## About JointJS

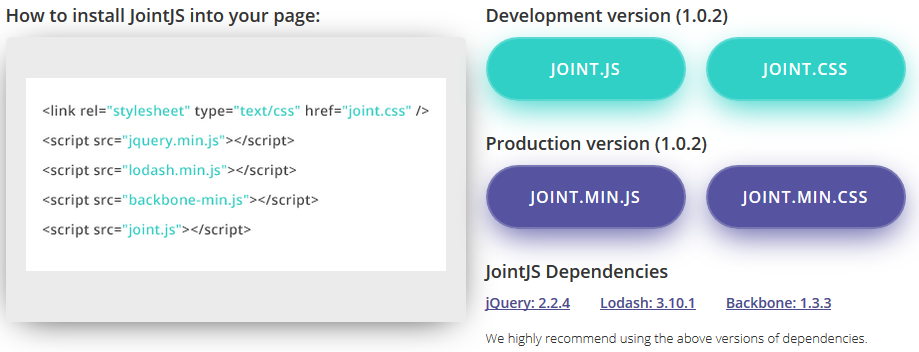
**The JointJS Core library is open source (under the MPL v2 license).**

JointJS Core library is licensed under the Open Source Mozilla Public License Version 2.0.( <https://www.mozilla.org/en-US/MPL/2.0/> )

**JointJS** is a modern HTML 5 JavaScript library for visualization and interaction with diagrams and graphs. It can be used to create anything from static diagrams to fully interactive diagramming tools and application builders

**JointJS Core Library**

To use JointJS you first need the library core files joint.js and joint.css (or their minified versions). Then you need to include the JointJS dependencies: jQuery, Backbone and Lodash libraries. Plugins can be intermixed arbitrarily or not used at all.



JointJS is based on [jQuery](http://jquery.com), [Underscore](http://underscorejs.org), [Backbone](http://backbonejs.org) and [SVG](http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html).

JointJS library exports three global variables: **joint**, **V** and **g**.

The joint namespace contains all the objects that you will use to build your diagrams. Additionally, joint.version property tells you which version of JointJS you're using.

The [V](http://jointjs.com/docs/vectorizer) global is lightweight SVG library that we call Vectorizer. This tiny library makes manipulation with SVG documents much easier. JointJS uses this library internally. Normally, you don't have to get in touch with this library at all but for advanced uses, it can be handy.

The [g](http://jointjs.com/docs/geometry) global is another lighweight library used internally by JointJS that provides many useful geometry operations. Again, you might not get in touch with this library but when you do have the need to perform geometric operations in your applications, you'll certainly find it helpful

<http://jointjs.com/faq>

1. Is JointJS/Rappid compatible with AngularJS and other application frameworks?

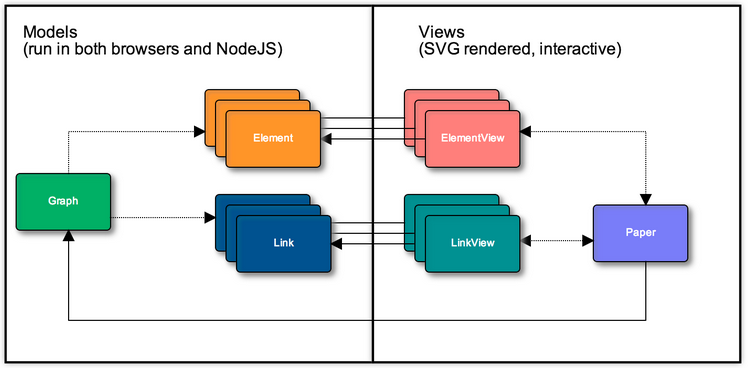
Definitely! Many customers use JointJS/Rappid in combination with widely adopted JS frameworks such as AngularJS, Backbone, React, Meteor and CSS framework Bootstrap - all without problems. We even provide a demo application showing integration of Rappid with AngularJS as part of the Rappid package.

2. What are the limitations of the framework?

JointJS/Rappid is only compatible with modern browsers (Chrome, Firefox, Safari, IE>9, Opera). However, this "limitation" allows us to iterate much faster and use the latest HTML 5 features.

## How to use

JointJS has built-in elements for basic shapes. These shapes are all in the joint.shapes.basic namespace and consist of joint.shapes.basic.Rect, joint.shapes.basic.Circle, joint.shapes.basic.Text and joint.shapes.basic.Image. Additionally, JointJS has plugins that contain shapes and links for elements of some well-known diagrams.



### Need knowledge:

* Jquery
* Backbon
* SVG

### Create a symbol diagram:

* Create a html selector to display diagram:

*<div id=”paper”></div>*

* Create a Graph object in Javascript (model object for diagram):

*var graph = new joint.dia.Graph;*

* Create a Paper object in Javascript (view object for diagram):

*var paper = new joint.dia.Paper({*

*el: $(‘#paper’), //html container*

*width: 800,*

*height: 600,*

*gridSize: 1,*

*model: graph //model of view*

*});*

* Create a shape:

*var rect = new joint.shapes.basic.Rect({*

*position: { x: 100, y: 30 },*

*size: { width: 100, height: 30 },*

*attrs: { rect: { fill: 'blue' }, text: { text: 'my box', fill: 'white' } }*

*});*

* Add this shape to Graph:

*graph.addCell(rect);*

### Event handling

*elementVariable.on(‘event name’, function(){*

*//event handling*

*});*

‘event name’ refer to event of each element: a list of events that you can react on for models (joint.dia.Element, joint.dia.Link and joint.dia.Graph) and views (joint.dia.Paper).

### Get/Set value of properties of elements

element.prop(properties)

Set properties, possibly nested, on the element model. This is an equivalent of the [attr()](http://resources.jointjs.com/docs/jointjs/v1.0/joint.html#dia.Element.prototype.attr) method but this time for custom data properties.

element.prop('name/first', 'John')

element.prop('name/first') // 'John'

element.prop({ name: { first: 'John' } })

// Nested arrays are supported too:

element.prop('mylist/0/data/0/value', 50)

element.prop({ mylist: [ { data: [ { value: 50 } ] } ] })

### Element styling

* User filters and gradients of SVG
* Input html into jointjs

### Link styling

The SVG structure of links is the following:

<path class="connection"/>

<path class="marker-source"/>

<path class="marker-target"/>

<path class="connection-wrap"/>

<g class="labels" />

<g class="marker-vertices"/>

<g class="marker-arrowheads"/>

<g class="link-tools" />

Let's see an example:

link.attr({

'.connection': { stroke: 'blue' },

'.marker-source': { fill: 'red', d: 'M 10 0 L 0 5 L 10 10 z' },

'.marker-target': { fill: 'yellow', d: 'M 10 0 L 0 5 L 10 10 z' }

});

Setting vertices on a link is as simple as:

link.set('vertices', [{ x: 300, y: 60 }, { x: 400, y: 60 }, { x: 400, y: 20 }])

If you prefer the link to not be sharply broken at the vertices but instead interpolated by a curve, set the smooth attribute:

link.set('smooth', true)

### Custom element