**JavaScript Style Guide**

Contents

1. 소개

1.1 용어 참고

1.2 가이드 노트

2. 소스 파일 기본 사항

2.1 파일 이름

2.2 파일 인코딩 : UTF – 8

2.3 특수 문자

3. 소스 파일 구조

3.1 라이선스 또는 저작권 정보 (있는 경우)

3.2 @fileoverview JSDoc (있는 경우)

3.3 ES 모듈

3.4 파일의 구현

4. 서식

4.1 교정기

4.2 블록 들여 쓰기 : +2 공백

4.3 진술

4.4 열 제한 : 80

4.5 줄 바꿈

4.6 공백

4.7 그룹화 괄호 : 권장

4.8 댓글

5. 언어 기능

5.1 지역 변수 선언

5.2 배열 리터럴

5.3 객체 리터럴

5.4 클래스

5.5 기능

5.6 문자열 리터럴

5.7 숫자 리터럴

5.8 제어 구조

5.9 this

5.10 평등 검사

5.11 허용되지 않는 기능

6. 명명규칙

6.1 모든 식별자에 공통적인 규칙

6.2 식별자 유형별 규칙

6.3 카멜 케이스

7. JSDoc

1.1 용어 참고

1.2 가이드 노트

8. 정책

8.1 Google 스타일에서 지정하지 않은 문제 : 일관성 유지!

8.2 컴파일러 경고

8.3 지원 중단

8.4 Google 스타일에없는 코드

8.5 지역 스타일 규칙

8.6 생성 된 코드 : 대부분 면제

9. 부록

9.1 JSDoc 태그 참조

9.2 일반적으로 오해되는 스타일 규칙

9.3 스타일 관련 도구

9.4 레거시 플랫폼에 대한 예외

**1. 소개**

이 문서는 JavaScript 프로그래밍 언어의 소스 코드에 대한 코딩 표준을 정의합니다. 다른 프로그래밍 스타일 가이드와 마찬가지로 다루는 문제는 형식화의 미적 문제 뿐 아니라 다른 유형의 규칙이나 코딩 표준에도 적용됩니다. 그러나이 문서는 주로 우리가 보편적으로 따르는 엄격하고 빠른 규칙에 초점을 맞추고 명확하게 시행 할 수없는 조언 (인간 또는 도구)을 제공하지 않습니다.

1.1 용어 참고

이 문서에서 달리 명시되지 않는 한 :

1. 주석 이라는 용어 는 항상 구현 주석을 의미합니다 . Google은 문서 주석 이라는 문구를 사용하지 않고 대신 사람이 읽을 수있는 텍스트와 /\*\* … \*/.

2. 이 스타일 가이드는 must , must not , should , should not , may 라는 문구 를 사용할 때 RFC 2119 용어를 사용합니다 . 조건은 선호 하고 회피 대응 해야 하고 해야하지 , 각각. 명령문 및 선언문은 규범 적이며 must에 해당 합니다 .

문서 전체에 다른 용어 참고 사항 이 가끔 표시됩니다.

1.2 가이드 노트

이 문서의 예제 코드는 **표준**이 **아닙니다** .  즉, 예제는 단순히 스타일이지만 코드를 표현하는 유일한 세련된 방법을 설명하지 못할 수 있습니다.  예제에서 선택한 선택적 형식 지정은 규칙으로 적용되지 않아야합니다.

**2. 소스 파일 기본 사항**

2.1 파일 이름

파일 이름은 모두 소문자 여야하며 밑줄 ( \_) 또는 대시 ( -)를 포함 할 수 있지만 추가 구두점은 없습니다. 프로젝트에서 사용하는 규칙을 따르십시오. 파일 이름의 확장자는 .js

2.2 파일 인코딩 : UTF-8

소스 파일은 **UTF-8** 로 인코딩됩니다.

2.3 특수 문자

2.3.1 공백 문자

줄 종결 자 시퀀스를 제외하고 ASCII 가로 공백 문자 (0x20)는 소스 파일의 모든 위치에 나타나는 유일한 공백 문자입니다. 이것은

- 문자열 리터럴의 다른 모든 공백 문자는 이스케이프됩니다.

- 탭 문자는 들여 쓰기에 사용 되지 않습니다 .

2.3.2 특수 이스케이프 시퀀스

특별한 이스케이프 시퀀스가 있는 모든 문자 ( \', \", \\, \b, \f, \n, \r, \t, \v), 그 시퀀스 (예 : 해당 숫자 탈출보다는 사용 \x0a, \u000a또는 \u{a}). 레거시 8 진 이스케이프는 사용되지 않습니다.

2.3.3 비 ASCII 문자

나머지 비 ASCII 문자의 경우 실제 유니 코드 문자 (예 ∞:) 또는 동등한 16 진수 또는 유니 코드 이스케이프 (예 :) \u221e가 사용됩니다. 이는 코드 를 읽고 이해하기 쉽게 만드는 요소에 따라 다릅니다 .

팁 : 유니 코드 이스케이프 케이스 및 실제 유니 코드 문자가 사용되는 경우에도 설명 주석이 매우 유용 할 수 있습니다.

/ \* 최고 : 댓글이 없어도 완벽하게 선명합니다. \* / const 단위 = 'μs' ;

/ \* 허용 : 그러나 μ는 인쇄 가능한 문자이므로 불필요합니다. \* / const 단위 = '\ u03bcs' ; // 'μs'

/ \* 좋음 : 명확성을 위해 주석과 함께 인쇄 할 수없는 문자에는 이스케이프를 사용합니다. \* / return '\ ufeff' + content ; // 바이트 순서 표시를 앞에 추가합니다.

/ \* 나쁨 : 독자는 이것이 어떤 캐릭터인지 모릅니다. \* / const 단위 = '\ u03bcs' ;

팁 : 일부 프로그램이 비 ASCII 문자를 제대로 처리하지 못할 수 있다는 두려움 때문에 코드의 가독성을 낮추지 마십시오. 이런 일이 발생하면 해당 프로그램이 **중단** 되고 **수정** 해야합니다 .

## 3. 소스 파일 구조

모든 새 소스 파일은 goog.module파일 ( goog.module호출을 포함하는 파일 ) 또는 ECMAScript (ES) 모듈 (사용 import및 export 문) 이어야합니다 . 파일은 다음 순서 로 구성됩니다 .

1. 라이센스 또는 저작권 정보 (있는 경우)

2. @fileoverview JSDoc (있는 경우)

3. ES import문 (ES 모듈 인 경우)

4. 파일의 구현

하나 또는 두 개의 빈 줄이 앞에 올 수있는 파일 구현을 제외하고 정확히 하나의 빈 줄 이 존재하는 각 섹션을 구분합니다.

**3.1 라이선스 또는 저작권 정보 (있는 경우)**

라이선스 또는 저작권 정보가 파일에 속하면 여기에 속합니다.

**3.2 @fileoverviewJSDoc (있는 경우)**

서식 규칙 은 7.5 최상위 / 파일 수준 주석 을 참조하십시오 .

**3.3 import**

import문은 줄 바꿈되어서는 안되므로 80 열 제한에 대한 예외입니다.

3.3.1.1 import 경로

ES 모듈 파일은 import명령문을 사용하여 다른 ES 모듈 파일을 가져와야합니다.

import './sideeffects.js' ;

import \* as goog from '../closure/goog/goog.js' ; import \* as parent from '../parent.js' ;

import { name } from './sibling.js' ;

3.3.1.1.1 import 경로의 파일 확장자

.js파일 확장자는 import경로에서 선택하지 않고 항상 포함해야합니다.

import '../directory/file' ; (x)

import '../directory/file.js' ;

3.3.1.2 동일한 파일을 여러번 import

동일한 파일을 여러번 import하지 마십시오. 이로 인해 파일의 모든 import를 결정하기가 어려울 수 있습니다.

// import는 동일한 경로를 갖지만 정렬되지 않으므로보기 어려울 수 있습니다. import { short } from './long/path/to/a/file.js' ; import { aLongNameThatBreaksAlignment } from './long/path/to/a/file.js' ;

3.3.1.3 이름 지정 import

3.3.1.3.1 이름 지정 모듈 import

모듈 import 이름 지정 ( import \* as name)은 lowerCamelCase가져온 파일 이름에서 파생된 이름입니다.

import \* as fileOne from '../file-one.js' ; import \* as fileTwo from '../file\_two.js' ; import \* as fileThree from '../filethree.js' ;

import \* as libString from './lib/string.js' ; import \* as math from './math/math.js' ; import \* as vectorMath from './vector/math.js' ;

3.3.1.3.2 기본 import 이름 지정

기본 import 이름은 가져온 파일이름에서 파생되며 6.2 규칙 별 식별자 유형 의 규칙을 따릅니다 .

import MyClass from '../my-class.js' ; import myFunction from '../my\_function.js' ; import SOME\_CONSTANT from '../someconstant.js' ;

참고 : 일반적으로이 스타일 가이드에 의해 기본 내보내기가 금지되어 있으므로 이는 발생하지 않아야합니다 . 3.4.2.1 명명 된 내보내기와 기본 내보내기를 참조하세요 . 기본 가져 오기는이 스타일 가이드를 따르지 않는 모듈을 가져 오는 데만 사용됩니다.

3.3.1.3.3 명명된 import 이름 지정

일반적으로 명명된 import ( import {name}) 를 통해 가져온 심볼은 동일한 이름을 유지해야합니다. import 앨리어싱 ( import {SomeThing as SomeOtherThing})을 피하십시오 . 모듈 가져 오기 ( import \*)를 사용하거나 내보내기 자체의 이름을 변경 하여 이름 충돌을 수정하는 것을 선호합니다.

import \* as bigAnimals from './biganimals.js' ; import \* as domesticatedAnimals from './domesticatedanimals.js' ;

new bigAnimals.cat ();

new domesticatedAnimals.cat();

명명 된 가져 오기의 이름을 변경해야하는 경우 가져온 모듈의 파일 이름 또는 결과 별칭에서 경로의 구성 요소를 사용합니다.

import { Cat as BigCat } from './biganimals.js' ; import { Cat as DomesticatedCat } from './domesticatedanimals.js' ;

new BigCat ();

new DomesticatedCat ();

3.3.2 export

심볼은 모듈 외부에서 사용되는 경우에만 내보내집니다. 내보내지 않은 모듈 로컬 기호는 선언 @private되지 않으며 이름이 밑줄로 끝나지 않습니다. 내보낸 기호 및 모듈 로컬 기호에 대해 규정 된 순서는 없습니다.

3.3.2.1 명명된 내보내기와 기본 내보내기

모든 코드에서 명명 된 내보내기를 사용합니다. export선언에 키워드를 적용 하거나 export {name};구문을 사용할 수 있습니다.

기본 내보내기를 사용하지 마십시오. 모듈을 가져 오면 이러한 값에 이름을 지정해야하므로 모듈 간의 이름 지정에 불일치가 발생할 수 있습니다.

// 기본 내보내기 사용 안 함 :

export default class Foo { ... } // BAD!

// 명명 된 내보내기 사용 :

export class Foo { ... }

// exports라는 대체 스타일 :

class Foo { ... }

export { Foo };

3.4.2.2 정적 컨테이너 클래스 및 개체 내보내기

네임 스페이스를 위해 컨테이너 클래스 또는 정적 메서드 또는 속성이있는 개체를 내 보내지 마십시오.

// container.js

// Bad : Container는 정적 메서드와 필드만 있는 export 클래스입니다.

export class Container {

/ \*\* @return {number} \* /

static bar () {

return 1 ;

}

}

/ \*\* @const {number} \* /

Container. FOO = 1 ;

대신 개별 상수 및 함수를 내보내십시오.

/ \*\* @return {number} \* /

export function bar () {

return 1 ;

}

export const / \*\* 숫자 \* / FOO = 1 ;

3.4.2.3 export의 가변성

내보낸 변수는 모듈 초기화 외부에서 변경해서는 안됩니다.

변경이 필요한 경우 변경 가능한 필드가있는 개체에 대한 상수 참조를 내보내거나 변경 가능한 데이터에 대한 접근자 함수를 내보내는 등의 대안이 있습니다.

// Bad : foo와 mutateFoo가 모두 내보내지고 변경됩니다.

export let / \*\* number \* / foo = 0 ;

/ \*\*

\* foo를 변형합니다.

\* /

export function mutateFoo () {

++ foo ;

}

/ \*\*

\* @param {함수 (숫자) : 숫자} newMutateFoo

\* /

export function setMutateFoo ( newMutateFoo ) {

// 내보낸 클래스와 함수를 변경할 수 있습니다!

mutateFoo = () => {

foo = newMutateFoo ( foo );

};

}

// 좋음 : 변경 가능한 변수 foo 및 mutateFoo를 직접 내보내는 대신 // 모듈 범위를 지정하고 foo에 대한 getter와 mutateFooFunc에 대한 래퍼를 // 내 보냅니다. let / \*\* number \* / foo = 0 ; let / \*\* function (number) : number \* / mutateFooFunc = foo => foo + 1 ;

/ \*\* @return {number} \* / export function getFoo () { return foo ; }

내보내기 함수 mutateFoo () {

foo = mutateFooFunc ( foo ); }

/ \*\* @param {function (number) : number} mutateFoo \* / export function setMutateFoo ( mutateFoo ) {

mutateFooFunc = mutateFoo ; }

3.4.2.4 내보내기

export from문은 줄 바꿈되어서는 안되며 따라서 80 열 제한에 대한 예외입니다. 이것은 두 export from가지 맛 에 모두 적용됩니다 .

'./other.js'에서 {specificName} 내보내기;

'./another.js'에서 \* 내보내기;

3.4.3 ES 모듈의 순환 종속성

ECMAScript 사양이이를 허용하더라도 ES 모듈간에주기를 생성하지 마십시오. import및 export문 을 모두 사용하여주기를 만들 수 있습니다 .

// a.js import './b.js' ;

// b.js import './a.js' ;

// ʻexport from`도 순환 종속성을 유발할 수 있습니다! './c.js' 에서 { x } 내보내기 ;

// c.js import './b.js' ;

내보내기 let x ;

3.4.4 클로저와의 상호 운용

3.4.4.1 goog 참조

Closure goog네임 스페이스 를 참조하려면 Closure의 goog.js.

import \* as goog from '../closure/goog/goog.js' ;

const 이름 = goog . 필요 ( 'a.name' );

export const CONSTANT = name . 계산 ();

goog.jsgoogES 모듈에서 사용할 수있는 전역 속성의 하위 집합 만 내 보냅니다 .

3.4.4.2 ES 모듈의 goog.require

goog.requireES 모듈에서는 goog.module파일 에서와 같이 작동 합니다. 당신은 (즉, 기호가 만든 폐쇄 공간 기호 요구할 수 있습니다 goog.provide또는 goog.module)과 goog.require값을 반환됩니다.

import \* as goog from '../closure/goog/goog.js' ; import \* as anEsModule from './anEsModule.js' ;

const GoogPromise = goog . 필요 ( 'goog.Promise' ); const myNamespace = goog . 필요 ( 'my.namespace' );

3.4.4.3 ES 모듈에서 클로저 모듈 ID 선언

goog.declareModuleIdES 모듈 내에서 goog.module유사한 모듈 ID 를 선언하는 데 사용할 수 있습니다 . 즉,이 모듈 ID goog.require는를 호출하지 않은 a 인 것처럼 d, goog.module.getd, goog.forwardDeclare'd 등이 될 수 있습니다. 전역 적으로 사용 가능한 JavaScript 심볼로 모듈 ID를 생성하지 않습니다.goog.modulegoog.module.declareLegacyNamespace

의 모듈 ID에 대한 A goog.require(또는 goog.module.get) goog.declareModuleId는 항상 모듈 객체를 반환합니다 (예 : import \*'d). 따라서 to 인수 goog.declareModuleId는 항상 lowerCamelCaseName.

참고 : goog.module.declareLegacyNamespaceES 모듈에서 호출하는 것은 오류 이며 goog.module파일 에서만 호출 할 수 있습니다. 레거시 네임 스페이스를 ES 모듈과 직접 연결하는 방법은 없습니다 .

goog.declareModuleId Closure 파일을 명명 된 내보내기가 사용되는 ES 모듈로 업그레이드하는 데만 사용해야합니다.

import \* as goog from '../closure/goog.js' ;

goog . declareModuleId ( 'my.esm' );

내보내기 클래스 클래스 {};

3.5 goog.setTestOnly

A의 goog.module파일 goog.module문을 선택적으로의 호출에 의해 올 수 있습니다 goog.setTestOnly().

ES 모듈에서 import명령문 뒤에 선택적으로 goog.setTestOnly().

3.6 goog.require및 goog.requireType진술

가져 오기는 goog.require및 goog.requireType문 으로 수행됩니다 . goog.require명령문에서 가져온 이름 은 코드 및 유형 주석 모두에서 사용할 수있는 반면, a에서 가져온 이름 goog.requireType은 유형 주석에서만 사용할 수 있습니다.

goog.require및 goog.requireType문이없는 빈 줄 연속 블록을 형성한다. 이 블록 은 하나의 빈 줄로goog.module 구분 된 선언을 따릅니다 . goog.require또는에 대한 전체 인수 는 별도의 파일에서에 goog.requireType의해 정의 된 네임 스페이스 goog.module입니다. goog.require및 goog.requireType명령문은 파일의 다른 곳에 표시되지 않을 수 있습니다.

goog.require또는 각각 goog.requireType은 단일 상수 별칭에 할당되거나 여러 상수 별칭으로 분해됩니다. 이러한 별칭은 유형 주석 또는 코드에서 종속성을 참조 할 수있는 유일한 방법입니다. goog.require또는에 대한 인수로를 제외하고는 정규화 된 네임 스페이스를 어디에도 사용해서는 안됩니다 goog.requireType.

예외 : externs 파일에 선언 된 유형, 변수 및 함수는 유형 주석 및 코드에서 정규화 된 이름을 사용해야합니다.

별칭은 가져온 모듈 네임 스페이스의 마지막 점으로 구분 된 구성 요소와 일치해야합니다.

예외 : 경우에 따라 네임 스페이스의 추가 구성 요소를 사용하여 더 긴 별칭을 만들 수 있습니다. 결과 별칭은 해당 유형을 올바르게 식별 할 수 있도록 원래 식별자의 대 / 소문자를 유지해야합니다. 동일한 별칭을 명확하게하거나 가독성을 크게 향상시키는 경우 더 긴 별칭을 사용할 수 있습니다. 또한, 이상 별칭이 같은 기본 유형을 마스킹 방지하기 위해 사용되어야한다 Element, Event, Error, Map, 그리고 Promise(더 완전한 목록은 참조 표준 내장 객체 및 웹 API를 MDN에서). 구조화되지 않은 별칭의 이름을 바꿀 때 4.6.2 수평 공백 에서 요구하는대로 콜론 뒤에 공백이 있어야합니다 .

파일은 동일한 네임 스페이스에 대해 a goog.require와 goog.requireType문을 모두 포함해서는 안됩니다 . 가져온 이름이 코드 및 유형 주석 모두에서 사용되는 경우 단일 goog.require명령문 으로 가져와야 합니다.

부작용을 위해서만 모듈을 가져 오는 경우 호출은 a goog.require(a goog.requireType아님) 여야 하며 할당을 생략 할 수 있습니다. 이것이 필요한 이유를 설명하고 컴파일러 경고를 억제하려면 주석이 필요합니다.

행은 다음 규칙에 따라 정렬됩니다. 왼쪽에 이름이있는 모든 요구 사항이 먼저 나오고 해당 이름에 따라 알파벳순으로 정렬됩니다. 그런 다음 디스트 럭처링이 필요하며, 다시 왼쪽에있는 이름으로 정렬됩니다. 마지막으로, 독립형 요구 호출 (일반적으로 부작용을 위해 가져온 모듈 용)입니다.

도움말 :이 주문을 기억하고 수동으로 시행 할 필요가 없습니다. IDE를 사용하여 올바르게 정렬되지 않은 요구 사항을보고 할 수 있습니다.

긴 별칭 또는 모듈 이름으로 인해 행이 80 개 열 제한을 초과하는 경우 줄 바꿈 되지 않아야 합니다. require 줄은 80 개 열 제한에 대한 예외입니다.

예:

// 표준 별칭 스타일. const MyClass = goog . 필요 ( 'some.package.MyClass' ); const MyType = goog . requireType ( 'some.package.MyType' ); // 명확성에 사용되는 네임 스페이스 기반 별칭입니다. const NsMyClass = goog . 필요 ( 'other.ns.MyClass' ); // 네이티브 유형 마스킹을 방지하는 데 사용되는 네임 스페이스 기반 별칭입니다. const RendererElement = goog . 필요 ( 'web.renderer.Element' );

// 가독성을 높이기 위해 사용되는 비 순차적 네임 스페이스 기반 별칭입니다. // 또한 80 개 이상의 열을 요구하는 줄은 래핑해서는 안됩니다. const SomeDataStructureModel = goog . requireType ( 'identical.package.identifiers.models.SomeDataStructure' ); const SomeDataStructureProto = goog . 필요 ( 'proto.identical.package.identifiers.SomeDataStructure' ); // 표준 별칭 스타일. const asserts = goog . 필요 ( 'goog.asserts' ); // 명확성에 사용되는 네임 스페이스 기반 별칭입니다.

const testingAsserts = goog . 필요 ( 'goog.testing.asserts' ); // 별칭으로 표준 분해. const { clear , clone } = goog . 필요 ( 'goog.array' ); const { Rgb } = goog . 필요 ( 'goog.color' ); // 명확성을 위해 네임 스페이스 기반의 별칭으로 분해. const { SomeType : FooSomeType } = goog

. requireType ( 'foo.types' ); const { clear : objectClear , clone : objectClone } = goog . 필요 ( 'goog.object' ); // 부작용을 유발하기 위해 별칭없이 goog.require. / \*\* @suppress {extraRequire} MyFramework를 초기화합니다. \* /

goog . 필요 ( 'my.framework.initialization' );

낙심 :

// 명확하게하기 위해 필요한 경우 모듈 이름의 형식에 더 가깝기 때문에 SomeClass보다 PackageClass를 선호합니다 . const SomeClass = goog . 필요 ( 'some.package.Class' );

허용되지 않음 :

// 추가 용어는 네임 스페이스에서 가져와야합니다. const MyClassForBizzing = goog . 필요 ( 'some.package.MyClass' ); // 별칭에는 전체 최종 네임 스페이스 구성 요소가 포함되어야합니다. const MyClass = goog . 필요 ( 'some.package.MyClassForBizzing' ); // 별칭은 네이티브 유형을 마스크해서는 안됩니다 (여기서는 'const JspbMap'이어야 함). const Map = goog . 필요 ( 'jspb.Map' ); // goog.require 줄을 80 개 이상의 열로 나누지 마세요. const SomeDataStructure =

goog . 필요 ( 'proto.identical.package.identifiers.SomeDataStructure' ); // 별칭은 네임 스페이스를 기반으로해야합니다. const randomName = goog . 필요 ( 'something.else' ); // 콜론 뒤에 공백이 없습니다. const { Foo : FooProto } = goog . 필요 ( 'some.package.proto.Foo' ); // 별칭이없는 goog.requireType.

goog . requireType ( 'some.package.with.a.Type' );

/ \*\*

\* @param {! some.unimported.Dependency} param JSDoc에서 사용되는 모든 외부 유형

\* 주석은 externs에서 선언되지 않는 한 goog.require이어야합니다.

\* / function someFunction ( param ) { // goog.require 줄은 다른 코드보다 먼저 최상위 수준에 있어야합니다. const 별칭 = goog . 필요 ( 'my.long.name.alias' ); // ... }

3.7 파일의 구현

실제 구현은 모든 종속성 정보가 선언 된 후에 수행됩니다 (최소 하나의 빈 줄로 구분).

이는 모듈 로컬 선언 (상수, 변수, 클래스, 함수 등)과 내 보낸 기호로 구성 될 수 있습니다.