A Hacker's Guide to the Quartermaster

Oblong Industries, Inc.

Last updated: 25 January 2011

Table of Contents

1 Overview

The Quartermaster is the name given to the motley gang of processes that serve to capture, encode, and transcode video and audio sources, both automatically and in response to user controls.

Notably, the Quartermaster is used to capture and encode DVI input from Westar HRED PCI cards. When a user plugs in a DVI cable, the *provisioner*—the part of the Quartermaster that can turn media capture on or off—forks off a process (a *viddle*) to actually do the DVI capture. When the DVI source goes away, the viddle dies, and the provisioner reaps it.

One provisioner can supervise a number of *provisions*: possible video and audio streams. Typically this set is fixed when the provisioner is started. Some provisions can be unavailable, for example a DVI capture source whose cable isn't plugged in. The provisioner will notify a pool as to any changes in source availability.

The name "Quartermaster" applies just as much to the pools used by the various processes as to the processes themselves. There is one main pool that binds all processes, the *coordination* pool. Typically this pool is named '`qm`'. Each viddle deposits encoded video and audio into its own pool. The provisioner deposits its state periodically into the coordination pools, and also listens on it for requests from users to provision or deprovision resources. Viddles also deposit their state on the coordination pool.

Besides the coordination pool, there are some other minor pools: a configuration pool used by the provisioner to save its state between runs, and viddle drome pools (*inboxes*), used to communicate directly with particular viddles.

2 The Provisioner

`qm-provisioner` is a daemon that watches a coordination pool for requests to "provision viddles", which means to provide video streams to pools.

If yovo detected the presence of Westar development headers at build-time, `qm-provisioner` also monitors all Westar HRED cards for DVI input, automatically provisioning viddles as cables are plugged and unplugged.

2.1 Invoking the Provisioner

The provisioner may be invoked as `qm-provisioner` *`[option...] coord-pool [conf-file]`*.

coord-pool is the name of the coordination pool, typically '`qm`', and *conf-file* is a path to a configuration file, described in the following section.

2.1.1 Configuration Format

The provisioner configuration is expressed as a slaw map with three keys: `provisions`, describing the provisions on offer; `meta`, parameterizing how the provisions are offered; and `provisioner-params`, parameterizing the provisioner itself.

On disk, the configuration is in the textual (Yaml) slaw representation. A basic configuration file might look like this:

```
provisioner-params:
  name: test-provisioner
provisions:
  test:
    resource:
      type: pipeline
      pipeline: videotestsrc is-live=true
meta:
  thumbnail-bounding-box: [ 320, 320 ]
```

In this example configuration, the provisioner is given the name '`test-provisioner`'. One provision is specified, with a low-level GStreamer test pattern specified as the resource. All provisions provided by this provisioner will periodically take thumbnail snapshots of the stream they are encoding, and deposit those thumnails in a pool. The `meta` block specifies the bounding box for these thumbnails, namely, 320 by 320 pixels.

The following excessively nested table describes the format of these maps in more detail.

`provisions` (`map`)
: The set of provisions that will be on offer from this provisioner. Each provision has a corresponding entry in the map, keyed on the name of the provision, as a string.

  provision-name (`map`)
  : Here by *provision-name*, we mean the specific name of the provision. There may be many provisions, keyed under different names. For example, in the example configuration above, the one provision was named `test`. Provisioner names should be strings.

The value corresponding to the provisioner name is a map, of which three keys are understood:

`resource` (`map`)
: A map specifying the source of this provision. This map should be suitable to passing to `Fontana::Open`; that is to say, it should have a `type` entry, which may be `file`, `pool`, or `pipeline`, indicating whether the resource is coming from a file, from a pool, or from a low-level GStreamer pipeline description.

  The form of the rest of the map depends on the resource type. For example, a resource with `type` of `pipeline` expects a further `pipeline` entry in the map, containing a GStreamer pipeline description, as a string.

`meta` (`map`, optional)
: A meta entry; see below.

`properties` (`map`, optional)
: A map, enumerating the set of properties that can be changed at run-time. Additionally this map specifies to the viddle, when present, exactly how to implement the properties.

  property-name (`cons`*)*
  : The name of the property. The value is a slaw cons consisting of the name of a GStreamer element, and a property name within that element.

  For example, the pair [‘`vidsource`’, ‘`chrominance-weight`’] could be indexed under the `chrominance-weight-0` property – indicating that there should be an element named ‘`vidsource`’ in the resulting GStreamer pipeline, which has a GStreamer property `chrominance-weight`, which we will expose as ‘`chrominance-weight-0`’.

  In Yaml, this would be written like:

```
properties:
  chrominance-weight-0: !cons
    vidsource: chrominance-weight
```

  Note that `properties` is really the schema of what properties are available, and does not specify their values. You can set their values through either the provision’s or the default `meta` block, described below, and get values using the `get-property` viddle protein interface described later on.

`meta` (`map`, optional)
: A default set of "meta-parameters": parameters that describe *how* to provision a viddle, not actually describing the resource being provisioned.

  For example, there is a meta-parameter, `video-encoder`, which specifies a pipeline used to encode a provisioned viddle.

  For any given provision, there are three sources of meta-parameters. The first source, and the most specific, are meta-parameters that are specific to a given provision. These are those that are stored under a provision's own `meta` key.

  The next source of meta-parameterization is this `meta` block, at the top level in the provisioner configuration, constituting global default values. If a provision doesn't specify a value for a meta-parameter, it is taken from the global defaults.

  Finally if nothing specifies the value of a parameter, there are defaults within the provisioner and viddle code. For example, if no `thumbnail-bounding-box` meta-parameter is specified, the default is 320 by 240 pixels.

  The set of meta parameters can expand over time. Here is a list of currently known meta-parameters:

  `pool-byte-size` (`s64`)
  : If this value is set, when the provisioner creates viddle pools, they will be made big enough to hold this many bytes. `pool-byte-size` is not set by default.

  `pool-time-size` (`s64`)
  : When making a new pool in which to deposit video, and the `pool-byte-size` meta-parameter is not set, the provisioner attempts to create a pool big enough to hold a number of seconds of video. For example, a value of 900 indicates 900 seconds.

  `pool-index-capacity` (`s64`)
  : New pools created by the Quartermaster will have space for this many entries in their indices. If this value is unset, as is the case by default, a new pool will not have an index.

  `video-passthrough-caps` (`str`)
  : If set, indicates a video format that should not be re-encoded by a viddle. The string should be a valid "caps" string that GStreamer can understand, like `image/x-wj2`. The default is unset, indicating that all streams will be decoded, if needed, then encoded.

`video-encoder` (`str`)
: A GStreamer pipeline string, used for encoding video streams in a viddle. The default is to encode to H.264 with `x264enc`.

`video-bounding-box` (`list`)
: A two-element list, [*width*, *height*], indicating the maximum size of the rendered video. If needed, the video will be scaled down to fit in the bounding box, while preserving the aspect ratio. The default is no bounding box, indicating no limits on the size of the output video. This option is mostly useful when running the provisioner in transcoder mode (see below).

`audio-passthrough-caps` (`str`)
: Like `video-passthrough-caps`, but for audio streams. The default is unset.

`audio-encoder` (`str`)
: Like `video-encoder`, but for audio. The default is to encode to Vorbis.

`thumbnail-enabled` (`bool`)
: Whether to emit thumbnail images periodically (currently every four seconds). Defaults to `true`.

`thumbnail-bounding-box` (`list`)
: A two-element list specifying a bounding box for thumbnail sizes. Thumbnails will maintain the aspect ratio of the original image, though they will have square pixels. The default is 320 pixels wide by 200 tall, which is written as `[320, 200]` in Yaml form.

`thumbnail-encoder` (`str`)
: A GStreamer encoding pipeline for the thumbnails. Currently the default is to encode to PNG images.

`thumbnail-passthrough-decoder` (`str`)
: For passthrough video streams (see `video-passthrough-caps` above), a GStreamer pipeline used to decode the passthrough stream. The default is to use the generic `decodebin2`, but you may want to specify a value here if you are using a hardware codec like WJ2. An example value for WJ2 would be '`wj2softdec decoder-pipeline=kakadudec`'.

`compression-fraction` (`f64`)
: A target compression ratio, expressed as a decimal value between 0.0 and 1.0. The default is `0.05`.

`auto-provision` (`bool`)
: Whether to automatically provision a viddle, if is is available; defaults to `true`. See the discussion of avail-

ability below in the description of the `provisions` protein.

`allowed-meta-tweak-keys` (`list`)
: It makes sense to allow a user to tweak a number of these keys at run-time, over the pools interface. `allowed-meta-tweak-keys` allows the administrator to specify which keys are "safe" for tweaking. By default, the provisioner allows all keys listed as provision properties (e.g. `chrominance-weight-0` above) to be tweaked, as well as the `auto-provision` parameter.

  For more information, see the documentation below on the `tweak-meta` protein.

As hinted above, `meta` may also contain values for properties. Recall above that we specified a number of run-time parameters, in the `properties` block of a provision. `meta` may contain set values for those parameters: either default values, if the entry is in the default `meta` block, or provision-specific values, if the entry is in the `meta` block of a particular provision.

So, to continue the previous `chrominance-weight-0` example, we might specify a default chrominance weight of 0.3 by adding a '`chrominance-weight-0: 0.3`' entry to the default meta block.

`provisioner-params` (`map`, optional)
: Parameters for the provisioner itself. The current set of recognized provisioner parameters is as follows:

  `name` (`str`, optional)
  : A name for this provisioner. Defaults to the generated provisioner ID.

  `viddle-pool-stem` (`str`, optional)
  : A string to add to the pool name for a rendered viddle.

    By default, viddles are rendered to local pools having the same name as their resource; so the above '`test`' provision would render to a pool name '`test`', by default. Specifying a '`viddle-pool-stem`' of '`foo-`' would change that to render to `foo-test`.

    Perhaps more usefully, one may specify a stem like '`tcp://otherhost/`' to cause viddles to be written to another machine instead of being written locally.

  `auto-pattern` (`str`, optional)
  : A regular expression to apply to provision names, used to limit auto-provisioning to only some subset of provisions. Only provisions whose names that match this pattern will be auto-provisioned. No `auto-pattern` is set by default.

`dvi-provision` (`bool`, optional)
: Whether to capture DVI input from Westar HRED cards, if the provisioner was compiled with support for HRED cards. By default this is enabled, if your provisioner has HRED support.

`dvi-audio-capture` (`str`, optional)
: How to capture audio corresponding to HRED DVI capture provisions, if supported, expressed as a GStreamer pipeline. By default, DVI capture streams will not have corresponding audio streams.

  The current best option that we have here is to use PulseAudio, if it is available, with the pipeline '`pulsesrc name=audsource`'. This way a technician can figure out how to map audio inputs to provisions, as the provision names will be indicated in the PulseAudio mixer application.

2.1.2 Persistent Run-Time Changes

As hinted by the `allowed-meta-tweak-keys` meta-parameter, users have some ability to change settings at run-time. The provisioner stores these changes persistently into a pool so it can read them back in the next time the provisioner runs.

Every time a provisioner starts, or its properties change, it will write out a protein to the `qm-conf/`*`provisioner-name`* pool. This pool will be created as necessary.

The protein will have the following format:

`state` [Protein]
: A serialization of the state of a provisioner. Since this protein is deposited to a pool with a specific name corresponding to the specific provisioner, the ingests are simply those keys and values which may appear in a provisioner configuration file; namely:

  `provisions` (`map`)
  `meta` (`map`, optional)
  `provisioner-params` (`map`, optional)

Just in case something happens to the pool, a backup file is also written to disk any time the provisioner configuration changes. This file is written to the current directory, with the name *`provisioner-name`*`.conf.backup`. This backup file is in the same format as the original configuration file, so it may be passed directly to `qm-provisioner` when it is invoked.

2.1.3 Command-Line Options

Generally speaking, a provisioner is mainly configured through its configuration file. However, some things are more convenient to set on the command line, so we offer a number of command-line options as well. All of these options override the settings given in the configuration file.

‘`--name` *`name`*’
: Sets the name of the provisioner: the `name` entry in the `provisioner-params` map.

  This option is particularly useful in combination with the provisioner's ability to recover its state from a pool. If the provisioner is invoked without a configuration file, a configuration will be searched for in the `qm-conf/`*`name`* pool, and in the *`name`*`.conf.backup` file, in that order. The provisioner only signals an error if no configuration could be found in any of these sources (command-line configuration file, pool, or backup file).

‘`--debug-viddles`’
: Cause the provisioner to write out the slawx that it sends to viddles to disk, for debugging purposes. See the documentation below on invoking `qm-viddle` for more information on how to use these files.

‘`--auto`’
: Set the default `auto-provision` meta-parameter to `true`, or via ‘`--noauto`’, set it to `false`.

‘`--auto-pattern` *`pat`*’
: Set the `auto-pattern` meta-parameter.

‘`--restore-state`’
: By default, the provisioner can recover its state from a pool or backup file. If you don't want this, invoke `qm-provisioner` with ‘`--norestore-state`’.

‘`--conf-pool` *`pool`*’
: Specify a pool to use for configuration updates. The default is `qm-conf/`*`name`*.

‘`--transcode` *`coord`*’
: Provide transcoded provisions from a coord pool. Transcoding isn't really well documented, but the basic idea is that the provisioner will watch a coordination pool, and re-provision viddles that it sees on that pool, prepending `trans-` to their provisioner names.

‘`--dvi`’
: Watch Westar HRED cards for DVI input, provisioning any available video. Only available if Westar support was compiled in. Disable via ‘`--nodvi`’.

‘`--help`’
: Show a brief help message.

2.2 Provisioner Proteins

The provisioner periodically outputs a representation of its state on the coordination pool, so that remote clients can robustly maintain an idea of what provisions are available.

The heartbeat is deposited once every 4 seconds. Clients should expire stale resources after 8 seconds.

It's possible, though not advisable, to have multiple provisioners with the same name operating on the same coordination pool. Since we often need to be able to distinguish between the various instances, provisioners are assigned a pseudo-random identifier at startup, the *provisioner ID*. When needed, proteins carry this identifier in them, to identify the originating provisioner.

The format of the hearbeat protein is as follows:

`provisioner` *provisioner-id* [Protein]

A representation of the state of a provisioner. Ingests are as follows:

`provisions` (`map`, optional)

A map of all of the provisions on offer by this provisioner, keyed by provisioner name.

provision-name (`map`)

A provision map, containing a `resource` entry, and optionally `meta` and `properties` as well.

By default, all provisions are "available", meaning that all of them are considered to be in a state in which they can be provisioned.

This obviously isn't the case however for all viddles; for example, you might have DVI capture provisions, but they are only available when there is a cable connected. So the provisioner keeps another bit at run-time, `available`, indicating the availability of a stream is actually available, or not. This extra run-time information is added to the `meta` map.

`meta` (`map`, optional)

In addition to all the keys that were in the `meta` map for this provision, as statically configured, the provisioner may add the following keys:

`available` (`bool`, optional)

A flag indicating the availability of this provision. If not present, a provision is considered available.

`width` (`s32`, optional)

The width in pixels of this available video stream, if known. May not be present.

`height` (`s32`, optional)

The height in pixels of this available video stream, if known. May not be present.

DVI capture resources typically have a number of properties available on them. They might have initial properties set on them:

`bitrate` (`u64`, optional)

The target bitrate of this stream, in kilobits per second. For example, 1000. Typically you only set either `compression-fraction` or `bitrate`, and not both.

`compression-fraction` (`f64`, optional)

The target ratio of compressed to uncompressed image size for this stream, for example 0.05.

`chrominance-weight` (`f64`, optional)
: The relative bitrate of the chrominance component in the stream, relative to luminance. 1.0 assigns equal weight. The default is 0.3, which assigns fewer bits to chrominance.

`frame-skip` (`s64`, optional)
: The number of frames to skip relative to the original DVI refresh rate. The default of 1 causes capture at about 30 Hz.

`meta` (`map`, optional)
: The default set of meta-parameters for this provisioner.

`qm-provisioner` also responds to proteins deposited on the coordination pool, as follows:

`ping` [Protein]

Causes the provisioner to heartbeat immediately, instead of waiting for the next timeout. Useful for clients to get an idea of the state without waiting for the next heartbeat.

No ingests.

`provision!` *provisioner-id* [Protein]

A request for the provisioner with the specific *provisioner-id* to, well, provide the provision. If no viddle is actively servicing the provision, a new one will be forked off. See the `qm-viddle` documentation below, for more information.

Ingests are as follows:

`provision` (`str`)
: The name of the provision to provide.

`pool` (`str`, optional)
: The name of the pool in which to render video. By default the output pool has the same name as the provision itself, but see the above discussion of `viddle-pool-stem` for more.

`meta` (`map`, optional)
: A map of meta-parameters to merge with the meta-parameters statically specified for this provision, and the defaults.

`persistently` (`bool`, optional)
: If set to `true`, persistently record this change by setting `auto-provision: true` in this provision's meta-parameters.

If the pool does not exist, it is created according to the various size and index meta-parameters. In any case the video will deposit with a pseudo-unique stream identifier, or *mtag*. This allows sequential capture runs to share the same pool and still be distinguishable.

When the viddle terminates, for any reason, a "provision-end" protein is deposited on the coordination pool, described below.

If a viddle was actually spawned off, a standard retort (see below) will be deposited on the coordination pool whose value is a map with keys `mtag` and `pool-name`.

The Quartermaster's protein interface generally eschews procedure-call semantics, where possible, but it does admit to it where appropriate. To indicate that a particular protein is a response to another, the Quartermaster uses *retort proteins*.

`retort` [Protein]

A reply to another protein. Replies go out to the same pool that requests come in on.

`value` The reply value. The type depends on the particular protein being responded to.

`from-index` (`s64`)
The index of the protein in this pool that caused this retort.

`deprovision!` *provisioner-id* [Protein]

Deprovisions a viddle with the given mtag, if any. A `deprovision!` protein tells the provisioner to stop a viddle.

`mtag` (`str`)
The mtag of the viddle to stop. The viddle is named by mtag in an attempt to pseudo-uniquely address particular viddles, even ones that might have the same name.

`persistently` (`bool`, optional)
If set to `true`, persistently record this change by setting `auto-provision: false` in this provision's meta-parameters.

`provision-end` [Protein]

Deposited by the provisioner to indicate that a viddle has died. Note that "died" simply means that the process went away; it may have been successful, or it may have quit due to an error, but in any case it's not there any more.

`mtag` (`str`)
The mtag of the viddle that just died.

`tweak-meta` *provisioner-id* [Protein]

Set or unset meta-parameters.

`set` (`list`, optional)
Set a provision's meta-parameter. The value should be a two- or three-element list of the form '`[`*`provision-name`*`,` *`meta-key`*`,` *`meta-val?`*`]`'. Omit the *meta-val* to unset a key.

`set-default` (`list`, optional)
Set a default meta-parameter. The value should be a one- or two-element list of the form '`[`*`meta-key`*`,` *`meta-val?`*`]`'. Omit the *meta-val* to unset a key.

Note that the `allowed-meta-tweak-keys` meta-parameter constrains the set of permissible changes.

3 The Viddle

When the provisioner receives a `provision!` request, it forks off a `qm-viddle` process to actually do the work of capturing and encoding the video.

3.1 Viddle Configuration

The viddle process is usually invoked without any arguments. In that case, the viddle expects to be able to read a binary slaw on its standard input, specifying the viddle configuration.

The viddle configuration should be a slaw map with the following keys:

`mtag` (`str`)
: The mtag with which this stream should render video.

`name` (`str`)
: The name of the provision being rendered.

`coord-pool` (`str`)
: The name of the coordination pool on which to deposit heartbeat and, possibly, thumbnail proteins.

`conf-pool` (`str`)
: The name of the pool to which the provisioner stores its configuration.

`pool` (`str`)
: The name of the pool on which to deposit video. Yeah!

`resource` (`map`)
: The resource map to use to construct the viddle capture source. Passed directly to `Fontana::Open`.

`provisioner` (`str`)
: The ID of the provisioner that spawned this viddle.

`properties` (`map`, optional)
: A schema list of properties for this viddle. See the discussion of `properties` in the documentation for the provisioner configuration file, for more information.

`meta` (`map`, optional)
: A map of meta-properties for this viddle, including any initial values to set for the properties.

  Meta parameters that are interpreted by the viddle include:

`video-passthrough-caps` (`str`)
`video-encoder` (`str`)
`video-bounding-box` (`list`)
`audio-passthrough-caps` (`str`)
`audio-encoder` (`str`)
`thumbnail-enabled` (`bool`)
`thumbnail-bounding-box` (`list`)
`thumbnail-encoder` (`str`)
`thumbnail-passthrough-decoder` (`str`)

For more information on the meanings of these keys, see the documentation above for provisioner configuration.

3.2 Invoking the Viddle Process Manually

You usually don't have to invoke the viddle yourself, but hey, sometimes a body needs to debug, right? So if you run `qm-provisioner` with the '`--debug-viddles`' option, the provisioner will serialize the viddle configuration slawx to disk. You can then invoke the viddle as `qm-viddle` *`conf-file`*.

3.3 Viddle Proteins

While it is running, a viddle process will periodically (every 4 seconds) output status messages on its coordination pool.

`viddle` [Protein]

A heartbeat message, indicating that a viddle is alive and kicking.

Ingests are as follows:

`name` (`str`)
: The name of the provision offered by this viddle.

`mtag` (`str`)
: The mtag with which this viddle is encoding. A viddle may be identified by mtag.

`pool` (`str`)
: The name of the pool to which this viddle is rendering. If possible, this will be an *absolute* pool name; see the pool addressing discussion below for more information.

`resource` (`map`)
: The state of the media source being provisioned. Note that in general this slaw map may contain more keys than the resource that was used to construct this viddle. For example, it might include the current timestamp, or other video state information. This value is whatever `Fontana::GetState()` returns.

`inbox` (`str`)
: The name of the drome pool for this viddle. The drome pool is used for communicating directly with the viddle.

`provisioner` (`str`, optional)
: The provisioner ID of the provisioner that spawned this viddle, if any.

`properties` (`map`, optional)
: The properties schema list for this viddle.

In addition to the status heartbeat protein, the viddle will render thumbnail images of what is currently playing, and deposit them on the coordination pool.

`thumbnail` [Protein]
: A thumbnail image.

`mtag` (`str`)
: The mtag of the stream that this thumbnail applies to.

`width` (`u16`)
`height` (`u16`)
: The width and height of this thumbnail image, in pixels.

`buf` (`protein`)
: A GStreamer buffer containing the actual thumbnail.

  Surely this topic merits some other documentation file; but until then, we will mention that the "data" ingest probably contains a `u8` array which is probably a PNG-format image.

  See Video in Yovo, for more information. (Scroll down to the section entitled "Representation of media via proteins".)

Finally, when the viddle finishes rendering, it will deposit a `viddle-finished` protein on the coordination pool.

`viddle-finished` *reason* [Protein]
: An indication that a viddle has shut down. The *reason* descrip indicates the condition that caused the viddle to shut down; current possibilities include `eos`, for end-of-stream, `error`, for an error condition, and `interrupted`, due to a `SIGINT` or `SIGTERM` signal.

`name` (`str`)
`mtag` (`str`)
`pool` (`str`)
`resource` (`map`)
`provisioner` (`str`, optional)
: The name, mtag, pool, resource, and provisioner ID that were used to create this viddle.

Note that robust clients should not rely on receiving a `viddle-finished` protein; `provision-end` is more reliable, and clients always have to fall back on expiry due to lack of heartbeats.

3.3.1 Drome-Pool Proteins

As hinted above in the discussion of viddle configuration, viddles provide for run-time settable and gettable properties.

To get the value of a property, deposit a `get-property` protein on the viddle's `inbox` (drome) pool:

`get-property` [Protein]

A request to get a property from a viddle. The one ingest is the property name:

`property` (`str`)
: The name of the property to fetch.

If the viddle does actually have a property with this name, a standard retort (see above) will be deposited on the viddle's `inbox`, with the value being the current value of that property. The type of the value depends on the property being fetched.

Similarly, setting a property also has a protein protocol:

`set-property!` [Protein]

A request to set a property on a viddle.

`property` (`str`)
: The name of the property to fetch.

`value`
: The value to set. The type of the value depends on the property being set.

There is no response to a `set-property!` protein; deposit another `get-property` if you want to know what, if anything, happened to the viddle.

4 Pool Addressing in a Distributed World

The Quartermaster is all about processes and pools. But obviously those processes and pools reside on certain machines, and while this detail can often be overlooked, in the end you probably want to play a video. The question is, how can we avoid making "early binding" decisions about where these pools and processes reside? How can we build flexibility *and* ease of use into the system?

The answer is twofold. First, as a point of synchronization, the coordination pool really binds the entire Quartermaster together. If it's not in the coordination pool, it's at least pointed to—addressed—from that pool. Our problem then becomes one of naming, to ensure that names are accessible by all participants that access the coordination pool.

The approach taken by the Quartermaster is to always communicate in *fully-qualified* pool names: pools with '`tcp://`' schema, and explicit ports. We rely on the Internet Protocol and the domain name system for the universal transport layer. All that's needed is the ability to transform a local pool name to a fully-qualified pool name, and vice versa.

Unfortunately, this isn't as easy as it sounds. There may be no `pool-tcp-server` instances running for a given set of pools. There may be more than one. You might be using a non-standard or user-specific pools path. Even if a server is there, `pool-tcp-server` may be, and probably is, running on a non-default port; and might even be running on only one of a number of IP addresses that a given machine has.

In the end the combinatorics win, and eventually the pool server has to tell pool users on what port and addresses it is offering the pools. The only channels available to a pool server are the pools themselves, so our convention is to require that a pool server deposit this information on a well-known pool: the `pool-server` pool.

`pool-server alive` [Protein]

A heartbeat, indicating the presence of a pool server available for locally-addressable pools.

`type` (`str`)
: The server type, for example '`tcp`'.

`hosts` (`list`)
: A list of IP addresses or hostnames on which the server is listening, as strings.

`port` (`s64`)
: The port on which the server is listening.

Heartbeats are deposited every 60 seconds. If no heartbeat has been received in the last 90 seconds for a given server, users should assume that it has died in some way.

In addition, when a pool server stops, it outputs a deadbeat.

`pool-server dead` [Protein]

A deadbeat, indicating that a pool server has stopped.

`type` (`str`)
`hosts` (`list`)
`port` (`s64`)
: The type, hostnames, and port on which the server was running.

Ideally the `pool-tcp-server` should itself output these proteins. Until that is the case, though, the user may work around this problem by running the prosthetic `qm-middle-manager`, which takes a Yaml slaw map as an argument, depositing it as the ingests for a heartbeat protein.

The end is that if everything is working, and there are heartbeats to the `pool-server` pool, pool names in the Quartermaster are expressed in a fully-qualified form. A programmer can perform the absolute-to-relative conversions with the `Router` class, included in `libQuartermaster`. Users can use the `pool-is-local`, `pool-absolute-name`, and `pool-relative-name` commands.

5 The ViddleSet

So! It's a complicated protocol, no? Well never fear, intrepid C++ hacker—you are a C++ hacker, no?—there is a class that handles all of the messy pool polling for you, the ViddleSet.

Link libQuartermaster into your application, and include '`ViddleSet.h`' header. The `ViddleSet` object is a `KneeObject` that can be added to your drome, enabling viddly goodness. There are hooks defined that will fire when viddles become known, or unknown in the case that they expire (`ViddleAdded` and `ViddleRemoved`). Additionally there are hooks that fire as we become aware of particular provisioners being associated with viddles (`ViddleHasProvisioner` and `ViddleLostProvisioner`), and that those viddles become available for provisioning (`ViddleAvailable` and `ViddleUnavailable`). (Recall that a DVI capture provision is known from the start, but only available when there is input.)

Finally, there are hooks that fire when a provision is actually live (`ViddleLive` and `ViddleDied`): when it actually has a viddle process rendering into a pool.

Well, I said "finally", but there is another hook, `ViddleHasThumbnail`, called when a thumbnail image is received for a known viddle.

All of these hooks will be called with three arguments: the `ViddleSet`, the `Viddle` object itself, and an `Atmosphere`. There are lots of data fields and accessors and methods and such on the `Viddle` objects themselves. Check '`ViddleSet.h`' for all the gory details.

6 The Sidler

Most of the operations afforded by the ViddleSet are exposed directly in the simple console utility, the *sidler*. You see, the sidler sidles up the Quartermaster, checks out what's available, perhaps engages in transactions, hangs around for a bit, and eventually saunters off somewhere.

Run the sidler as `qm-sidler` *`coord-pool`*. You will be presented with a list of known provisions on the console. The set of available commands is as follows:

`q` Quit the sidler

`[` Select the previous provision in the list. The selected viddle is indicated by a '`>`' character.

`]` Select the next provision in the list.

`w` Choose a provision by number.

`i` Print out information about the current provision on the console.

`*` Provision a viddle. When the viddle is provisioned, an asterisk ('`*`') will appear next to the name. Viddles that are available have an '`@`' next to them.

`.` Deprovision a viddle.

`g` Get a viddle property.

`s` Set a viddle property.

`+` Increase a numeric viddle property.

`-` Decrease a numeric viddle property.

`?`
`h` Show a brief help message. Actually, anything that's not recognized will cause the help message to be printed out.