

Using Chat GPT for coding

ChatGPT is a very good tool to support you for coding. It will not do your job - but help you especially for the boring tasks. It is on one hand supprisingly intelligent but sometimes also supprisingly stupid and still very convincing. Always check, what he is doing.

The following script describes a way how to use ChatGPT. He remembers many things and you can influence it well how and how good he helps you. I use a simple class here to demonstrate how ChatGPT can be used.

ChatGPT - even in the Plus version - has a limit of 2048 characters for generation. You have to ask him to generate specific things and not extended classes or programs.

I use the ChatGPT Plus version (20\$ per month) in the version 3.5, bringen fastest results.

The example code

It is used in front of a serial connection, pushes incoming transfer commands in a queue and then send them one after the other. As the serial connection is a long distance connection, delays between sending data packages reduces transfer errors.

```
'use strict'

const { Callbacks, delay } = require('./index')

/**
 * Callback to publish messages to the mqtt broker
 * @callback Task
 * @param {Any} parameter parameter for the task
 */

/**
 * Creates a queue to manage one task after another with a guaranteed delay.
 *
 * @param {number} delayInMilliseconds - The delay, in milliseconds, between two
tasks.
 */
class TaskQueue {
  constructor(delayInMilliseconds) {
    this._delayInMilliseconds = delayInMilliseconds
    this._taskQueue = []
    this._callbacks = new Callbacks(['task'])
    this._running = false
  }

  /**
   * Sets a callback for a specific event.
   *
   * @param {string} event - The name of the event (not case sensitive) for the
callback. Supported events: 'task'.

```

```

    * @param {function} callback - The function to be called when the event
occurs.
    * @throws {Error} If the event is not supported.
    * @throws {Error} If the callback is not a function.
    */
    on(event, callback) {
        this._callbacks.on(event, callback)
    }

    /**
     * Runs the next task in the queue, with a delay between tasks.
     *
     * @param {any} parameter - The parameter to be passed to the task function.
     * @private
     */
    async _runTasks() {
        if (this._running) {
            return
        }
        this._running = true
        while (this._taskQueue.length !== 0) {
            const parameter = this._taskQueue.pop()
            await this._callbacks.invokeCallbackAsync('task', parameter)
            await delay(this._delayInMilliseconds)
        }
        this._running = false
    }

    /**
     * Adds a new task to the queue and starts processing it.
     *
     * @param {any} taskParameter - The parameter to be passed to the task
function.
     */
    addTask(taskParameter) {
        this._taskQueue.push(taskParameter)
        this._runTasks()
    }
}

module.exports = TaskQueue

```

The first thing I do, is to send him the code:

I have new code for you, I will tell you what to do with it afterwards. *Sure, please share the new code with me and let me know what you need me to do with it.*

Then I pasted the code: *What would you like me to do with the new code?*

Is there any logical error?

I didn't notice any logical errors in the code. However, there are a few potential issues:

If a new task is added while the queue is being processed, it will not be immediately processed. Instead, it will be processed after the current batch of tasks is finished processing. This may not be the desired behavior for some use cases.

If the `delayInMilliseconds` is set to zero, the tasks will be executed without any delay. This may cause performance issues if there are a large number of tasks to process.

The `addTask` method does not return a promise, so it may be difficult to know when all tasks have been processed.

Improving your code

Maybe Chat GPT found improvements, maybe not. If so, he can help to implement the improvements. Ask him what you want, he will generate Code, example:

Let `addTask` return a promise

```
/**
 * Adds a new task to the queue and starts processing it.
 *
 * @param {any} taskParameter - The parameter to be passed to the task
function.
 * @returns {Promise} - A Promise that resolves when the task has been added
to the queue.
 */
async addTask(taskParameter) {
    return new Promise((resolve) => {
        this._taskQueue.push(taskParameter)
        this._runTasks()
        resolve()
    })
}
```

This is indeed a solution, but is it the best one? Let me try ...

`_runTasks` is already an async function can't we just "return `this._runTasks()`" and forget about the explicit Promise?

Yes, you are correct. Since `_runTasks` is already an async function, we can just return its result instead of explicitly creating a new promise. So we can replace `return new Promise(resolve => resolve(this._runTasks()))`; with `return this._runTasks()`;. Here's the updated code:

```
addTask(taskParameter) {
    this._taskQueue.push(taskParameter)
    return this._runTasks()
}
```

This really impresses me. If you are new on Promises and async this is not obvious. You even do not have to declare "addTask" as async. "_runTasks" is async and thus it will ALWAYS return a promise. Thus addTask will return a promise - even if it is not async. Let me ask ChatGPT

Do we have to declare addTask as async?

No, we don't have to declare addTask as async anymore, since we changed its implementation to return a promise instead of using a callback. We can simply return the promise returned by _runTasks() inside addTask.

Please add a comment to the function

He now generated something different. Its a matter of tast, still perhaps more clear that we have a promise ...

```
/**
 * Adds a new task to the queue and starts processing it.
 *
 * @param {any} taskParameter - The parameter to be passed to the task
function.
 * @returns {Promise} A promise that resolves when the task is added to the
queue.
 */
addTask(taskParameter) {
  this._taskQueue.push(taskParameter)
  const promise = this._runTasks()
  console.log(`Task added to queue with parameter: ${taskParameter}`)
  return promise
}
```

You can really use it to generate documentation. It is impressive how much he gets out of the code and the variable names to create documentation. In this case he generated a log output - this is not intended.

I do not like you to add console.log outputs without demand

My apologies, I will avoid adding console.log outputs in the future without your request.

Are there optimizations for _runTask?

One possible optimization for _runTasks() could be to use a while loop instead of a for loop to continuously process tasks as long as there are any in the queue, rather than only processing one task per function call. This would allow for more efficient use of the function by reducing the number of function calls needed to process all tasks in the queue. Additionally, since this is an asynchronous function, it could be optimized by making use of Promise.all() to allow for parallel processing of tasks, potentially reducing the total execution time. However, the suitability of these optimizations would depend on the specific requirements and constraints of the use case.

The use case is to run one task after the other, please remember this

Yes, you're right. Since the use case is to run one task after the other, there is no need to process the entire queue in a loop. Instead, we can simply process one task at a time and then wait for the delay before processing the next task.

Here is an optimized version of _runTasks() that processes only one task at a time:

```

async _runTasks() {
  if (this._running) {
    return
  }
  this._running = true
  while (this._taskQueue.length !== 0) {
    const parameter = this._taskQueue.shift()
    await this._callbacks.invokeCallbackAsync('task', parameter)
    if (this._taskQueue.length !== 0) {
      await delay(this._delayInMilliseconds)
    }
  }
  this._running = false
}

```

In this version, we use `shift()` instead of `pop()` to remove the first task from the queue, and we check if there are any more tasks in the queue before waiting for the delay. This ensures that we only process one task at a time and wait for the delay before processing the next task.

"Shift" is of course better than "pop". Suspending the delay when the queue is empty is also a sensible hint. In this case, however, it does not work - the delay between tasks must be guaranteed. So it must not be possible that the queue runs empty, immediately followed by a new task, which is processed directly without waiting.

You can see that the documentation does not match the code - parameters are now saved to the task and no longer passed to `_runTask`. Generating documentation is one of my favourite functions.

please generate documentation for the function

```

/**
 * Runs the next task in the queue, with a delay between tasks.
 *
 * @private
 * @returns {Promise<void>} - A Promise that resolves when all tasks are
 * completed.
 */
async _runTasks() {
  // implementation details
}

```

Unit tests

Writing good unit tests is a lot of work. ChatGPT helps here too. I have already found errors in my own code through test cases generated by Chat GPT.

For good unit testing, functions or methods should perform individual tasks, get data via input parameters and return results as function results. "Internal data transfer" is difficult to test and requires a lot of mocking. For example, you should not write a method that reads from the database and then performs many

operations on the read data. It is best to separate this, one method only reads the data and the other gets it as a parameter and then transforms it into the desired format. This way you can test the transformation function without having to read a dataset from a database. But back to ChatGPT

Which public functions are in my class?

The public functions in your TaskQueue class are:

on(event, callback): Sets a callback for a specific event. Supported events: 'task'. addTask(taskParameter): Adds a new task to the queue and starts processing it. All other functions are either private or internal to the class and not meant to be used by external code.

Please generate test cases for these public functions

```
const TaskQueue = require('./TaskQueue')

describe('TaskQueue', () => {
  describe('addTask()', () => {
    it('should add task to the queue', () => {
      const taskQueue = new TaskQueue(10)
      const taskParameter = 'foo'
      taskQueue.addTask(taskParameter)
      expect(taskQueue._taskQueue).toEqual([taskParameter])
    })

    it('should process task after delay', async () => {
      const taskQueue = new TaskQueue(10)
      const taskParameter = 'foo'
      const taskMock = jest.fn()
      taskQueue.on('task', taskMock)
      taskQueue.addTask(taskParameter)
      await new Promise(resolve => setTimeout(resolve, 20))
      expect(taskMock).toHaveBeenCalled()
    })
  })

  describe('on()', () => {
    it('should add a callback to the specified event', () => {
      const taskQueue = new TaskQueue(10)
      const taskMock = jest.fn()
      taskQueue.on('task', taskMock)
      expect(taskQueue._callbacks._callbacks.task).toEqual([taskMock])
    })
  })
})
```

This is what Chat GPT usually does: it generates Jest code for JavaScript. For those who don't know: Jest is a test framework and quite popular. At some point I decided to separate test data and test code. Test-data is kept in a test-object. I defined the format:

```
[
  {
    description: ''
    setup:
    tests: [
      {
        description:
        setup:
        expected:
      }
    ]
  }
]
```

In setup I specify e.g. mock data, expected contains the expected result. For many test cases, I am very satisfied with this method. I can add new test cases at any time without always having to read up on the syntax of the object under test. GPT can easily be made to adapt to your own formats - but then it is often not as good as Jest ...

I use a JavaScript format to describe test cases. I will give you the format and then ask you to generate test cases ok?

Sure, please provide me with the format and I will generate the test cases for you.

The above format...

Sure, I can use that format to generate test cases. Just let me know which class or functions you want me to create test cases for.

Generate test cases for addTask

```
[
  {
    description: 'Adds a single task to the queue',
    setup: () => {
      const taskQueue = new TaskQueue(1000)
      return { taskQueue }
    },
    tests: [
      {
        description: 'The task is added to the queue',
        setup: ({ taskQueue }) => {
          taskQueue.addTask(42)
          return { taskQueue }
        },
        expected: ({ taskQueue }) => {
          assert.deepStrictEqual(taskQueue._taskQueue, [42])
        }
      },
      {

```

```

        description: 'The task is executed after the delay',
        setup: ({ taskQueue }) => {
            const promise = taskQueue.addTask(42)
            return { taskQueue, promise }
        },
        expected: async ({ taskQueue }) => {
            await new Promise(resolve => setTimeout(resolve, 1500))
            assert.deepStrictEqual(taskQueue._taskQueue, [])
        }
    }
]
},
{
    description: 'Adds multiple tasks to the queue',
    setup: () => {
        const taskQueue = new TaskQueue(1000)
        return { taskQueue }
    },
    tests: [
        {
            description: 'The tasks are added to the queue in the correct
order',
            setup: ({ taskQueue }) => {
                taskQueue.addTask(1)
                taskQueue.addTask(2)
                taskQueue.addTask(3)
                return { taskQueue }
            },
            expected: ({ taskQueue }) => {
                assert.deepStrictEqual(taskQueue._taskQueue, [1, 2, 3])
            }
        },
        {
            description: 'The tasks are executed in the correct order',
            setup: ({ taskQueue }) => {
                const promise1 = taskQueue.addTask(1)
                const promise2 = taskQueue.addTask(2)
                const promise3 = taskQueue.addTask(3)
                return { taskQueue, promise1, promise2, promise3 }
            },
            expected: async ({ taskQueue }) => {
                await new Promise(resolve => setTimeout(resolve, 3500))
                assert.deepStrictEqual(taskQueue._taskQueue, [])
            }
        }
    ]
}
]

```

He took my format, still he did not separate code and data

Please separate code and data using my format for the data (I needed to press "Regenerate Response" to get the desired result)

Certainly! Here's an example test suite with separated code and data using the format you provided:

...

I skip the results that are always a combination of test data and code. He even placed the code in a JSON format ("please generate JSON format") in a string...

He will always include code, with the "setup" in the format. I tried a different format without seupt:

I have a new format for test data, please use it now:

```
[
  {
    description:
    functionName:
    tests: [
      {
        description:
        parameter:
        expected:
      }
    ]
  }
]
```

Sure! Here's an example test data in the format you provided:

```
[
  {
    description: "Test addTask() with single task",
    functionName: "addTask",
    tests: [
      {
        description: "Should add and execute a single task",
        parameter: "task1",
        expected: 1,
      },
    ],
  },
  {
    description: "Test addTask() with multiple tasks",
    functionName: "addTask",
    tests: [
      {
        description: "Should add and execute all tasks in the correct order",
        parameter: ["task1", "task2", "task3"],
        expected: 3,
      },
    ],
  },
]
```

```

    ],
  },
  {
    description: "Test addTask() with delay",
    functionName: "addTask",
    tests: [
      {
        description: "Should add and execute tasks with delay between them",
        parameter: ["task1", "task2", "task3"],
        expected: 3,
        delay: 1000,
      },
    ],
  },
},
{
  description: "Test on() with 'task' event",
  functionName: "on",
  tests: [
    {
      description:
        "Should register a callback function for the 'task' event",
      parameter: ["task", () => {}],
      expected: true,
    },
  ],
},
{
  description: "Test _runTasks() with single task",
  functionName: "_runTasks",
  tests: [
    {
      description: "Should execute a single task",
      setup: {
        taskQueue: ["task1"],
        running: false,
      },
      expected: { taskQueue: [], running: false },
    },
  ],
},
{
  description: "Test _runTasks() with multiple tasks",
  functionName: "_runTasks",
  tests: [
    {
      description: "Should execute all tasks in the correct order",
      setup: {
        taskQueue: ["task1", "task2", "task3"],
        running: false,
      },
      expected: { taskQueue: [], running: false },
    },
  ],
},
},

```

```
{
  description: "Test _runTasks() with delay",
  functionName: "_runTasks",
  tests: [
    {
      description: "Should execute tasks with delay between them",
      setup: {
        taskQueue: ["task1", "task2", "task3"],
        running: false,
      },
      expected: { taskQueue: [], running: false },
      delay: 1000,
    },
  ],
},
]
```

It is simpler, if you like to use Jest. My point here is not to provide a better idea than using Jest, but to show you that you can convince ChatGPT to generate what you need. And he will remember it - even tomorrow. Sometimes you need to remember him, that he should your format.