javaScript Coding-Guidelines
2. Ausgabe Februar 2015

Der Guideline beschreibt den verwendeten Coding-Stil von javaScript als eigene Richtline.

Inhalts-Verzeichnis

1. Allgemeine Richtlinien

1.1 Anzahl der Leerzeichen

Es werden 4 Leerzeichen als Einrückung verwendet. Tabbs dürfen nicht zur Formatierung eingesetzt werden.

1.3. Zeichen-Codierung

Alle Datei-Inhalte werden mit UTF-8 gelesen, verarbeitet und gespeichert. Es wird zu keinem Zeitpunkt der Verarbeitung das Byte-Order-Mark verwendet. Weitere Infos zu BOM bietet z.B. der Wikipedia-Text unter http://de.wikipedia.org/wiki/Byte_Order_Mark.

1.4. Maximale Zeilen-Länge

Die maximale Zeichen-Länge darf 120 Zeichen nicht überschreiten, nach Möglichkeit werden im Höchstfall 80 Zeichen verwendet.

1.5. Zeilen-Abschluß

Alle Anweisungen werden mit einem Semikolon beendet, je Zeile ist nur eine Anweisung zu schreiben.

2. Namens-Konvention und Integration

2.1 Dateinamen-Konvention

Minimierte Jquery-javaScript-Dateien erhalten im Rahmen der Standard-Konvention den Dateinamen jquery.name-1.10.min.js. Bei unkomprimierten Skripten wird die Form jquery.name-1.10.js verwendet. 1.10 bezeichnet hier exemplarisch die Version.

2.2 Code-Integration

JavaScript-Datein werden HTML5-konform mit

```
<script src="pfad/zur/jsdatei">
    Inhalt des Skriptes
</script>
```

eingebunden. Js-Skripte werden möglichst hoch im Quelltext plaziert, nach Möglichkeit im Header-Tag. Der Code wird nicht direkt in der HTML-Struktur eingebettet. Das Laden von externen Js-Dateien erfolgt hinter der Einbindung von CSS.

3. Code-Stil

3.1 Quellcode-Kommentierung

Mehrzeilige Kommentare werden mit Blöcken realisiert, Einzeilige entsprechend mit Zeilen-Kommentaren. Bei der Kommentierung in Blöcken ist darauf zu achten, dass keine Programm-Fehler auftreten können.

```
// Kommentar über eine Zeile
/*
mehrzeiliger Kommentar
```

```
*/


/*
führt zu einem Fehler:
var rm_a = /a*/.match(s);
*/
```

3.2 Namen und Variablen

Der Name darf ein Buchstabe sein, auf den optional ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen. Ein Name darf kein reserviertes Wort enthalten.

Jede Variable erhält eine eigene Zeile. Mehrere Variablen werden in alphabetischer Reihenfolge notiert. Auf globale Variablen wird verzichtet, alle globale Werte werden in einem einzigen Objekt-Literal notiert.

```
// Globale Werte mit Objekt-Literal
      var global = {
          "os" : "win",
          "browser" : "firefox",
          "version" : "16"
       };
```

3.3 Funktionen

Für den gesamten Code wird der K&R-Stil für die Klammerung verwendet. Vor einer sich öffnenden geschweiften Klammer wird ein Leerzeichen gesetzt.

Zur Vermeidung von gleichnamigen Funktionen werden Funktions-Sammlungen verwendet.

```
function foo() {
    return true;
}
```

```
var nav = {
    over: function () {
        alert('over');
    },
    out: function () {
        alert('out');
    }
};
```

Die Funktion wird vor ihrer Benutzung deklariert. Lokale Variablen werden mit den ersten Anweisungen innerhalb der Funktion ebenfalls für ihren Geltungsbereich deklariert. Innere Funktionen folgen diesen var-Statements, damit im Quellcode der aktuelle Scope deutlicher wird.

```
    function outer(c, d) {
        var e = c * d;

        function inner(a, b) {
            return (e * a) + b;
        }

        return inner(0, 1);
    }
```

3.4 Kontroll-Statements

Auch für Kontroll-Strukturen wird durchgängig der K&R-Stil verwendet. In Blöcken werden immer geschweifte Klammern verwendet. Ausdrücke werden bei Bedingungen nicht in der gleichen Zeile geschrieben. „else/else if/catch“ werden in der gleichen Zeile geschrieben wie die geschweifte Klammer. Ternäre Operatoren werden in if-else-Statements nicht verwendet.

```
// nicht sauber ohne geschweifte Blöcke:
   if ( true )
      hello();

// saubere Variante mit Klammerung
   if ( true ) {
```

```
    hello();
  }

// Beispiel für Bedingungen mit -else-
  if (x) {
    hello();
  } else {
    world();
  }
```

3.5 Whitespace

- Im Kontroll-Bereich von for-Statements wird nach dem Semikolon jeweils ein Whitespace eingefügt.
- Nach einem Schlüsselwort wird vor einer folgenden linksöffnenen geschweiften Klammer ein Whitespace gesetzt (Beispiel 1).
- Bei Funktions-Werten wird kein Whitespae beim ersten und letzten Wert gesetzt (Beispiel 2).
- Bei mehren Parametern werden die einzelnen Parameter mit einem Komma und Whitespace gestrennt (Beispiel 3).
- Methoden- und Eigenschafts-Werte in Objekt-Literalen werden mit einem Whitespace hinter dem Doppelpunkt geschrieben (Beispiel 4).

```
// Beispiel 1
while (true) { …

// Beispiel 2
function (a) …
// Beispiel 3
function hello(world, x, y)


// Beispiel 4
var hello = {
world: 40000,
```

```
x: [a, b, c]
};
```

4.) Typ-Überprüfungen

In Konvention zum Jquery Core Style-Guideline sind folgene Prüfüberprüfungen bei javaScript und jQuery gültig (Quelle: http://docs.jquery.com/JQuery_Core_Style_Guidelines, November 2012):

- String: typeof object === "string"
- Number: typeof object === "number"
- Boolean: typeof object === "boolean"
- Object: typeof object === "object"
- Plain Object: jQuery.isPlainObject(object)
- Function: jQuery.isFunction(object)
- Array: jQuery.isArray(object)
- Element: object.nodeType
- null: object === null
- null or undefined: object == null
- undefined:
    - Global Variables: typeof variable === "undefined"
    - Local Variables: variable === undefined
    - Properties: object.prop === undefined

5.) Objekt-Instanziierung

Für die Erzeugung eines neuen Objektes wird „{}“ verwendet und ensprechend „[]“ für ein neues Array.