Interface between server interface and driver in Queuing Streaming Interface PoC

Goals:

* support all interface needs between server interface and driver
* calls that pass data should be efficient
* support low latency in responding to consumer requests
* no need to hard code any driver name in server interface code (instead be based on configuration of server interface)
* give driver opportunity to set up and tear down
* be sure meets needs of Kafka driver

Main design decisions for the interface:

* driver contained in a class -- we create an instance of driver class
* expose outqueue object (added a MessageStream wrapper class) for driver to add it; convenience function to create MessageAndMetadata
* we pass an asyncio event loop to driver for it to schedule itself to run -- must use this in consume direction to fill outqueue

# Driver class

driver class must provide the following

\_\_init\_\_(self, driver\_args, event\_loop):

new instance created whenever server instance wants to do some work (may reuse instance if not already doing append or get work and if arguments are unchanged)

* driver\_args is passed-through admin configuration (a string that the server interface doesn’t understand)
* event\_loop is an [asyncio event loop](https://docs.python.org/3/library/asyncio-eventloop.html) that the driver can use

prepare\_for\_append\_stream(self, queue\_name):

called to tell driver of a new stream of appends than are going to come in; these should go to the end of the named queue

append(self, payload, ttl):

called to add new a messages to the queue. payload is a message payloads and ttl is the TTL for it

cancel\_append\_stream(self):

called to let the driver know that there will be no new appends for now and that it should free up any associated resources

init\_get\_stream(self, get\_message\_stream, queue\_name, starting\_marker, echo\_requested, include\_claimed):

called to tell driver that a new stream of messages is needed for return to a client. message\_stream\_queue is an instance of [MessageStream](#h.a7okh9irk4v4) to use to put messages the driver has available as a response to this request. Other arguments have same meaning as in the Marconi API.

cancel\_get\_stream(self):

called to let the driver know that there no more messages are needed for the previously requested stream of messages and that it should free up any associated resources.

# MessageStream

MessageStream is a class representing a finite-length buffer of messages in a stream. This is used for the driver providing messages to the server interface for get requests. The following methods are available to the driver:

MessageStream.space\_avail(self):

this method returns how many messages the buffer can currently accommodate. The driver can assume that this number will not go down until the driver adds some messages.

MessageStream.add\_message(self,payload,marker,id,ttl,age,claim\_id=None,claim\_client\_id=None):

this method adds a message to the message stream. The payload, marker, id (message ID), ttl, and age are the same as in Marconi. claim\_id and claim\_client\_id have the same meaning as in Marconi and must be provided if relevant.