

## 5 Minutes to Min-Safe Angular Code with Grunt

By [Thomas Greco \(https://www.sitepoint.com/author/tgreco/\)](https://www.sitepoint.com/author/tgreco/) July 23, 2015

Optimizing page speed is undoubtedly a primary focus for any developer building web applications. Task runners such as [Grunt \(http://gruntjs.com/\)](http://gruntjs.com/) can play a pivotal role in the development process as they automate the activities of code concatenation and minification, which will be the main topics of this tutorial. Specifically, we're going to use a set of Grunt plugins that will ensure our [AngularJS \(https://angularjs.org/\)](https://angularjs.org/) application is safe for minification. Before I begin to discuss about AngularJS and minification, I want to highlight that developers of all skill levels can benefit from this tutorial, however basic knowledge of Grunt is desirable. In this article, we'll be generating new folders with Grunt, so those new to using task runners will get a nice feel for how things work.

## The Problem with Minifying Angular Applications

AngularJS applications are not min-safe by default. They must be written using the array syntax. Don't worry if you're confused as to what the array syntax exactly is, you have probably already written code that utilizes it. Let's take a look at two examples of AngularJS controllers that are being passed the `$scope` and `$http` parameters.

In the first example below, the module's factory and controller are wrapped in arrays that begin with DI annotations, and as you can see it does not follow the [DRY \(Don't Repeat Yourself\) principle \(https://en.wikipedia.org/wiki/Don%27t\\_repeat\\_yourself\)](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself).

```
var form = angular.module('ControllerOne', [])
form.factory('Users', ['$http', function($http) {
  return {
    get: function() {
      return $http.get('/api/users');
    },
    create: function(userData) {
      return $http.post('/api/users', userData);
    },
    delete: function(id) {
      return $http.delete('/api/users/' + id);
    }
  };
}]);

form.controller('InputController', ['$scope', '$http', 'Users', function($scope, $http, Users) {
  formData = {};
  $scope.createUser = function () {
    if ($scope.formData !== undefined) {
      Users.create($scope.formData)
        .success(function (data) {
          $scope.users = data;
          $scope.formData = {};
          $scope.myForm.$setPristine(true);
        });
    }
  };
}]);
```

In the next example, the `crud.config` module code is still not min-safe, but the code is shorter than the previous one. It simply names the services and then passes the necessary dependencies into the function as parameters, without having to first write them out as strings. This code will run just fine, as long as it is not minified. So, it's easy to see why people often choose this syntax when writing AngularJS code.

```

var form = angular.module('ControllerTwo', [])
form.factory('Users', function($http) {
  return {
    get: function() {
      return $http.get('/api/users');
    },
    create: function(userData) {
      return $http.post('/api/users', userData);
    },
    delete: function(id) {
      return $http.delete('/api/users/' + id);
    }
  };
});

form.controller('InputController', function($scope, $http, Users) {
  formData = {};
  $scope.createUser = function() {
    if ($scope.formData != undefined) {
      Users.create($scope.formData)
      .success(function(data) {
        $scope.users = data;
        $scope.formData = {};
        $scope.myForm.$setPristine(true);
      });
    }
  };
});

```

Now that you have learned the physical differences between the two codes, I'll quickly explain to you why this syntax is not safe for minification.

## How the Array Notation Works

As I stated above, the array notation begins with DI annotations, which play a pivotal role in making this code min-safe. When [UglifyJS](https://github.com/mishoo/UglifyJS) (<https://github.com/mishoo/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.

If these annotations aren't present, the code will break. As you can see, it's extremely inefficient to manually write code in such a manner. To help you avoid it, I am now going to show how to employ Grunt to annotate, concatenate, and minify your AngularJS applications in a way that they are fully optimized in regards to minification and ready for use in production.

## Using Grunt

The entire repository for the project, including the files we will be targeting, can be [found on GitHub \(https://github.com/sitepoint-editors/SpNgAnnotate\)](https://github.com/sitepoint-editors/SpNgAnnotate). For those who are comfortable using Grunt, feel free to follow along and create your own build, or add this code to an existing project. If you are working from an empty directory, you've to ensure that you have a "package.json" file in your directory. This file can be created by running the command `npm init`.

Once you have a "package.json" file in your project, you can download the plugins by running the following command:

```
npm install grunt-contrib-concat grunt-contrib-uglify grunt-ng-annotate --save-dev
```

This will install Grunt into your project, as well as the three plugins we'll be using:

```

grunt-contrib-concat
grunt-contrib-uglify
grunt-ng-annotate

```

Although `ng-annotate` can be used without Grunt, you'll soon see how seamless Grunt makes the process of annotating, concatenating, and minifying code. It offers a simple, yet effective solution to minify AngularJS code. If you've been following this project from scratch, you should have a Gruntfile.js in the root directory of your project, which will hold all of your Grunt code. If you haven't already, create it now.

# Three Steps to Min-Safe Code

## Step 1 – Configure Grunt to Read the “package.json” file

To access the plugins we installed before, you first need to configure the Gruntfile's `pkg` property to read the contents of the “package.json” file. The config object begins immediately at the top of Grunt's wrapper function, and extends from lines 3 to 5 in the example below, but will soon hold the majority of the code.

```
module.exports = function(grunt) {  
  //grunt wrapper function  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    //grunt task configuration will go here  
  });  
}
```

## Step 2 – Load and Register Grunt Tasks

After configuring Grunt to read our “package.json” file, the plugins need to be loaded so that Grunt can access them. This is done by passing the name of the plugin into `grunt.loadNpmTask()` as a string. It's important to make sure that these plugins are loaded *inside* of the wrapper function, but *outside* of the config object. If these conditions aren't met, Grunt won't work correctly.

The next thing we need to do is to create a default task that will be execute when Grunt is invoked without a specific target. You should note the order in which these tasks are added as they will run according to their configuration. Here, `ngAnnotate` is configured to run first, before `concat` and `UglifyJS`, which I believe is the best way to build your code. Additionally, it's important to remember that `grunt.registerTask()` must be placed after the plugins are loaded.

Based on what we've just discussed, the Gruntfile.js should be as follow:

```
module.exports = function(grunt) {  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    //grunt task configuration will go here  
  });  
  
  //load grunt tasks  
  grunt.loadNpmTasks('grunt-contrib-concat');  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.loadNpmTasks('grunt-ng-annotate');  
  
  //register grunt default task  
  grunt.registerTask('default', ['ngAnnotate', 'concat', 'uglify']);  
}
```

## Step 3 – Configuring the plugins

### ngAnnotate

Now that our Gruntfile is ready to go, let's dive back into the config object and specify the files we want to the `ngAnnotate` plugin to target. In order to do this, we have to first create a section for `ngAnnotate` and create a target, which in this case is called `spApp`. Inside this target you'll specify the files that you wish to add the DI annotations to, as well as the folder into which they should be generated. In this example, Grunt will take the three files specified in `public/js`, and generate them into a new folder named `public/min-safe`.

Once you have configured this, you can run `grunt ngAnnotate` and see how the code is generated. Additionally, you can visit [the GitHub page for grunt-ng-annotate \(https://github.com/mzgol/grunt-ng-annotate\)](https://github.com/mzgol/grunt-ng-annotate), and check out the different options it allows you to specify.

```

ngAnnotate: {
  options: {
    singleQuotes: true
  },
  app: {
    files: {
      './public/min-safe/js/appFactory.js': ['./public/js/appFactory.js'],
      './public/min-safe/js/FormController.js': ['./public/js/FormController.js'],
      './public/min-safe/app.js': ['./public/js/app.js']
    }
  }
}

```

## Concatenation

Now that you have generated a folder filled with newly annotated AngularJS code, let's move ahead by compiling, or concatenating, this code into one single file. In the same way that we created a section for `ngAnnotate`, we'll now do the same for `concat` (<https://github.com/gruntjs/grunt-contrib-concat>) and `UglifyJS` (<https://github.com/gruntjs/grunt-contrib-uglify>). Just like `ngAnnotate`, both of these tasks take a target, which in this case is `js`. There are a number of configuration options that can be passed into these tasks but we're simply going to specify the `src` and `dest` to point to the correct files. As you may have guessed, these plugins will take the contents of the files passed into `src` object, and process them into the folder specified after `dest`.

(/)

Let's try to understand what's going on here. You can test this out by simply running `grunt concat` in your terminal, and it should result in the creation of `./public/min/app.js`.

```

concat: {
  js: { //target
    src: ['./public/min-safe/app.js', './public/min-safe/js/*.js'],
    dest: './public/min/app.js'
  }
}

```

## Minification

The last thing we need to do is remove the useless space from our code by minifying it. This is where the UglifyJS plugin comes into play. When working with UglifyJS, we want Grunt to complete the final process of minifying our application. Therefore, we want to target the file which holds all of our newly concatenated code, which in this case is `public/min/app.js`. To test this out, run `grunt uglify`, and get a look at your newly minified.

Here is the relative configuration for this task:

```



uglify: {
  js: { //target
    src: ['./public/min/app.js'],
    dest: './public/min/app.js'
  }
}

```

During this lesson we used all of these tasks separately. Now, let's employ the default task we created before. It'll allow Grunt to run all the tasks specified immediately after one another, in the order they are registered. Now, your code will be annotated, concatenated, and minified by simply running `grunt` inside your project.

## Conclusions

I hope that thanks to this short tutorial you have a good understanding of the array notation, and why it is essential for making AngularJS apps min-safe. If you are new to Grunt, I highly encourage you to play around with these plugins, as well as other ones, because of the huge amount of time they can save you. As always, feel free to comment below, or email me at the address in my bio if you have any questions.

Was this helpful?  

More: [angular](https://www.sitepoint.com/tag/angular/) (<https://www.sitepoint.com/tag/angular/>), [angularjs](https://www.sitepoint.com/tag/angularjs-2/) (<https://www.sitepoint.com/tag/angularjs-2/>), [concatenation](https://www.sitepoint.com/tag/concatenation/) (<https://www.sitepoint.com/tag/concatenation/>), [Grunt](https://www.sitepoint.com/tag/grunt/) (<https://www.sitepoint.com/tag/grunt/>), [grunt plugin](#)