How to write JSXGraph source code

API docs

```
/**
 * @class Midpoint of two points.
 * @pseudo
 * @description A midpoint is given by two points. It is collinear to the given points and
 * is the same to each of the given points, i.e. it is in the middle of the given points.
 * @constructor
 * @name Midpoint
 * @type JXG.Point
 * @augments JXG.Point
 * @throws {Error} If the element cannot be constructed with the given parent objects an ex
 * @param {JXG.Point_JXG.Point} p1,p2 The constructed point will be in the middle of p1 and
 * @param {JXG.Line} l The midpoint will be in the middle of {@link JXG.Line#point1} and {
 * the given line l.
 */
```

Method `createElement`

```
JXG.createMidpoint = function (board, parents, attributes) {
    var a, b, t, i, attr;

    for (i = 0; i < parents.length; ++i) {
        parents[i] = board.select(parents[i]);
    }
    if (
        parents.length === 2 &&
        Type.isPointType(board, parents[0]) &&
        Type.isPointType(board, parents[1])
    ) {
        parents = Type.providePoints(board, parents, attributes, "point");
        a = parents[0];
        b = parents[1];
    } else if (parents.length === 1 && parents[0].elementClass === Const.OBJECT_CLASS_LINE)
        a = parents[0].point1;
        b = parents[0].point2;
    } else {
        throw new Error(
            "JSXGraph: Can't create midpoint." +
                "\nPossible parent types: [point,point], [line]"
        );
    }

```

```
        attr = Type.copyAttributes(attributes, board.options, "midpoint");
        /**
         * @type JXG.Element
         * @ignore
         */
        t = board.create(
            "point",
            [
                function () {
                    var x = a.coords.usrCoords[1] + b.coords.usrCoords[1];
                    if (
                        isNaN(x) ||
                        Math.abs(a.coords.usrCoords[0]) < Mat.eps ||
                        Math.abs(b.coords.usrCoords[0]) < Mat.eps
                    ) {
                        return NaN;
                    }

                    return x * 0.5;
                },
                function () {
                    var y = a.coords.usrCoords[2] + b.coords.usrCoords[2];
                    if (
                        isNaN(y) ||
                        Math.abs(a.coords.usrCoords[0]) < Mat.eps ||
                        Math.abs(b.coords.usrCoords[0]) < Mat.eps
                    ) {
                        return NaN;
                    }

                    return y * 0.5;
                }
            ],
            attr
        );
        if (Type.exists(a._is_new)) {
            t.addChild(a);
            delete a._is_new;
        } else {
            a.addChild(t);
        }
        if (Type.exists(b._is_new)) {
            t.addChild(b);
            delete b._is_new;
        } else {
            b.addChild(t);
```

```
71      }
72
73      t.elType = "midpoint";
74      t.setParents([a.id, b.id]);
75
76      t.prepareUpdate().update();
77
78      /**
79       * Used to generate a polynomial for the midpoint.
80       * @name Midpoint#generatePolynomial
81       * @returns {Array} An array containing the generated polynomial.
82       * @private
83       * @function
84       * @ignore
85       */
86      t.generatePolynomial = function () {
87          /*
88           *  Midpoint takes two point A and B or line L (with points P and Q) and creates po
89           *
90           *  L (not necessarily)
91           *  ----------x------------------x------------------x--------
92           *            A (a1,a2)          T (t1,t2)          B (b1,b2)
93           *
94           * So we have two conditions:
95           *
96           *   (a)   AT  ||  TB           (collinearity condition)
97           *   (b)  [AT] == [TB]          (equidistant condition)
98           *
99           *      a2-t2       t2-b2
100          *     -------  =  -------                                        (1)
101          *      a1-t1       t1-b1
102          *
103          *     (a1 - t1)^2 + (a2 - t2)^2 = (b1 - t1)^2 + (b2 - t2)^2       (2)
104          *
105          *
106          * Multiplying (1) with denominators and simplifying (1) and (2) gives
107          *
108          *    a2t1 - a2b1 + t2b1 - a1t2 + a1b2 - t1b2 = 0                    (1')
109          *
110          *    a1^2 - 2a1t1 + a2^2 - 2a2t2 - b1^2 + 2b1t1 - b2^2 + 2b2t2 = 0    (2')
111          *
112          */
113         var a1 = a.symbolic.x,
114             a2 = a.symbolic.y,
115             b1 = b.symbolic.x,
116             b2 = b.symbolic.y,
```

```
117             t1 = t.symbolic.x,
118             t2 = t.symbolic.y,
119             poly1 = "(" + a2 + ")*(" + t1 + ")-(" + a2 + ")*(" + b1 + ")+(" + t2 + ")*(" + b
120                 ")-(" + t1 + ")*(" + b2 + ")",
121             poly2 = "(" + a1 + ")^2 - 2*(" + a1 + ")*(" + t1 + ")+(" + a2 + ")^2-2*(" + a2 +
122                 ")-(" + b2 + ")^2+2*(" + b2 + ")*(" + t2 + ")";
123 
124         return [poly1, poly2];
125     };
126 
127     return t;
128 };
```