guest time from the host, you can set **clock** to `rt` instead, which provides a host monotonic clock if host support it. To even prevent the RTC from progressing during suspension, you can set **clock** to `vm` (virtual clock). **clock=vm** is recommended especially in icount mode in order to preserve determinism; however, note that in icount mode the speed of the virtual clock is variable and can in general differ from the host clock.

Enable **driftfix** (i386 targets only) if you experience time drift problems, specifically with Windows' ACPI HAL. This option will try to figure out how many timer interrupts were not processed by the Windows guest and will re-inject them.

−icount [shift=*N***|auto][,rr=record|replay,rrfile=***filename***,rrsnapshot=***snapshot***]**
Enable virtual instruction counter. The virtual cpu will execute one instruction every 2^*N* ns of virtual time. If `auto` is specified then the virtual cpu speed will be automatically adjusted to keep virtual time within a few seconds of real time.

When the virtual cpu is sleeping, the virtual time will advance at default speed unless **sleep=on|off** is specified. With **sleep=on|off**, the virtual time will jump to the next timer deadline instantly whenever the virtual cpu goes to sleep mode and will not advance if no timer is enabled. This behavior give deterministic execution times from the guest point of view.

Note that while this option can give deterministic behavior, it does not provide cycle accurate emulation. Modern CPUs contain superscalar out of order cores with complex cache hierarchies. The number of instructions executed often has little or no correlation with actual performance.

align=on will activate the delay algorithm which will try to synchronise the host clock and the virtual clock. The goal is to have a guest running at the real frequency imposed by the shift option. Whenever the guest clock is behind the host clock and if **align=on** is specified then we print a message to the user to inform about the delay. Currently this option does not work when **shift** is `auto`. Note: The sync algorithm will work for those shift values for which the guest clock runs ahead of the host clock. Typically this happens when the shift value is high (how high depends on the host machine).

When **rr** option is specified deterministic record/replay is enabled. Replay log is written into *filename* file in record mode and read from this file in replay mode.

Option rrsnapshot is used to create new vm snapshot named *snapshot* at the start of execution recording. In replay mode this option is used to load the initial VM state.

−watchdog *model*

Create a virtual hardware watchdog device. Once enabled (by a guest action), the watchdog must be periodically polled by an agent inside the guest or else the guest will be restarted. Choose a model for which your guest has drivers.

The *model* is the model of hardware watchdog to emulate. Use `-watchdog help` to list available hardware models. Only one watchdog can be enabled for a guest.

The following models may be available:

ib700

iBASE 700 is a very simple ISA watchdog with a single timer.

i6300esb

Intel 6300ESB I/O controller hub is a much more featureful PCI-based dual-timer watchdog.

diag288

A virtual watchdog for s390x backed by the diagnose 288 hypercall (currently KVM only).

−watchdog−action *action*

The *action* controls what QEMU will do when the watchdog timer expires. The default is `reset` (forcefully reset the guest). Other possible actions are: `shutdown` (attempt to gracefully shutdown the guest), `poweroff` (forcefully poweroff the guest), `inject-nmi` (inject a NMI into the guest), `pause` (pause the guest), `debug` (print a debug message and continue), or `none` (do nothing).

Note that the `shutdown` action requires that the guest responds to ACPI signals, which it may not be able to do in the sort of situations where the watchdog would have expired, and thus `-watchdog-action shutdown` is not recommended for production use.

Examples:

```
-watchdog i6300esb -watchdog-action pause
-watchdog ib700
```

−echr *numeric_ascii_value*

Change the escape character used for switching to the monitor when using monitor and serial sharing. The default is `0x01` when using the `-nographic` option. `0x01` is equal to pressing `Control-a`. You can select a different character from the ascii control keys where 1 through 26 map to Control-a through Control-z. For instance you could use the either of the following to change the escape character to Control-t.

```
-echr 0x14
-echr 20
```

−show−cursor

Show cursor.

−tb−size *n*

Set TB size.

−incoming tcp:[*host***]:***port***[,to=***maxport***][,ipv4][,ipv6]**
−incoming rdma:*host***:***port***[,ipv4][,ipv6]**

Prepare for incoming migration, listen on a given tcp port.

−incoming unix:*socketpath*

Prepare for incoming migration, listen on a given unix socket.

−incoming fd:*fd*

Accept incoming migration from a given filedescriptor.

−incoming exec:*cmdline*

Accept incoming migration as an output from specified external command.

−incoming defer

Wait for the URI to be specified via migrate_incoming. The monitor can be used to change settings (such as migration parameters) prior to issuing the migrate_incoming to allow the migration to begin.

−only−migratable
Only allow migratable devices. Devices will not be allowed to enter an unmigratable state.

−nodefaults
Don’t create default devices. Normally, QEMU sets the default devices like serial port, parallel port, virtual console, monitor device, VGA adapter, floppy and CD-ROM drive and others. The `-nodefaults` option will disable all those default devices.

−chroot *dir*
Immediately before starting guest execution, chroot to the specified directory. Especially useful in combination with −runas.

−runas *user*
Immediately before starting guest execution, drop root privileges, switching to the specified user.

−prom−env *variable***=***value*
Set OpenBIOS nvram *variable* to given *value* (PPC, SPARC only).

−semihosting
Enable semihosting mode (ARM, M68K, Xtensa, MIPS, Nios II only).

−semihosting−config [enable=on|off][,target=native|gdb|auto][,chardev=id][,arg=str[,...]]
Enable and configure semihosting (ARM, M68K, Xtensa, MIPS, Nios II only).

target=`native|gdb|auto`
Defines where the semihosting calls will be addressed, to QEMU (`native`) or to GDB (`gdb`). The default is `auto`, which means `gdb` during debug sessions and `native` otherwise.

chardev=*str1*
Send the output to a chardev backend output for native or auto output when not in gdb

arg=*str1***,arg=***str2***,...**
Allows the user to pass input arguments, and can be used multiple times to build up a list. The old-style `-kernel`/`-append` method of passing a command line is still supported for backward compatibility. If both the `--semihosting-config arg` and the `-kernel`/`-append` are specified, the former is passed to semihosting as it always takes precedence.

−old−param
Old param mode (ARM only).

−sandbox *arg***[,obsolete=***string***][,elevateprivileges=***string***][,spawn=***string***][,resourcecontrol=***string***]**
Enable Seccomp mode 2 system call filter. ’on’ will enable syscall filtering and ’off’ will disable it. The default is ’off’.

obsolete=*string*
Enable Obsolete system calls

elevateprivileges=*string*
Disable set*uid|gid system calls

spawn=*string*
Disable *fork and execve

resourcecontrol=*string*
Disable process affinity and schedular priority

−readconfig *file*
Read device configuration from *file*. This approach is useful when you want to spawn QEMU process with many command line options but you don’t want to exceed the command line character limit.

−writeconfig *file*
Write device configuration to *file*. The *file* can be either filename to save command line and device configuration into file or dash `-`) character to print the output to stdout. This can be later used as input file for `-readconfig` option.

−no−user−config

The `-no-user-config` option makes QEMU not load any of the user-provided config files on *sysconfdir*.

−trace [[enable=]*pattern***][,events=***file***][,file=***file***]**

Specify tracing options.

[enable=]*pattern*

Immediately enable events matching *pattern* (either event name or a globbing pattern). This option is only available if QEMU has been compiled with the *simple*, *log* or *ftrace* tracing backend. To specify multiple events or patterns, specify the **−trace** option multiple times.

Use `-trace help` to print a list of names of trace points.

events=*file*

Immediately enable events listed in *file*. The file must contain one event name (as listed in the *trace-events-all* file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *log* or *ftrace* tracing backend.

file=*file*

Log output traces to *file*. This option is only available if QEMU has been compiled with the *simple* tracing backend.

−plugin file=*file***[,arg=***string***]**

Load a plugin.

file=*file*

Load the given plugin from a shared library file.

arg=*string*

Argument string passed to the plugin. (Can be given multiple times.)

−enable−fips

Enable FIPS 140−2 compliance mode.

−msg timestamp[=on|off]

prepend a timestamp to each log message.(default:on)

−dump−vmstate *file*

Dump json-encoded vmstate information for current machine type to file in *file*

−enable−sync−profile

Enable synchronization profiling.

Generic object creation

−object *typename***[,***prop1***=***value1***,...]**

Create a new object of type *typename* setting properties in the order they are specified. Note that the 'id' property must be set. These objects are placed in the '/objects' path.

−object
memory−backend−file,id=*id***,size=***size***,mem−path=***dir***,share=***on*|*off***,discard−data=***on*|*off***,merge=***on*|*off***,dump=***on*|*off***,**
*nodes***,policy=***default*|*preferred*|*bind*|*interleave***,align=***align*

Creates a memory file backend object, which can be used to back the guest RAM with huge pages.

The **id** parameter is a unique ID that will be used to reference this memory region when configuring the **−numa** argument.

The **size** option provides the size of the memory region, and accepts common suffixes, eg **500M**.

The **mem-path** provides the path to either a shared memory or huge page filesystem mount.

The **share** boolean option determines whether the memory region is marked as private to QEMU, or shared. The latter allows a co-operating external process to access the QEMU memory region.

The **share** is also required for pvrdma devices due to limitations in the RDMA API provided by Linux.

Setting share=on might affect the ability to configure NUMA bindings for the memory backend under some circumstances, see Documentation/vm/numa_memory_policy.txt on the Linux kernel source tree for additional details.

Setting the **discard-data** boolean option to *on* indicates that file contents can be destroyed when QEMU exits, to avoid unnecessarily flushing data to the backing file. Note that **discard-data** is only an optimization, and QEMU might not discard file contents if it aborts unexpectedly or is terminated using SIGKILL.

The **merge** boolean option enables memory merge, also known as MADV_MERGEABLE, so that Kernel Samepage Merging will consider the pages for memory deduplication.

Setting the **dump** boolean option to *off* excludes the memory from core dumps. This feature is also known as MADV_DONTDUMP.

The **prealloc** boolean option enables memory preallocation.

The **host-nodes** option binds the memory range to a list of NUMA host nodes.

The **policy** option sets the NUMA policy to one of the following values:

default
default host policy

preferred
prefer the given host node list for allocation

bind
restrict memory allocation to the given host node list

interleave
interleave memory allocations across the given host node list

The **align** option specifies the base address alignment when QEMU **mmap**(2) **mem-path**, and accepts common suffixes, eg **2M**. Some backend store specified by **mem-path** requires an alignment different than the default one used by QEMU, eg the device DAX /dev/dax0.0 requires 2M alignment rather than 4K. In such cases, users can specify the required alignment via this option.

The **pmem** option specifies whether the backing file specified by **mem-path** is in host persistent memory that can be accessed using the SNIA NVM programming model (e.g. Intel NVDIMM). If **pmem** is set to 'on', QEMU will take necessary operations to guarantee the persistence of its own writes to **mem-path** (e.g. in vNVDIMM label emulation and live migration). Also, we will map the backend-file with MAP_SYNC flag, which ensures the file metadata is in sync for **mem-path** in case of host crash or a power failure. MAP_SYNC requires support from both the host kernel (since Linux kernel 4.15) and the filesystem of **mem-path** mounted with DAX option.

−object memory−backend−ram,id=*id***,merge=***on*|*off***,dump=***on*|*off***,share=***on*|*off***,prealloc=***on*|*off***,size=***size***,host−nodes=***host-nodes***,policy=***default*|*preferred*|*bind*|*interleave*

Creates a memory backend object, which can be used to back the guest RAM. Memory backend objects offer more control than the **−m** option that is traditionally used to define guest RAM. Please refer to **memory-backend-file** for a description of the options.

−object memory−backend−memfd,id=*id***,merge=***on*|*off***,dump=***on*|*off***,share=***on*|*off***,prealloc=***on*|*off***,size=***size***,host−nodes=***host-nodes***,policy=***default*|*preferred*|*bind*|*interleave***,seal=***on*|*off***,hugetlb=***on*|*off***,hugetlbsize=***size*

Creates an anonymous memory file backend object, which allows QEMU to share the memory with an external process (e.g. when using vhost-user). The memory is allocated with memfd and optional sealing. (Linux only)

The **seal** option creates a sealed-file, that will block further resizing the memory ('on' by default).

The **hugetlb** option specify the file to be created resides in the hugetlbfs filesystem (since Linux 4.14). Used in conjunction with the **hugetlb** option, the **hugetlbsize** option specify the hugetlb page size on systems that support multiple hugetlb page sizes (it must be a power of 2 value supported by the system).

In some versions of Linux, the **hugetlb** option is incompatible with the **seal** option (requires at least Linux 4.16).

Please refer to **memory-backend-file** for a description of the other options.

The **share** boolean option is *on* by default with memfd.

−object rng−builtin,id=*id*

Creates a random number generator backend which obtains entropy from QEMU builtin functions. The **id** parameter is a unique ID that will be used to reference this entropy backend from the **virtio-rng** device. By default, the **virtio-rng** device uses this RNG backend.

−object rng−random,id=*id***,filename=***/dev/random*

Creates a random number generator backend which obtains entropy from a device on the host. The **id** parameter is a unique ID that will be used to reference this entropy backend from the **virtio-rng** device. The **filename** parameter specifies which file to obtain entropy from and if omitted defaults to **/dev/urandom**.

−object rng−egd,id=*id***,chardev=***chardevid*

Creates a random number generator backend which obtains entropy from an external daemon running on the host. The **id** parameter is a unique ID that will be used to reference this entropy backend from the **virtio-rng** device. The **chardev** parameter is the unique ID of a character device backend that provides the connection to the RNG daemon.

−object tls−creds−anon,id=*id***,endpoint=***endpoint***,dir=***/path/to/cred/dir***,verify−peer=***on***|***off*

Creates a TLS anonymous credentials object, which can be used to provide TLS support on network backends. The **id** parameter is a unique ID which network backends will use to access the credentials. The **endpoint** is either **server** or **client** depending on whether the QEMU network backend that uses the credentials will be acting as a client or as a server. If **verify-peer** is enabled (the default) then once the handshake is completed, the peer credentials will be verified, though this is a no-op for anonymous credentials.

The *dir* parameter tells QEMU where to find the credential files. For server endpoints, this directory may contain a file *dh−params.pem* providing diffie-hellman parameters to use for the TLS server. If the file is missing, QEMU will generate a set of DH parameters at startup. This is a computationally expensive operation that consumes random pool entropy, so it is recommended that a persistent set of parameters be generated upfront and saved.

−object tls−creds−psk,id=*id***,endpoint=***endpoint***,dir=***/path/to/keys/dir***[,username=***username***]**

Creates a TLS Pre-Shared Keys (PSK) credentials object, which can be used to provide TLS support on network backends. The **id** parameter is a unique ID which network backends will use to access the credentials. The **endpoint** is either **server** or **client** depending on whether the QEMU network backend that uses the credentials will be acting as a client or as a server. For clients only, **username** is the username which will be sent to the server. If omitted it defaults to ‘‘qemu’’.

The *dir* parameter tells QEMU where to find the keys file. It is called "*dir*/keys.psk‘‘ and contains ’’username:key" pairs. This file can most easily be created using the GnuTLS `psktool` program.

For server endpoints, *dir* may also contain a file *dh−params.pem* providing diffie-hellman parameters to use for the TLS server. If the file is missing, QEMU will generate a set of DH parameters at startup. This is a computationally expensive operation that consumes random pool entropy, so it is recommended that a persistent set of parameters be generated up front and saved.

−object tls−creds−x509,id=*id***,endpoint=***endpoint***,dir=***/path/to/cred/dir***,priority=***priority***,verify−peer=***on*|*off***,passwordid=***id*

Creates a TLS anonymous credentials object, which can be used to provide TLS support on network backends. The **id** parameter is a unique ID which network backends will use to access the credentials. The **endpoint** is either **server** or **client** depending on whether the QEMU network backend that uses the credentials will be acting as a client or as a server. If **verify-peer** is enabled (the default) then once the handshake is completed, the peer credentials will be verified. With x509 certificates, this implies that the clients must be provided with valid client certificates too.

The *dir* parameter tells QEMU where to find the credential files. For server endpoints, this directory may contain a file *dh−params.pem* providing diffie-hellman parameters to use for the TLS server. If the file is missing, QEMU will generate a set of DH parameters at startup. This is a computationally expensive operation that consumes random pool entropy, so it is recommended that a persistent set of parameters be generated upfront and saved.

For x509 certificate credentials the directory will contain further files providing the x509 certificates. The certificates must be stored in PEM format, in filenames *ca−cert.pem*, *ca−crl.pem* (optional), *server−cert.pem* (only servers), *server−key.pem* (only servers), *client−cert.pem* (only clients), and *client−key.pem* (only clients).

For the *server−key.pem* and *client−key.pem* files which contain sensitive private keys, it is possible to use an encrypted version by providing the *passwordid* parameter. This provides the ID of a previously created `secret` object containing the password for decryption.

The *priority* parameter allows to override the global default priority used by gnutls. This can be useful if the system administrator needs to use a weaker set of crypto priorities for QEMU without potentially forcing the weakness onto all applications. Or conversely if one wants wants a stronger default for QEMU than for all other applications, they can do this through this parameter. Its format is a gnutls priority string as described at **<https://gnutls.org/manual/html_node/Priority−Strings.html>**.

−object filter−buffer,id=*id***,netdev=***netdevid***,interval=***t***[,queue=***all*|*rx*|*tx***][,status=***on*|*off***]**

Interval *t* can't be 0, this filter batches the packet delivery: all packets arriving in a given interval on netdev *netdevid* are delayed until the end of the interval. Interval is in microseconds. **status** is optional that indicate whether the netfilter is on (enabled) or off (disabled), the default status for netfilter will be 'on'.

queue *all*|*rx*|*tx* is an option that can be applied to any netfilter.

all: the filter is attached both to the receive and the transmit queue of the netdev (default).

rx: the filter is attached to the receive queue of the netdev, where it will receive packets sent to the netdev.

tx: the filter is attached to the transmit queue of the netdev, where it will receive packets sent by the netdev.

−object filter−mirror,id=*id***,netdev=***netdevid***,outdev=***chardevid***,queue=***all*|*rx*|*tx***[,vnet_hdr_support]**

filter-mirror on netdev *netdevid*,mirror net packet to chardev*chardevid*, if it has the vnet_hdr_support flag, filter-mirror will mirror packet with vnet_hdr_len.

−object filter−redirector,id=*id***,netdev=***netdevid***,indev=***chardevid***,outdev=***chardevid***,queue=***all*|*rx*|*tx***[,vnet_hdr_support]**

filter-redirector on netdev *netdevid*,redirect filter's net packet to chardev *chardevid*,and redirect indev's packet to filter.if it has the vnet_hdr_support flag, filter-redirector will redirect packet with vnet_hdr_len. Create a filter-redirector we need to differ outdev id from indev id, id can not be the same. we can just use indev or outdev, but at least one of indev or outdev need to be specified.

−object filter−rewriter,id=*id***,netdev=***netdevid***,queue=***all*|*rx*|*tx***,[vnet_hdr_support]**

Filter-rewriter is a part of COLO project.It will rewrite tcp packet to secondary from primary to keep secondary tcp connection,and rewrite tcp packet to primary from secondary make tcp packet can be handled by client.if it has the vnet_hdr_support flag, we can parse packet with vnet header.

usage: colo secondary: −object filter−redirector,id=f1,netdev=hn0,queue=tx,indev=red0 −object filter−redirector,id=f2,netdev=hn0,queue=rx,outdev=red1 −object filter−rewriter,id=rew0,netdev=hn0,queue=all

−object filter−dump,id=*id*,netdev=*dev*[,file=*filename*][,maxlen=*len*]

Dump the network traffic on netdev *dev* to the file specified by *filename*. At most *len* bytes (64k by default) per packet are stored. The file format is libpcap, so it can be analyzed with tools such as tcpdump or Wireshark.

**−object
colo−compare,id=*id*,primary_in=*chardevid*,secondary_in=*chardevid*,outdev=*chardevid*,iothread=*id*[,vnet_hdr_supp**

Colo-compare gets packet from primary_in*chardevid* and secondary_in*chardevid*, than compare primary packet with secondary packet. If the packets are same, we will output primary packet to outdev*chardevid*, else we will notify colo-frame do checkpoint and send primary packet to outdev*chardevid*. In order to improve efficiency, we need to put the task of comparison in another thread. If it has the vnet_hdr_support flag, colo compare will send/recv packet with vnet_hdr_len. If you want to use Xen COLO, will need the notify_dev to notify Xen colo-frame to do checkpoint.

we must use it with the help of filter-mirror and filter-redirector.

```
KVM COLO

primary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,downscript=/etc/qem
-device e1000,id=e0,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=mirror0,host=3.3.3.3,port=9003,server,nowait
-chardev socket,id=compare1,host=3.3.3.3,port=9004,server,nowait
-chardev socket,id=compare0,host=3.3.3.3,port=9001,server,nowait
-chardev socket,id=compare0-0,host=3.3.3.3,port=9001
-chardev socket,id=compare_out,host=3.3.3.3,port=9005,server,nowait
-chardev socket,id=compare_out0,host=3.3.3.3,port=9005
-object iothread,id=iothread1
-object filter-mirror,id=m0,netdev=hn0,queue=tx,outdev=mirror0
-object filter-redirector,netdev=hn0,id=redire0,queue=rx,indev=compare
-object filter-redirector,netdev=hn0,id=redire1,queue=rx,outdev=compa
-object colo-compare,id=comp0,primary_in=compare0-0,secondary_in=compa

secondary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,down script=/etc/qe
-device e1000,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=red0,host=3.3.3.3,port=9003
-chardev socket,id=red1,host=3.3.3.3,port=9004
-object filter-redirector,id=f1,netdev=hn0,queue=tx,indev=red0
-object filter-redirector,id=f2,netdev=hn0,queue=rx,outdev=red1


Xen COLO

primary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,downscript=/etc/qem
-device e1000,id=e0,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=mirror0,host=3.3.3.3,port=9003,server,nowait
-chardev socket,id=compare1,host=3.3.3.3,port=9004,server,nowait
-chardev socket,id=compare0,host=3.3.3.3,port=9001,server,nowait
```

```
-chardev socket,id=compare0-0,host=3.3.3.3,port=9001
-chardev socket,id=compare_out,host=3.3.3.3,port=9005,server,nowait
-chardev socket,id=compare_out0,host=3.3.3.3,port=9005
-chardev socket,id=notify_way,host=3.3.3.3,port=9009,server,nowait
-object filter-mirror,id=m0,netdev=hn0,queue=tx,outdev=mirror0
-object filter-redirector,netdev=hn0,id=redire0,queue=rx,indev=compare
-object filter-redirector,netdev=hn0,id=redire1,queue=rx,outdev=compar
-object iothread,id=iothread1
-object colo-compare,id=comp0,primary_in=compare0-0,secondary_in=compa

secondary:
-netdev tap,id=hn0,vhost=off,script=/etc/qemu-ifup,down script=/etc/qe
-device e1000,netdev=hn0,mac=52:a4:00:12:78:66
-chardev socket,id=red0,host=3.3.3.3,port=9003
-chardev socket,id=red1,host=3.3.3.3,port=9004
-object filter-redirector,id=f1,netdev=hn0,queue=tx,indev=red0
-object filter-redirector,id=f2,netdev=hn0,queue=rx,outdev=red1
```

If you want to know the detail of above command line, you can read the colo-compare git log.

−object cryptodev−backend−builtin,id=*id***[,queues=***queues***]**

Creates a cryptodev backend which executes crypto opreation from the QEMU cipher APIS. The *id* parameter is a unique ID that will be used to reference this cryptodev backend from the **virtio-crypto** device. The *queues* parameter is optional, which specify the queue number of cryptodev backend, the default of *queues* is 1.

```
# qemu-system-x86_64 \
[...] \
-object cryptodev-backend-builtin,id=cryptodev0 \
-device virtio-crypto-pci,id=crypto0,cryptodev=cryptodev0 \
[...]
```

−object cryptodev−vhost−user,id=*id***,chardev=***chardevid***[,queues=***queues***]**

Creates a vhost-user cryptodev backend, backed by a chardev *chardevid*. The *id* parameter is a unique ID that will be used to reference this cryptodev backend from the **virtio-crypto** device. The chardev should be a unix domain socket backed one. The vhost-user uses a specifically defined protocol to pass vhost ioctl replacement messages to an application on the other end of the socket. The *queues* parameter is optional, which specify the queue number of cryptodev backend for multiqueue vhost-user, the default of *queues* is 1.

```
# qemu-system-x86_64 \
[...] \
-chardev socket,id=chardev0,path=/path/to/socket \
-object cryptodev-vhost-user,id=cryptodev0,chardev=chardev0 \
-device virtio-crypto-pci,id=crypto0,cryptodev=cryptodev0 \
[...]
```

−object secret,id=*id***,data=***string***,format=***raw***|***base64***[,keyid=***secretid***,iv=***string***]**
−object secret,id=*id***,file=***filename***,format=***raw***|***base64***[,keyid=***secretid***,iv=***string***]**

Defines a secret to store a password, encryption key, or some other sensitive data. The sensitive data can either be passed directly via the *data* parameter, or indirectly via the *file* parameter. Using the *data* parameter is insecure unless the sensitive data is encrypted.

The sensitive data can be provided in raw format (the default), or base64. When encoded as JSON, the raw format only supports valid UTF−8 characters, so base64 is recommended for sending binary data. QEMU will convert from which ever format is provided to the format it needs internally. eg, an RBD password can be provided in raw format, even though it will be base64 encoded when passed onto the RBD sever.

For added protection, it is possible to encrypt the data associated with a secret using the AES−256−CBC cipher. Use of encryption is indicated by providing the *keyid* and *iv* parameters. The *keyid* parameter provides the ID of a previously defined secret that contains the AES−256 decryption key. This key should be 32−bytes long and be base64 encoded. The *iv* parameter provides the random initialization vector used for encryption of this particular secret and should be a base64 encrypted string of the 16−byte IV.

The simplest (insecure) usage is to provide the secret inline

```
# qemu-system-x86_64 -object secret,id=sec0,data=letmein,format=raw
```

The simplest secure usage is to provide the secret via a file

# printf "letmein" > mypasswd.txt # qemu−system−x86_64 −object secret,id=sec0,file=mypasswd.txt,format=raw

For greater security, AES−256−CBC should be used. To illustrate usage, consider the openssl command line tool which can encrypt the data. Note that when encrypting, the plaintext must be padded to the cipher block size (32 bytes) using the standard PKCS#5/6 compatible padding algorithm.

First a master key needs to be created in base64 encoding:

```
# openssl rand -base64 32 > key.b64
# KEY=$(base64 -d key.b64 | hexdump  -v -e '/1 "%02X"')
```

Each secret to be encrypted needs to have a random initialization vector generated. These do not need to be kept secret

```
# openssl rand -base64 16 > iv.b64
# IV=$(base64 -d iv.b64 | hexdump  -v -e '/1 "%02X"')
```

The secret to be defined can now be encrypted, in this case we're telling openssl to base64 encode the result, but it could be left as raw bytes if desired.

```
# SECRET=$(printf "letmein" |
openssl enc -aes-256-cbc -a -K $KEY -iv $IV)
```

When launching QEMU, create a master secret pointing to `key.b64` and specify that to be used to decrypt the user password. Pass the contents of `iv.b64` to the second secret

```
# qemu-system-x86_64 \
-object secret,id=secmaster0,format=base64,file=key.b64 \
-object secret,id=sec0,keyid=secmaster0,format=base64,\
data=$SECRET,iv=$(<iv.b64)
```

−object sev−guest,id=*id***,cbitpos=***cbitpos***,reduced−phys−bits=***val***,[sev−device=***string***,policy=***policy***,handle=***handle***,dh−cert−fi**

Create a Secure Encrypted Virtualization (SEV) guest object, which can be used to provide the guest memory encryption support on AMD processors.

When memory encryption is enabled, one of the physical address bit (aka the C−bit) is utilized to mark if a memory page is protected. The **cbitpos** is used to provide the C−bit position. The C−bit position is Host family dependent hence user must provide this value. On EPYC, the value should be 47.

When memory encryption is enabled, we loose certain bits in physical address space. The **reduced-phys-bits** is used to provide the number of bits we loose in physical address space. Similar to C−bit, the value is Host family dependent. On EPYC, the value should be 5.

The **sev-device** provides the device file to use for communicating with the SEV firmware running inside AMD Secure Processor. The default device is '/dev/sev'. If hardware supports memory encryption then /dev/sev devices are created by CCP driver.

The **policy** provides the guest policy to be enforced by the SEV firmware and restrict what configuration and operational commands can be performed on this guest by the hypervisor. The policy should be provided by the guest owner and is bound to the guest and cannot be changed throughout the lifetime of the guest. The default is 0.

If guest **policy** allows sharing the key with another SEV guest then **handle** can be use to provide handle of the guest from which to share the key.

The **dh-cert-file** and **session-file** provides the guest owner's Public Diffie-Hillman key defined in SEV spec. The PDH and session parameters are used for establishing a cryptographic session with the guest owner to negotiate keys used for attestation. The file must be encoded in base64.

e.g to launch a SEV guest

```
# qemu-system-x86_64 \
......
-object sev-guest,id=sev0,cbitpos=47,reduced-phys-bits=5 \
-machine ...,memory-encryption=sev0
.....
```

–object authz–simple,id=*id***,identity=***string*

Create an authorization object that will control access to network services.

The **identity** parameter is identifies the user and its format depends on the network service that authorization object is associated with. For authorizing based on TLS x509 certificates, the identity must be the x509 distinguished name. Note that care must be taken to escape any commas in the distinguished name.

An example authorization object to validate a x509 distinguished name would look like:

```
# qemu-system-x86_64 \
...
-object 'authz-simple,id=auth0,identity=CN=laptop.example.com,,O=Examp
...
```

Note the use of quotes due to the x509 distinguished name containing whitespace, and escaping of ','.

–object authz–listfile,id=*id***,filename=***path***,refresh=***yes*|*no*

Create an authorization object that will control access to network services.

The **filename** parameter is the fully qualified path to a file containing the access control list rules in JSON format.

An example set of rules that match against SASL usernames might look like:

```
{
"rules": [
{ "match": "fred", "policy": "allow", "format": "exact" },
{ "match": "bob", "policy": "allow", "format": "exact" },
{ "match": "danb", "policy": "deny", "format": "glob" },
{ "match": "dan*", "policy": "allow", "format": "exact" },
],
"policy": "deny"
}
```

When checking access the object will iterate over all the rules and the first rule to match will have its **policy** value returned as the result. If no rules match, then the default **policy** value is returned.

The rules can either be an exact string match, or they can use the simple UNIX glob pattern matching to allow wildcards to be used.

If **refresh** is set to true the file will be monitored and automatically reloaded whenever its content changes.

As with the `authz-simple` object, the format of the identity strings being matched depends on the network service, but is usually a TLS x509 distinguished name, or a SASL username.

An example authorization object to validate a SASL username would look like:

```
# qemu-system-x86_64 \
...
-object authz-simple,id=auth0,filename=/etc/qemu/vnc-sasl.acl,refresh=
...
```

–object authz–pam,id=*id***,service=***string*

Create an authorization object that will control access to network services.

The **service** parameter provides the name of a PAM service to use for authorization. It requires that a file `/etc/pam.d/`*`service`* exist to provide the configuration for the `account` subsystem.

An example authorization object to validate a TLS x509 distinguished name would look like:

```
# qemu-system-x86_64 \
...
-object authz-pam,id=auth0,service=qemu-vnc
...
```

There would then be a corresponding config file for PAM at `/etc/pam.d/qemu-vnc` that contains:

```
account requisite  pam_listfile.so item=user sense=allow \
file=/etc/qemu/vnc.allow
```

Finally the `/etc/qemu/vnc.allow` file would contain the list of x509 distingished names that are permitted access

```
CN=laptop.example.com,O=Example Home,L=London,ST=London,C=GB
```

During the graphical emulation, you can use special key combinations to change modes. The default key mappings are shown below, but if you use `-alt-grab` then the modifier is Ctrl-Alt-Shift (instead of Ctrl-Alt) and if you use `-ctrl-grab` then the modifier is the right Ctrl key (instead of Ctrl-Alt):

Ctrl-Alt-f

Toggle full screen

Ctrl–Alt–+

Enlarge the screen

Ctrl–Alt —

Shrink the screen

Ctrl-Alt-u

Restore the screen's un-scaled dimensions

Ctrl-Alt-n

Switch to virtual console 'n'. Standard console mappings are:

1 Target system display

2 Monitor

3 Serial port

Ctrl-Alt

Toggle mouse and keyboard grab.

In the virtual consoles, you can use **Ctrl-Up**, **Ctrl-Down**, **Ctrl-PageUp** and **Ctrl-PageDown** to move in the back log.

During emulation, if you are using a character backend multiplexer (which is the default if you are using **−nographic**) then several commands are available via an escape sequence. These key sequences all start with an escape character, which is **Ctrl-a** by default, but can be changed with **−echr**. The list below assumes you're using the default.

Ctrl-a h
Print this help

Ctrl-a x
Exit emulator

Ctrl-a s
Save disk data back to file (if −snapshot)

Ctrl-a t
Toggle console timestamps

Ctrl-a b
Send break (magic sysrq in Linux)

Ctrl-a c
Rotate between the frontends connected to the multiplexer (usually this switches between the monitor and the console)

Ctrl-a Ctrl-a
Send the escape character to the frontend

The following options are specific to the PowerPC emulation:

−g ***W*x*H*[x*DEPTH*]**
Set the initial VGA graphic mode. The default is 800x600x32.

−prom−env *string*
Set OpenBIOS variables in NVRAM, for example:

```
        qemu-system-ppc -prom-env 'auto-boot?=false' \
         -prom-env 'boot-device=hd:2,\yaboot' \
         -prom-env 'boot-args=conf=hd:2,\yaboot.conf'
```

These variables are not used by Open Hack'Ware.

The following options are specific to the Sparc32 emulation:

−g ***W*x*H*x[x*DEPTH*]**
Set the initial graphics mode. For TCX, the default is 1024x768x8 with the option of 1024x768x24. For cgthree, the default is 1024x768x8 with the option of 1152x900x8 for people who wish to use OBP.

−prom−env *string*
Set OpenBIOS variables in NVRAM, for example:

```
        qemu-system-sparc -prom-env 'auto-boot?=false' \
         -prom-env 'boot-device=sd(0,2,0):d' -prom-env 'boot-args=linux single'
```

−M [SS−4|SS−5|SS−10|SS−20|SS−600MP|LX|Voyager|SPARCClassic] [|SPARCbook]
Set the emulated machine type. Default is SS−5.

The following options are specific to the Sparc64 emulation:

−prom−env *string*
Set OpenBIOS variables in NVRAM, for example:

```
        qemu-system-sparc64 -prom-env 'auto-boot?=false'
```

−M [sun4u|sun4v|niagara]
Set the emulated machine type. The default is sun4u.

The following options are specific to the ARM emulation:

−semihosting

Enable semihosting syscall emulation.

On ARM this implements the ‘‘Angel’’ interface.

Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

The following options are specific to the ColdFire emulation:

−semihosting

Enable semihosting syscall emulation.

On M68K this implements the ‘‘ColdFire GDB’’ interface used by libgloss.

Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

The following options are specific to the Xtensa emulation:

−semihosting

Enable semihosting syscall emulation.

Xtensa semihosting provides basic file IO calls, such as open/read/write/seek/select. Tensilica baremetal libc for ISS and linux platform ‘‘sim’’ use this interface.

Note that this allows guest direct access to the host filesystem, so should only be used with trusted guest OS.

NOTES

In addition to using normal file images for the emulated storage devices, QEMU can also use networked resources such as iSCSI devices. These are specified using a special URL syntax.

iSCSI

iSCSI support allows QEMU to access iSCSI resources directly and use as images for the guest storage. Both disk and cdrom images are supported.

Syntax for specifying iSCSI LUNs is ‘‘iscsi://<target−ip>[:<port>]/<target−iqn>/<lun>’’

By default qemu will use the iSCSI initiator-name 'iqn.2008−11.org.linux−kvm[:<name>]' but this can also be set from the command line or a configuration file.

Since version Qemu 2.4 it is possible to specify a iSCSI request timeout to detect stalled requests and force a reestablishment of the session. The timeout is specified in seconds. The default is 0 which means no timeout. Libiscsi 1.15.0 or greater is required for this feature.

Example (without authentication):

```
qemu−system−x86_64 −iscsi initiator−name=iqn.2001−04.com.example:my−initi
                   −cdrom iscsi://192.0.2.1/iqn.2001−04.com.example/2 \
                   −drive file=iscsi://192.0.2.1/iqn.2001−04.com.example/1
```

Example (CHAP username/password via URL):

```
qemu−system−x86_64 −drive file=iscsi://user%password@192.0.2.1/iqn.2001−0
```

Example (CHAP username/password via environment variables):

```
LIBISCSI_CHAP_USERNAME="user" \
LIBISCSI_CHAP_PASSWORD="password" \
qemu−system−x86_64 −drive file=iscsi://192.0.2.1/iqn.2001−04.com.example/
```

NBD

QEMU supports NBD (Network Block Devices) both using TCP protocol as well as Unix Domain Sockets. With TCP, the default port is 10809.

Syntax for specifying a NBD device using TCP, in preferred URI form: "nbd://<server−ip>[:<port>]/[<export>]"

Syntax for specifying a NBD device using Unix Domain Sockets; remember that '?' is a shell glob character and may need quoting: "nbd+unix:///[<export>]?socket=<domain−socket>"

Older syntax that is also recognized: "nbd:<server−ip>:<port>[:exportname=<export>]"

Syntax for specifying a NBD device using Unix Domain Sockets "nbd:unix:<domain−socket>[:exportname=<export>]"

Example for TCP

```
        qemu-system-x86_64 --drive file=nbd:192.0.2.1:30000
```

Example for Unix Domain Sockets

```
        qemu-system-x86_64 --drive file=nbd:unix:/tmp/nbd-socket
```

SSH

QEMU supports SSH (Secure Shell) access to remote disks.

Examples:

```
        qemu-system-x86_64 -drive file=ssh://user@host/path/to/disk.img
        qemu-system-x86_64 -drive file.driver=ssh,file.user=user,file.host=host,f
```

Currently authentication must be done using ssh-agent. Other authentication methods may be supported in future.

Sheepdog

Sheepdog is a distributed storage system for QEMU. QEMU supports using either local sheepdog devices or remote networked devices.

Syntax for specifying a sheepdog device

```
        sheepdog[+tcp|+unix]://[host:port]/vdiname[?socket=path][#snapid|#tag]
```

Example

```
        qemu-system-x86_64 --drive file=sheepdog://192.0.2.1:30000/MyVirtualMachi
```

See also **<https://sheepdog.github.io/sheepdog/>**.

GlusterFS

GlusterFS is a user space distributed file system. QEMU supports the use of GlusterFS volumes for hosting VM disk images using TCP, Unix Domain Sockets and RDMA transport protocols.

Syntax for specifying a VM disk image on GlusterFS volume is

```
        URI:
        gluster[+type]://[host[:port]]/volume/path[?socket=...][,debug=N][,logfil

        JSON:
        'json:{"driver":"qcow2","file":{"driver":"gluster","volume":"testvol","pa
                                         "server":[{"type":"tcp","host":"...","po
                                                  {"type":"unix","socket":"..."}
```

Example

```
        URI:
        qemu-system-x86_64 --drive file=gluster://192.0.2.1/testvol/a.img,
                                        file.debug=9,file.logfile=/var/log/qemu-gl

        JSON:
        qemu-system-x86_64 'json:{"driver":"qcow2",
                                  "file":{"driver":"gluster",
                                           "volume":"testvol","path":"a.img",
                                           "debug":9,"logfile":"/var/log/qemu-glu
                                           "server":[{"type":"tcp","host":"1.2.3.
                                                     {"type":"unix","socket":"/va
        qemu-system-x86_64 -drive driver=qcow2,file.driver=gluster,file.volume=te
                                              file.debug=9,file.logfile=/var/log/
                                              file.server.0.type=tcp,file.server.
                                              file.server.1.type=unix,file.server
```

See also **<http://www.gluster.org>**.

HTTP/HTTPS/FTP/FTPS

QEMU supports read-only access to files accessed over http(s) and ftp(s).

Syntax using a single filename:

```
<protocol>://[<username>[:<password>]@]<host>/<path>
```

where:

protocol
’http’, ’https’, ’ftp’, or ’ftps’.

username
Optional username for authentication to the remote server.

password
Optional password for authentication to the remote server.

host
Address of the remote server.

path
Path on the remote server, including any query string.

The following options are also supported:

url The full URL when passing options to the driver explicitly.

readahead
The amount of data to read ahead with each range request to the remote server. This value may optionally have the suffix ’T’, ’G’, ’M’, ’K’, ’k’ or ’b’. If it does not have a suffix, it will be assumed to be in bytes. The value must be a multiple of 512 bytes. It defaults to 256k.

sslverify
Whether to verify the remote server’s certificate when connecting over SSL. It can have the value ’on’ or ’off’. It defaults to ’on’.

cookie
Send this cookie (it can also be a list of cookies separated by ’;’) with each outgoing request. Only supported when using protocols such as HTTP which support cookies, otherwise ignored.

timeout
Set the timeout in seconds of the CURL connection. This timeout is the time that CURL waits for a response from the remote server to get the size of the image to be downloaded. If not set, the default timeout of 5 seconds is used.

Note that when passing options to qemu explicitly, **driver** is the value of <protocol>.

Example: boot from a remote Fedora 20 live ISO image

```
qemu-system-x86_64 --drive media=cdrom,file=https://archives.fedoraprojec

qemu-system-x86_64 --drive media=cdrom,file.driver=http,file.url=http://a
```

Example: boot from a remote Fedora 20 cloud image using a local overlay for writes, copy-on-read, and a readahead of 64k

```
        qemu-img create -f qcow2 -o backing_file='json:{"file.driver":"http",, "f

        qemu-system-x86_64 -drive file=/tmp/Fedora-x86_64-20-20131211.1-sda.qcow2
```

Example: boot from an image stored on a VMware vSphere server with a self-signed certificate using a local overlay for writes, a readahead of 64k and a timeout of 10 seconds.

```
        qemu-img create -f qcow2 -o backing_file='json:{"file.driver":"https",, "

        qemu-system-x86_64 -drive file=/tmp/test.qcow2
```

SEE ALSO

The HTML documentation of QEMU for more precise information and Linux user mode emulator invocation.

AUTHOR

Fabrice Bellard