

Node JS-2

URL module

The URL module contains functions that help in parsing a URL. In other words, we can split the different parts of a URL easily with the help of utilities provided by the URL module.

The syntax for including the url module in your application:

```
var url=require("url");
```

Parse an address with the **url.parse() method**, and it will return a URL object with each part of the address as properties.

url.parse() – This method takes the url string as a parameter and parses it. The url module returns an object with each part of the url as property of the object.

Syntax of url.parse() :

```
url.parse(url_string, parse_query_string, slashes_host)
```

Example: Localhost as URL

```
var u=require("url");
varaddr="http://localhost:8080/default.htm?year=2017&month=february";
var q1=u.parse(addr,true);
console.log(q1);
console.log(q1.host);
console.log(q1.pathname);
console.log(q1.query);
```

Output:

```
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:8080',
  port: '8080',
  hash: null,
  search: '?year=2023&month=may',
  query: [Object: null prototype] { year: '2023', month:
'may' }, pathname: '/default.htm',
  path: '/default.htm?year=2023&month=may',
  href:
'http://localhost:8080/default.htm?year=2023&month=may'
}
localhost:8080
/default.htm
[Object: null prototype] { year: '2023', month: 'may' }
```

Example: Google Search Link as URL

```
var u=require("url");
var adr1="https://www.google.com/search?q=good+morning";
var q=u.parse(adr1,true); //query will be given as JSON Object
console.log(q);
```

Output:

```
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'www.google.com',
  port: null,
  hostname: 'www.google.com',
  hash: null,
  search: '?q=good+morning',
  query: [Object: null prototype] { q: 'good morning' },
  pathname: '/search',
  path: '/search?q=good+morning',
  href: 'https://www.google.com/search?q=good+morning'
}
```

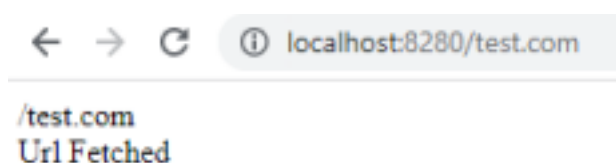
Read the URL

The function passed into the `http.createServer()` has a `req` argument that represents the request from the client, as an object (`http.IncomingMessage` object).

This object has a property called `"url"` which holds the part of the url that comes after the domain name:

Example:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8280);
```



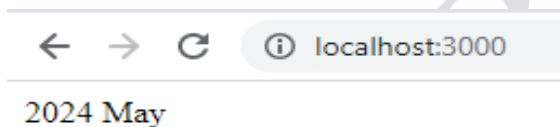
Query string

Get the details from query string

We can fetch the values from url query string as mentioned below using URL module. Add static url in code and request server to display data of query string

Example:

```
var http = require('http');
var url = require('url');
var addr="http://localhost:8080/default.html?year=2024&month=feb";
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  /*Use the url module to get the querystring*/
  var q = url.parse(addr, true).query;
  /*Return the year and month from the query object: */
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(3000);
```



Task: Find a leap year from static url

```
var u=require("url");
var addr="http://localhost:8080/default.html?year=2026&month=FEB";
var q=u.parse(addr,true);
console.log(q);
var qdata=q.query;
console.log(qdata.year);
if(qdata.year%4==0)
{
  console.log("Its a leap year") }
else{ console.log("Its not a leap year") }
```

OUTPUT:

```
PS D:\SEM-4\fsd-2\node> node q1.js
```

```
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:8080',
  port: '8080',
  hostname: 'localhost',
  hash: null,
  search: '?year=2026&month=FEB',
  query: [Object: null prototype] { year: '2026', month: 'FEB' },
  pathname: '/default.html',
  path: '/default.html?year=2026&month=FEB',
  href: 'http://localhost:8080/default.html?year=2026&month=FEB'
}
2026
Its not a leap year
```

Task: Write a nodejs script to print query string of url in file using ES6 callback.

Fetch query as an object. Need to convert query to string using stringy

```
var u=require("url");
var ps=require("fs");
var adr1=" http://localhost:8080/default.html?year=2023&month=may";
var q1=u.parse(adr1,true);
var qdata=JSON.stringify(q1.query);
ps.writeFile("fsd2.txt",qdata,(err)=>
{ console.log("completed"); });
```

Example: Fetch query as a string. We can directly write query string in a file.

```
var u=require("url");
```

```
var ps=require("fs");
var adr1=" http://localhost:8080/default.html?year=2025&month=feb";
var q1=u.parse(adr1, false);
var qdata=q1.query;
ps.writeFile("fsd2.txt",qdata,(err)=>
{
console.log("completed");
});
```

Task: Write a nodejs program load a simple html file on nodejs web server and print its content as html content.

URL2.js

```
var url = require("url");
var http = require('http');
var fs = require('fs');
var server = http.createServer(function(req, res) {
    const urlOBJ = url.parse(req.url, true);
    const fileName = "." + urlOBJ.pathname;
    fs.readFile(fileName, function(err, data) {
        if (err){
            res.write("404: File not found");
            res.end();
        }
    else{
        res.writeHead(200, { "content-type": "text/html" });
        // res.writeHead(200, { "content-type": "image/jfif" });
        res.write(data);
        res.end(); }
    });
}); server.listen(700);
```

```
console.log("Server running at http://localhost:700/");
```

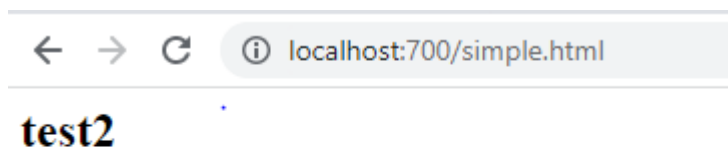
simple.html

```
<html>
  <body>
    <h2>test2</h2>
  </body>
</html>
```

OUTPUT

```
PS D:\SEM-4\fsd-2\node> node url2.js
```

Server running at http://localhost:700/



NPM (Node Package Manager)

What is a Package?

A package in Node.js contains all the files you need for a module. Modules are JavaScript libraries you can include in your project.

Installing NPM:

To install NPM, it is required to install Node.js as NPM gets installed with Node.js automatically.

1)Node.js Chalk Module

Chalk module in Node.js is the third-party module that is used for styling the format of text and allows us to create our own themes in the node.js project.

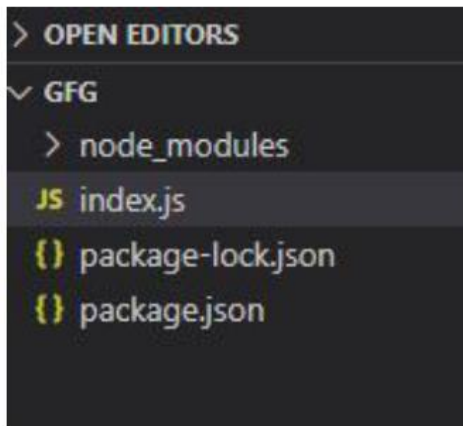
Advantages of Chalk Module:

1. It helps to customize the color of the output of the command-line output
2. It helps to improve the quality of the output by providing several color options like for warning message red color and many more

Installing Module

npm install chalk

Project Structure:



in package.json add "type": "module" and save.

Example

```
import ch from 'chalk';  
//var ch=require("chalk");  
console.log(ch.green("123"));  
console.log(ch.blue("hello"));  
console.log(ch.bgRed("Thanks"));  
console.log(ch.bold("Thanks"));  
console.log(ch.blue("hello","123","hcdoi"));  
console.log(ch.bgRed("Thanks")+"world"+ch.red("hii"));  
console.log(ch.red("hello",ch.underline.bgRed("world")));
```

```
PS D:\SEM-4\fsd-2\node> node chalkm.js  
123  
hello  
Thanks  
Thanks  
hello 123 hcdoi  
Thanksworldhii  
hello world
```

2) Node.js Validator Module

The Validator module is popular for validation. Validation is necessary to check whether the data is correct or not, so this module is easy to use and validates data quickly and easily.

Feature of validator module:

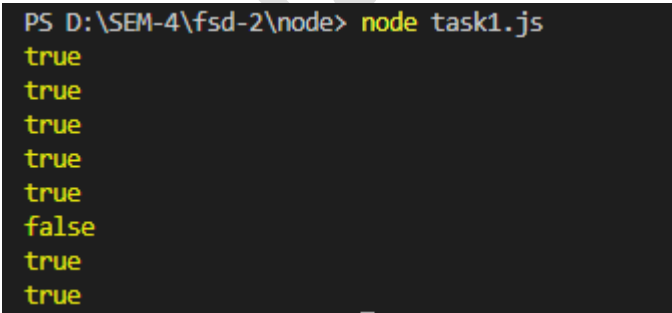
- It is easy to get started and easy to use.
- It is a widely used and popular module for validation.
- Simple functions for validation like `isEmail()`, `isEmpty()`, etc.

Installation of validator module:

```
npm install validator
```

Example

```
const v = require("validator");
console.log(v.isEmail("hello@gmail.com"));
console.log(v.isHexadecimal("ABC"));
console.log(v.isLowercase("xyz"));
console.log(v.isEmpty(""));
console.log(v.isCurrency("123"));
console.log(v.isCurrency("zmp"));
console.log(v.isDecimal("1.22"));
console.log(v.isJSON('{ "name1": "ABC", "age": 30 }'));
```



```
PS D:\SEM-4\fsd-2\node> node task1.js
true
true
true
true
true
false
true
true
```


How to create, export and use our own modules

Own Module

The node.js modules are a kind of package that contains certain functions or methods to be used by those who imports them. Some modules are present on the web to be used by developers such as fs, path, http, url etc. You can also make a package of your own and use it in your code.

Syntax:

```
exports.function_name = function(arg1, arg2, ....argN) {  
  
  // function body  
  
};
```

Example - Create two file with name – calc.js and index.js and copy the below code snippet. The calc.js is the custom node module which will hold the node functions. The index.js will import calc.js and use it in the node process.

Method 1

In 1.js file:

```
const add=(a,b)=>  
{  
  return(a+b);  
}  
module.exports=add;
```

In 2.js file:

```
var d=require("./1");  
console.log(d(10,15));
```

Method 2

In 1.js file:

```
const sub=(a,b)=>  
{  
  return(a-b);  
}  
const mul=(a,b)=>  
{  
  return(a*b);  
}  
module.exports.s=sub;  
module.exports.m=mul;
```

In 2.js file:

```
var d1=require("./1");  
console.log(d1.s(10,5));  
console.log(d1.m(10,15));
```

Method 3**In 1.js file:**

```
const sub=(a,b)=>  
{  
  return(a-b);  
}  
const mul=(a,b)=>  
{  
  return(a*b);  
}  
module.exports.d2=sub;  
module.exports.e2=mul;
```

In 2.js file:

```
var {d2,e2}=require("./1");  
console.log(d2(10,7));  
console.log(e2(10,12));
```

Method 4**In 1.js file:**

```
const sub=(a,b)=>  
{  
  return(a-b);  
}  
const mul=(a,b)=>  
{  
  return(a*b);  
}  
const name="Hello"  
module.exports={sub,mul,name};
```

In 2.js file:

```
Var{sub,mul,name}=require("./1");  
console.log(sub(100,20));  
  
console.log(mul(10,2));  
  
console.log(name)
```

Task: Write a node.js script to create calculator using external module having a function add(), sub(), mul(), div(). This function returns result of calculation. Write all necessary .js files.

1.js

```
exports.add = function (x, y) {  
  return x + y;  
};  
exports.sub = function (x, y) {  
  return x - y;  
};  
exports.mult = function (x, y) {  
  return x * y;  
};  
exports.div = function (x, y) {  
  return x / y;  
};
```

2.js

```
const calculator = require('./1.js');  
let x = 50, y = 20;  
console.log("Addition of 50 and 20 is " + calculator.add(x, y));  
console.log("Subtraction of 50 and 20 is " + calculator.sub(x, y));  
console.log("Multiplication of 50 and 20 is " + calculator.mult(x, y));  
console.log("Division of 50 and 20 is " + calculator.div(x, y));
```

Output:

```
Addition of 50 and 20 is 70  
Subtraction of 50 and 20 is 30  
Multiplication of 50 and 20 is 1000  
Division of 50 and 20 is 2.5
```

Task: Write all necessary .js files to create module having a function to check numbers from 2 to 50 are prime number or not.

1.js

```
const PrimeNo = (num) =>  
{  
  let temp = 0  
  for(let i=2;i<num;i++)  
  {  
    if(num%i==0)  
    { temp++;  
    } }  
  if(temp==0)  
  {
```

```

return true;
}
else{
return false;
} }
module.exports=PrimeNo;

```

2.js

```

var PrimeNumber = require("./1.js")
for(i=2;i<=50;i++)
{
let x=PrimeNumber(i);
if(x==true)
{
console.log(i+" Prime Number");
}
else{
console.log(i+" Not a Prime Number")
} }

```

Task: Write a nodejs script to create my own module to calculate reverse of a given number. That module should be used to compute all numbers between 1 to 100 in which square of reverse and reverse of square is same. These has call of reverse twice so call it from module.

1.js

```

function reversenum(num)
{
let rev=0;
while(num>0)
{
rev=rev*10+(num%10);
num=parseInt(num/10);
}
return rev;
}
function square(num1)
{
return num1*num1;
}
function checknum(num2)
{
a=square(num2);
b=square(reversenum(num2));
if(a==reversenum(b))
{

```

```
console.log("eEqual")
}  
else  
{  
console.log("Not equal")  
}  
}  
module.exports.cs = checksum
```

In another file:

```
var c1=require("./1.js");  
c1.cs(12);
```

Output: Equal

Events

According to the official documentation of Node.js, it is an asynchronous event-driven JavaScript runtime. Node.js has an **event-driven architecture** which can perform asynchronous tasks. Node.js has '**events**' module which emits named events that can cause corresponding functions or callbacks to be called.

Functions(Callbacks) listen or subscribe to a particular event to occur and when that event triggers, all the callbacks subscribed to that event are fired one by one in order to which they were registered.

1) EventEmitter.on("event-name", callback):

This function registers an event with its handler(listener) and calls the callback function once event is emitted.

2) EventEmitterObj.emit("event-name"):

Triggers an event with a specific string name.

Steps: -

- 1) Import "events" module
`Var e=require("events");`
- 2) Create EventEmitter object
`Var ee=new e.EventEmitter();`
- 3) Define event listeners using the **on** method.
`ee.on('eventName', () => {
 // Event handler code
});`
- 4) Emit /fire an event to trigger the listeners.
`ee.emit('eventName');`

Example.

```
var e=require("events");  
var ee=new e.EventEmitter();  
ee.on("sayName",()=>{  
  console.log("your name is xyz")  
});  
ee.emit("sayName");
```

The code snippet you provided is an example of using the **events** module in Node.js to create and emit events.

In this code, you first require the **events** module by assigning it to the variable **e**. Then, you create a new instance of the **EventEmitter** class and assign it to the variable **ee**.

Next, you define an event listener for the event named **"sayName"**. When this event is emitted, the listener will be triggered, and it will log the message **"your name is xyz"** to the console.

Finally, you emit the event **"sayName"** by calling **ee.emit("sayName")**. This will trigger the event listener, and you will see the corresponding message printed in the console: **"your name is xyz"**.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.emit("sayName");
ee.on("sayName",()=>{
    console.log("your name is xyz")
});
```

In this case, when you emit the event before defining the listener, there are no registered listeners yet, so nothing will happen.

Example: create an event emitter instance and register a couple of call-back.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.on("sayName",()=>{
    console.log("your name is Zalak")
});
ee.on("sayName",()=>{
    console.log("your name is M")
});
ee.on("sayName",()=>{
    console.log("your name is Prajapati")
});
ee.emit("sayName");
```

o/p

PS D:\SEM-4\fsd-2\node> node eventhandle.js

your name is Zalak

your name is M

your name is Prajapati

OR

```
var e=require("events");
var ee=new e.EventEmitter();
//ee.emit("sayName");
ee.on("sayName",()=>{
    console.log("your name is Zalak")
    ee.emit("xyz");
});
ee.on("xyz",()=>{
    console.log("your name is M")
    ee.emit("abc");
});
ee.on("abc",()=>{
    console.log("your name is Prajapati")
});
ee.emit("sayName");
```

o/p

PS D:\SEM-4\fsd-2\node> node eventhandle.js

your name is Zalak

your name is M

your name is Prajapati

1. Require the **events** module and create a new instance of **EventEmitter**.
2. Commented out the line **ee.emit("sayName");**, so it won't be executed.
3. Define an event listener for the event named **"sayName"**. When this event is emitted, it will log the message **"your name is Zalak"** to the console and emit the event **"xyz"**.
4. Define an event listener for the event named **"xyz"**. When this event is emitted, it will log the message **"your name is M"** to the console and emit the event **"abc"**.
5. Define an event listener for the event named **"abc"**. When this event is emitted, it will log the message **"your name is Prajapati"** to the console.
6. Emit the event **"sayName"** by calling **ee.emit("sayName")**.
7. The **"sayName"** event listener is triggered, logging **"your name is Zalak"** to the console, and emitting the **"xyz"** event.
8. The **"xyz"** event listener is triggered, logging **"your name is M"** to the console, and emitting the **"abc"** event.
9. The **"abc"** event listener is triggered, logging **"your name is Prajapati"** to the console.

Example: registering for the event with call-back parameter.

```
var e=require("events");
var ee=new e.EventEmitter();
ee.on("sayName",(statusCode,msg)=>{
  console.log(`status code id ${statusCode} and page is ${msg}`);
});
ee.emit("sayName",200,"ok");
o/p
```

PS D:\SEM-4\fsd-2\node> node eventhandle2.js
status code id 200 and page is ok

Removing Listener:

The `eventEmitter.removeListener()` takes two argument event and listener, and removes that listener from the listeners array that is subscribed to that event. While `eventEmitter.removeAllListeners()` removes all the listener from the array which are subscribed to the mentioned event.

Syntax:

```
eventEmitter.removeListener(event, listener)
eventEmitter.removeAllListeners([event])
```

Note:

- ☐ Removing the listener from the array will change the sequence of the listener's array, hence it must be carefully used.
- ☐ The `eventEmitter.removeListener()` will remove at most one instance of the listener which is in front of the queue.

Task-1: write a nodejs script to create two listeners for common event.call their respective callbacks print no of events accociated with an emitter.remove one of the listener and call remaining listener again also print no of remaining listener

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
const listener1 = () => console.log('First listener called');
const listener2 = () => console.log('Second listener called');
emitter.on('event', listener1);
emitter.on('event', listener2);

console.log(`Number of events associated with the emitter:
${emitter.listenerCount('event')}`);

emitter.removeListener('event', listener1);

emitter.emit('event');

console.log(`Number of remaining listeners: ${emitter.listenerCount('event')}`);
```

OR

```
const EventEmitter = require('events');  
// Create an emitter instance  
const emitter = new EventEmitter();  
  
// Define the callbacks for the listeners  
const listener1 = () => {  
  console.log('Listener 1 called');  
};  
const listener2 = () => {  
  console.log('Listener 2 called');  
};  
// Attach the listeners to the emitter  
emitter.on('commonEvent', listener1);  
emitter.on('commonEvent', listener2);  
// Get the number of listeners for the 'commonEvent' event  
const eventListenersCount = emitter.listenerCount('commonEvent');  
console.log(`Number of event listeners: ${eventListenersCount}`);  
// Emit the 'commonEvent' event, triggering the callbacks  
emitter.emit('commonEvent');  
// Remove one of the listeners  
emitter.off('commonEvent', listener1);  
// Get the number of remaining listeners for the 'commonEvent' event  
const remainingListenersCount = emitter.listenerCount('commonEvent');  
console.log(`Number of remaining listeners: ${remainingListenersCount}`);  
// Emit the 'commonEvent' event again, triggering the remaining callback  
emitter.emit('commonEvent');
```

Task-2 Write a node js script to write the text “This is data” to new.txt file. After that append the text “that is data” to same ne .txt file. After that read the file and print file concept on console. After finishing read operation, print the line “Thanks for using my program” on console. All read/write operations are asynchronous.

```
const fs = require('fs');

const e = require('events');

const ee = new e.EventEmitter();

ee.on('write', () => {

  fs.writeFile("new.txt", 'this is data', () => {

    console.log('Successfully wrote to file.');
```

Vaibhav Prajapati

```
    ee.emit('append');

  });

});

ee.on('append', () => {

  fs.appendFile("new.txt", '\nthat is data', () => {

    console.log('Successfully appended to file.');
```

Vaibhav Prajapati

```
    ee.emit('read');

  });

});

ee.on('read', () => {

  fs.readFile("new.txt", 'utf8', (err, data) => {

    console.log(data);

    ee.emit('finish');
```

Vaibhav Prajapati

```
  });

});

ee.on('finish', () => {

  console.log("Thanks for using my program.");

});

ee.emit('write');
```

Task: Write node js script to handle events as asked below.

1) Check the radius is negative or not. If negative then display message “Radius” must be positive” else calculate the perimeter of circle.

2) Check side is negative or not. If negative then display message “Side must be positive” else calculate the perimeter of square.

```
var e = require("events");

var ee = new e();

const radiushandler = () => {
  console.log("Radius must be positive");
}

ee.on("negside", sidehandler = () => {
  console.log("Side must be positive");
});

const findval = (r,s) => {
  if (r < 0) {
    ee.emit("negradius");
  }else{
    var rperi = 2 * 3.14 * r;
    console.log(rperi);
  }
  if (s < 0)
  {
    ee.emit("negside");
  }
  else{
    var speri = 4*s;
    console.log(speri);
  } }

ee.on("negradius", radiushandler);
//ee.on("negside", sidehandler);

ee.on("findval", findval);
ee.emit("findval",-2,-3);
```

JSON Processing in Node JS

If you have [JSON](#) data as part of a string, the best way to parse it is by using the `JSON.parse` method that's part of the JavaScript standard since ECMAScript 5, and it's provided by [V8](#), the JavaScript engine that powers [Node.js](#).

Task: Defining an array of objects with properties name and age. Write this object in a file named `student.txt` then read the file and display the object on console.

```
const student =
[
  {
    name: "ABC",
    age: 30
  },
  {
    name: "XYZ",
    age: 32
  }
]
var ps=require("fs");
ps.writeFileSync("student.txt",JSON.stringify(student));
data=ps.readFileSync("student.txt","utf-8");
b=JSON.parse(data);
console.log(b);
```

Task: Create JSON object which contains array of objects. Calculate perimeter of square and perimeter of circle by using side value and diameter value respectively.

```
const shape =
[
  {
    name: "circle",
    diameter: 8
  },
  {
    name: "square",
    side: 10
  }
]
var ps=require("fs");
ps.writeFileSync("shape.txt",JSON.stringify(shape));
data=ps.readFileSync("shape.txt","utf-8");
```

```

b=JSON.parse(data);
if( b[0].name == 'circle'){
    var perimeter = (b[0].diameter/2) * 3.14 * 2 ;
    console.log(perimeter);
}
if ( b[1].name == 'square'){
    var peri = (b[1].side) *4 ;
    console.log(peri);
}
ps.appendFileSync("shape.txt","\nPerimeter of circle = "+ JSON.stringify(perimeter)+
"\nPerimeter of square = "+JSON.stringify(peri));

```

Task: Write node.js script to create a class named person by assigning name and age in the form of members. Create one function named elder which returns an elder person object. Details of the elderly person should be printed in console as well as in file.

```

class person
{
    constructor(name,age)
    {
        this.age=age;
        this.name=name;
    }
    elder(P)
    {
        if(this.age>P.age)
        {
            return this;
        }
        else{
            return P;
        }
    }
}
var p1= new person("xyz",23);
var p2= new person("abc",34);
var p3=p1.elder(p2);
const jsonstr=JSON.stringify(p3);
var ps=require("fs");
ps.writeFileSync("d2.txt",jsonstr);
console.log(jsonstr);

```