

# **Minification and Uglification of Javascript/ AngularJs application using Grunt**

## **Contents:**

1. What is Grunt?
2. Prerequisites
3. Understanding Files and setup of the project
4. Integration with Jenkins

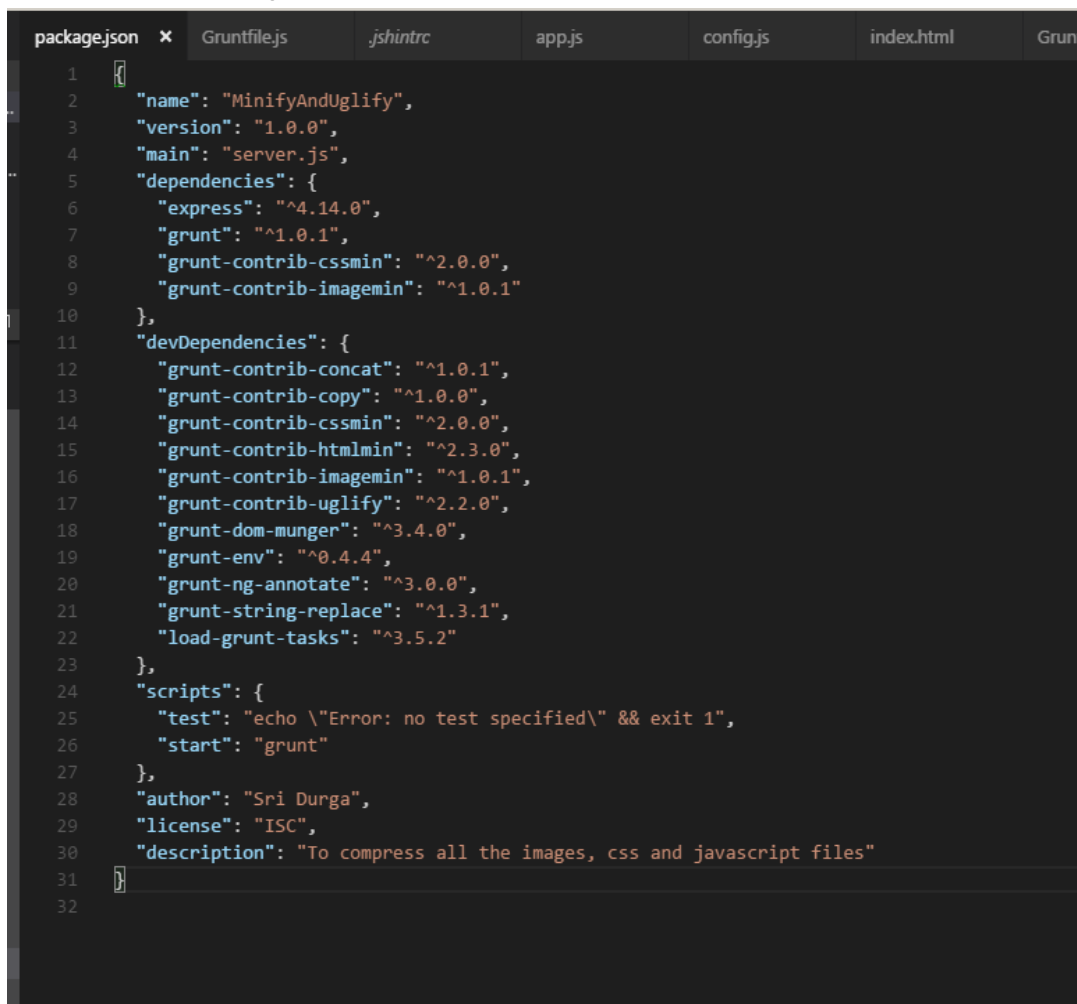
## What is Grunt?

It is JavaScript task runner useful for performing repetitive tasks like minification, compilation, unit testing, linting, etc.

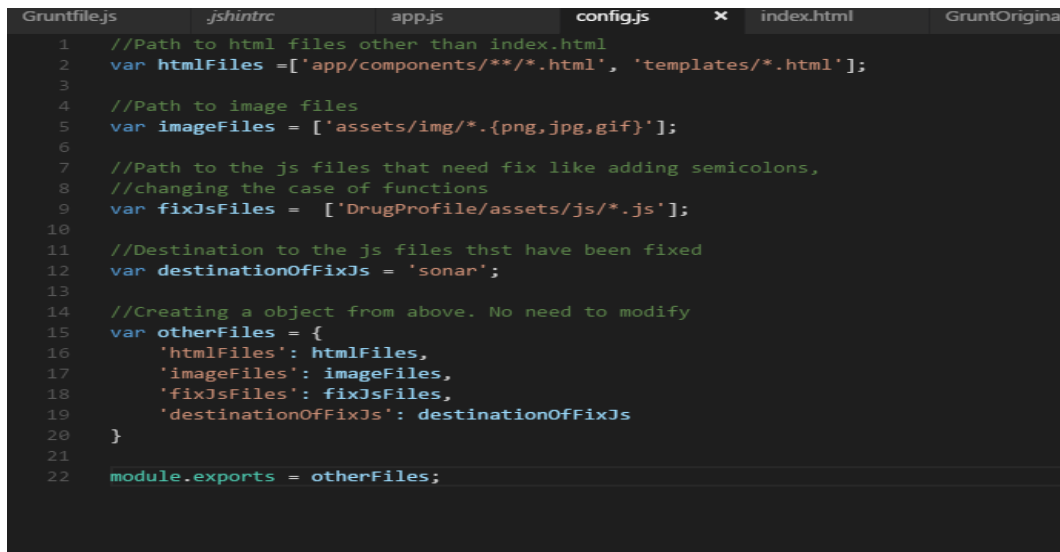
**Prerequisites:** Node version  $\geq 0.8.0$

### Understanding Files in the project:

1. **package.json:** It contains the names of the packages that needs to be installed. One can create a new package.json file by running the command “**npm init**”. The details like project name, description, author name, version etc need to be provided. The artifact package.json has the required dependencies that are needed for the project. To install the packages run the command “**npm install –save-dev**”.



```
package.json x Gruntfile.js .jshinttrc app.js config.js index.html Grunt
1 {
2   "name": "MinifyAndUglify",
3   "version": "1.0.0",
4   "main": "server.js",
5   "dependencies": {
6     "express": "^4.14.0",
7     "grunt": "^1.0.1",
8     "grunt-contrib-cssmin": "^2.0.0",
9     "grunt-contrib-imagemin": "^1.0.1"
10  },
11  "devDependencies": {
12    "grunt-contrib-concat": "^1.0.1",
13    "grunt-contrib-copy": "^1.0.0",
14    "grunt-contrib-cssmin": "^2.0.0",
15    "grunt-contrib-htmlmin": "^2.3.0",
16    "grunt-contrib-imagemin": "^1.0.1",
17    "grunt-contrib-uglify": "^2.2.0",
18    "grunt-dom-munger": "^3.4.0",
19    "grunt-env": "^0.4.4",
20    "grunt-ng-annotate": "^3.0.0",
21    "grunt-string-replace": "^1.3.1",
22    "load-grunt-tasks": "^3.5.2"
23  },
24  "scripts": {
25    "test": "echo \"Error: no test specified\" && exit 1",
26    "start": "grunt"
27  },
28  "author": "Sri Durga",
29  "license": "ISC",
30  "description": "To compress all the images, css and javascript files"
31 }
32
```



```
1 //Path to html files other than index.html
2 var htmlFiles = ['app/components/**/*.html', 'templates/*.html'];
3
4 //Path to image files
5 var imageFiles = ['assets/img/*.png,*.jpg,*.gif'];
6
7 //Path to the js files that need fix like adding semicolons,
8 //changing the case of functions
9 var fixJsFiles = ['DrugProfile/assets/js/*.js'];
10
11 //Destination to the js files thst have been fixed
12 var destinationOfFixJs = 'sonar';
13
14 //Creating a object from above. No need to modify
15 var otherFiles = {
16   'htmlFiles': htmlFiles,
17   'imageFiles': imageFiles,
18   'fixJsFiles': fixJsFiles,
19   'destinationOfFixJs': destinationOfFixJs
20 }
21
22 module.exports = otherFiles;
```

2. **config.js:** It contains the path details of html files and images. One must provide the location details to their html files and images. If you need to add semicolons in your js give the location to these files also otherwise they can be left unmodified.
3. **Gruntfile.js:** Let us take an example and see how we can minify css files using grunt and details that are provided in the gruntfile. To minify css files we need grunt-contrib-cssmin. “**npm install –save-dev**” installs this package also. If this command is used then one can skip the next command otherwise to install this use the command “**npm install grunt-contrib-cssmin –save-dev**”. The **–save-dev** is used to indicate it as a dev dependency. Running the command updates your package.json file.

A Gruntfile is comprised of the following parts:

- The "wrapper" function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks

### The "wrapper" function:

- Every Gruntfile (and gruntplugin) uses this basic format, and all of your Grunt code must be specified inside this function:

```
module.exports = function(grunt) {  
    // Do grunt-related things in here  
};
```

### Project and task configuration:

Most Grunt tasks rely on configuration data defined in an object passed to the **grunt.initConfig** method. `grunt.file.readJSON('package.json')` imports the JSON metadata stored in `package.json` into the grunt config.

```
module.exports = function (grunt) {  
    grunt.initConfig({  
        pkg: grunt.file.readJSON('package.json'),  
        cssmin: {  
            dist: {  
                files: {destination: [source]}  
            }  
        }  
    })  
    grunt.loadNpmTasks('grunt-contrib-cssmin');  
    grunt.registerTask('default', ['cssmin']);  
}
```

cssmin is the task name. dist is subtask name under cssmin. The order followed while writing in files is destination and source. The plugin that is required by your task must be loaded. `grunt.loadNpmTasks` command in the above loads the task plugin. The task must also be registered. In the above code cssmin is given in the default task. Now your gruntfile is ready to minify css, run the command **grunt** to start the process. There can be more than one subtask. “`grunt cssmin`” runs all the subtasks. To run a particular subtask one can use “`grunt cssmin subtask-name`”

If you get command not found you must set the path. Run this command  
**path=%PATH%;%AppData%\npm** to set the path

### **Custom tasks:**

If you have many tasks and you want to run only few tasks then use the following command

**grunt.registerTask('dom\_html',['dom\_munger','htmlmin',])**

In the above command dom\_html is the name of task followed by the tasks needed to be done. You can give your own name and run the command grunt followed by name of task like grunt dom\_html;

### **Artifact Gruntfile.js:**

Running grunt creates a folder name “dist”, which contains the minified js,css and html files. It also optimizes images. The contents inside the “dist” folder can be used for deployment purpose.

### **Details of tasks in the artifact gruntfile:**

The artifact default task reads the index.html to get location of javascript and css files using the plugin dom\_munger and writes the location to variables jsRefs and cssRefs. After this, script and link tags are removed and are replaced with that referencing to that of minified file. The new file that is created is placed in distInter folder

```

19
20 //Used to read the script files and css files from index.html
21
22 dom_munger: {
23   your_target: {
24     options: {
25       read: [
26         { selector: 'link', attribute: 'href', writeto: 'cssRefs' },
27         { selector: 'script', attribute: 'src', writeto: 'jsRefs' }
28       ],
29       remove: ['script', 'link'],
30       append: [
31         { selector: 'head', html: '<link href="css/app.full.min.css" rel="stylesheet">' },
32         { selector: 'body', html: '<script src="js/app.full.min.js"></script>' }
33       ]
34     },
35     src: 'index.html',
36     dest: 'distInter/index.html'
37   },
38 },
39

```

Now css files are minified by cssmin plugin. The dom\_munger.data.cssRefs variable gives the location to css files. The minified css file is placed in dist/css as given in the destination.

```

// Minifying the css files

cssmin: {
  main: {
    src: '<%= dom_munger.data.cssRefs %>',
    dest: 'dist/css/app.full.min.css'
  }
},

```

The JS files are concatenated, annotated and uglified. Uglify does not retain order so they are concatenated first. Annotate is used to make angular code min-safe. It changes the

```
form.factory('Users', function($http) {
```

to

```
form.factory('Users', ['$http', function($http) {
```

When UglifyJS runs, it will rename our parameters from \$scope and \$http to a and b respectively. The presence of DI annotations being passed in as strings in an array, blocks them from being renamed. Therefore, these renamed parameters can still access to the necessary dependencies

```
// Concats all the js files

concat: {
  js: {
    src: '<%= dom_munger.data.jsRefs %>',
    dest: 'distInter/built.js',
  },
},

//annoates all the files

ngAnnotate: {
  options: {
    singleQuotes: true
  },
  app: {
    files: { 'distInter/annoted.js': ['distInter/built.js'] }
  }
},
```

```
uglify: {
  dist: {
    options: {
      sourceMap: false,
      compress: {
        drop_console: false,
        hoist_funs: false
      },
      mangle: {
        except: ['angular']
      }
    },
    files: {
      'dist/js/app.full.min.js': ['distInter/annoted.js']
    }
  }
},
```

Htmlmin is used to minify the html files. There are two tasks, one minifies the index.html, and other minifies the remaining html files. The options given are to remove comments, collapse whitespace and to preserve line breaks. One can change them to true or false based on the need.



```
// Minifying the html files

htmlmin: {
  dist: {
    options: {
      removeComments: true,
      collapseWhitespace: true,
      preserveLineBreaks: true
    },
    files: {
      'dist/index.html': 'distInter/index.html',
    }
  },
  dev: {
    options: {
      removeComments: true,
      collapseWhitespace: true,
      preserveLineBreaks: true
    },
    files: [{
      expand: true,
      src: otherFiles.htmlFiles,
      dest: 'dist'
    }]
  }
},
},
```

Imagemin is used to optimize images. The optimization level used is 5. One can change the optimization level according to the need. Expand puts the images in the same folder structure as it was used in the project. Using this prevents from changing the location of images in the files.

```
imagemin: {
  dist: {
    options: {
      optimizationLevel: 5
    },
    files: [{
      expand: true,
      src: otherFiles.imageFiles,
      dest: 'dist'
    }]
  }
},
```

Any intermediate files like concatenated file etc are placed in distInter. As this folder is not useful, we can delete this folder. Clean plugin is used to delete the folder.

```
//To clean the intermediate folder

clean: {
  folder: ['distInter']
},
```

So the default grunt task runs the above all.

```
grunt.registerTask('default', ['dom_munger', 'imagemin', 'concat', 'ngAnnotate', 'cssmin', 'uglify', 'htmlmin', 'clean']);
```

Sometimes we might have fonts or you have files to which you do not want to make changes. Then they can be copied using the copy plugin. The syntax is provided in the artifact file. One must give the location of files and must include the task in the default task.

```
//Can be used to copy the files like fonts

copy: {
  main: {
    files: [
      { expand: true, src: ['assets/css/fonts/**'], dest: 'dist/assets/css/fonts', flatten: true, filter: 'isFile' },
    ]
  }
},
```

The artifact can also be used to improve your Javascript code. The task **fixmyjs** reads configuration from “**.jshintrc**” file. The configuration is set to fix semicolons. Running the command “**grunt sonar**” fixes all the semicolons and formats your code. The source and destination to these files must be provided in the **config.js** or one provide the path directly in the Gruntfile.

The Configuration in the **.jshintrc** file can be also changed to changes case of functions to camelcase, but one must be careful while using this. “**asi**” to true fixes semicolons. Case of functions can be changed to camelcase by setting “**camelcase**” to true. Other configuration details can be found on <https://github.com/jshint/fixmyjs>

```
Gruntfile.js  jshintrc  x
1  {
2    "asi": true,
3    "boss": true,
4    "camelcase": false,
5    "curly": true,
6    "eqeqeq": true,
7    "eqnull": true,
8    "esnext": true,
9    "immed": true,
10   "latedef": false,
11   "laxcomma": true,
12   "mocha": true,
13   "newcap": true,
14   "noarg": true,
15   "node": true,
16   "sub": true,
17   "undef": true,
18   "unused": true
19 }
```

```
fixmyjs: {
  options: {
    config: '.jshintrc',
    indentpref: 'tabs'
  },
  test: {
    files: [
      { expand: true, src: otherFiles.fixJsFiles, dest: otherFiles.destinationOfFixJs}
    ]
  }
},
```

## **Integration with Jenkins/looper:**

The contents inside dist folder can be directly used for deployment. Minification and deployment process can be directly done by looper/Jenkins. For this one must provide the tools required in their config file.

### **tools:**

#### **nodejs:**

**- 6.9.1**

#### **npm:**

**- 3.10.9**

In their flow to their deployment one can add the following steps

### **npm install**

### **gulp**

The above steps creates a folder dist which can be used for deployment. In the command to deploy one create a war file of the above folder

Example :

**mvn deploy:deploy-file -Dfile=dist/dist.war \**