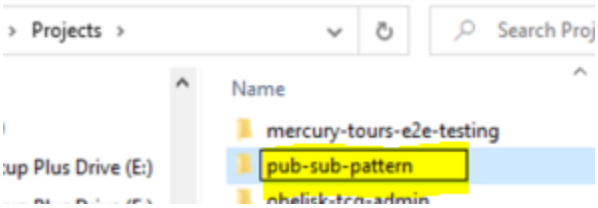
Table of Contents

Design Pattern Publisher-Subscriber

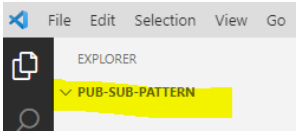
Aug 27 Lecture Project: pub-sub-pattern

Create a new folder in your project folder

This is Not an Angular Project.
This is a plain JavaScript project.



Open the folder in VS Code

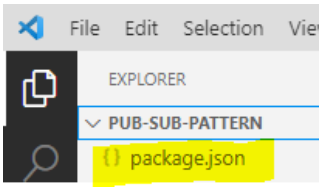


Open Git Bash window in the project folder

Type and press enter

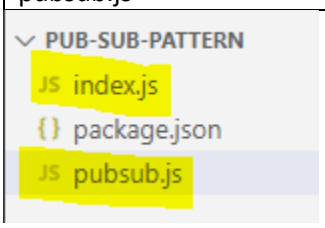
```
npm init

package name: (pub-sub-pattern) [Press enter accept default]
version: (1.0.0) [Press enter accept default]
description: A project to demonstrate the pub-sub design pattern [Add, Press enter]
entry point: (index.js) [Press enter accept default]
test command: [Press enter accept default]
git repository: [Press enter accept default]
keywords: [Press enter accept default]
author: <Enter your name> [Press Enter]
license: (ISC) [Press enter accept default]
About to write to C:\Users\tlw8748253\Desktop\Projects\pub-sub-pattern\package.json
:

{
  "name": "pub-sub-pattern",
  "version": "1.0.0",
  "description": "A project to demonstrate the pub-sub design pattern",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Tom Weikel",
  "license": "ISC"
}

Is this OK? (yes) yes [Type yes, Press Enter]
```

File created: package.json



Create project files

```
index.js
pubsub.js
```



Create shell code

File: pubsub.js

Create a function and export it.

```
module.exports.pubSub = pubSub;

function pubSub() {
   console.log('inside the pubSub() function')
   }
```

File: index.js

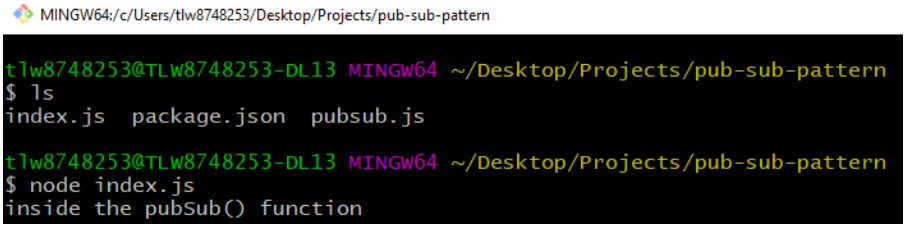
Import the function from pubsub.js

```
const { pubSub } = require('./pubsub');

pubSub();
```

Test the basic program

In the Git Bash window

```
node index.js
```



Update the shells

The updates in this section are the final project updates. The lecture went through various iteration to get to this point.

Update file: pubsub.js

Change the code to the following

```
module.exports.pubSub = pubSub;

function pubSub() {

  const subscribers = {}; // the subscribers object will contain properties whose values are arrays
  // each property is the channelName, and the array is composed of subscriber elements (callbacks)

  function publish(channelName, data) {
    if(!Array.isArray(subscribers[channelName])) {
      return;
    }

    subscribers[channelName].forEach((subscriber) => {
      subscriber(data);
    });
  }

  function subscribe(channelName, subscriber) {
    if(!Array.isArray(subscribers[channelName])) {
      subscribers[channelName] = [];
    }

    subscribers[channelName].push(subscriber);
  }

  return { publish, subscribe };
};
```

Update file: index.js

Change the code to the following

```
const { pubSub } = require('./pubsub');

const { publish, subscribe } = pubSub();

subscribe('weather', (data) => {
  console.log(data);
});

subscribe('weather', (data) => {
  console.log(data);
});

subscribe('weather', (data) => {
  console.log(data);
});

subscribe('investing', (data) => {
  console.log(data);
});

subscribe('investing', (data) => {
  console.log(data);
});

publish('weather', 'Today is cloudy with a 50% chance of rain');
publish('investing', 'The SP500 is up 1% today');
publish('someOtherChannel', 'some other data');
```

Run the final project

In the Git Bash window

```
node index.js
```

```
MINGW64:/c/Users/tlw8748253/Desktop/Projects/pub-sub-pattern

tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/pub-sub-pattern
$ node index.js
Today is cloudy with a 50% chance of rain
Today is cloudy with a 50% chance of rain
Today is cloudy with a 50% chance of rain
The SP500 is up 1% today
The SP500 is up 1% today
```

This demonstrates the publish code lines in index.js pushing the information and the subscribers catching the data for display to the console.

For more information:
[Appendix: Pub/Sub design pattern]

Appendix: Pub/Sub design pattern

From the training calendar
https://app.revature.com/curriculum/8105/batch/994/viewCalendar

Pub/Sub design pattern

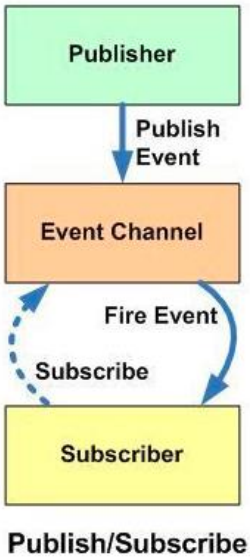
Publisher/Subscriber Design Pattern

The Publisher/Subscriber design pattern describes the flow of messages between applications, devices, or services.

A message is published by **Publishers** to a **Channel** that will be consumed by all **Subscribers** monitoring that channel.

When the Publisher pushes messages to a channel (live-feed data streams), the subscribers who subscribed to this channel are immediately notified. Any publisher may also be a subscriber. Messages can be text, sensor data, audio, video, or other digital content.

The Pub-Sub pattern is usually implemented in an asynchronous way. In the Observer pattern the observers are aware of the observable, but in Pub-Sub pattern, publishers and subscribers don't need to know each other. They simply communicate with the help of message queues.



Example - Pub-Sub pattern

```
class PubSub {
  constructor() {
    this.handlers = [];
  }
//publisher publishes the topic to the channel
  publish(event, args) {
    this.handlers.forEach(topic => {
      if (topic.event === event) {
        topic.handler(args)
      }
```

```
    })
  }

// subscriber gets notifications when there is a new feed in the subscribed channel
  subscribe(event, handler, context) {
    if (typeof context === 'undefined') { context = handler; }
    this.handlers.push({ event: event, handler: handler.bind(context) });
  }
}

export default new PubSub();
```