1. Sending messages

Messages can be sent with the `send-message!` procedure, which can be called as `(send-message! mq message #:priority priority)`, where *`mq`* is the message queue and *`message`* is the message to send as a readable bytevector slice. This is an asynchronuous operation, so this procedure can return before the service has processed the message.

Depending on the transport, it might be possible for messages to be lost or received out-of-order. Some transports allow to explicitely allow messages to be lost or received out-of-order and would by default retransmit lost messages and reorder out-of-order messages; this behaviour can to a degree be controlled by setting the *priority-preference* flags.

These flags are not absolute, e.g. even if reliable transmission is requested, it is possible that the transport fail to transmit the message. The exact behaviour is transport-dependent!

`pref:unreliable`. Unreliable delivery is acceptable.

`pref:low-latency`. Low latency is desired, this cannot be meaningfully combined with `pref:cork-allowed`.

`pref:cork-allowed`. The transmission of a message can be delayed to combine this message with other messages into a larger transmission with less per-message overhead.

`pref:good-throughput`. High bandwith is desired; the method chosen for transmission should focus on overall throughput.

`pref:out-of-order`. Out-of-order delivery is acceptable.

These flags can be combined into a numeric value with the macro `prio-prefs` from `(gnu gnunet mq prio-prefs)`; the following code defines *x* as the numeric value of the flags `pref:unreliable` and `pref:out-of-order`:

```
(import (gnu gnunet mq prio-prefs))

(define x (prio-prefs pref:unreliable pref:out-of-order))
```

This numeric priority-preference can be passsed to `send-message!` as the optional *`priority`* keyword argument of `send-message!`. The transport of `connect/fibers` is always reliable and in-order. [notify-sent! callbacks][cancellation][queue size limits, `%suspicious-length`]