JSXGraph Reference Card

Include JSXGraph in HTML

Three parts are needed: Include files containing the software, an HTML element, and JavaScript code.

Include files:

Three files have to be included: `jsxgraph.css`, `jsxgraphcore.js` and either `prototype.js` or `jquery.js`.

```
- <script type="text/javascript"
          src="domain/prototype.js"></script>
or - <script type="text/javascript"
          src="domain/jquery.min.js"></script>
- <link rel="stylesheet" type="text/css"
        href="domain/jsxgraph.css"/>
- <script type="text/javascript"
          src="domain/jsxgraphcore.js"></script>
```

`domain` is the location of the files. This can be a local directory or `http://jsxgraph.uni-bayreuth.de/distrib/`

HTML element containing the construction:

```
<div id="box" class="jxgbox"
   style="width:600px; height:600px;"></div>
```

JavaScript code:

```
<script type="text/javascript">
   var brd = JXG.JSXGraph.initBoard('box',{axis:true});
<script>
```

Initializing the board

```
var brd = JXG.JSXGraph.initBoard('box',{attributes});
```

– Attributes of the board

`unitX, unitY:`	number of pixels of one unit in x/y-axis direction
`originX, originY:`	the coordinates of the origin in pixel coordinates
`zoomX,zoomY:`	zoom factor in x/y-axis direction
`zoomfactor:`	overall zoom factor in both directions
`axis:true/false:`	show axes
`grid:true/false:`	show grid

Properties and methods of the board:

`brd.snapToGrid:true/false:`	grid mode
`brd.suspendUpdate()`	stop updating (is speed is needed)
`brd.unsuspendUpdate()`	restart updating

Basic commands

```
var el = brd.createElement('type',[parents],{attributes});
el.setProperty({key1:value1,key2:value2,...});
```

Point

```
brd.createElement('point',[parents],{attributes});
```

Parent elements:

`[x,y]`	Euclidean coordinates
`[z,x,y]`	Homogeneous coordinates (z in first place)

`[function(){return p1.X();},`
` function(){return p2.Y();}]`
	Functions for x, y, (and z)

Methods

`p.X(),p.Y()`	x-coordinate, y-coordinate
`p.Z()`	(Homogeneous) z-coordinate
`p.Dist(q)`	Distance from p to point q

Glider

Point on circle, line or curve.

```
brd.createElement('glider',[parents],{attributes});
```

Parent elements:

`[x,y,c]`	Initial coordinates and object to glide on
`[c]`	Object to glide on (initially at origin)

Coordinates may also be defined by functions, see Point.

Line

```
brd.createElement('line',[parents],{attributes});
```

Parent elements:

`[p1,p2]`	line through 2 points
`[c,a,b]`	line defined by 3 coordinates (can also be functions)
`[[x1,y1],[x2,y2]]`	line by 2 coordinate pairs

In case of coordinates as parents, the line is the set of solutions of the equation $a{\cdot}x+b{\cdot}y+c{\cdot}z=0$. Points may also be specified as array of coordinates.

Circle

```
brd.createElement('circle',[parents],{attributes});
```

Parent elements:

`[p1,p2]`	2 points: center and point on circle line
`[p,r]`	center, radius (constant or function)
`[p,c],[c,p]`	center, circle from which the radius is taken
`[p,l],[l,p]`	center, line segment for the radius
`[p1,p2,p3]`	circle through 3 points

Points may also be specified as array of coordinates.

Polygon

```
brd.createElement('polygon',[p1,p2,...],{attributes});
```

`[p1,p2,...]`	array of points

The points array connected by line segments and the inner area is filled.

Group

```
brd.createElement('group',[p1,p2,...],{attributes});
```

`[p1,p2,...]`	array of points

Invisible grouping of points. If one point is moved, the others are transformed accordingly.

Slider

```
var s = brd.createElement('slider',
                         [[a,b],[c,d],[e,f,g]],{atts});
```

`[a,b],[c,d]:`	visual start and end position of the slider
`[e,f,g]:`	the slider returns values between e and g, the initial position is at value f
`s.Value():`	returns the position of the slider $\in [e,g]$

Curve

– Function graph, $x \mapsto f(x)$:

```
brd.createElement('functiongraph',[parents],{atts});
```

Parent elements:

`[function(x){return x*x;},-1,1]`	function term optional: start, end

The other types of curves are defined through:

```
brd.createElement('curve',[parents],{attributes});
```

Parent elements:

– Parameter curve, $t \mapsto (f(t), g(t))$:

```
[function(t){return 5*t;},function(t){return t*t;},0,2]
```

x function, y function, optional: start, end

– Polar curve:

Defined by the equation $r = f(\phi)$.

```
[function(phi){return 5*phi;},[1,2],0,Math.PI]
```

Defining function, optional: center, start, end

– Data plot:

```
[[1,2,3],[4,-2,3]]
```

array of x-coordinates, array of y-coordinates, *or*

```
[[1,2,3],function(x){return x*x;}]
```

array of x-coordinates, function term

– Cubic spline:

```
brd.createElement('spline',[p1,p2,...],{attributes});
```

`[p1,p2,...]`	array of points

Tangent, normal

```
var el = brd.createElement('tangent',[g],{attributes});
var el = brd.createElement('normal',[g],{attributes});
```

`g`	circle, line, polygon, or curve to glide on

Turtle

```
var t = brd.createElement('turtle’);
var t = brd.createElement('turtle',[],{attributes});
var t = brd.createElement('turtle',[parents],{atts});
```

The turtle has a position and a direction (in degrees). All angles have to be supplied in degrees.

Parent elements:

`[x,y,angle]` Optional start values for x, y, and direction

Methods:

Most of the methods have an abbreviated alternative version.

`t.back(len); or t.bk(len);`
`t.clean();` erase the turtle lines without resetting the turtle
`t.clearScreen(); or t.cs();` call `t.home()` and `t.clean()`
`t.forward(len); t.fd(len);`
`t.hideTurtle(); or t.ht();`
`t.home();` Set the turtle to [0,0] and direction to 90.
`t.left(angle); or t.lt(angle);`
`t.lookTo(t2.pos);` Turtle looks to the turtle `t2`
`t.lookTo([x,y]);` Turtle looks to a coordinate pair
`t.moveTo([x,y]);` Move the turtle with drawing
`t.penDown(); or t.pd();`
`t.penUp(); or t.pu();`
`t.popTurtle();` pop turtle status from stack
`t.pushTurtle();` push turtle status on stack
`t.right(angle); or t.rt(angle);`
`t.setPos(x,y);` Move the turtle without drawing
`t.setPenColor(col);` `col`: colorString, e.g. ’red’ or ’#ff0000’
`t.setPenSize(size);` `size`: number
`t.showTurtle(); or t.st();`

Other geometric elements

– *angle:* filled area defined by 3 points
`el = brd.createElement('angle',[A,B,C],{attributes});`

– *arc:* circular arc defined by 3 points
`el = brd.createElement('arc',[A,B,C],{attributes});`

– *arrow:* line through 2 points with arrow head
`el = brd.createElement('arrow',[A,B],{attributes});`

– *arrowparallel:* arrow parallel to arrow a starting at point P
`el = brd.createElement('arrowparallel',[a,P],{atts});`
`el = brd.createElement('arrowparallel',[P,a],{atts});`

– *bisector:* angular bisector defined by 3 points, returns line
`el = brd.createElement('bisector',[A,B,C],{atts});`

– *circumcircle:* circle through 3 points (deprecated)
`el = brd.createElement('circumcircle',[A,B,C],{atts});`

– *circumcirclemidpoint:* center of circle through 3 points
`el = brd.createElement('circumcirclemidpoint',[A,B,C]);`

– *midpoint:* midpoint between 2 points or the 2 points defined by a line

`el = brd.createElement('midpoint',[A,B],{atts});`
`el = brd.createElement('midpoint',[line],{atts});`

– *mirrorpoint:* rotate point B around point A by 180°
`el = brd.createElement('mirrorpoint',[A,B],{atts});`

– *parallel:* line parallel to line l through point P
`el = brd.createElement('parallel',[l,P],{atts});`
`el = brd.createElement('parallel',[P,l],{atts});`

– *parallelpoint:* point D such that $ABCD$ from a parallelogram
`el = brd.createElement('parallelpoint',[A,B,C],{atts});`

– *perpendicular:* line perpendicular to line l through point P
`el = brd.createElement('perpendicular',[l,P],{atts});`
`el = brd.createElement('perpendicular',[P,l],{atts});`

– *perpendicularpoint:* point defining a perpendicular line to line l through point P

`el = brd.createElement('perpendicularpoint',[l,P],{});`
`el = brd.createElement('perpendicularpoint',[P,l],{});`

– *reflection:* reflection of point P over the line l. Superseded by transformations

`el = brd.createElement('reflection',[l,P],{atts});`
`el = brd.createElement('reflection',[P,l],{atts});`

– *sector:* circle sector defined by 3 points ???
`el = brd.createElement('sector',[A,B,C],{atts});`

Attributes of geometric elements

Generic attributes:

`strokeWidth:` number
`strokeColor,fillColor,highlightFillColor,`
`highlightStrokeColor,labelColor:` color string
`strokeOpacity,fillOpacity,highlightFillOpacity,`
`highlightStrokeOpacity:` value between 0 and 1
`visible,trace,draft:` true, false
`dash:` dash style for lines: $0, 1, \ldots, 6$

Attributes for point elements:

`style:` point style: $0, 1, \ldots, 12$
`fixed:` true, false

Attributes for line elements:

`straightFirst,straightLast,withTicks:`true, false

Attributes for line and arc elements:

`firstArrow,lastArrow:` true, false

Text

Display static or dynamic texts.

`el = brd.createElement('text',[x,y,"Hello"]);`
`el = brd.createElement('text',[x,y,f]);`
Example for a dynamic text: return the x coordinate of the point p.

`f = function(){ return p.X(); }`

Transform

Affine transformation of objects.

`t = brd.createElement('transform',[data,base],{type:'type'});`
`base`: the transformation is applied to the coordinates of this object.

Possible types:

– translate: `data=[x,y]`

– scale: `data=[x,y]`

– reflect: `data=[line]` or `[x1,y1,x2,y2]`

– rotate: `data=[angle,point]` or `[angle,x,y]`

– shear: `data=[angle]`

– generic: `data=[v11,v12,v13,v21,...,v33]` 3×3 matrix

Methods:

`t.bindTo(p)` the coordinates of p are defined by t
`t.applyOnce(p)` apply the transformation once
`t.melt(s)` combine two transformations to one: $t := t \cdot s$

Mathematical functions

Functions of the intrinsic JavaScript object *Math*:

```
Math.abs,Math.acos,Math.asin,Math.atan,Math.ceil,
Math.cos,Math.exp,Math.floor,Math.log,Math.max,
Math.min,Math.random,Math.sin,Math.sqrt,Math.tan
```

`(number).toFixed(3):` Rounding a number to fixed precision

Additional mathematical functions are methods of `JXG.Board`.

`brd.angle(A,B,C)`	angle ABC
`brd.cosh(x), board.sinh(x)`	
`brd.pow(a,b)`	a^b
`brd.D(f,x)`	compute $\frac{d}{dx}f$ numerically
`brd.I([a,b],f)`	compute $\int_a^b f(x)dx$ numerically
`brd.root(f,x)`	root of the function f. Uses Newton method with start value x
`brd.factorial(n)`	computes $n! = 1 \cdot 2 \cdot 3 \cdots n$
`brd.binomial(n,k)`	computes $\binom{n}{k}$
`brd.distance(arr1,arr2)`	Euclidean distance
`brd.lagrangePolynomial([p1,p2,...])`	returns a polynomial through the given points
`brd.neville([p1,p2,...])`	polynomial curve interpolation

– Intersection of objects:

`brd.intersection(el1,el2,i,j)` intersection of the elements el_1 and el_2 which can be lines, circles or curves

In case of circle and line intersection, $i \in \{0,1\}$ denotes the first or second intersection. In case of an intersection with a curve, i and j are floats which are the start values for the path positions in the Newton method for el_1 and el_2, resp.

Todo list

'axis', 'image', 'integral', 'ticks'.

Chart

To do ...

Links

Help pages are available at `http://jsxgraph.org`

Mar 2009 v0.2.