Chapter 27: TSLint - assuring code quality and consistency

TSLint performs static analysis of code and detect errors and potential problems in code.

Section 27.1: Configuration for fewer programming errors

This tslint.json example contains a set of configuration to enforce more typings, catch common errors or otherwise confusing constructs that are prone to producing bugs and following more the Coding Guidelines for TypeScript Contributors.

To enforce this rules, include tslint in your build process and check your code before compiling it with tsc.

```
{
  "rules": {
     // TypeScript Specific
     "member-access": true, // Requires explicit visibility declarations for class members.
     "no-any": true, // Disallows usages of any as a type declaration.
     // Functionality
     "label-position": true, // Only allows labels in sensible locations.
     "no-bitwise": true, // Disallows bitwise operators.
     "no-eval": true, // Disallows eval function invocations.
     "no-null-keyword": true, // Disallows use of the null keyword literal.
     "no-unsafe-finally": true, // Disallows control flow statements, such as return, continue,
break and throws in finally blocks.
     "no-var-keyword": true, // Disallows usage of the var keyword.
     "radix": true, // Requires the radix parameter to be specified when calling parseInt.
     "triple-equals": true, // Requires === and !== in place of == and !=.
     "use-isnan": true, // Enforces use of the isNaN() function to check for NaN references instead
of a comparison to the NaN constant.
     // Style
     "class-name": true, // Enforces PascalCased class and interface names.
     "interface-name": [ true, "never-prefix" ], // Requires interface names to begin with a capital
'I'
     "no-angle-bracket-type-assertion": true, // Requires the use of as Type for type assertions
instead of <Type>.
     "one-variable-per-declaration": true, // Disallows multiple variable definitions in the same
declaration statement.
     "quotemark": [ true, "double", "avoid-escape" ], // Requires double quotes for string literals.
     "semicolon": [ true, "always" ], // Enforces consistent semicolon usage at the end of every
statement.
     "variable-name": [true, "ban-keywords", "check-format", "allow-leading-underscore"] // Checks
variable names for various errors. Disallows the use of certain TypeScript keywords (any, Number,
number, String, string, Boolean, boolean, undefined) as variable or parameter. Allows only camelCased
or UPPER_CASED variable names. Allows underscores at the beginning (only has an effect if "check-
format" specified).
  }
}
```

Section 27.2: Installation and setup

To install tslint run command

```
npm install -g tslint
```

Tslint is configured via file `tslint.json`. To initialize default configuration run command

```
tslint --init
```

To check file for possible errors in file run command

```
tslint filename.ts
```

Section 27.3: Sets of TSLint Rules

- tslint-microsoft-contrib
- tslint-eslint-rules
- codelyzer

Yeoman generator supports all these presets and can be extends also:

- generator-tslint

Section 27.4: Basic tslint.json setup

This is a basic `tslint.json` setup which

- prevents use of `any`
- requires curly braces for `if`/`else`/`for`/`do`/`while` statements
- requires double quotes (`"`) to be used for strings

```
{
    "rules": {
        "no-any": true,
        "curly": true,
        "quotemark": [true, "double"]
    }
}
```

Section 27.5: Using a predefined ruleset as default

`tslint` can extend an existing rule set and is shipped with the defaults `tslint:recommended` and `tslint:latest`.

> `tslint:recommended` is a stable, somewhat opinionated set of rules which we encourage for general TypeScript programming. This configuration follows semver, so it will not have breaking changes across minor or patch releases.
>
> `tslint:latest` extends tslint:recommended and is continuously updated to include configuration for the latest rules in every TSLint release. Using this config may introduce breaking changes across minor releases as new rules are enabled which cause lint failures in your code. When TSLint reaches a major version bump, tslint:recommended will be updated to be identical to tslint:latest.

Docs and source code of predefined ruleset

So one can simply use:

```
{
  "extends": "tslint:recommended"
}
```

to have a sensible starting configuration.

One can then overwrite rules from that preset via `rules`, e.g. for node developers it made sense to set `no-console` to **`false`**:

```
{
  "extends": "tslint:recommended",
  "rules": {
    "no-console": false
  }
}
```