QAPI Interface Design: Best Practices

Eric Blake
Red Hat
23 Oct 2013

Overview

- What is QAPI?
- Why is it Important?
- Recent Additions
- Rules of Thumb
- Future Work Needed

What is QAPI?

History

- Up through qemu 0.11, just the HMP text monitor
  - Output was ad hoc, and could be misparsed by management apps
  - No support for async events
- qemu 0.12 introduced QMP monitor and QAPI
  - First patches in Nov 2009, by Luiz
  - Based on JSON data structure (RFC 4627)
- Libvirt has used QMP as sole management interface since qemu 1.3, once there were enough commands to no longer need to parse 'qemu -help' output

Sample QMP session

```
$ qemu-kvm -M none -nodefaults -nographic -qmp stdio

{"QMP": {"version": {"qemu": {"micro": 1, "minor": 6,
"major": 1}, "package": ""}, "capabilities": []}}

{"execute": "qmp_capabilities"}

{"return": {}}

{"execute": "query-version"}

{"return": {"qemu": {"micro": 1, "minor": 6, "major":
1}, "package": ""}}

{"execute": "quit"}

{"return": {}}

{"timestamp": {"seconds": 1382123176, "microseconds":
50452}, "event": "SHUTDOWN"}
```

Other options for testing

- Use a second monitor

```
$ 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

$ telnet localhost:4444

{"execute":"..."}
```

- Use libvirt's virsh

```
$ virsh qemu-monitor-command --pretty guest
'{"execute":"..."}'
```

Sample QMP session, revisited

```
$ qemu-kvm -M none -nodefaults -nographic -qmp stdio

{"QMP": {"version": {"qemu": {"micro": 1, "minor": 6,
"major": 1}, "package": ""}, "capabilities": []}}

{"execute": "qmp_capabilities"}

{"return": {}}

{"execute": "query-version"}

{"return": {"qemu": {"micro": 1, "minor": 6, "major":
1}, "package": ""}}

{"execute": "quit"}

{"return": {}}

{"timestamp": {"seconds": 1382123176, "microseconds":
50452}, "event": "SHUTDOWN"}
```

Sample QAPI

- qapi-schema.json:

```
##
# @VersionInfo:
#
# A description of QEMU's version.
...
##
{ 'type': 'VersionInfo',
  'data': {'qemu': {'major': 'int', 'minor': 'int', 'micro': 'int'},
           'package': 'str'} }

##
# @query-version:
#
# Returns the current version of QEMU.
#
# Returns:  A @VersionInfo object describing the current version of QEMU.
#
# Since: 0.14.0
##
{ 'command': 'query-version', 'returns': 'VersionInfo' }
```

Sample Generated C Code

- qmp-commands.h (generated):

```
VersionInfo * qmp_query_version(Error **errp);
```

- qapi-types.h (generated):

```
struct VersionInfo
{
    struct
    {
        int64_t major;
        int64_t minor;
        int64_t micro;
    } qemu;
    char * package;
};
void qapi_free_VersionInfo(VersionInfo * obj);
```

- Allocate with glib (g_malloc0 or g_new0)
- Deep cleanup with automatic qapi_free_*
- Conversions between structs and QDict to provide flexible usage

Use in qemu C Code

- qmp.c:

```
VersionInfo *qmp_query_version(Error **err)
{
    VersionInfo *info = g_malloc0(sizeof(*info));
    const char *version = QEMU_VERSION;
    char *tmp;

    info->qemu.major = strtol(version, &tmp, 10);
    tmp++;
    info->qemu.minor = strtol(tmp, &tmp, 10);
    tmp++;
    info->qemu.micro = strtol(tmp, &tmp, 10);
    info->package = g_strdup(QEMU_PKGVERSION);

    return info;
}
```

More complex

- qapi-schema.json supports enums, optional parameters:

```
{ 'enum': 'DataFormat',
  'data': [ 'utf8', 'base64' ] }
{ 'command': 'ringbuf-write',
  'data': {'device': 'str', 'data': 'str',
           '*format': 'DataFormat'} }
```

- Generated code then has distinction between present and absent:
  - qapi-types.h

```
extern const char *DataFormat_lookup[];
typedef enum DataFormat
{
    DATA_FORMAT_UTF8 = 0,
    DATA_FORMAT_BASE64 = 1,
    DATA_FORMAT_MAX = 2,
} DataFormat;
```

  - qmp-commands.h

```
void qmp_ringbuf_write(const char * device, const char * data,
   bool has_format, DataFormat format, Error **errp);
```

Why is QAPI Important?

Ease of Maintenance

- Tedious work is automated
  - Header files automatically align with wire format
  - Visitor patterns allow safe allocation and cleanup
  - Lets the developer focus on just the unique aspects of the implementation, rather than repeating boilerplate
- Lends itself to testing
  - Compiler enforces that implementation matches header
  - Wire format is described in a machine-parseable manner, so it is easier to exercise an interface in tests
- QAPI was basis for qemu-ga interface

Interface Contract

- Older management's use of QMP must continue to work with newer qemu
  - Once released, only backward-compatible changes are allowed
  - Exceptions:
    - Any command or field beginning with “x-” is experimental
    - Any command beginning with __*RFQDN*_ (such as __com.redhat_) is a non-portable downstream addition
- Documentation includes first upstream version
  - But downstream can backport features to earlier versions, so check feature rather than version

Designed for Extension

- On input:
  - Any new fields must be optional, with default behavior when field is omitted remaining unchanged
  - JSON's name/value pairing in dictionaries ensures order of arguments will not cause problems
  - Enums by name, not number
- On output:
  - Management app should ignore fields it does not recognize
  - qemu must continue to output all non-optional fields ever used in past

Recent Additions in QAPI

Base Classes

- Several improvements driven by Kevin's recent 'blockdev-add' work, but useful elsewhere
- QAPI base classes:

```
{ 'type': 'BlockdevOptionsGenericFormat',
  'data': { 'file': 'BlockdevRef' } }
{ 'type': 'BlockdevOptionsGenericCOWFormat',
  'base': 'BlockdevOptionsGenericFormat',
  'data': { '*backing': 'BlockdevRef' } }
```

- Allows inline use of base class fields in QMP:

```
{ "file": "/some/place/my-image",
  "backing": "/some/place/my-backing-file" }
```

Discriminated Unions

- QAPI:

```
{ 'type': 'Qcow2Options',
  'data': { 'backing-file': 'str', 'lazy-refcounts': 'bool' } }
{ 'type': 'BlockdevCommonOptions',
  'data': { 'driver': 'str', 'readonly': 'bool' } }
{ 'union': 'BlockdevOptions',
  'base': 'BlockdevCommonOptions',
  'discriminator': 'driver',
  'data': { 'raw': 'RawOptions',
            'qcow2': 'Qcow2Options' } }
```

- Allows inline use of discriminator to determine remaining valid fields in QMP:

```
{ "driver": "qcow2",
  "readonly": false,
  "backing-file": "/some/place/my-image",
  "lazy-refcounts": true }
```

Anonymous Unions

- QAPI:

```
{ 'union': 'BlockRef',
  'discriminator': {},
  'data': { 'definition': 'BlockdevOptions',
            'reference': 'str' } }
```

- Allows type-based discrimination in QMP (between JSON primitive vs. object):

```
{ "file": "my_existing_block_device_id" }
{ "file": { "driver": "file",
            "readonly": false,
            "filename": "/tmp/mydisk.qcow2" } }
```

'blockdev-add' example

- Command line (added in qemu 1.5 and 1.6):

```
qemu ... -drive file.driver=qcow2,file=test.qcow2,\
backing.file.filename=test.raw
```

- QMP (added in qemu 1.7):

```
{ "execute": "blockdev-add",
  "arguments": { "options" :
      { "driver": "qcow2",
        "file": { "driver": "file",
                  "filename": "test.qcow2" },
        "backing": { "driver": "file",
                     "filename": "test.raw" } } } }
```

- Both map to same C code, and gives us the option of using JSON on command line in future

'blockdev-add' example, continued

- QAPI:

```
{ 'enum': 'BlockdevDiscardOptions',
  'data': [ 'ignore', 'unmap' ] }
{ 'enum': 'BlockdevAioOptions',
  'data': [ 'threads', 'native' ] }
{ 'type': 'BlockdevCacheOptions',
  'data': { '*writeback': 'bool',
            '*direct': 'bool',
            '*no-flush': 'bool' } }
{ 'type': 'BlockdevOptionsBase',
  'data': { 'driver': 'str',
            '*id': 'str',
            '*discard': 'BlockdevDiscardOptions',
            '*cache': 'BlockdevCacheOptions',
            '*aio': 'BlockdevAioOptions',
            '*rerror': 'BlockdevOnError',
            '*werror': 'BlockdevOnError',
            '*read-only': 'bool' } }
```

'blockdev-add' example, continued

```
{ 'type': 'BlockdevOptionsFile',
  'data': { 'filename': 'str' } }
{ 'type': 'BlockdevOptionsGenericFormat',
  'data': { 'file': 'BlockdevRef' } }
{ 'type': 'BlockdevOptionsGenericCOWFormat',
  'base': 'BlockdevOptionsGenericFormat',
  'data': { '*backing': 'BlockdevRef' } }
{ 'type': 'BlockdevOptionsQcow2',
  'base': 'BlockdevOptionsGenericCOWFormat',
  'data': { '*lazy-refcounts': 'bool',
            '*pass-discard-request': 'bool',
            '*pass-discard-snapshot': 'bool',
            '*pass-discard-other': 'bool' } }
```

'blockdev-add' example, continued

```
{ 'union': 'BlockdevOptions',
  'base': 'BlockdevOptionsBase',
  'discriminator': 'driver',
  'data': {
      'file':       'BlockdevOptionsFile',
      'qcow2':      'BlockdevOptionsQcow2',
...
  } }
{ 'union': 'BlockdevRef',
  'discriminator': {},
  'data': { 'definition': 'BlockdevOptions',
            'reference': 'str' } }
{ 'command': 'blockdev-add', 'data': { 'options':
'BlockdevOptions' } }
```

Design Rules of Thumb

Don't Force a Reparse

- If the receiver must parse a string into individual components, the JSON was too high-level
- Compare Max's first attempt...

https://lists.gnu.org/archive/html/qemu-devel/2013-09/msg00976.html

```
# @info-string: #optional string supplying additional format-specific
# information (since 1.7)
{ 'type': 'ImageInfo',
  'data': {'filename': 'str', 'format': 'str', '*dirty-flag': 'bool',
           '*actual-size': 'int', 'virtual-size': 'int',
           '*cluster-size': 'int', '*encrypted': 'bool',
           '*backing-filename': 'str', '*full-backing-filename': 'str',
           '*backing-filename-format': 'str', '*snapshots': ['SnapshotInfo'],
           '*backing-image': 'ImageInfo', '*info-string': 'str' } }
```

Don't Force a Reparse, continued

- ...with his final implementation (commit f2bb8a8a):

  https://lists.gnu.org/archive/html/qemu-devel/2013-10/msg01067.html

```
{ 'type': 'ImageInfoSpecificQCow2',
  'data': {
      'compat': 'str',
      '*lazy-refcounts': 'bool'
  } }
{ 'union': 'ImageInfoSpecific',
  'data': {
     'qcow2': 'ImageInfoSpecificQCow2'
  } }
{ 'type': 'ImageInfo',
  'data': {'filename': 'str', 'format': 'str', '*dirty-flag': 'bool',
           '*actual-size': 'int', 'virtual-size': 'int',
           '*cluster-size': 'int', '*encrypted': 'bool',
           '*backing-filename': 'str', '*full-backing-filename': 'str',
           '*backing-filename-format': 'str', '*snapshots': ['SnapshotInfo'],
           '*format-specific': 'ImageInfoSpecific' } }
```

Avoid open-coded finite sets of strings

- Use an enum
  - QMP wire representation is the same
  - But C code gets to use enum constants that map to string in an easier manner
- Example: Fam's conversion of block job string to an enum

  https://lists.gnu.org/archive/html/qemu-devel/2013-10/msg00833.html

```
{ 'enum': 'BlockJobType',
  'data': ['commit', 'stream', 'mirror', 'backup'] }
```

- Allows for `'type':'BlockJobType'` instead of `'type':'str'` when representing a block job event

Favor Consistent Naming

- MixedCase types, dashed-name commands and fields
- New commands should use '-' in QAPI (the corresponding C code will still use '_')
  - But when extending old commands, stick with the prevailing style
- Pending idea of making the parser more lenient:

  https://lists.gnu.org/archive/html/qemu-devel/2013-04/msg00681.html
  - Careful what gets flattened – command names are okay, but values passed on to guest are not

Try for Human Legibility

- Although QAPI's main focus is machine parsing, it helps to also be human readable
  - Use consistent 'verb' or 'noun-verb' commands
  - Avoid abbreviations
  - Use full power of QAPI to avoid overly nesting structs
- Compare Amos' first attempt at macvtap programming

  https://lists.gnu.org/archive/html/qemu-devel/2013-05/msg02174.html
- with his final implementation (commit b1be4280)

  https://lists.gnu.org/archive/html/qemu-devel/2013-06/msg02223.html

Try for Human Legibility, continued

- First try:

```
{ 'type': 'MacTableInfo', 'data': {
    'name':              'str',    '*promisc':         'bool',
    '*allmulti':         'bool',   '*alluni':          'bool',
    '*nomulti':          'bool',   '*nouni':           'bool',
    '*nobcast':          'bool',   '*multi-overflow': 'bool',
    '*uni-overflow':     'bool',   '*unicast':         ['str'],
    '*multicast':        ['str'] }}
```

- Final version:

```
{ 'enum': 'RxState', 'data': [ 'normal', 'none', 'all' ] }
{ 'type': 'RxFilterInfo', 'data': {
  'name':               'str',    'promiscuous':        'bool',
  'multicast':          'RxState','unicast':            'RxState',
  'broadcast-allowed':'bool',     'multicast-overflow':'bool',
  'unicast-overflow': 'bool',     'main-mac':           'str',
  'vlan-table':       ['int'],    'unicast-table':      ['str'],
  'multicast-table':  ['str'] }}
```

Documentation is Essential

- Mention which release a command was introduced, and which release an optional field was added
- Document each field of a struct, each value of enum
- Don't forget qmp-commands.hx (although that may hopefully go away if we automate more from qapi-schema.json)
- Post ideas to the list, and respond to feedback

Documentation Example

```
##
# @ImageInfo:
...
# @snapshots: #optional list of VM snapshots
# @backing-image: #optional info of the backing image (since 1.6)
# @format-specific: #optional structure supplying additional format-specific
# information (since 1.7)
#
# Since: 1.3
##
{ 'type': 'ImageInfo',
  'data': {'filename': 'str', 'format': 'str', '*dirty-flag': 'bool',
           '*actual-size': 'int', 'virtual-size': 'int',
           '*cluster-size': 'int', '*encrypted': 'bool',
           '*backing-filename': 'str', '*full-backing-filename': 'str',
           '*backing-filename-format': 'str', '*snapshots': ['SnapshotInfo'],
           *backing-image': 'ImageInfo',
           *format-specific': 'ImageInfoSpecific' } }
```

Try to be Discoverable

- When adding a new feature by extending an existing enum or struct, can newer management safely determine whether the feature is usable?
  - Detection by failing on unrecognized argument is a last resort, but generally has poor error message quality
- No write-only interfaces: if something can be set, it should also be something that can be queried
- Introspection is a heavy hammer – is there something lighter-weight that does the same job?

Try to be Discoverable, example

- Migration command has added several features
- Introspecting the enum would work...

```
{ 'enum': 'MigrationCapability',
  'data': ['xbzrle', 'x-rdma-pin-all',
           'auto-converge', 'zero-blocks'] }
```

- But even nicer is a query command

```
{ 'type': 'MigrationCapabilityStatus',
  'data': { 'capability' : 'MigrationCapability',
            'state' : 'bool' } }
{ 'command': 'query-migrate-capabilities',
  'returns': ['MigrationCapabilityStatus']}
```

Good Interface goes beyond QMP

- Many of these design principles also apply to C code
- Command line design should also be structured
  - 'query-command-line-options' QMP command is useful
- Publish designs early, listen to feedback
- Contribute review time to share the load

Future Additions

Move Events into formal QAPI

- Events currently tracked ad hoc in docs/qmp/qmp-events.txt
- Wenchao's patches this week creates enum:
  https://lists.gnu.org/archive/html/qemu-devel/2013-10/msg02733.html
- Kevin's suggestion for using union type:
  https://lists.gnu.org/archive/html/qemu-devel/2013-09/msg02164.html
- Will guarantee which elements are part of each event

Introspection

- Amos' initial work in 1.6 time frame, stalled at the time

  https://lists.gnu.org/archive/html/qemu-devel/2013-07/msg02494.html
- Now that Kevin's QAPI improvements allow better type expressions, this should be revived
- Basically a way to query all commands, types, enums, and events from the QAPI

Other Cleanups

- Maintaining qapi-schema.json and qmp-commands.hx in parallel is prone to get out of sync
  - Would be nice to automate documentation of examples from one file
- HMP should serve as an ideal client of QMP
  - Several commands still exist as HMP only, such as 'savevm'
  - Various patches under review, but not done yet
  - Rewrite HMP to call into QMP layer
- Possibly expose qemu-ga directly through QMP

Patches Welcome!

- Check the to-do list:
  - https://lists.gnu.org/archive/html/qemu-devel/2013-09/msg01816.html
- Patch submission hints:
  - http://wiki.qemu.org/Contribute/SubmitAPatch
- Thanks to the QMP/QAPI maintainers:
  - Luiz C.
  - Michael R.
  - Markus A.

Thank You