Gulp

Quick guide to getting up and running *today*

@2015 by Robert B. Dunaway. All rights reserved.

ISBN: 9781514689608

Dedication

This book is dedicated to my soul mate and wife, Amy Dunaway. She has been a constant source of love, motivation, inspiration, and support.

Table of Contents

Contents

[Introduction 5](#_Toc423906616)

[Gulp Tutorials 6](#_Toc423906617)

[GULP Tutorial Part 1 – Reasons for Build Tools like Gulp 6](#_Toc423906618)

[Productivity 6](#_Toc423906619)

[GULP Tutorial Part 2 – Setup 7](#_Toc423906620)

[GULP Tutorial Part 3 – Adding Plugins 15](#_Toc423906621)

[GULP Tutorial Part 4 – Sequence and Parallel task processing 19](#_Toc423906622)

[GULP Tutorial Part 5 – Handling errors with Plumber 21](#_Toc423906623)

[GULP Tutorial Part 6 – Optimizing JavaScript/TypeScript 23](#_Toc423906624)

[Annotation 23](#_Toc423906625)

[Clean out ‘dist’ 25](#_Toc423906626)

[Copy all src files to ‘dist’ 27](#_Toc423906627)

[CONCATENATION 30](#_Toc423906628)

[Compress and Minify JavaScript 33](#_Toc423906629)

[GULP Tutorial Part 7 – Optimizing CSS 38](#_Toc423906630)

[GULP Tutorial Part 8 – Optimizing HTML 40](#_Toc423906631)

[GULP Tutorial Part 9 – Images 42](#_Toc423906632)

[GULP Tutorial Part 10 – JSON, calling grunt from gulp 44](#_Toc423906633)

[GULP Tutorial Part 11 – JSHINT 52](#_Toc423906634)

[GULP Tutorial Part 12 – TypeScript 58](#_Toc423906635)

[GULP Tutorial Part 13 – SASS 67](#_Toc423906636)

[GULP Tutorial Part 14 – Watch 78](#_Toc423906637)

[GULP Tutorial Part 15 – Useful Gulp Commands & Tips 84](#_Toc423906638)

[GULP Tutorial Part 16 – Glob Tips 86](#_Toc423906639)

[GULP Tutorial Part 17 – Useful NPM Packages/Commands 87](#_Toc423906640)

[Commands Cheat Sheet 87](#_Toc423906641)

[PowerShell (primer) 87](#_Toc423906642)

[Syntax 88](#_Toc423906643)

[Adding and removing files 88](#_Toc423906644)

[Installing NodeJS and NPM Packages 89](#_Toc423906645)

[NPM Version updates 90](#_Toc423906646)

### Introduction

The days of the casual web developer are over. Since the advent of Visual Studio and other IDEs it’s been possible to build post back styled line-of-business application with only a rudimentary understanding of web programming. ASP.NET developers tended toward tolerating JavaScript and hoping ASP.NET would generate the proper JavaScript.

Why

We are bombarded with a daily litany of new technologies “required” to build modern applications. While every new technology might not actually be “required” it’s important to have some understanding of each technologies place in this world.

This book is intended to get Gulp, a task runner, up and running for you today. It’s not intended to give you hidden secrets or even a thorough accounting of the Gulp API. Simply to get you up and running with working examples.

HOW TO USE THIS BOOK

Working through this tutorial styled book will expose you to a handful of the most important tasks you may need. I recommend following the tutorial, moving from chapter to chapter, in sequence. During this process you’ll experience some difficulties and resolving these will give you greater understanding of Gulp, Node, and NPM.

After you’ve completed the tutorial, pick a few tasks and apply them to your own project. Start with minification of your JavaScript and CSS.

Another common task is concatenating multiple JavaScript files together as this is a current common practice for performance. This is to compensate for a shortcoming of HTTP protocol. With the adoption of HTTP 2, you will no longer be required to concatenate files for performance.

The MashupJS

The MashupJS is a bootstrap for creating large enterprise applications. Its folder structure and Gulp implementation make it easy for development teams to build modules in isolation then combine their code with the MashupJS at build. The MashupJS is used as a code base for these tutorials to work against.

Start the tutorial using this code base: <https://github.com/MashupJS/gulp-tutorial>

A completed tutorial can be found here: <https://github.com/MashupJS/gulp-tutorial-end-result>

Gulp

# Gulp Tutorials

## GULP Tutorial Part 1 – Reasons for Build Tools like Gulp

### Productivity

**The reason for a build system is always productivity. Otherwise we wouldn’t invest time in it.**

**Build systems perform housecleaning work, allowing you to focus on code. Before build systems, if you were lucky, you could right click and select “minify” in your IDE. As lucky as this might have been, minification might not have been worth the additional development effort required. Build systems address this problem.**

**Build systems perform tasks with a level of precision humans are incapable of. For Continuous Integration and Continuous Delivery to work, a build system must be used to keep the human element out. Continuous Delivery requires automation at all levels, including testing, to mitigate common deployment defects.**

**There are thousands of plugins to perform just about any task imaginable. Here are a few.**

**Performance/Optimization**

* **Minification of JavaScript files**
* **Minification of CSS files**
* **Slimming down CSS classes that are not used**
* **Concatenating many JavaScript files to reduce get requests**
* **Creation of MAP files for debugging at run-time**

**Deployment**

* **Files can be optimized then copied to a folder to isolate deployment from development**
* **A zip file can be generated for deployment**
* **Automated tests can be executed**
* **Deployments can be created with a particular purpose; e.g., an app can be built for mobile.**

**Static analysis**

* **Linters can be executed against your code producing advice**
* **Cyclomatic complexity and other measures can be generated.**

**Documentation**

* **Documentation can be generated from code into readable formats.**
* **HTML documents can be generated from Markdown, a popular text format.**

**Additional resources**

<https://www.youtube.com/watch?v=XJ5F-Auhato>

## GULP Tutorial Part 2 – Setup

Here we need to install NodeJS, pull our tutorial project from GitHub, create our NPM package configuration file, and install Gulp both globally and locally.

**Installing NodeJS**

Download and install NodeJS.

<https://nodejs.org/>

**Get tutorial project from GitHub**

Code for this tutorial can be found at  
<https://github.com/MashupJS/gulp-tutorial>

On the GitHub repo page you’ll see an option to “Download ZIP”.

Download the ZIP file and extract it where you can work with it.

To see the end result, go to this repository.

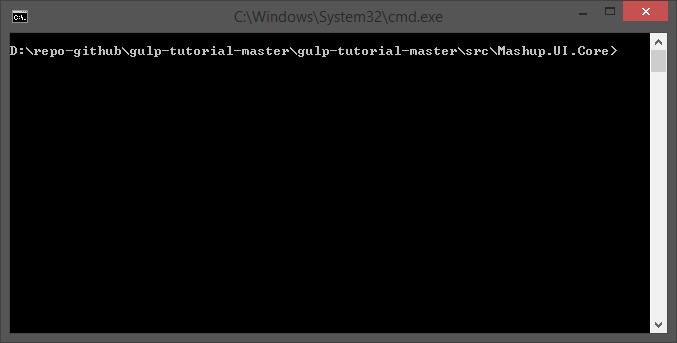
<https://github.com/MashupJS/gulp-tutorial-end-result>

**Setup the NPM Project Configuration File**

NPM packages are defined in the **package.json** file.

To create a package.json file, open a command prompt in the root of your client folder.

For this tutorial, open a command line to the Mashup.UI.Core folder.

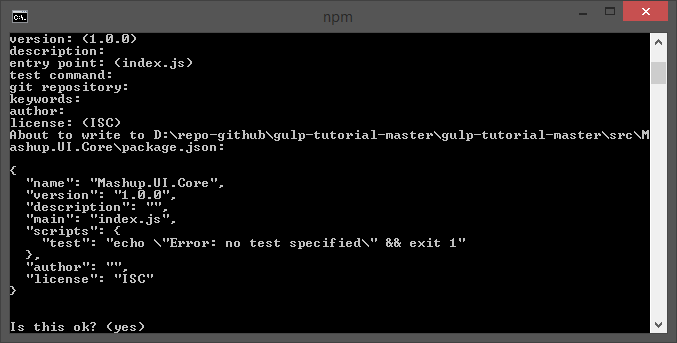


**At the command-line type**

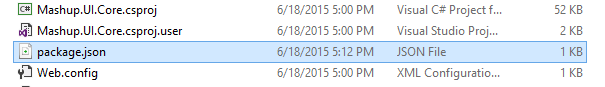
npm init

An empty package.json is created. Now we can begin installing NPM packages for use by Gulp.

You will be prompted for several configuration options. For the purposes of this tutorial I’ve skipped these and just pressed Enter for each prompt until complete.



And now your package.json is born.



For more information about NPM packages  
<https://docs.npmjs.com/files/package.json>

**Installing NPM packages**

First, let’s install the bower NPM module. We’ll need this to pull client side scripts of which we have many.

**From the command-line**

Npm install bower –g

**Retrieve all bower scripts from the command-line**

Bower install

**Installing Gulp**

At this point, Gulp can be installed with the following command. Notice the “-g” command. This causes the NPM package to be installed globally.

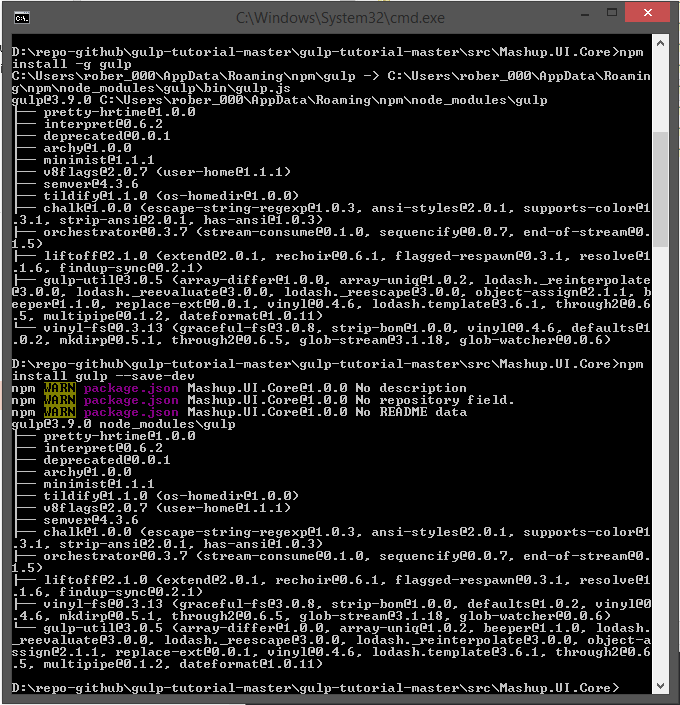
Installing Gulp (add “-g” to install globally)

npm install -g gulp

Gulp must also be installed locally for your project. The global install allows you to execute Gulp commands from the command-line by providing a Command-Line Interface or CLI. The local Gulp install is a plug-in to NodeJS and gets access to Gulp plugins via NPM.

Npm install gulp --save-dev

If you’re new to NPM, then just know that a screen that looks like this is completely normal.



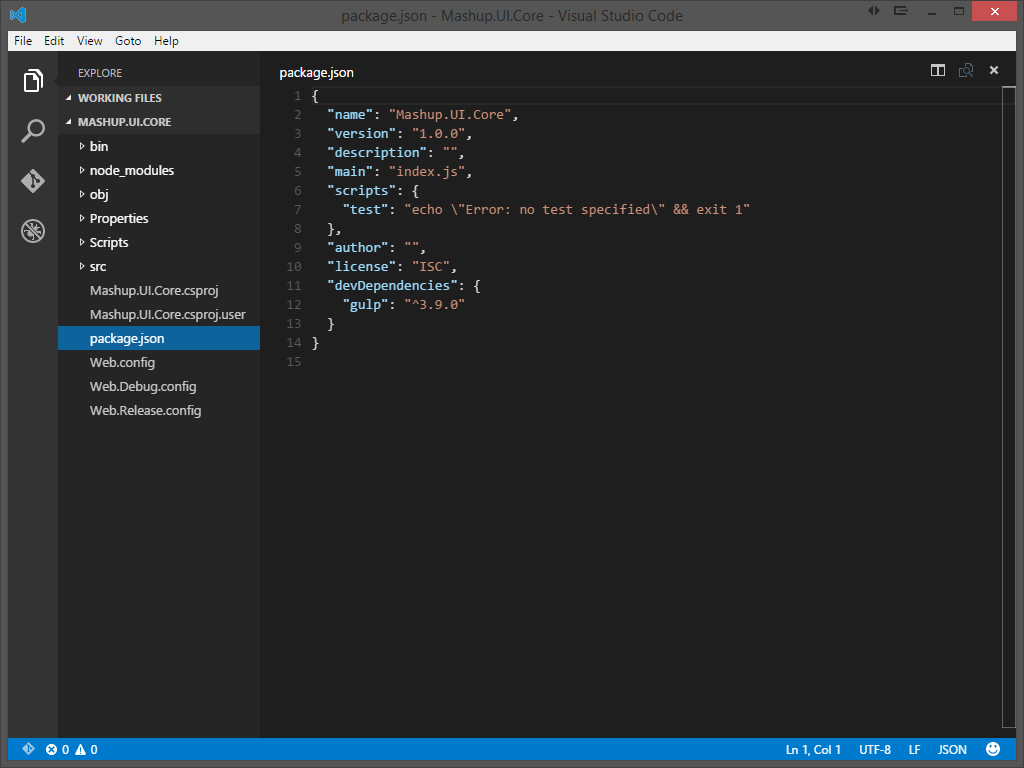
**TIP: Visual Studio Code**

Often I want to view a file without having to open an entire Integrated Development Environment like Visual Studio .NET. It’s just way more than I need to quickly view a file and in fact, often you can’t quickly view a file because VS .NET is so big.

Download and install Visual Studio Code. You can use programs like Notepad++ or Sublime as well.

In this case we just installed Gulp globally and locally. This tutorial is a learning tool so in the spirit of learning, each time I perform an action I’m going to poke around and see what changed. After installing Gulp I want to see what has changed.

Right click the package.json file and open it with your favorite code editor and see the change.



Package.json stores its information as JSON. Notice the first several attributes. These are the values we opted to provide, or, in my case, not provide.

When we applied the “--save-dev” options to the NPM install statement, the config was added to the “devDependencies” section.

The package.json file is a part of your code base and should be included/checked-in to source control. The “node\_modules” folder created by NPM should not be checked in to source control.

When setting up the development environment on a new machine, simply open a command-line to the folder where package.json resides and type:  
  
npm install

Or

npm update

NPM will then go download and install all the packages specified in the package.json.

For more information on installing NPM packages  
<https://docs.npmjs.com/getting-started/installing-npm-packages-locally>

**Creating the gulpfile.json**

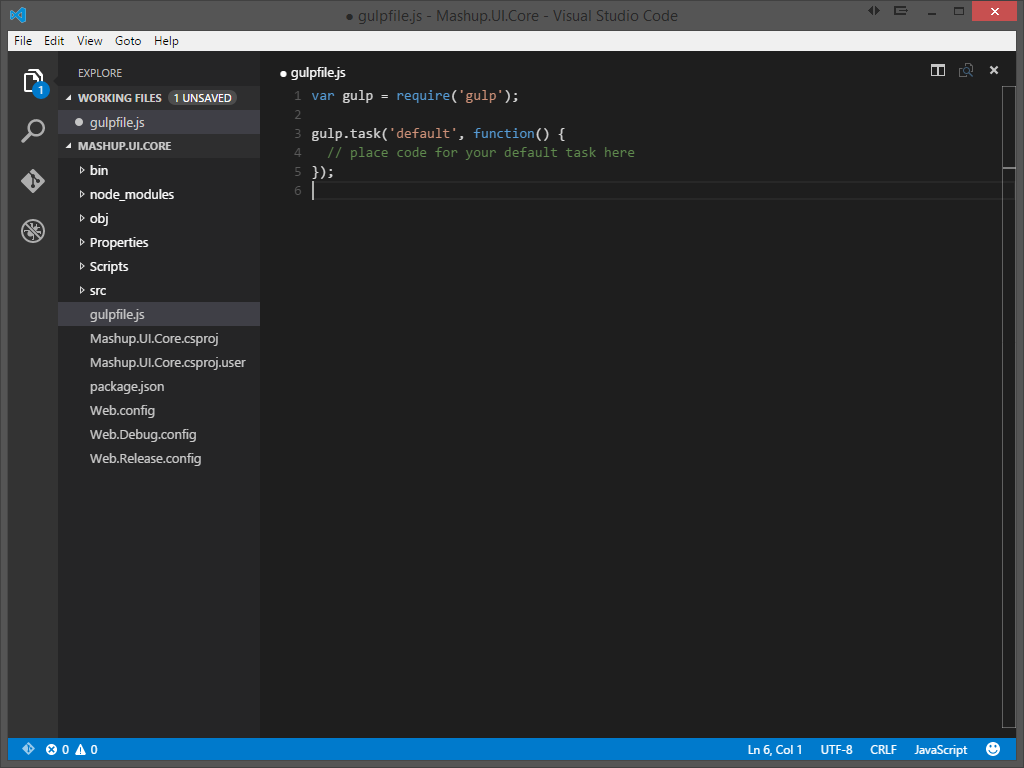
Create a text file named **gulpfile.js** with the following content, in the root of your project. Add the following scaffolding to the new gulpfile.js file.

var gulp = require('gulp');

gulp.task('default', function() {

// place code for your default task here

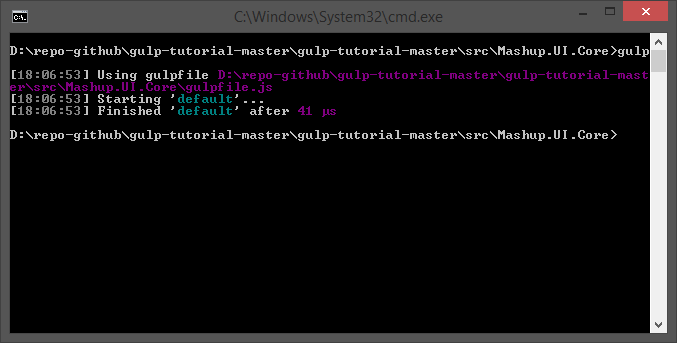
});



At this point you can execute Gulp from the command-line. There are no tasks in the “default” task, but Gulp will run.

At the command-line type “gulp” and press enter.

gulp



## GULP Tutorial Part 3 – Adding Plugins

Plugins provide function to the task runner.

Tip: When searching for plugins, consider the number of downloads the module has on NPM and activity on Github. These are indicators of how active the community is and how much support you can expect. You might find a dozen JavaScript minifiers. Choose the one with the most downloads and most recent activity on Github.

You can search for plugins here  
<http://gulpjs.com/plugins>

Once you’ve found a plugin, navigate to the plugins page. Here you’ll find general information on how to use the plugin and usually a couple examples to get you started.



Install a few useful plugins from the commandline of the root of your project. Notice “--save-dev”. This option includes the plugin in the package.json file.

You’ve already installed Gulp.

npm install gulp --save-dev

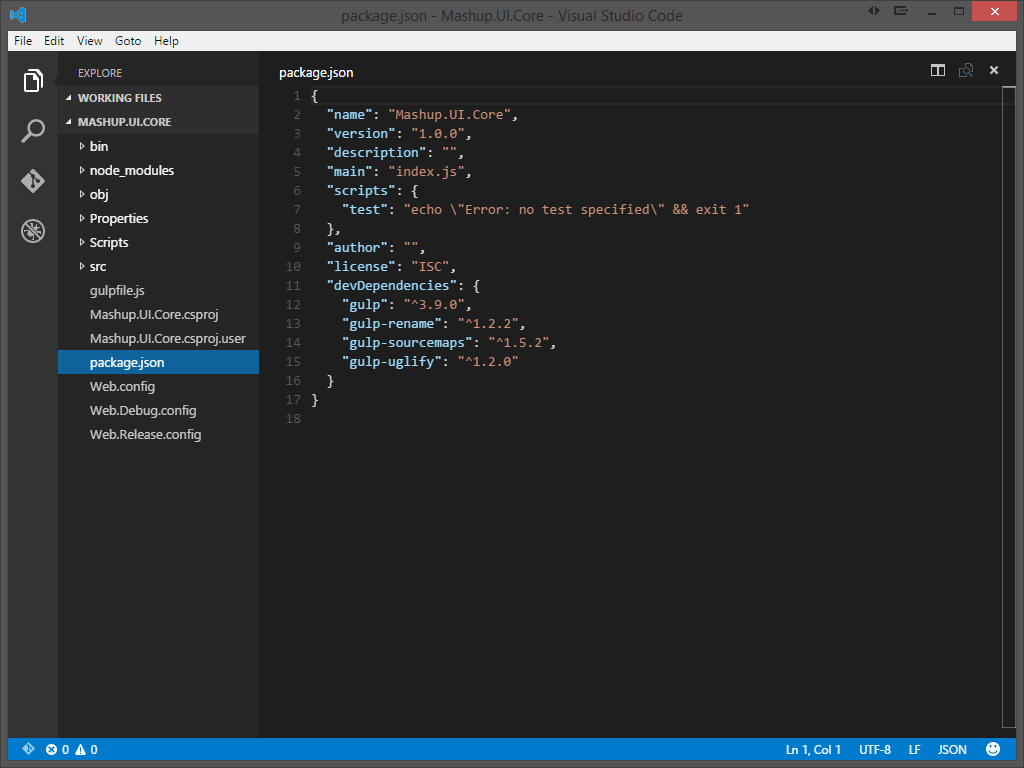
Go ahead and install a couple more. *(Just for fun)*

npm install gulp-uglify --save-dev

npm install gulp-rename --save-dev

npm install gulp-sourcemaps --save-dev

Viewing the package.json with VSC, you’ll notice the new plugin configurations are saved.



To add these plugins to your gulp implementation, add the new plugins to your gulpfile.js.

var gulp = require('gulp')

, uglify = require('gulp-uglify')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps')

;

gulp.task('default', function() {

// place code for your default task here

});

**Syntax for creating a task**

Gulp.task([task-name], function() {

Return gulp.src([glob-array]

.pipe([your-plugin])

.pipe([another-plugin])

.pipe(gulp.dest(dist));

});

Notice the “function” keyword. One of the more significant differences between Gulp and Grunt is configuration versus code. Gulp subscribes to a “code” approach while Grunt subscribes to “configuration”.

## GULP Tutorial Part 4 – Sequence and Parallel task processing

The ability to execute Gulp tasks in sequence and parallel is still a moving target. By default, Gulp leans toward executing all tasks in parallel because that is the more performant approach.

The option I chose was “run-sequence”. I chose this after battling with the other options. The Gulp 4 release should resolve many of the issues I struggled with.

We have not yet created tasks but in preparation, install this plug-in.

**From the command-line install**

npm install run-sequence --save-dev

**Add the module to the gulp file**

, runSequence = require('run-sequence')

All tasks created in this tutorial will have no other dependencies except the tasks we execute via the Gulp default task. The **runSequence** function will manage our dependencies. Creating dependencies is a simple option provided by Gulp but this causes tight coupling between tasks.

For instance, before optimizing files, we will clean out the distribution folder and then copy optimized files back to it. During development, however, cleaning the distribution folder breaks when all files are not copied and optimized back to it. An attempt to copy and optimize every file would cause a delay during development.

To resolve this, we will not tightly couple tasks, allowing all tasks to be executed when Gulp is run, but only the changes files are executed against when watching files during development.

Another example where sequence matters is how JavaScript is optimized. Some JavaScript is found in \*.js files while other JavaScript is found after the compilation of TypeScript down to JavaScript. To minify and optimize JavaScript effectively, it’s better to perform the JavaScript **Uglify** after the \*.ts, TypeScript, files have been transpiled. So a dependency exists between TypeScript and JavaScript.

We can accomplish this with **runSequence()**.

Here is what a default task might look like when using the runSequence function to manage tasks.  
*(Don’t add this code)*

gulp.task('default', function() { runSequence('clean-dist',

'annotate',

'copy',

['coreservices', 'routeconfig', 'sass', 'tscompile', 'libs', 'grunt-merge-json:menu',

'tslint', 'jshint', 'minifyhtml', 'minifyimage'],

['uglifyalljs', 'minifycss'],

'watch'

);

});

**Here is the sequence of execution**

1 clean-dist

2 annotate

3 copy

4 (run in parallel) coreservices, routeconfig, sass, tscompile, libs, grunt-merge-json:menu, tslint, jshint, minifyhtml, minifyimage

5 uglifyalljs, minifycss

6 watch

These are all tasks you will have created by the end of this multi-part tutorial.

**Optimizing task performance**

Notice the number of tasks executed in step 4. The more tasks you can run in parallel, the faster your process will be. It’s important to optimize your process as much as possible so you can change a piece of code and immediately execute the optimized version without delay.

**Other options**

Gulp 4.0 will have new methods series() and parallel(). This will be the preferred approach once released.

Orchestrator – is an NPM module that supports series and parallel processing.

## GULP Tutorial Part 5 – Handling errors with Plumber

**From the command-line install**

npm install gulp-plumber --save-dev

**Add the module to the Gulp file**

, plumber = require('gulp-plumber');

Add this to the top of your script file. Our **plumber** will use this function for logging errors to the console.

var onError = function(err) {

console.log(err);

};

Here is what a task might look like with the **plumber()** function.   
*(Don’t add this code)*

// ---------------------------------------------------------------

// Watch specific tasks. This is to support the use of newer.

// ---------------------------------------------------------------

gulp.task('watch:annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!/\*\*/\*.min.js'], { base: 'src/./' })

**.pipe(plumber({**

**errorHandler: onError**

**}))**

.pipe(newer('src/./'))

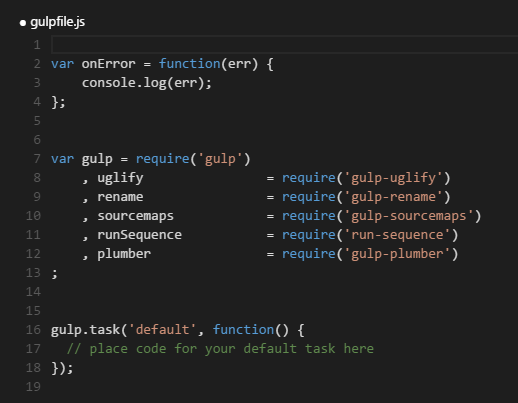
.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

The following tutorials will implement this **plumber** function with each task we create.

Currently, your **gulpfile.js** should look like this:



## GULP Tutorial Part 6 – Optimizing JavaScript/TypeScript

In this part of the tutorial, we are setting up our first Gulp task.

### Annotation

Before we concatenate and minify, let’s make sure our Angular code is in good shape. To make a long story short… Angular is based, largely, on the ability to directly inject dependencies. This ability is made possible because the name of the injected dependency is interpreted. As soon as a file is minified, function names are changed to ‘a’ or ‘b’ or whatever is the next available small variable name. This breaks dependency injection.

Passing the dependency name as a string corrects this problem. Static string names are not minified. You can do this yourself or let Gulp run a task to do this for you. Even if you decided to handle this while writing code, it’s a good idea to run an annotation task as a precaution.

**From the command-line install**

npm install gulp-ng-annotate --save-dev

**Add the module to the gulp file**

, ngAnnotate = require('gulp-ng-annotate')

**Add task to Gulp file**

gulp.task('annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!src/\*\*/\*.min.js'], { base: 'src/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

The challenge presented by this task is updating the original files. In most cases, the original files are left alone. In this case, we can do the same, but since annotation is actually a correction to code, we will update the original code files. To make this work, we are using the **{ base: ‘src/./’ }** option.

TIP: To keep your tasks running fast, eliminate unnecessary processing by telling the task to ignore your JavaScript libraries, i.e., '!src/core/lib/\*\*/\*'

**Add the new task to the default task**

gulp.task('default', function () {

runSequence(**'annotate'**);

});

**Run the default task**

gulp



**For more information**

<http://christian.fei.ninja/DRY-dependency-injection-in-Angular-with-gulp-ng-annotate/>

### Clean out ‘dist’

Now, let’s clean out our ‘dist’ folder so we’re starting afresh. Execute the following from PowerShell.

**From the command-line install**

npm install gulp-clean --save-dev

**Add the module to the Gulp file**

, clean = require('gulp-clean')

**Add the task to the Gulp file**

gulp.task('clean-dist', function () {

return gulp.src('dist', { read: false })

.pipe(plumber({

errorHandler: onError

}))

.pipe(clean());

});

**Add the new task to the default task**  
This task will run in sequence after the annotate task completes.

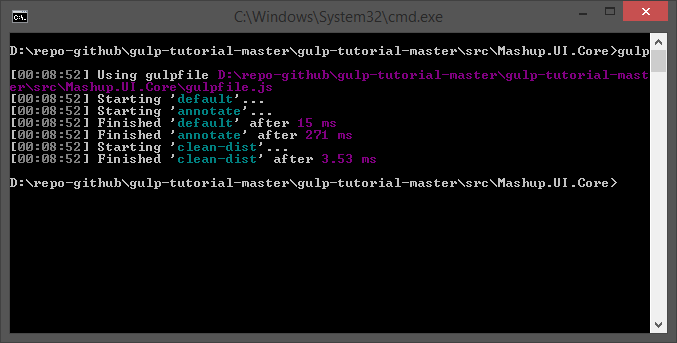
gulp.task('default', function () {

runSequence('annotate'**, 'clean-dist'**);

});

**Run the default task**

gulp



**For more information**

<https://www.npmjs.com/package/gulp-clean>

### Copy all src files to ‘dist’

We will keep our source (‘scr’) code separate from our distribution code so we don’t pollute our development environment. When executing the application we’ll set the **index.html** file of the ‘dist’ folder as the startup. This approach might seem to introduce challenges because when debugging you’ll need the ability to read the compressed and minified versions of JavaScript and CSS. Gulp will give us that ability.

Now that we’ve cleaned out the ‘dist’ folder, in preparation for new files, let’s go ahead and copy all our source code to ‘dist’. Once we’ve copied the source code we can begin running tasks to optimize our code.

The newer module can be used by any task to ensure only the changed file is streamed to the task. This increases performance especially during development when only a single files is changed.

**From the command-line install**

Npm install gulp-newer --save-dev

**Add the module to the Gulp file**

, newer = require('gulp-newer')

**Add the task to the Gulp file**

gulp.task('copy', function () {

return gulp.src('src/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(newer('dist'))

.pipe(gulp.dest('dist'));

});

**Add the new task to the default task**

gulp.task('default', function () {

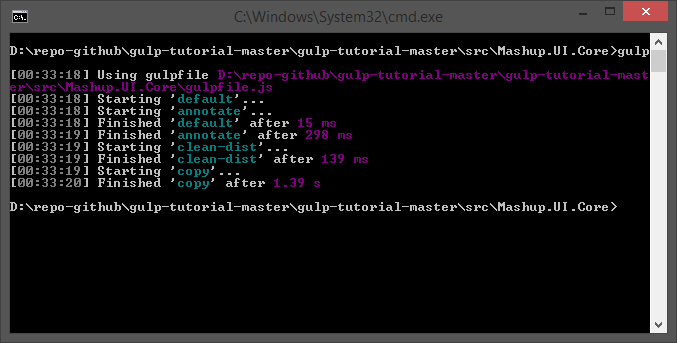
runSequence('annotate', 'clean-dist', **'copy'**);

});

**Run the default task**

The Gulp command and the dist folder will be deleted and rebuilt.

Gulp



**For more information**

<https://www.npmjs.com/package/gulp-newer>

### CONCATENATION

Before we minify our JavaScript files, let’s see if there are any files we wish to combine. We could simply minify all JavaScript files, then concatenate them together, but then we would lose the ability to map minified code to source code for debugging purposes. We will concatenate any code we desire and then execute a general minification task.

**From the command-line install**

npm install gulp-concat --save-dev

**Add the module to the Gulp file**

, concat = require('gulp-concat')

**Add the task to the Gulp file**

Create a task that combines all the **core/common** files into one.

gulp.task('coreservices', function () {

return gulp.src('src/core/common/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('core.services.js'))

.pipe(gulp.dest('./dist/'));

});

Create a task that combines all the **route.config**.js files together.

gulp.task('routeconfig', function () {

return gulp.src(['src/core/config/route.config.js', 'src/apps/\*\*/route.config.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('route.config.js'))

.pipe(gulp.dest('./dist/'));

});

NOTE: This capability to combine the route config is what makes the drop-in application style of MashupJS work.

Now copy all your **bower libraries** together. Typically, you’d combine as many of these files as possible. For now, we will work with third party libraries as separate files.

gulp.task('libs', function () {

return gulp.src(['bower\_components/\*\*//bootstrap/dist/js/bootstrap.min.js'

, 'bower\_components/\*\*//normalize.css/normalize.css'

, 'bower\_components/\*\*//fontawesome/css/font-awesome.min.css'

, 'bower\_components/\*\*/fontawesome/fonts/\*.\*'

, 'bower\_components/\*\*//jquery/dist/jquery.min.js'

, 'bower\_components/\*\*//angular/\*.min.js'

, 'bower\_components/\*\*//angular-route/angular-route.min.js'

, 'bower\_components/\*\*//angular-sanitize/angular-sanitize.min.js'

, 'bower\_components/\*\*//angular-bootstrap/ui-bootstrap-tpls.min.js'

, 'bower\_components/\*\*//lodash/lodash.min.js'])

.pipe(plumber({

errorHandler: onError

}))

//.pipe(concat('libs.js'))

.pipe(gulp.dest('dist/core/lib/bower/./'));

});

**Add the new tasks to the default task**

gulp.task('default', function () {

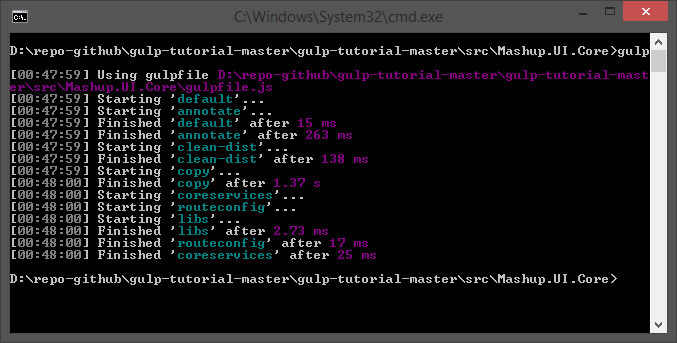
runSequence('annotate', 'clean-dist', 'copy',

**['coreservices', 'routeconfig', 'libs']**);

});

For additional documentation on gulp-concat: <https://github.com/wearefractal/gulp-concat>

**Run the default task**  
gulp



**For more information**

<https://www.npmjs.com/package/gulp-concat>

### Compress and Minify JavaScript

It’s finally time to minify and compress JavaScript files. A normal solution would be to concatenate all \*.js files into a single file name,”app.js” or “all.app.js” or something similar. The MashupJS is built to scale so large that a single concatenated \*.js file might become too large and lazy loading will be desired.

Here we will minify and compress individual JavaScript files. Source maps will be created for troubleshooting and debugging. Source maps link between the original source code and the optimized code.

**From the command-line install**

npm install gulp-rename --save-dev

npm install gulp-uglify --save-dev

npm install gulp-sourcemaps --save-dev

**Add the new modules to the Gulp file**

, rename = require('gulp-rename')

, uglify = require('gulp-uglify')

, sourcemaps = require('gulp-sourcemaps')

**Add the task to the Gulp file**

gulp.task('uglifyalljs', function () {

//gulp.task('uglifyalljs', ['copy', 'coreservices', 'routeconfig', 'tscompile'], function () {

return gulp.src(['dist/\*\*/\*.js', '!/\*\*/\*.min.js', '!dist/core/lib/\*\*/\*', '!dist/core/common/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

// .pipe(newer('dist/./'))

.pipe(uglify())

.pipe(rename({

extname: '.min.js'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

**Add the new task to the default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs'],

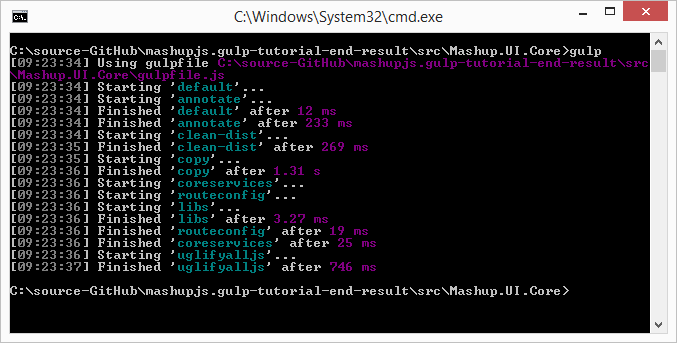
**['uglifyalljs']**);

});

Notice the “uglifyalljs” task is placed in sequence to execute only after the previous array of tasks executes. This is to allow the “coreservices” and “routeconfig” JavaScripts to be created. Then they can be optimized by “uglifyalljs”.

**Run the default task**

gulp



**For more information**

<https://www.npmjs.com/package/gulp-rename>

<https://www.npmjs.com/package/gulp-uglify>

<https://www.npmjs.com/package/gulp-sourcemaps>

**Current state of code**

package.json

{

"name": "Mashup.UI.Core",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC",

"devDependencies": {

"gulp": "^3.9.0",

"gulp-clean": "^0.3.1",

"gulp-concat": "^2.5.2",

"gulp-newer": "^0.5.1",

"gulp-ng-annotate": "^1.0.0",

"gulp-plumber": "^1.0.1",

"gulp-rename": "^1.2.2",

"gulp-sourcemaps": "^1.5.2",

"gulp-uglify": "^1.2.0",

"run-sequence": "^1.1.1"

}

}

gulpfile.js

var onError = function(err) {

console.log(err);

};

var gulp = require('gulp')

, uglify = require('gulp-uglify')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps')

, runSequence = require('run-sequence')

, plumber = require('gulp-plumber')

, ngAnnotate = require('gulp-ng-annotate')

, clean = require('gulp-clean')

, newer = require('gulp-newer')

, concat = require('gulp-concat')

, rename = require('gulp-rename')

, uglify = require('gulp-uglify')

, sourcemaps = require('gulp-sourcemaps')

;

gulp.task('annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!src/\*\*/\*.min.js'], { base: 'src/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

gulp.task('clean-dist', function () {

return gulp.src('dist', { read: false })

.pipe(plumber({

errorHandler: onError

}))

.pipe(clean());

});

gulp.task('copy', function () {

return gulp.src('src/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(newer('dist'))

.pipe(gulp.dest('dist'));

});

gulp.task('coreservices', function () {

return gulp.src('src/core/common/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('core.services.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('routeconfig', function () {

return gulp.src(['src/core/config/route.config.js', 'src/apps/\*\*/route.config.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('route.config.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('libs', function () {

return gulp.src(['bower\_components/\*\*/\*.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('libs.js'))

.pipe(gulp.dest('dist/core/lib/'));

});

gulp.task('uglifyalljs', function () {

//gulp.task('uglifyalljs', ['copy', 'coreservices', 'routeconfig', 'tscompile'], function () {

return gulp.src(['dist/\*\*/\*.js', '!/\*\*/\*.min.js', '!dist/core/lib/\*\*/\*', '!dist/core/common/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

// .pipe(newer('dist/./'))

.pipe(uglify())

.pipe(rename({

extname: '.min.js'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

// ----------------------------------------------------------------

// Default Task

// ----------------------------------------------------------------

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs'],

['uglifyalljs']);

});

## GULP Tutorial Part 7 – Optimizing CSS

CSS gives us another opportunity for optimization. SASS is a higher-level language for CSS that can be transpiled down to CSS similar to how TypeScript is transpiled down to JavaScript. Later in this tutorial, we will add a task for transpiling SASS to CSS.

In this task, we will optimize all existing CSS files.

**From the command-line install**

npm install gulp-minify-css --save-dev

**Add the module to the Gulp file**

, minifycss = require('gulp-minify-css')

**Add the task to the Gulp file**

gulp.task('minifycss', ['copy'], function () {

return gulp.src(['dist/\*\*/\*.css', '!/\*\*/\*.min.css', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(sourcemaps.init())

.pipe(minifycss())

.pipe(rename({

extname: '.min.css'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

**Add the new task to the default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs'],

['uglifyalljs'**, 'minifycss'**]);

});

**Run the default task**

gulp



Rather than concatenating CSS files, we are simply minifying them in place and creating maps.

Later, when transpiling from SASS, we won’t need concatenation because the “@import” statement will pull multiple source files together for us.

**For more information**  
<https://www.npmjs.com/package/gulp-minify-css>

## GULP Tutorial Part 8 – Optimizing HTML

HTML presents another opportunity for optimization. With every “bit” we can squeeze out of the code, the healthier is the application. HTML optimization is more noticeable as the HTML file becomes larger.

**From the command-line install**

npm install gulp-minify-html --save-dev

**Add the module to the Gulp file**

, minifyhtml = require('gulp-minify-html')

**Add the task to the Gulp file**

gulp.task('minifyhtml', function () {

return gulp.src(['dist/\*\*/\*.html', '!/\*\*/\*.min.html', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(minifyhtml())

.pipe(rename({

extname: '.min.html'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

Small templates won’t realize much improvement with HTML minification, but every little bit helps. Larger HTML files will benefit, but while we’re at it, let’s just minify all HTML files.

**Add new task to the default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs'**, 'minifyhtml'**],

['uglifyalljs'**,** 'minifycss']);

});

**Run the default task**

gulp



For more information on gulp-minify-html.  
<https://www.npmjs.com/package/gulp-minify-html>

## GULP Tutorial Part 9 – Images

Install the **gulp-imagemin** and **imagemin-pngquant** plugins. The gulp-imagemin comes with several optimizers for different image types, but does very little for PNG files. The imagemin-pngquant will focus on optimizing PNG files.

I’ve decided to leave the third party libraries alone, but you might choose to optimize them as well. This demo is focused on optimizing our own images.

**From the command-line install**

npm install gulp-imagemin --save-dev

npm install imagemin-pngquant --save-dev

**Add the modules to the Gulp file**

, imagemin = require('gulp-imagemin')

, pngquant = require('imagemin-pngquant')

**Add the task to the Gulp file**

gulp.task('minifyimage', function () {

return gulp.src(['dist/\*\*/\*.{png,jpg,gif,ico}', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(imagemin({ progressive: true, optimizationLevel: 7, use: [pngquant()] }))

.pipe(gulp.dest('dist/./'));

});

**Add the new task to the default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

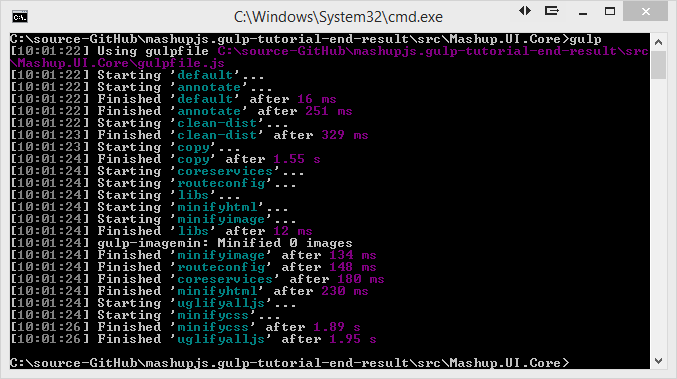
['coreservices', 'routeconfig', 'libs', 'minifyhtml'**, 'minifyimage'**],

['uglifyalljs', 'minifycss']);

});

**Run the default task**

gulp



**For more information**

<https://www.npmjs.com/package/gulp-imagemin>

<https://www.npmjs.com/package/imagemin-pngquant>

## GULP Tutorial Part 10 – JSON, calling grunt from gulp

On occasion you might need to combine JSON files. The MashupJS allows each app in the apps folder to define its own menu items. At build time we need these JSON files to be combined and saved with a specific file name so the menu.html template has access to all the menu items.

**Edge case: Calling Grunt tasks from Gulp**

So as it turns out, some of our favorite plugins used in Grunt are not available in Gulp. Before using Gulp I used Grunt and a plugin named “grunt-merge-json” to combine separate JSON files. Not only did it combine the JSON but merged them and eliminated duplication. I tried several Gulp plugins but nothing worked as expected. In time, a developer will build this plugin for Gulp but until then we can execute Grunt commands from our Gulp implementation.

**From the command-line install**

npm install -g grunt-cli

npm install grunt-merge-json --save-dev

**Create a Gruntfile**

Create a basic Gruntfile.js with the following content, in the root of your project.

module.exports = function (grunt) {

grunt.initConfig({

distFolder: 'dist',

pkg: grunt.file.readJSON('package.json'),

"merge-json": {

menu: {

src: ['src/apps/\*\*/menu.json.txt'],

dest: '<%= distFolder %>/menu.json.txt',

},

},

});

// Load modules, register tasks

grunt.loadNpmTasks('grunt-merge-json');

};

**Install Gulp-Grunt from the command-line**

npm install gulp-grunt --save-dev

**For more information**

<https://www.npmjs.com/package/gulp-grunt>

**Add Grunt configuration to Gulp**

Place this Grunt configuration code in the gulpfile.js just as you would with any other task.

// -------------------------------------------------

// Grunt configuration

require('gulp-grunt')(gulp, {

// These are the default options but included here for readability.

base: null,

prefix: 'grunt-',

verbose: false

});

// -------------------------------------------------

These are default configurations. The base represents the path to the Gruntfile.js. Because the Gruntfile.js is in the root folder base: can be null.

We can now call the Grunt task the same way we would a Gulp task but with the prefix “grunt=”.

**Test Grunt Plugin**

Gulp grunt-merge-json:menu

**Add the new Grunt task to the default task.**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

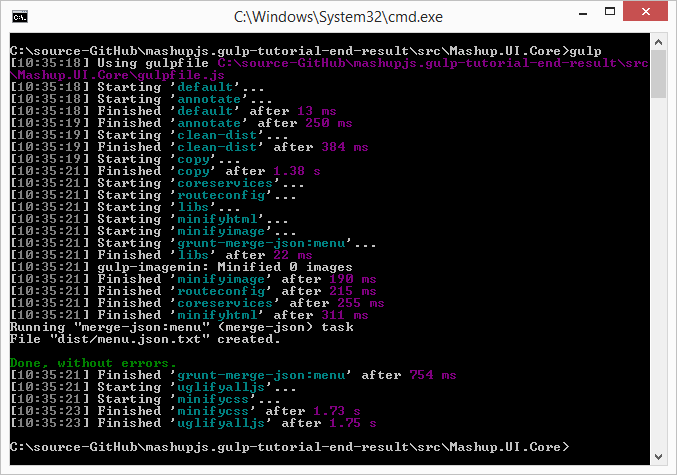
**, 'grunt-merge-json:menu'**],

['uglifyalljs', 'minifycss']);

});

**Run the default task**

gulp



**Current state of code**

**package.json**

{

"name": "Mashup.UI.Core",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC",

"devDependencies": {

"grunt": "^0.4.5",

"grunt-merge-json": "^0.9.5",

"gulp": "^3.9.0",

"gulp-clean": "^0.3.1",

"gulp-concat": "^2.5.2",

"gulp-grunt": "^0.5.2",

"gulp-imagemin": "^2.2.1",

"gulp-minify-css": "^1.1.6",

"gulp-minify-html": "^1.0.3",

"gulp-newer": "^0.5.1",

"gulp-ng-annotate": "^1.0.0",

"gulp-plumber": "^1.0.1",

"gulp-rename": "^1.2.2",

"gulp-sourcemaps": "^1.5.2",

"gulp-uglify": "^1.2.0",

"imagemin-pngquant": "^4.1.0",

"run-sequence": "^1.1.1"

}

}

**gulpfile.js**

var onError = function (err) {

console.log(err);

};

var gulp = require('gulp')

, uglify = require('gulp-uglify')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps')

, runSequence = require('run-sequence')

, plumber = require('gulp-plumber')

, ngAnnotate = require('gulp-ng-annotate')

, clean = require('gulp-clean')

, newer = require('gulp-newer')

, concat = require('gulp-concat')

, rename = require('gulp-rename')

, uglify = require('gulp-uglify')

, sourcemaps = require('gulp-sourcemaps')

, minifycss = require('gulp-minify-css')

, minifyhtml = require('gulp-minify-html')

, imagemin = require('gulp-imagemin')

, pngquant = require('imagemin-pngquant')

;

gulp.task('annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!src/\*\*/\*.min.js'], { base: 'src/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

gulp.task('clean-dist', function () {

return gulp.src('dist', { read: false })

.pipe(plumber({

errorHandler: onError

}))

.pipe(clean());

});

gulp.task('copy', function () {

return gulp.src('src/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(newer('dist'))

.pipe(gulp.dest('dist'));

});

gulp.task('coreservices', function () {

return gulp.src('src/core/common/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('core.services.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('routeconfig', function () {

return gulp.src(['src/core/config/route.config.js', 'src/apps/\*\*/route.config.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('route.config.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('libs', function () {

return gulp.src(['bower\_components/\*\*/\*.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('libs.js'))

.pipe(gulp.dest('dist/core/lib/'));

});

gulp.task('uglifyalljs', function () {

//gulp.task('uglifyalljs', ['copy', 'coreservices', 'routeconfig', 'tscompile'], function () {

return gulp.src(['dist/\*\*/\*.js', '!/\*\*/\*.min.js', '!dist/core/lib/\*\*/\*', '!dist/core/common/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

// .pipe(newer('dist/./'))

.pipe(uglify())

.pipe(rename({

extname: '.min.js'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifycss', function () {

return gulp.src(['dist/\*\*/\*.css', '!dist/\*\*/\*.min.css', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(minifycss())

.pipe(rename({

extname: '.min.css'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifyhtml', function () {

return gulp.src(['dist/\*\*/\*.html', '!/\*\*/\*.min.html', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(minifyhtml())

.pipe(rename({

extname: '.min.html'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifyimage', function () {

return gulp.src(['dist/\*\*/\*.{png,jpg,gif,ico}', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(imagemin({ progressive: true, optimizationLevel: 7, use: [pngquant()] }))

.pipe(gulp.dest('dist/./'));

});

// -------------------------------------------------

// Grunt configuration

require('gulp-grunt')(gulp, {

// These are the default options but included here for readability.

base: null,

prefix: 'grunt-',

verbose: false

});

// -------------------------------------------------

// ----------------------------------------------------------------

// Default Task

// ----------------------------------------------------------------

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

, 'grunt-merge-json:menu'],

['uglifyalljs', 'minifycss']);

});

**Gruntfile.js**

module.exports = function (grunt) {

grunt.initConfig({

distFolder: 'dist',

pkg: grunt.file.readJSON('package.json'),

"merge-json": {

menu: {

src: ['src/apps/\*\*/menu.json.txt'],

dest: '<%= distFolder %>/menu.json.txt',

},

},

});

// Load modules, register tasks

grunt.loadNpmTasks('grunt-merge-json');

};

## GULP Tutorial Part 11 – JSHINT

JSHint is a static analysis tool used to analyze JavaScript code for quality and standards/style enforcement. The example below demonstrates how to analyze your code but it can also be piped into transpilation of code from TypeScript to JavaScript.

Install the **gulp-jshint**, **jshint-stylish**, and **gulp-jshint-html-reporter**.

npm install gulp-jshint --save-dev

npm install jshint-stylish --save-dev

npm install gulp-jshint-html-reporter --save-dev

Add the new plugins to your Gulp required list

, jshint = require('gulp-jshint')

, stylish = require('jshint-stylish')

, jshintfileoutput = require('gulp-jshint-html-reporter')

Add the new task to your gulpfile.js

gulp.task('jshint', function () {

//gulp.task('jshint', ['copy', 'tscompile'], function () {

return gulp.src(['./dist/\*\*/\*.js', '!dist/core/lib/\*\*/\*.\*', '!\*\*/\*.min.js', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(jshint('.jshintrc'))

.pipe(jshint.reporter(stylish))

.pipe(jshint.reporter('gulp-jshint-html-reporter', { filename: 'jshint-output.html' }))

;

});

**Creating a** **.jshintrc file**

You can customize your own jshint rules in a file called .jshintrc. I’m starting with options recommended by John Papa.

<https://github.com/johnpapa/angular-styleguide#use-an-options-file>

**.jshintrc TIP**

You cannot easily create files beginning with a dot in Windows.

Steps to create the .jshintrc file:

* Create a file named **“.jshintrc.”** in the root alongside gulpfile.js and Gruntfile.js. Notice the ending dot. You must do this from Explorer.
* Windows will prompt you to confirm the extension.
* Finally Windows simply removes the ending dot for you. Now you have the file “.jshintrc”.
* If you are using Visual Studio .NET or some other IDE, you can now go and add the existing file to your project.

**.jshintrc content**

We are going to use the options promoted by John Papa.

Copy and paste this into your new .jshintrc file or visit John Papa’s page with the link above to see if any new updates have become available.

{

"bitwise": true,

"camelcase": true,

"curly": true,

"eqeqeq": true,

"es3": false,

"forin": true,

"freeze": true,

"immed": true,

"indent": 4,

"latedef": "nofunc",

"newcap": true,

"noarg": true,

"noempty": true,

"nonbsp": true,

"nonew": true,

"plusplus": false,

"quotmark": "single",

"undef": true,

"unused": false,

"strict": false,

"maxparams": 10,

"maxdepth": 5,

"maxstatements": 40,

"maxcomplexity": 8,

"maxlen": 120,

"asi": false,

"boss": false,

"debug": false,

"eqnull": true,

"esnext": false,

"evil": false,

"expr": false,

"funcscope": false,

"globalstrict": false,

"iterator": false,

"lastsemic": false,

"laxbreak": false,

"laxcomma": false,

"loopfunc": true,

"maxerr": false,

"moz": false,

"multistr": false,

"notypeof": false,

"proto": false,

"scripturl": false,

"shadow": false,

"sub": true,

"supernew": false,

"validthis": false,

"noyield": false,

"browser": true,

"node": true,

"globals": {

"angular": false,

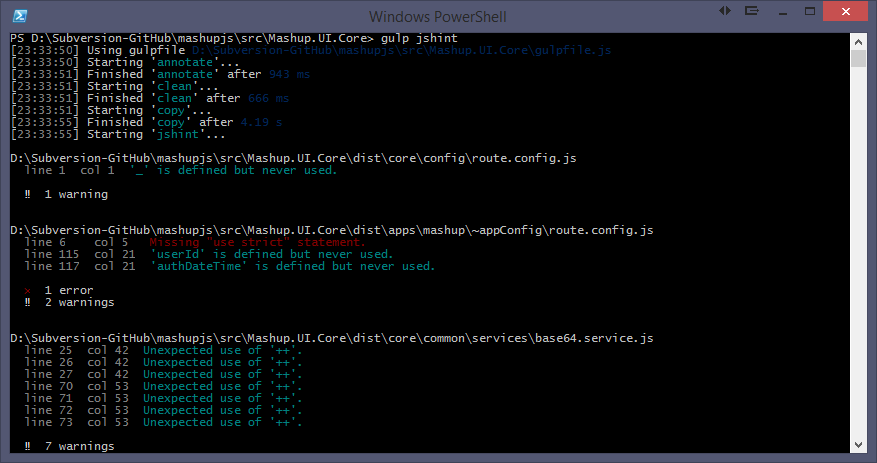
"$": false

}

}

**Using JSHint reporters**

Reporters receive feedback from JSHint and format it into something human readable. By default, you’ll get raw text from JSHint and it will be displayed at the command line. Adding the ‘jshint-stylish’ plugin and passing it to the jshint.reporter(stylish) gives you a more readable output to the command line.



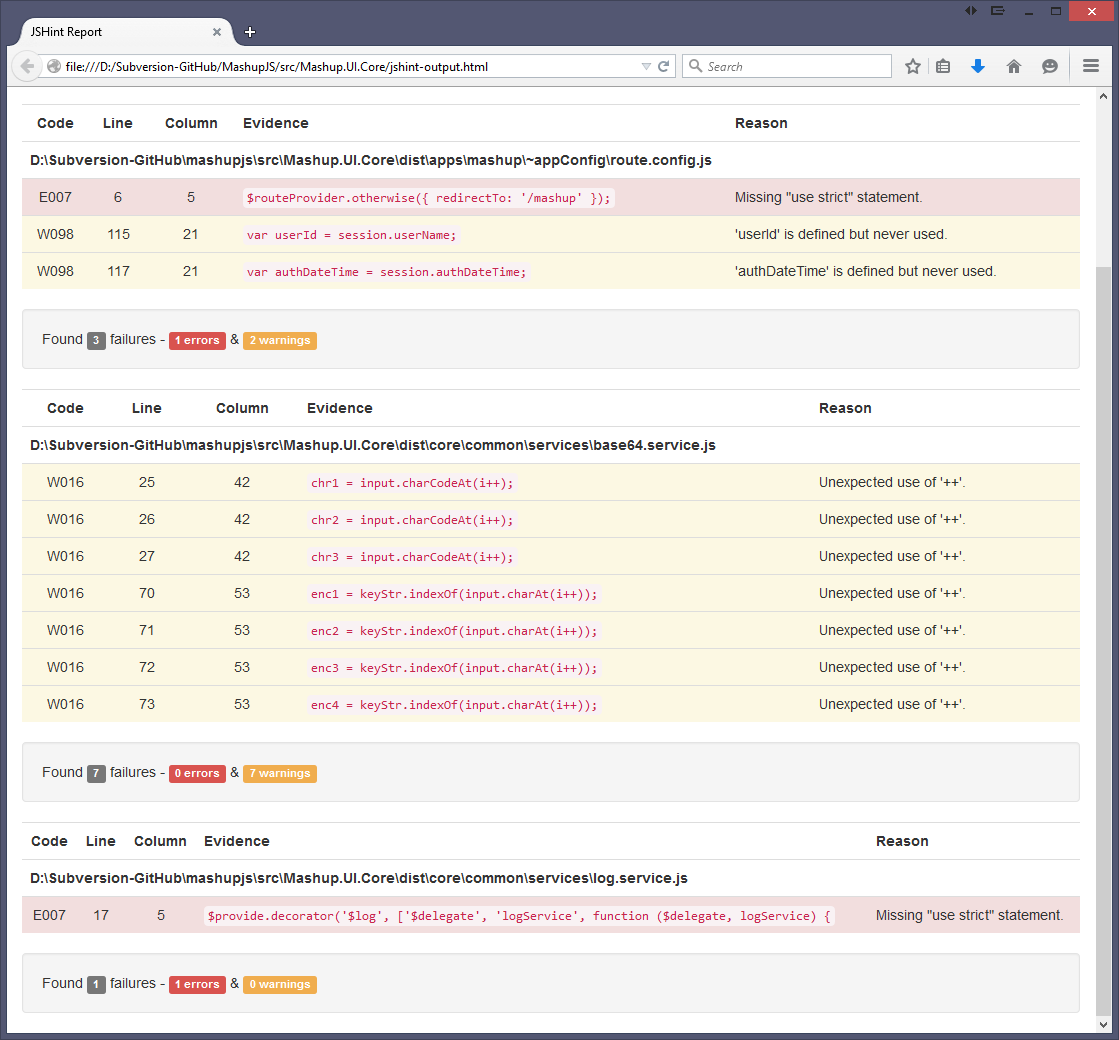
Taking this concept a step further, the ‘gulp-jshint-html-reporter’ generates an html document, as the name of the plugin implies.

**For more information**

<https://www.npmjs.com/package/gulp-jshint-html-reporter>

<https://github.com/ivan-vesely/gulp-jshint-html-reporter>

There isn’t much in the way of documentation but the source code is available and you have access to the author here.



**Options list**

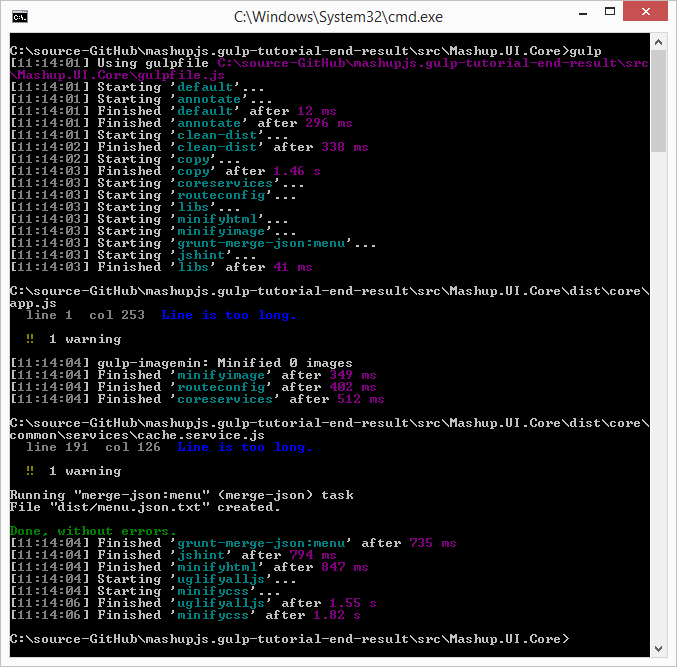
<http://jshint.com/docs/options/>

**For more information**

<https://www.npmjs.com/package/gulp-jshint>

**Run the default task**

gulp



**Tip**

The jshint-output.html file is a good indicator of how your recent changes might have deviated from your standard or best practices. By checking this file into source control you can see if the file changes. If it does not, then no new errors are being reported. If it does change, then either old errors have been corrected or new errors have been detected.

## GULP Tutorial Part 12 – TypeScript

TypeScript is a language used to build large-scale JavaScript applications. TypeScript code is transpiled down to ES5 JavaScript. Angular 2.0 is written in TypeScript and ES2015/ES2016, formerly ES6/ES7, features are available in TypeScript.

Links for installing TypeScript, for your version of Visual Studio, are here  
<http://www.typescriptlang.org/>

Dan Wahlin’s take on TypeScript in a Gulp workflow  
<http://weblogs.asp.net/dwahlin/creating-a-typescript-workflow-with-gulp>

**The TypeScript file for this tutorial**

The MashupJS isn’t using TypeScript yet so I created a TypeScript file for this example.

I created a file in the root of the scr folder named myTypeScript.ts.

I added the following code snippet from <http://www.typescriptlang.org/Playground> to the new file.

class Greeter {

greeting: string;

constructor(message: string) {

this.greeting = message;

}

greet() {

return "Hello, " + this.greeting;

}

}

var greeter = new Greeter("world");

var button = document.createElement('button');

button.textContent = "Say Hello";

button.onclick = function() {

alert(greeter.greet());

}

document.body.appendChild(button);

**Expected result**

Once we’ve transpiled the TypeScript down to JavaScript (ES5) we will get a myTypeScript.js file and a minified version. Here is what the JS file should look like.

var Greeter = (function () {

function Greeter(message) {

this.greeting = message;

}

Greeter.prototype.greet = function () {

return "Hello, " + this.greeting;

};

return Greeter;

})();

var greeter = new Greeter("world");

var button = document.createElement('button');

button.textContent = "Say Hello";

button.onclick = function () {

alert(greeter.greet());

};

document.body.appendChild(button);

**Installing plugins**

npm install gulp-typescript --save-dev

npm install gulp-tslint --save-dev

npm install gulp-tslint-stylish --save-dev

**Add the new plugins to your Gulp required list**

, ts = require('gulp-typescript')

, tslint = require('gulp-tslint')

, tsstylish = require('gulp-tslint-stylish')

**Add the new task to your gulpfile.js**

gulp.task('tscompile', function () {

return gulp.src(['./dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(ts({

target: 'ES5',

declarationFiles: false,

noExternalResolve: true

}))

.pipe(rename({ extname: '.js' }))

.pipe(gulp.dest('dist/./'));

});

**And add the new task to your default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

, 'grunt-merge-json:menu', 'jshint', 'tscompile'],

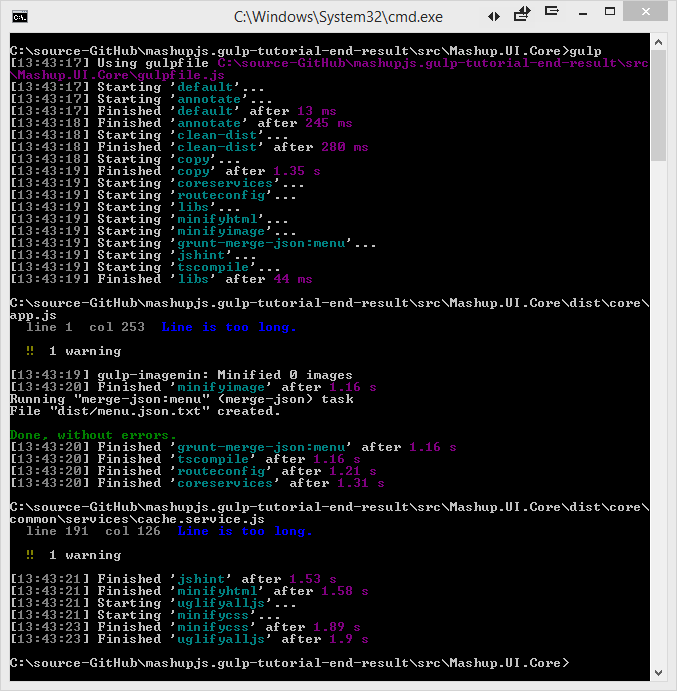
['uglifyalljs', 'minifycss']);

});

**Executing the task**

You can execute the task individually

gulp



**How it works**

First the task transpiles the TypeScript code down to ES5, ECMAScript 5.

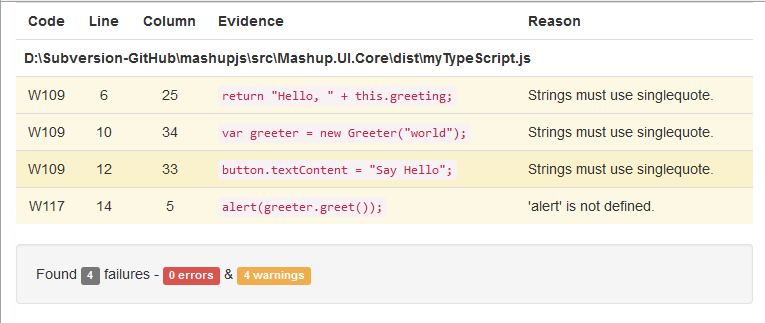
Then the new ES5 JavaScript is emitted. This file will not be used by the min.js.map. The map file will point directly to the TypeScript, “.ts”, file. The reason we are emitting the “.js” is so the TypeScript code can participate in the JSHint process. You’ll notice the ‘jshint’ task now has a dependency on the ‘tscompile’ task via the runSequence function.

Finally the TypeScript file is emitted as a minified JavaScript file with an associated map file linking it back to the TypeScript file.

**jshint-output.html**

If you run the new task with gulp tscompile, you’ll notice the jshint-output.html file has been updated.

If you double-click this file from explorer.exe, then you’ll notice our JavaScript, emitted by TypeScript, has a few issues.



**Fixing the JSHint warnings**

These are pretty simple changes to make. Replace all the double quotes with single quotes and define the alert() method and JSHint is satisfied. The end result should be this.

**/\*global alert \*/**

class Greeter {

greeting: string;

constructor(message: string) {

this.greeting = message;

}

greet() {

return 'Hello, ' + this.greeting;

}

}

var greeter = new Greeter('world');

var button = document.createElement('button');

button.textContent = 'Say Hello';

button.onclick = function () {

alert(greeter.greet());

}

document.body.appendChild(button);

**Linting TypeScript**TypeScript has language constructs ES5 JavaScript does next so automating a TypeScript specific linter might seem redundant. It’s not.

Add the following task to your gulpfile.js

gulp.task('tslint', ['copy'], function () {

return gulp.src(['./dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(tslint())

.pipe(tslint.report('verbose', {

emitError: false,

sort: true,

bell: true

}))

});

**TSLint reporter**

I was unable to find a plugin to export TSLint errors to an HTML file as we did for JSHint. If this plugin becomes available, I’ll add it to this post.

**TSLint configuration**

Configuration information for TSLint is stored in a file named “tslint.json”.

Here is a good starting point. I’ve borrowed this from other developer posts and it seems to be a good list.

Create a tslint.json file in the same folder as your gulpfile.js and package.json and add the following code to it.

{

"rules": {

"class-name": true,

"curly": true,

"eofline": false,

"forin": true,

"indent": [true, 4],

"label-position": true,

"label-undefined": true,

"max-line-length": [true, 140],

"no-arg": true,

"no-bitwise": true,

"no-console": [true,

"debug",

"info",

"time",

"timeEnd",

"trace"

],

"no-construct": true,

"no-debugger": true,

"no-duplicate-key": true,

"no-duplicate-variable": true,

"no-empty": true,

"no-eval": true,

"no-imports": true,

"no-string-literal": false,

"no-trailing-comma": true,

"no-trailing-whitespace": true,

"no-unused-variable": false,

"no-unreachable": true,

"no-use-before-declare": true,

"one-line": [true,

"check-open-brace",

"check-catch",

"check-else",

"check-whitespace"

],

"quotemark": [true, "single"],

"radix": true,

"semicolon": true,

"triple-equals": [true, "allow-null-check"],

"variable-name": false,

"whitespace": [true,

"check-branch",

"check-decl",

"check-operator",

"check-separator"

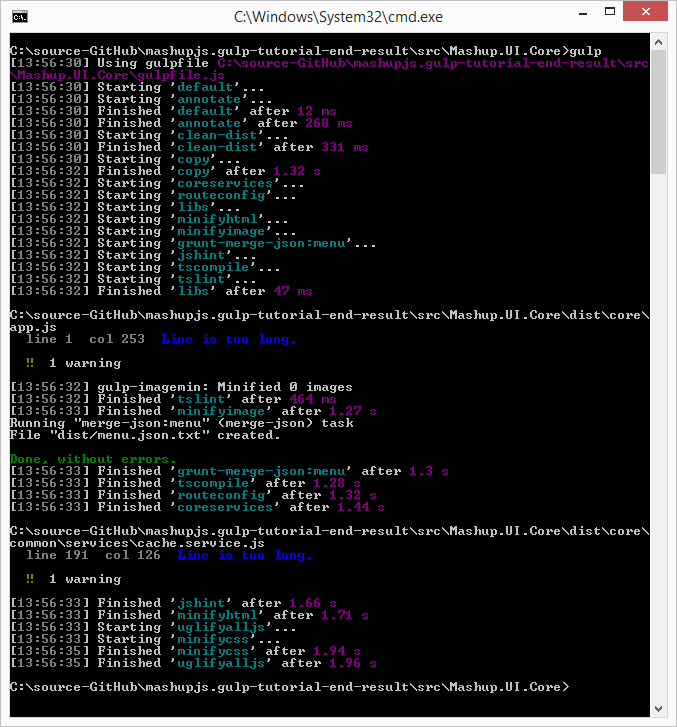
]

}

}

**Run the default task**

gulp



**For more information**

<https://www.npmjs.com/package/gulp-typescript>

## GULP Tutorial Part 13 – SASS

For your SASS implementation, we will use the Gulp-SASS plugin.

**From the command-line install**

npm install gulp-sass --save-dev

**Add the module to the Gulp file**

, sass = require('gulp-sass')

**Add the task to the Gulp file**

gulp.task('sass', function () {

gulp.src('./dist/\*\*/\*.scss', { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sass())

.pipe(gulp.dest('dist/./'));

});

**Add the new task to the default task**

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

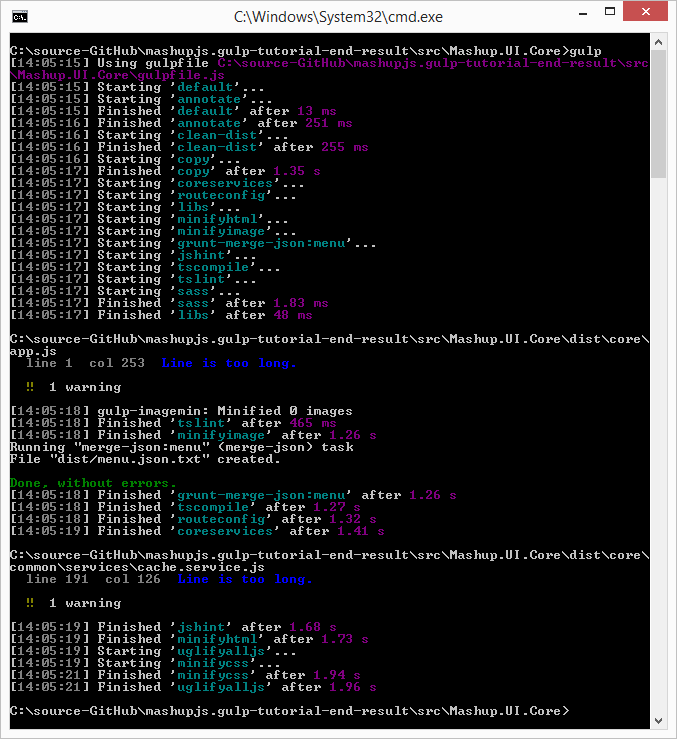
, 'grunt-merge-json:menu', 'jshint', 'tscompile', 'tslint'**, 'sass'**],

['uglifyalljs', 'minifycss']);

});

**Run the default task**

gulp



Learn more about SASS here  
<http://sass-lang.com/>

**The current state of code**

We are starting to build up a nice collection of files. Here is the state the code should be in. If you’re having trouble, then use this to level set yourself so you can move forward.

**package.json**

{

"name": "Mashup.UI.Core",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC",

"devDependencies": {

"grunt": "^0.4.5",

"grunt-merge-json": "^0.9.5",

"gulp": "^3.9.0",

"gulp-clean": "^0.3.1",

"gulp-concat": "^2.5.2",

"gulp-grunt": "^0.5.2",

"gulp-imagemin": "^2.2.1",

"gulp-jshint": "^1.11.0",

"gulp-jshint-html-reporter": "^0.1.2",

"gulp-minify-css": "^1.1.6",

"gulp-minify-html": "^1.0.3",

"gulp-newer": "^0.5.1",

"gulp-ng-annotate": "^1.0.0",

"gulp-plumber": "^1.0.1",

"gulp-rename": "^1.2.2",

"gulp-sass": "^2.0.1",

"gulp-sourcemaps": "^1.5.2",

"gulp-tslint": "^3.0.2-beta",

"gulp-tslint-stylish": "^1.0.1",

"gulp-typescript": "^2.7.7",

"gulp-uglify": "^1.2.0",

"imagemin-pngquant": "^4.1.0",

"jshint-stylish": "^2.0.0",

"run-sequence": "^1.1.1"

}

}

**gulpfile.js**

var onError = function (err) {

console.log(err);

};

var gulp = require('gulp')

, uglify = require('gulp-uglify')

, rename = require('gulp-rename')

, sourcemaps = require('gulp-sourcemaps')

, runSequence = require('run-sequence')

, plumber = require('gulp-plumber')

, ngAnnotate = require('gulp-ng-annotate')

, clean = require('gulp-clean')

, newer = require('gulp-newer')

, concat = require('gulp-concat')

, rename = require('gulp-rename')

, uglify = require('gulp-uglify')

, sourcemaps = require('gulp-sourcemaps')

, minifycss = require('gulp-minify-css')

, minifyhtml = require('gulp-minify-html')

, imagemin = require('gulp-imagemin')

, pngquant = require('imagemin-pngquant')

, jshint = require('gulp-jshint')

, stylish = require('jshint-stylish')

, jshinthtmlreporter = require('gulp-jshint-html-reporter')

, ts = require('gulp-typescript')

, tslint = require('gulp-tslint')

, tsstylish = require('gulp-tslint-stylish')

, sass = require('gulp-sass')

;

gulp.task('annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!src/\*\*/\*.min.js'], { base: 'src/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

gulp.task('clean-dist', function () {

return gulp.src('dist', { read: false })

.pipe(plumber({

errorHandler: onError

}))

.pipe(clean());

});

gulp.task('copy', function () {

return gulp.src('src/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(newer('dist'))

.pipe(gulp.dest('dist'));

});

gulp.task('coreservices', function () {

return gulp.src('src/core/common/\*\*/\*')

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('core.services.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('routeconfig', function () {

return gulp.src(['src/core/config/route.config.js', 'src/apps/\*\*/route.config.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('route.config.js'))

.pipe(gulp.dest('./dist/'));

});

gulp.task('libs', function () {

return gulp.src(['bower\_components/\*\*/\*.js'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(concat('libs.js'))

.pipe(gulp.dest('dist/core/lib/'));

});

gulp.task('uglifyalljs', function () {

//gulp.task('uglifyalljs', ['copy', 'coreservices', 'routeconfig', 'tscompile'], function () {

return gulp.src(['dist/\*\*/\*.js', '!/\*\*/\*.min.js', '!dist/core/lib/\*\*/\*', '!dist/core/common/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

// .pipe(newer('dist/./'))

.pipe(uglify())

.pipe(rename({

extname: '.min.js'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifycss', function () {

return gulp.src(['dist/\*\*/\*.css', '!dist/\*\*/\*.min.css', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(minifycss())

.pipe(rename({

extname: '.min.css'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifyhtml', function () {

return gulp.src(['dist/\*\*/\*.html', '!/\*\*/\*.min.html', '!dist/core/lib/\*\*/\*'], { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(minifyhtml())

.pipe(rename({

extname: '.min.html'

}))

.pipe(sourcemaps.write('./'))

.pipe(gulp.dest('dist/./'));

});

gulp.task('minifyimage', function () {

return gulp.src(['dist/\*\*/\*.{png,jpg,gif,ico}', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(imagemin({ progressive: true, optimizationLevel: 7, use: [pngquant()] }))

.pipe(gulp.dest('dist/./'));

});

// -------------------------------------------------

// Grunt configuration

require('gulp-grunt')(gulp, {

// These are the default options but included here for readability.

base: null,

prefix: 'grunt-',

verbose: false

});

// -------------------------------------------------

gulp.task('jshint', function () {

//gulp.task('jshint', ['copy', 'tscompile'], function () {

return gulp.src(['./dist/\*\*/\*.js', '!dist/core/lib/\*\*/\*.\*', '!\*\*/\*.min.js', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(jshint('.jshintrc'))

.pipe(jshint.reporter(stylish))

.pipe(jshint.reporter('gulp-jshint-html-reporter', { filename: 'jshint-output.html' }))

;

});

gulp.task('tscompile', function () {

return gulp.src(['./dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(sourcemaps.init())

.pipe(ts({

target: 'ES5',

declarationFiles: false,

noExternalResolve: true

}))

.pipe(rename({ extname: '.js' }))

.pipe(gulp.dest('dist/./'));

});

gulp.task('tslint', function () {

return gulp.src(['./dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'])

.pipe(plumber({

errorHandler: onError

}))

.pipe(tslint())

.pipe(tslint.report('verbose', {

emitError: false,

sort: true,

bell: true

}));

});

gulp.task('sass', function () {

gulp.src('./dist/\*\*/\*.scss', { base: 'dist/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(sass())

.pipe(gulp.dest('dist/./'));

});

// ----------------------------------------------------------------

// Default Task

// ----------------------------------------------------------------

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

, 'grunt-merge-json:menu', 'jshint', 'tscompile', 'tslint', 'sass'],

['uglifyalljs', 'minifycss']);

});

**Gruntfile.js**

module.exports = function (grunt) {

grunt.initConfig({

distFolder: 'dist',

pkg: grunt.file.readJSON('package.json'),

"merge-json": {

menu: {

src: ['src/apps/\*\*/menu.json.txt'],

dest: '<%= distFolder %>/menu.json.txt',

},

},

});

// Load modules, register tasks

grunt.loadNpmTasks('grunt-merge-json');

};

**.jshintrc**

{

"bitwise": true,

"camelcase": true,

"curly": true,

"eqeqeq": true,

"es3": false,

"forin": true,

"freeze": true,

"immed": true,

"indent": 4,

"latedef": "nofunc",

"newcap": true,

"noarg": true,

"noempty": true,

"nonbsp": true,

"nonew": true,

"plusplus": false,

"quotmark": "single",

"undef": true,

"unused": false,

"strict": false,

"maxparams": 10,

"maxdepth": 5,

"maxstatements": 40,

"maxcomplexity": 8,

"maxlen": 120,

"asi": false,

"boss": false,

"debug": false,

"eqnull": true,

"esnext": false,

"evil": false,

"expr": false,

"funcscope": false,

"globalstrict": false,

"iterator": false,

"lastsemic": false,

"laxbreak": false,

"laxcomma": false,

"loopfunc": true,

"maxerr": false,

"moz": false,

"multistr": false,

"notypeof": false,

"proto": false,

"scripturl": false,

"shadow": false,

"sub": true,

"supernew": false,

"validthis": false,

"noyield": false,

"browser": true,

"node": true,

"globals": {

"angular": false,

"$": false

}

}

**tslint.json**

{

"rules": {

"class-name": true,

"curly": true,

"eofline": false,

"forin": true,

"indent": [true, 4],

"label-position": true,

"label-undefined": true,

"max-line-length": [true, 140],

"no-arg": true,

"no-bitwise": true,

"no-console": [true,

"debug",

"info",

"time",

"timeEnd",

"trace"

],

"no-construct": true,

"no-debugger": true,

"no-duplicate-key": true,

"no-duplicate-variable": true,

"no-empty": true,

"no-eval": true,

"no-imports": true,

"no-string-literal": false,

"no-trailing-comma": true,

"no-trailing-whitespace": true,

"no-unused-variable": false,

"no-unreachable": true,

"no-use-before-declare": true,

"one-line": [true,

"check-open-brace",

"check-catch",

"check-else",

"check-whitespace"

],

"quotemark": [true, "single"],

"radix": true,

"semicolon": true,

"triple-equals": [true, "allow-null-check"],

"variable-name": false,

"whitespace": [true,

"check-branch",

"check-decl",

"check-operator",

"check-separator"

]

}

}

## GULP Tutorial Part 14 – Watch

Running the default task for Gulp, with all our tasks included, will consume more CPU and time than is required during development where files are changed one at a time. For dealing with individual files as they change, we can use the Watch plugin.

When the Watch is triggered, gulp.watch tasks execute.

**From the command-line install**

npm install gulp-watch --save-dev

**Add the module to the Gulp file**

, watch = require('gulp-watch')

**Add the task to the Gulp file**

The “watch” task is large and uses many other tasks, so I place it at the bottom of the file so it is separate.

gulp.task('watch', function () {

// ---------------------------------------------------------------

// Watching JS files

// ---------------------------------------------------------------

// Copy all files except \*.js files.

gulp.watch(['src/\*\*/\*', '!src/\*\*/\*.js', '!bower\_components/\*\*.\*'], function () { runSequence('copy'); });

// Annotates and copies \*.js files

gulp.watch(['src/\*\*/\*.js',

'!src/core/config/route.config.js', '!src/apps/\*\*/route.config.js',

'!bower\_components/\*\*/\*.js'], function () { runSequence('watch:annotate', 'copy'); });

// routeConfig file changes.

gulp.watch(['src/core/config/route.config.js', 'src/apps/\*\*/route.config.js'], function () { runSequence('routeconfig'); });

// Uglify JS files

gulp.watch(['dist/\*\*/\*.js', '!dist/\*\*/\*.min.js', '!dist/core/lib/\*\*/\*', '!dist/core/common/\*\*/\*'], function () { runSequence('uglifyalljs'); });

// ---------------------------------------------------------------

// Watching Bower components

// ---------------------------------------------------------------

gulp.watch(['bower\_components/\*\*/\*.js'], function () { runSequence('libs'); });

// TODO: Add other bower component types like css, scss and images

// ---------------------------------------------------------------

// Watching css and scss files

// ---------------------------------------------------------------

gulp.watch(['dist/\*\*/\*.css', '!dist/\*\*/\*.min.css', '!dist/core/lib/\*\*/\*'], function () { runSequence('minifycss'); });

gulp.watch(['dist/\*\*/\*.scss', '!dist/core/lib/\*\*/\*'], function () { runSequence('sass'); });

// ---------------------------------------------------------------

// Watching TypeScript files

// ---------------------------------------------------------------

gulp.watch(['dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'], function () { runSequence('tscompile'); });

// ---------------------------------------------------------------

// Watch - Execute linters

// ---------------------------------------------------------------

gulp.watch(['dist/\*\*/\*.ts', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'], function () { runSequence('tslint'); });

//gulp.watch(['dist/\*\*/\*.js', '!dist/core/lib/\*\*/\*.\*', '!dist/\*\*/\*.min.js', '!dist/core/css/\*\*/\*.\*'], function() { runSequence('jshint'); });

gulp.watch(['dist/\*\*/\*.js', '!dist/core/lib/\*\*/\*.\*', '!dist/\*\*/\*.min.js', '!dist/core/css/\*\*/\*.\*'], ['jshint']);

// ---------------------------------------------------------------

// Watching image files

// ---------------------------------------------------------------

// unable to get this watch to ever notice a file changed. This will be handled on the initial build.

//gulp.watch(['dist/\*\*/\*.{png,jpg,gif,ico}', '!dist/core/lib/\*\*/\*.\*', '!dist/core/css/\*\*/\*.\*'], function() { runSequence('minifyimage'); });

});

**Add another task to the gulp file**

Also add this supporting Watch task. This helps improve performance with the “newer” plugin.

// ---------------------------------------------------------------

// Watch specific tasks. This is to support the use of newer.

// ---------------------------------------------------------------

gulp.task('watch:annotate', function () {

return gulp.src(['src/index.controller.js', 'src/core/\*\*/\*.js', 'src/apps/\*\*/\*.js', '!src/core/lib/\*\*/\*', '!/\*\*/\*.min.js'], { base: 'src/./' })

.pipe(plumber({

errorHandler: onError

}))

.pipe(newer('src/./'))

.pipe(ngAnnotate())

.pipe(gulp.dest('src/./'));

});

**Add the new task to the default task**

Add the new “watch” task to the default task. The default task will execute all the tasks in the sequence specified by the runSequence function. The last task run is the Watch task which contains 10 individual Watch tasks. This time, when you run the Gulp default task, the command line will not return control to you. To break out of this, press CTRL + C and then answer the prompt with “y”.

gulp.task('default', function () {

runSequence('annotate', 'clean-dist', 'copy',

['coreservices', 'routeconfig', 'libs', 'minifyhtml', 'minifyimage'

, 'grunt-merge-json:menu', 'jshint', 'tscompile', 'tslint', 'sass']

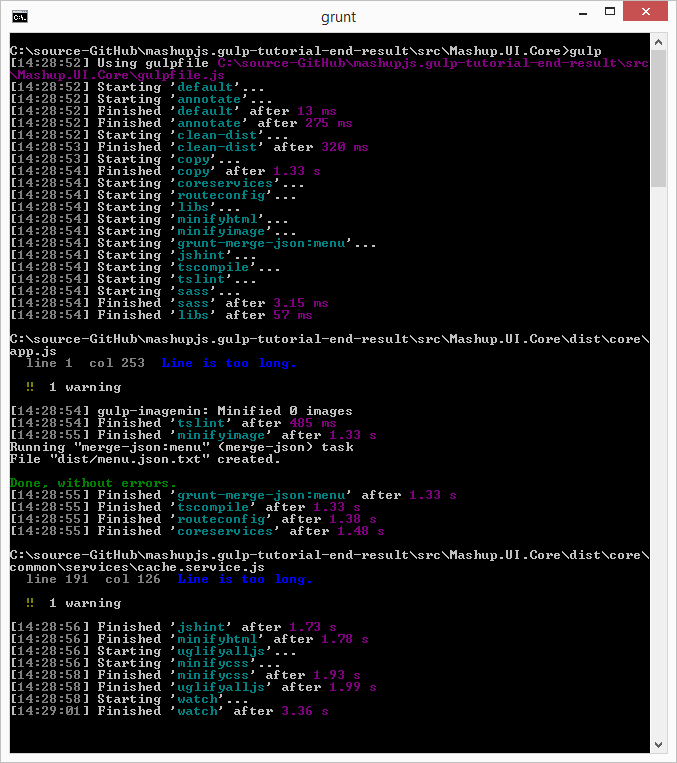
, ['uglifyalljs', 'minifycss']

**,'watch'**);

});

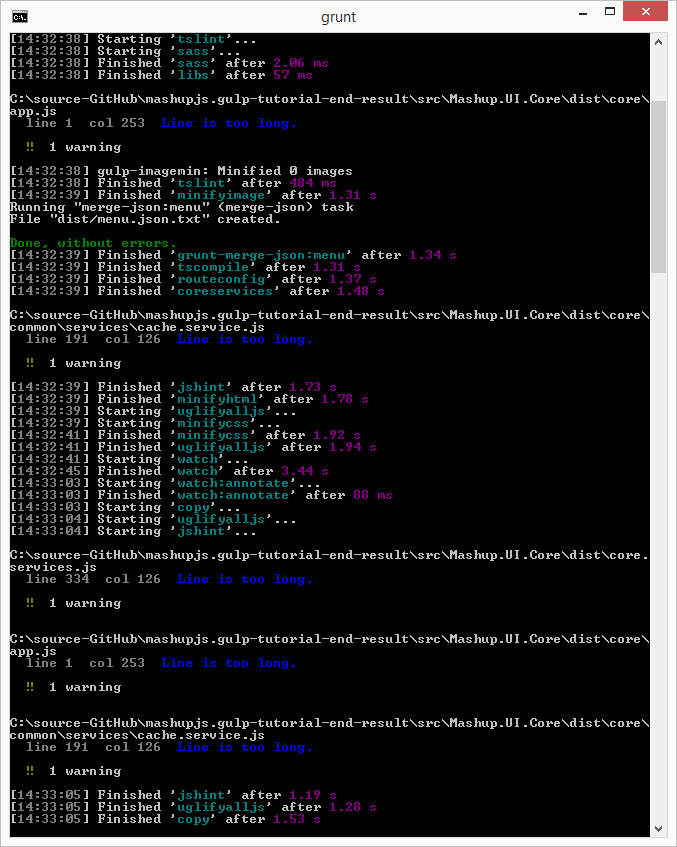
**Run the default task**

gulp



If you modify any files the Watch is configured to watch, then you’ll see tasks run.

Here is what happens after changing the index.controller.js file.



First the annotate task runs against the changed JavaScript code. Then the file is copied to the dist folder and minified by the uglifyalljs task. Finally the JavaScript code is linted with the jshint task.

**Tutorial end result**

The completed code for this tutorial can be found here

<https://github.com/MashupJS/gulp-tutorial-end-result>

## GULP Tutorial Part ? – Setting up environments

Most applications are expected to work in multiple environments. When coding you expect the application to function in the local environment. When you deploy to the test environment you expect it to function there and the same for production.

We will use Gulp to set an Angular constant used to build the connection URI to each WebApi.

Create a json file representing each possible environment in the “src” folder.

env.config.localdev.json

env.config.dev.json

env.config.stage.json

env.config.prod.json

In each file will be the root connection string for each WebApi used.

Here is the content of the env.config.localdev.json.

{

"myfirstApi": "http://localhost:52335",

"mysecondApi": "http://localhost:52336",

}

**From the command-line install**

npm install gulp-replace-task --save-dev

npm install yargs --save-dev

npm install fs --save-dev

**Add the module to the Gulp file**

replace = require('gulp-replace-task'),  
 args = require('yargs').argv,  
 fs = require('fs')

**Add the task to the Gulp file**

This will be different for your environment.

gulp.task('replace', function () {

// Get the environment from the command line

var env = args.env || 'localdev';

// Read the settings from the right file

var filename = 'env.config.' + env + '.json';

var settings = JSON.parse(fs.readFileSync('dist/' + filename, 'utf8'));

// Replace each placeholder with the correct value for the variable.

gulp.src('src/app.js')

.pipe(replace({

patterns: [

{

match: 'qualityApi',

replacement: settings.qualityApi

}

]

}))

.pipe(gulp.dest('dist/./'));

});

**You can execute the task individually**

The first option uses the prod configuration. The second uses the default which happens to be the localdev configuration.

Gulp replace --env prod

Or

Gulp replace

**Add to the default task**

I’ve added the replace task to the early part of the default configure so it executes before the minification task is run.

Now I can simply type Gulp and all my tasks, including the Replace task will run.

gulp

## GULP Tutorial Part 15 – Useful Gulp Commands & Tips

**Installing Gulp**

Execute both of these. The first adds Gulp locally so it can be used by NPM. The second installs Gulp globally so it can be accessed from the command line.

npm install gulp --save-dev

npm install gulp -g

**Retrieve Gulp version**

Grunt -version

**Installing plugins**

The syntax for Grunt plugins is

install [plugin-name] --save-dev

For example, if you want to minify and concatenate your JavaScript for performance, you would install two plugins.

Perform a quick Google search and you’ll find this site

<https://github.com/gruntjs/grunt-contrib-uglify>

npm install grunt-contrib-uglify --save-dev

Perform a quick google search and you’ll find this site

<https://github.com/gruntjs/grunt-contrib-concat>

npm install grunt-contrib-concat --save-dev

**Retrieve Gulp version**

Gulp --v

**Every Gulp file needs a default task. To execute Gulp’s default task**

grunt

**It’s useful to run specific tasks that you have configured**

grunt [task-name]

**Get a list of Grunt commands**

Grunt –help

**To verify a plugin is not blacklisted**

Gulp --verify

**Testing tasks while building your gulpfile.js**

You can type gulp [task-name] and your task will run. If it has any dependencies then those dependencies will run first.

gulp [task-name]

## GULP Tutorial Part 16 – Glob Tips

Here are some of the more common Glob patterns.

“dir/\*” – includes all files

“dir/\*\*” – includes all files and folders

“\*\*/\*.js” – all JavaScript files

“!\*\*/\*.min.js” – excludes all minified JavaScript files

“!app/lib/\*\*/\*” – excludes all files in the lib folder.

<https://github.com/isaacs/node-glob>

<http://mywiki.wooledge.org/glob#preview>

## GULP Tutorial Part 17 – Useful NPM Packages/Commands

### Commands Cheat Sheet

**Find outdated modules**

npm outdated -–depth=0

npm outdated –-json -–depth=0

**Installing a package**

npm install grunt-contrib-uglify@\* --save-dev

**Updating local packages**

When someone has added modules since your last check-in just run.

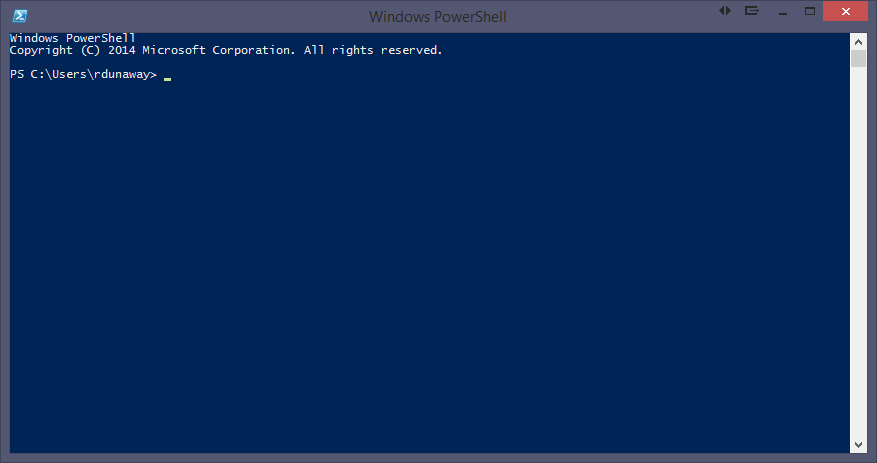
npm update

### PowerShell (primer)

Windows users can use either the Command Prompt or PowerShell.

PowerShell is pre-installed on Windows 8 or newer installations.

From Start: *Search programs and files* type “powershell”. Select “powershell.exe”.



### Syntax

Powershell Command Syntax: **application action –flags arguments**

For help with any application add the **–h** or **–help** flags for additional instructions.

The **tab** key autocompletes your statement.

### Adding and removing files

To create a new item use the **ni** command. This might not seem useful with Visual Studio 2013 because any file added must also be added to your project file. Visual Studio 2015 does not have a project file needing updates. Instead a Glob pattern is used to determine what files should and should not be included in the project. That being the case, suddenly, **ni** makes more sense.

**Adding files**

ni newjsfile.js -type file

new-item newjsfile.js –type file

**Removing files**

ri newjsfile.js or remove-item newjsfile.js

### Installing NodeJS and NPM Packages

Install NodeJS from:

<https://nodejs.org/>

Install NPM packages with the following syntax

npm install [global option –g] [package-name] [options]

Example: (You need to install Gulp both locally and globally)

npm install gulp --save-dev

npm install gulp –g

<https://docs.npmjs.com/getting-started/installing-npm-packages-locally>

### NPM Version updates

There are multiple options for keeping NPM packages up to date. The approach you choose might depend on your development workflow and automated testing solution, i.e., if you have good automated testing, it might be safe to allow the latest versions. If not, you might want to choose a more deliberate approach to NPM versioning.

#### Version updates: Option 1 – Using node tools

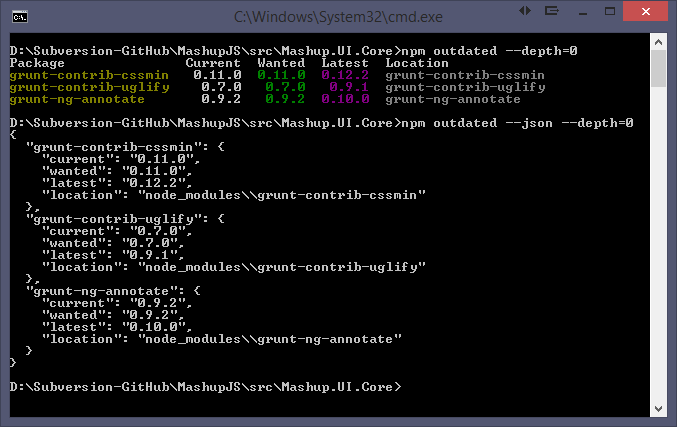
**Check to see which NPM packages are out of date**

Display colored rows

npm outdated -–depth=0

Display in json which includes current, wanted, latest version numbers

npm outdated –-json -–depth=0



Note: Not all your packages will be displayed. Only the outdated packages will be displayed.

Note: If you modify the command to include “-g” then you’ll get a list of your outdated global packages.

**To update packages one at a time**

npm install [package-name]@\* [save?]

npm install grunt-contrib-uglify@\* --save-dev

#### Version updates: Option 2 – Using npm-check-updates

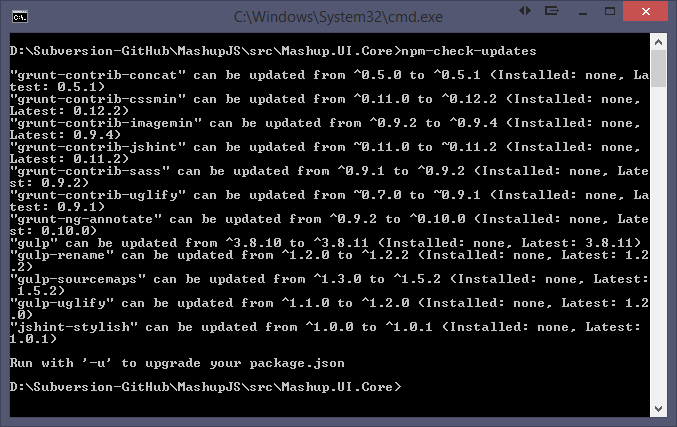
Using the npm-check-updates package, you can keep all your packages updated.

<https://www.npmjs.com/package/npm-check-updates>

npm install -g npm-check-updates

Then execute the following command to see what packages can be updated.

npm-check-updates



**To upgrade all your packages**

npm-check-updates –u [-g option for global packages]

Now your package.json is updated.

Then execute an NPM install to update the package installations.

npm install [-g option for global packages]

#### npm versioning semantics

<https://docs.npmjs.com/misc/semver>

<http://semver.org/>

The End