Core concepts

When working with Fabric.js there are many classes and concepts you will have to learn to know. This is an overview of the main building blocks of Fabric.js

Canvas

The main container for Fabric.js is the `StaticCavas` or the interactive version called simply `Canvas`. This is a class that provides you the surface where you will draw and will also provide the tools to:

- Handle selections and object interactions
- Reorder the stack of objects
- Render imperatively
- Serialize and Deserialize the state
- Export the graphic state to JSON or SVG or an image
- Handle the viewport of your current application

Objects

Objects are the items we add on the Canvas or StaticCanvas. The prebuilt objects offer some basic shapes and Text. Each of these objects represent a visually different shape that can be added on the canvas and freely transformed or edited.

- `Path`
- `Polyline`, `Polygon`

Home Demos Docs Api specs Team

- `FabricImage`
- `FabricText`, `IText`, `Textbox`

Patterns, Gradients and Shadows

On top of classes to represent shape/objects there are smaller classes that are used to paint object fills or strokes. You can't add a `Gradient` or a `Shadow` to a canvas, you set those as properties of object to obtain determined effects

Image filters

FabricImage class represent a raster image on the canvas. FabricImage can be filtered with one or more filters. Filters are small programs written in WEBGL (with an optional JS fallback) that change image pixel value to obtain certain effects. Fabric.js supports numerous prebuilt fitlers for common operations and also a stack to combine more than one filter to build a particular effect.

Interactions

Fabric.js offers some prebuilt interactions between the objects on the canvas

- Selection
- Dragging
- Scaling, rotation, skewing through customizable controls
- Brushing

Selection

Fabric supports the following selection modes out of the box:

- Single object selection
- Area selection
- Multi selection

Performing state changes on an object is done using its controls. Fabric exposes the following controls:

- scaling
- rotating
- resizing
- skewing

The `Control` Class and the API is designed especially for creating custom controls and customizing existing ones either in appearances or functionalities. More on `Controls` here

Drawing & Brushes

The Canvas offer an embedded drawing mode in which the mouse move event is forwarded to a brush class that has the scope of creating an object that will represent your brush strokes. Brushing is based on the `Path` object or on a collection of circles/rect to represent a spray.

Available brushes:

- `CircleBrush` and `SprayBrush` (a spray example)
- `PencilBrush` (a classic pencil)
- `PatternBrush` (a pencil brush filled up with a pattern)

Events

Some interaction between the user of the application and the code written by the developer are handled through events. You will get an event every time a final user interact with the canvas through Fabric.js embedded functionalities for low level events like:

- mouse up/down/move
- mouse wheel
- mouse in/out
- drag and drop

- object selection created/destroyed/changed
- object added to canvas/group
- object remove from canvas/group
- object created from brushing

Animations

Fabric.js also support some basic animation utility. You can use animation library that supports objects to work with fabricJS. You can animate object positions, transform properties like scale, colors or matrices. As long as you can change the state during time from a value to aother you can create an animation.

Fabric.js animation utility is barebone if you have some specific need for some specific animation effect you are better searching a specific library to do that.

Exports

Fabric supports `JSON` and `SVG` exports.

JSON

JSON export is meant for saving and restoring visual state on the canvas. Every fabricJS object is equipped with its own `toObject` method that will output a simple JS object that can be put in storage and used in pair with `fromObject` to get back an instance of the same type. This state is supposed to restore the visual state of the canvas and not functionalities like controls. Fabric.js assumes that custom controls and custom handler are set up in your codebase as part of your app and are not part of your state.

SVG

SVG export is meant to ouput your visual canvas to a vector format output that can be imported in other softwares or printed. SVG and Canvas share a lot of similarities but are not identical. As a result SVG import and SVG export are not 1:1. Some features are supported in SVG export but are not supported in SVG import for example like TSPAN or