ewd-qoper8

Installation & Reference Guide

Version 1
24 January 2016

What is ewd-qoper8?

ewd-qoper8 is a Node.js-based message queue module. It provides you with:

- a memory-based queue within your main process onto which you can add JSON messages
- a pool of persistent child processes (*aka* worker processes) that run your message handler functions
- a worker process pool manager that will start up and shut down worker processes based on demand
- a dispatcher that processes the queue whenever a message is added to it, and attempts to send the message to an available worker process

It differs from most other message queues by preventing a worker process from handling more than one message at a time. This is by deliberate design.

You determine the maximum size of the worker process pool. If no free worker processes are available, messages will remain on the queue. The queue is automatically processed whenever:

- a new message is added to the queue
- a worker process completes its processing of a message and returns itself to the available pool

The structure of messages is entirely up to you, but:

- they are JavaScript objects
- they should always have a type property

How messages are handled within a worker process is up to you. You define a handler method/function for each message type you expect to be added to the queue.

ewd-qoper8 is highly customisable. For example, the master and/or worker processes can be customised to connect to any database you wish, and it can be integrated with a Node.js-based web-server module such as Express, and with a web-socket module such as socket.io. You can also over-ride the memory-based queue and implement your own persistent one.

Installing ewd-qoper8

```
npm install ewd-qoper8
```

Getting Started With ewd-qoper8

ewd-qoper8 is pre-configured with a set of default methods that essentially take a "do nothing" action. You can therefore very simply test ewd-qoper8 by writing and running the following script file:

```
var qoper8 = require('ewd-qoper8');
var q = new qoper8.masterProcess();
console.log(q.version() + ' running in process ' + process.pid);
q.start();
```

(You'll find a copy of this in the /test sub-folder within the ewd-qoper8 module folder - see test1.js)

This example will start the ewd-qoper8 master process, with a worker process pool size of 1. It will then sit waiting for messages to be added to the queue - which isn't going to happen in this script. When ewd-qoper8 starts, it does not start any worker processes. A worker processes is only started when:

- a message is added to the queue; AND
- no free worker process is available; AND
- the maximum worker process pool size has not yet been reached

So if you run the above script you should just see something like this:

```
robtweed@ubuntu:~/ewdjs$ node node_modules/ewd-qoper8/lib/test/test1
ewd-qoper8 Build 1; 22 February 2016 running in process 10947
Worker Bootstrap Module file written to node_modules/ewd-qoper8-worker.js
========================================================
ewd-qoper8 is up and running.  Max worker pool size: 1
========================================================
```

Notice the second line of output: ewd-qoper8 always automatically creates a file containing the core worker process logic from which it bootstraps itself. However, since the test script doesn't add any messages to ewd-qoper8's queue, no worker processes are created and the master process will just sit waiting.

To stop ewd-qoper8, just press CTRL & C within the process console, or send a SIGINT message from another process, eg:

```
kill 10947
```

Note: the number should be the process Id for the ewd-qoper8 master process

In either case you should see something like the following:

```
^C*** CTRL & C detected: shutting down gracefully...
Master process will now shut down
```

Alternatively we could add something like the following at the end of the test script:

```
setTimeout(function() {
  q.stop();
}, 10000);
```

This time, after 10 seconds you'll now see ewd-qoper8 shut itself down

Testing Adding a Message to the ewd-qoper8 Queue

You use ewd-qoper8's *addToQueue()* method to add messages to the queue:

Messages are JavaScript objects and must, at the very least, have a type property defined. The type value is entirely up to you.

So we could do the following test (see test2.js in the /test sub-folder):

```
var qoper8 = require('ewd-qoper8');
var q = new qoper8.masterProcess();

q.on('started', function() {
  console.log(q.version() + ' running in process ' + process.pid);

  var messageObj = {
    type: 'testMessage1',
    hello: 'world'
  };
  this.addToQueue(messageObj);

});

q.start();

setTimeout(function() {
  q.stop();
}, 10000);
```

Any ewd-qoper8 activity should be defined within its ‘started’ event handler. In the example above you can see a message object being created and queued using this.addToQueue() within the q.on(‘started’) handler.

Running the above script should produce output similar to the following:

```
robtweed@ubuntu:~/ewdjs$ node node_modules/ewd-qoper8/lib/test/test2
Worker Bootstrap Module file written to node_modules/ewd-qoper8-worker.js
========================================================
ewd-qoper8 is up and running.  Max worker pool size: 1
========================================================
ewd-qoper8 Build 1; 22 February 2016 running in process 10997
no available workers
No worker module was specified, so
defaultModule loaded into child process 11002
Worker process 11002 starting...
master process received response from worker 11002: {"type":"workerProcessStarted","ok":11002}
new worker 11002 started and ready so process queue again
checking if more on queue
worker 11002 received message: {"type":"testMessage1","hello":"world"}
Worker 11002: no handler defined for message {"type":"testMessage1","hello":"world"}
master process received response from worker 11002: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11002 which is back in
available pool
signalling worker 11002 to stop
worker 11002 received message: {"type":"qoper8-exit"}
Worker process 11002 stopping...
worker process 11002 will now shut down
Master process will now shut down
```

You’ll see that it reports starting a worker process to handle the message that was added to the queue - in this case with a process id (PID) of 11002.

As we’ve not yet specified a specific handler function for incoming messages, by default the worker process simply sends a “finished” message back to the master process, which, as a result, returns the worker process to the available pool.

Note that worker process 11002 is not shut down, but persists, waiting and available for handling any future messages that are added to the queue (which, of course, in this example, doesn't happen).

As in the previous example, after 10 seconds, ewd-qoper8 is instructed to shut down. This time you can see that the master process first signals the worker process to shut down, then waits for a few seconds before shutting itself down.

If you leave ewd-qoper8 running, you'll see it regularly reporting the duration that each worker process has been idle. If the default idle limit (1 hour, or 3,600,000ms) is exceeded, ewd-qoper8 will shut down the worker process.

As before, if you don't use ewd-qoper8's stop() method within your script, CTRL&C or a SIGINT message will shut down ewd-qoper8.

Handling Multiple Messages

So far we've only queued a single message. It's worth looking at what happens if multiple messages are added to the queue. The following example (see test3.js in the /test folder) will queue two messages:

```
var qoper8 = require('ewd-qoper8');
var q = new qoper8.masterProcess();

q.on('started', function() {

  var messageObj = {
    type: 'testMessage1',
    hello: 'world'
  };
  this.addToQueue(messageObj);

  messageObj = {
    type: 'testMessage2',
    hello: 'rob'
  };
  this.addToQueue(messageObj);
});

q.start();

setTimeout(function() {
  q.stop();
}, 5000);
```

The output should look something like this:

```
robtweed@ubuntu:~/ewdjs$ node node_modules/ewd-qoper8/lib/test/test3
Worker Bootstrap Module file written to node_modules/ewd-qoper8-worker.js
========================================================
ewd-qoper8 is up and running.  Max worker pool size: 1
========================================================
no available workers
no available workers
No worker module was specified, so
defaultModule loaded into child process 11105
Worker process 11105 starting...
master process received response from worker 11105: {"type":"workerProcessStarted","ok":11105}
new worker 11105 started and ready so process queue again
checking if more on queue
more items found on queue - processing again
no available workers
worker 11105 received message: {"type":"testMessage1","hello":"world"}
Worker 11105: no handler defined for message {"type":"testMessage1","hello":"world"}
master process received response from worker 11105: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11105 which is back in
available pool
checking if more on queue
worker 11105 received message: {"type":"testMessage2","hello":"rob"}
Worker 11105: no handler defined for message {"type":"testMessage2","hello":"rob"}
master process received response from worker 11105: {"type":"testMessage2","finished":true}
Master process has finished processing response from worker process 11105 which is back in
available pool
signalling worker 11105 to stop
worker 11105 received message: {"type":"qoper8-exit"}
Worker process 11105 stopping...
worker process 11105 will now shut down
Master process will now shut down
```

Both messages are immediately added to the queue. Adding the first triggers a worker process to be started and the message is dispatched to the worker as soon as it has signalled back that it has started up and is ready for use.

Because we haven't specified otherwise, ewd-qoper8 defaults to a worker process pool size of 1, so the second message will remain on the queue until the single worker process (PID 11105 in this case) has finished handling the first message. As soon as the worker process becomes available, the queued message is dispatched to it for processing.

This sequence will continue automatically until the queue is empty.

Try editing the example script so that it adds more messages to the queue, and see what happens.

Defining The Worker Process Pool Size

Configuration of ewd-qoper8 is done by modifying its various default properties. The best place to do this is within a "start" event handler.

For example, you can specify a worker process pool size of 2 in one of two ways:

```
q.on('start', function() {
  this.setWorkerPoolSize(2);
});
```

or:

```
 q.on('start', function() {
   this.worker.poolSize = 2;
 });
```

If you modify the previous example to use 2 worker processes, you should see a quite different result, particularly if you queue up a larger number of message. For example, take the following script (see test4.js in the /test folder):

```
var qoper8 = require('ewd-qoper8');
var q = new qoper8.masterProcess();

q.on('start', function() {
  this.setWorkerPoolSize(2);
});

q.on('started', function() {
  var noOfMessages = 5;
  var messageObj;
  for (var i = 0; i < noOfMessages; i++) {
    messageObj = {
      type: 'testMessage1',
      hello: 'world'
    };
    this.addToQueue(messageObj);
  }
});

q.start();

setTimeout(function() {
  q.getAllWorkerStats();
}, 5000);

setTimeout(function() {
  q.stop();
}, 10000);
```

The worker pool size is being increased to 2 within the q.on('start') handler. Within the q.on('started') handler, 5 messages are being queued.

After 5 seconds, the q.getAllWorkerStats() method is invoked: this will ask each worker process to report back a number of vital statistics, including the number of messages it has processed.

Here's an example of the output generated by this script:

```
robtweed@ubuntu:~/ewdjs$ node node_modules/ewd-qoper8/lib/test/test4
Worker Bootstrap Module file written to node_modules/ewd-qoper8-worker.js
========================================================
ewd-qoper8 is up and running.  Max worker pool size: 2
========================================================
no available workers
no available workers
no available workers
no available workers
no available workers
No worker module was specified, so
master process received response from worker 11248: {"type":"workerProcessStarted","ok":11248}
new worker 11248 started and ready so process queue again
checking if more on queue
```

```
more items found on queue - processing again
no available workers
defaultModule loaded into child process 11253
Worker process 11253 starting...
master process received response from worker 11253: {"type":"workerProcessStarted","ok":11253}
new worker 11253 started and ready so process queue again
checking if more on queue
more items found on queue - processing again
no available workers
master process received response from worker 11248: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11248 which is back in
available pool
checking if more on queue
more items found on queue - processing again
no available workers
worker 11253 received message: {"type":"testMessage1","hello":"world"}
Worker 11253: no handler defined for message {"type":"testMessage1","hello":"world"}
master process received response from worker 11248: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11248 which is back in
available pool
checking if more on queue
more items found on queue - processing again
no available workers
master process received response from worker 11253: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11253 which is back in
available pool
checking if more on queue
master process received response from worker 11248: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11248 which is back in
available pool
worker 11253 received message: {"type":"testMessage1","hello":"world"}
No worker module was specified, so
master process received response from worker 11253: {"type":"testMessage1","finished":true}
Master process has finished processing response from worker process 11253 which is back in
available pool
defaultModule loaded into child process 11248
Worker 11253: no handler defined for message {"type":"testMessage1","hello":"world"}
Worker process 11248 starting...
worker 11248 received message: {"type":"testMessage1","hello":"world"}
Worker 11248: no handler defined for message {"type":"testMessage1","hello":"world"}
worker 11248 received message: {"type":"testMessage1","hello":"world"}
Worker 11248: no handler defined for message {"type":"testMessage1","hello":"world"}
worker 11248 received message: {"type":"testMessage1","hello":"world"}
Worker 11248: no handler defined for message {"type":"testMessage1","hello":"world"}
worker 11253 received message: {"type":"qoper8-getStats"}
master process received response from worker 11253: {"type":"qoper8-stats","stats":{"pid":
11253,"upTime":"0 days 0:00:04","noOfMessages":4,"memory":
{"rss":"22.11","heapTotal":"8.85","heapUsed":"5.25"}}}
worker 11248 received message: {"type":"qoper8-getStats"}
master process received response from worker 11248: {"type":"qoper8-stats","stats":{"pid":
11248,"upTime":"0 days 0:00:04","noOfMessages":5,"memory":
{"rss":"22.02","heapTotal":"8.85","heapUsed":"5.27"}}}
signalling worker 11248 to stop
signalling worker 11253 to stop
worker 11253 received message: {"type":"qoper8-exit"}
Worker process 11253 stopping...
worker process 11253 will now shut down
worker 11248 received message: {"type":"qoper8-exit"}
Worker process 11248 stopping...
worker process 11248 will now shut down
```

```
Master process will now shut down
```

This time you'll see that ewd-qoper8 started 2 worker processes (PIds 11248 and 11253). In the example above, they handled 4 and 5 messages respectively. You'll find that the balance of messages handled by each worker may vary each time you run the script: it takes time for each worker process to start up and become ready for handling messages, so the first one may have handled several messages before the second one is ready.

You're probably wondering why a total of 9 messages was handled by the worker processes when only 5 were added to the queue. The reason is that a message is sent to each worker as part of its startup sequence, and another message was sent to each worker to ask for its vital statistics (as a result of running the q.getAllWorlerStats() method).

Defining a Worker Process Message Handler

In the examples above the worker processes have been applying a default "message" event handle. That's what's generating output lines such as this:

```
worker 11253 received message: {"type":"testMessage1","hello":"world"}
Worker 11253: no handler defined for message {"type":"testMessage1","hello":"world"}
```

You customise the behaviour of the worker processes by creating a Worker Handler Module that ewd-qoper8 will load into each worker process when they are started.

Your module can define any of the following event handlers:

- s**tart**: this will be invoked whenever a worker process starts, at the point at which it is ready for use
- **stop**: this will be invoked just before a worker process closes down
- **message**: this is invoked whenever a message is received by the worker. This is where you define your logic for handling all messages (with the exception of ewd-qoper8's own control messages)

Example Worker Module

Here's a simple example of a worker module. In fact this is the default module that is loaded into each worker process if you don't specify your own:

```
function workerListeners(worker) {

  worker.on('start', function() {
    if (this.log) console.log('Worker process ' + process.pid + ' starting...');
  });

  worker.on('message', function(messageObj) {
    if (this.log) console.log('Worker ' + process.pid + ': no handler defined for message ' +
JSON.stringify(messageObj));
    this.hasFinished(messageObj.type);
  });

  worker.on('stop', function() {
    if (this.log) console.log('Worker process ' + process.pid + ' stopping...');
  });
```

```
        };

        module.exports = workerListeners;
```

You should always adhere to the pattern shown above:

- create a function that is exported from the module
- the function should have a single argument: worker
- within the function you can define any or all of the worker.on('start'), worker.on('end') and worker.on('message') handler functions.

The start event handler

Your start event handler is where you can do things such as connect to databases or load other modules that you'll need in your worker process.

Within the handler's callback function, 'this' provides you access to all the worker's methods and properties.

The stop event handler

Your stop event handler is where you can do things such as cleanly disconnect from databases or tidy up other resources before the worker process is terminated.

Within the handler's callback function, 'this' provides you access to all the worker's methods and properties.

The message event handler

The message event handler is where you'll define how to process all incoming messages that you have added to the queue. How they are processed is entirely up to you.

Its callback function takes one argument:

- **messageObj:** the raw incoming message object, sent from the master process's queue.

Within the handler's callback function, 'this' provides you access to all the worker's methods and properties. What you do with the message within the worker process is entirely up to you.

Once you've finished processing the message, you send the results back to the master process by invoking the worker's ***hasFinished()*** method. This takes two optional arguments:

- **type**: the message type that has been handled
- **resultObj**: an object containing the results that are to be returned to the master process

On receipt of the message created by the hasFinished() method, the master process will return the worker back to the available pool

You can optionally send more messages back to the master process during processing, prior to using the hasFinished() method. To do this, use the worker's ***returnMessage()*** method. This takes the same two arguments as the hasFinished() method (type and resultObj). The difference is that on receipt of the message, the master process does not return the worker process to the available pool.

Make sure that your ***on('message')*** handler logic always ends with an invocation of the ***this.hasFinished()*** function. ***this.hasFinished()*** should only be invoked once within an ***on('message')*** handler function.

Configuring ewd-qoper8 To Use Your Worker Handler Module

You instruct ewd-qoper8 to load your worker module by setting the property this.worker.module from within the on('start') method handler.

The module you specify will be loaded (using *require()*) into each worker process when it is started.

For example, if you saved your module in *./node_modules/exampleModule.js*, then you instruct ewd-qoper8 to load it as follows, eg:

```
q.on('start', function() {
  this.worker.module = 'exampleModule';
});
```

If your module is saved elsewhere, modify the module path accordingly. For example if you look at the example script test5.js in the /test folder, you'll see that it specifies:

```
q.on('start', function() {
  this.setWorkerPoolSize(2);
  this.worker.module = 'ewd-qoper8/lib/test/test-workerModule1';
});
```

If you look at this module, it contains the following:

```
var count = 0;

function workerListeners(worker) {

  worker.on('message', function(messageObj) {
    count++;
    var results = {
      count: count,
      time: new Date().toString()
    };
    this.hasFinished(messageObj.type, results);
  });

};

module.exports = workerListeners;
```

Note that it doesn't specify a start or stop handler - these are optional. Try running the test5.js script to see the effect of this module on the messages returned to the master process.

Handling the Results Object Returned from a Worker

When a results object is returned from a Worker process, you normally define how the master process should handle it. So far we've been letting ewd-qoper8's default action to take place, which is to simply report the returned result message to the console.

To handle messages returned by worker processes, you define an on('response') handler, eg:

```
q.on('response', function(responseObj, pid) {
  console.log('Received from worked ' + pid + ': ' + JSON.stringify(responseObj, null, 2));
});
```

As you can see above, the on('response') handler callback function takes two arguments:

- **resultsObj:** the raw incoming results object, sent from the worker process.
- **pid**: the process Id of the worker that handled the original message and sent this response

How you handle each returned message and what you do with it is up to you. Within the on('response') handler's callback function, 'this' provides access to all of the master process's properties and methods.

Note that your on('response') handler function method intercepts all messages returned by worker processes, including ewd-qoper8's own ones. You'll be able to distinguish them because their type will have 'qoper8-' as a prefix.

For a worked example, take a look at test6.js in the /test folder:

```
var qoper8 = require('ewd-qoper8');

var q = new qoper8.masterProcess();

q.on('start', function() {
  this.setWorkerPoolSize(2);
  this.worker.module = 'ewd-qoper8/lib/test/test-workerModule1';
});

q.on('response', function(responseObj, pid) {
  console.log('Received from worked ' + pid + ': ' + JSON.stringify(responseObj, null, 2));
});

q.on('started', function() {
  var noOfMessages = 5;
  var messageObj;
  for (var i = 0; i < noOfMessages; i++) {
    messageObj = {
      type: 'testMessage1',
      hello: 'world'
    };
    this.addToQueue(messageObj);
  }
});

q.start();

setTimeout(function() {
  q.getAllWorkerStats();
}, 5000);

setTimeout(function() {
  q.stop();
}, 10000);
```

Customising What Happens to the Master Process When it is Started

You've already seen how the q.on('start') event can be used to over-ride default properties such as the worker pool size and to specify the path of your worker module.

The on('start') event is also where you can customise your master process, for example if you want to connect it to a database and load other modules. You can augment the master process's properties and methods from within this event's callback function: simply augment 'this'.

Customising What Happens to the Master Process When it is Stopped

You can optionally intercept the ewd-qoper8 master process shut-down sequence, allowing you to release and tidy up resources, and shut down any linked services tidily.

You can do this by handling the on('stop') event e.g.:

```
var qoper8 = require('ewd-qoper8');
var q = new qoper8.masterProcess();

q.on('stop', function() {
  // perform your shutdown logic here
});

q.start();
```

Within the 'stop' event handler's callback function, 'this' provides access to all the master process's properties and methods.

Examples Included with ewd-qoper8

You'll find a number of example scripts in the /lib/test folder within your node_modules/ewd-qoper8 directory.

To run them, ensure you're in the directory that you'd been in when you first installed ewd-qoper8.

From that directory, you can invoke the command:

```
node node_modules/ewd-qoper8/lib/test/[name]
```

where [name] is the script name, e.g. test1

For example, to run test1.js:

```
node node_modules/ewd-qoper8/lib/test/test1
```

You can use the test files as examples of how to use ewd-qoper8.

Benchmark

Included in the /test folder is a script named benchmark.js
This will generate a specified number of messages and add them to the queue in batches. The messages are round-tripped to your worker process(es).

The benchmark will measure how long it takes to process the complete set of messages and provide statistics such as the rate per second and the number of messages handled by each worker process.

To run the benchmark:

```
node node_modules/ewd-qoper8/lib/test/benchmark [[no of workers] [no of messages] [messages/batch] [pause between each batch (ms)]
```

The default values, if not specified as command line parameters are:

- no of workers: 1
- no of messages: 100,000
- messages per batch: 500
- pause between each batch: 51ms

When you run the benchmark, it will show you if the queue continues to grow, or any time the queue is exhausted. Maximum throughput will be achieved if you can approximate a steady-state whereby the queue doesn't exceed 2,000 - 3,000 and it is never exhausted. You should also find that maximum throughput will be when the number of worker processes plus the master process equals the number of CPU cores available.

Use the defaults as starting points.

If you find that the queue just keeps building up throughout a run, increase the pause between batches. This will allow ewd-qoper8 to consume more of the queued messages before a next batch is added.

Conversely if you see lots of reports of the queue being exhausted during a run, decrease the pause between batches, so you're topping up the queue as fast as it is being consumed.

The default queue used by ewd-qoper8 is simply a JavaScript array. JavaScript performance when processing arrays decreases as array size increases, so it's a good idea to try to maintain as small a queue size as possible for maximum throughput.

Here's some examples of how to run the benchmark:

```
node node_modules/ewd-qoper8/lib/test/benchmark
```

- no of workers: 1
- no of messages: 100,000
- messages per batch: 500
- pause between each batch: 51ms

node node_modules/ewd-qoper8/lib/test/benchmark 2

- no of workers: 2
- no of messages: 100,000
- messages per batch: 500
- pause between each batch: 51ms

node node_modules/ewd-qoper8/lib/test/benchmark 1 10000

- no of workers: 1
- no of messages: 10,000
- messages per batch: 500
- pause between each batch: 51ms

node node_modules/ewd-qoper8/lib/test/benchmark 2 5000 100

- no of workers: 2
- no of messages: 5,000
- messages per batch: 100
- pause between each batch: 51ms

node node_modules/ewd-qoper8/lib/test/benchmark 2 10000 1000 102

- no of workers: 2
- no of messages: 10,000
- messages per batch: 1000
- pause between each batch: 102ms

By way of reference, running with the defaults on a Ubuntu 14.04 64-bit virtual machine (with 1 virtual CPU) under VMWare Fusion on an i5 MacBook Air shows a throughput of 8,500 - 9,500 messages/second

Integrating ewd-qoper8 with Express

ewd-qoper8 is not really designed to be a standalone module. More sensibly, it is designed to be used in conjunction with a server module of some sort. The very popular and increasingly standard web server module Express is a good example of something with which ewd-qoper8 can be integrated.

So, for example, ewd-qoper8's worker processes can be used to handle incoming HTTP requests, with the output from your worker-process module message handlers being returned to the browser or web server client.

To make this integration even easier, you should install a module named ewd-qoper8-express:

```
npm install ewd-qoper8-express
```

You'll find an example of how to integrate Express with ewd-qoper8 in the /test directory: look for the file express1.js. You'll see that it contains the following:

```
var express = require('express');
var bodyParser = require('body-parser');
var qoper8 = require('ewd-qoper8');
var qx = require('ewd-qoper8-express');

var app = express();
app.use(bodyParser.json());

var q = new qoper8.masterProcess();
qx.init(q);

app.post('/qoper8', function (req, res) {
  qx.handleMessage(req.body, function(resultObj) {
    res.send(resultObj);
  });
});

q.on('started', function() {

  this.worker.module = 'ewd-qoper8/lib/test/express-module';

  var server = app.listen(8080, function () {

    var host = server.address().address
    var port = server.address().port

    console.log("EWD-Express listening at http://%s:%s", host, port);
    console.log('__dirname = ' + __dirname);
  });

});

q.start();
```

This example will start ewd-qoper8, and when it has started and ready, it will start Express, listening on port 8080. It also sets the worker module file to be the express-module.js file that you'll also find in the /test folder.

It's pretty simple, just echoing back the incoming message and adding a few properties of its own:

```
function workerListeners(worker) {

  worker.on('message', function(messageObj) {
    var results = {
      youSent: messageObj,
      workerSent: 'hello from worker ' + process.pid,
      time: new Date().toString()
    };
    this.hasFinished(messageObj.type, results);
  });

};

module.exports = workerListeners;
```

Express will be watching for incoming HTTP POST requests with a URL path of /qoper8. The body of these requests will contain a JSON payload. Make sure the Content-Type for these HTTP requests is application/json

Note the use of the standard bodyParser.json middleware to parse the JSON and make it available as req.body.

This JSON payload is then handled by the ewd-qoper8-express function named handleMessage. This queues the message and awaits the response from the ewd-qoper8 worker process to which the message was dispatched. The returned result object is provided to you in its callback function. All you need to do is return it to the HTTP client using:

```
res.send(resultObj);
```

Note the way ewd-qoper8-express is initialised:

```
var qx = require('ewd-qoper8-express');
var q = new qoper8.masterProcess();
qx.init(q);
```

Try using a REST client (eg the Chrome Advanced REST Client extension), and POST to the URL /qoper8 a JSON payload such as this:

```
{
  "type": "HTTPMessage",
  "a": "hello"
}
```

You should get a response back such as this:

```
{
  “type":"HTTPMessage",
  “finished":true,
  “message":{
    “youSent":{
      “type":"HTTPMessage",
      “a":"hello"
    },
    "workerSent":"hello from worker 13809”,
    "time":"Fri Feb 19 2016 02:02:33 GMT+0000 (GMT)”
  }
}
```

Using WebSockets With ewd-qoper8 and Express

ewd-qoper8 will also work with web socket messages. See the express2.js test file which contains the following:

```
var express = require('express');
var bodyParser = require('body-parser');
var qoper8 = require('ewd-qoper8');
var qx = require('ewd-qoper8-express');

var app = express();
app.use(bodyParser.json());

var q = new qoper8.masterProcess();
qx.init(q);

app.post('/qoper8', function (req, res) {
  qx.handleMessage(req.body, function(resultObj) {
    delete resultObj.finished;
    res.send(resultObj);
  });
});

app.get('/', function (req, res) {
 res.sendFile(__dirname + '/index.html');
});

q.on('started', function() {

 this.worker.module = 'ewd-qoper8/lib/test/express-module';
 var q = this;

 var server = app.listen(8080, function () {

   var host = server.address().address
   var port = server.address().port

   console.log("EWD-Express listening at http://%s:%s", host, port);
   console.log('__dirname = ' + __dirname);
 });

 var io = require('socket.io')(server);

 io.on('connection', function (socket) {
   socket.on('my-request', function (data) {
     qx.handleMessage(data, function(resultObj) {
       delete resultObj.finished;
       socket.emit('my-response', resultObj);
     });
   });
 });

});

q.start();
```

This example uses the same worker module as before, but this time it also loads the socket.io module. The socket.io event handler for incoming web socket messages simply makes use of the ewd-qoper8-express handleMessage() function, just as in the previous example. The only difference is that instead of returning the message using:

```
res.send(resultObj);
```

it returns a WebSocket message as the response using:

```
socket.emit('my-response', resultObj);
```

You'll notice that this example deletes the finished property from the response before sending it. This is optional - the finished property is really for use within ewd-qoper8 to instruct the master process to release the worker process back to the available pool. For completeness, it's included in the response object that's made available to you in the handleMessage's callback function - in many/most cases you can probably delete it too before returning the response to the awaiting client or browser.

In order to test this example, you need to load an HTML file - there's one already created in the /test folder (index.html). If you start the example script and put the following URL into a browser:

http://192.168.1.193:8080/

change the IP address as appropriate

it should load the index.html page. This just contains a button that contains the text "Click Me":

```
<html>
  <head>
    <title id="ewd-qoper8 Demo"></title>
  </head>
  <body>

    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect();
      socket.on('my-response', function (data) {
        console.log('ewd-qoper8 message received: ' + JSON.stringify(data, null, 2));
      });

      function sendMessage() {
        var message = {
          type: 'testSocketRequest',
          hello: 'world'
        };
        socket.emit('my-request', message);
      }
    </script>

    <div>
      <button onClick='sendMessage()'>Click me</button>
    </div>

  </body>
</html>
```

When the button is clicked it will send a web socket message to Express / socket.io, and will show the returned response web socket message in the console.

Just as in the first example, it should look something like this:

```
ewd-qoper8 message received: {
  "type": "testSocketRequest",
  "message": {
    "youSent": {
      "type": "testSocketRequest",
      "hello": "world"
    },
    "workerSent": "hello from worker 13826",
    "time": "Fri Feb 19 2016 02:27:16 GMT+0000 (GMT)"
  }
}
```