QEMU Emulator 2.7.50
QAPI Reference Manual

Table of Contents

1 Introduction

The QEMU Machine Protocol (QMP) allows applications to operate a QEMU instance.

QMP is JSON (`http://www.json.org`) based and features the following:

- Lightweight, text-based, easy to parse data format
- Asynchronous messages support (ie. events)
- Capabilities Negotiation

For detailed information on QEMU Machine Protocol, the specification is in `qmp-spec.txt`.

1.1 Usage

You can use the `-qmp` option to enable QMP. For example, the following makes QMP available on localhost port 4444:

```
$ qemu [...] -qmp tcp:localhost:4444,server,nowait
```

However, for more flexibility and to make use of more options, the `-mon` command-line option should be used. For instance, the following example creates one HMP instance (human monitor) on stdio and one QMP instance on localhost port 4444:

```
$ qemu [...] -chardev stdio,id=mon0 -mon chardev=mon0,mode=readline \
             -chardev socket,id=mon1,host=localhost,port=4444,server,nowait \
             -mon chardev=mon1,mode=control,pretty=on
```

Please, refer to QEMU's manpage for more information.

1.2 Simple testing

To manually test QMP one can connect with telnet and issue commands by hand:

```
$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
{
    "QMP": {
        "version": {
            "qemu": {
                "micro": 50,
                "minor": 6,
                "major": 1
            },
            "package": ""
        },
        "capabilities": [
        ]
    }
}

{ "execute": "qmp_capabilities" }
```

```
{
    "return": {
    }
}

{ "execute": "query-status" }
{
    "return": {
        "status": "prelaunch",
        "singlestep": false,
        "running": false
    }
}
```

1.3 Wiki

Please refer to the QMP QEMU wiki page (`http://wiki.qemu-project.org/QMP`) for more details on QMP.

2 API Reference

2.1 Introduction

This document describes all commands currently supported by QMP.

Most of the time their usage is exactly the same as in the user Monitor, this means that any other document which also describe commands (the manpage, QEMU's manual, etc) can and should be consulted.

QMP has two types of commands: regular and query commands. Regular commands usually change the Virtual Machine's state someway, while query commands just return information. The sections below are divided accordingly.

It's important to observe that all communication examples are formatted in a reader-friendly way, so that they're easier to understand. However, in real protocol usage, they're emitted as a single line.

Also, the following notation is used to denote data flow:

Example:

```
-> data issued by the Client
<- Server data response
```

Please, refer to the QMP specification (QMP/qmp-spec.txt) for detailed information on the Server command and response formats.

2.2 Stability Considerations

The current QMP command set (described in this file) may be useful for a number of use cases, however it's limited and several commands have bad defined semantics, specially with regard to command completion.

These problems are going to be solved incrementally in the next QEMU releases and we're going to establish a deprecation policy for badly defined commands.

If you're planning to adopt QMP, please observe the following:

1. The deprecation policy will take effect and be documented soon, please check the documentation of each used command as soon as a new release of QEMU is available
2. DO NOT rely on anything which is not explicit documented
3. Errors, in special, are not documented. Applications should NOT check for specific errors classes or data (it's strongly recommended to only check for the `"error"` key)

2.3 QAPI common definitions

`QapiErrorClass` [Enum]

'`GenericError`'
: this is used for errors that don't require a specific error class. This should be the default case for most errors

'`CommandNotFound`'
: the requested command has not been found

‘`DeviceEncrypted`’
: the requested operation can't be fulfilled because the selected device is encrypted

‘`DeviceNotActive`’
: a device has failed to be become active

‘`DeviceNotFound`’
: the requested device has not been found

‘`KVMMissingCap`’
: the requested operation can't be fulfilled because a required KVM capability is missing

QEMU error classes

Since: 1.2

`VersionTriple` *{ 'major': `int`, 'minor': `int`, 'micro': `int` }* [Struct]

major The major version number.

minor The minor version number.

micro The micro version number.

A three-part version number.

Since: 2.4

`VersionInfo` *{ 'qemu': `VersionTriple`, 'package': `str` }* [Struct]

qemu The version of QEMU. By current convention, a micro version of 50 signifies a development branch. A micro version greater than or equal to 90 signifies a release candidate for the next minor version. A micro version of less than 50 signifies a stable release.

package QEMU will always set this field to an empty string. Downstream versions of QEMU should set this to a non-empty string. The exact format depends on the downstream however it highly recommended that a unique name is used.

A description of QEMU's version.

Since: 0.14.0

`VersionInfo query-version ()` [Command]

Returns the current version of QEMU.

Returns: A *VersionInfo* object describing the current version of QEMU.

Since: 0.14.0

Example:

```
-> { "execute": "query-version" }
<- {
      "return":{
         "qemu":{
            "major":0,
```

```
                "minor":11,
                "micro":5
             },
             "package":""
          }
       }
```

`CommandInfo` *{ 'name': `str` }* [Struct]

name The command name

Information about a QMP command

Since: 0.14.0

`['CommandInfo'] query-commands ()` [Command]

Return a list of supported QMP commands by this server

Returns: A list of *CommandInfo* for all supported commands

Note: This example has been shortened as the real response is too long.

Since: 0.14.0

Example:

```
-> { "execute": "query-commands" }
<- {
     "return":[
        {
           "name":"query-balloon"
        },
        {
           "name":"system_powerdown"
        }
     ]
   }
```

`OnOffAuto` [Enum]

'`auto`' QEMU selects the value between on and off

'`on`' Enabled

'`off`' Disabled

An enumeration of three options: on, off, and auto

Since: 2.2

`OnOffSplit` [Enum]

'`on`' Enabled

'`off`' Disabled

'`split`' Mixed

An enumeration of three values: on, off, and split

Since: 2.6

2.4 QAPI crypto definitions

`QCryptoTLSCredsEndpoint` [Enum]

'`client`' the network endpoint is acting as the client

'`server`' the network endpoint is acting as the server

The type of network endpoint that will be using the credentials. Most types of credential require different setup / structures depending on whether they will be used in a server versus a client.

Since: 2.5

`QCryptoSecretFormat` [Enum]

'`raw`' raw bytes. When encoded in JSON only valid UTF-8 sequences can be used

'`base64`' arbitrary base64 encoded binary data

The data format that the secret is provided in

Since: 2.6

`QCryptoHashAlgorithm` [Enum]

'`md5`' MD5. Should not be used in any new code, legacy compat only

'`sha1`' SHA-1. Should not be used in any new code, legacy compat only

'`sha224`' SHA-224. (since 2.7)

'`sha256`' SHA-256. Current recommended strong hash.

'`sha384`' SHA-384. (since 2.7)

'`sha512`' SHA-512. (since 2.7)

'`ripemd160`'
RIPEMD-160. (since 2.7)

The supported algorithms for computing content digests

Since: 2.6

`QCryptoCipherAlgorithm` [Enum]

'`aes-128`' AES with 128 bit / 16 byte keys

'`aes-192`' AES with 192 bit / 24 byte keys

'`aes-256`' AES with 256 bit / 32 byte keys

'`des-rfb`' RFB specific variant of single DES. Do not use except in VNC.

'`cast5-128`'
Cast5 with 128 bit / 16 byte keys

'`serpent-128`'
Serpent with 128 bit / 16 byte keys

‘`serpent-192`’
: Serpent with 192 bit / 24 byte keys

‘`serpent-256`’
: Serpent with 256 bit / 32 byte keys

‘`twofish-128`’
: Twofish with 128 bit / 16 byte keys

‘`twofish-192`’
: Twofish with 192 bit / 24 byte keys

‘`twofish-256`’
: Twofish with 256 bit / 32 byte keys

The supported algorithms for content encryption ciphers

Since: 2.6

`QCryptoCipherMode` [Enum]

‘`ecb`’ Electronic Code Book

‘`cbc`’ Cipher Block Chaining

‘`xts`’ XEX with tweaked code book and ciphertext stealing

The supported modes for content encryption ciphers

Since: 2.6

`QCryptoIVGenAlgorithm` [Enum]

‘`plain`’ 64-bit sector number truncated to 32-bits

‘`plain64`’ 64-bit sector number

‘`essiv`’ 64-bit sector number encrypted with a hash of the encryption key

The supported algorithms for generating initialization vectors for full disk encryption. The 'plain' generator should not be used for disks with sector numbers larger than 2^32, except where compatibility with pre-existing Linux dm-crypt volumes is required.

Since: 2.6

`QCryptoBlockFormat` [Enum]

‘`qcow`’ QCow/QCow2 built-in AES-CBC encryption. Use only for liberating data from old images.

‘`luks`’ LUKS encryption format. Recommended for new images

The supported full disk encryption formats

Since: 2.6

`QCryptoBlockOptionsBase` *{ 'format': `QCryptoBlockFormat` }* [Struct]

format the encryption format

The common options that apply to all full disk encryption formats

Since: 2.6

`QCryptoBlockOptionsQCow` *{ ['key-secret':* `str`*] }* [Struct]

*key-secret**
: the ID of a QCryptoSecret object providing the decryption key. Mandatory except when probing image for metadata only.

The options that apply to QCow/QCow2 AES-CBC encryption format

Since: 2.6

`QCryptoBlockOptionsLUKS` *{ ['key-secret':* `str`*] }* [Struct]

*key-secret**
: the ID of a QCryptoSecret object providing the decryption key. Mandatory except when probing image for metadata only.

The options that apply to LUKS encryption format

Since: 2.6

`QCryptoBlockCreateOptionsLUKS` *{ ['cipher-alg':* `QCryptoCipherAlgorithm`*], ['cipher-mode':* `QCryptoCipherMode`*], ['ivgen-alg':* `QCryptoIVGenAlgorithm`*], ['ivgen-hash-alg':* `QCryptoHashAlgorithm`*], ['hash-alg':* `QCryptoHashAlgorithm`*], ['iter-time':* `int`*] }* [Struct]

*cipher-alg**
: the cipher algorithm for data encryption Currently defaults to 'aes'.

*cipher-mode**
: the cipher mode for data encryption Currently defaults to 'cbc'

*ivgen-alg** the initialization vector generator Currently defaults to 'essiv'

*ivgen-hash-alg**
: the initialization vector generator hash Currently defaults to 'sha256'

*hash-alg** the master key hash algorithm Currently defaults to 'sha256'

*iter-time** number of milliseconds to spend in PBKDF passphrase processing. Currently defaults to 2000. (since 2.8)

The options that apply to LUKS encryption format initialization

Since: 2.6

`QCryptoBlockOpenOptions` [*'qcow':* `QCryptoBlockOptionsQCow`*, 'luks':* `QCryptoBlockOptionsLUKS`] [Union]

The options that are available for all encryption formats when opening an existing volume

Since: 2.6

`QCryptoBlockCreateOptions` [*'qcow':* `QCryptoBlockOptionsQCow`*, 'luks':* `QCryptoBlockCreateOptionsLUKS`] [Union]

The options that are available for all encryption formats when initializing a new volume

Since: 2.6

`QCryptoBlockInfoBase` *{ 'format':* `QCryptoBlockFormat` *}* [Struct]

format the encryption format

The common information that applies to all full disk encryption formats

Since: 2.7

`QCryptoBlockInfoLUKSSlot` *{ 'active':* `bool`*, ['iters':* `int`*], ['stripes':* `int`*], 'key-offset':* `int` *}* [Struct]

active whether the key slot is currently in use

key-offset offset to the key material in bytes

*iters** number of PBKDF2 iterations for key material

*stripes** number of stripes for splitting key material

Information about the LUKS block encryption key slot options

Since: 2.7

`QCryptoBlockInfoLUKS` *{ 'cipher-alg':* `QCryptoCipherAlgorithm`*, 'cipher-mode':* `QCryptoCipherMode`*, 'ivgen-alg':* `QCryptoIVGenAlgorithm`*, ['ivgen-hash-alg':* `QCryptoHashAlgorithm`*], 'hash-alg':* `QCryptoHashAlgorithm`*, 'payload-offset':* `int`*, 'master-key-iters':* `int`*, 'uuid':* `str`*, 'slots':* [`'QCryptoBlockInfoLUKSSlot'`] *}* [Struct]

cipher-alg the cipher algorithm for data encryption

cipher-mode the cipher mode for data encryption

ivgen-alg the initialization vector generator

*ivgen-hash-alg** the initialization vector generator hash

hash-alg the master key hash algorithm

payload-offset offset to the payload data in bytes

master-key-iters number of PBKDF2 iterations for key material

uuid unique identifier for the volume

slots information about each key slot

Information about the LUKS block encryption options

Since: 2.7

`QCryptoBlockInfoQCow` *{ }* [Struct]

Information about the QCow block encryption options

Since: 2.7

`QCryptoBlockInfo` [*'qcow':* `QCryptoBlockInfoQCow`, *'luks':* `QCryptoBlockInfoLUKS`] [Union]

Information about the block encryption options

Since: 2.7

2.5 QAPI block definitions (vm related)

2.5.1 QAPI block core definitions (vm unrelated)

`SnapshotInfo` *{ 'id':* `str`, *'name':* `str`, *'vm-state-size':* `int`, *'date-sec':* `int`, *'date-nsec':* `int`, *'vm-clock-sec':* `int`, *'vm-clock-nsec':* `int` *}* [Struct]

id unique snapshot id

name user chosen name

vm-state-size
size of the VM state

date-sec UTC date of the snapshot in seconds

date-nsec fractional part in nano seconds to be used with date-sec

vm-clock-sec
VM clock relative to boot in seconds

vm-clock-nsec
fractional part in nano seconds to be used with vm-clock-sec

Since: 1.3

`ImageInfoSpecificQCow2` *{ 'compat':* `str`, [*'lazy-refcounts':* `bool`], [*'corrupt':* `bool`], *'refcount-bits':* `int` *}* [Struct]

compat compatibility level

*lazy-refcounts**
on or off; only valid for compat >= 1.1

*corrupt** true if the image has been marked corrupt; only valid for compat >= 1.1 (since 2.2)

refcount-bits
width of a refcount entry in bits (since 2.3)

Since: 1.7

`ImageInfoSpecificVmdk` *{ 'create-type':* `str`, *'cid':* `int`, *'parent-cid':* `int`, *'extents':* [`'ImageInfo'`] *}* [Struct]

create-type
The create type of VMDK image

cid Content id of image

parent-cid Parent VMDK image's cid

extents List of extent files

Since: 1.7

`ImageInfoSpecific` [*'qcow2':* `ImageInfoSpecificQCow2`, *'vmdk':* `ImageInfoSpecificVmdk`, *'luks':* `QCryptoBlockInfoLUKS`] [Union]

A discriminated record of image format specific information structures.

Since: 1.7

`ImageInfo` *{ 'filename':* `str`, *'format':* `str`, [*'dirty-flag':* `bool`], [*'actual-size':* `int`], *'virtual-size':* `int`, [*'cluster-size':* `int`], [*'encrypted':* `bool`], [*'compressed':* `bool`], [*'backing-filename':* `str`], [*'full-backing-filename':* `str`], [*'backing-filename-format':* `str`], [*'snapshots':* [`'SnapshotInfo'`]], [*'backing-image':* `ImageInfo`], [*'format-specific':* `ImageInfoSpecific`] *}* [Struct]

filename name of the image file

format format of the image file

virtual-size
: maximum capacity in bytes of the image

*actual-size**
: actual size on disk in bytes of the image

*dirty-flag** true if image is not cleanly closed

*cluster-size**
: size of a cluster in bytes

*encrypted**
: true if the image is encrypted

*compressed**
: true if the image is compressed (Since 1.7)

*backing-filename**
: name of the backing file

*full-backing-filename**
: full path of the backing file

*backing-filename-format**
: the format of the backing file

*snapshots**
: list of VM snapshots

*backing-image**
: info of the backing image (since 1.6)

*format-specific**
: structure supplying additional format-specific information (since 1.7)

Information about a QEMU image file

Since: 1.3

`ImageCheck` *{ 'filename':* `str`*, 'format':* `str`*, 'check-errors':* `int`*,* [Struct]
['image-end-offset': `int`*], ['corruptions':* `int`*], ['leaks':* `int`*],*
['corruptions-fixed': `int`*], ['leaks-fixed':* `int`*], ['total-clusters':* `int`*],*
['allocated-clusters': `int`*], ['fragmented-clusters':* `int`*],*
['compressed-clusters': `int`*] }*

filename name of the image file checked

format format of the image file checked

check-errors
number of unexpected errors occurred during check

*image-end-offset**
offset (in bytes) where the image ends, this field is present if the driver for the image format supports it

*corruptions**
number of corruptions found during the check if any

*leaks** number of leaks found during the check if any

*corruptions-fixed**
number of corruptions fixed during the check if any

*leaks-fixed**
number of leaks fixed during the check if any

*total-clusters**
total number of clusters, this field is present if the driver for the image format supports it

*allocated-clusters**
total number of allocated clusters, this field is present if the driver for the image format supports it

*fragmented-clusters**
total number of fragmented clusters, this field is present if the driver for the image format supports it

*compressed-clusters**
total number of compressed clusters, this field is present if the driver for the image format supports it

Information about a QEMU image file check

Since: 1.4

`MapEntry` *{ 'start':* `int`*, 'length':* `int`*, 'data':* `bool`*, 'zero':* `bool`*,* [Struct]
'depth': `int`*, ['offset':* `int`*], ['filename':* `str`*] }*

start the start byte of the mapped virtual range

length the number of bytes of the mapped virtual range

data whether the mapped range has data

zero whether the virtual blocks are zeroed

depth the depth of the mapping

*offset** the offset in file that the virtual sectors are mapped to

*filename** filename that is referred to by *offset*

Mapping information from a virtual block range to a host file range

Since: 2.6

`BlockdevCacheInfo` *{ 'writeback':* `bool`*, 'direct':* `bool`*, 'no-flush':* `bool` *}* [Struct]

writeback true if writeback mode is enabled

direct true if the host page cache is bypassed (O_DIRECT)

no-flush true if flush requests are ignored for the device

Cache mode information for a block device

Since: 2.3

`BlockDeviceInfo` *{ 'file':* `str`*, ['node-name':* `str`*], 'ro':* `bool`*, 'drv':* `str`*, ['backing_file':* `str`*], 'backing_file_depth':* `int`*, 'encrypted':* `bool`*, 'encryption_key_missing':* `bool`*, 'detect_zeroes':* `BlockdevDetectZeroesOptions`*, 'bps':* `int`*, 'bps_rd':* `int`*, 'bps_wr':* `int`*, 'iops':* `int`*, 'iops_rd':* `int`*, 'iops_wr':* `int`*, 'image':* `ImageInfo`*, ['bps_max':* `int`*], ['bps_rd_max':* `int`*], ['bps_wr_max':* `int`*], ['iops_max':* `int`*], ['iops_rd_max':* `int`*], ['iops_wr_max':* `int`*], ['bps_max_length':* `int`*], ['bps_rd_max_length':* `int`*], ['bps_wr_max_length':* `int`*], ['iops_max_length':* `int`*], ['iops_rd_max_length':* `int`*], ['iops_wr_max_length':* `int`*], ['iops_size':* `int`*], ['group':* `str`*], 'cache':* `BlockdevCacheInfo`*, 'write_threshold':* `int` *}* [Struct]

file the filename of the backing device

*node-name**
the name of the block driver node (Since 2.0)

ro true if the backing device was open read-only

drv the name of the block format used to open the backing device. As of 0.14.0 this can be: 'blkdebug', 'bochs', 'cloop', 'cow', 'dmg', 'file', 'file', 'ftp', 'ftps', 'host_cdrom', 'host_device', 'http', 'https', 'luks', 'nbd', 'parallels', 'qcow', 'qcow2', 'raw', 'tftp', 'vdi', 'vmdk', 'vpc', 'vvfat' 2.2: 'archipelago' added, 'cow' dropped 2.3: 'host_floppy' deprecated 2.5: 'host_floppy' dropped 2.6: 'luks' added 2.8: 'replication' added

*backing_file**
the name of the backing file (for copy-on-write)

backing_file_depth
number of files in the backing file chain (since: 1.2)

encrypted true if the backing device is encrypted

encryption_key_missing
: true if the backing device is encrypted but an valid encryption key is missing

detect_zeroes
: detect and optimize zero writes (Since 2.1)

bps
: total throughput limit in bytes per second is specified

bps_rd
: read throughput limit in bytes per second is specified

bps_wr
: write throughput limit in bytes per second is specified

iops
: total I/O operations per second is specified

iops_rd
: read I/O operations per second is specified

iops_wr
: write I/O operations per second is specified

image
: the info of image used (since: 1.6)

*bps_max**
: total throughput limit during bursts, in bytes (Since 1.7)

*bps_rd_max**
: read throughput limit during bursts, in bytes (Since 1.7)

*bps_wr_max**
: write throughput limit during bursts, in bytes (Since 1.7)

*iops_max**
: total I/O operations per second during bursts, in bytes (Since 1.7)

*iops_rd_max**
: read I/O operations per second during bursts, in bytes (Since 1.7)

*iops_wr_max**
: write I/O operations per second during bursts, in bytes (Since 1.7)

*bps_max_length**
: maximum length of the *bps_max* burst period, in seconds. (Since 2.6)

*bps_rd_max_length**
: maximum length of the *bps_rd_max* burst period, in seconds. (Since 2.6)

*bps_wr_max_length**
: maximum length of the *bps_wr_max* burst period, in seconds. (Since 2.6)

*iops_max_length**
: maximum length of the *iops* burst period, in seconds. (Since 2.6)

*iops_rd_max_length**
: maximum length of the *iops_rd_max* burst period, in seconds. (Since 2.6)

*iops_wr_max_length**
: maximum length of the *iops_wr_max* burst period, in seconds. (Since 2.6)

*iops_size**
: an I/O size in bytes (Since 1.7)

*group**
: throttle group name (Since 2.4)

cache the cache mode used for the block device (since: 2.3)

write_threshold
configured write threshold for the device. 0 if disabled. (Since 2.3)

Information about the backing device for a block device.

Since: 0.14.0

`BlockDeviceIoStatus` [Enum]

'`ok`' The last I/O operation has succeeded

'`failed`' The last I/O operation has failed

'`nospace`' The last I/O operation has failed due to a no-space condition

An enumeration of block device I/O status.

Since: 1.0

`BlockDeviceMapEntry` *{ 'start':* `int`*, 'length':* `int`*, 'depth':* `int`*, 'zero':* `bool`*, 'data':* `bool`*, ['offset':* `int`*] }* [Struct]

start Offset in the image of the first byte described by this entry (in bytes)

length Length of the range described by this entry (in bytes)

depth Number of layers (0 = top image, 1 = top image's backing file, etc.) before reaching one for which the range is allocated. The value is in the range 0 to the depth of the image chain - 1.

zero the sectors in this range read as zeros

data reading the image will actually read data from a file (in particular, if *offset* is present this means that the sectors are not simply preallocated, but contain actual data in raw format)

offset if present, the image file stores the data for this range in raw format at the given offset.

Entry in the metadata map of the device (returned by "qemu-img map")

Since: 1.7

`DirtyBitmapStatus` [Enum]

'`frozen`' The bitmap is currently in-use by a backup operation or block job, and is immutable.

'`disabled`'
The bitmap is currently in-use by an internal operation and is read-only. It can still be deleted.

'`active`' The bitmap is actively monitoring for new writes, and can be cleared, deleted, or used for backup operations.

An enumeration of possible states that a dirty bitmap can report to the user.

Since: 2.4

`BlockDirtyInfo` *{ ['name':* `str`*], 'count':* `int`*, 'granularity':* `uint32`*, 'status':* `DirtyBitmapStatus` *}* [Struct]

*name** the name of the dirty bitmap (Since 2.4)

count number of dirty bytes according to the dirty bitmap

granularity granularity of the dirty bitmap in bytes (since 1.4)

status current status of the dirty bitmap (since 2.4)

Block dirty bitmap information.

Since: 1.3

`BlockInfo` *{ 'device':* `str`*, 'type':* `str`*, 'removable':* `bool`*, 'locked':* `bool`*, ['inserted':* `BlockDeviceInfo`*], ['tray_open':* `bool`*], ['io-status':* `BlockDeviceIoStatus`*], ['dirty-bitmaps':* `['BlockDirtyInfo']`*] }* [Struct]

device The device name associated with the virtual device.

type This field is returned only for compatibility reasons, it should not be used (always returns 'unknown')

removable True if the device supports removable media.

locked True if the guest has locked this device from having its media removed

*tray_open** True if the device's tray is open (only present if it has a tray)

*dirty-bitmaps** dirty bitmaps information (only present if the driver has one or more dirty bitmaps) (Since 2.0)

*io-status** *BlockDeviceIoStatus*. Only present if the device supports it and the VM is configured to stop on errors (supported device models: virtio-blk, ide, scsi-disk)

*inserted** *BlockDeviceInfo* describing the device if media is present

Block device information. This structure describes a virtual device and the backing device associated with it.

Since: 0.14.0

`['BlockInfo'] query-block ()` [Command]

Get a list of BlockInfo for all virtual block devices.

Returns: a list of *BlockInfo* describing each virtual block device

Since: 0.14.0

Example:

```
-> { "execute": "query-block" }
<- {
      "return":[
         {
```

```
"io-status": "ok",
"device":"ide0-hd0",
"locked":false,
"removable":false,
"inserted":{
   "ro":false,
   "drv":"qcow2",
   "encrypted":false,
   "file":"disks/test.qcow2",
   "backing_file_depth":1,
   "bps":1000000,
   "bps_rd":0,
   "bps_wr":0,
   "iops":1000000,
   "iops_rd":0,
   "iops_wr":0,
   "bps_max": 8000000,
   "bps_rd_max": 0,
   "bps_wr_max": 0,
   "iops_max": 0,
   "iops_rd_max": 0,
   "iops_wr_max": 0,
   "iops_size": 0,
   "detect_zeroes": "on",
   "write_threshold": 0,
   "image":{
      "filename":"disks/test.qcow2",
      "format":"qcow2",
      "virtual-size":2048000,
      "backing_file":"base.qcow2",
      "full-backing-filename":"disks/base.qcow2",
      "backing-filename-format":"qcow2",
      "snapshots":[
         {
            "id": "1",
            "name": "snapshot1",
            "vm-state-size": 0,
            "date-sec": 10000200,
            "date-nsec": 12,
            "vm-clock-sec": 206,
            "vm-clock-nsec": 30
         }
      ],
      "backing-image":{
          "filename":"disks/base.qcow2",
          "format":"qcow2",
          "virtual-size":2048000
```

```
                    }
                 }
              },
              "type":"unknown"
           },
           {
              "io-status": "ok",
              "device":"ide1-cd0",
              "locked":false,
              "removable":true,
              "type":"unknown"
           },
           {
              "device":"floppy0",
              "locked":false,
              "removable":true,
              "type":"unknown"
           },
           {
              "device":"sd0",
              "locked":false,
              "removable":true,
              "type":"unknown"
           }
        ]
     }
```

`BlockDeviceTimedStats` *{ 'interval_length':* `int`*, 'min_rd_latency_ns':* `int`*, 'max_rd_latency_ns':* `int`*, 'avg_rd_latency_ns':* `int`*, 'min_wr_latency_ns':* `int`*, 'max_wr_latency_ns':* `int`*, 'avg_wr_latency_ns':* `int`*, 'min_flush_latency_ns':* `int`*, 'max_flush_latency_ns':* `int`*, 'avg_flush_latency_ns':* `int`*, 'avg_rd_queue_depth':* `number`*, 'avg_wr_queue_depth':* `number` *}* [Struct]

interval_length
: Interval used for calculating the statistics, in seconds.

min_rd_latency_ns
: Minimum latency of read operations in the defined interval, in nanoseconds.

min_wr_latency_ns
: Minimum latency of write operations in the defined interval, in nanoseconds.

min_flush_latency_ns
: Minimum latency of flush operations in the defined interval, in nanoseconds.

max_rd_latency_ns
: Maximum latency of read operations in the defined interval, in nanoseconds.

max_wr_latency_ns
: Maximum latency of write operations in the defined interval, in nanoseconds.

max_flush_latency_ns
: Maximum latency of flush operations in the defined interval, in nanoseconds.

avg_rd_latency_ns
: Average latency of read operations in the defined interval, in nanoseconds.

avg_wr_latency_ns
: Average latency of write operations in the defined interval, in nanoseconds.

avg_flush_latency_ns
: Average latency of flush operations in the defined interval, in nanoseconds.

avg_rd_queue_depth
: Average number of pending read operations in the defined interval.

avg_wr_queue_depth
: Average number of pending write operations in the defined interval.

Statistics of a block device during a given interval of time.

Since: 2.5

`BlockDeviceStats` *{ 'rd_bytes':* `int`*, 'wr_bytes':* `int`*, 'rd_operations':* `int`*, 'wr_operations':* `int`*, 'flush_operations':* `int`*, 'flush_total_time_ns':* `int`*, 'wr_total_time_ns':* `int`*, 'rd_total_time_ns':* `int`*, 'wr_highest_offset':* `int`*, 'rd_merged':* `int`*, 'wr_merged':* `int`*,* [*'idle_time_ns':* `int`]*, 'failed_rd_operations':* `int`*, 'failed_wr_operations':* `int`*, 'failed_flush_operations':* `int`*, 'invalid_rd_operations':* `int`*, 'invalid_wr_operations':* `int`*, 'invalid_flush_operations':* `int`*, 'account_invalid':* `bool`*, 'account_failed':* `bool`*, 'timed_stats':* [`'BlockDeviceTimedStats'`] *}* [Struct]

rd_bytes The number of bytes read by the device.

wr_bytes The number of bytes written by the device.

rd_operations
: The number of read operations performed by the device.

wr_operations
: The number of write operations performed by the device.

flush_operations
: The number of cache flush operations performed by the device (since 0.15.0)

flush_total_time_ns
: Total time spend on cache flushes in nano-seconds (since 0.15.0).

wr_total_time_ns
: Total time spend on writes in nano-seconds (since 0.15.0).

rd_total_time_ns
: Total_time_spend on reads in nano-seconds (since 0.15.0).

wr_highest_offset
: The offset after the greatest byte written to the device. The intended use of this information is for growable sparse files (like qcow2) that are used on top of a physical device.

rd_merged Number of read requests that have been merged into another request (Since 2.3).

wr_merged
: Number of write requests that have been merged into another request (Since 2.3).

*idle_time_ns**
: Time since the last I/O operation, in nanoseconds. If the field is absent it means that there haven't been any operations yet (Since 2.5).

failed_rd_operations
: The number of failed read operations performed by the device (Since 2.5)

failed_wr_operations
: The number of failed write operations performed by the device (Since 2.5)

failed_flush_operations
: The number of failed flush operations performed by the device (Since 2.5)

invalid_rd_operations
: The number of invalid read operations performed by the device (Since 2.5)

invalid_wr_operations
: The number of invalid write operations performed by the device (Since 2.5)

invalid_flush_operations
: The number of invalid flush operations performed by the device (Since 2.5)

account_invalid
: Whether invalid operations are included in the last access statistics (Since 2.5)

account_failed
: Whether failed operations are included in the latency and last access statistics (Since 2.5)

timed_stats
: Statistics specific to the set of previously defined intervals of time (Since 2.5)

Statistics of a virtual block device or a block backing device.

Since: 0.14.0

`BlockStats` *{* [*'device':* `str`], [*'node-name':* `str`], *'stats':* `BlockDeviceStats`, [*'parent':* `BlockStats`], [*'backing':* `BlockStats`] *}* [Struct]

*device** If the stats are for a virtual block device, the name corresponding to the virtual block device.

*node-name**
: The node name of the device. (Since 2.3)

stats A *BlockDeviceStats* for the device.

*parent** This describes the file block device if it has one. Contains recursively the statistics of the underlying protocol (e.g. the host file for a qcow2 image). If there is no underlying protocol, this field is omitted

*backing** This describes the backing block device if it has one. (Since 2.0)

Statistics of a virtual block device or a block backing device.

Since: 0.14.0

`['BlockStats'] query-blockstats` ([*'query-nodes':* `bool`]) [Command]

*query-nodes**
: If true, the command will query all the block nodes that have a node name, in a list which will include "parent" information, but not "backing". If false or omitted, the behavior is as before - query all the device backends, recursively including their "parent" and "backing". (Since 2.3)

Query the *BlockStats* for all virtual block devices.

Returns: A list of *BlockStats* for each virtual block devices.

Since: 0.14.0

Example:

```
-> { "execute": "query-blockstats" }
<- {
      "return":[
         {
            "device":"ide0-hd0",
            "parent":{
               "stats":{
                  "wr_highest_offset":3686448128,
                  "wr_bytes":9786368,
                  "wr_operations":751,
                  "rd_bytes":122567168,
                  "rd_operations":36772
                  "wr_total_times_ns":313253456
```

```
            "rd_total_times_ns":3465673657
            "flush_total_times_ns":49653
            "flush_operations":61,
            "rd_merged":0,
            "wr_merged":0,
            "idle_time_ns":2953431879,
            "account_invalid":true,
            "account_failed":false
         }
      },
      "stats":{
         "wr_highest_offset":2821110784,
         "wr_bytes":9786368,
         "wr_operations":692,
         "rd_bytes":122739200,
         "rd_operations":36604
         "flush_operations":51,
         "wr_total_times_ns":313253456
         "rd_total_times_ns":3465673657
         "flush_total_times_ns":49653,
         "rd_merged":0,
         "wr_merged":0,
         "idle_time_ns":2953431879,
         "account_invalid":true,
         "account_failed":false
      }
   },
   {
      "device":"ide1-cd0",
      "stats":{
         "wr_highest_offset":0,
         "wr_bytes":0,
         "wr_operations":0,
         "rd_bytes":0,
         "rd_operations":0
         "flush_operations":0,
         "wr_total_times_ns":0
         "rd_total_times_ns":0
         "flush_total_times_ns":0,
         "rd_merged":0,
         "wr_merged":0,
         "account_invalid":false,
         "account_failed":false
      }
   },
   {
      "device":"floppy0",
```

```
                "stats":{
                   "wr_highest_offset":0,
                   "wr_bytes":0,
                   "wr_operations":0,
                   "rd_bytes":0,
                   "rd_operations":0
                   "flush_operations":0,
                   "wr_total_times_ns":0
                   "rd_total_times_ns":0
                   "flush_total_times_ns":0,
                   "rd_merged":0,
                   "wr_merged":0,
                   "account_invalid":false,
                   "account_failed":false
                }
             },
             {
                "device":"sd0",
                "stats":{
                   "wr_highest_offset":0,
                   "wr_bytes":0,
                   "wr_operations":0,
                   "rd_bytes":0,
                   "rd_operations":0
                   "flush_operations":0,
                   "wr_total_times_ns":0
                   "rd_total_times_ns":0
                   "flush_total_times_ns":0,
                   "rd_merged":0,
                   "wr_merged":0,
                   "account_invalid":false,
                   "account_failed":false
                }
             }
          ]
       }
```

`BlockdevOnError` [Enum]

‘`report`’ for guest operations, report the error to the guest; for jobs, cancel the job

‘`ignore`’ ignore the error, only report a QMP event (BLOCK_IO_ERROR or BLOCK_JOB_ERROR)

‘`enospc`’ same as *stop* on ENOSPC, same as *report* otherwise.

‘`stop`’ for guest operations, stop the virtual machine; for jobs, pause the job

‘`auto`’ inherit the error handling policy of the backend (since: 2.7)

An enumeration of possible behaviors for errors on I/O operations. The exact meaning depends on whether the I/O was initiated by a guest or by a block job

Since: 1.3

`MirrorSyncMode` [Enum]

‘`top`’ copies data in the topmost image to the destination

‘`full`’ copies data from all images to the destination

‘`none`’ only copy data written from now on

‘`incremental`’
only copy data described by the dirty bitmap. Since: 2.4

An enumeration of possible behaviors for the initial synchronization phase of storage mirroring.

Since: 1.3

`BlockJobType` [Enum]

‘`commit`’ block commit job type, see "block-commit"

‘`stream`’ block stream job type, see "block-stream"

‘`mirror`’ drive mirror job type, see "drive-mirror"

‘`backup`’ drive backup job type, see "drive-backup"

Type of a block job.

Since: 1.7

`BlockJobInfo` *{ 'type':* `str`*, 'device':* `str`*, 'len':* `int`*, 'offset':* `int`*, 'busy':* `bool`*, 'paused':* `bool`*, 'speed':* `int`*, 'io-status':* `BlockDeviceIoStatus`*, 'ready':* `bool` *}* [Struct]

type the job type ('stream' for image streaming)

device The job identifier. Originally the device name but other values are allowed since QEMU 2.7

len the maximum progress value

busy false if the job is known to be in a quiescent state, with no pending I/O. Since 1.3.

paused whether the job is paused or, if *busy* is true, will pause itself as soon as possible. Since 1.3.

offset the current progress value

speed the rate limit, bytes per second

io-status the status of the job (since 1.3)

ready true if the job may be completed (since 2.2)

Information about a long-running block device operation.

Since: 1.1

`['BlockJobInfo'] query-block-jobs ()` [Command]

Return information about long-running block device operations.

Returns: a list of *BlockJobInfo* for each active block job

Since: 1.1

`block_passwd (['device': str], ['node-name': str], 'password': str)` [Command]

change interface.

In the event that the block device is created through the initial command line, the VM will start in the stopped state regardless of whether '-S' is used. The intention is for a management tool to query the block devices to determine which ones are encrypted, set the passwords with this command, and then start the guest with the *cont* command.

Either *device* or *node*-name must be set but not both.

*device** the name of the block backend device to set the password on

*node-name**

graph node name to set the password on (Since 2.0)

password the password to use for the device

This command sets the password of a block device that has not been open with a password and requires one.

The two cases where this can happen are a block device is created through QEMU's initial command line or a block device is changed through the legacy

Returns: nothing on success If *device* is not a valid block device, DeviceNotFound If *device* is not encrypted, DeviceNotEncrypted

Notes: Not all block formats support encryption and some that do are not able to validate that a password is correct. Disk corruption may occur if an invalid password is specified.

Since: 0.14.0

Example:

```
-> { "execute": "block_passwd", "arguments": { "device": "ide0-hd0",
                                               "password": "12345" } }
<- { "return": {} }
```

`block_resize (['device': str], ['node-name': str], 'size': int)` [Command]

*device** the name of the device to get the image resized

*node-name**

graph node name to get the image resized (Since 2.0)

size new image size in bytes

Resize a block image while a guest is running.

Either *device* or *node*-name must be set but not both.

Returns: nothing on success If *device* is not a valid block device, DeviceNotFound

Since: 0.14.0

Example:

```
-> { "execute": "block_resize",
     "arguments": { "device": "scratch", "size": 1073741824 } }
<- { "return": {} }
```

`NewImageMode` [Enum]

'`existing`'
: QEMU should look for an existing image file.

'`absolute-paths`'
: QEMU should create a new image with absolute paths for the backing file. If there is no backing file available, the new image will not be backed either.

An enumeration that tells QEMU how to set the backing file path in a new image file.

Since: 1.1

`BlockdevSnapshotSync` *{ ['device': `str`], ['node-name': `str`], ['snapshot-file': `str`, ['snapshot-node-name': `str`], ['format': `str`], ['mode': `NewImageMode`] }* [Struct]

*device** the name of the device to generate the snapshot from.

*node-name**
: graph node name to generate the snapshot from (Since 2.0)

snapshot-file
: the target of the new image. If the file exists, or if it is a device, the snapshot will be created in the existing file/device. Otherwise, a new file will be created.

*snapshot-node-name**
: the graph node name of the new image (Since 2.0)

*format** the format of the snapshot image, default is 'qcow2'.

*mode** whether and how QEMU should create a new image, default is 'absolute-paths'.

Either *device* or *node*-name must be set but not both.

`BlockdevSnapshot` *{ 'node': `str`, 'overlay': `str` }* [Struct]

node device or node name that will have a snapshot created.

overlay reference to the existing block device that will become the overlay of *node*, as part of creating the snapshot. It must not have a current backing file (this can be achieved by passing "backing": "" to blockdev-add).

Since: 2.5

`DriveBackup` *{ ['job-id': `str`], 'device': `str`, 'target': `str`, ['format': `str`], 'sync': `MirrorSyncMode`, ['mode': `NewImageMode`], ['speed': `int`], ['bitmap': `str`], ['compress': `bool`], ['on-source-error': `BlockdevOnError`], ['on-target-error': `BlockdevOnError`] }* [Struct]

*job-id** identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device the device name or node-name of a root node which should be copied.

target the target of the new image. If the file exists, or if it is a device, the existing file/device will be used as the new destination. If it does not exist, a new file will be created.

*format** the format of the new destination, default is to probe if *mode* is 'existing', else the format of the source

sync what parts of the disk image should be copied to the destination (all the disk, only the sectors allocated in the topmost image, from a dirty bitmap, or only new I/O).

*mode** whether and how QEMU should create a new image, default is 'absolute-paths'.

*speed** the maximum speed, in bytes per second

*bitmap** the name of dirty bitmap if sync is "incremental". Must be present if sync is "incremental", must NOT be present otherwise. (Since 2.4)

*compress** true to compress data, if the target format supports it. (default: false) (since 2.8)

*on-source-error**
the action to take on an error on the source, default 'report'. 'stop' and 'enospc' can only be used if the block device supports io-status (see BlockInfo).

*on-target-error**
the action to take on an error on the target, default 'report' (no limitations, since this applies to a different block device than *device*).

Note that *on*-source-error and *on*-target-error only affect background I/O. If an error occurs during a guest write request, the device's rerror/werror actions will be used.

Since: 1.6

`BlockdevBackup` *{ ['job-id': `str`], 'device': `str`, 'target': `str`, 'sync': `MirrorSyncMode`, ['speed': `int`], ['compress': `bool`], ['on-source-error': `BlockdevOnError`], ['on-target-error': `BlockdevOnError`] }* [Struct]

*job-id** identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device the device name or node-name of a root node which should be copied.

target the device name or node-name of the backup target node.

sync what parts of the disk image should be copied to the destination (all the disk, only the sectors allocated in the topmost image, or only new I/O).

*speed** the maximum speed, in bytes per second. The default is 0, for unlimited.

*compress** true to compress data, if the target format supports it. (default: false) (since 2.8)

*on-source-error**
the action to take on an error on the source, default 'report'. 'stop' and 'enospc' can only be used if the block device supports io-status (see BlockInfo).

*on-target-error**
the action to take on an error on the target, default 'report' (no limitations, since this applies to a different block device than *device*).

Note that *on*-source-error and *on*-target-error only affect background I/O. If an error occurs during a guest write request, the device's rerror/werror actions will be used.

Since: 2.3

`blockdev-snapshot-sync` (*BlockdevSnapshotSync*) [Command]

Generates a synchronous snapshot of a block device.

For the arguments, see the documentation of BlockdevSnapshotSync.

Returns: nothing on success If *device* is not a valid block device, DeviceNotFound

Since: 0.14.0

Example:

```
 -> { "execute": "blockdev-snapshot-sync",
      "arguments": { "device": "ide-hd0",
                     "snapshot-file":
                     "/some/place/my-image",
                     "format": "qcow2" } }
 <- { "return": {} }
```

`blockdev-snapshot` (*BlockdevSnapshot*) [Command]

Generates a snapshot of a block device.

Create a snapshot, by installing 'node' as the backing image of 'overlay'. Additionally, if 'node' is associated with a block device, the block device changes to using 'overlay' as its new active image.

For the arguments, see the documentation of BlockdevSnapshot.

Since: 2.5

Example:

```
 -> { "execute": "blockdev-add",
      "arguments": { "options": { "driver": "qcow2",
                                  "node-name": "node1534",
                                  "file": { "driver": "file",
```

```
                                                "filename": "hd1.qcow2" },
                                      "backing": "" } } }

      <- { "return": {} }

      -> { "execute": "blockdev-snapshot",
           "arguments": { "node": "ide-hd0",
                          "overlay": "node1534" } }
      <- { "return": {} }
```

`change-backing-file` (*'device':* ***str****, 'image-node-name':* ***str****, 'backing-file':* ***str***) [Command]

image-node-name
: The name of the block driver state node of the image to modify. The "device" argument is used to verify "image-node-name" is in the chain described by "device".

device
: The device name or node-name of the root node that owns image-node-name.

backing-file
: The string to write as the backing file. This string is not validated, so care should be taken when specifying the string or the image chain may not be able to be reopened again.

Change the backing file in the image file metadata. This does not cause QEMU to reopen the image file to reparse the backing filename (it may, however, perform a reopen to change permissions from r/o -> r/w -> r/o, if needed). The new backing file string is written into the image file metadata, and the QEMU internal strings are updated.

Returns: Nothing on success

If "device" does not exist or cannot be determined, DeviceNotFound

Since: 2.1

`block-commit` ([*'job-id':* ***str***], *'device':* ***str***, [*'base':* ***str***], [*'top':* ***str***], [*'backing-file':* ***str***], [*'speed':* ***int***]) [Command]

*job-id**
: identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device
: the device name or node-name of a root node

*base**
: The file name of the backing image to write data into. If not specified, this is the deepest backing image.

*top**
: The file name of the backing image within the image chain, which contains the topmost data to be committed down. If not specified, this is the active layer.

*backing-file**
: The backing file string to write into the overlay image of 'top'. If 'top' is the active layer, specifying a backing file string is an error. This filename is not validated.

If a pathname string is such that it cannot be resolved by QEMU, that means that subsequent QMP or HMP commands must use node-names for the image in question, as filename lookup methods will fail.

If not specified, QEMU will automatically determine the backing file string to use, or error out if there is no obvious choice. Care should be taken when specifying the string, to specify a valid filename or protocol. (Since 2.1)

If top == base, that is an error. If top == active, the job will not be completed by itself, user needs to complete the job with the block-job-complete command after getting the ready event. (Since 2.0)

If the base image is smaller than top, then the base image will be resized to be the same size as top. If top is smaller than the base image, the base will not be truncated. If you want the base image size to match the size of the smaller top, you can safely truncate it yourself once the commit operation successfully completes.

*speed** the maximum speed, in bytes per second

Live commit of data from overlay image nodes into backing nodes - i.e., writes data between 'top' and 'base' into 'base'.

Returns: Nothing on success If commit or stream is already active on this device, DeviceInUse If *device* does not exist, DeviceNotFound If image commit is not supported by this device, NotSupported If *base* or *top* is invalid, a generic error is returned If *speed* is invalid, InvalidParameter

Since: 1.3

Example:

```
-> { "execute": "block-commit",
     "arguments": { "device": "virtio0",
                    "top": "/tmp/snap1.qcow2" } }
<- { "return": {} }
```

`drive-backup` (*DriveBackup*) [Command]

Start a point-in-time copy of a block device to a new destination. The status of ongoing drive-backup operations can be checked with query-block-jobs where the BlockJobInfo.type field has the value 'backup'. The operation can be stopped before it has completed using the block-job-cancel command.

For the arguments, see the documentation of DriveBackup.

Returns: nothing on success If *device* is not a valid block device, GenericError

Since: 1.6

Example:

```
-> { "execute": "drive-backup",
     "arguments": { "device": "drive0",
                    "sync": "full",
                    "target": "backup.img" } }
<- { "return": {} }
```

`blockdev-backup` (*BlockdevBackup*) [Command]

Start a point-in-time copy of a block device to a new destination. The status of ongoing blockdev-backup operations can be checked with query-block-jobs where the BlockJobInfo.type field has the value 'backup'. The operation can be stopped before it has completed using the block-job-cancel command.

For the arguments, see the documentation of BlockdevBackup.

Returns: nothing on success If *device* is not a valid block device, DeviceNotFound

Since: 2.3

Example:

```
 -> { "execute": "blockdev-backup",
      "arguments": { "device": "src-id",
                     "sync": "full",
                     "target": "tgt-id" } }
 <- { "return": {} }
```

`['BlockDeviceInfo'] query-named-block-nodes ()` [Command]

Get the named block driver list

Returns: the list of BlockDeviceInfo

Since: 2.0

Example:

```
 -> { "execute": "query-named-block-nodes" }
 <- { "return": [ { "ro":false,
                    "drv":"qcow2",
                    "encrypted":false,
                    "file":"disks/test.qcow2",
                    "node-name": "my-node",
                    "backing_file_depth":1,
                    "bps":1000000,
                    "bps_rd":0,
                    "bps_wr":0,
                    "iops":1000000,
                    "iops_rd":0,
                    "iops_wr":0,
                    "bps_max": 8000000,
                    "bps_rd_max": 0,
                    "bps_wr_max": 0,
                    "iops_max": 0,
                    "iops_rd_max": 0,
                    "iops_wr_max": 0,
                    "iops_size": 0,
                    "write_threshold": 0,
                    "image":{
                       "filename":"disks/test.qcow2",
                       "format":"qcow2",
                       "virtual-size":2048000,
```

```
                    "backing_file":"base.qcow2",
                    "full-backing-filename":"disks/base.qcow2",
                    "backing-filename-format":"qcow2",
                    "snapshots":[
                       {
                          "id": "1",
                          "name": "snapshot1",
                          "vm-state-size": 0,
                          "date-sec": 10000200,
                          "date-nsec": 12,
                          "vm-clock-sec": 206,
                          "vm-clock-nsec": 30
                       }
                    ],
                    "backing-image":{
                        "filename":"disks/base.qcow2",
                        "format":"qcow2",
                        "virtual-size":2048000
                    }
                 } } ] }
```

`drive-mirror` (*DriveMirror*) [Command]

Start mirroring a block device's writes to a new destination. target specifies the target of the new image. If the file exists, or if it is a device, it will be used as the new destination for writes. If it does not exist, a new file will be created. format specifies the format of the mirror image, default is to probe if mode='existing', else the format of the source.

See DriveMirror for parameter descriptions

Returns: nothing on success If *device* is not a valid block device, GenericError

Since: 1.3

Example:

```
 -> { "execute": "drive-mirror",
      "arguments": { "device": "ide-hd0",
                     "target": "/some/place/my-image",
                     "sync": "full",
                     "format": "qcow2" } }
 <- { "return": {} }
```

`DriveMirror` *{ ['job-id': `str`], 'device': `str`, 'target': `str`, ['format': `str`], ['node-name': `str`], ['replaces': `str`], 'sync': `MirrorSyncMode`, ['mode': `NewImageMode`], ['speed': `int`], ['granularity': `uint32`], ['buf-size': `int`], ['on-source-error': `BlockdevOnError`], ['on-target-error': `BlockdevOnError`], ['unmap': `bool`] }* [Struct]

*job-id** identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device the device name or node-name of a root node whose writes should be mirrored.

target the target of the new image. If the file exists, or if it is a device, the existing file/device will be used as the new destination. If it does not exist, a new file will be created.

*format** the format of the new destination, default is to probe if *mode* is 'existing', else the format of the source

*node-name**
the new block driver state node name in the graph (Since 2.1)

*replaces** with sync=full graph node name to be replaced by the new image when a whole image copy is done. This can be used to repair broken Quorum files. (Since 2.1)

*mode** whether and how QEMU should create a new image, default is 'absolute-paths'.

*speed** the maximum speed, in bytes per second

sync what parts of the disk image should be copied to the destination (all the disk, only the sectors allocated in the topmost image, or only new I/O).

*granularity**
granularity of the dirty bitmap, default is 64K if the image format doesn't have clusters, 4K if the clusters are smaller than that, else the cluster size. Must be a power of 2 between 512 and 64M (since 1.4).

*buf-size** maximum amount of data in flight from source to target (since 1.4).

*on-source-error**
the action to take on an error on the source, default 'report'. 'stop' and 'enospc' can only be used if the block device supports io-status (see BlockInfo).

*on-target-error**
the action to take on an error on the target, default 'report' (no limitations, since this applies to a different block device than *device*).

*unmap** Whether to try to unmap target sectors where source has only zero. If true, and target unallocated sectors will read as zero, target image sectors will be unmapped; otherwise, zeroes will be written. Both will result in identical contents. Default is true. (Since 2.4)

A set of parameters describing drive mirror setup.

Since: 1.3

`BlockDirtyBitmap` *{ 'node':* `str`*, 'name':* `str` *}* [Struct]

node name of device/node which the bitmap is tracking

name name of the dirty bitmap

Since: 2.4

`BlockDirtyBitmapAdd` *{ 'node':* `str`*, 'name':* `str`*, ['granularity':* `uint32`*] }* [Struct]

node name of device/node which the bitmap is tracking

name name of the dirty bitmap

*granularity**
the bitmap granularity, default is 64k for block-dirty-bitmap-add

Since: 2.4

`block-dirty-bitmap-add` (*BlockDirtyBitmapAdd*) [Command]

Create a dirty bitmap with a name on the node, and start tracking the writes.

Returns: nothing on success If *node* is not a valid block device or node, DeviceNotFound If *name* is already taken, GenericError with an explanation

Since: 2.4

Example:

```
-> { "execute": "block-dirty-bitmap-add",
     "arguments": { "node": "drive0", "name": "bitmap0" } }
<- { "return": {} }
```

`block-dirty-bitmap-remove` (*BlockDirtyBitmap*) [Command]

Stop write tracking and remove the dirty bitmap that was created with block-dirty-bitmap-add.

Returns: nothing on success If *node* is not a valid block device or node, DeviceNotFound If *name* is not found, GenericError with an explanation if *name* is frozen by an operation, GenericError

Since: 2.4

Example:

```
-> { "execute": "block-dirty-bitmap-remove",
     "arguments": { "node": "drive0", "name": "bitmap0" } }
<- { "return": {} }
```

`block-dirty-bitmap-clear` (*BlockDirtyBitmap*) [Command]

Clear (reset) a dirty bitmap on the device, so that an incremental backup from this point in time forward will only backup clusters modified after this clear operation.

Returns: nothing on success If *node* is not a valid block device, DeviceNotFound If *name* is not found, GenericError with an explanation

Since: 2.4

Example:

```
-> { "execute": "block-dirty-bitmap-clear",
     "arguments": { "node": "drive0", "name": "bitmap0" } }
<- { "return": {} }
```

`blockdev-mirror` (['*job-id*': `str`], '*device*': `str`, '*target*': `str`, ['*replaces*': `str`], '*sync*': `MirrorSyncMode`, ['*speed*': `int`], ['*granularity*': `uint32`], ['*buf-size*': `int`], ['*on-source-error*': `BlockdevOnError`], ['*on-target-error*': `BlockdevOnError`]) [Command]

*job-id** identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device The device name or node-name of a root node whose writes should be mirrored.

target the id or node-name of the block device to mirror to. This mustn't be attached to guest.

*replaces** with sync=full graph node name to be replaced by the new image when a whole image copy is done. This can be used to repair broken Quorum files.

*speed** the maximum speed, in bytes per second

sync what parts of the disk image should be copied to the destination (all the disk, only the sectors allocated in the topmost image, or only new I/O).

*granularity**
granularity of the dirty bitmap, default is 64K if the image format doesn't have clusters, 4K if the clusters are smaller than that, else the cluster size. Must be a power of 2 between 512 and 64M

*buf-size** maximum amount of data in flight from source to target

*on-source-error**
the action to take on an error on the source, default 'report'. 'stop' and 'enospc' can only be used if the block device supports io-status (see BlockInfo).

*on-target-error**
the action to take on an error on the target, default 'report' (no limitations, since this applies to a different block device than *device*).

Start mirroring a block device's writes to a new destination.

Returns: nothing on success.

Since: 2.6

Example:

```
 -> { "execute": "blockdev-mirror",
      "arguments": { "device": "ide-hd0",
                     "target": "target0",
                     "sync": "full" } }
 <- { "return": {} }
```

`block_set_io_throttle` (*BlockIOThrottle*) [Command]

Change I/O throttle limits for a block drive.

Returns: Nothing on success If *device* is not a valid block device, DeviceNotFound

Since: 1.1

Example:

```
-> { "execute": "block_set_io_throttle",
     "arguments": { "id": "ide0-1-0",
                    "bps": 1000000,
                    "bps_rd": 0,
                    "bps_wr": 0,
                    "iops": 0,
                    "iops_rd": 0,
                    "iops_wr": 0,
                    "bps_max": 8000000,
                    "bps_rd_max": 0,
                    "bps_wr_max": 0,
                    "iops_max": 0,
                    "iops_rd_max": 0,
                    "iops_wr_max": 0,
                    "bps_max_length": 60,
                    "iops_size": 0 } }
<- { "return": {} }
```

`BlockIOThrottle` *{ ['device': `str`], ['id': `str`], 'bps': `int`, 'bps_rd': `int`, 'bps_wr': `int`, 'iops': `int`, 'iops_rd': `int`, 'iops_wr': `int`, ['bps_max': `int`], ['bps_rd_max': `int`], ['bps_wr_max': `int`], ['iops_max': `int`], ['iops_rd_max': `int`], ['iops_wr_max': `int`], ['bps_max_length': `int`], ['bps_rd_max_length': `int`], ['bps_wr_max_length': `int`], ['iops_max_length': `int`], ['iops_rd_max_length': `int`], ['iops_wr_max_length': `int`], ['iops_size': `int`], ['group': `str`] }* [Struct]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

bps total throughput limit in bytes per second

bps_rd read throughput limit in bytes per second

bps_wr write throughput limit in bytes per second

iops total I/O operations per second

iops_rd read I/O operations per second

iops_wr write I/O operations per second

*bps_max** total throughput limit during bursts, in bytes (Since 1.7)

*bps_rd_max**
read throughput limit during bursts, in bytes (Since 1.7)

*bps_wr_max**
write throughput limit during bursts, in bytes (Since 1.7)

*iops_max** total I/O operations per second during bursts, in bytes (Since 1.7)

*iops_rd_max**
: read I/O operations per second during bursts, in bytes (Since 1.7)

*iops_wr_max**
: write I/O operations per second during bursts, in bytes (Since 1.7)

*bps_max_length**
: maximum length of the *bps_max* burst period, in seconds. It must only be set if *bps_max* is set as well. Defaults to 1. (Since 2.6)

*bps_rd_max_length**
: maximum length of the *bps_rd_max* burst period, in seconds. It must only be set if *bps_rd_max* is set as well. Defaults to 1. (Since 2.6)

*bps_wr_max_length**
: maximum length of the *bps_wr_max* burst period, in seconds. It must only be set if *bps_wr_max* is set as well. Defaults to 1. (Since 2.6)

*iops_max_length**
: maximum length of the *iops* burst period, in seconds. It must only be set if *iops_max* is set as well. Defaults to 1. (Since 2.6)

*iops_rd_max_length**
: maximum length of the *iops_rd_max* burst period, in seconds. It must only be set if *iops_rd_max* is set as well. Defaults to 1. (Since 2.6)

*iops_wr_max_length**
: maximum length of the *iops_wr_max* burst period, in seconds. It must only be set if *iops_wr_max* is set as well. Defaults to 1. (Since 2.6)

*iops_size** an I/O size in bytes (Since 1.7)

*group** throttle group name (Since 2.4)

A set of parameters describing block throttling.

Since: 1.1

`block-stream` (['*job-id*': `str`], '*device*': `str`, ['*base*': `str`], ['*backing-file*': `str`], ['*speed*': `int`], ['*on-error*': `BlockdevOnError`]) [Command]

*job-id** identifier for the newly-created block job. If omitted, the device name will be used. (Since 2.7)

device the device name or node-name of a root node

*base** the common backing file name

*backing-file**
: The backing file string to write into the active layer. This filename is not validated.

  If a pathname string is such that it cannot be resolved by QEMU, that means that subsequent QMP or HMP commands must use node-names for the image in question, as filename lookup methods will fail.

  If not specified, QEMU will automatically determine the backing file string to use, or error out if there is no obvious choice. Care should be

taken when specifying the string, to specify a valid filename or protocol. (Since 2.1)

*speed** the maximum speed, in bytes per second

*on-error** the action to take on an error (default report). 'stop' and 'enospc' can only be used if the block device supports io-status (see BlockInfo). Since 1.3.

Copy data from a backing file into a block device.

The block streaming operation is performed in the background until the entire backing file has been copied. This command returns immediately once streaming has started. The status of ongoing block streaming operations can be checked with query-block-jobs. The operation can be stopped before it has completed using the block-job-cancel command.

If a base file is specified then sectors are not copied from that base file and its backing chain. When streaming completes the image file will have the base file as its backing file. This can be used to stream a subset of the backing file chain instead of flattening the entire image.

On successful completion the image file is updated to drop the backing file and the BLOCK_JOB_COMPLETED event is emitted.

Returns: Nothing on success. If *device* does not exist, DeviceNotFound.

Since: 1.1

Example:

```
 -> { "execute": "block-stream",
      "arguments": { "device": "virtio0",
                     "base": "/tmp/master.qcow2" } }
 <- { "return": {} }
```

`block-job-set-speed` (*'device': `str`, 'speed': `int`*) [Command]

device The job identifier. This used to be a device name (hence the name of the parameter), but since QEMU 2.7 it can have other values.

speed the maximum speed, in bytes per second, or 0 for unlimited. Defaults to 0.

Set maximum speed for a background block operation.

This command can only be issued when there is an active block job.

Throttling can be disabled by setting the speed to 0.

Returns: Nothing on success If no background operation is active on this device, DeviceNotActive

Since: 1.1

`block-job-cancel` (*'device': `str`*, [*'force': `bool`*]) [Command]

device The job identifier. This used to be a device name (hence the name of the parameter), but since QEMU 2.7 it can have other values.

*force** whether to allow cancellation of a paused job (default false). Since 1.3.

Stop an active background block operation.

This command returns immediately after marking the active background block operation for cancellation. It is an error to call this command if no operation is in progress.

The operation will cancel as soon as possible and then emit the BLOCK_JOB_CANCELLED event. Before that happens the job is still visible when enumerated using query-block-jobs.

For streaming, the image file retains its backing file unless the streaming operation happens to complete just as it is being cancelled. A new streaming operation can be started at a later time to finish copying all data from the backing file.

Returns: Nothing on success If no background operation is active on this device, DeviceNotActive

Since: 1.1

`block-job-pause` (*'device':* `str`) [Command]

device The job identifier. This used to be a device name (hence the name of the parameter), but since QEMU 2.7 it can have other values.

Pause an active background block operation.

This command returns immediately after marking the active background block operation for pausing. It is an error to call this command if no operation is in progress. Pausing an already paused job has no cumulative effect; a single block-job-resume command will resume the job.

The operation will pause as soon as possible. No event is emitted when the operation is actually paused. Cancelling a paused job automatically resumes it.

Returns: Nothing on success If no background operation is active on this device, DeviceNotActive

Since: 1.3

`block-job-resume` (*'device':* `str`) [Command]

device The job identifier. This used to be a device name (hence the name of the parameter), but since QEMU 2.7 it can have other values.

Resume an active background block operation.

This command returns immediately after resuming a paused background block operation. It is an error to call this command if no operation is in progress. Resuming an already running job is not an error.

This command also clears the error status of the job.

Returns: Nothing on success If no background operation is active on this device, DeviceNotActive

Since: 1.3

`block-job-complete` (*'device':* `str`) [Command]

device The job identifier. This used to be a device name (hence the name of the parameter), but since QEMU 2.7 it can have other values.

Manually trigger completion of an active background block operation. This is supported for drive mirroring, where it also switches the device to write to the target path only. The ability to complete is signaled with a BLOCK_JOB_READY event.

This command completes an active background block operation synchronously. The ordering of this command's return with the BLOCK_JOB_COMPLETED event is not defined. Note that if an I/O error occurs during the processing of this command: 1) the command itself will fail; 2) the error will be processed according to the rerror/werror arguments that were specified when starting the operation.

A cancelled or paused job cannot be completed.

Returns: Nothing on success If no background operation is active on this device, DeviceNotActive

Since: 1.3

`BlockdevDiscardOptions` [Enum]

'`ignore`' Ignore the request

'`unmap`' Forward as an unmap request

Determines how to handle discard requests.

Since: 1.7

`BlockdevDetectZeroesOptions` [Enum]

'`off`' Disabled (default)

'`on`' Enabled

'`unmap`' Enabled and even try to unmap blocks if possible. This requires also that *BlockdevDiscardOptions* is set to unmap for this device.

Describes the operation mode for the automatic conversion of plain zero writes by the OS to driver specific optimized zero write commands.

Since: 2.1

`BlockdevAioOptions` [Enum]

'`threads`' Use qemu's thread pool

'`native`' Use native AIO backend (only Linux and Windows)

Selects the AIO backend to handle I/O requests

Since: 1.7

`BlockdevCacheOptions` *{* ['*direct*': `bool`], ['*no-flush*': `bool`] *}* [Struct]

*direct** enables use of O_DIRECT (bypass the host page cache; default: false)

*no-flush** ignore any flush requests for the device (default: false)

Includes cache-related options for block devices

Since: 1.7

`BlockdevDriver` [Enum]

‘`host_device`,’
host_cdrom: Since 2.1

‘`gluster`’ Since 2.7

Drivers that are supported in block device operations.

Since: 2.0

`BlockdevOptionsFile` *{ 'filename': `str` }* [Struct]

filename path to the image file

Driver specific block device options for the file backend and similar protocols.

Since: 1.7

`BlockdevOptionsNull` *{* [*'size': `int`*], [*'latency-ns': `uint64`*] *}* [Struct]

*size** size of the device in bytes.

*latency-ns**
emulated latency (in nanoseconds) in processing requests. Default to zero which completes requests immediately. (Since 2.4)

Driver specific block device options for the null backend.

Since: 2.2

`BlockdevOptionsVVFAT` *{ 'dir': `str`,* [*'fat-type': `int`*], [*'floppy': `bool`*], [*'label': `str`*], [*'rw': `bool`*] *}* [Struct]

dir directory to be exported as FAT image

*fat-type** FAT type: 12, 16 or 32

*floppy** whether to export a floppy image (true) or partitioned hard disk (false; default)

*label** set the volume label, limited to 11 bytes. FAT16 and FAT32 traditionally have some restrictions on labels, which are ignored by most operating systems. Defaults to "QEMU VVFAT". (since 2.4)

*rw** whether to allow write operations (default: false)

Driver specific block device options for the vvfat protocol.

Since: 1.7

`BlockdevOptionsGenericFormat` *{ 'file': `BlockdevRef` }* [Struct]

file reference to or definition of the data source block device

Driver specific block device options for image format that have no option besides their data source.

Since: 1.7

`BlockdevOptionsLUKS` *{ ['key-secret': `str`] }* [Struct]

*key-secret**
: the ID of a QCryptoSecret object providing the decryption key (since 2.6). Mandatory except when doing a metadata-only probe of the image.

Driver specific block device options for LUKS.

Since: 2.6

`BlockdevOptionsGenericCOWFormat` *{ ['backing': `BlockdevRef`] }* [Struct]

*backing** reference to or definition of the backing file block device (if missing, taken from the image file content). It is allowed to pass an empty string here in order to disable the default backing file.

Driver specific block device options for image format that have no option besides their data source and an optional backing file.

Since: 1.7

`Qcow2OverlapCheckMode` [Enum]

'`none`' Do not perform any checks

'`constant`'
: Perform only checks which can be done in constant time and without reading anything from disk

'`cached`' Perform only checks which can be done without reading anything from disk

'`all`' Perform all available overlap checks

General overlap check modes.

Since: 2.2

`Qcow2OverlapCheckFlags` *{ ['template': `Qcow2OverlapCheckMode`], ['main-header': `bool`], ['active-l1': `bool`], ['active-l2': `bool`], ['refcount-table': `bool`], ['refcount-block': `bool`], ['snapshot-table': `bool`], ['inactive-l1': `bool`], ['inactive-l2': `bool`] }* [Struct]

template Specifies a template mode which can be adjusted using the other flags, defaults to 'cached'

Structure of flags for each metadata structure. Setting a field to 'true' makes qemu guard that structure against unintended overwriting. The default value is chosen according to the template given.

Since: 2.2

`Qcow2OverlapChecks` [*'flags': `Qcow2OverlapCheckFlags`, 'mode': `Qcow2OverlapCheckMode`*] [Alternate]

flags set of flags for separate specification of each metadata structure type

mode named mode which chooses a specific set of flags

Specifies which metadata structures should be guarded against unintended overwriting.

Since: 2.2

`BlockdevOptionsQcow2` *{ ['lazy-refcounts':* `bool`], [Struct]
['pass-discard-request': `bool`], ['pass-discard-snapshot': `bool`],
['pass-discard-other': `bool`], ['overlap-check': `Qcow2OverlapChecks`],
['cache-size': `int`], ['l2-cache-size': `int`], ['refcount-cache-size': `int`],
['cache-clean-interval': `int`] *}*

*lazy-refcounts**
: whether to enable the lazy refcounts feature (default is taken from the image file)

*pass-discard-request**
: whether discard requests to the qcow2 device should be forwarded to the data source

*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

*pass-discard-other**
: whether discard requests for the data source should be issued on other occasions where a cluster gets freed

*overlap-check**
: which overlap checks to perform for writes to the image, defaults to 'cached' (since 2.2)

*cache-size**
: the maximum total size of the L2 table and refcount block caches in bytes (since 2.2)

*l2-cache-size**
: the maximum size of the L2 table cache in bytes (since 2.2)

*refcount-cache-size**
: the maximum size of the refcount block cache in bytes (since 2.2)

*cache-clean-interval**
: clean unused entries in the L2 and refcount caches. The interval is in seconds. The default value is 0 and it disables this feature (since 2.5)

Driver specific block device options for qcow2.

Since: 1.7

`BlockdevOptionsArchipelago` *{ 'volume:* `str`, ['mport': `int`], ['vport': [Struct]
`int`], ['segment': `str`] *}*

volume Name of the Archipelago volume image

*mport** The port number on which mapperd is listening. This is optional and if not specified, QEMU will make Archipelago use the default port (1001).

*vport** The port number on which vlmcd is listening. This is optional and if not specified, QEMU will make Archipelago use the default port (501).

*segment** The name of the shared memory segment Archipelago stack is using. This is optional and if not specified, QEMU will make Archipelago use the default value, 'archipelago'.

Driver specific block device options for Archipelago.

Since: 2.2

`BlkdebugEvent` [Enum]

Trigger events supported by blkdebug.

Since: 2.0

`BlkdebugInjectErrorOptions` *{ 'event':* `BlkdebugEvent`*, ['state':* `int`*], ['errno':* `int`*], ['sector':* `int`*], ['once':* `bool`*], ['immediately':* `bool`*] }* [Struct]

event trigger event

*state** the state identifier blkdebug needs to be in to actually trigger the event; defaults to "any"

*errno** error identifier (errno) to be returned; defaults to EIO

*sector** specifies the sector index which has to be affected in order to actually trigger the event; defaults to "any sector"

*once** disables further events after this one has been triggered; defaults to false

*immediately**
fail immediately; defaults to false

Describes a single error injection for blkdebug.

Since: 2.0

`BlkdebugSetStateOptions` *{ 'event':* `BlkdebugEvent`*, ['state':* `int`*], 'new_state':* `int` *}* [Struct]

event trigger event

*state** the current state identifier blkdebug needs to be in; defaults to "any"

new_state the state identifier blkdebug is supposed to assume if this event is triggered

Describes a single state-change event for blkdebug.

Since: 2.0

`BlockdevOptionsBlkdebug` *{ 'image':* `BlockdevRef`*, ['config':* `str`*], ['align':* `int`*], ['inject-error': ['*`BlkdebugInjectErrorOptions`*']], ['set-state': ['*`BlkdebugSetStateOptions`*']] }* [Struct]

image underlying raw block device (or image file)

*config** filename of the configuration file

*align** required alignment for requests in bytes, must be power of 2, or 0 for default

*inject-error** array of error injection descriptions

*set-state** array of state-change descriptions

Driver specific block device options for blkdebug.

Since: 2.0

`BlockdevOptionsBlkverify` *{ 'test':* `BlockdevRef`*, 'raw':* `BlockdevRef` *}* [Struct]

test block device to be tested

raw raw image used for verification

Driver specific block device options for blkverify.

Since: 2.0

`QuorumReadPattern` [Enum]

'`quorum`' read all the children and do a quorum vote on reads

'`fifo`' read only from the first child that has not failed

An enumeration of quorum read patterns.

Since: 2.2

`BlockdevOptionsQuorum` *{* ['*blkverify':* `bool`], *'children':* ['`BlockdevRef`'], *'vote-threshold':* `int`, ['*rewrite-corrupted':* `bool`], ['*read-pattern':* `QuorumReadPattern`] *}* [Struct]

*blkverify** true if the driver must print content mismatch set to false by default

children the children block devices to use

vote-threshold
the vote limit under which a read will fail

*rewrite-corrupted**
rewrite corrupted data when quorum is reached (Since 2.1)

*read-pattern**
choose read pattern and set to quorum by default (Since 2.2)

Driver specific block device options for Quorum

Since: 2.0

`GlusterTransport` [Enum]

'`tcp`' TCP - Transmission Control Protocol

'`unix`' UNIX - Unix domain socket

An enumeration of Gluster transport types

Since: 2.7

`GlusterServer` [*'unix':* `UnixSocketAddress`, *'tcp':* `InetSocketAddress`] [Union]

type Transport type used for gluster connection

unix socket file

tcp host address and port number

This is similar to SocketAddress, only distinction:

1. GlusterServer is a flat union, SocketAddress is a simple union. A flat union is nicer than simple because it avoids nesting (i.e. more {}) on the wire.
2. GlusterServer lacks case 'fd', since gluster doesn't let you pass in a file descriptor.

GlusterServer is actually not Gluster-specific, its a compatibility evolved into an alternate for SocketAddress.

Captures the address of a socket

Details for connecting to a gluster server

Since: 2.7

`BlockdevOptionsGluster` *{ 'volume':* `str`, *'path':* `str`, *'server':* [*'*`GlusterServer`*'*], [*'debug-level':* `int`], [*'logfile':* `str`] *}* [Struct]

volume name of gluster volume where VM image resides

path absolute path to image file in gluster volume

server gluster servers description

*debug-level** libgfapi log level (default '4' which is Error)

*logfile** libgfapi log file (default /dev/stderr)

Driver specific block device options for Gluster

Since: 2.7

`ReplicationMode` [Enum]

'`primary`' Primary mode, the vm's state will be sent to secondary QEMU.

'`secondary`' Secondary mode, receive the vm's state from primary QEMU.

An enumeration of replication modes.

Since: 2.8

`BlockdevOptionsReplication` *{ 'mode':* `ReplicationMode`, [*'top-id':* `str`] *}* [Struct]

mode the replication mode

*top-id** In secondary mode, node name or device ID of the root node who owns the replication node chain. Ignored in primary mode.

Driver specific block device options for replication

Since: 2.8

`BlockdevOptions` [*'archipelago':* `BlockdevOptionsArchipelago`, *'blkdebug':* `BlockdevOptionsBlkdebug`, *'blkverify':* `BlockdevOptionsBlkverify`, *'bochs':* `BlockdevOptionsGenericFormat`, *'cloop':* `BlockdevOptionsGenericFormat`, *'dmg':* `BlockdevOptionsGenericFormat`, *'file':* `BlockdevOptionsFile`, *'ftp':* `BlockdevOptionsFile`, *'ftps':* `BlockdevOptionsFile`, *'gluster':* `BlockdevOptionsGluster`, *'host_cdrom':* `BlockdevOptionsFile`, *'host_device':* `BlockdevOptionsFile`, *'http':* `BlockdevOptionsFile`, *'https':* `BlockdevOptionsFile`, *'luks':* `BlockdevOptionsLUKS`, *'null-aio':* `BlockdevOptionsNull`, *'null-co':* `BlockdevOptionsNull`, *'parallels':* `BlockdevOptionsGenericFormat`, *'qcow2':* `BlockdevOptionsQcow2`, *'qcow':* `BlockdevOptionsGenericCOWFormat`, *'qed':* `BlockdevOptionsGenericCOWFormat`, *'quorum':* `BlockdevOptionsQuorum`, *'raw':* `BlockdevOptionsGenericFormat`, *'replication':* `BlockdevOptionsReplication`, *'tftp':* `BlockdevOptionsFile`, *'vdi':* `BlockdevOptionsGenericFormat`, *'vhdx':* `BlockdevOptionsGenericFormat`, *'vmdk':* `BlockdevOptionsGenericCOWFormat`, *'vpc':* `BlockdevOptionsGenericFormat`, *'vvfat':* `BlockdevOptionsVVFAT`] [Union]

driver block driver name

*node-name** the node name of the new node (Since 2.0). This option is required on the top level of blockdev-add.

*discard** discard-related options (default: ignore)

*cache** cache-related options

*aio** AIO backend (default: threads)

*read-only** whether the block device should be read-only (default: false)

*detect-zeroes** detect and optimize zero writes (Since 2.1) (default: off)

Remaining options are determined by the block driver.

Options for creating a block device. Many options are available for all block devices, independent of the block driver:

Since: 1.7

`BlockdevRef` [*'definition':* `BlockdevOptions`, *'reference':* `str`] [Alternate]

definition defines a new block device inline

reference references the ID of an existing block device. An empty string means that no block device should be referenced.

Reference to a block device.

Since: 1.7

`blockdev-add` (*'options': `BlockdevOptions`*) [Command]

options block device options for the new device

Creates a new block device. If the *id* option is given at the top level, a BlockBackend will be created; otherwise, *node*-name is mandatory at the top level and no BlockBackend will be created.

Note: This command is still a work in progress. It doesn't support all block drivers among other things. Stay away from it unless you want to help with its development.

Since: 1.7

Example:

```
1.
-> { "execute": "blockdev-add",
     "arguments": {
         "options" : { "driver": "qcow2",
                       "file": { "driver": "file",
                                 "filename": "test.qcow2" } } } }
<- { "return": {} }

2.
-> { "execute": "blockdev-add",
     "arguments": {
         "options": {
           "driver": "qcow2",
           "node-name": "node0",
           "discard": "unmap",
           "cache": {
               "direct": true,
               "writeback": true
           },
           "file": {
               "driver": "file",
               "filename": "/tmp/test.qcow2"
           },
           "backing": {
               "driver": "raw",
               "file": {
                   "driver": "file",
                   "filename": "/dev/fdset/4"
               }
           }
         }
       }

<- { "return": {} }
```

`x-blockdev-del` (*'node-name':* `str`) [Command]

node-name
: Name of the graph node to delete.

Deletes a block device that has been added using blockdev-add. The command will fail if the node is attached to a device or is otherwise being used.

This command is still a work in progress and is considered experimental. Stay away from it unless you want to help with its development.

Since: 2.5

Example:

```
 -> { "execute": "blockdev-add",
      "arguments": {
          "options": {
              "driver": "qcow2",
              "node-name": "node0",
              "file": {
                  "driver": "file",
                  "filename": "test.qcow2"
              }
          }
      }
   }
 <- { "return": {} }

 -> { "execute": "x-blockdev-del",
      "arguments": { "node-name": "node0" }
   }
 <- { "return": {} }
```

`blockdev-open-tray` (['*device':* `str`], ['*id':* `str`], ['*force':* `bool`]) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

*force** if false (the default), an eject request will be sent to the guest if it has locked the tray (and the tray will not be opened immediately); if true, the tray will be opened regardless of whether it is locked

Opens a block device's tray. If there is a block driver state tree inserted as a medium, it will become inaccessible to the guest (but it will remain associated to the block device, so closing the tray will make it accessible again).

If the tray was already open before, this will be a no-op.

Once the tray opens, a DEVICE_TRAY_MOVED event is emitted. There are cases in which no such event will be generated, these include:

- if the guest has locked the tray, *force* is false and the guest does not respond to the eject request
- if the BlockBackend denoted by *device* does not have a guest device attached to it

- if the guest device does not have an actual tray

Since: 2.5

Example:

```
-> { "execute": "blockdev-open-tray",
     "arguments": { "id": "ide0-1-0" } }

<- { "timestamp": { "seconds": 1418751016,
                    "microseconds": 716996 },
     "event": "DEVICE_TRAY_MOVED",
     "data": { "device": "ide1-cd0",
               "tray-open": true } }

<- { "return": {} }
```

`blockdev-close-tray` (['*device*': `str`], ['*id*': `str`]) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

Closes a block device's tray. If there is a block driver state tree associated with the block device (which is currently ejected), that tree will be loaded as the medium.

If the tray was already closed before, this will be a no-op.

Since: 2.5

Example:

```
-> { "execute": "blockdev-close-tray",
     "arguments": { "id": "ide0-1-0" } }

<- { "timestamp": { "seconds": 1418751345,
                    "microseconds": 272147 },
     "event": "DEVICE_TRAY_MOVED",
     "data": { "device": "ide1-cd0",
               "tray-open": false } }

<- { "return": {} }
```

`x-blockdev-remove-medium` (['*device*': `str`], ['*id*': `str`]) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

Removes a medium (a block driver state tree) from a block device. That block device's tray must currently be open (unless there is no attached guest device).

If the tray is open and there is no medium inserted, this will be a no-op.

This command is still a work in progress and is considered experimental. Stay away from it unless you want to help with its development.

Since: 2.5

Example:

```
-> { "execute": "x-blockdev-remove-medium",
     "arguments": { "id": "ide0-1-0" } }

<- { "error": { "class": "GenericError",
                "desc": "Tray of device 'ide0-1-0' is not open" } }

-> { "execute": "blockdev-open-tray",
     "arguments": { "id": "ide0-1-0" } }

<- { "timestamp": { "seconds": 1418751627,
                    "microseconds": 549958 },
     "event": "DEVICE_TRAY_MOVED",
     "data": { "device": "ide1-cd0",
               "tray-open": true } }

<- { "return": {} }

-> { "execute": "x-blockdev-remove-medium",
     "arguments": { "device": "ide0-1-0" } }

<- { "return": {} }
```

`x-blockdev-insert-medium` (['device': **str**], ['id': **str**], 'node-name': **str**) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

node-name
name of a node in the block driver state graph

Inserts a medium (a block driver state tree) into a block device. That block device's tray must currently be open (unless there is no attached guest device) and there must be no medium inserted already.

This command is still a work in progress and is considered experimental. Stay away from it unless you want to help with its development.

Since: 2.5

Example:

```
-> { "execute": "blockdev-add",
     "arguments": {
         "options": { "node-name": "node0",
                      "driver": "raw",
                      "file": { "driver": "file",
                                "filename": "fedora.iso" } } } }
<- { "return": {} }
```

```
-> { "execute": "x-blockdev-insert-medium",
     "arguments": { "id": "ide0-1-0",
                    "node-name": "node0" } }

<- { "return": {} }
```

`BlockdevChangeReadOnlyMode` [Enum]

'`blockdev-change-medium`' command.

'`retain`' Retains the current read-only mode

'`read-only`' Makes the device read-only

'`read-write`' Makes the device writable

Specifies the new read-only mode of a block device subject to the

Since: 2.3

`blockdev-change-medium` (['*device*': `str`], ['*id*': `str`], '*filename*': `str`, ['*format*': `str`], ['*read-only-mode*': `BlockdevChangeReadOnlyMode`]) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

filename filename of the new image to be loaded

*format** format to open the new image with (defaults to the probed format)

*read-only-mode** change the read-only mode of the device; defaults to 'retain'

Changes the medium inserted into a block device by ejecting the current medium and loading a new image file which is inserted as the new medium (this command combines blockdev-open-tray, x-blockdev-remove-medium, x-blockdev-insert-medium and blockdev-close-tray).

Since: 2.5

Examples:

1. Change a removable medium

```
-> { "execute": "blockdev-change-medium",
     "arguments": { "id": "ide0-1-0",
                    "filename": "/srv/images/Fedora-12-x86_64-DVD.iso",
                    "format": "raw" } }
<- { "return": {} }
```

2. Load a read-only medium into a writable drive

```
-> { "execute": "blockdev-change-medium",
```

```
          "arguments": { "id": "floppyA",
                         "filename": "/srv/images/ro.img",
                         "format": "raw",
                         "read-only-mode": "retain" } }

     <- { "error":
          { "class": "GenericError",
            "desc": "Could not open '/srv/images/ro.img': Permission denied" } }

     -> { "execute": "blockdev-change-medium",
          "arguments": { "id": "floppyA",
                         "filename": "/srv/images/ro.img",
                         "format": "raw",
                         "read-only-mode": "read-only" } }

     <- { "return": {} }
```

`BlockErrorAction` [Enum]

‘`ignore`’ error has been ignored

‘`report`’ error has been reported to the device

‘`stop`’ error caused VM to be stopped

An enumeration of action that has been taken when a DISK I/O occurs

Since: 2.1

`BLOCK_IMAGE_CORRUPTED` (*'device'*: `str`, [*'node-name'*: `str`], *'msg'*: `str`, [*'offset'*: `int`], [*'size'*: `int`], *'fatal'*: `bool`) [Event]

device device name. This is always present for compatibility reasons, but it can be empty (`""`) if the image does not have a device name associated.

*node-name**

node name (Since: 2.4)

msg informative message for human consumption, such as the kind of corruption being detected. It should not be parsed by machine as it is not guaranteed to be stable

*offset** if the corruption resulted from an image access, this is the host's access offset into the image

*size** if the corruption resulted from an image access, this is the access size

fatal if set, the image is marked corrupt and therefore unusable after this event and must be repaired (Since 2.2; before, every BLOCK_IMAGE_CORRUPTED event was fatal)

Emitted when a disk image is being marked corrupt. The image can be identified by its device or node name. The 'device' field is always present for compatibility reasons, but it can be empty (`""`) if the image does not have a device name associated.

Since: 1.7

Example:

```
<- { "event": "BLOCK_IMAGE_CORRUPTED",
     "data": { "device": "ide0-hd0", "node-name": "node0",
               "msg": "Prevented active L1 table overwrite", "offset": 196608,
               "size": 65536 },
     "timestamp": { "seconds": 1378126126, "microseconds": 966463 } }
```

BLOCK_IO_ERROR (*'device': `str`, 'operation': `IoOperationType`, 'action': `BlockErrorAction`, ['nospace': `bool`], 'reason': `str`*) [Event]

device device name

operation I/O operation

action action that has been taken

*nospace** true if I/O error was caused due to a no-space condition. This key is only present if query-block's io-status is present, please see query-block documentation for more information (since: 2.2)

reason human readable string describing the error cause. (This field is a debugging aid for humans, it should not be parsed by applications) (since: 2.2)

Emitted when a disk I/O error occurs

Note: If action is "stop", a STOP event will eventually follow the BLOCK_IO_ERROR event

Since: 0.13.0

Example:

```
<- { "event": "BLOCK_IO_ERROR",
     "data": { "device": "ide0-hd1",
               "operation": "write",
               "action": "stop" },
     "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

BLOCK_JOB_COMPLETED (*'type': `BlockJobType`, 'device': `str`, 'len': `int`, 'offset': `int`, 'speed': `int`, ['error': `str`]*) [Event]

type job type

device The job identifier. Originally the device name but other values are allowed since QEMU 2.7

len maximum progress value

offset current progress value. On success this is equal to len. On failure this is less than len

speed rate limit, bytes per second

*error** error message. Only present on failure. This field contains a human-readable error message. There are no semantics other than that streaming has failed and clients should not try to interpret the error string

Emitted when a block job has completed

Since: 1.1

Example:

```
<- { "event": "BLOCK_JOB_COMPLETED",
     "data": { "type": "stream", "device": "virtio-disk0",
               "len": 10737418240, "offset": 10737418240,
               "speed": 0 },
     "timestamp": { "seconds": 1267061043, "microseconds": 959568 } }
```

BLOCK_JOB_CANCELLED (*'type':* `BlockJobType`*, 'device':* `str`*, 'len':* `int`*, 'offset':* `int`*, 'speed':* `int`) [Event]

type job type

device The job identifier. Originally the device name but other values are allowed since QEMU 2.7

len maximum progress value

offset current progress value. On success this is equal to len. On failure this is less than len

speed rate limit, bytes per second

Emitted when a block job has been cancelled

Since: 1.1

Example:

```
<- { "event": "BLOCK_JOB_CANCELLED",
     "data": { "type": "stream", "device": "virtio-disk0",
               "len": 10737418240, "offset": 134217728,
               "speed": 0 },
     "timestamp": { "seconds": 1267061043, "microseconds": 959568 } }
```

BLOCK_JOB_ERROR (*'device':* `str`*, 'operation':* `IoOperationType`*, 'action':* `BlockErrorAction`) [Event]

device The job identifier. Originally the device name but other values are allowed since QEMU 2.7

operation I/O operation

action action that has been taken

Emitted when a block job encounters an error

Since: 1.3

Example:

```
<- { "event": "BLOCK_JOB_ERROR",
     "data": { "device": "ide0-hd1",
               "operation": "write",
               "action": "stop" },
     "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

`BLOCK_JOB_READY` (*'type':* `BlockJobType`*, 'device':* `str`*, 'len':* `int`*, 'offset':* `int`*, 'speed':* `int`) [Event]

type job type

device The job identifier. Originally the device name but other values are allowed since QEMU 2.7

len maximum progress value

offset current progress value. On success this is equal to len. On failure this is less than len

speed rate limit, bytes per second

Emitted when a block job is ready to complete

Note: The "ready to complete" status is always reset by a *BLOCK_JOB_ERROR* event

Since: 1.3

Example:

```
<- { "event": "BLOCK_JOB_READY",
     "data": { "device": "drive0", "type": "mirror", "speed": 0,
               "len": 2097152, "offset": 2097152 }
     "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

`PreallocMode` [Enum]

'`off`' no preallocation

'`metadata`' preallocate only for metadata

'`falloc`' like *full* preallocation but allocate disk space by posix_fallocate() rather than writing zeros.

'`full`' preallocate all data by writing zeros to device to ensure disk space is really available. *full* preallocation also sets up metadata correctly.

Preallocation mode of QEMU image file

Since: 2.2

`BLOCK_WRITE_THRESHOLD` (*'node-name':* `str`*, 'amount-exceeded':* `uint64`*, 'write-threshold':* `uint64`) [Event]

node-name graph node name on which the threshold was exceeded.

amount-exceeded amount of data which exceeded the threshold, in bytes.

write-threshold last configured threshold, in bytes.

Emitted when writes on block device reaches or exceeds the configured write threshold. For thin-provisioned devices, this means the device should be extended to avoid pausing for disk exhaustion. The event is one shot. Once triggered, it needs to be re-registered with another block-set-threshold command.

Since: 2.3

`block-set-write-threshold` (*'node-name':* `str`, *'write-threshold':* `uint64`) [Command]

node-name
: graph node name on which the threshold must be set.

write-threshold
: configured threshold for the block device, bytes. Use 0 to disable the threshold.

Change the write threshold for a block drive. An event will be delivered if a write to this block drive crosses the configured threshold. The threshold is an offset, thus must be non-negative. Default is no write threshold. Setting the threshold to zero disables it.

This is useful to transparently resize thin-provisioned drives without the guest OS noticing.

Since: 2.3

Example:

```
-> { "execute": "block-set-write-threshold",
     "arguments": { "node-name": "mydev",
                    "write-threshold": 17179869184 } }
<- { "return": {} }
```

`x-blockdev-change` (*'parent':* `str`, [*'child':* `str`], [*'node':* `str`]) [Command]

parent the id or name of the parent node.

*child** the name of a child under the given parent node.

*node** the name of the node that will be added.

Dynamically reconfigure the block driver state graph. It can be used to add, remove, insert or replace a graph node. Currently only the Quorum driver implements this feature to add or remove its child. This is useful to fix a broken quorum child.

If *node* is specified, it will be inserted under *parent*. *child* may not be specified in this case. If both *parent* and *child* are specified but *node* is not, *child* will be detached from *parent*.

Note: this command is experimental, and its API is not stable. It does not support all kinds of operations, all kinds of children, nor all block drivers.

Warning: The data in a new quorum child MUST be consistent with that of the rest of the array.

Since: 2.7

Example:

```
1. Add a new node to a quorum
-> { "execute": "blockdev-add",
     "arguments": {
         "options": { "driver": "raw",
                      "node-name": "new_node",
                       "file": { "driver": "file",
```

```
                                    "filename": "test.raw" } } } }
<- { "return": {} }
-> { "execute": "x-blockdev-change",
     "arguments": { "parent": "disk1",
                    "node": "new_node" } }
<- { "return": {} }

2. Delete a quorum's node
-> { "execute": "x-blockdev-change",
     "arguments": { "parent": "disk1",
                    "child": "children.1" } }
<- { "return": {} }
```

`BiosAtaTranslation` [Enum]

'`auto`' If cylinder/heads/sizes are passed, choose between none and LBA depending on the size of the disk. If they are not passed, choose none if QEMU can guess that the disk had 16 or fewer heads, large if QEMU can guess that the disk had 131072 or fewer tracks across all heads (i.e. cylinders*heads<131072), otherwise LBA.

'`none`' The physical disk geometry is equal to the logical geometry.

'`lba`' Assume 63 sectors per track and one of 16, 32, 64, 128 or 255 heads (if fewer than 255 are enough to cover the whole disk with 1024 cylinders/head). The number of cylinders/head is then computed based on the number of sectors and heads.

'`large`' The number of cylinders per head is scaled down to 1024 by correspondingly scaling up the number of heads.

'`rechs`' Same as *large*, but first convert a 16-head geometry to 15-head, by proportionally scaling up the number of cylinders/head.

Policy that BIOS should use to interpret cylinder/head/sector addresses. Note that Bochs BIOS and SeaBIOS will not actually translate logical CHS to physical; instead, they will use logical block addressing.

Since: 2.0

`FloppyDriveType` [Enum]

'`144`' 1.44MB 3.5" drive

'`288`' 2.88MB 3.5" drive

'`120`' 1.2MB 5.25" drive

'`none`' No drive connected

'`auto`' Automatically determined by inserted media at boot

Type of Floppy drive to be emulated by the Floppy Disk Controller.

Since: 2.6

`BlockdevSnapshotInternal` *{ 'device': `str`, 'name': `str` }* [Struct]

device the device name or node-name of a root node to generate the snapshot from

name the name of the internal snapshot to be created

Notes: In transaction, if *name* is empty, or any snapshot matching *name* exists, the operation will fail. Only some image formats support it, for example, qcow2, rbd, and sheepdog.

Since: 1.7

`blockdev-snapshot-internal-sync` (*BlockdevSnapshotInternal*) [Command]

Synchronously take an internal snapshot of a block device, when the format of the image used supports it. If the name is an empty string, or a snapshot with name already exists, the operation will fail.

For the arguments, see the documentation of BlockdevSnapshotInternal.

Returns: nothing on success

If *device* is not a valid block device, GenericError

If any snapshot matching *name* exists, or *name* is empty, GenericError

If the format of the image used does not support it, BlockFormatFeatureNotSupported

Since: 1.7

Example:

```
-> { "execute": "blockdev-snapshot-internal-sync",
     "arguments": { "device": "ide-hd0",
                    "name": "snapshot0" }
   }
<- { "return": {} }
```

`SnapshotInfo blockdev-snapshot-delete-internal-sync` [Command]
(*'device': `str`*, [*'id': `str`*], [*'name': `str`*])

device the device name or node-name of a root node to delete the snapshot from

id optional the snapshot's ID to be deleted

name optional the snapshot's name to be deleted

Synchronously delete an internal snapshot of a block device, when the format of the image used support it. The snapshot is identified by name or id or both. One of the name or id is required. Return SnapshotInfo for the successfully deleted snapshot.

Returns: SnapshotInfo on success If *device* is not a valid block device, GenericError If snapshot not found, GenericError If the format of the image used does not support it, BlockFormatFeatureNotSupported If *id* and *name* are both not specified, GenericError

Since: 1.7

Example:

```
-> { "execute": "blockdev-snapshot-delete-internal-sync",
     "arguments": { "device": "ide-hd0",
```

```
                          "name": "snapshot0" }
      }
     <- { "return": {
                         "id": "1",
                         "name": "snapshot0",
                         "vm-state-size": 0,
                         "date-sec": 1000012,
                         "date-nsec": 10,
                         "vm-clock-sec": 100,
                         "vm-clock-nsec": 20
        }
      }
```

`eject` (['*device*': `str`], ['*id*': `str`], ['*force*': `bool`]) [Command]

*device** Block device name (deprecated, use *id* instead)

*id** The name or QOM path of the guest device (since: 2.8)

*force** If true, eject regardless of whether the drive is locked. If not specified, the default value is false.

Ejects a device from a removable drive.

Returns: Nothing on success

If *device* is not a valid block device, DeviceNotFound

Notes: Ejecting a device with no media results in success

Since: 0.14.0

Example:

```
 -> { "execute": "eject", "arguments": { "device": "ide1-0-1" } }
 <- { "return": {} }
```

`nbd-server-start` ('*addr*': `SocketAddress`, ['*tls-creds*': `str`]) [Command]

addr Address on which to listen.

tls-creds (optional) ID of the TLS credentials object. Since 2.6

Start an NBD server listening on the given host and port. Block devices can then be exported using *nbd*-server-add. The NBD server will present them as named exports; for example, another QEMU instance could refer to them as "nbd:HOST:PORT:exportname=NAME".

Returns: error if the server is already running.

Since: 1.3.0

`nbd-server-add` ('*device*': `str`, ['*writable*': `bool`]) [Command]

device The device name or node name of the node to be exported

writable Whether clients should be able to write to the device via the NBD connection (default false). #optional

Export a block node to QEMU's embedded NBD server.

Returns: error if the device is already marked for export.

Since: 1.3.0

`nbd-server-stop` () [Command]

Stop QEMU's embedded NBD server, and unregister all devices previously added via *nbd*-server-add.

Since: 1.3.0

`DEVICE_TRAY_MOVED` (*'device':* `str`*, 'tray-open':* `bool`) [Event]

device device name

tray-open true if the tray has been opened or false if it has been closed

Emitted whenever the tray of a removable device is moved by the guest or by HMP/QMP commands

Since: 1.1

Example:

```
<- { "event": "DEVICE_TRAY_MOVED",
     "data": { "device": "ide1-cd0",
               "tray-open": true
     },
     "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

`QuorumOpType` [Enum]

'`read`' read operation

'`write`' write operation

'`flush`' flush operation

An enumeration of the quorum operation types

Since: 2.6

2.6 Events

`SHUTDOWN` () [Event]

Emitted when the virtual machine has shut down, indicating that qemu is about to exit.

Note: If the command-line option "-no-shutdown" has been specified, qemu will not exit, and a STOP event will eventually follow the SHUTDOWN event

Since: 0.12.0

Example:

```
<- { "event": "SHUTDOWN",
     "timestamp": { "seconds": 1267040730, "microseconds": 682951 } }
```

`POWERDOWN` () [Event]

Emitted when the virtual machine is powered down through the power control system, such as via ACPI.

Since: 0.12.0

Example:

```
<- { "event": "POWERDOWN",
     "timestamp": { "seconds": 1267040730, "microseconds": 682951 } }
```

RESET () [Event]

Emitted when the virtual machine is reset

Since: 0.12.0

Example:

```
<- { "event": "RESET",
     "timestamp": { "seconds": 1267041653, "microseconds": 9518 } }
```

STOP () [Event]

Emitted when the virtual machine is stopped

Since: 0.12.0

Example:

```
<- { "event": "STOP",
     "timestamp": { "seconds": 1267041730, "microseconds": 281295 } }
```

RESUME () [Event]

Emitted when the virtual machine resumes execution

Since: 0.12.0

Example:

```
<- { "event": "RESUME",
     "timestamp": { "seconds": 1271770767, "microseconds": 582542 } }
```

SUSPEND () [Event]

Emitted when guest enters a hardware suspension state, for example, S3 state, which is sometimes called standby state

Since: 1.1

Example:

```
<- { "event": "SUSPEND",
     "timestamp": { "seconds": 1344456160, "microseconds": 309119 } }
```

SUSPEND_DISK () [Event]

Emitted when guest enters a hardware suspension state with data saved on disk, for example, S4 state, which is sometimes called hibernate state

Note: QEMU shuts down (similar to event *SHUTDOWN*) when entering this state

Since: 1.2

Example:

```
<-   { "event": "SUSPEND_DISK",
       "timestamp": { "seconds": 1344456160, "microseconds": 309119 } }
```

WAKEUP () [Event]

Emitted when the guest has woken up from suspend state and is running

Since: 1.1

Example:

```
<- { "event": "WAKEUP",
     "timestamp": { "seconds": 1344522075, "microseconds": 745528 } }
```

`RTC_CHANGE` (*'offset':* `int`) [Event]

offset offset between base RTC clock (as specified by -rtc base), and new RTC clock value

Emitted when the guest changes the RTC time.

Note: This event is rate-limited.

Since: 0.13.0

Example:

```
<-   { "event": "RTC_CHANGE",
       "data": { "offset": 78 },
       "timestamp": { "seconds": 1267020223, "microseconds": 435656 } }
```

`WATCHDOG` (*'action':* `WatchdogExpirationAction`) [Event]

action action that has been taken

Emitted when the watchdog device's timer is expired

Note: This event is rate-limited.

Since: 0.13.0

Example:

```
<- { "event": "WATCHDOG",
     "data": { "action": "reset" },
     "timestamp": { "seconds": 1267061043, "microseconds": 959568 } }
```

`DEVICE_DELETED` ([*'device':* `str`], *'path':* `str`) [Event]

*device** device name

path device path

Emitted whenever the device removal completion is acknowledged by the guest. At this point, it's safe to reuse the specified device ID. Device removal can be initiated by the guest or by HMP/QMP commands.

Since: 1.5

Example:

```
<- { "event": "DEVICE_DELETED",
     "data": { "device": "virtio-net-pci-0",
               "path": "/machine/peripheral/virtio-net-pci-0" },
     "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

`NIC_RX_FILTER_CHANGED` ([*'name':* `str`], *'path':* `str`) [Event]

*name** net client name

path device path

Emitted once until the 'query-rx-filter' command is executed, the first event will always be emitted

Since: 1.6

Example:

```
<- { "event": "NIC_RX_FILTER_CHANGED",
```

```
        "data": { "name": "vnet0",
                  "path": "/machine/peripheral/vnet0/virtio-backend" },
        "timestamp": { "seconds": 1368697518, "microseconds": 326866 } }
      }
```

VNC_CONNECTED (*'server': VncServerInfo, 'client': VncBasicInfo*) [Event]

server server information

client client information

Emitted when a VNC client establishes a connection

Note: This event is emitted before any authentication takes place, thus the authentication ID is not provided

Since: 0.13.0

Example:

```
<- { "event": "VNC_CONNECTED",
     "data": {
           "server": { "auth": "sasl", "family": "ipv4",
                       "service": "5901", "host": "0.0.0.0" },
           "client": { "family": "ipv4", "service": "58425",
                       "host": "127.0.0.1" } },
     "timestamp": { "seconds": 1262976601, "microseconds": 975795 } }
```

VNC_INITIALIZED (*'server': VncServerInfo, 'client': VncClientInfo*) [Event]

server server information

client client information

Emitted after authentication takes place (if any) and the VNC session is made active

Since: 0.13.0

Example:

```
<-  { "event": "VNC_INITIALIZED",
      "data": {
           "server": { "auth": "sasl", "family": "ipv4",
                       "service": "5901", "host": "0.0.0.0"},
           "client": { "family": "ipv4", "service": "46089",
                       "host": "127.0.0.1", "sasl_username": "luiz" } },
      "timestamp": { "seconds": 1263475302, "microseconds": 150772 } }
```

VNC_DISCONNECTED (*'server': VncServerInfo, 'client': VncClientInfo*) [Event]

server server information

client client information

Emitted when the connection is closed

Since: 0.13.0

Example:

```
<- { "event": "VNC_DISCONNECTED",
```

```
        "data": {
              "server": { "auth": "sasl", "family": "ipv4",
                          "service": "5901", "host": "0.0.0.0" },
              "client": { "family": "ipv4", "service": "58425",
                          "host": "127.0.0.1", "sasl_username": "luiz" } },
        "timestamp": { "seconds": 1262976601, "microseconds": 975795 } }
```

SPICE_CONNECTED (*'server': SpiceBasicInfo, 'client': SpiceBasicInfo*) [Event]

server server information

client client information

Emitted when a SPICE client establishes a connection

Since: 0.14.0

Example:

```
 <- { "timestamp": {"seconds": 1290688046, "microseconds": 388707},
      "event": "SPICE_CONNECTED",
      "data": {
        "server": { "port": "5920", "family": "ipv4", "host": "127.0.0.1"},
        "client": {"port": "52873", "family": "ipv4", "host": "127.0.0.1"}
    }}
```

SPICE_INITIALIZED (*'server': SpiceServerInfo, 'client': SpiceChannel*) [Event]

server server information

client client information

Emitted after initial handshake and authentication takes place (if any) and the SPICE channel is up and running

Since: 0.14.0

Example:

```
 <- { "timestamp": {"seconds": 1290688046, "microseconds": 417172},
      "event": "SPICE_INITIALIZED",
      "data": {"server": {"auth": "spice", "port": "5921",
                          "family": "ipv4", "host": "127.0.0.1"},
                "client": {"port": "49004", "family": "ipv4", "channel-type": 3,
                          "connection-id": 1804289383, "host": "127.0.0.1",
                          "channel-id": 0, "tls": true}
    }}
```

SPICE_DISCONNECTED (*'server': SpiceBasicInfo, 'client': SpiceBasicInfo*) [Event]

server server information

client client information

Emitted when the SPICE connection is closed

Since: 0.14.0

Example:

```
<- { "timestamp": {"seconds": 1290688046, "microseconds": 388707},
     "event": "SPICE_DISCONNECTED",
     "data": {
       "server": { "port": "5920", "family": "ipv4", "host": "127.0.0.1"},
       "client": {"port": "52873", "family": "ipv4", "host": "127.0.0.1"}
   }}
```

SPICE_MIGRATE_COMPLETED () [Event]

Emitted when SPICE migration has completed

Since: 1.3

Example:

```
<- { "timestamp": {"seconds": 1290688046, "microseconds": 417172},
     "event": "SPICE_MIGRATE_COMPLETED" }
```

MIGRATION (*'status':* ***MigrationStatus***) [Event]

status *MigrationStatus* describing the current migration status.

Emitted when a migration event happens

Since: 2.4

Example:

```
<- {"timestamp": {"seconds": 1432121972, "microseconds": 744001},
    "event": "MIGRATION",
    "data": {"status": "completed"} }
```

MIGRATION_PASS (*'pass':* ***int***) [Event]

pass An incrementing count (starting at 1 on the first pass)

Emitted from the source side of a migration at the start of each pass (when it syncs the dirty bitmap)

Since: 2.6

Example:

```
{ "timestamp": {"seconds": 1449669631, "microseconds": 239225},
  "event": "MIGRATION_PASS", "data": {"pass": 2} }
```

ACPI_DEVICE_OST (*'info':* ***ACPIOSTInfo***) [Event]

info ACPIOSTInfo type as described in qapi-schema.json

Emitted when guest executes ACPI _OST method.

Since: 2.1

Example:

```
<- { "event": "ACPI_DEVICE_OST",
     "data": { "device": "d1", "slot": "0",
               "slot-type": "DIMM", "source": 1, "status": 0 } }
```

`BALLOON_CHANGE` (*'actual':* `int`) [Event]

actual actual level of the guest memory balloon in bytes

Emitted when the guest changes the actual BALLOON level. This value is equivalent to the *actual* field return by the 'query-balloon' command

Note: this event is rate-limited.

Since: 1.2

Example:

```
<- { "event": "BALLOON_CHANGE",
     "data": { "actual": 944766976 },
     "timestamp": { "seconds": 1267020223, "microseconds": 435656 } }
```

`GUEST_PANICKED` (*'action':* `GuestPanicAction`) [Event]

action action that has been taken, currently always "pause"

Emitted when guest OS panic is detected

Since: 1.5

Example:

```
<- { "event": "GUEST_PANICKED",
     "data": { "action": "pause" } }
```

`QUORUM_FAILURE` (*'reference':* `str`, *'sector-num':* `int`, *'sectors-count':* `int`) [Event]

reference device name if defined else node name

sector-num
number of the first sector of the failed read operation

sectors-count
failed read operation sector count

Emitted by the Quorum block driver if it fails to establish a quorum

Note: This event is rate-limited.

Since: 2.0

Example:

```
<- { "event": "QUORUM_FAILURE",
     "data": { "reference": "usr1", "sector-num": 345435, "sectors-count": 5 },
     "timestamp": { "seconds": 1344522075, "microseconds": 745528 } }
```

`QUORUM_REPORT_BAD` (*'type':* `QuorumOpType`, [*'error':* `str`], *'node-name':* `str`, *'sector-num':* `int`, *'sectors-count':* `int`) [Event]

type quorum operation type (Since 2.6)

*error** error message. Only present on failure. This field contains a human-readable error message. There are no semantics other than that the block layer reported an error and clients should not try to interpret the error string.

node-name
: the graph node name of the block driver state

sector-num
: number of the first sector of the failed read operation

sectors-count
: failed read operation sector count

Emitted to report a corruption of a Quorum file

Note: This event is rate-limited.

Since: 2.0

Example:

```
1. Read operation

{ "event": "QUORUM_REPORT_BAD",
     "data": { "node-name": "node0", "sector-num": 345435, "sectors-count": 5,
               "type": "read" },
     "timestamp": { "seconds": 1344522075, "microseconds": 745528 } }

2. Flush operation

{ "event": "QUORUM_REPORT_BAD",
     "data": { "node-name": "node0", "sector-num": 0, "sectors-count": 2097120,
               "type": "flush", "error": "Broken pipe" },
     "timestamp": { "seconds": 1456406829, "microseconds": 291763 } }
```

VSERPORT_CHANGE (*'id': `str`, 'open': `bool`*) [Event]

id device identifier of the virtio-serial port

open true if the guest has opened the virtio-serial port

Emitted when the guest opens or closes a virtio-serial port.

Since: 2.1

Example:

```
<- { "event": "VSERPORT_CHANGE",
     "data": { "id": "channel0", "open": true },
     "timestamp": { "seconds": 1401385907, "microseconds": 422329 } }
```

MEM_UNPLUG_ERROR (*'device': `str`, 'msg': `str`*) [Event]

device device name

msg Informative message

Emitted when memory hot unplug error occurs.

Since: 2.4

Example:

```
<- { "event": "MEM_UNPLUG_ERROR"
     "data": { "device": "dimm1",
```

```
                "msg": "acpi: device unplug for unsupported device"
        },
        "timestamp": { "seconds": 1265044230, "microseconds": 450486 } }
```

`DUMP_COMPLETED` (*'result':* `DumpQueryResult`, [*'error':* `str`]) [Event]

result DumpQueryResult type described in qapi-schema.json.

*error** human-readable error string that provides hint on why dump failed. Only presents on failure. The user should not try to interpret the error string.

Emitted when background dump has completed

Since: 2.6

Example:

```
{ "event": "DUMP_COMPLETED",
  "data": {"result": {"total": 1090650112, "status": "completed",
                      "completed": 1090650112} } }
```

2.7 Tracing commands

`TraceEventState` [Enum]

'`unavailable`'
The event is statically disabled.

'`disabled`'
The event is dynamically disabled.

'`enabled`' The event is dynamically enabled.

State of a tracing event.

Since: 2.2

`TraceEventInfo` *{ 'name':* `str`*, 'state':* `TraceEventState`*, 'vcpu':* `bool` *}* [Struct]

name Event name.

state Tracing state.

vcpu Whether this is a per-vCPU event (since 2.7).

An event is per-vCPU if it has the "vcpu" property in the "trace-events" files.

Information of a tracing event.

Since: 2.2

`['TraceEventInfo'] trace-event-get-state` (*'name':* `str`, [*'vcpu':* `int`]) [Command]

name Event name pattern (case-sensitive glob).

*vcpu** The vCPU to query (any by default; since 2.7).

Query the state of events.

Returns: a list of *TraceEventInfo* for the matching events

An event is returned if:

- its name matches the *name* pattern, and
- if *vcpu* is given, the event has the "vcpu" property.

Therefore, if *vcpu* is given, the operation will only match per-vCPU events, returning their state on the specified vCPU. Special case: if *name* is an exact match, *vcpu* is given and the event does not have the "vcpu" property, an error is returned.

Since: 2.2

Example:

```
-> { "execute": "trace-event-get-state",
     "arguments": { "name": "qemu_memalign" } }
<- { "return": [ { "name": "qemu_memalign", "state": "disabled" } ] }
```

`trace-event-set-state` (*'name':* `str`*, 'enable':* `bool`*,* [*'ignore-unavailable':* `bool`], [*'vcpu':* `int`]) [Command]

name Event name pattern (case-sensitive glob).

enable Whether to enable tracing.

*ignore-unavailable** Do not match unavailable events with *name*.

*vcpu** The vCPU to act upon (all by default; since 2.7).

An event's state is modified if:

- its name matches the *name* pattern, and
- if *vcpu* is given, the event has the "vcpu" property.

Therefore, if *vcpu* is given, the operation will only match per-vCPU events, setting their state on the specified vCPU. Special case: if *name* is an exact match, *vcpu* is given and the event does not have the "vcpu" property, an error is returned.

Set the dynamic tracing state of events.

Since: 2.2

Example:

```
-> { "execute": "trace-event-set-state",
     "arguments": { "name": "qemu_memalign", "enable": "true" } }
<- { "return": {} }
```

`['SchemaInfo'] query-qmp-schema` () [Command]

Command query-qmp-schema exposes the QMP wire ABI as an array of SchemaInfo. This lets QMP clients figure out what commands and events are available in this QEMU, and their parameters and results.

However, the SchemaInfo can't reflect all the rules and restrictions that apply to QMP. It's interface introspection (figuring out what's there), not interface specification. The specification is in the QAPI schema.

Furthermore, while we strive to keep the QMP wire format backwards-compatible across qemu versions, the introspection output is not guaranteed to have the same stability. For example, one version of qemu may list an object member as an optional non-variant, while another lists the same member only through the object's variants; or the type of a member may change from a generic string into a specific enum or from one specific type into an alternate that includes the original type alongside something else.

Returns: array of *SchemaInfo*, where each element describes an entity in the ABI: command, event, type, ...

The order of the various SchemaInfo is unspecified; however, all names are guaranteed to be unique (no name will be duplicated with different meta-types).

Note: the QAPI schema is also used to help define *internal* interfaces, by defining QAPI types. These are not part of the QMP wire ABI, and therefore not returned by this command.

Since: 2.5

`SchemaMetaType` [Enum]

'`builtin`' a predefined type such as 'int' or 'bool'.

'`enum`' an enumeration type

'`array`' an array type

'`object`' an object type (struct or union)

'`alternate`'
an alternate type

'`command`' a QMP command

'`event`' a QMP event

This is a *SchemaInfo*'s meta type, i.e. the kind of entity it describes.

Since: 2.5

`SchemaInfo` [*'builtin':* `SchemaInfoBuiltin`*, 'enum':* `SchemaInfoEnum`*, 'array':* `SchemaInfoArray`*, 'object':* `SchemaInfoObject`*, 'alternate':* `SchemaInfoAlternate`*, 'command':* `SchemaInfoCommand`*, 'event':* `SchemaInfoEvent`] [Union]

name the entity's name, inherited from *base*. Commands and events have the name defined in the QAPI schema. Unlike command and event names, type names are not part of the wire ABI. Consequently, type names are meaningless strings here, although they are still guaranteed unique regardless of *meta*-type.

All references to other SchemaInfo are by name.

meta-type the entity's meta type, inherited from *base*.

Additional members depend on the value of *meta*-type.

Since: 2.5

`SchemaInfoBuiltin` *{ 'json-type': `JSONType` }* [Struct]

json-type the JSON type used for this type on the wire.

Additional SchemaInfo members for meta-type 'builtin'.

Since: 2.5

`JSONType` [Enum]

The four primitive and two structured types according to RFC 7159 section 1, plus 'int' (split off 'number'), plus the obvious top type 'value'.

Since: 2.5

`SchemaInfoEnum` *{ 'values': ['`str`'] }* [Struct]

values the enumeration type's values, in no particular order.

Values of this type are JSON string on the wire.

Additional SchemaInfo members for meta-type 'enum'.

Since: 2.5

`SchemaInfoArray` *{ 'element-type': `str` }* [Struct]

element-type
the array type's element type.

Values of this type are JSON array on the wire.

Additional SchemaInfo members for meta-type 'array'.

Since: 2.5

`SchemaInfoObject` *{ 'members': ['`SchemaInfoObjectMember`'], ['tag': `str`], ['variants': ['`SchemaInfoObjectVariant`']] }* [Struct]

members the object type's (non-variant) members, in no particular order.

tag is present. The variants are in no particular order, and may even differ from the order of the values of the enum type of the *tag*.

Values of this type are JSON object on the wire.

*variants** variant members, i.e. additional members that depend on the type tag's value. Present exactly when

Additional SchemaInfo members for meta-type 'object'.

Since: 2.5

`SchemaInfoObjectMember` *{ 'name': `str`, 'type': `str`, ['default': `any`] }* [Struct]

name the member's name, as defined in the QAPI schema.

type the name of the member's type.

*default** default when used as command parameter. If absent, the parameter is mandatory. If present, the value must be null. The parameter is optional, and behavior when it's missing is not specified here. Future extension: if present and non-null, the parameter is optional, and defaults to this value.

An object member.

Since: 2.5

`SchemaInfoObjectVariant` *{ 'case':* `str`*, 'type':* `str` *}* [Struct]

case a value of the type tag.

type the name of the object type that provides the variant members when the type tag has value *case*.

The variant members for a value of the type tag.

Since: 2.5

`SchemaInfoAlternate` *{ 'members':* [`'SchemaInfoAlternateMember'`] *}* [Struct]

members the alternate type's members, in no particular order. The members' wire encoding is distinct, see docs/qapi-code-gen.txt section Alternate types.

On the wire, this can be any of the members.

Additional SchemaInfo members for meta-type 'alternate'.

Since: 2.5

`SchemaInfoAlternateMember` *{ 'type':* `str` *}* [Struct]

type the name of the member's type.

An alternate member.

Since: 2.5

`SchemaInfoCommand` *{ 'arg-type':* `str`*, 'ret-type':* `str` *}* [Struct]

arg-type the name of the object type that provides the command's parameters.

ret-type the name of the command's result type.

TODO *success*-response (currently irrelevant, because it's QGA, not QMP)

Additional SchemaInfo members for meta-type 'command'.

Since: 2.5

`SchemaInfoEvent` *{ 'arg-type':* `str` *}* [Struct]

arg-type the name of the object type that provides the event's parameters.

Additional SchemaInfo members for meta-type 'event'.

Since: 2.5

2.8 QMP commands

`qmp_capabilities` () [Command]

Enable QMP capabilities.

Arguments: None.

Notes: This command is valid exactly when first connecting: it must be issued before any other command will be accepted, and will fail once the monitor is accepting other commands. (see qemu docs/qmp-spec.txt)

Since: 0.13

Example:

```
-> { "execute": "qmp_capabilities" }
<- { "return": {} }
```

`LostTickPolicy` [Enum]

- '`discard`' throw away the missed tick(s) and continue with future injection normally. Guest time may be delayed, unless the OS has explicit handling of lost ticks
- '`delay`' continue to deliver ticks at the normal rate. Guest time will be delayed due to the late tick
- '`merge`' merge the missed tick(s) into one tick and inject. Guest time may be delayed, depending on how the OS reacts to the merging of ticks
- '`slew`' deliver ticks at a higher rate to catch up with the missed tick. The guest time should not be delayed once catchup is complete.

Policy for handling lost ticks in timer devices.

Since: 2.0

`add_client` (*'protocol':* `str`, *'fdname':* `str`, [*'skipauth':* `bool`], [*'tls':* `bool`]) [Command]

- *protocol* protocol name. Valid names are "vnc", "spice" or the name of a character device (eg. from -chardev id=XXXX)
- *fdname* file descriptor name previously passed via 'getfd' command
- *skipauth** whether to skip authentication. Only applies to "vnc" and "spice" protocols
- *tls** whether to perform TLS. Only applies to the "spice" protocol

Allow client connections for VNC, Spice and socket based character devices to be passed in to QEMU via SCM_RIGHTS.

Returns: nothing on success.

Since: 0.14.0

Example:

```
-> { "execute": "add_client", "arguments": { "protocol": "vnc",
                                             "fdname": "myclient" } }
<- { "return": {} }
```

`NameInfo` *{ ['name': `str`] }* [Struct]

*name** The name of the guest

Guest name information.

Since: 0.14.0

`NameInfo query-name` () [Command]

Return the name information of a guest.

Returns: *NameInfo* of the guest

Since: 0.14.0

Example:

```
-> { "execute": "query-name" }
<- { "return": { "name": "qemu-name" } }
```

`KvmInfo` *{ 'enabled': `bool`, 'present': `bool` }* [Struct]

enabled true if KVM acceleration is active

present true if KVM acceleration is built into this executable

Information about support for KVM acceleration

Since: 0.14.0

`KvmInfo query-kvm` () [Command]

Returns information about KVM acceleration

Returns: *KvmInfo*

Since: 0.14.0

Example:

```
-> { "execute": "query-kvm" }
<- { "return": { "enabled": true, "present": true } }
```

`RunState` [Enum]

'`debug`' QEMU is running on a debugger

'`finish-migrate`'
guest is paused to finish the migration process

'`inmigrate`'
guest is paused waiting for an incoming migration. Note that this state does not tell whether the machine will start at the end of the migration. This depends on the command-line -S option and any invocation of 'stop' or 'cont' that has happened since QEMU was started.

'`internal-error`'
An internal error that prevents further guest execution has occurred

'`io-error`'
the last IOP has failed and the device is configured to pause on I/O errors

'`paused`' guest has been paused via the 'stop' command

‘`postmigrate`’
guest is paused following a successful 'migrate'

‘`prelaunch`’
QEMU was started with -S and guest has not started

‘`restore-vm`’
guest is paused to restore VM state

‘`running`’ guest is actively running

‘`save-vm`’ guest is paused to save the VM state

‘`shutdown`’
guest is shut down (and -no-shutdown is in use)

‘`suspended`’
guest is suspended (ACPI S3)

‘`watchdog`’
the watchdog action is configured to pause and has been triggered

‘`guest-panicked`’
guest has been panicked as a result of guest OS panic

An enumeration of VM run states.

`StatusInfo` *{ 'running':* `bool`*, 'singlestep':* `bool`*, 'status':* `RunState` *}* [Struct]

running true if all VCPUs are runnable, false if not runnable

singlestep true if VCPUs are in single-step mode

status the virtual machine *RunState*

Information about VCPU run state

Notes: *singlestep* is enabled through the GDB stub

Since: 0.14.0

`StatusInfo query-status ()` [Command]

Query the run status of all VCPUs

Returns: *StatusInfo* reflecting all VCPUs

Since: 0.14.0

Example:

```
-> { "execute": "query-status" }
<- { "return": { "running": true,
                 "singlestep": false,
                 "status": "running" } }
```

`UuidInfo` *{ 'UUID':* `str` *}* [Struct]

UUID the UUID of the guest

Guest UUID information (Universally Unique Identifier).

Notes: If no UUID was specified for the guest, a null UUID is returned.

Since: 0.14.0

`UuidInfo query-uuid ()` [Command]

Query the guest UUID information.

Returns: The *UuidInfo* for the guest

Since: 0.14.0

Example:

```
-> { "execute": "query-uuid" }
<- { "return": { "UUID": "550e8400-e29b-41d4-a716-446655440000" } }
```

`ChardevInfo` *{ 'label':* `str`*, 'filename':* `str`*, 'frontend-open':* `bool` *}* [Struct]

label the label of the character device

filename the filename of the character device

frontend-open
shows whether the frontend device attached to this backend (eg. with the chardev=... option) is in open or closed state (since 2.1)

Information about a character device.

Notes: *filename* is encoded using the QEMU command line character device encoding. See the QEMU man page for details.

Since: 0.14.0

`['ChardevInfo'] query-chardev ()` [Command]

Returns information about current character devices.

Returns: a list of *ChardevInfo*

Since: 0.14.0

Example:

```
-> { "execute": "query-chardev" }
<- {
      "return": [
         {
            "label": "charchannel0",
            "filename": "unix:/var/lib/libvirt/qemu/seabios.rhel6.agent,server",
            "frontend-open": false
         },
         {
            "label": "charmonitor",
            "filename": "unix:/var/lib/libvirt/qemu/seabios.rhel6.monitor,server",
            "frontend-open": true
         },
         {
            "label": "charserial0",
            "filename": "pty:/dev/pts/2",
            "frontend-open": true
         }
      ]
   }
```

`ChardevBackendInfo` *{ 'name': `str` }* [Struct]

name The backend name

Information about a character device backend

Since: 2.0

`['ChardevBackendInfo'] query-chardev-backends` () [Command]

Returns information about character device backends.

Returns: a list of *ChardevBackendInfo*

Since: 2.0

Example:

```
-> { "execute": "query-chardev-backends" }
<- {
      "return":[
         {
            "name":"udp"
         },
         {
            "name":"tcp"
         },
         {
            "name":"unix"
         },
         {
            "name":"spiceport"
         }
      ]
   }
```

`DataFormat` [Enum]

'`utf8`' Data is a UTF-8 string (RFC 3629)

'`base64`' Data is Base64 encoded binary (RFC 3548)

An enumeration of data format.

Since: 1.4

`ringbuf-write` (*'device': `str`, 'data': `str`,* [*'format': `DataFormat`*]) [Command]

device the ring buffer character device name

data data to write

*format** data encoding (default 'utf8').

- base64: data must be base64 encoded text. Its binary decoding gets written.
- utf8: data's UTF-8 encoding is written
- data itself is always Unicode regardless of format, like any other string.

Write to a ring buffer character device.

Returns: Nothing on success

Since: 1.4

Example:

```
-> { "execute": "ringbuf-write",
     "arguments": { "device": "foo",
                    "data": "abcdefgh",
                    "format": "utf8" } }
<- { "return": {} }
```

`str ringbuf-read` (*'device': `str`, 'size': `int`, ['format': `DataFormat`]*) [Command]

device the ring buffer character device name

size how many bytes to read at most

*format** data encoding (default 'utf8').

- base64: the data read is returned in base64 encoding.
- utf8: the data read is interpreted as UTF-8. Bug: can screw up when the buffer contains invalid UTF-8 sequences, NUL characters, after the ring buffer lost data, and when reading stops because the size limit is reached.
- The return value is always Unicode regardless of format, like any other string.

Read from a ring buffer character device.

Returns: data read from the device

Since: 1.4

Example:

```
-> { "execute": "ringbuf-read",
     "arguments": { "device": "foo",
                    "size": 1000,
                    "format": "utf8" } }
<- { "return": "abcdefgh" }
```

`EventInfo` *{ 'name': `str` }* [Struct]

name The event name

Information about a QMP event

Since: 1.2.0

`['EventInfo'] query-events` () [Command]

Return a list of supported QMP events by this server

Returns: A list of *EventInfo* for all supported events

Note: This example has been shortened as the real response is too long.

Since: 1.2.0

Example:

```
 -> { "execute": "query-events" }
 <- {
      "return": [
          {
             "name":"SHUTDOWN"
          },
          {
             "name":"RESET"
          }
       ]
    }
```

`MigrationStats` *{ 'transferred':* `int`*, 'remaining':* `int`*, 'total':* `int`*,* [Struct]
'duplicate': `int`*, 'skipped':* `int`*, 'normal':* `int`*, 'normal-bytes':* `int`*,*
'dirty-pages-rate': `int`*, 'mbps':* `number`*, 'dirty-sync-count':* `int`*,*
'postcopy-requests': `int` *}*

transferred
: amount of bytes already transferred to the target VM

remaining
: amount of bytes remaining to be transferred to the target VM

total
: total amount of bytes involved in the migration process

duplicate
: number of duplicate (zero) pages (since 1.2)

skipped
: number of skipped zero pages (since 1.5)

normal
: : number of normal pages (since 1.2)

normal-bytes
: number of normal bytes sent (since 1.2)

dirty-pages-rate
: number of pages dirtied by second by the guest (since 1.3)

mbps
: throughput in megabits/sec. (since 1.6)

dirty-sync-count
: number of times that dirty ram was synchronized (since 2.1)

postcopy-requests
: The number of page requests received from the destination (since 2.7)

Detailed migration status.

Since: 0.14.0

`XBZRLECacheStats` *{ 'cache-size':* `int`*, 'bytes':* `int`*, 'pages':* `int`*,* [Struct]
'cache-miss': `int`*, 'cache-miss-rate':* `number`*, 'overflow':* `int` *}*

cache-size
: XBZRLE cache size

bytes
: amount of bytes already transferred to the target VM

pages
: amount of pages transferred to the target VM

cache-miss
: number of cache miss

cache-miss-rate
: rate of cache miss (since 2.1)

overflow : number of overflows

Detailed XBZRLE migration cache statistics

Since: 1.2

`MigrationStatus` [Enum]

'`none`' : no migration has ever happened.

'`setup`' : migration process has been initiated.

'`cancelling`'
: in the process of cancelling migration.

'`cancelled`'
: cancelling migration is finished.

'`active`' : in the process of doing migration.

'`postcopy-active`'
: like active, but now in postcopy mode. (since 2.5)

'`completed`'
: migration is finished.

'`failed`' : some error occurred during migration process.

An enumeration of migration status.

Since: 2.3

`MigrationInfo` *{* [*'status':* `MigrationStatus`], [*'ram':* `MigrationStats`], [*'disk':* `MigrationStats`], [*'xbzrle-cache':* `XBZRLECacheStats`], [*'total-time':* `int`], [*'expected-downtime':* `int`], [*'downtime':* `int`], [*'setup-time':* `int`], [*'cpu-throttle-percentage':* `int`], [*'error-desc':* `str`] *}* [Struct]

status : is 'failed'. Clients should not attempt to parse the error strings. (Since 2.7)

*ram** : *MigrationStats* containing detailed migration status, only returned if status is 'active' or 'completed'(since 1.2)

*disk** : *MigrationStats* containing detailed disk migration status, only returned if status is 'active' and it is a block migration

*xbzrle-cache**
: *XBZRLECacheStats* containing detailed XBZRLE migration statistics, only returned if XBZRLE feature is on and status is 'active' or 'completed' (since 1.2)

*total-time**
: total amount of milliseconds since migration started. If migration has ended, it returns the total migration time. (since 1.2)

*downtime**
: only present when migration finishes correctly total downtime in milliseconds for the guest. (since 1.3)

*expected-downtime**
: only present while migration is active expected downtime in milliseconds for the guest in last walk of the dirty bitmap. (since 1.3)

*setup-time**
: amount of setup time in milliseconds *before* the iterations begin but *after* the QMP command is issued. This is designed to provide an accounting of any activities (such as RDMA pinning) which may be expensive, but do not actually occur during the iterative migration rounds themselves. (since 1.6)

*cpu-throttle-percentage**
: percentage of time guest cpus are being throttled during auto-converge. This is only present when auto-converge has started throttling guest cpus. (Since 2.7)

*error-desc**
: the human readable error description string, when

Information about current migration process.

Since: 0.14.0

`MigrationInfo query-migrate ()` [Command]

Returns information about current migration process. If migration is active there will be another json-object with RAM migration status and if block migration is active another one with block migration status.

Returns: *MigrationInfo*

Since: 0.14.0

Example:

```
1. Before the first migration

-> { "execute": "query-migrate" }
<- { "return": {} }

2. Migration is done and has succeeded

-> { "execute": "query-migrate" }
<- { "return": {
        "status": "completed",
        "ram":{
          "transferred":123,
```

```
           "remaining":123,
           "total":246,
           "total-time":12345,
           "setup-time":12345,
           "downtime":12345,
           "duplicate":123,
           "normal":123,
           "normal-bytes":123456,
           "dirty-sync-count":15
         }
      }
   }
```

3. Migration is done and has failed

```
-> { "execute": "query-migrate" }
<- { "return": { "status": "failed" } }
```

4. Migration is being performed and is not a block migration:

```
-> { "execute": "query-migrate" }
<- {
      "return":{
         "status":"active",
         "ram":{
            "transferred":123,
            "remaining":123,
            "total":246,
            "total-time":12345,
            "setup-time":12345,
            "expected-downtime":12345,
            "duplicate":123,
            "normal":123,
            "normal-bytes":123456,
            "dirty-sync-count":15
         }
      }
   }
```

5. Migration is being performed and is a block migration:

```
-> { "execute": "query-migrate" }
<- {
      "return":{
         "status":"active",
         "ram":{
            "total":1057024,
```

```
               "remaining":1053304,
               "transferred":3720,
               "total-time":12345,
               "setup-time":12345,
               "expected-downtime":12345,
               "duplicate":123,
               "normal":123,
               "normal-bytes":123456,
               "dirty-sync-count":15
            },
            "disk":{
               "total":20971520,
               "remaining":20880384,
               "transferred":91136
            }
         }
      }
```

6. Migration is being performed and XBZRLE is active:

```
-> { "execute": "query-migrate" }
<- {
      "return":{
         "status":"active",
         "capabilities" : [ { "capability": "xbzrle", "state" : true } ],
         "ram":{
            "total":1057024,
            "remaining":1053304,
            "transferred":3720,
            "total-time":12345,
            "setup-time":12345,
            "expected-downtime":12345,
            "duplicate":10,
            "normal":3333,
            "normal-bytes":3412992,
            "dirty-sync-count":15
         },
         "xbzrle-cache":{
            "cache-size":67108864,
            "bytes":20971520,
            "pages":2444343,
            "cache-miss":2244,
            "cache-miss-rate":0.123,
            "overflow":34434
         }
      }
   }
```

`MigrationCapability` [Enum]

'`xbzrle`' Migration supports xbzrle (Xor Based Zero Run Length Encoding). This feature allows us to minimize migration traffic for certain work loads, by sending compressed difference of the pages

'`rdma-pin-all`'
Controls whether or not the entire VM memory footprint is mlock()'d on demand or all at once. Refer to docs/rdma.txt for usage. Disabled by default. (since 2.0)

'`zero-blocks`'
During storage migration encode blocks of zeroes efficiently. This essentially saves 1MB of zeroes per block on the wire. Enabling requires source and target VM to support this feature. To enable it is sufficient to enable the capability on the source VM. The feature is disabled by default. (since 1.6)

'`compress`'
Use multiple compression threads to accelerate live migration. This feature can help to reduce the migration traffic, by sending compressed pages. Please note that if compress and xbzrle are both on, compress only takes effect in the ram bulk stage, after that, it will be disabled and only xbzrle takes effect, this can help to minimize migration traffic. The feature is disabled by default. (since 2.4)

'`events`' generate events for each migration state change (since 2.4)

'`auto-converge`'
If enabled, QEMU will automatically throttle down the guest to speed up convergence of RAM migration. (since 1.6)

'`postcopy-ram`'
Start executing on the migration target before all of RAM has been migrated, pulling the remaining pages along as needed. NOTE: If the migration fails during postcopy the VM will fail. (since 2.6)

Migration capabilities enumeration

Since: 1.2

`MigrationCapabilityStatus` *{ 'capability':* `MigrationCapability`*, 'state':* `bool` *}* [Struct]

capability capability enum

state capability state bool

Migration capability information

Since: 1.2

`migrate-set-capabilities` (*'capabilities':* [*'*`MigrationCapabilityStatus`*'*]) [Command]

capabilities
json array of capability modifications to make

Enable/Disable the following migration capabilities (like xbzrle)

Since: 1.2

Example:

```
-> { "execute": "migrate-set-capabilities" , "arguments":
     { "capabilities": [ { "capability": "xbzrle", "state": true } ] } }
```

`['MigrationCapabilityStatus'] query-migrate-capabilities` [Command]
`()`

Returns information about the current migration capabilities status

Returns: *MigrationCapabilitiesStatus*

Since: 1.2

Example:

```
-> { "execute": "query-migrate-capabilities" }
<- { "return": [
      {"state": false, "capability": "xbzrle"},
      {"state": false, "capability": "rdma-pin-all"},
      {"state": false, "capability": "auto-converge"},
      {"state": false, "capability": "zero-blocks"},
      {"state": false, "capability": "compress"},
      {"state": true, "capability": "events"},
      {"state": false, "capability": "postcopy-ram"}
   ]}
```

`MigrationParameter` [Enum]

'`compress-level`'
: Set the compression level to be used in live migration, the compression level is an integer between 0 and 9, where 0 means no compression, 1 means the best compression speed, and 9 means best compression ratio which will consume more CPU.

'`compress-threads`'
: Set compression thread count to be used in live migration, the compression thread count is an integer between 1 and 255.

'`decompress-threads`'
: Set decompression thread count to be used in live migration, the decompression thread count is an integer between 1 and 255. Usually, decompression is at least 4 times as fast as compression, so set the decompress-threads to the number about 1/4 of compress-threads is adequate.

'`cpu-throttle-initial`'
: Initial percentage of time guest cpus are throttled when migration auto-converge is activated. The default value is 20. (Since 2.7)

'`cpu-throttle-increment`'
: throttle percentage increase each time auto-converge detects that migration is not making progress. The default value is 10. (Since 2.7)

‘`tls-creds`’
: ID of the 'tls-creds' object that provides credentials for establishing a TLS connection over the migration data channel. On the outgoing side of the migration, the credentials must be for a 'client' endpoint, while for the incoming side the credentials must be for a 'server' endpoint. Setting this will enable TLS for all migrations. The default is unset, resulting in unsecured migration at the QEMU level. (Since 2.7)

‘`tls-hostname`’
: hostname of the target host for the migration. This is required when using x509 based TLS credentials and the migration URI does not already include a hostname. For example if using fd: or exec: based migration, the hostname must be provided so that the server's x509 certificate identity can be validated. (Since 2.7)

Migration parameters enumeration

Since: 2.4

`migrate-set-parameters` (*'compress-level':* `int`], [*'compress-threads':* `int`], [*'decompress-threads':* `int`], [*'cpu-throttle-initial':* `int`], [*'cpu-throttle-increment':* `int`], [*'tls-creds':* `str`], [*'tls-hostname':* `str`]) [Command]

compress-level
: compression level

compress-threads
: compression thread count

decompress-threads
: decompression thread count

cpu-throttle-initial
: Initial percentage of time guest cpus are throttled when migration auto-converge is activated. The default value is 20. (Since 2.7)

cpu-throttle-increment
: throttle percentage increase each time auto-converge detects that migration is not making progress. The default value is 10. (Since 2.7)

tls-creds ID of the 'tls-creds' object that provides credentials for establishing a TLS connection over the migration data channel. On the outgoing side of the migration, the credentials must be for a 'client' endpoint, while for the incoming side the credentials must be for a 'server' endpoint. Setting this will enable TLS for all migrations. The default is unset, resulting in unsecured migration at the QEMU level. (Since 2.7)

tls-hostname
: hostname of the target host for the migration. This is required when using x509 based TLS credentials and the migration URI does not already include a hostname. For example if using fd: or exec: based migration, the hostname must be provided so that the server's x509 certificate identity can be validated. (Since 2.7)

Set the following migration parameters

Since: 2.4

Example:

```
-> { "execute": "migrate-set-parameters" ,
     "arguments": { "compress-level": 1 } }
```

`MigrationParameters` *{ 'compress-level':* `int`*, 'compress-threads':* `int`*, 'decompress-threads':* `int`*, 'cpu-throttle-initial':* `int`*, 'cpu-throttle-increment':* `int`*, 'tls-creds':* `str`*, 'tls-hostname':* `str` *}* [Struct]

compress-level
: compression level

compress-threads
: compression thread count

decompress-threads
: decompression thread count

cpu-throttle-initial
: Initial percentage of time guest cpus are throttled when migration auto-converge is activated. The default value is 20. (Since 2.7)

cpu-throttle-increment
: throttle percentage increase each time auto-converge detects that migration is not making progress. The default value is 10. (Since 2.7)

tls-creds
: ID of the 'tls-creds' object that provides credentials for establishing a TLS connection over the migration data channel. On the outgoing side of the migration, the credentials must be for a 'client' endpoint, while for the incoming side the credentials must be for a 'server' endpoint. Setting this will enable TLS for all migrations. The default is unset, resulting in unsecured migration at the QEMU level. (Since 2.7)

tls-hostname
: hostname of the target host for the migration. This is required when using x509 based TLS credentials and the migration URI does not already include a hostname. For example if using fd: or exec: based migration, the hostname must be provided so that the server's x509 certificate identity can be validated. (Since 2.7)

Since: 2.4

`MigrationParameters query-migrate-parameters` () [Command]

Returns information about the current migration parameters

Returns: *MigrationParameters*

Since: 2.4

Example:

```
-> { "execute": "query-migrate-parameters" }
<- { "return": {
```

```
            "decompress-threads": 2,
            "cpu-throttle-increment": 10,
            "compress-threads": 8,
            "compress-level": 1,
            "cpu-throttle-initial": 20
         }
      }
```

`client_migrate_info` (*'protocol':* `str`, *'hostname':* `str`, [*'port':* `int`], [*'tls-port':* `int`], [*'cert-subject':* `str`]) [Command]

protocol must be "spice"

hostname migration target hostname

*port** spice tcp port for plaintext channels

*tls-port** spice tcp port for tls-secured channels

*cert-subject**
server certificate subject

Set migration information for remote display. This makes the server ask the client to automatically reconnect using the new parameters once migration finished successfully. Only implemented for SPICE.

Since: 0.14.0

Example:

```
 -> { "execute": "client_migrate_info",
      "arguments": { "protocol": "spice",
                     "hostname": "virt42.lab.kraxel.org",
                     "port": 1234 } }
 <- { "return": {} }
```

`migrate-start-postcopy` () [Command]

Followup to a migration command to switch the migration to postcopy mode. The postcopy-ram capability must be set before the original migration command.

Since: 2.5

Example:

```
 -> { "execute": "migrate-start-postcopy" }
 <- { "return": {} }
```

`MouseInfo` *{ 'name':* `str`, *'index':* `int`, *'current':* `bool`, *'absolute':* `bool` *}* [Struct]

name the name of the mouse device

index the index of the mouse device

current true if this device is currently receiving mouse events

absolute true if this device supports absolute coordinates as input

Information about a mouse device.

Since: 0.14.0

`['MouseInfo'] query-mice ()` [Command]

Returns information about each active mouse device

Returns: a list of *MouseInfo* for each device

Since: 0.14.0

Example:

```
-> { "execute": "query-mice" }
<- { "return": [
          {
             "name":"QEMU Microsoft Mouse",
             "index":0,
             "current":false,
             "absolute":false
          },
          {
             "name":"QEMU PS/2 Mouse",
             "index":1,
             "current":true,
             "absolute":true
          }
       ]
    }
```

`CpuInfoArch` [Enum]

'`query-cpus.`'

An enumeration of cpu types that enable additional information during

Since: 2.6

`CpuInfo` [*'x86':* `CpuInfoX86`*, 'sparc':* `CpuInfoSPARC`*, 'ppc':* `CpuInfoPPC`*, 'mips':* `CpuInfoMIPS`*, 'tricore':* `CpuInfoTricore`*, 'other':* `CpuInfoOther`] [Union]

CPU	the index of the virtual CPU
current	this only exists for backwards compatibility and should be ignored
halted	true if the virtual CPU is in the halt state. Halt usually refers to a processor specific low power mode.
qom_path	path to the CPU object in the QOM tree (since 2.4)
thread_id	ID of the underlying host thread
arch	architecture of the cpu, which determines which additional fields will be listed (since 2.6)

Information about a virtual CPU

Notes: *halted* is a transient state that changes frequently. By the time the data is sent to the client, the guest may no longer be halted.

Since: 0.14.0

`CpuInfoX86` *{ 'pc': `int` }* [Struct]

pc the 64-bit instruction pointer

Additional information about a virtual i386 or x86_64 CPU

Since: 2.6

`CpuInfoSPARC` *{ 'pc': `int`, 'npc': `int` }* [Struct]

pc the PC component of the instruction pointer

npc the NPC component of the instruction pointer

Additional information about a virtual SPARC CPU

Since: 2.6

`CpuInfoPPC` *{ 'nip': `int` }* [Struct]

nip the instruction pointer

Additional information about a virtual PPC CPU

Since: 2.6

`CpuInfoMIPS` *{ 'PC': `int` }* [Struct]

PC the instruction pointer

Additional information about a virtual MIPS CPU

Since: 2.6

`CpuInfoTricore` *{ 'PC': `int` }* [Struct]

PC the instruction pointer

Additional information about a virtual Tricore CPU

Since: 2.6

`CpuInfoOther` *{ }* [Struct]

No additional information is available about the virtual CPU

Since: 2.6

`['CpuInfo'] query-cpus` () [Command]

Returns a list of information about each virtual CPU.

Returns: a list of *CpuInfo* for each virtual CPU

Since: 0.14.0

Example:

```
 -> { "execute": "query-cpus" }
 <- { "return": [
          {
             "CPU":0,
             "current":true,
             "halted":false,
             "qom_path":"/machine/unattached/device[0]",
```

```
                "arch":"x86",
                "pc":3227107138,
                "thread_id":3134
             },
             {
                "CPU":1,
                "current":false,
                "halted":true,
                "qom_path":"/machine/unattached/device[2]",
                "arch":"x86",
                "pc":7108165,
                "thread_id":3135
             }
          ]
       }
```

`IOThreadInfo` *{ 'id': `str`, 'thread-id': `int` }* [Struct]

id the identifier of the iothread

thread-id ID of the underlying host thread

Information about an iothread

Since: 2.0

`['IOThreadInfo'] query-iothreads ()` [Command]

Returns a list of information about each iothread.

Note this list excludes the QEMU main loop thread, which is not declared using the -object iothread command-line option. It is always the main thread of the process.

Returns: a list of *IOThreadInfo* for each iothread

Since: 2.0

Example:

```
 -> { "execute": "query-iothreads" }
 <- { "return": [
          {
             "id":"iothread0",
             "thread-id":3134
          },
          {
             "id":"iothread1",
             "thread-id":3135
          }
       ]
    }
```

`NetworkAddressFamily` [Enum]

'`ipv4`' IPV4 family

'`ipv6`' IPV6 family

'`unix`' unix socket

'`unknown`' otherwise

The network address family

Since: 2.1

`VncBasicInfo` *{ 'host':* `str`*, 'service':* `str`*, 'family':* `NetworkAddressFamily`*, 'websocket':* `bool` *}* [Struct]

host IP address

service The service name of the vnc port. This may depend on the host system's service database so symbolic names should not be relied on.

family address family

websocket true in case the socket is a websocket (since 2.3).

The basic information for vnc network connection

Since: 2.1

`VncServerInfo` *{* [*'auth':* `str`] *}* [Struct]

*auth** authentication method

The network connection information for server

Since: 2.1

`VncClientInfo` *{* [*'x509_dname':* `str`], [*'sasl_username':* `str`] *}* [Struct]

*x509_dname**
If x509 authentication is in use, the Distinguished Name of the client.

*sasl_username**
If SASL authentication is in use, the SASL username used for authentication.

Information about a connected VNC client.

Since: 0.14.0

`VncInfo` *{ 'enabled':* `bool`*,* [*'host':* `str`], [*'family':* `NetworkAddressFamily`], [*'service':* `str`], [*'auth':* `str`], [*'clients':* [*'VncClientInfo'*]] *}* [Struct]

enabled true if the VNC server is enabled, false otherwise

*host** The hostname the VNC server is bound to. This depends on the name resolution on the host and may be an IP address.

*family** 'ipv6' if the host is listening for IPv6 connections 'ipv4' if the host is listening for IPv4 connections 'unix' if the host is listening on a unix domain socket 'unknown' otherwise

*service** The service name of the server's port. This may depends on the host system's service database so symbolic names should not be relied on.

*auth** the current authentication type used by the server 'none' if no authentication is being used 'vnc' if VNC authentication is being used 'vencrypt+plain' if VEncrypt is used with plain text authentication 'vencrypt+tls+none' if VEncrypt is used with TLS and no authentication 'vencrypt+tls+vnc' if VEncrypt is used with TLS and VNC authentication 'vencrypt+tls+plain' if VEncrypt is used with TLS and plain text auth 'vencrypt+x509+none' if VEncrypt is used with x509 and no auth 'vencrypt+x509+vnc' if VEncrypt is used with x509 and VNC auth 'vencrypt+x509+plain' if VEncrypt is used with x509 and plain text auth 'vencrypt+tls+sasl' if VEncrypt is used with TLS and SASL auth 'vencrypt+x509+sasl' if VEncrypt is used with x509 and SASL auth

clients a list of *VncClientInfo* of all currently connected clients

Information about the VNC session.

Since: 0.14.0

`VncPriAuth` [Enum]

vnc primary authentication method.

Since: 2.3

`VncVencryptSubAuth` [Enum]

vnc sub authentication method with vencrypt.

Since: 2.3

`VncInfo2` *{ 'id':* `str`*, 'server':* [`'VncBasicInfo'`]*, 'clients':* [`'VncClientInfo'`]*, 'auth':* `VncPrimaryAuth`*, ['vencrypt':* `VncVencryptSubAuth`*], ['display':* `str`*] }* [Struct]

id vnc server name.

server A list of *VncBasincInfo* describing all listening sockets. The list can be empty (in case the vnc server is disabled). It also may have multiple entries: normal + websocket, possibly also ipv4 + ipv6 in the future.

clients A list of *VncClientInfo* of all currently connected clients. The list can be empty, for obvious reasons.

auth The current authentication type used by the server

*vencrypt** The vencrypt sub authentication type used by the server, only specified in case auth == vencrypt.

*display** The display device the vnc server is linked to.

Information about a vnc server

Since: 2.3

`VncInfo query-vnc ()` [Command]

Returns information about the current VNC server

Returns: *VncInfo*

Since: 0.14.0

Example:

```
 -> { "execute": "query-vnc" }
 <- { "return": {
          "enabled":true,
          "host":"0.0.0.0",
          "service":"50402",
          "auth":"vnc",
          "family":"ipv4",
          "clients":[
             {
                "host":"127.0.0.1",
                "service":"50401",
                "family":"ipv4"
             }
          ]
       }
    }
```

`['VncInfo2'] query-vnc-servers ()` [Command]

Returns a list of vnc servers. The list can be empty.

Returns: a list of *VncInfo2*

Since: 2.3

`SpiceBasicInfo` *{ 'host':* `str`*, 'port':* `str`*, 'family':* `NetworkAddressFamily` *}* [Struct]

host IP address

port port number

family address family

The basic information for SPICE network connection

Since: 2.1

`SpiceServerInfo` *{* [*'auth':* `str`] *}* [Struct]

*auth** authentication method

Information about a SPICE server

Since: 2.1

`SpiceChannel` *{ 'connection-id':* `int`*, 'channel-type':* `int`*, 'channel-id':* `int`*, 'tls':* `bool` *}* [Struct]

connection-id

SPICE connection id number. All channels with the same id belong to the same SPICE session.

channel-type

SPICE channel type number. "1" is the main control channel, filter for this one if you want to track spice sessions only

channel-id SPICE channel ID number. Usually `"0"`, might be different when multiple channels of the same type exist, such as multiple display channels in a multihead setup

tls true if the channel is encrypted, false otherwise.

Information about a SPICE client channel.

Since: 0.14.0

`SpiceQueryMouseMode` [Enum]

'`client`' Mouse cursor position is determined by the client.

'`server`' Mouse cursor position is determined by the server.

'`unknown`' No information is available about mouse mode used by the spice server.

An enumeration of Spice mouse states.

Note: spice/enums.h has a SpiceMouseMode already, hence the name.

Since: 1.1

`SpiceInfo` *{ 'enabled':* `bool`*, 'migrated':* `bool`*,* [*'host':* `str`]*,* [*'port':* `int`]*,* [*'tls-port':* `int`]*,* [*'auth':* `str`]*,* [*'compiled-version':* `str`]*, 'mouse-mode':* `SpiceQueryMouseMode`*,* [*'channels':* [`'SpiceChannel'`]] *}* [Struct]

enabled true if the SPICE server is enabled, false otherwise

migrated true if the last guest migration completed and spice migration had completed as well. false otherwise. (since 1.4)

*host** The hostname the SPICE server is bound to. This depends on the name resolution on the host and may be an IP address.

*port** The SPICE server's port number.

*compiled-version**
SPICE server version.

*tls-port** The SPICE server's TLS port number.

*auth** the current authentication type used by the server 'none' if no authentication is being used 'spice' uses SASL or direct TLS authentication, depending on command line options

mouse-mode
The mode in which the mouse cursor is displayed currently. Can be determined by the client or the server, or unknown if spice server doesn't provide this information. (since: 1.1)

channels a list of *SpiceChannel* for each active spice channel

Information about the SPICE session.

Since: 0.14.0

`SpiceInfo query-spice ()` [Command]

Returns information about the current SPICE server

Returns: *SpiceInfo*

Since: 0.14.0

Example:

```
-> { "execute": "query-spice" }
<- { "return": {
          "enabled": true,
          "auth": "spice",
          "port": 5920,
          "tls-port": 5921,
          "host": "0.0.0.0",
          "channels": [
             {
                "port": "54924",
                "family": "ipv4",
                "channel-type": 1,
                "connection-id": 1804289383,
                "host": "127.0.0.1",
                "channel-id": 0,
                "tls": true
             },
             {
                "port": "36710",
                "family": "ipv4",
                "channel-type": 4,
                "connection-id": 1804289383,
                "host": "127.0.0.1",
                "channel-id": 0,
                "tls": false
             },
             [ ... more channels follow ... ]
          ]
       }
    }
```

`BalloonInfo` *{ 'actual': `int` }* [Struct]

actual the number of bytes the balloon currently contains

Information about the guest balloon device.

Since: 0.14.0

`BalloonInfo query-balloon ()` [Command]

Return information about the balloon device.

Returns: *BalloonInfo* on success

If the balloon driver is enabled but not functional because the KVM kernel module cannot support it, KvmMissingCap

If no balloon device is present, DeviceNotActive

Since: 0.14.0

Example:

```
-> { "execute": "query-balloon" }
<- { "return": {
          "actual": 1073741824,
       }
   }
```

`PciMemoryRange` *{ 'base':* `int`*, 'limit':* `int` *}* [Struct]

base the starting address (guest physical)

limit the ending address (guest physical)

A PCI device memory region

Since: 0.14.0

`PciMemoryRegion` *{ 'bar':* `int`*, 'type':* `str`*, 'address':* `int`*, 'size':* `int`*,* [Struct]
['prefetch': `bool`*], ['mem_type_64':* `bool`*] }*

bar the index of the Base Address Register for this region

type 'io' if the region is a PIO region 'memory' if the region is a MMIO region

size memory size

*prefetch** if *type* is 'memory', true if the memory is prefetchable

*mem_type_64**
if *type* is 'memory', true if the BAR is 64-bit

Information about a PCI device I/O region.

Since: 0.14.0

`PciBusInfo` *{ 'number':* `int`*, 'secondary':* `int`*, 'subordinate':* `int`*,* [Struct]
'io_range': `PciMemoryRange`*, 'memory_range':* `PciMemoryRange`*,*
'prefetchable_range': `PciMemoryRange` *}*

number primary bus interface number. This should be the number of the bus the device resides on.

secondary secondary bus interface number. This is the number of the main bus for the bridge

subordinate
This is the highest number bus that resides below the bridge.

io_range The PIO range for all devices on this bridge

memory_range
The MMIO range for all devices on this bridge

prefetchable_range
: The range of prefetchable MMIO for all devices on this bridge

Information about a bus of a PCI Bridge device

Since: 2.4

`PciBridgeInfo` *{ 'bus':* `PciBusInfo`*, ['devices': ['PciDeviceInfo']] }* [Struct]

bus information about the bus the device resides on

devices a list of *PciDeviceInfo* for each device on this bridge

Information about a PCI Bridge device

Since: 0.14.0

`PciDeviceClass` *{ ['desc':* `str`*], 'class':* `int` *}* [Struct]

*desc** a string description of the device's class

class the class code of the device

Information about the Class of a PCI device

Since: 2.4

`PciDeviceId` *{ 'device':* `int`*, 'vendor':* `int` *}* [Struct]

device the PCI device id

vendor the PCI vendor id

Information about the Id of a PCI device

Since: 2.4

`PciDeviceInfo` *{ 'bus':* `int`*, 'slot':* `int`*, 'function':* `int`*, 'class_info':* `PciDeviceClass`*, 'id':* `PciDeviceId`*, ['irq':* `int`*], 'qdev_id':* `str`*, ['pci_bridge':* `PciBridgeInfo`*], 'regions': ['PciMemoryRegion'] }* [Struct]

bus the bus number of the device

slot the slot the device is located in

function the function of the slot used by the device

class_info the class of the device

id the PCI device id

*irq** if an IRQ is assigned to the device, the IRQ number

qdev_id the device name of the PCI device

pci_bridge if the device is a PCI bridge, the bridge information

regions a list of the PCI I/O regions associated with the device

Information about a PCI device

Notes: the contents of *class_info*.desc are not stable and should only be treated as informational.

Since: 0.14.0

`PciInfo` *{ 'bus':* `int`*, 'devices':* [`'PciDeviceInfo'`] *}* [Struct]

bus the bus index

devices a list of devices on this bus

Information about a PCI bus

Since: 0.14.0

`['PciInfo']` `query-pci` () [Command]

Return information about the PCI bus topology of the guest.

Returns: a list of *PciInfo* for each PCI bus. Each bus is represented by a json-object, which has a key with a json-array of all PCI devices attached to it. Each device is represented by a json-object.

Note: This example has been shortened as the real response is too long.

Since: 0.14.0

Example:

```
-> { "execute": "query-pci" }
<- { "return": [
          {
             "bus": 0,
             "devices": [
                {
                   "bus": 0,
                   "qdev_id": "",
                   "slot": 0,
                   "class_info": {
                      "class": 1536,
                      "desc": "Host bridge"
                   },
                   "id": {
                      "device": 32902,
                      "vendor": 4663
                   },
                   "function": 0,
                   "regions": [
                   ]
                },
                {
                   "bus": 0,
                   "qdev_id": "",
                   "slot": 1,
                   "class_info": {
                      "class": 1537,
                      "desc": "ISA bridge"
                   },
                   "id": {
```

```
            "device": 32902,
            "vendor": 28672
         },
         "function": 0,
         "regions": [
         ]
      },
      {
         "bus": 0,
         "qdev_id": "",
         "slot": 1,
         "class_info": {
            "class": 257,
            "desc": "IDE controller"
         },
         "id": {
            "device": 32902,
            "vendor": 28688
         },
         "function": 1,
         "regions": [
            {
               "bar": 4,
               "size": 16,
               "address": 49152,
               "type": "io"
            }
         ]
      },
      {
         "bus": 0,
         "qdev_id": "",
         "slot": 2,
         "class_info": {
            "class": 768,
            "desc": "VGA controller"
         },
         "id": {
            "device": 4115,
            "vendor": 184
         },
         "function": 0,
         "regions": [
            {
               "prefetch": true,
               "mem_type_64": false,
               "bar": 0,
```

```
                        "size": 33554432,
                        "address": 4026531840,
                        "type": "memory"
                     },
                     {
                        "prefetch": false,
                        "mem_type_64": false,
                        "bar": 1,
                        "size": 4096,
                        "address": 4060086272,
                        "type": "memory"
                     },
                     {
                        "prefetch": false,
                        "mem_type_64": false,
                        "bar": 6,
                        "size": 65536,
                        "address": -1,
                        "type": "memory"
                     }
                  ]
               },
               {
                  "bus": 0,
                  "qdev_id": "",
                  "irq": 11,
                  "slot": 4,
                  "class_info": {
                     "class": 1280,
                     "desc": "RAM controller"
                  },
                  "id": {
                     "device": 6900,
                     "vendor": 4098
                  },
                  "function": 0,
                  "regions": [
                     {
                        "bar": 0,
                        "size": 32,
                        "address": 49280,
                        "type": "io"
                     }
                  ]
               }
            ]
         }
```

```
           ]
        }
```

`quit` () [Command]

This command will cause the QEMU process to exit gracefully. While every attempt is made to send the QMP response before terminating, this is not guaranteed. When using this interface, a premature EOF would not be unexpected.

Since: 0.14.0

Example:

```
-> { "execute": "quit" }
<- { "return": {} }
```

`stop` () [Command]

Stop all guest VCPU execution.

Notes: This function will succeed even if the guest is already in the stopped state. In "inmigrate" state, it will ensure that the guest remains paused once migration finishes, as if the -S option was passed on the command line.

Since: 0.14.0

Example:

```
-> { "execute": "stop" }
<- { "return": {} }
```

`system_reset` () [Command]

Performs a hard reset of a guest.

Since: 0.14.0

Example:

```
-> { "execute": "system_reset" }
<- { "return": {} }
```

`system_powerdown` () [Command]

Requests that a guest perform a powerdown operation.

Notes: A guest may or may not respond to this command. This command returning does not indicate that a guest has accepted the request or that it has shut down. Many guests will respond to this command by prompting the user in some way.

Since: 0.14.0

Example:

```
-> { "execute": "system_powerdown" }
<- { "return": {} }
```

`cpu` (*'index':* `int`) [Command]

This command is a nop that is only provided for the purposes of compatibility.

Notes: Do not use this command.

Since: 0.14.0

`cpu-add` (*'id':* `int`) [Command]

id ID of CPU to be created, valid values [0..max_cpus)

Adds CPU with specified ID

Returns: Nothing on success

Since: 1.5

Example:

```
-> { "execute": "cpu-add", "arguments": { "id": 2 } }
<- { "return": {} }
```

`memsave` (*'val':* `int`, *'size':* `int`, *'filename':* `str`, [*'cpu-index':* `int`]) [Command]

val the virtual address of the guest to start from

size the size of memory region to save

filename the file to save the memory to as binary data

*cpu-index** the index of the virtual CPU to use for translating the virtual address (defaults to CPU 0)

Save a portion of guest memory to a file.

Returns: Nothing on success

Notes: Errors were not reliably returned until 1.1

Since: 0.14.0

Example:

```
-> { "execute": "memsave",
     "arguments": { "val": 10,
                    "size": 100,
                    "filename": "/tmp/virtual-mem-dump" } }
<- { "return": {} }
```

`pmemsave` (*'val':* `int`, *'size':* `int`, *'filename':* `str`) [Command]

val the physical address of the guest to start from

size the size of memory region to save

filename the file to save the memory to as binary data

Save a portion of guest physical memory to a file.

Returns: Nothing on success

Notes: Errors were not reliably returned until 1.1

Since: 0.14.0

Example:

```
-> { "execute": "pmemsave",
     "arguments": { "val": 10,
                    "size": 100,
                    "filename": "/tmp/physical-mem-dump" } }
<- { "return": {} }
```

`cont` () [Command]

Resume guest VCPU execution.

Returns: If successful, nothing If QEMU was started with an encrypted block device and a key has not yet been set, DeviceEncrypted.

Notes: This command will succeed if the guest is currently running. It will also succeed if the guest is in the "inmigrate" state; in this case, the effect of the command is to make sure the guest starts once migration finishes, removing the effect of the -S command line option if it was passed.

Since: 0.14.0

Example:

```
-> { "execute": "cont" }
<- { "return": {} }
```

`system_wakeup` () [Command]

Wakeup guest from suspend. Does nothing in case the guest isn't suspended.

Returns: nothing.

Since: 1.1

Example:

```
-> { "execute": "system_wakeup" }
<- { "return": {} }
```

`inject-nmi` () [Command]

Injects a Non-Maskable Interrupt into the default CPU (x86/s390) or all CPUs (ppc64). The command fails when the guest doesn't support injecting.

Returns: If successful, nothing

Note: prior to 2.1, this command was only supported for x86 and s390 VMs

Since: 0.14.0

Example:

```
-> { "execute": "inject-nmi" }
<- { "return": {} }
```

`set_link` (*'name':* `str`*, 'up':* `bool`) [Command]

name the device name of the virtual network adapter

up true to set the link status to be up

Sets the link status of a virtual network adapter.

Returns: Nothing on success If *name* is not a valid network device, DeviceNotFound

Notes: Not all network adapters support setting link status. This command will succeed even if the network adapter does not support link status notification.

Since: 0.14.0

Example:

```
-> { "execute": "set_link",
     "arguments": { "name": "e1000.0", "up": false } }
<- { "return": {} }
```

`balloon` (*'value':* `int`) [Command]

value the target size of the balloon in bytes

Request the balloon driver to change its balloon size.

Returns: Nothing on success If the balloon driver is enabled but not functional because the KVM kernel module cannot support it, KvmMissingCap If no balloon device is present, DeviceNotActive

Notes: This command just issues a request to the guest. When it returns, the balloon size may not have changed. A guest can change the balloon size independent of this command.

Since: 0.14.0

Example:

```
-> { "execute": "balloon", "arguments": { "value": 536870912 } }
<- { "return": {} }
```

`Abort` *{ }* [Struct]

This action can be used to test transaction failure.

Since: 1.6

`ActionCompletionMode` [Enum]

'`individual`'
: Do not attempt to cancel any other Actions if any Actions fail after the Transaction request succeeds. All Actions that can complete successfully will do so without waiting on others. This is the default.

'`grouped`' If any Action fails after the Transaction succeeds, cancel all Actions. Actions do not complete until all Actions are ready to complete. May be rejected by Actions that do not support this completion mode.

An enumeration of Transactional completion modes.

Since: 2.5

`TransactionAction` [*'blockdev-snapshot':* `BlockdevSnapshot`, *'blockdev-snapshot-sync':* `BlockdevSnapshotSync`, *'drive-backup':* `DriveBackup`, *'blockdev-backup':* `BlockdevBackup`, *'abort':* `Abort`, *'blockdev-snapshot-internal-sync':* `BlockdevSnapshotInternal`, *'block-dirty-bitmap-add':* `BlockDirtyBitmapAdd`, *'block-dirty-bitmap-clear':* `BlockDirtyBitmap`] [Union]

transaction.

A discriminated record of operations that can be performed with

Since: 1.1

drive-backup since 1.6 abort since 1.6 blockdev-snapshot-internal-sync since 1.7 blockdev-backup since 2.3 blockdev-snapshot since 2.5 block-dirty-bitmap-add since 2.5 block-dirty-bitmap-clear since 2.5

`TransactionProperties` *{ ['completion-mode':* `ActionCompletionMode`*] }* [Struct]

*completion-mode**

Controls how jobs launched asynchronously by Actions will complete or fail as a group. See *ActionCompletionMode* for details.

Optional arguments to modify the behavior of a Transaction.

Since: 2.5

`transaction` (*'actions':* [`'TransactionAction'`], [*'properties':* `TransactionProperties`]) [Command]

actions List of *TransactionAction*; information needed for the respective operations.

*properties**

structure of additional options to control the execution of the transaction. See *TransactionProperties* for additional detail.

Executes a number of transactionable QMP commands atomically. If any operation fails, then the entire set of actions will be abandoned and the appropriate error returned.

For external snapshots, the dictionary contains the device, the file to use for the new snapshot, and the format. The default format, if not specified, is qcow2.

Each new snapshot defaults to being created by QEMU (wiping any contents if the file already exists), but it is also possible to reuse an externally-created file. In the latter case, you should ensure that the new image file has the same contents as the current one; QEMU cannot perform any meaningful check. Typically this is achieved by using the current image file as the backing file for the new image.

On failure, the original disks pre-snapshot attempt will be used.

For internal snapshots, the dictionary contains the device and the snapshot's name. If an internal snapshot matching name already exists, the request will be rejected. Only some image formats support it, for example, qcow2, rbd, and sheepdog.

On failure, qemu will try delete the newly created internal snapshot in the transaction. When an I/O error occurs during deletion, the user needs to fix it later with qemu-img or other command.

Returns: nothing on success

Errors depend on the operations of the transaction

Note: The transaction aborts on the first failure. Therefore, there will be information on only one failed operation returned in an error condition, and subsequent actions will not have been attempted.

Since: 1.1

Example:

```
 -> { "execute": "transaction",
      "arguments": { "actions": [
          { "type": "blockdev-snapshot-sync", "data" : { "device": "ide-hd0",
                                      "snapshot-file": "/some/place/my-image",
```

```
                                    "format": "qcow2" } },
              { "type": "blockdev-snapshot-sync", "data" : { "node-name": "myfile",
                                    "snapshot-file": "/some/place/my-image2",
                                    "snapshot-node-name": "node3432",
                                    "mode": "existing",
                                    "format": "qcow2" } },
              { "type": "blockdev-snapshot-sync", "data" : { "device": "ide-hd1",
                                    "snapshot-file": "/some/place/my-image2",
                                    "mode": "existing",
                                    "format": "qcow2" } },
              { "type": "blockdev-snapshot-internal-sync", "data" : {
                                    "device": "ide-hd2",
                                    "name": "snapshot0" } } ] } }
<- { "return": {} }
```

`str human-monitor-command` (*'command-line':* `str`, [*'cpu-index':* `int`]) [Command]

command-line
: the command to execute in the human monitor

*cpu-index**
: The CPU to use for commands that require an implicit CPU

Execute a command on the human monitor and return the output.

Returns: the output of the command as a string

Notes: This command only exists as a stop-gap. Its use is highly discouraged. The semantics of this command are not guaranteed: this means that command names, arguments and responses can change or be removed at ANY time. Applications that rely on long term stability guarantees should NOT use this command.

Known limitations:

- This command is stateless, this means that commands that depend on state information (such as getfd) might not work
- Commands that prompt the user for data (eg. 'cont' when the block device is encrypted) don't currently work

Since: 0.14.0

Example:

```
-> { "execute": "human-monitor-command",
     "arguments": { "command-line": "info kvm" } }
<- { "return": "kvm support: enabled\r\n" }
```

`migrate_cancel` () [Command]

Cancel the current executing migration process.

Returns: nothing on success

Notes: This command succeeds even if there is no migration process running.

Since: 0.14.0

Example:

```
-> { "execute": "migrate_cancel" }
<- { "return": {} }
```

`migrate_set_downtime` (*'value':* ***number***) [Command]

value maximum downtime in seconds

Set maximum tolerated downtime for migration.

Returns: nothing on success

Since: 0.14.0

Example:

```
-> { "execute": "migrate_set_downtime", "arguments": { "value": 0.1 } }
<- { "return": {} }
```

`migrate_set_speed` (*'value':* ***int***) [Command]

value maximum speed in bytes per second.

Set maximum speed for migration.

Returns: nothing on success

Notes: A value lesser than zero will be automatically round up to zero.

Since: 0.14.0

Example:

```
-> { "execute": "migrate_set_speed", "arguments": { "value": 1024 } }
<- { "return": {} }
```

`migrate-set-cache-size` (*'value':* ***int***) [Command]

value cache size in bytes

The size will be rounded down to the nearest power of 2. The cache size can be modified before and during ongoing migration

Set cache size to be used by XBZRLE migration

Returns: nothing on success

Since: 1.2

Example:

```
-> { "execute": "migrate-set-cache-size",
     "arguments": { "value": 536870912 } }
<- { "return": {} }
```

`int query-migrate-cache-size ()` [Command]

Query migration XBZRLE cache size

Returns: XBZRLE cache size in bytes

Since: 1.2

Example:

```
-> { "execute": "query-migrate-cache-size" }
<- { "return": 67108864 }
```

`ObjectPropertyInfo` *{ 'name':* `str`*, 'type':* `str` *}* [Struct]

name the name of the property

type the type of the property. This will typically come in one of four forms:

1. A primitive type such as 'u8', 'u16', 'bool', 'str', or 'double'. These types are mapped to the appropriate JSON type.
2. A child type in the form 'child<subtype>' where subtype is a qdev device type name. Child properties create the composition tree.
3. A link type in the form 'link<subtype>' where subtype is a qdev device type name. Link properties form the device model graph.

Since: 1.2

`['ObjectPropertyInfo'] qom-list` (*'path':* `str`) [Command]

path the path within the object model. See *qom*-get for a description of this parameter.

This command will list any properties of a object given a path in the object model.

Returns: a list of *ObjectPropertyInfo* that describe the properties of the object.

Since: 1.2

`any qom-get` (*'path':* `str`*, 'property':* `str`) [Command]

path The path within the object model. There are two forms of supported paths–absolute and partial paths.

Absolute paths are derived from the root object and can follow child<> or link<> properties. Since they can follow link<> properties, they can be arbitrarily long. Absolute paths look like absolute filenames and are prefixed with a leading slash.

Partial paths look like relative filenames. They do not begin with a prefix. The matching rules for partial paths are subtle but designed to make specifying objects easy. At each level of the composition tree, the partial path is matched as an absolute path. The first match is not returned. At least two matches are searched for. A successful result is only returned if only one match is found. If more than one match is found, a flag is return to indicate that the match was ambiguous.

property The property name to read

This command will get a property from a object model path and return the value.

Returns: The property value. The type depends on the property type. child<> and link<> properties are returned as #str pathnames. All integer property types (u8, u16, etc) are returned as #int.

Since: 1.2

`qom-set` (*'path':* `str`*, 'property':* `str`*, 'value':* `any`) [Command]

path see *qom*-get for a description of this parameter

property the property name to set

value a value who's type is appropriate for the property type. See *qom*-get for a description of type mapping.

This command will set a property from a object model path.

Since: 1.2

`set_password` (*'protocol':* `str`, *'password':* `str`, [*'connected':* `str`]) [Command]

protocol 'vnc' to modify the VNC server password 'spice' to modify the Spice server password

password the new password

*connected** how to handle existing clients when changing the password. If nothing is specified, defaults to 'keep' 'fail' to fail the command if clients are connected 'disconnect' to disconnect existing clients 'keep' to maintain existing clients

Sets the password of a remote display session.

Returns: Nothing on success If Spice is not enabled, DeviceNotFound

Since: 0.14.0

Example:

```
-> { "execute": "set_password", "arguments": { "protocol": "vnc",
                                               "password": "secret" } }
<- { "return": {} }
```

`expire_password` (*'protocol':* `str`, *'time':* `str`) [Command]

protocol the name of the remote display protocol 'vnc' or 'spice'

time when to expire the password. 'now' to expire the password immediately 'never' to cancel password expiration '+INT' where INT is the number of seconds from now (integer) 'INT' where INT is the absolute time in seconds

Expire the password of a remote display server.

Returns: Nothing on success If *protocol* is 'spice' and Spice is not active, DeviceNotFound

Notes: Time is relative to the server and currently there is no way to coordinate server time with client time. It is not recommended to use the absolute time version of the *time* parameter unless you're sure you are on the same machine as the QEMU instance.

Since: 0.14.0

Example:

```
-> { "execute": "expire_password", "arguments": { "protocol": "vnc",
                                                  "time": "+60" } }
<- { "return": {} }
```

`change-vnc-password` (*'password':* `str`) [Command]

password the new password to use with VNC authentication

Change the VNC server password.

Notes: An empty password in this command will set the password to the empty string. Existing clients are unaffected by executing this command.

Since: 1.1

`change` (*'device':* `str`, *'target':* `str`, [*'arg':* `str`]) [Command]

device This is normally the name of a block device but it may also be 'vnc'. when it's 'vnc', then sub command depends on *target*

target If *device* is a block device, then this is the new filename. If *device* is 'vnc', then if the value 'password' selects the vnc change password command. Otherwise, this specifies a new server URI address to listen to for VNC connections.

arg If *device* is a block device, then this is an optional format to open the device with. If *device* is 'vnc' and *target* is 'password', this is the new VNC password to set. If this argument is an empty string, then no future logins will be allowed.

This command is multiple commands multiplexed together.

Returns: Nothing on success. If *device* is not a valid block device, DeviceNotFound If the new block device is encrypted, DeviceEncrypted. Note that if this error is returned, the device has been opened successfully and an additional call to *block_passwd* is required to set the device's password. The behavior of reads and writes to the block device between when these calls are executed is undefined.

Notes: This interface is deprecated, and it is strongly recommended that you avoid using it. For changing block devices, use blockdev-change-medium; for changing VNC parameters, use change-vnc-password.

Since: 0.14.0

Example:

```
1. Change a removable medium

-> { "execute": "change",
     "arguments": { "device": "ide1-cd0",
                    "target": "/srv/images/Fedora-12-x86_64-DVD.iso" } }
<- { "return": {} }

2. Change VNC password

-> { "execute": "change",
     "arguments": { "device": "vnc", "target": "password",
                    "arg": "foobar1" } }
<- { "return": {} }
```

`ObjectTypeInfo` *{ 'name':* `str` *}* [Struct]

name the type name found in the search

This structure describes a search result from *qom*-list-types

Notes: This command is experimental and may change syntax in future releases.

Since: 1.1

`['ObjectTypeInfo'] qom-list-types` (*['implements':* `str`], *['abstract':* `bool`]) [Command]

implements
if specified, only return types that implement this type name

abstract if true, include abstract types in the results

This command will return a list of types given search parameters

Returns: a list of *ObjectTypeInfo* or an empty list if no results are found

Since: 1.1

`DevicePropertyInfo` *{ 'name':* `str`*, 'type':* `str`*, ['description':* `str`*] }* [Struct]

name the name of the property

type the typename of the property

*description**
if specified, the description of the property. (since 2.2)

Information about device properties.

Since: 1.2

`['DevicePropertyInfo'] device-list-properties` (*'typename':* `str`) [Command]

typename the type name of a device

List properties associated with a device.

Returns: a list of DevicePropertyInfo describing a devices properties

Since: 1.2

`migrate` (*'uri':* `str`*, ['blk':* `bool`*], ['inc':* `bool`*], ['detach':* `bool`*]*) [Command]

uri the Uniform Resource Identifier of the destination VM

*blk** do block migration (full disk copy)

*inc** incremental disk copy migration

detach this argument exists only for compatibility reasons and is ignored by QEMU

Migrates the current running guest to another Virtual Machine.

1. The 'query-migrate' command should be used to check migration's progress and final result (this information is provided by the 'status' member)
2. All boolean arguments default to false

3. The user Monitor's "detach" argument is invalid in QMP and should not be used

Returns: nothing on success

Notes:

Since: 0.14.0

Example:

```
-> { "execute": "migrate", "arguments": { "uri": "tcp:0:4446" } }
<- { "return": {} }
```

`migrate-incoming` (*'uri':* `str`) [Command]

uri The Uniform Resource Identifier identifying the source or address to listen on

Start an incoming migration, the qemu must have been started with -incoming defer

1. It's a bad idea to use a string for the uri, but it needs to stay compatible with -incoming and the format of the uri is already exposed above libvirt.
2. QEMU must be started with -incoming defer to allow migrate-incoming to be used.
3. The uri format is the same as for -incoming

Returns: nothing on success

Notes:

Since: 2.3

Example:

```
-> { "execute": "migrate-incoming",
     "arguments": { "uri": "tcp::4446" } }
<- { "return": {} }
```

`xen-save-devices-state` (*'filename':* `str`) [Command]

filename the file to save the state of the devices to as binary data. See xen-save-devices-state.txt for a description of the binary format.

Save the state of all devices to file. The RAM and the block devices of the VM are not saved by this command.

Returns: Nothing on success

Since: 1.1

Example:

```
-> { "execute": "xen-save-devices-state",
     "arguments": { "filename": "/tmp/save" } }
<- { "return": {} }
```

`xen-set-global-dirty-log` (*'enable':* `bool`) [Command]

enable true to enable, false to disable.

Enable or disable the global dirty log mode.

Returns: nothing

Since: 1.3

Example:

```
-> { "execute": "xen-set-global-dirty-log",
     "arguments": { "enable": true } }
<- { "return": {} }
```

`device_add` (*'driver': `str`, 'id': `str`*) [Command]

driver	the name of the new device's driver
*bus**	the device's parent bus (device tree path)
id	the device's ID, must be unique
	Additional arguments depend on the type.

Add a device.

Notes:

1. For detailed information about this command, please refer to the 'docs/qdev-device-use.txt' file.
2. It's possible to list device properties by running QEMU with the "-device DEVICE,help" command-line argument, where DEVICE is the device's name

Since: 0.13

Example:

```
-> { "execute": "device_add",
     "arguments": { "driver": "e1000", "id": "net1",
                    "bus": "pci.0",
                    "mac": "52:54:00:12:34:56" } }
<- { "return": {} }

TODO This command effectively bypasses QAPI completely due to its
"additional arguments" business.  It shouldn't have been added to
the schema in this form.  It should be qapified properly, or
replaced by a properly qapified command.
```

`device_del` (*'id': `str`*) [Command]

id	the device's ID or QOM path

Remove a device from a guest

Returns: Nothing on success If *id* is not a valid device, DeviceNotFound

Notes: When this command completes, the device may not be removed from the guest. Hot removal is an operation that requires guest cooperation. This command merely requests that the guest begin the hot removal process. Completion of the device removal process is signaled with a DEVICE_DELETED event. Guest reset will automatically complete removal for all devices.

Since: 0.14.0

Example:

```
-> { "execute": "device_del",
     "arguments": { "id": "net1" } }
<- { "return": {} }

-> { "execute": "device_del",
     "arguments": { "id": "/machine/peripheral-anon/device[0]" } }
<- { "return": {} }
```

`DumpGuestMemoryFormat` [Enum]

'`elf`' elf format

'`kdump-zlib`'
kdump-compressed format with zlib-compressed

'`kdump-lzo`'
kdump-compressed format with lzo-compressed

'`kdump-snappy`'
kdump-compressed format with snappy-compressed

An enumeration of guest-memory-dump's format.

Since: 2.0

`dump-guest-memory` (*'paging':* `bool`, *'protocol':* `str`, [*'detach':* `bool`], [*'begin':* `int`], [*'length':* `int`], [*'format':* *DumpGuestMemoryFormat*]) [Command]

paging if true, do paging to get guest's memory mapping. This allows using gdb to process the core file.

IMPORTANT: this option can make QEMU allocate several gigabytes of RAM. This can happen for a large guest, or a malicious guest pretending to be large.

Also, paging=true has the following limitations:

1. The guest may be in a catastrophic state or can have corrupted memory, which cannot be trusted
2. The guest can be in real-mode even if paging is enabled. For example, the guest uses ACPI to sleep, and ACPI sleep state goes in real-mode
3. Currently only supported on i386 and x86_64.

protocol the filename or file descriptor of the vmcore. The supported protocols are:

1. file: the protocol starts with "file:", and the following string is the file's path.
2. fd: the protocol starts with "fd:", and the following string is the fd's name.

*detach** if true, QMP will return immediately rather than waiting for the dump to finish. The user can track progress using "query-dump". (since 2.6).

*begin** if specified, the starting physical address.

length is not allowed to be specified with non-elf *format* at the same time (since 2.0)

*format** if specified, the format of guest memory dump. But non-elf format is conflict with paging and filter, ie. *paging*, *begin* and

Dump guest's memory to vmcore. It is a synchronous operation that can take very long depending on the amount of guest memory.

Returns: nothing on success

Note: All boolean arguments default to false

Since: 1.2

Example:

```
-> { "execute": "dump-guest-memory",
     "arguments": { "protocol": "fd:dump" } }
<- { "return": {} }
```

`DumpStatus` [Enum]

'`none`' no dump-guest-memory has started yet.

'`active`' there is one dump running in background.

'`completed`'
the last dump has finished successfully.

'`failed`' the last dump has failed.

Describe the status of a long-running background guest memory dump.

Since: 2.6

`DumpQueryResult` *{ 'status':* `DumpStatus`*, 'completed':* `int`*, 'total':* `int` *}* [Struct]

status enum of *DumpStatus*, which shows current dump status

completed bytes written in latest dump (uncompressed)

total total bytes to be written in latest dump (uncompressed)

The result format for 'query-dump'.

Since: 2.6

`DumpQueryResult query-dump ()` [Command]

Query latest dump status.

Returns: A *DumpStatus* object showing the dump status.

Since: 2.6

Example:

```
-> { "execute": "query-dump" }
<- { "return": { "status": "active", "completed": 1024000,
                 "total": 2048000 } }
```

`DumpGuestMemoryCapability` *{ 'formats':* [*'DumpGuestMemoryFormat'*] *}* [Struct]

A list of the available formats for dump-guest-memory

Since: 2.0

`DumpGuestMemoryCapability` [Command]

`query-dump-guest-memory-capability` ()

Returns the available formats for dump-guest-memory

Returns: A *DumpGuestMemoryCapability* object listing available formats for dump-guest-memory

Since: 2.0

Example:

```
-> { "execute": "query-dump-guest-memory-capability" }
<- { "return": { "formats":
                 ["elf", "kdump-zlib", "kdump-lzo", "kdump-snappy"] }
```

`dump-skeys` (*'filename':* ***str***) [Command]

filename the path to the file to dump to

This command is only supported on s390 architecture.

Dump guest's storage keys

Since: 2.5

Example:

```
-> { "execute": "dump-skeys",
     "arguments": { "filename": "/tmp/skeys" } }
<- { "return": {} }
```

`netdev_add` (*'type':* ***str***, *'id':* ***str***) [Command]

type the type of network backend. Current valid values are 'user', 'tap', 'vde', 'socket', 'dump' and 'bridge'

id the name of the new network backend

Additional arguments depend on the type.

TODO This command effectively bypasses QAPI completely due to its "additional arguments" business. It shouldn't have been added to the schema in this form. It should be qapified properly, or replaced by a properly qapified command.

Add a network backend.

Returns: Nothing on success If *type* is not a valid network backend, DeviceNotFound

Since: 0.14.0

Example:

```
-> { "execute": "netdev_add",
     "arguments": { "type": "user", "id": "netdev1",
                    "dnssearch": "example.org" } }
<- { "return": {} }
```

`netdev_del` (*'id': `str`*) [Command]

id the name of the network backend to remove

Remove a network backend.

Returns: Nothing on success If *id* is not a valid network backend, DeviceNotFound

Since: 0.14.0

Example:

```
-> { "execute": "netdev_del", "arguments": { "id": "netdev1" } }
<- { "return": {} }
```

`object-add` (*'qom-type': `str`, 'id': `str`, ['props': `any`]*) [Command]

qom-type the class name for the object to be created

id the name of the new object

*props** a dictionary of properties to be passed to the backend

Create a QOM object.

Returns: Nothing on success Error if *qom*-type is not a valid class name

Since: 2.0

Example:

```
-> { "execute": "object-add",
     "arguments": { "qom-type": "rng-random", "id": "rng1",
                    "props": { "filename": "/dev/hwrng" } } }
<- { "return": {} }
```

`object-del` (*'id': `str`*) [Command]

id the name of the QOM object to remove

Remove a QOM object.

Returns: Nothing on success Error if *id* is not a valid id for a QOM object

Since: 2.0

Example:

```
-> { "execute": "object-del", "arguments": { "id": "rng1" } }
<- { "return": {} }
```

`NetdevNoneOptions` *{ }* [Struct]

Use it alone to have zero network devices.

Since: 1.2

`NetLegacyNicOptions` *{ ['netdev': `str`], ['macaddr': `str`], ['model': `str`], ['addr': `str`], ['vectors': `uint32`] }* [Struct]

*netdev** id of -netdev to connect to

*macaddr** MAC address

*model** device model (e1000, rtl8139, virtio etc.)

*addr** PCI device address

*vectors** number of MSI-x vectors, 0 to disable MSI-X

Create a new Network Interface Card.

Since: 1.2

`String` *{ 'str': `str` }* [Struct]

A fat type wrapping 'str', to be embedded in lists.

Since: 1.2

`NetdevUserOptions` *{ ['hostname': `str`], ['restrict': `bool`], ['ipv4': `bool`], ['ipv6': `bool`], ['ip': `str`], ['net': `str`], ['host': `str`], ['tftp': `str`], ['bootfile': `str`], ['dhcpstart': `str`], ['dns': `str`], ['dnssearch': ['`String`']], ['ipv6-prefix': `str`], ['ipv6-prefixlen': `int`], ['ipv6-host': `str`], ['ipv6-dns': `str`], ['smb': `str`], ['smbserver': `str`], ['hostfwd': ['`String`']], ['guestfwd': ['`String`']] }* [Struct]

*hostname** client hostname reported by the builtin DHCP server

*restrict** isolate the guest from the host

*ipv4** whether to support IPv4, default true for enabled (since 2.6)

*ipv6** whether to support IPv6, default true for enabled (since 2.6)

*ip** legacy parameter, use net= instead

*net** IP network address that the guest will see, in the form addr[/netmask] The netmask is optional, and can be either in the form a.b.c.d or as a number of valid top-most bits. Default is 10.0.2.0/24.

*host** guest-visible address of the host

*tftp** root directory of the built-in TFTP server

*bootfile** BOOTP filename, for use with tftp=

*dhcpstart** the first of the 16 IPs the built-in DHCP server can assign

*dns** guest-visible address of the virtual nameserver

*dnssearch** list of DNS suffixes to search, passed as DHCP option to the guest

*ipv6-prefix** IPv6 network prefix (default is fec0::) (since 2.6). The network prefix is given in the usual hexadecimal IPv6 address notation.

*ipv6-prefixlen** IPv6 network prefix length (default is 64) (since 2.6)

*ipv6-host** guest-visible IPv6 address of the host (since 2.6)

*ipv6-dns** guest-visible IPv6 address of the virtual nameserver (since 2.6)

*smb** root directory of the built-in SMB server

*smbserver**
: IP address of the built-in SMB server

*hostfwd**
: redirect incoming TCP or UDP host connections to guest endpoints

*guestfwd**
: forward guest TCP connections

Use the user mode network stack which requires no administrator privilege to run.

Since: 1.2

`NetdevTapOptions` *{* ['*ifname*': `str`], ['*fd*': `str`], ['*fds*': `str`], ['*script*': `str`], ['*downscript*': `str`], ['*br*': `str`], ['*helper*': `str`], ['*sndbuf*': `size`], ['*vnet_hdr*': `bool`], ['*vhost*': `bool`], ['*vhostfd*': `str`], ['*vhostfds*': `str`], ['*vhostforce*': `bool`], ['*queues*': `uint32`], ['*poll-us*': `uint32`] *}* [Struct]

*ifname**
: interface name

*fd**
: file descriptor of an already opened tap

*fds**
: multiple file descriptors of already opened multiqueue capable tap

*script**
: script to initialize the interface

*downscript**
: script to shut down the interface

*br**
: bridge name (since 2.8)

*helper**
: command to execute to configure bridge

*sndbuf**
: send buffer limit. Understands [TGMKkb] suffixes.

*vnet_hdr**
: enable the IFF_VNET_HDR flag on the tap interface

*vhost**
: enable vhost-net network accelerator

*vhostfd**
: file descriptor of an already opened vhost net device

*vhostfds**
: file descriptors of multiple already opened vhost net devices

*vhostforce**
: vhost on for non-MSIX virtio guests

*queues**
: number of queues to be created for multiqueue capable tap

*poll-us**
: maximum number of microseconds that could be spent on busy polling for tap (since 2.7)

Connect the host TAP network interface name to the VLAN.

Since: 1.2

`NetdevSocketOptions` *{* ['*fd*': `str`], ['*listen*': `str`], ['*connect*': `str`], ['*mcast*': `str`], ['*localaddr*': `str`], ['*udp*': `str`] *}* [Struct]

*fd**
: file descriptor of an already opened socket

*listen**
: port number, and optional hostname, to listen on

*connect**
: port number, and optional hostname, to connect to

*mcast** UDP multicast address and port number

*localaddr** source address and port for multicast and udp packets

*udp** UDP unicast address and port number

Connect the VLAN to a remote VLAN in another QEMU virtual machine using a TCP socket connection.

Since: 1.2

`NetdevL2TPv3Options` *{ 'src':* `str`*, 'dst':* `str`*, ['srcport':* `str`*], ['dstport':* `str`*], ['ipv6':* `bool`*], ['udp':* `bool`*], ['cookie64':* `bool`*], ['counter':* `bool`*], ['pincounter':* `bool`*], ['txcookie':* `uint64`*], ['rxcookie':* `uint64`*], 'txsession':* `uint32`*, ['rxsession':* `uint32`*], ['offset':* `uint32`*] }* [Struct]

src source address

dst destination address

*srcport** source port - mandatory for udp, optional for ip

*dstport** destination port - mandatory for udp, optional for ip

*ipv6**

- force the use of ipv6

*udp**

- use the udp version of l2tpv3 encapsulation

*cookie64**

- use 64 bit coookies

*counter** have sequence counter

*pincounter**

pin sequence counter to zero - workaround for buggy implementations or networks with packet reorder

*txcookie** 32 or 64 bit transmit cookie

*rxcookie** 32 or 64 bit receive cookie

txsession 32 bit transmit session

*rxsession** 32 bit receive session - if not specified set to the same value as transmit

*offset** additional offset - allows the insertion of additional application-specific data before the packet payload

Connect the VLAN to Ethernet over L2TPv3 Static tunnel

Since: 2.1

`NetdevVdeOptions` *{ ['sock':* `str`*], ['port':* `uint16`*], ['group':* `str`*], ['mode':* `uint16`*] }* [Struct]

*sock** socket path

*port** port number

*group** group owner of socket

*mode** permissions for socket

Connect the VLAN to a vde switch running on the host.

Since: 1.2

`NetdevDumpOptions` *{ ['len': `size`], ['file': `str`] }* [Struct]

*len** per-packet size limit (64k default). Understands [TGMKkb] suffixes.

*file** dump file path (default is qemu-vlan0.pcap)

Dump VLAN network traffic to a file.

Since: 1.2

`NetdevBridgeOptions` *{ ['br': `str`], ['helper': `str`] }* [Struct]

*br** bridge name

*helper** command to execute to configure bridge

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

Since: 1.2

`NetdevHubPortOptions` *{ 'hubid': `int32` }* [Struct]

hubid hub identifier number

Connect two or more net clients through a software hub.

Since: 1.2

`NetdevNetmapOptions` *{ 'ifname': `str`, ['devname': `str`] }* [Struct]

ifname Either the name of an existing network interface supported by netmap, or the name of a VALE port (created on the fly). A VALE port name is in the form 'valeXXX:YYY', where XXX and YYY are non-negative integers. XXX identifies a switch and YYY identifies a port of the switch. VALE ports having the same XXX are therefore connected to the same switch.

*devname** path of the netmap device (default: '/dev/netmap').

Connect a client to a netmap-enabled NIC or to a VALE switch port

Since: 2.0

`NetdevVhostUserOptions` *{ 'chardev': `str`, ['vhostforce': `bool`], ['queues': `int`] }* [Struct]

chardev name of a unix socket chardev

*vhostforce** vhost on for non-MSIX virtio guests (default: false).

*queues** number of queues to be created for multiqueue vhost-user (default: 1) (Since 2.5)

Vhost-user network backend

Since: 2.1

`NetClientDriver` [Enum]

Available netdev drivers.

Since: 2.7

`Netdev` [*'none':* `NetdevNoneOptions`, *'nic':* `NetLegacyNicOptions`, *'user':* `NetdevUserOptions`, *'tap':* `NetdevTapOptions`, *'l2tpv3':* `NetdevL2TPv3Options`, *'socket':* `NetdevSocketOptions`, *'vde':* `NetdevVdeOptions`, *'dump':* `NetdevDumpOptions`, *'bridge':* `NetdevBridgeOptions`, *'hubport':* `NetdevHubPortOptions`, *'netmap':* `NetdevNetmapOptions`, *'vhost-user':* `NetdevVhostUserOptions`] [Union]

id identifier for monitor commands.

type Specify the driver used for interpreting remaining arguments.

Captures the configuration of a network device.

Since: 1.2

'l2tpv3' - since 2.1

`NetLegacy` *{* [*'vlan':* `int32`], [*'id':* `str`], [*'name':* `str`], *'opts':* `NetLegacyOptions` *}* [Struct]

*vlan** vlan number

*id** identifier for monitor commands

*name** identifier for monitor commands, ignored if *id* is present

opts device type specific properties (legacy)

Captures the configuration of a network device; legacy.

Since: 1.2

`NetLegacyOptions` [*'none':* `NetdevNoneOptions`, *'nic':* `NetLegacyNicOptions`, *'user':* `NetdevUserOptions`, *'tap':* `NetdevTapOptions`, *'l2tpv3':* `NetdevL2TPv3Options`, *'socket':* `NetdevSocketOptions`, *'vde':* `NetdevVdeOptions`, *'dump':* `NetdevDumpOptions`, *'bridge':* `NetdevBridgeOptions`, *'netmap':* `NetdevNetmapOptions`, *'vhost-user':* `NetdevVhostUserOptions`] [Union]

Like Netdev, but for use only by the legacy command line options

Since: 1.2

`NetFilterDirection` [Enum]

'`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.

Indicates whether a netfilter is attached to a netdev's transmit queue or receive queue or both.

Since: 2.5

`InetSocketAddress` *{ 'host':* `str`*, 'port':* `str`*, ['to':* `uint16`*], ['ipv4':* `bool`*], ['ipv6':* `bool`*] }* [Struct]

host host part of the address

port port part of the address, or lowest port if *to* is present

to highest port to try

ipv4 whether to accept IPv4 addresses, default try both IPv4 and IPv6 #optional

ipv6 whether to accept IPv6 addresses, default try both IPv4 and IPv6 #optional

Captures a socket address or address range in the Internet namespace.

Since: 1.3

`UnixSocketAddress` *{ 'path':* `str` *}* [Struct]

path filesystem path to use

Captures a socket address in the local ("Unix socket") namespace.

Since: 1.3

`SocketAddress` [*'inet':* `InetSocketAddress`*, 'unix':* `UnixSocketAddress`*, 'fd':* `String`] [Union]

Captures the address of a socket, which could also be a named file descriptor

Since: 1.3

`getfd` (*'fdname':* `str`) [Command]

fdname file descriptor name

Receive a file descriptor via SCM rights and assign it a name

Returns: Nothing on success

Notes: If *fdname* already exists, the file descriptor assigned to it will be closed and replaced by the received file descriptor.

The 'closefd' command can be used to explicitly close the file descriptor when it is no longer needed.

Since: 0.14.0

Example:

```
-> { "execute": "getfd", "arguments": { "fdname": "fd1" } }
<- { "return": {} }
```

`closefd` (*'fdname':* `str`) [Command]

fdname file descriptor name

Close a file descriptor previously passed via SCM rights

Returns: Nothing on success

Since: 0.14.0

Example:

```
-> { "execute": "closefd", "arguments": { "fdname": "fd1" } }
<- { "return": {} }
```

`MachineInfo` *{ 'name':* `str`*, ['alias':* `str`*], ['is-default':* `bool`*], 'cpu-max':* `int`*, 'hotpluggable-cpus':* `bool` *}* [Struct]

name the name of the machine

*alias** an alias for the machine name

*default** whether the machine is default

cpu-max maximum number of CPUs supported by the machine type (since 1.5.0)

hotpluggable-cpus
cpu hotplug via -device is supported (since 2.7.0)

Information describing a machine.

Since: 1.2.0

`['MachineInfo'] query-machines ()` [Command]

Return a list of supported machines

Returns: a list of MachineInfo

Since: 1.2.0

`CpuDefinitionInfo` *{ 'name':* `str`*, ['migration-safe':* `bool`*], 'static':* `bool` *}* [Struct]

name the name of the CPU definition

*migration-safe**
whether a CPU definition can be safely used for migration in combination with a QEMU compatibility machine when migrating between different QMU versions and between hosts with different sets of (hardware or software) capabilities. If not provided, information is not available and callers should not assume the CPU definition to be migration-safe. (since 2.8)

static whether a CPU definition is static and will not change depending on QEMU version, machine type, machine options and accelerator options. A static model is always migration-safe. (since 2.8)

Virtual CPU definition.

Since: 1.2.0

`['CpuDefinitionInfo'] query-cpu-definitions ()` [Command]

Return a list of supported virtual CPU definitions

Returns: a list of CpuDefInfo

Since: 1.2.0

`CpuModelInfo` *{ 'name':* `str`*, ['props':* `any`*] }* [Struct]

name the name of the CPU definition the model is based on

*props** a dictionary of QOM properties to be applied

Virtual CPU model.

A CPU model consists of the name of a CPU definition, to which delta changes are applied (e.g. features added/removed). Most magic values that an architecture might require should be hidden behind the name. However, if required, architectures can expose relevant properties.

Since: 2.8.0

`CpuModelExpansionType` [Enum]

'`static`' Expand to a static CPU model, a combination of a static base model name and property delta changes. As the static base model will never change, the expanded CPU model will be the same, independant of independent of QEMU version, machine type, machine options, and accelerator options. Therefore, the resulting model can be used by tooling without having to specify a compatibility machine - e.g. when displaying the "host" model. static CPU models are migration-safe.

'`full`' Expand all properties. The produced model is not guaranteed to be migration-safe, but allows tooling to get an insight and work with model details.

An enumeration of CPU model expansion types.

Since: 2.8.0

`CpuModelExpansionInfo` *{ 'model': `CpuModelInfo` }* [Struct]

model the expanded CpuModelInfo.

The result of a cpu model expansion.

Since: 2.8.0

`CpuModelExpansionInfo query-cpu-model-expansion` (*'type': `CpuModelExpansionType`, 'model': `CpuModelInfo`*) [Command]

Expands a given CPU model (or a combination of CPU model + additional options) to different granularities, allowing tooling to get an understanding what a specific CPU model looks like in QEMU under a certain configuration.

This interface can be used to query the "host" CPU model.

The data returned by this command may be affected by:

- QEMU version: CPU models may look different depending on the QEMU version. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine-type: CPU model may look different depending on the machine-type. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine options (including accelerator): in some architectures, CPU models may look different depending on machine and accelerator options. (Except for CPU models reported as "static" in query-cpu-definitions.)
- "-cpu" arguments and global properties: arguments to the -cpu option and global properties may affect expansion of CPU models. Using query-cpu-model-expansion while using these is not advised.

Some architectures may not support all expansion types. s390x supports "full" and "static".

Returns: a CpuModelExpansionInfo. Returns an error if expanding CPU models is not supported, if the model cannot be expanded, if the model contains an unknown CPU definition name, unknown properties or properties with a wrong type. Also returns an error if an expansion type is not supported.

Since: 2.8.0

`CpuModelCompareResult` [Enum]

'`incompatible`'
: If model A is incompatible to model B, model A is not guaranteed to run where model B runs and the other way around.

'`identical`'
: If model A is identical to model B, model A is guaranteed to run where model B runs and the other way around.

'`superset`'
: If model A is a superset of model B, model B is guaranteed to run where model A runs. There are no guarantees about the other way.

'`subset`'
: If model A is a subset of model B, model A is guaranteed to run where model B runs. There are no guarantees about the other way.

An enumeration of CPU model comparation results. The result is usually calculated using e.g. CPU features or CPU generations.

Since: 2.8.0

`CpuModelCompareInfo` *{ 'result':* *`CpuModelCompareResult`*, *'responsible-properties':* [*`'str'`*] *}* [Struct]

result
: The result of the compare operation.

responsible-properties
: is a list of QOM property names that led to both CPUs not being detected as identical. For identical models, this list is empty. If a QOM property is read-only, that means there's no known way to make the CPU models identical. If the special property name "type" is included, the models are by definition not identical and cannot be made identical.

The result of a CPU model comparison.

Since: 2.8.0

`CpuModelCompareInfo query-cpu-model-comparison` (*'modela':* *`CpuModelInfo`*, *'modelb':* *`CpuModelInfo`*) [Command]

Compares two CPU models, returning how they compare in a specific configuration. The results indicates how both models compare regarding runnability. This result can be used by tooling to make decisions if a certain CPU model will run in a certain configuration or if a compatible CPU model has to be created by baselining.

Usually, a CPU model is compared against the maximum possible CPU model of a certain configuration (e.g. the "host" model for KVM). If that CPU model is identical or a subset, it will run in that configuration.

The result returned by this command may be affected by:

- QEMU version: CPU models may look different depending on the QEMU version. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine-type: CPU model may look different depending on the machine-type. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine options (including accelerator): in some architectures, CPU models may look different depending on machine and accelerator options. (Except for CPU models reported as "static" in query-cpu-definitions.)
- "-cpu" arguments and global properties: arguments to the -cpu option and global properties may affect expansion of CPU models. Using query-cpu-model-expansion while using these is not advised.

Some architectures may not support comparing CPU models. s390x supports comparing CPU models.

Returns: a CpuModelBaselineInfo. Returns an error if comparing CPU models is not supported, if a model cannot be used, if a model contains an unknown cpu definition name, unknown properties or properties with wrong types.

Since: 2.8.0

`CpuModelBaselineInfo` *{ 'model': `CpuModelInfo` }* [Struct]

model the baselined CpuModelInfo.

The result of a CPU model baseline.

Since: 2.8.0

`CpuModelBaselineInfo query-cpu-model-baseline` (*'modela':* *`CpuModelInfo`*, *'modelb':* *`CpuModelInfo`*) [Command]

Baseline two CPU models, creating a compatible third model. The created model will always be a static, migration-safe CPU model (see "static" CPU model expansion for details).

This interface can be used by tooling to create a compatible CPU model out two CPU models. The created CPU model will be identical to or a subset of both CPU models when comparing them. Therefore, the created CPU model is guaranteed to run where the given CPU models run.

The result returned by this command may be affected by:

- QEMU version: CPU models may look different depending on the QEMU version. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine-type: CPU model may look different depending on the machine-type. (Except for CPU models reported as "static" in query-cpu-definitions.)
- machine options (including accelerator): in some architectures, CPU models may look different depending on machine and accelerator options. (Except for CPU models reported as "static" in query-cpu-definitions.)

– "-cpu" arguments and global properties: arguments to the -cpu option and global properties may affect expansion of CPU models. Using query-cpu-model-expansion while using these is not advised.

Some architectures may not support baselining CPU models. s390x supports baselining CPU models.

Returns: a CpuModelBaselineInfo. Returns an error if baselining CPU models is not supported, if a model cannot be used, if a model contains an unknown cpu definition name, unknown properties or properties with wrong types.

Since: 2.8.0

`AddfdInfo` *{ 'fdset-id':* `int`*, 'fd':* `int` *}* [Struct]

fdset-id The ID of the fd set that *fd* was added to.

fd The file descriptor that was received via SCM rights and added to the fd set.

Information about a file descriptor that was added to an fd set.

Since: 1.2.0

`AddfdInfo add-fd` (['*fdset-id':* `int`], ['*opaque':* `str`]) [Command]

*fdset-id** The ID of the fd set to add the file descriptor to.

*opaque** A free-form string that can be used to describe the fd.

Add a file descriptor, that was passed via SCM rights, to an fd set.

Returns: *AddfdInfo* on success

If file descriptor was not received, FdNotSupplied

If *fdset*-id is a negative value, InvalidParameterValue

Notes: The list of fd sets is shared by all monitor connections.

If *fdset*-id is not specified, a new fd set will be created.

Since: 1.2.0

Example:

```
-> { "execute": "add-fd", "arguments": { "fdset-id": 1 } }
<- { "return": { "fdset-id": 1, "fd": 3 } }
```

`remove-fd` (*'fdset-id':* `int`, ['*fd':* `int`]) [Command]

fdset-id The ID of the fd set that the file descriptor belongs to.

*fd** The file descriptor that is to be removed.

Remove a file descriptor from an fd set.

Returns: Nothing on success If *fdset*-id or *fd* is not found, FdNotFound

Notes: The list of fd sets is shared by all monitor connections.

If *fd* is not specified, all file descriptors in *fdset*-id will be removed.

Since: 1.2.0

Example:

```
-> { "execute": "remove-fd", "arguments": { "fdset-id": 1, "fd": 3 } }
<- { "return": {} }
```

`FdsetFdInfo` *{ 'fd':* `int`, [*'opaque':* `str`] *}* [Struct]

fd The file descriptor value.

*opaque** A free-form string that can be used to describe the fd.

Information about a file descriptor that belongs to an fd set.

Since: 1.2.0

`FdsetInfo` *{ 'fdset-id':* `int`, *'fds':* [`'FdsetFdInfo'`] *}* [Struct]

fdset-id The ID of the fd set.

fds A list of file descriptors that belong to this fd set.

Information about an fd set.

Since: 1.2.0

`['FdsetInfo'] query-fdsets ()` [Command]

Return information describing all fd sets.

Returns: A list of *FdsetInfo*

Note: The list of fd sets is shared by all monitor connections.

Since: 1.2.0

Example:

```
 -> { "execute": "query-fdsets" }
 <- { "return": [
        {
          "fds": [
            {
              "fd": 30,
              "opaque": "rdonly:/path/to/file"
            },
            {
              "fd": 24,
              "opaque": "rdwr:/path/to/file"
            }
          ],
          "fdset-id": 1
        },
        {
          "fds": [
            {
              "fd": 28
            },
            {
              "fd": 29
            }
          ],
          "fdset-id": 0
```

```
          }
        ]
      }
```

`TargetInfo` *{ 'arch': `str` }* [Struct]

arch the target architecture (eg "x86_64", "i386", etc)

Information describing the QEMU target.

Since: 1.2.0

`TargetInfo query-target` () [Command]

Return information about the target for this QEMU

Returns: TargetInfo

Since: 1.2.0

`QKeyCode` [Enum]

An enumeration of key name.

This is used by the send-key command.

Since: 1.3.0

'unmapped' and 'pause' since 2.0 'ro' and 'kp_comma' since 2.4 'kp_equals' and 'power' since 2.6

`KeyValue` [*'number': `int`, 'qcode': `QKeyCode`*] [Union]

Represents a keyboard key.

Since: 1.3.0

`send-key` (*'keys': ['`KeyValue`']*, [*'hold-time': `int`*]) [Command]

keys An array of *KeyValue* elements. All *KeyValues* in this array are simultaneously sent to the guest. A *KeyValue*.number value is sent directly to the guest, while *KeyValue*.qcode must be a valid

QKeyCode
value

*hold-time**
time to delay key up events, milliseconds. Defaults to 100

Send keys to guest.

Returns: Nothing on success If key is unknown or redundant, InvalidParameter

Since: 1.3.0

Example:

```
-> { "execute": "send-key",
     "arguments": { "keys": [ { "type": "qcode", "data": "ctrl" },
                              { "type": "qcode", "data": "alt" },
                              { "type": "qcode", "data": "delete" } ] } }
<- { "return": {} }
```

`screendump` (*'filename':* `str`) [Command]

filename the path of a new PPM file to store the image

Write a PPM of the VGA screen to a file.

Returns: Nothing on success

Since: 0.14.0

Example:

```
-> { "execute": "screendump",
     "arguments": { "filename": "/tmp/image" } }
<- { "return": {} }
```

`ChardevCommon` *{* [*'logfile':* `str`], [*'logappend':* `bool`] *}* [Struct]

*logfile** The name of a logfile to save output

*logappend**
true to append instead of truncate (default to false to truncate)

Configuration shared across all chardev backends

Since: 2.6

`ChardevFile` *{* [*'in':* `str`], *'out':* `str`, [*'append':* `bool`] *}* [Struct]

*in** The name of the input file

out The name of the output file

*append** Open the file in append mode (default false to truncate) (Since 2.6)

Configuration info for file chardevs.

Since: 1.4

`ChardevHostdev` *{ 'device':* `str` *}* [Struct]

device The name of the special file for the device, i.e. /dev/ttyS0 on Unix or COM1: on Windows

type What kind of device this is.

Configuration info for device and pipe chardevs.

Since: 1.4

`ChardevSocket` *{ 'addr':* `SocketAddress`, [*'tls-creds':* `str`], [*'server':* `bool`], [*'wait':* `bool`], [*'nodelay':* `bool`], [*'telnet':* `bool`], [*'reconnect':* `int`] *}* [Struct]

addr socket address to listen on (server=true) or connect to (server=false)

*tls-creds** the ID of the TLS credentials object (since 2.6)

*server** create server socket (default: true)

*wait** wait for incoming connection on server sockets (default: false).

*nodelay** set TCP_NODELAY socket option (default: false)

*telnet** enable telnet protocol on server sockets (default: false)

*reconnect**
: For a client socket, if a socket is disconnected, then attempt a reconnect after the given number of seconds. Setting this to zero disables this function. (default: 0) (Since: 2.2)

Configuration info for (stream) socket chardevs.

Since: 1.4

`ChardevUdp` *{ 'remote':* `SocketAddress`*, ['local':* `SocketAddress`*] }* [Struct]

remote
: remote address

*local**
: local address

Configuration info for datagram socket chardevs.

Since: 1.5

`ChardevMux` *{ 'chardev':* `str` *}* [Struct]

chardev
: name of the base chardev.

Configuration info for mux chardevs.

Since: 1.5

`ChardevStdio` *{ ['signal':* `bool`*] }* [Struct]

*signal**
: Allow signals (such as SIGINT triggered by ^C) be delivered to qemu. Default: true in -nographic mode, false otherwise.

Configuration info for stdio chardevs.

Since: 1.5

`ChardevSpiceChannel` *{ 'type':* `str` *}* [Struct]

type
: kind of channel (for example vdagent).

Configuration info for spice vm channel chardevs.

Since: 1.5

`ChardevSpicePort` *{ 'fqdn':* `str` *}* [Struct]

fqdn
: name of the channel (see docs/spice-port-fqdn.txt)

Configuration info for spice port chardevs.

Since: 1.5

`ChardevVC` *{ ['width':* `int`*], ['height':* `int`*], ['cols':* `int`*], ['rows':* `int`*] }* [Struct]

width
: console width, in pixels

height
: console height, in pixels

cols
: console width, in chars

rows
: console height, in chars

Configuration info for virtual console chardevs.

Since: 1.5

`ChardevRingbuf` *{ ['size':* `int`*] }* [Struct]

*size** ring buffer size, must be power of two, default is 65536

Configuration info for ring buffer chardevs.

Since: 1.5

`ChardevBackend` [*'file':* `ChardevFile`*, 'serial':* `ChardevHostdev`*, 'parallel':* `ChardevHostdev`*, 'pipe':* `ChardevHostdev`*, 'socket':* `ChardevSocket`*, 'udp':* `ChardevUdp`*, 'pty':* `ChardevCommon`*, 'null':* `ChardevCommon`*, 'mux':* `ChardevMux`*, 'msmouse':* `ChardevCommon`*, 'braille':* `ChardevCommon`*, 'testdev':* `ChardevCommon`*, 'stdio':* `ChardevStdio`*, 'console':* `ChardevCommon`*, 'spicevmc':* `ChardevSpiceChannel`*, 'spiceport':* `ChardevSpicePort`*, 'vc':* `ChardevVC`*, 'ringbuf':* `ChardevRingbuf`*, 'memory':* `ChardevRingbuf`] [Union]

Configuration info for the new chardev backend.

Since: 1.4 (testdev since 2.2)

`ChardevReturn` *{ ['pty':* `str`*] }* [Struct]

*pty** name of the slave pseudoterminal device, present if and only if a chardev of type 'pty' was created

Return info about the chardev backend just created.

Since: 1.4

`ChardevReturn` `chardev-add` (*'id':* `str`*, 'backend':* `ChardevBackend`) [Command]

id the chardev's ID, must be unique

backend backend type and parameters

Add a character device backend

Returns: ChardevReturn.

Since: 1.4

Example:

```
-> { "execute" : "chardev-add",
     "arguments" : { "id" : "foo",
                     "backend" : { "type" : "null", "data" : {} } } }
<- { "return": {} }

-> { "execute" : "chardev-add",
     "arguments" : { "id" : "bar",
                     "backend" : { "type" : "file",
                                   "data" : { "out" : "/tmp/bar.log" } } } }
<- { "return": {} }

-> { "execute" : "chardev-add",
     "arguments" : { "id" : "baz",
                     "backend" : { "type" : "pty", "data" : {} } } }
<- { "return": { "pty" : "/dev/pty/42" } }
```

`chardev-remove` (*'id': `str`*) [Command]

id the chardev's ID, must exist and not be in use

Remove a character device backend

Returns: Nothing on success

Since: 1.4

Example:

```
-> { "execute": "chardev-remove", "arguments": { "id" : "foo" } }
<- { "return": {} }
```

`TpmModel` [Enum]

'`tpm-tis`' TPM TIS model

An enumeration of TPM models

Since: 1.5

`['TpmModel'] query-tpm-models` () [Command]

Return a list of supported TPM models

Returns: a list of TpmModel

Since: 1.5

Example:

```
-> { "execute": "query-tpm-models" }
<- { "return": [ "tpm-tis" ] }
```

`TpmType` [Enum]

'`passthrough`'
TPM passthrough type

An enumeration of TPM types

Since: 1.5

`['TpmType'] query-tpm-types` () [Command]

Return a list of supported TPM types

Returns: a list of TpmType

Since: 1.5

Example:

```
-> { "execute": "query-tpm-types" }
<- { "return": [ "passthrough" ] }
```

`TPMPassthroughOptions` *{ ['path': `str`], ['cancel-path': `str`] }* [Struct]

*path** string describing the path used for accessing the TPM device

*cancel-path**
string showing the TPM's sysfs cancel file for cancellation of TPM commands while they are executing

Information about the TPM passthrough type

Since: 1.5

`TpmTypeOptions` [*'passthrough': `TPMPassthroughOptions`*] [Union]

passthrough
: The configuration options for the TPM passthrough type

A union referencing different TPM backend types' configuration options

Since: 1.5

`TpmInfo` *{ 'id': `str`, 'model': `TpmModel`, 'options': `TpmTypeOptions` }* [Struct]

id The Id of the TPM

model The TPM frontend model

options The TPM (backend) type configuration options

Information about the TPM

Since: 1.5

`['TPMInfo'] query-tpm` () [Command]

Return information about the TPM device

Returns: *TPMInfo* on success

Since: 1.5

Example:

```
-> { "execute": "query-tpm" }
<- { "return":
     [
       { "model": "tpm-tis",
         "options":
           { "type": "passthrough",
             "data":
               { "cancel-path": "/sys/class/misc/tpm0/device/cancel",
                 "path": "/dev/tpm0"
               }
           },
         "id": "tpm0"
       }
     ]
   }
```

`AcpiTableOptions` *{* [*'sig': `str`*], [*'rev': `uint8`*], [*'oem_id': `str`*], [*'oem_table_id': `str`*], [*'oem_rev': `uint32`*], [*'asl_compiler_id': `str`*], [*'asl_compiler_rev': `uint32`*], [*'file': `str`*], [*'data': `str`*] *}* [Struct]

*data** colon (:) separated list of pathnames to load and concatenate as table data. The resultant binary blob must not have an ACPI table header. At least one file is required. This field excludes

*sig** table signature / identifier (4 bytes)

*rev** table revision number (dependent on signature, 1 byte)

*oem_id** OEM identifier (6 bytes)

*oem_table_id**
: OEM table identifier (8 bytes)

*oem_rev** OEM-supplied revision number (4 bytes)

*asl_compiler_id**
: identifier of the utility that created the table (4 bytes)

*asl_compiler_rev**
: revision number of the utility that created the table (4 bytes)

*file** colon (:) separated list of pathnames to load and concatenate as table data. The resultant binary blob is expected to have an ACPI table header. At least one file is required. This field excludes *data*.

file.

Specify an ACPI table on the command line to load.

At most one of *file* and *data* can be specified. The list of files specified by any one of them is loaded and concatenated in order. If both are omitted,

Since: 1.5

`CommandLineParameterType` [Enum]

'`string`' accepts a character string

'`boolean`' accepts "on" or "off"

'`number`' accepts a number

'`size`' accepts a number followed by an optional suffix (K)ilo, (M)ega, (G)iga, (T)era

Possible types for an option parameter.

Since: 1.5

`CommandLineParameterInfo` *{ 'name': `str`, 'type': `CommandLineParameterType`, ['help': `str`], ['default': `str`] }* [Struct]

name parameter name

type parameter *CommandLineParameterType*

*help** human readable text string, not suitable for parsing.

*default** default value string (since 2.1)

Details about a single parameter of a command line option.

Since: 1.5

`CommandLineOptionInfo` *{ 'option': `str`, 'parameters': ['`CommandLineParameterInfo`'] }* [Struct]

option option name

parameters
: an array of *CommandLineParameterInfo*

Details about a command line option, including its list of parameter details

Since: 1.5

`['CommandLineOptionInfo'] query-command-line-options` [Command]
([*'option': str*])

*option** option name

option). Returns an error if the given *option* doesn't exist.

Query command line option schema.

Returns: list of *CommandLineOptionInfo* for all options (or for the given

Since: 1.5

Example:

```
 -> { "execute": "query-command-line-options",
      "arguments": { "option": "option-rom" } }
 <- { "return": [
         {
             "parameters": [
                 {
                     "name": "romfile",
                     "type": "string"
                 },
                 {
                     "name": "bootindex",
                     "type": "number"
                 }
             ],
             "option": "option-rom"
         }
       ]
     }
```

`X86CPURegister32` [Enum]

A X86 32-bit register

Since: 1.5

`X86CPUFeatureWordInfo` *{ 'cpuid-input-eax': `int`, ['cpuid-input-ecx': `int`], 'cpuid-register': `X86CPURegister32`, 'features': `int` }* [Struct]

cpuid-input-eax
Input EAX value for CPUID instruction for that feature word

*cpuid-input-ecx**
Input ECX value for CPUID instruction for that feature word

cpuid-register
Output register containing the feature bits

features value of output register, containing the feature bits

Information about a X86 CPU feature word

Since: 1.5

`DummyForceArrays` *{ 'unused':* [`'X86CPUFeatureWordInfo'`] *}* [Struct]

Not used by QMP; hack to let us use X86CPUFeatureWordInfoList internally

Since: 2.5

`RxState` [Enum]

'`normal`' filter assigned packets according to the mac-table

'`none`' don't receive any assigned packet

'`all`' receive all assigned packets

Packets receiving state

Since: 1.6

`RxFilterInfo` *{ 'name':* `str`*, 'promiscuous':* `bool`*, 'multicast':* `RxState`*, 'unicast':* `RxState`*, 'vlan':* `RxState`*, 'broadcast-allowed':* `bool`*, 'multicast-overflow':* `bool`*, 'unicast-overflow':* `bool`*, 'main-mac':* `str`*, 'vlan-table':* [`'int'`]*, 'unicast-table':* [`'str'`]*, 'multicast-table':* [`'str'`] *}* [Struct]

name net client name

promiscuous
whether promiscuous mode is enabled

multicast multicast receive state

unicast unicast receive state

vlan vlan receive state (Since 2.0)

broadcast-allowed
whether to receive broadcast

multicast-overflow
multicast table is overflowed or not

unicast-overflow
unicast table is overflowed or not

main-mac the main macaddr string

vlan-table a list of active vlan id

unicast-table
a list of unicast macaddr string

multicast-table
a list of multicast macaddr string

Rx-filter information for a NIC.

Since: 1.6

[`'RxFilterInfo'`] `query-rx-filter` ([*'name':* `str`]) [Command]

*name** net client name

Return rx-filter information for all NICs (or for the given NIC).

Returns: list of *RxFilterInfo* for all NICs (or for the given NIC). Returns an error if the given *name* doesn't exist, or given NIC doesn't support rx-filter querying, or given net client isn't a NIC.

Since: 1.6

Example:

```
-> { "execute": "query-rx-filter", "arguments": { "name": "vnet0" } }
<- { "return": [
        {
            "promiscuous": true,
            "name": "vnet0",
            "main-mac": "52:54:00:12:34:56",
            "unicast": "normal",
            "vlan": "normal",
            "vlan-table": [
                4,
                0
            ],
            "unicast-table": [
            ],
            "multicast": "normal",
            "multicast-overflow": false,
            "unicast-overflow": false,
            "multicast-table": [
                "01:00:5e:00:00:01",
                "33:33:00:00:00:01",
                "33:33:ff:12:34:56"
            ],
            "broadcast-allowed": false
        }
    ]
}
```

`InputButton` [Enum]

Button of a pointer input device (mouse, tablet).

Since: 2.0

`InputAxis` [Enum]

Position axis of a pointer input device (mouse, tablet).

Since: 2.0

`InputKeyEvent` *{ 'key':* `KeyValue`*, 'down':* `bool` *}* [Struct]

key Which key this event is for.

down True for key-down and false for key-up events.

Keyboard input event.

Since: 2.0

`InputBtnEvent` *{ 'button':* `InputButton`*, 'down':* `bool` *}* [Struct]

button Which button this event is for.

down True for key-down and false for key-up events.

Pointer button input event.

Since: 2.0

`InputMoveEvent` *{ 'axis':* `InputAxis`*, 'value':* `int` *}* [Struct]

axis Which axis is referenced by *value*.

value Pointer position. For absolute coordinates the valid range is 0 -> 0x7ffff

Pointer motion input event.

Since: 2.0

`InputEvent` [*'key':* `InputKeyEvent`*, 'btn':* `InputBtnEvent`*, 'rel':* `InputMoveEvent`*, 'abs':* `InputMoveEvent`] [Union]

key Input event of Keyboard

btn Input event of pointer buttons

rel Input event of relative pointer motion

abs Input event of absolute pointer motion

Input event union.

Since: 2.0

`input-send-event` ([*'device':* `str`], [*'head':* `int`], *'events':* [*'*`InputEvent`*'*]) [Command]

*device** display device to send event(s) to.

*head** head to send event(s) to, in case the display device supports multiple scanouts.

events List of InputEvent union.

Send input event(s) to guest.

Returns: Nothing on success.

The *device* and *head* parameters can be used to send the input event to specific input devices in case (a) multiple input devices of the same kind are added to the virtual machine and (b) you have configured input routing (see docs/multiseat.txt) for those input devices. The parameters work exactly like the device and head properties of input devices. If *device* is missing, only devices that have no input routing config are admissible. If *device* is specified, both input devices with and without input routing config are admissible, but devices with input routing config take precedence.

Note: The consoles are visible in the qom tree, under /backend/console[$index]. They have a device link and head property, so it is possible to map which console belongs to which device and display.

Since: 2.6

Example:

```
1. Press left mouse button.

-> { "execute": "input-send-event",
    "arguments": { "device": "video0",
                   "events": [ { "type": "btn",
                   "data" : { "down": true, "button": "left" } } ] } }
<- { "return": {} }

-> { "execute": "input-send-event",
    "arguments": { "device": "video0",
                   "events": [ { "type": "btn",
                   "data" : { "down": false, "button": "left" } } ] } }
<- { "return": {} }

2. Press ctrl-alt-del.

-> { "execute": "input-send-event",
     "arguments": { "events": [
        { "type": "key", "data" : { "down": true,
          "key": {"type": "qcode", "data": "ctrl" } } },
        { "type": "key", "data" : { "down": true,
          "key": {"type": "qcode", "data": "alt" } } },
        { "type": "key", "data" : { "down": true,
          "key": {"type": "qcode", "data": "delete" } } } ] } }
<- { "return": {} }

3. Move mouse pointer to absolute coordinates (20000, 400).

-> { "execute": "input-send-event" ,
  "arguments": { "events": [
                { "type": "abs", "data" : { "axis": "x", "value" : 20000 } },
                { "type": "abs", "data" : { "axis": "y", "value" : 400 } } ] } }
<- { "return": {} }
```

`NumaOptions` [*'node':* ***`NumaNodeOptions`***] [Union]

A discriminated record of NUMA options. (for OptsVisitor)

Since: 2.1

`NumaNodeOptions` *{* [*'nodeid':* ***`uint16`***], [*'cpus':* [***`'uint16'`***]], [*'mem':* ***`size`***], [*'memdev':* ***`str`***] *}* [Struct]

*nodeid** NUMA node ID (increase by 1 from 0 if omitted)

*cpus** VCPUs belonging to this node (assign VCPUS round-robin if omitted)

*mem** memory size of this node; mutually exclusive with *memdev*. Equally divide total memory among nodes if both *mem* and *memdev* are omitted.

*memdev** memory backend object. If specified for one node, it must be specified for all nodes.

Create a guest NUMA node. (for OptsVisitor)

Since: 2.1

`HostMemPolicy` [Enum]

'`default`' restore default policy, remove any nondefault policy

'`preferred`'
set the preferred host nodes for allocation

'`bind`' a strict policy that restricts memory allocation to the host nodes specified

'`interleave`'
memory allocations are interleaved across the set of host nodes specified

Host memory policy types

Since: 2.1

`Memdev` *{ 'size':* `size`*, 'merge':* `bool`*, 'dump':* `bool`*, 'prealloc':* `bool`*, 'host-nodes':* [`'uint16'`]*, 'policy':* `HostMemPolicy` *}* [Struct]

size memory backend size

merge enables or disables memory merge support

dump includes memory backend's memory in a core dump or not

prealloc enables or disables memory preallocation

host-nodes
host nodes for its memory policy

policy memory policy of memory backend

Information about memory backend

Since: 2.1

`['Memdev'] query-memdev ()` [Command]

Returns information for all memory backends.

Returns: a list of *Memdev*.

Since: 2.1

Example:

```
-> { "execute": "query-memdev" }
<- { "return": [
       {
         "size": 536870912,
         "merge": false,
         "dump": true,
         "prealloc": false,
         "host-nodes": [0, 1],
         "policy": "bind"
```

```
            },
            {
              "size": 536870912,
              "merge": false,
              "dump": true,
              "prealloc": true,
              "host-nodes": [2, 3],
              "policy": "preferred"
            }
          ]
        }
```

`PCDIMMDeviceInfo` *{ ['id': `str`], 'addr': `int`, 'size': `int`, 'slot': `int`, 'node': `int`, 'memdev': `str`, 'hotplugged': `bool`, 'hotpluggable': `bool` }* [Struct]

*id** device's ID

addr physical address, where device is mapped

size size of memory that the device provides

slot slot number at which device is plugged in

node NUMA node number where device is plugged in

memdev memory backend linked with device

hotplugged true if device was hotplugged

hotpluggable true if device if could be added/removed while machine is running

PCDIMMDevice state information

Since: 2.1

`MemoryDeviceInfo` [*'dimm': `PCDIMMDeviceInfo`*] [Union]

Union containing information about a memory device

Since: 2.1

`['MemoryDeviceInfo'] query-memory-devices` () [Command]

Lists available memory devices and their state

Since: 2.1

Example:

```
 -> { "execute": "query-memory-devices" }
 <- { "return": [ { "data":
                        { "addr": 5368709120,
                          "hotpluggable": true,
                          "hotplugged": true,
                          "id": "d1",
                          "memdev": "/objects/memX",
```

```
                                "node": 0,
                                "size": 1073741824,
                                "slot": 0},
                          "type": "dimm"
                        } ] }
```

`ACPISlotType` [Enum]

‘`DIMM`’ memory slot

‘`CPU`’ logical CPU slot (since 2.7)

`ACPIOSTInfo` *{ ['device': `str`], 'slot': `str`, 'slot-type': `ACPISlotType`, 'source': `int`, 'status': `int` }* [Struct]

*device** device ID associated with slot

slot slot ID, unique per slot of a given *slot*-type

slot-type type of the slot

source an integer containing the source event

status an integer containing the status code

OSPM Status Indication for a device For description of possible values of *source* and *status* fields see "_OST (OSPM Status Indication)" chapter of ACPI5.0 spec.

Since: 2.1

`['ACPIOSTInfo'] query-acpi-ospm-status ()` [Command]

Return a list of ACPIOSTInfo for devices that support status reporting via ACPI _OST method.

Since: 2.1

Example:

```
 -> { "execute": "query-acpi-ospm-status" }
 <- { "return": [ { "device": "d1", "slot": "0", "slot-type": "DIMM", "source": 1, "st
                  { "slot": "1", "slot-type": "DIMM", "source": 0, "status": 0},
                  { "slot": "2", "slot-type": "DIMM", "source": 0, "status": 0},
                  { "slot": "3", "slot-type": "DIMM", "source": 0, "status": 0}
    ]}
```

`WatchdogExpirationAction` [Enum]

‘`reset`’ system resets

‘`shutdown`’ system shutdown, note that it is similar to *powerdown*, which tries to set to system status and notify guest

‘`poweroff`’ system poweroff, the emulator program exits

‘`pause`’ system pauses, similar to *stop*

‘`debug`’ system enters debug state

‘`none`’ nothing is done

‘`inject-nmi`’
a non-maskable interrupt is injected into the first VCPU (all VCPUS on x86) (since 2.4)

An enumeration of the actions taken when the watchdog device's timer is expired

Since: 2.1

`IoOperationType` [Enum]

‘`read`’ read operation

‘`write`’ write operation

An enumeration of the I/O operation types

Since: 2.1

`GuestPanicAction` [Enum]

‘`pause`’ system pauses

An enumeration of the actions taken when guest OS panic is detected

Since: 2.1

`rtc-reset-reinjection` () [Command]

This command will reset the RTC interrupt reinjection backlog. Can be used if another mechanism to synchronize guest time is in effect, for example QEMU guest agent's guest-set-time command.

Since: 2.1

Example:

```
-> { "execute": "rtc-reset-reinjection" }
<- { "return": {} }
```

2.9 Rocker API

`Rocker` *{ 'name':* `str`*, 'id':* `uint64`*, 'ports':* `uint32` *}* [Struct]

name switch name

id switch ID

ports number of front-panel ports

Rocker switch information.

Since: 2.4

`RockerSwitch query-rocker` (*'name':* `str`) [Command]

Return rocker switch information.

Returns: *Rocker* information

Since: 2.4

Example:

```
-> { "execute": "query-rocker", "arguments": { "name": "sw1" } }
<- { "return": {"name": "sw1", "ports": 2, "id": 1327446905938}}
```

`RockerPortDuplex` [Enum]

‘`half`’ half duplex

‘`full`’ full duplex

An eumeration of port duplex states.

Since: 2.4

`RockerPortAutoneg` [Enum]

‘`off`’ autoneg is off

‘`on`’ autoneg is on

An eumeration of port autoneg states.

Since: 2.4

`RockerPort` *{ 'name':* `str`*, 'enabled':* `bool`*, 'link-up':* `bool`*, 'speed':* `uint32`*, 'duplex':* `RockerPortDuplex`*, 'autoneg':* `RockerPortAutoneg` *}* [Struct]

name port name

enabled port is enabled for I/O

link-up physical link is UP on port

speed port link speed in Mbps

duplex port link duplex

autoneg port link autoneg

Rocker switch port information.

Since: 2.4

`['RockerPort'] query-rocker-ports` (*'name':* `str`) [Command]

Return rocker switch port information.

Returns: a list of *RockerPort* information

Since: 2.4

Example:

```
-> { "execute": "query-rocker-ports", "arguments": { "name": "sw1" } }
<- { "return": [ {"duplex": "full", "enabled": true, "name": "sw1.1",
                  "autoneg": "off", "link-up": true, "speed": 10000},
                 {"duplex": "full", "enabled": true, "name": "sw1.2",
                  "autoneg": "off", "link-up": true, "speed": 10000}
   ]}
```

`RockerOfDpaFlowKey` *{ 'priority':* `uint32`*, 'tbl-id':* `uint32`*, ['in-pport':* `uint32`*], ['tunnel-id':* `uint32`*], ['vlan-id':* `uint16`*], ['eth-type':* `uint16`*], ['eth-src':* `str`*], ['eth-dst':* `str`*], ['ip-proto':* `uint8`*], ['ip-tos':* `uint8`*], ['ip-dst':* `str`*] }* [Struct]

priority key priority, 0 being lowest priority

tbl-id flow table ID

*in-pport** physical input port

*tunnel-id** tunnel ID

*vlan-id** VLAN ID

*eth-type** Ethernet header type

*eth-src** Ethernet header source MAC address

*eth-dst** Ethernet header destination MAC address

*ip-proto** IP Header protocol field

*ip-tos** IP header TOS field

*ip-dst** IP header destination address

Rocker switch OF-DPA flow key

Note: fields are marked #optional to indicate that they may or may not appear in the flow key depending if they're relevant to the flow key.

Since: 2.4

`RockerOfDpaFlowMask` *{* ['*in-pport*': `uint32`], ['*tunnel-id*': `uint32`], ['*vlan-id*': `uint16`], ['*eth-src*': `str`], ['*eth-dst*': `str`], ['*ip-proto*': `uint8`], ['*ip-tos*': `uint8`] *}* [Struct]

*in-pport** physical input port

*tunnel-id** tunnel ID

*vlan-id** VLAN ID

*eth-src** Ethernet header source MAC address

*eth-dst** Ethernet header destination MAC address

*ip-proto** IP Header protocol field

*ip-tos** IP header TOS field

Rocker switch OF-DPA flow mask

Note: fields are marked #optional to indicate that they may or may not appear in the flow mask depending if they're relevant to the flow mask.

Since: 2.4

`RockerOfDpaFlowAction` *{* ['*goto-tbl*': `uint32`], ['*group-id*': `uint32`], ['*tunnel-lport*': `uint32`], ['*vlan-id*': `uint16`], ['*new-vlan-id*': `uint16`], ['*out-pport*': `uint32`] *}* [Struct]

*goto-tbl** next table ID

*group-id** group ID

*tunnel-lport**
tunnel logical port ID

*vlan-id** VLAN ID

*new-vlan-id**
: new VLAN ID

*out-pport**
: physical output port

Rocker switch OF-DPA flow action

Note: fields are marked #optional to indicate that they may or may not appear in the flow action depending if they're relevant to the flow action.

Since: 2.4

`RockerOfDpaFlow` *{ 'cookie':* `uint64`*, 'hits':* `uint64`*, 'key':* `RockerOfDpaFlowKey`*, 'mask':* `RockerOfDpaFlowMask`*, 'action':* `RockerOfDpaFlowAction` *}* [Struct]

cookie flow unique cookie ID

hits count of matches (hits) on flow

key flow key

mask flow mask

action flow action

Rocker switch OF-DPA flow

Since: 2.4

`['RockerOfDpaFlow'] query-rocker-of-dpa-flows` (*'name':* `str`*,* [*'tbl-id':* `uint32`]) [Command]

name switch name

*tbl-id** flow table ID. If tbl-id is not specified, returns flow information for all tables.

Return rocker OF-DPA flow information.

Returns: rocker OF-DPA flow information

Since: 2.4

Example:

```
 -> { "execute": "query-rocker-of-dpa-flows",
      "arguments": { "name": "sw1" } }
 <- { "return": [ {"key": {"in-pport": 0, "priority": 1, "tbl-id": 0},
                   "hits": 138,
                   "cookie": 0,
                   "action": {"goto-tbl": 10},
                   "mask": {"in-pport": 4294901760}
                  },
                  {...more...},
    ]}
```

`RockerOfDpaGroup` *{ 'id':* `uint32`*, 'type':* `uint8`*, ['vlan-id':* `uint16`*],* [Struct]
['pport': `uint32`*], ['index':* `uint32`*], ['out-pport':* `uint32`*], ['group-id':* `uint32`*], ['set-vlan-id':* `uint16`*], ['pop-vlan':* `uint8`*], ['group-ids':* `['uint32']`*]], ['set-eth-src':* `str`*], ['set-eth-dst':* `str`*], ['ttl-check':* `uint8`*] }*

id group unique ID

type group type

*vlan-id** VLAN ID

*pport** physical port number

*index** group index, unique with group type

*out-pport**
output physical port number

*group-id** next group ID

*set-vlan-id**
VLAN ID to set

*pop-vlan** pop VLAN headr from packet

*group-ids**
list of next group IDs

*set-eth-src**
set source MAC address in Ethernet header

*set-eth-dst**
set destination MAC address in Ethernet header

*ttl-check** perform TTL check

Rocker switch OF-DPA group

Note: fields are marked #optional to indicate that they may or may not appear in the group depending if they're relevant to the group type.

Since: 2.4

`['RockerOfDpaGroup'] query-rocker-of-dpa-groups` (*'name':* `str`*, ['type':* `uint8`*]*) [Command]

name switch name

*type** group type. If type is not specified, returns group information for all group types.

Return rocker OF-DPA group information.

Returns: rocker OF-DPA group information

Since: 2.4

Example:

```
-> { "execute": "query-rocker-of-dpa-groups",
     "arguments": { "name": "sw1" } }
<- { "return": [ {"type": 0, "out-pport": 2,
```

```
                     "pport": 2, "vlan-id": 3841,
                     "pop-vlan": 1, "id": 251723778},
                    {"type": 0, "out-pport": 0,
                     "pport": 0, "vlan-id": 3841,
                     "pop-vlan": 1, "id": 251723776},
                    {"type": 0, "out-pport": 1,
                     "pport": 1, "vlan-id": 3840,
                     "pop-vlan": 1, "id": 251658241},
                    {"type": 0, "out-pport": 0,
                     "pport": 0, "vlan-id": 3840,
                     "pop-vlan": 1, "id": 251658240}
        ]}
```

`ReplayMode` [Enum]

'`none`' normal execution mode. Replay or record are not enabled.

'`record`' record mode. All non-deterministic data is written into the replay log.

'`play`' replay mode. Non-deterministic data required for system execution is read from the log.

Mode of the replay subsystem.

Since: 2.5

`xen-load-devices-state` (*'filename':* `str`) [Command]

filename the file to load the state of the devices from as binary data. See xen-save-devices-state.txt for a description of the binary format.

Load the state of all devices from file. The RAM and the block devices of the VM are not loaded by this command.

Since: 2.7

Example:

```
-> { "execute": "xen-load-devices-state",
     "arguments": { "filename": "/tmp/resume" } }
<- { "return": {} }
```

`GICCapability` *{ 'version':* `int`*, 'emulated':* `bool`*, 'kernel':* `bool` *}* [Struct]

version version of GIC to be described. Currently, only 2 and 3 are supported.

emulated whether current QEMU/hardware supports emulated GIC device in user space.

kernel whether current QEMU/hardware supports hardware accelerated GIC device in kernel.

The struct describes capability for a specific GIC (Generic Interrupt Controller) version. These bits are not only decided by QEMU/KVM software version, but also decided by the hardware that the program is running upon.

Since: 2.6

`['GICCapability'] query-gic-capabilities ()` [Command]

This command is ARM-only. It will return a list of GICCapability objects that describe its capability bits.

Returns: a list of GICCapability objects.

Since: 2.6

Example:

```
-> { "execute": "query-gic-capabilities" }
<- { "return": [{ "version": 2, "emulated": true, "kernel": false },
                { "version": 3, "emulated": false, "kernel": true } ] }
```

`CpuInstanceProperties` *{ ['node-id': `int`], ['socket-id': `int`], ['core-id': `int`], ['thread-id': `int`] }* [Struct]

*node-id** NUMA node ID the CPU belongs to

*socket-id** socket number within node/board the CPU belongs to

*core-id** core number within socket the CPU belongs to

*thread-id** thread number within core the CPU belongs to

List of properties to be used for hotplugging a CPU instance, it should be passed by management with device_add command when a CPU is being hotplugged.

Note: currently there are 4 properties that could be present but management should be prepared to pass through other properties with device_add command to allow for future interface extension. This also requires the filed names to be kept in sync with the properties passed to -device/device_add.

Since: 2.7

`HotpluggableCPU` *{ 'type': `str`, 'vcpus-count': `int`, 'props': `CpuInstanceProperties`, ['qom-path': `str`] }* [Struct]

type CPU object type for usage with device_add command

props list of properties to be used for hotplugging CPU

vcpus-count
number of logical VCPU threads *HotpluggableCPU* provides

*qom-path**
link to existing CPU object if CPU is present or omitted if CPU is not present.

Since: 2.7

`['HotpluggableCPU'] query-hotpluggable-cpus ()` [Command]

Returns: a list of HotpluggableCPU objects.

Since: 2.7

Example:

```
For pseries machine type started with -smp 2,cores=2,maxcpus=4 -cpu POWER8:
```

```
-> { "execute": "query-hotpluggable-cpus" }
<- {"return": [
     { "props": { "core": 8 }, "type": "POWER8-spapr-cpu-core",
       "vcpus-count": 1 },
     { "props": { "core": 0 }, "type": "POWER8-spapr-cpu-core",
       "vcpus-count": 1, "qom-path": "/machine/unattached/device[0]"}
   ]}'

For pc machine type started with -smp 1,maxcpus=2:

-> { "execute": "query-hotpluggable-cpus" }
<- {"return": [
     {
        "type": "qemu64-x86_64-cpu", "vcpus-count": 1,
        "props": {"core-id": 0, "socket-id": 1, "thread-id": 0}
     },
     {
        "qom-path": "/machine/unattached/device[0]",
        "type": "qemu64-x86_64-cpu", "vcpus-count": 1,
        "props": {"core-id": 0, "socket-id": 0, "thread-id": 0}
     }
   ]}
```

The HTML documentation of QEMU for more precise information.

Commands and Events Index

N

O

P

Q

R

S

T

V

W

X

Data Types Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X