**Node.js Upload Files**

**The Formidable Module**

There is a very good module for working with file uploads, called "Formidable".

The Formidable module can be downloaded and installed using NPM:

**C:\Users\Your Name>npm install formidable**

After you have downloaded the Formidable module, you can include the module in any application:

**var formidable = require('formidable');**

**Upload Files**

Now you are ready to make a web page in Node.js that lets the user upload files to your computer:

**Step 1: Create an Upload Form**

Create a Node.js file that writes an HTML form, with an upload field:

Example

This code will produce an HTML form:

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

res.write('<input type="file" name="filetoupload"><br>');

res.write('<input type="submit">');

res.write('</form>');

return res.end();

}).listen(8080);

**Step 2: Parse the Uploaded File**

Include the Formidable module to be able to parse the uploaded file once it reaches the server.

When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

**Example**

The file will be uploaded, and placed on a temporary folder:

var http = require('http');

var formidable = require('formidable');

http.createServer(function (req, res) {

if (req.url == '/fileupload') {

var form = new formidable.IncomingForm();

form.parse(req, function (err, fields, files) {

res.write('File uploaded');

res.end();

});

} else {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

res.write('<input type="file" name="filetoupload"><br>');

res.write('<input type="submit">');

res.write('</form>');

return res.end();

}

}).listen(8080);

**Step 3: Save the File**

When a file is successfully uploaded to the server, it is placed on a temporary folder.

The path to this directory can be found in the "files" object, passed as the third argument in the parse() method's callback function.

To move the file to the folder of your choice, use the File System module, and rename the file:

**Example**

Include the fs module, and move the file to the current folder:

var http = require('http');

var formidable = require('formidable');

var fs = require('fs');

http.createServer(function (req, res) {

if (req.url == '/fileupload') {

var form = new formidable.IncomingForm();

form.parse(req, function (err, fields, files) {

var oldpath = files.filetoupload.filepath;

var newpath = 'C:/Users/Your Name/' + files.filetoupload.originalFilename;

fs.rename(oldpath, newpath, function (err) {

if (err) throw err;

res.write('File uploaded and moved!');

res.end();

});

});

} else {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');

res.write('<input type="file" name="filetoupload"><br>');

res.write('<input type="submit">');

res.write('</form>');

return res.end();

}

}).listen(8080);

**Node.js Send an Email**

**The Nodemailer Module**

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm:

**C:\Users\Your Name>npm install nodemailer**

After you have downloaded the Nodemailer module, you can include the module in any application:

**var nodemailer = require('nodemailer');**

**Send an Email**

Now you are ready to send emails from your server.

Use the username and password from your selected email provider to send an email. This tutorial will show you how to use your Gmail account to send an email:

**Example**

var nodemailer = require('nodemailer');

var transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: 'youremail@gmail.com',

pass: 'yourpassword'

}

});

var mailOptions = {

from: 'youremail@gmail.com',

to: 'myfriend@yahoo.com',

subject: 'Sending Email using Node.js',

text: 'That was easy!'

};

transporter.sendMail(mailOptions, function(error, info){

if (error) {

console.log(error);

} else {

console.log('Email sent: ' + info.response);

}

});

And that's it! Now your server is able to send emails.

**Multiple Receivers**

To send an email to more than one receiver, add them to the "to" property of the mailOptions object, separated by commas:

**Example**

Send email to more than one address:

var mailOptions = {

from: 'youremail@gmail.com',

to: 'myfriend@yahoo.com, myotherfriend@yahoo.com',

subject: 'Sending Email using Node.js',

text: 'That was easy!'

}

**Send HTML**

To send HTML formatted text in your email, use the "html" property instead of the "text" property:

Example

Send email containing HTML:

var mailOptions = {

from: 'youremail@gmail.com',

to: 'myfriend@yahoo.com',

subject: 'Sending Email using Node.js',

html: '<h1>Welcome</h1><p>That was easy!</p>'

**What is Callback?**

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

**Blocking Code Example**

Create a text file named input.txt with the following content −

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

Create a js file named main.js with the following code −

var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());

console.log("Program Ended");

**Now run the main.js to see the result −**

$ node main.js

**Verify the Output.**

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

Program Ended

**Non-Blocking Code Example**

Create a text file named input.txt with the following content.

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

**Update main.js to have the following code −**

var fs = require("fs");

fs.readFile('input.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("Program Ended");

Now run the main.js to see the result −

$ node main.js

**Verify the Output.**

Program Ended

Tutorials Point is giving self learning content

to teach the world in simple and easy way!!!!!

These two examples explain the concept of blocking and non-blocking calls.

The first example shows that the program blocks until it reads the file and then only it proceeds to end the program.

The second example shows that the program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.

Thus, a blocking program executes very much in sequence. From the programming point of view, it is easier to implement the logic but non-blocking programs do not execute in sequence. In case a program needs to use any data to be processed, it should be kept within the same block to make it sequential execution.