Programmierstyleguide für JavaScript unter SIMATIC WinCC Unified

SIMATIC WinCC Unified V16,
SIMATIC HMI Unified Comfort Panels

https://support.industry.siemens.com/cs/ww/de/view/109758536

Siemens Industry Online Support

Rechtliche Hinweise

Nutzung der Anwendungsbeispiele

In den Anwendungsbeispielen wird die Lösung von Automatisierungsaufgaben im Zusammenspiel mehrerer Komponenten in Form von Text, Grafiken und/oder Software-Bausteinen beispielhaft dargestellt. Die Anwendungsbeispiele sind ein kostenloser Service der Siemens AG und/oder einer Tochtergesellschaft der Siemens AG („Siemens“). Sie sind unverbindlich und erheben keinen Anspruch auf Vollständigkeit und Funktionsfähigkeit hinsichtlich Konfiguration und Ausstattung. Die Anwendungsbeispiele stellen keine kundenspezifischen Lösungen dar, sondern bieten lediglich Hilfestellung bei typischen Aufgabenstellungen. Sie sind selbst für den sachgemäßen und sicheren Betrieb der Produkte innerhalb der geltenden Vorschriften verantwortlich und müssen dazu die Funktion des jeweiligen Anwendungsbeispiels überprüfen und auf Ihre Anlage individuell anpassen.
Sie erhalten von Siemens das nicht ausschließliche, nicht unterlizenzierbare und nicht übertragbare Recht, die Anwendungsbeispiele durch fachlich geschultes Personal zu nutzen. Jede Änderung an den Anwendungsbeispielen erfolgt auf Ihre Verantwortung. Die Weitergabe an Dritte oder Vervielfältigung der Anwendungsbeispiele oder von Auszügen daraus ist nur in Kombination mit Ihren eigenen Produkten gestattet. Die Anwendungsbeispiele unterliegen nicht zwingend den üblichen Tests und Qualitätsprüfungen eines kostenpflichtigen Produkts, können Funktions- und Leistungsmängel enthalten und mit Fehlern behaftet sein. Sie sind verpflichtet, die Nutzung so zu gestalten, dass eventuelle Fehlfunktionen nicht zu Sachschäden oder der Verletzung von Personen führen.

Haftungsausschluss

Siemens schließt seine Haftung, gleich aus welchem Rechtsgrund, insbesondere für die Verwendbarkeit, Verfügbarkeit, Vollständigkeit und Mangelfreiheit der Anwendungsbeispiele, sowie dazugehöriger Hinweise, Projektierungs- und Leistungsdaten und dadurch verursachte Schäden aus. Dies gilt nicht, soweit Siemens zwingend haftet, z. B. nach dem Produkthaftungsgesetz, in Fällen des Vorsatzes, der groben Fahrlässigkeit, wegen der schuldhaften Verletzung des Lebens, des Körpers oder der Gesundheit, bei Nichteinhaltung einer übernommenen Garantie, wegen des arglistigen Verschweigens eines Mangels oder wegen der schuldhaften Verletzung wesentlicher Vertragspflichten. Der Schadensersatzanspruch für die Verletzung wesentlicher Vertragspflichten ist jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht Vorsatz oder grobe Fahrlässigkeit vorliegen oder wegen der Verletzung des Lebens, des Körpers oder der Gesundheit gehaftet wird. Eine Änderung der Beweislast zu Ihrem Nachteil ist mit den vorstehenden Regelungen nicht verbunden. Von in diesem Zusammenhang bestehenden oder entstehenden Ansprüchen Dritter stellen Sie Siemens frei, soweit Siemens nicht gesetzlich zwingend haftet.
Durch Nutzung der Anwendungsbeispiele erkennen Sie an, dass Siemens über die beschriebene Haftungsregelung hinaus nicht für etwaige Schäden haftbar gemacht werden kann.

Weitere Hinweise

Siemens behält sich das Recht vor, Änderungen an den Anwendungsbeispielen jederzeit ohne Ankündigung durchzuführen. Bei Abweichungen zwischen den Vorschlägen in den Anwendungsbeispielen und anderen Siemens Publikationen, wie z. B. Katalogen, hat der Inhalt der anderen Dokumentation Vorrang.
Ergänzend gelten die Siemens Nutzungsbedingungen (https://support.industry.siemens.com).

Securityhinweise

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen.
Um Anlagen, Systeme, Maschinen und Netzwerke gegen Cyber-Bedrohungen zu sichern, ist es erforderlich, ein ganzheitliches Industrial Security-Konzept zu implementieren (und kontinuierlich aufrechtzuerhalten), das dem aktuellen Stand der Technik entspricht. Die Produkte und Lösungen von Siemens formen nur einen Bestandteil eines solchen Konzepts.
Der Kunde ist dafür verantwortlich, unbefugten Zugriff auf seine Anlagen, Systeme, Maschinen und Netzwerke zu verhindern. Systeme, Maschinen und Komponenten sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn und soweit dies notwendig ist und entsprechende Schutzmaßnahmen (z. B. Nutzung von Firewalls und Netzwerksegmentierung) ergriffen wurden.
Zusätzlich sollten die Empfehlungen von Siemens zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Industrial Security finden Sie unter: https://www.siemens.com/industrialsecurity.
Die Produkte und Lösungen von Siemens werden ständig weiterentwickelt, um Sie noch sicherer zu machen. Siemens empfiehlt ausdrücklich, Aktualisierungen durchzuführen, sobald die entsprechenden Updates zur Verfügung stehen und immer nur die aktuellen Produktversionen zu verwenden. Die Verwendung veralteter oder nicht mehr unterstützter Versionen kann das Risiko von Cyber-Bedrohungen erhöhen.
Um stets über Produkt-Updates informiert zu sein, abonnieren Sie den Siemens Industrial Security RSS Feed unter: https://www.siemens.com/industrialsecurity.

Inhaltsverzeichnis

1 Einleitung

Bei der Erstellung von JavaScript Code hat der Programmierer die Aufgabe, das Anwenderprogramm so übersichtlich und lesbar wie möglich zu gestalten.

Jeder Programmierer wendet eine eigene Strategie an, wie z. B. Variablen oder Funktionen benannt werden oder in welcher Art und Weise kommentiert wird.

Durch die unterschiedlichen Philosophien der Programmierer entstehen sehr unterschiedliche Anwenderprogramme, die häufig nur vom jeweiligen Ersteller interpretiert werden können.

Hinweis Als Grundlage für dieses Dokument gilt der Programmierstyleguide für SIMATIC S7-1200/ S7-1500.

https://support.industry.siemens.com/cs/ww/de/view/109478084

1.1 Ziele

Die hier beschriebenen Regeln und Empfehlungen helfen Ihnen, einen einheitlichen Programmcode zu erstellen, der besser gewartet und wiederverwendet werden kann.

Falls mehrere Programmierer am selben Programm arbeiten, empfiehlt es sich deshalb, eine projektweit gültige Terminologie festzulegen, sowie einen gemeinsamen und abgestimmten Programmierstil einzuhalten. Somit können möglichst frühzeitig Fehler erkannt bzw. vermieden werden.

Für die Wartbarkeit und Übersichtlichkeit des Quellcodes ist es zunächst erforderlich, sich an eine gewisse äußere Form zu halten. Optische Effekte tragen nur unwesentlich zur Qualität der Software bei. Viel wichtiger ist es beispielsweise auch Regeln zu finden, die die Entwickler folgendermaßen unterstützen:

- Vermeiden von Tipp- und Flüchtigkeitsfehlern, die der Compiler anschließend falsch interpretiert.

  Ziel: Der Compiler soll so viele Fehler wie möglich erkennen.

- Unterstützung des Programmierers bei der Diagnose von Programmfehlern, z. B. versehentliche Wiederverwendung einer temporären Variablen über einen Zyklus hinaus.

  Ziel: Der Bezeichner weist frühzeitig auf Probleme hin.

- Vereinheitlichung von Standardapplikationen und Standardbibliotheken

  Ziel: Die Einarbeitung soll einfach sein und die Wiederverwendbarkeit von Programmcode erhöht werden.

- Einfache Wartung und Vereinfachung der Weiterentwicklung

  Ziel: Änderungen von Programmcode an den einzelnen Verwendungsstellen, Skripte an Bildobjekten, in globalen Modulen sollen minimale Auswirkungen auf die Gesamtprojektierung haben. Änderungen von Programmcode in den einzelnen Modulen sollen von verschiedenen Programmierern durchführbar sein.

Hinweis Beachten Sie, dass die in diesem Dokument beschriebene Regeln und Empfehlungen aufeinander abgestimmt sind und aufeinander aufbauen.

1.2 Vorteile eines einheitlichen Programmierstils

- Einheitlicher durchgängiger Stil
- Leicht lesbar und verständlich
- Einfache Wartung und Wiederverwendbarkeit
- Einfache und schnelle Fehlererkennung und -korrektur
- Effiziente Zusammenarbeit zwischen mehreren Programmierern

1.3 Gültigkeit

Dieses Dokument gilt für Projekte und Bibliotheken im TIA Portal im Zusammenhang mit der Skriptumgebung JavaScript entsprechend der ECMAScript Language Specification.

Als Skript-Engine wird Google V8 verwendet, die weitestgehend ECMAScript 2015, 6th Edition vom Juni 2015 (ECMAScript 6) implementiert.

ESLint-Regeln

Dieses Dokument verwendet als Grundlage die ESLint-Regeln. Die hier aufgeführten Regeln können mitunter leicht zu den ESLint-Regeln abweichen, da in erster Linie eine Durchgängigkeit in der TIA Portal Projektierung (Zusammenspiel mit Programmierstyleguide für SIMATIC S7-1200/ S7-1500) angestrebt wird.

1.4 Abgrenzung

Dieses Dokument enthält keine Beschreibung von:

- HMI-Projektierung im TIA Portal
- Inbetriebnahme von HMI Projekten

Ausreichende Erfahrung in diesen Themen wird vorausgesetzt, um die Regeln und Empfehlungen sinnvoll interpretieren und anwenden zu können. Das Dokument ersetzt keinen Know-how Aufbau in der Softwareentwicklung, sondern dient als Referenz.

1.5 Regelverletzung und andere Vorgaben

Bei Kundenprojekten sind die geforderten Normen, sowie die kunden- oder branchenspezifischen Standards des Kunden oder der verwendeten Technologie einzuhalten und besitzen Priorität gegenüber diesem Styleguide oder Teilen davon.

Bei einer Kombination von Kundenvorgaben und diesem Styleguide ist auf die Integrität der Kombination beider Vorgaben und des Gesamtprojekts zu achten. Eine Regelverletzung ist an der entsprechenden Stelle im Anwenderprogramm zu begründen und zu dokumentieren. Die vom Kunden definierten Regeln sind in geeigneter Form zu dokumentieren.

2 Begriffsklärung

2.1 Regeln/ Empfehlungen

Die Vorgaben dieses Dokuments werden in Empfehlungen und Regeln unterteilt:

- **Regeln** sind verbindliche Vorgaben und sind unbedingt einzuhalten. Sie sind für eine wiederverwendbare und performante Programmierung unabdingbar. Im Ausnahmefall können Regeln auch verletzt werden. Dies muss aber entsprechend dokumentiert werden.
- **Empfehlungen** sind Vorgaben, die zum einen der Einheitlichkeit des Codes dienen und zum anderen als Unterstützung und Hinweis gedacht sind. Empfehlungen sollten prinzipiell befolgt werden, es kann aber durchaus Fälle geben, in denen eine Empfehlung nicht befolgt wird. Gründe hierfür können bessere Effizienz oder bessere Lesbarkeit sein.

2.2 Regelnummerierung

Für eine eindeutige Regelzuordnung zwischen verschiedenen Kategorien werden die Regeln und Empfehlungen entsprechend ihrer Zugehörigkeit mit einem Präfix (zwei Zeichen) gekennzeichnet und nummeriert (3 Ziffern).

Entfällt eine Regel, wird die Nummer nicht neu vergeben. Wenn Sie weitere Regeln definieren, können Sie eine Nummerierung zwischen 901 und 999 nutzen.

Tabelle 2-1

Präfix	Kategorie
ES	Engineering System (Programmierumgebung)
GR	General rules (Allgemeine Regeln)
SR	Stylistic rules (Stilistische Regeln)
ST	Script tags (Skriptvariablen)
FO	Functions and objects (Funktionen und Objekte)
IL	Instructions and Loops (Anweisungen und Schleifen)
EA	Expressions to avoid (zu vermeidende Ausdrücke)
DD	Diagnosis and Debugging (Diagnose und Debugging)

2.3 Begriffe (Block, Funktion und Parameter)

In einigen Regeln dieses Programmierstyleguides wird von Funktionen, Funktionsparametern und Blöcken gesprochen.

Die nachfolgenden Beschreibungen geben eine kurze Erklärung dieser Begriffe und stellen diese klar. Diese Klarstellung ist erforderlich, damit im Weiteren ein gemeinsames Verständnis über die verwendeten Begriffe herrscht.

Funktion und Parameter

Wenn Sie eine Funktion in JavaScript erstellen bzw. diese verwenden, dann können Sie auch Werte an die Funktion übergeben. Dies können Sie mithilfe von "Parametern" realisieren. "Parameter" werden häufig auch als "Argumente" bezeichnet, sind aber von der Bedeutung identisch.

```
const tagValue1, tagValue2;


function doSomething(parameter1, parameter2) {
  // …function code
}


// …code
doSomething(tagValue1, tagValue2);
```

Block

Ein Block oder auch Anweisungsblock wird dazu genutzt, um ein oder mehrere Anweisungen zu gruppieren. Ein Block ist davon gekennzeichnet, dass er von geschweiften Klammern umfasst wird.

Das folgende Beispiel zeigt einen Block am Beispiel der if-Anweisung.

```
if (a > 2) {
  tagValue = a * 6;
  HMIRuntime.Trace(a);
}
```

3 Einstellungen im TIA Portal

Dieses Kapitel beschreibt die Regeln und Empfehlungen für die Grundeinstellungen in der Programmierumgebung TIA Portal.

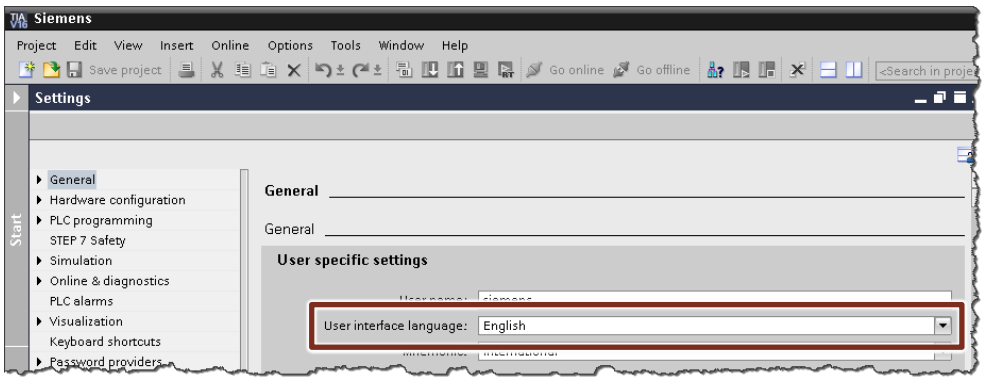
Hinweis Die nachfolgenden Einstellungen haben keine direkte Auswirkung auf den Skripteditor von JavaScript. Sie sind jedoch im Sinne der Durchgängigkeit mit andere Styleguides empfohlen.

ES001 Regel: Oberflächensprache "English"

Die Oberflächensprache in TIA Portal wird auf "English" eingestellt. Dadurch werden alle neu angelegten Projekte automatisch in der Editier- und Referenzsprache, sowie die Systemkonstanten in "English" angelegt.

Begründung: Damit die Systemkonstanten in allen Projekten in derselben Sprache vorliegen, muss die Oberflächensprache einheitlich eingestellt sein.

Abbildung 3-1

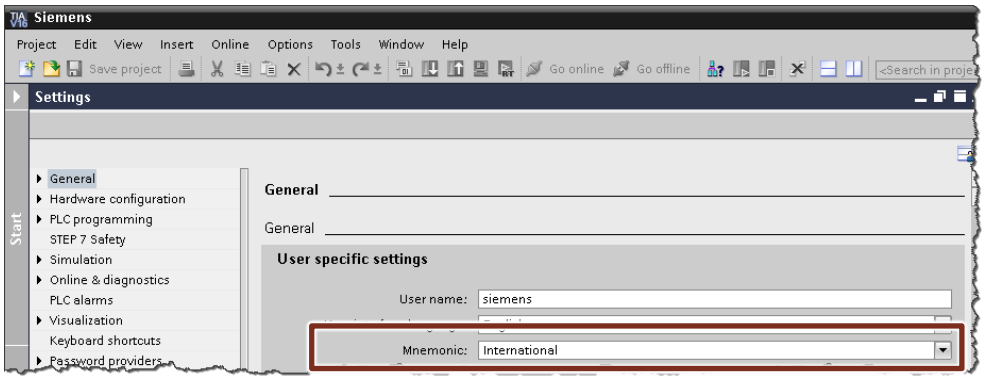


ES002 Regel: Mnemonik "International"

Die Mnemonik (Spracheinstellung für Programmiersprachen) wird auf "International" eingestellt.

Begründung: Alle Systemsprachen und Systemparameter sind somit eindeutig und sprachunabhängig. Das gewährleistet eine reibungslose Zusammenarbeit im Team.

Abbildung 3-2



4 Allgemeine Regeln

GR001 Regel: Geschweifte Klammern bei mehrzeiligen Anweisungen

Wenn Sie Anweisungen (z.B. if, while, function) haben, die sich über mehrere Zeilen erstrecken, nutzen Sie immer geschweifte Klammern.

Weitere Informationen: http://eslint.org/docs/rules/curly

GR002 Regel: Valide reguläre Ausdrücke verwenden

Verwenden Sie in Ihrem JavaScript Code nur valide reguläre Ausdrücke. Hierzu können Sie folgende Webseite nutzen: https://regexr.com/

Weitere Informationen: http://eslint.org/docs/rules/no-invalid-regexp

GR003 Regel: Typsichere Gleichheitsoperatoren verwenden

Verwenden Sie bei einem Vergleich von Werten immer die typsicheren Gleichheitsoperatoren (z.B. " ===", "!==") an Stelle des Gegenstücks "==" und !=".

Weitere Informationen: http://eslint.org/docs/rules/eqeqeq

Speziell, wenn Sie eine Anweisung mit "null" vergleichen wollen, sollten Sie die typsicheren Gleichheitsoperanden verwenden

Weitere Informationen: http://eslint.org/docs/rules/no-eq-null

GR004 Empfehlung: Mehrdeutige Zuweisungsoperatoren in bedingten Anweisungen vermeiden

Vermeiden Sie in bedingten Anweisungen (z.B. if, for, while-Anweisung) mehrdeutige Zuweisungsoperatoren (=). Verwenden Sie stattdessen typsichere Gleichheitsoperatoren.

Weitere Informationen: http://eslint.org/docs/rules/no-cond-assign

```
// Allowed:
if (x === 0) {
  const b = 1;
}

// Disallowed:
if (x = 0) {
  const b = 1;
}
```

GR005 Empfehlung: Leere Anweisungen vermeiden

Vermeiden Sie Anweisungen und Funktionen ohne Inhalt, da dies ebenfalls beim Lesen zu Verwirrung führen kann.

Weitere Informationen: http://eslint.org/docs/rules/no-empty

GR006 Empfehlung: Einfache Anführungszeichen nutzen

JavaScript ermöglicht es Ihnen, Zeichenfolgen auf eine von drei Arten zu definieren:

- doppelte Anführungszeichen ("),
- einfache Anführungszeichen (') und
- Backticks (`)

In WinCC Unified wird empfohlen standardmäßig einfache Anführungszeichen für Strings zu verwenden. Wenn Sie in einem String weitere Anführungszeichen benötigen, sollten doppelten Anwendungszeichen verwendet werden.

Weitere Informationen: http://eslint.org/docs/rules/quotes

```
// Allowed:
const value = 'myValue';
const string = 'a string containing "double" quotes';

// Disallowed:
const value = "myValue";
const string = "a string containing "double" quotes";
```

Hinweis

Sie können Back-Ticks (`) auch als sogenannte Template-Strings verwenden. Dadurch können Sie Text und Variable in einem Ausdruck miteinander verbinden. Die Variablen werden dabei mit Dollarzeichen und geschweiften Klammern "${Variable}" angegeben.

```
// Allowed:
const value = 'myValue';
const string = `The tag value is ${value}`;
```


GR007 Empfehlung: Kommas vermeiden bei einem erwarteten Ausdruck

Vermeiden Sie Kommas, wenn nur ein Ausdruck erwartet wird.

Der Komma-Operator umfasst mehrere Ausdrücke, wobei nur einer erwartet wird. Er wertet jeden Operanden von links nach rechts aus und gibt den Wert des letzten Operanden zurück. Dadurch werden jedoch häufig Nebenwirkungen verschleiert, und seine Verwendung ist häufig ein Versehen.

Weitere Informationen: http://eslint.org/docs/rules/no-sequences

5 Stilistische Regeln

SR001 Regel: Zwei Leerzeichen zur Codestrukturierung

Um Ihren JavaScript Code besser zu strukturieren, wird empfohlen in WinCC Unified zwei Leerzeichen zu verwenden, um Inhalte einzurücken. Dies hilft bei der Nachvollziehbarkeit und Lesbarkeit des Codes.

Weitere Informationen: http://eslint.org/docs/rules/indent

```
// Allowed:
if (x === 0) {
··const b = 1;
}

// Disallowed:
if (x === 0) {
const b = 1;
}
```

SR002 Regel: Maximal zwei Leerzeilen im Code

Um Ihren Code zu strukturieren können Sie Leerzeilen verwenden, z.B. um Funktionen oder If-Anweisungen logisch und visuell abzusetzen. Des Weiteren verbessert es die Lesbarkeit des Codes.

Empfohlen wird maximal zwei Leerzeilen zwischen Codeblöcken zu verwenden.

Weitere Informationen: http://eslint.org/docs/rules/no-multiple-empty-lines

SR003 Regel: Leerzeichen nach jedem Satzzeichen

In JavaScript sind folgende Satzzeichen wichtige Elemente bei der Code-Erstellung:

- Komma (",")
- Doppelpunkt (":")
- Semikolon (";")

Wenn Sie in Ihrem Code eines dieser Satzzeichen verwenden, sollte vor diesem kein Leerzeichen verwendet werden. Nach einem Satzzeichen hingegen sollte ein Leerzeichen eingefügt werden.

Weitere Informationen:

- Komma (","): http://eslint.org/docs/rules/comma-spacing
- Doppelpunkt (":"): http://eslint.org/docs/rules/key-spacing
- Semikolon (";"): http://eslint.org/docs/rules/semi-spacing

SR004 Regel: Codezeile mit Semikolon (";") abschließen

Jede Codezeile muss in JavaScript mit einem Semikolon abgeschlossen werden, da dies sonst zu einem Fehler führt.

Weitere Informationen: http://eslint.org/docs/rules/semi

Achten Sie jedoch darauf, dass Sie nicht zu viele Semikolons verwenden da dies zu Verwirrungen führen kann.

Weitere Informationen: http://eslint.org/docs/rules/no-extra-semi

SR005 Regel: Schlüsselwörter mit Leerzeichen umschließen

Wenn Sie Schlüsselwörter (z.B. "if", "else", "switch") in ihrem JavaScript verwenden, sollten diese mit Leerzeichen umschlossen werden (d.h. vor und nach dem Schlüsselwort steht ein Leerzeichen.

Weitere Informationen: http://eslint.org/docs/rules/keyword-spacing

```
// Allowed:
if (true) {
  //...
} else {
  //...
}

// Disallowed:
if (true) {
  //...
}else{
  //...
}
```


SR006 Empfehlung: Leerzeichen vor jedem Anweisungsblock

Wenn Sie einen Anweisungsblock beginnen, z.B. nach einer If-Anweisung, sollten Sie ein Leerzeichen einfügen.

Weitere Informationen: http://eslint.org/docs/rules/space-before-blocks

```
// Allowed:
if (true) {
  //...
}

// Disallowed:
if (true){
  //...
}
```

SR007 Empfehlung: Leerzeichen zwischen Operanden

Wenn Sie in Ihrem Code eine mathematische Berechnung haben, dann sollte auf jeden Operanden und jeden Operator ein Leerzeichen folgen.

Weitere Informationen: http://eslint.org/docs/rules/space-infix-ops

```
// Allowed:
a = b + c;

// Disallowed:
a=b+c;
```

SR008 Empfehlung: Komma am Zeilenanfang vermeiden

Es wird empfohlen Kommas (",") am Zeilenende statt des Zeilenanfangs zu verwenden.

Weitere Informationen: http://eslint.org/docs/rules/comma-style

```
// Allowed:
const foo = 1, bar = 2;
const value1 = 1,
    value2 = 2;

// Disallowed:
const foo = 1
, bar = 2;
const value1 = 1
  , value2 = 2;
```

SR009 Empfehlung: Keine Leerzeichen am Zeilenende

Wenn Sie Ihren Code erstellen und diesen kommentieren, sollten Sie darauf achten, dass Sie die Zeile nicht mit einem Leerzeichen beenden.

Dies ist kein genereller Fehler es kann jedoch zu Fehlinterpretation kommen.

Weitere Informationen: http://eslint.org/docs/rules/no-trailing-spaces

```
// Allowed:
const foo = 1, bar = 2; //comment

// Disallowed:
const foo = 1, bar = 2; //comment...
//...
```

SR010 Empfehlung: Keine Leerzeilen an Blockanfängen und Klassen

Vermeiden Sie am Beginn eines Blockes oder einer Klasse eine Leerzeile.

Weitere Informationen: http://eslint.org/docs/rules/padded-blocks

SR011 Empfehlung: Leerzeichen zwischen Funktion und Identifier vermeiden

Beachten Sie das Sie kein Leerzeichen zwischen der Funktion und dem Identifier setzen.

Beispiel

```
// Allowed:
myFunction();

// Disallowed:
myFunction ();
```

Weitere Informationen: http://eslint.org/docs/rules/no-spaced-func

SR012 Empfehlung: Leerzeichen vor Funktionsklammern

Lassen Sie immer ein Leerzeichen zwischen der Funktion und den geschweiften Funktionsklammern stehen.

Weitere Informationen: http://eslint.org/docs/rules/space-before-function-paren

```
// Allowed:
myFunction(a) {};

// Disallowed:
myFunction(a){};
```

SR013 Empfehlung: Kommentare mit Leerzeichen beginnen

Wenn Sie eine Codezeile kommentieren, dann fügen Sie bevor Sie mit dem Kommentar-Text beginnen ein Leerzeichen ein.

Ausgenommen von dieser Regel sind Strukturierungszeilen ("//*********") um Codeblöcke visuell voneinander zu trennen.

Weitere Informationen: http://eslint.org/docs/rules/spaced-comment

```
// Allowed:
// This is an example comment
/* This is an example comment */
//******************************

// Disallowed:
//This is an example comment
/*This is an example comment*/
```


SR014 Empfehlung: Normale Tabulator und Leerzeichen verwenden

Verwenden Sie nur die Tabulator- oder die Leerzeichen-Taste, um ein Leerzeichen einzufügen.

Weitere Informationen: http://eslint.org/docs/rules/no-irregular-whitespace

SR015 Empfehlung: Zeichenketten auf eine Zeile begrenzen

Geben Sie keine Zeichenketten in Ihren Code an, die länger sind als eine Zeile. Falls Sie dennoch eine mehrzeilige Zeichenkette benötigen, können Sie dies mit einem Zeilenumbruch "\n" realisieren.

Weitere Informationen: http://eslint.org/docs/rules/no-multi-str

```
// Allowed:
const x = "Line 1\n" +
        "Line 2";

// Disallowed:
const x = "Line 1 \
        Line 2";
```

6 Skriptvariablen, Arrays und Strukturen

6.1 Allgemeine Regeln für Skriptvariablen

ST001 Regel: Schreibweise "camelCase" benutzen

Benutzen Sie für die Bezeichnung von Variablen die camelCase-Schreibweise.

Weitere Informationen: http://eslint.org/docs/rules/camelcase

```
// Allowed:
let machineState;
let standardValue;

// Disallowed:
let machine_State;
let StandardValue;
```

ST002 Regel: Eindeutige Variablenbezeichnung

Wenn Sie Variablen definieren, achten Sie darauf, dass diese einen eindeutigen Namen haben und auch nur einmal verwendet werden.

Weitere Informationen: http://eslint.org/docs/rules/no-redeclare

Innerhalb einer Funktion sollten die Variablen ebenfalls eindeutig benannt sein und sich nicht mit globalen Variablen überschneiden.

Weitere Informationen: http://eslint.org/docs/rules/no-shadow

ST003 Regel: Punktnotation verwenden

Nutzen Sie die Punktnotation, um auf Eigenschaften von Objekten und Elementen zugreifen zu können.

Weitere Informationen: http://eslint.org/docs/rules/dot-notation

```
Screen.Items('Rectangle1').BackColor = 0xFF00FF00;
```

ST004 Regel: Keine Schlüsselwörter für Funktionen oder Variablen verwenden

Nutzen Sie keine nativen Schlüsselworte oder Objektnamen (z.B. "length", "top", "undefined", "NaN", "Infinity", "eval" oder "arguments") als Name einer Funktion oder einer Variablen. Dies kann zu Verwirrungen führen und erschwert die Nachvollziehbarkeit des Codes.

Weitere Informationen: http://eslint.org/docs/rules/no-native-reassign
http://eslint.org/docs/rules/no-shadow-restricted-names

ST005 Regel: Keine neuen primitiven Wrapper erzeugen

In JavaScript gibt es drei primitive Datentypen die Wrapper-Objekte haben:

- String
- Zahlen

- Boolean.

Vermeiden Sie eigene manuelle Wrapper-Instanzen zu erzeugen, auch wenn das JavaScript zulässt.

Weitere Informationen: http://eslint.org/docs/rules/no-new-wrappers

ST006 Empfehlung: Oktale Escape-Sequenzen in String-Literalen verbieten

Seit der JavaScript Version ECMAScript 5-Spezifikation sind oktale Escape-Sequenzen in String-Literalen veraltet und sollten nicht mehr verwendet werden. Stattdessen sollten Unicode-Escape-Sequenzen verwendet werden.

Weitere Informationen: http://eslint.org/docs/rules/no-octal-escape

ST007 Empfehlung: Unnötige boolesche Operationen vermeiden

Vermeiden Sie bei Vergleichen oder Zuweisungen unnötige boolesche Operationen z.B. doppelte Negation.

Weitere Informationen: http://eslint.org/docs/rules/no-extra-boolean-cast

6.2 Variablen deklarieren und verwenden

ST008 Regel: Variablen deklarieren (Blockgültigkeitsbereiche)

Damit Sie Variablen in ihrem Skript verwenden können, müssen Sie diese zunächst deklarieren.

Grundsätzlich unterscheidet man zwischen den Deklarationsarten:

Tabelle 6-1

Deklaration	Erklärung
var	Deklariert eine Variable und haben keinen Blockgültigkeitsbereich.
let	Deklariert eine Variable mit Gültigkeit im aktuellen Block.
const	Deklariert eine Konstante mit Gültigkeit im aktuellen Block. Es tritt ein Fehler auf, wenn versucht wird, die Variable zu überschrieben.

Hinweis Die Deklarationsart "var" ist veraltet und sollte für neue JavaScript-Skripte vermieden werden.

Als Faustregel für die Deklaration von Variablen gilt:

- Verwenden Sie standardmäßig die Variablendeklaration "const".
- Wenn sich der Wert oder das Objekt der Variablen im Skriptverlauf ändert, wird die Deklarationsart "const" durch "let" ersetzt.

```
var a = 5;
let b = 50;
const c = 500;

if (a > 2) {
  a = 6;
  let b = 60;           //only in the block visible
  HMIRuntime.Trace(a); //Trace output: 6
  HMIRuntime.Trace(b); //Trace output: 60
  HMIRuntime.Trace(c); //Trace output: 500
}

HMIRuntime.Trace(a);   //Trace output: 6
HMIRuntime.Trace(b);   //Trace output: 50
HMIRuntime.Trace(c);   //Trace output: 500
```

ST009 Regel: Variablen am Anfang eines Codeblockes deklarieren

Beachten Sie bei der Codeerstellung, dass Sie die Variablen zu Beginn eines Codeblockes deklarieren.

Weitere Informationen: http://eslint.org/docs/rules/vars-on-top

ST010 Regel: "var" deklarierte Skriptvariablen im definierten Codeblock verwenden

Bitte vermeiden Sie die Deklarationsart "var". Wenn Sie dennoch "var" benötigen, dann verwenden Sie die Skriptvariablen, die mit "var" deklariert wurde, stets

innerhalb des Codeblocks indem Sie diese definiert haben, da es sonst zu Fehlermeldungen kommen kann.

Weitere Informationen: http://eslint.org/docs/rules/block-scoped-var

```
// Allowed:
function doSomething() {
  var result;

  if (x > 10) {
    result = true;
  } else {
    result = false;
  }
}

// Disallowed:
function doSomething() {
  if (x > 10) {
    var result = true;
  } else {
    var result = false;
  }
}
```


ST011 Regel: Pro Variablendeklaration eine Zeile

Nutzen Sie für jede Variablendeklaration eine separate Zeile.

Weitere Informationen: http://eslint.org/docs/rules/one-var

ST012 Empfehlung: Unnötige Variablendeklarationen vermeiden

Definieren Sie nur so viele Variablen in ihrem Code, wie Sie auch tatsächlich benötigen. Zu viele definierte Variablen lassen schnell einen komplexen Eindruck vermuten.

Weitere Informationen: http://eslint.org/docs/rules/no-unused-vars

ST013 Empfehlung: "var"-Deklaration in Unterfunktionen/Anweisungen vermeiden

Bitte vermeiden Sie die Deklarationsart "var". Wenn Sie dennoch "var" benötigen, dann definieren Sie Variablen und Funktionen als "var" stets im Hauptteil einer Funktion und vermeiden Sie diese in verschachtelten Unterfunktionen oder Anweisungen zu definieren.

Weitere Informationen: http://eslint.org/docs/rules/no-inner-declarations

6.3 Variablen vergleichen

ST014 Regel: Variablen nicht mit sich selbst vergleichen

Vergleichen Sie eine Variable nie mit sich selbst, da dies zu einem Fehler führt. Als Ausnahme zählt die Überprüfung auf die Funktion "isNaN()".

Weitere Informationen: http://eslint.org/docs/rules/no-self-compare

ST015 Regel: Zahl prüfen, ob dies keine Zahl ist

Wenn Sie einen Variablenwert darauf überprüfen, ob diese keine Zahl ist, dann verwenden Sie die Funktion "isNaN()" an Stelle des Ausdrucks "NaN" (Abkürzung für "Not a Number")

Weitere Informationen: http://eslint.org/docs/rules/use-isnan

```
// Allowed:
if (isNaN(x))

// Disallowed:
if (x == NaN)
```


ST016 Regel: Ausgangsformat (Radix-Argument) bei parseInt() Funktion angeben

Die parseInt()-Funktion hat neben dem zu konvertierenden Ausdruck als zweites Argument (die Radix) das Ausgangsformat, des zu konvertierenden Ausdrucks. Standardmäßig erkennt die parseInt()-Funktion automatisch ob es sich um einen Integer oder einen Hexadezimalzahl handelt (mittels 0x-Präfix).

Um die Angaben jedoch eindeutig anzugeben, wird empfohlen das Ausgangsformat in der Funktion mit anzugeben.

Weitere Informationen: http://eslint.org/docs/rules/radix

6.4 Angabe von Zahlen

ST017 Regel: Gleitkommazahlen komplett angeben

Wenn Sie Gleitkommazahlen angeben, nutzen Sie immer die gängigen Zahlenformate um eine eindeutige Darstellung als Dezimalzahl zu erhalten. Dies vermeidet Fehlinterpretationen beim Lesen des Codes.

Weitere Informationen: http://eslint.org/docs/rules/no-floating-decimal

```
// Allowed:
const x = -0.2;
const y = 5.0;

// Disallowed:
const x = -.2;
const y = 5.;
```

ST018 Regel: Nummern ohne "0" beginnen

Wenn Sie Zahlen in ihrem Skriptcode nutzen, dann beachten Sie, dass Sie diese ohne führende Nullstellen eingeben, da diese sonst als Oktalzahl ausgewertet werden kann.

Weitere Informationen: http://eslint.org/docs/rules/no-octal

6.5 Arrays und Strukturen

ST019 Regel: Kommas zwischen Struktur- und Array-Elementen

Die Aufzählung von Elementen eines Arrays werden durch Kommas getrennt. Nach dem letzten Element darf kein Komma stehen.

Weitere Informationen: http://eslint.org/docs/rules/comma-dangle

ST020 Empfehlung: Leere Array-Elemente vermeiden

Achten Sie darauf, dass Sie bei der Erstellung eines Arrays keine leeren Arrayelemente einfügen.

Weitere Informationen: http://eslint.org/docs/rules/no-sparse-arrays

```
// Allowed:
const colors = ['red','green','blue'];

// Disallowed:
const colors = ['red', ,'blue'];
```

7 Funktionen und Objekte

7.1 Funktionen

FO001 Regel: Funktionen nicht neu definieren

Wenn Sie Funktionen deklariert haben, achten Sie darauf, dass Sie diese an einer anderen Stelle nicht neu definieren.

Weitere Informationen: http://eslint.org/docs/rules/no-func-assign

```
// Disallowed:
function multiply(x, y) {
  return x * y;
}
multiply = 22;
```

Wenn Sie hingegen einer Variablen eine anonyme Funktion zuweisen, dann können Sie die Variable an einer anderen Stelle überschreiben.

```
// Allowed:
let multiply = function (x, y) {
  return x * y;
}
multiply = 22;
```

FO002 Regel: Rückgabewert definieren

Definieren Sie in einer Funktion ob diese einen Rückgabewert hat oder nicht. Falls die Funktion einen Rückgabewert hat, dann definieren Sie auch welchen.

Weitere Informationen: http://eslint.org/docs/rules/consistent-return

FO003 Regel: Übergabeparameter nicht überschreiben

Achten Sie darauf, dass Sie die Übergabeparameter von einer Funktion oder innerhalb einer Funktion nicht überschreiben.

Weitere Informationen: http://eslint.org/docs/rules/no-ex-assign
Weitere Informationen: http://eslint.org/docs/rules/no-param-reassign

FO004 Regel: Funktionsaufruf bei -deklaration in Klammern

Wenn eine deklarierte Funktion sofort aufgerufen werden soll, müssen Sie diese in Klammern hüllen.

Weitere Informationen: http://eslint.org/docs/rules/wrap-iife

FO005 Regel: Anweisungen nicht in einen String schreiben

Vermeiden Sie JavaScript-Anweisungen einem String zuzuweisen, da dies zu einem Fehler führt.

Weitere Informationen: http://eslint.org/docs/rules/no-implied-eval

FO006 Regel: Funktion erstellen ohne "new"

Achten Sie bei der Erstellung einer Funktion darauf das Sie diese ohne "new" erstellen.

Weitere Informationen: http://eslint.org/docs/rules/no-new-func

FO007 Regel: Globale Objekteigenschaften nicht als Funktion verwenden

Verwenden Sie keine globalen Objekteigenschaften als Funktion (Math, JSON,...).

Weitere Informationen: http://eslint.org/docs/rules/no-obj-calls

FO008 Empfehlung: Einzeilige Codeblöcke vermeiden

Vermeiden Sie einzeilige Codeblöcke in ihrem Skript, da diese schnell unübersichtlich und schwer nachzuvollziehen sind.

Weitere Informationen: http://eslint.org/docs/rules/no-lone-blocks

```
// Allowed:
function doSomething () {
  test();
}

// Disallowed:
function doSomething () {
  {
    test();
  }
}
```

FO009 Empfehlung: Funktion "eval()"vermeiden

Die "eval()"-Funktion wertet JavaScript-Code aus, und stellt diese als Zeichenkette dar. Diese Funktion ist potenziell gefährlich und wird oft missbraucht. Vermeiden Sie diese Funktion und nutzen Sie Alternativen wie im Link beschrieben.

Weitere Informationen: http://eslint.org/docs/rules/no-eval

FO010 Empfehlung: Unnötige Bindungen von Funktionen vermeiden

Vermeiden Sie eine unnötige Verwendung der Methode "bind()".

Weitere Informationen: http://eslint.org/docs/rules/no-extra-bind

7.2 Objekte

FO011 Regel: Neue Objekte in Variablen speichern

Wenn Sie ein neues Objekt erstellen, dann wird empfohlen dieses in einer Skriptvariablen zu speichern.

Weitere Informationen: http://eslint.org/docs/rules/no-new

```
// Allowed:
const myObject = new Object();
myObject();

// Disallowed:
new object();
```

FO012 Regel: Objektnamen beginnen mit Großbuchstaben

Der Objektname nach dem Operator "new" beginnt mit einem Großbuchstaben.

Weitere Informationen: http://eslint.org/docs/rules/new-cap

```
// Allowed:
const myObject = new Object();

// Disallowed:
const myObject = new object();
```


FO013 Regel: Keine doppelte Eigenschafts-Deklaration bei Objekten

Weisen Sie einer Variablen nicht mehrmals Werte zu, da diese zu einem unerwarteten Verhalten führen kann.

Weitere Informationen: http://eslint.org/docs/rules/no-dupe-keys

```
// Allowed:
const testObject = {
  value1 = 'red',
  value2 = 'green'
};

// Disallowed:
const testObject = {
  value = 'red',
  value = 'green'
};
```

FO014 Regel: Native Objekte nicht verändern

JavaScript erlaubt es Objekte zu erweitern u.a. auch native Objekte. Das kann dazu führen, dass das Verhalten der Objekte geändert wird, was wiederrum zu Irritationen führen kann, wenn Sie das "native Objekt in einem weiteren Codeblock verwenden.

Vermeiden Sie deswegen "native" Objekte zu verändern oder zu erweitern.

Weitere Informationen: http://eslint.org/docs/rules/no-extend-native

FO015 Empfehlung: Objekt-Konstruktor bei Objekterstellung vermeiden

Wenn Sie ein neues Objekt erstellen, sollten Sie darauf achten, dass Sie die objektliterale Syntax bevorzugen, wonach Sie den Objekt-Konstruktor vermeiden sollten.

Weitere Informationen: http://eslint.org/docs/rules/no-new-object

```
// Allowed:
const a = {};

// Disallowed:
const a = new Object();
```

8 Bedingte Anweisungen, Verzweigungen und Schleifen

IL001 Regel: "if": Bei "return" ist "else" überflüssig

Wenn eine "if"-Anweisung eine Rückgabeanweisung enthält, wird der "else"-Block überflüssig. Der Inhalt kann somit außerhalb des Blocks platziert werden.

Weitere Informationen: http://eslint.org/docs/rules/no-else-return

IL002 Regel: Konstanten nicht alleine in Bedingungen verwenden

Wenn Sie eine Anweisung mit einer Bedingung verwenden, achten Sie darauf, dass Sie bei der Bedingung immer einen Vergleich verwenden, z.B. mit einer Konstanten.

Wenn Sie in der Bedingung eine Konstante ohne Vergleich angeben, dann wird die Anweisung immer oder niemals ausgeführt.

Dies betrifft im Wesentlichen folgende Anweisungen:

- "if" - Anweisung
- "for" - Anweisung
- "while" - Anweisung
- "do...while" - Anweisung

Weitere Informationen: http://eslint.org/docs/rules/no-constant-condition

```
// Allowed:
if (value === 0) {
  myFunction();
}

// Disallowed:
if (false) {
  myFunction();
}
```

IL003 Regel: "for…in": Eigenschaften erst prüfen, bevor damit gearbeitet wird

Prüfen Sie in einer "for...in"-Schleife zuerst ob eine Eigenschaft existiert, bevor Sie mit dieser weiterarbeiten.

Weitere Informationen: http://eslint.org/docs/rules/guard-for-in

IL004 Regel: "Default" in "for"/"switch" Anweisungen definieren

Definieren Sie in einer "for" oder "switch" Anweisung, sofern möglich, immer einen "default" Zustand.

Wenn dies der Anwendungsfall nicht zulässt, dann ergänzen Sie im Code einen entsprechenden Kommentar "// No Default".

Weitere Informationen: http://eslint.org/docs/rules/default-case

IL005 Regel: Keine doppelten "case" Anweisungen

Achten Sie darauf, das Sie in einer "switch..case" Anweisung keine "case"-Bedingung doppelt haben (z. B. durch einen Copy&Paste-Fehler).

Weitere Informationen: http://eslint.org/docs/rules/no-duplicate-case

```
// Allowed:
switch (value) {
  case 1:
    break;
  case 2:
    break;
  case 3:
    break;
  default:
    break;
}
// Disallowed:
switch (value) {
  case 1:
    break;
  case 2:
    break;
  case 1:          // duplicate test expression
    break;
  default:
    break;
}
```


IL006 Regel: Jedes "case" hat ein "break"

In einer "switch"-Anweisung sollte jeder Fall bzw. jede "case"-Anweisung mit einem "break" versehen werden. Dies verhindert ein sogenanntes "durchfallen" durch alle "case" Anweisungen. Ist ein durchfallen gewünscht, sollte der Kommentar "//fall through" ergänzt werden.

Weitere Informationen: http://eslint.org/docs/rules/no-fallthrough

IL007 Empfehlung: Verschachtelte dreifach Operationen vermeiden

Vermeiden Sie verschachtelte dreifach Operationen innerhalb einer Zeile. Setzen Sie alternativ die "if"-Anweisung ein.

Weitere Informationen: http://eslint.org/docs/rules/no-nested-ternary

```
// Allowed:
const a = b ? 1 : c;

// Disallowed:
const a = b ? 1 : c ? 2 : 3;
```

9 Zu vermeidende Ausdrücke

EA001 Regel: Ausdruck "with" vermeiden

Vermeiden Sie die Anweisung "with()", weil Sie Mitglieder eines Objekts zum aktuellen Bereich hinzufügen und es unmöglich macht, zu erkennen, worauf sich eine Variable innerhalb des Blocks tatsächlich bezieht.

Weitere Informationen: http://eslint.org/docs/rules/no-with

EA002 Regel: Ausdruck "arguments.caller" und "arguments.callee" vermeiden

Vermeiden Sie veralteten und suboptimalen Code, wie "arguments.caller" und "arguments.callee".

Weitere Informationen: http://eslint.org/docs/rules/no-caller

EA003 Regel: Eigenschaft "_proto_" vermeiden

Die Eigenschaft "_proto_" ist seit ECMAScript 3.1 veraltet und sollte im Code nicht mehr verwendet werden. Verwenden Sie stattdessen "Object.getPrototypeOf" und "Object.setPrototypeOf"

Weitere Informationen: http://eslint.org/docs/rules/no-proto

10 Fehlersuche und Debugging

DD001 Regel: Trace()-Funktion zur Diagnose

Um Ihren Code zu debuggen und eventuelle Fehler zu beseitigen, benutzen Sie ausschließlich die Trace()-Funktion von SIMATIC WinCC Unified.

Achten Sie am Ende darauf, dass Sie alle überflüssigen Trace()-Meldungen aus ihrem Code entfernen, da dieser sonst schnell unübersichtlich wird.

Weitere Informationen: http://eslint.org/docs/rules/no-unreachable
http://eslint.org/docs/rules/no-console
http://eslint.org/docs/rules/no-debugger

11 Anhang

11.1 Service und Support

Industry Online Support

Sie haben Fragen oder brauchen Unterstützung?

Über den Industry Online Support greifen Sie rund um die Uhr auf das gesamte Service und Support Know-how sowie auf unsere Dienstleistungen zu.

Der Industry Online Support ist die zentrale Adresse für Informationen zu unseren Produkten, Lösungen und Services.

Produktinformationen, Handbücher, Downloads, FAQs und Anwendungsbeispiele – alle Informationen sind mit wenigen Mausklicks erreichbar:

support.industry.siemens.com

Technical Support

Der Technical Support von Siemens Industry unterstützt Sie schnell und kompetent bei allen technischen Anfragen mit einer Vielzahl maßgeschneiderter Angebote – von der Basisunterstützung bis hin zu individuellen Supportverträgen.

Anfragen an den Technical Support stellen Sie per Web-Formular:
support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

Mit unseren weltweit verfügbaren Trainings für unsere Produkte und Lösungen unterstützen wir Sie praxisnah, mit innovativen Lernmethoden und mit einem kundenspezifisch abgestimmten Konzept.

Mehr zu den angebotenen Trainings und Kursen sowie deren Standorte und Termine erfahren Sie unter:

siemens.de/sitrain

Serviceangebot

Unser Serviceangebot umfasst folgendes:

- Plant Data Services
- Ersatzteilservices
- Reparaturservices
- Vor-Ort und Instandhaltungsservices
- Retrofit- und Modernisierungsservices
- Serviceprogramme und Verträge

Ausführliche Informationen zu unserem Serviceangebot finden Sie im Servicekatalog:

support.industry.siemens.com/cs/sc

Industry Online Support App

Mit der App "Siemens Industry Online Support" erhalten Sie auch unterwegs die optimale Unterstützung. Die App ist für iOS und Android verfügbar:

support.industry.siemens.com/cs/ww/de/sc/2067

11.2 Links und Literatur

Tabelle 11-1

Nr.	Thema
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link auf die Beitragsseite des Anwendungsbeispiels https://support.industry.siemens.com/cs/ww/de/view/109758536
\3\	Find and fix problems in your JavaScript code https://eslint.org/

11.3 Änderungsdokumentation

Tabelle 11-2

Version	Datum	Änderung
V1.0	11/2020	Erste Ausgabe