```
/* build: `node build.js modules=ALL exclude=json,gestures minifier=uglifyjs` */
 /*! Fabric.js Copyright 2008-2015, Printio (Juriy Zaytsev, Maxim Chernyak) */

var fabric = fabric || { version: "1.7.18" };
if (typeof exports !== 'undefined') {
  exports.fabric = fabric;
}

if (typeof document !== 'undefined' && typeof window !== 'undefined') {
  fabric.document = document;
  fabric.window = window;
  // ensure globality even if entire library were function wrapped (as in
Meteor.js packaging system)
  window.fabric = fabric;
}
else {
  // assume we're running under node.js when document/window are not present
  fabric.document = require("jsdom")
    .jsdom(
      decodeURIComponent("%3C!DOCTYPE%20html%3E%3Chtml%3E%3Chead%3E%3C%2Fhead%3E
%3Cbody%3E%3C%2Fbody%3E%3C%2Fhtml%3E")
    );

  if (fabric.document.createWindow) {
    fabric.window = fabric.document.createWindow();
  } else {
    fabric.window = fabric.document.parentWindow;
  }
}

/**
 * True when in environment that supports touch events
 * @type boolean
 */
fabric.isTouchSupported = "ontouchstart" in fabric.document.documentElement;

/**
 * True when in environment that's probably Node.js
 * @type boolean
 */
fabric.isLikelyNode = typeof Buffer !== 'undefined' &&
                      typeof window === 'undefined';

/* _FROM_SVG_START_ */
/**
 * Attributes parsed from all SVG elements
 * @type array
 */
fabric.SHARED_ATTRIBUTES = [
  "display",
  "transform",
  "fill", "fill-opacity", "fill-rule",
  "opacity",
  "stroke", "stroke-dasharray", "stroke-linecap",
  "stroke-linejoin", "stroke-miterlimit",
  "stroke-opacity", "stroke-width",
  "id"
];
/* _FROM_SVG_END_ */

/**
 * Pixel per Inch as a default value set to 96. Can be changed for more
realistic conversion.
 */
```

```
fabric.DPI = 96;
fabric.reNum = '(?:[-+]?(?:\\d+|\\d*\\.\\d+)(?:e[-+]?\\d+)?)';
fabric.fontPaths = { };
fabric.iMatrix = [1, 0, 0, 1, 0, 0];
fabric.canvasModule = 'canvas';

/**
 * Pixel limit for cache canvases. 1Mpx , 4Mpx should be fine.
 * @since 1.7.14
 * @type Number
 * @default
 */
fabric.perfLimitSizeTotal = 2097152;

/**
 * Pixel limit for cache canvases width or height. IE fixes the maximum at 5000
 * @since 1.7.14
 * @type Number
 * @default
 */
fabric.maxCacheSideLimit = 4096;

/**
 * Lowest pixel limit for cache canvases, set at 256PX
 * @since 1.7.14
 * @type Number
 * @default
 */
fabric.minCacheSideLimit = 256;

/**
 * Cache Object for widths of chars in text rendering.
 */
fabric.charWidthsCache = { };

/**
 * Device Pixel Ratio
 * @see https://developer.apple.com/library/safari/documentation/AudioVideo/
Conceptual/HTML-canvas-guide/SettingUptheCanvas/SettingUptheCanvas.html
 */
fabric.devicePixelRatio = fabric.window.devicePixelRatio ||
                          fabric.window.webkitDevicePixelRatio ||
                          fabric.window.mozDevicePixelRatio ||
                          1;


(function() {

  /**
   * @private
   * @param {String} eventName
   * @param {Function} handler
   */
  function _removeEventListener(eventName, handler) {
    if (!this.__eventListeners[eventName]) {
      return;
    }
    var eventListener = this.__eventListeners[eventName];
    if (handler) {
      eventListener[eventListener.indexOf(handler)] = false;
    }
    else {
      fabric.util.array.fill(eventListener, false);
    }
```

```
  }

  /**
   * Observes specified event
   * @deprecated `observe` deprecated since 0.8.34 (use `on` instead)
   * @memberOf fabric.Observable
   * @alias on
   * @param {String|Object} eventName Event name (eg. 'after:render') or object
with key/value pairs (eg. {'after:render': handler, 'selection:cleared':
handler})
   * @param {Function} handler Function that receives a notification when an
event of the specified type occurs
   * @return {Self} thisArg
   * @chainable
   */
  function observe(eventName, handler) {
    if (!this.__eventListeners) {
      this.__eventListeners = { };
    }
    // one object with key/value pairs was passed
    if (arguments.length === 1) {
      for (var prop in eventName) {
        this.on(prop, eventName[prop]);
      }
    }
    else {
      if (!this.__eventListeners[eventName]) {
        this.__eventListeners[eventName] = [];
      }
      this.__eventListeners[eventName].push(handler);
    }
    return this;
  }

  /**
   * Stops event observing for a particular event handler. Calling this method
   * without arguments removes all handlers for all events
   * @deprecated `stopObserving` deprecated since 0.8.34 (use `off` instead)
   * @memberOf fabric.Observable
   * @alias off
   * @param {String|Object} eventName Event name (eg. 'after:render') or object
with key/value pairs (eg. {'after:render': handler, 'selection:cleared':
handler})
   * @param {Function} handler Function to be deleted from EventListeners
   * @return {Self} thisArg
   * @chainable
   */
  function stopObserving(eventName, handler) {
    if (!this.__eventListeners) {
      return;
    }

    // remove all key/value pairs (event name -> event handler)
    if (arguments.length === 0) {
      for (eventName in this.__eventListeners) {
        _removeEventListener.call(this, eventName);
      }
    }
    // one object with key/value pairs was passed
    else if (arguments.length === 1 && typeof arguments[0] === 'object') {
      for (var prop in eventName) {
        _removeEventListener.call(this, prop, eventName[prop]);
      }
    }
```

```
    else {
      _removeEventListener.call(this, eventName, handler);
    }
    return this;
  }

  /**
   * Fires event with an optional options object
   * @deprecated `fire` deprecated since 1.0.7 (use `trigger` instead)
   * @memberOf fabric.Observable
   * @alias trigger
   * @param {String} eventName Event name to fire
   * @param {Object} [options] Options object
   * @return {Self} thisArg
   * @chainable
   */
  function fire(eventName, options) {
    if (!this.__eventListeners) {
      return;
    }

    var listenersForEvent = this.__eventListeners[eventName];
    if (!listenersForEvent) {
      return;
    }

    for (var i = 0, len = listenersForEvent.length; i < len; i++) {
      listenersForEvent[i] && listenersForEvent[i].call(this, options || { });
    }
    this.__eventListeners[eventName] = listenersForEvent.filter(function(value)
{
      return value !== false;
    });
    return this;
  }

  /**
   * @namespace fabric.Observable
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-2#events}
   * @see {@link http://fabricjs.com/events|Events demo}
   */
  fabric.Observable = {
    observe: observe,
    stopObserving: stopObserving,
    fire: fire,

    on: observe,
    off: stopObserving,
    trigger: fire
  };
})();


/**
 * @namespace fabric.Collection
 */
fabric.Collection = {

  _objects: [],

  /**
   * Adds objects to collection, Canvas or Group, then renders canvas
   * (if `renderOnAddRemove` is not `false`).
   * in case of Group no changes to bounding box are made.
```

```
   * Objects should be instances of (or inherit from) fabric.Object
   * Use of this function is highly discouraged for groups.
   * you can add a bunch of objects with the add method but then you NEED
   * to run a addWithUpdate call for the Group class or position/bbox will be
wrong.
   * @param {...fabric.Object} object Zero or more fabric instances
   * @return {Self} thisArg
   * @chainable
   */
  add: function () {
    this._objects.push.apply(this._objects, arguments);
    if (this._onObjectAdded) {
      for (var i = 0, length = arguments.length; i < length; i++) {
        this._onObjectAdded(arguments[i]);
      }
    }
    this.renderOnAddRemove && this.renderAll();
    return this;
  },

  /**
   * Inserts an object into collection at specified index, then renders canvas
(if `renderOnAddRemove` is not `false`)
   * An object should be an instance of (or inherit from) fabric.Object
   * Use of this function is highly discouraged for groups.
   * you can add a bunch of objects with the insertAt method but then you NEED
   * to run a addWithUpdate call for the Group class or position/bbox will be
wrong.
   * @param {Object} object Object to insert
   * @param {Number} index Index to insert object at
   * @param {Boolean} nonSplicing When `true`, no splicing (shifting) of objects
occurs
   * @return {Self} thisArg
   * @chainable
   */
  insertAt: function (object, index, nonSplicing) {
    var objects = this.getObjects();
    if (nonSplicing) {
      objects[index] = object;
    }
    else {
      objects.splice(index, 0, object);
    }
    this._onObjectAdded && this._onObjectAdded(object);
    this.renderOnAddRemove && this.renderAll();
    return this;
  },

  /**
   * Removes objects from a collection, then renders canvas (if
`renderOnAddRemove` is not `false`)
   * @param {...fabric.Object} object Zero or more fabric instances
   * @return {Self} thisArg
   * @chainable
   */
  remove: function() {
    var objects = this.getObjects(),
        index, somethingRemoved = false;

    for (var i = 0, length = arguments.length; i < length; i++) {
      index = objects.indexOf(arguments[i]);

      // only call onObjectRemoved if an object was actually removed
      if (index !== -1) {
```

```
        somethingRemoved = true;
        objects.splice(index, 1);
        this._onObjectRemoved && this._onObjectRemoved(arguments[i]);
      }
    }

    this.renderOnAddRemove && somethingRemoved && this.renderAll();
    return this;
  },

  /**
   * Executes given function for each object in this group
   * @param {Function} callback
   *                   Callback invoked with current object as first argument,
   *                   index - as second and an array of all objects - as third.
   *                   Callback is invoked in a context of Global Object (e.g.
`window`)
   *                   when no `context` argument is given
   *
   * @param {Object} context Context (aka thisObject)
   * @return {Self} thisArg
   * @chainable
   */
  forEachObject: function(callback, context) {
    var objects = this.getObjects();
    for (var i = 0, len = objects.length; i < len; i++) {
      callback.call(context, objects[i], i, objects);
    }
    return this;
  },

  /**
   * Returns an array of children objects of this instance
   * Type parameter introduced in 1.3.10
   * @param {String} [type] When specified, only objects of this type are
returned
   * @return {Array}
   */
  getObjects: function(type) {
    if (typeof type === 'undefined') {
      return this._objects;
    }
    return this._objects.filter(function(o) {
      return o.type === type;
    });
  },

  /**
   * Returns object at specified index
   * @param {Number} index
   * @return {Self} thisArg
   */
  item: function (index) {
    return this.getObjects()[index];
  },

  /**
   * Returns true if collection contains no objects
   * @return {Boolean} true if collection is empty
   */
  isEmpty: function () {
    return this.getObjects().length === 0;
  },
```

```
  /**
   * Returns a size of a collection (i.e: length of an array containing its
objects)
   * @return {Number} Collection size
   */
  size: function() {
    return this.getObjects().length;
  },

  /**
   * Returns true if collection contains an object
   * @param {Object} object Object to check against
   * @return {Boolean} `true` if collection contains an object
   */
  contains: function(object) {
    return this.getObjects().indexOf(object) > -1;
  },

  /**
   * Returns number representation of a collection complexity
   * @return {Number} complexity
   */
  complexity: function () {
    return this.getObjects().reduce(function (memo, current) {
      memo += current.complexity ? current.complexity() : 0;
      return memo;
    }, 0);
  }
};


/**
 * @namespace fabric.CommonMethods
 */
fabric.CommonMethods = {

  /**
   * Sets object's properties from options
   * @param {Object} [options] Options object
   */
  _setOptions: function(options) {
    for (var prop in options) {
      this.set(prop, options[prop]);
    }
  },

  /**
   * @private
   * @param {Object} [filler] Options object
   * @param {String} [property] property to set the Gradient to
   */
  _initGradient: function(filler, property) {
    if (filler && filler.colorStops && !(filler instanceof fabric.Gradient)) {
      this.set(property, new fabric.Gradient(filler));
    }
  },

  /**
   * @private
   * @param {Object} [filler] Options object
   * @param {String} [property] property to set the Pattern to
   * @param {Function} [callback] callback to invoke after pattern load
   */
  _initPattern: function(filler, property, callback) {
```

```
    if (filler && filler.source && !(filler instanceof fabric.Pattern)) {
      this.set(property, new fabric.Pattern(filler, callback));
    }
    else {
      callback && callback();
    }
  },

  /**
   * @private
   * @param {Object} [options] Options object
   */
  _initClipping: function(options) {
    if (!options.clipTo || typeof options.clipTo !== 'string') {
      return;
    }

    var functionBody = fabric.util.getFunctionBody(options.clipTo);
    if (typeof functionBody !== 'undefined') {
      this.clipTo = new Function('ctx', functionBody);
    }
  },

  /**
   * @private
   */
  _setObject: function(obj) {
    for (var prop in obj) {
      this._set(prop, obj[prop]);
    }
  },

  /**
   * Sets property to a given value. When changing position/dimension -related
properties (left, top, scale, angle, etc.) `set` does not update position of
object's borders/controls. If you need to update those, call `setCoords()`.
   * @param {String|Object} key Property name or object (if object, iterate over
the object properties)
   * @param {Object|Function} value Property value (if function, the value is
passed into it and its return value is used as a new one)
   * @return {fabric.Object} thisArg
   * @chainable
   */
  set: function(key, value) {
    if (typeof key === 'object') {
      this._setObject(key);
    }
    else {
      if (typeof value === 'function' && key !== 'clipTo') {
        this._set(key, value(this.get(key)));
      }
      else {
        this._set(key, value);
      }
    }
    return this;
  },

  _set: function(key, value) {
    this[key] = value;
  },

  /**
   * Toggles specified property from `true` to `false` or from `false` to `true`
```

```
   * @param {String} property Property to toggle
   * @return {fabric.Object} thisArg
   * @chainable
   */
  toggle: function(property) {
    var value = this.get(property);
    if (typeof value === 'boolean') {
      this.set(property, !value);
    }
    return this;
  },

  /**
   * Basic getter
   * @param {String} property Property name
   * @return {*} value of a property
   */
  get: function(property) {
    return this[property];
  }
};


(function(global) {

  var sqrt = Math.sqrt,
      atan2 = Math.atan2,
      pow = Math.pow,
      abs = Math.abs,
      PiBy180 = Math.PI / 180;

  /**
   * @namespace fabric.util
   */
  fabric.util = {

    /**
     * Removes value from an array.
     * Presence of value (and its position in an array) is determined via
`Array.prototype.indexOf`
     * @static
     * @memberOf fabric.util
     * @param {Array} array
     * @param {*} value
     * @return {Array} original array
     */
    removeFromArray: function(array, value) {
      var idx = array.indexOf(value);
      if (idx !== -1) {
        array.splice(idx, 1);
      }
      return array;
    },

    /**
     * Returns random number between 2 specified ones.
     * @static
     * @memberOf fabric.util
     * @param {Number} min lower limit
     * @param {Number} max upper limit
     * @return {Number} random value (between min and max)
     */
    getRandomInt: function(min, max) {
      return Math.floor(Math.random() * (max - min + 1)) + min;
```

```
},

/**
 * Transforms degrees to radians.
 * @static
 * @memberOf fabric.util
 * @param {Number} degrees value in degrees
 * @return {Number} value in radians
 */
degreesToRadians: function(degrees) {
  return degrees * PiBy180;
},

/**
 * Transforms radians to degrees.
 * @static
 * @memberOf fabric.util
 * @param {Number} radians value in radians
 * @return {Number} value in degrees
 */
radiansToDegrees: function(radians) {
  return radians / PiBy180;
},

/**
 * Rotates `point` around `origin` with `radians`
 * @static
 * @memberOf fabric.util
 * @param {fabric.Point} point The point to rotate
 * @param {fabric.Point} origin The origin of the rotation
 * @param {Number} radians The radians of the angle for the rotation
 * @return {fabric.Point} The new rotated point
 */
rotatePoint: function(point, origin, radians) {
  point.subtractEquals(origin);
  var v = fabric.util.rotateVector(point, radians);
  return new fabric.Point(v.x, v.y).addEquals(origin);
},

/**
 * Rotates `vector` with `radians`
 * @static
 * @memberOf fabric.util
 * @param {Object} vector The vector to rotate (x and y)
 * @param {Number} radians The radians of the angle for the rotation
 * @return {Object} The new rotated point
 */
rotateVector: function(vector, radians) {
  var sin = Math.sin(radians),
      cos = Math.cos(radians),
      rx = vector.x * cos - vector.y * sin,
      ry = vector.x * sin + vector.y * cos;
  return {
    x: rx,
    y: ry
  };
},

/**
 * Apply transform t to point p
 * @static
 * @memberOf fabric.util
 * @param  {fabric.Point} p The point to transform
 * @param  {Array} t The transform
```

```
     * @param  {Boolean} [ignoreOffset] Indicates that the offset should not be
applied
     * @return {fabric.Point} The transformed point
     */
    transformPoint: function(p, t, ignoreOffset) {
      if (ignoreOffset) {
        return new fabric.Point(
          t[0] * p.x + t[2] * p.y,
          t[1] * p.x + t[3] * p.y
        );
      }
      return new fabric.Point(
        t[0] * p.x + t[2] * p.y + t[4],
        t[1] * p.x + t[3] * p.y + t[5]
      );
    },

    /**
     * Returns coordinates of points's bounding rectangle (left, top, width,
height)
     * @param {Array} points 4 points array
     * @return {Object} Object with left, top, width, height properties
     */
    makeBoundingBoxFromPoints: function(points) {
      var xPoints = [points[0].x, points[1].x, points[2].x, points[3].x],
          minX = fabric.util.array.min(xPoints),
          maxX = fabric.util.array.max(xPoints),
          width = Math.abs(minX - maxX),
          yPoints = [points[0].y, points[1].y, points[2].y, points[3].y],
          minY = fabric.util.array.min(yPoints),
          maxY = fabric.util.array.max(yPoints),
          height = Math.abs(minY - maxY);

      return {
        left: minX,
        top: minY,
        width: width,
        height: height
      };
    },

    /**
     * Invert transformation t
     * @static
     * @memberOf fabric.util
     * @param {Array} t The transform
     * @return {Array} The inverted transform
     */
    invertTransform: function(t) {
      var a = 1 / (t[0] * t[3] - t[1] * t[2]),
          r = [a * t[3], -a * t[1], -a * t[2], a * t[0]],
          o = fabric.util.transformPoint({ x: t[4], y: t[5] }, r, true);
      r[4] = -o.x;
      r[5] = -o.y;
      return r;
    },

    /**
     * A wrapper around Number#toFixed, which contrary to native method returns
number, not string.
     * @static
     * @memberOf fabric.util
     * @param {Number|String} number number to operate on
     * @param {Number} fractionDigits number of fraction digits to "leave"
```

```
     * @return {Number}
     */
    toFixed: function(number, fractionDigits) {
      return parseFloat(Number(number).toFixed(fractionDigits));
    },

    /**
     * Converts from attribute value to pixel value if applicable.
     * Returns converted pixels or original value not converted.
     * @param {Number|String} value number to operate on
     * @param {Number} fontSize
     * @return {Number|String}
     */
    parseUnit: function(value, fontSize) {
      var unit = /\D{0,2}$/.exec(value),
          number = parseFloat(value);
      if (!fontSize) {
        fontSize = fabric.Text.DEFAULT_SVG_FONT_SIZE;
      }
      switch (unit[0]) {
        case 'mm':
          return number * fabric.DPI / 25.4;

        case 'cm':
          return number * fabric.DPI / 2.54;

        case 'in':
          return number * fabric.DPI;

        case 'pt':
          return number * fabric.DPI / 72; // or * 4 / 3

        case 'pc':
          return number * fabric.DPI / 72 * 12; // or * 16

        case 'em':
          return number * fontSize;

        default:
          return number;
      }
    },

    /**
     * Function which always returns `false`.
     * @static
     * @memberOf fabric.util
     * @return {Boolean}
     */
    falseFunction: function() {
      return false;
    },

    /**
     * Returns klass "Class" object of given namespace
     * @memberOf fabric.util
     * @param {String} type Type of object (eg. 'circle')
     * @param {String} namespace Namespace to get klass "Class" object from
     * @return {Object} klass "Class"
     */
    getKlass: function(type, namespace) {
      // capitalize first letter only
      type = fabric.util.string.camelize(type.charAt(0).toUpperCase() +
type.slice(1));
```

```
      return fabric.util.resolveNamespace(namespace)[type];
    },

    /**
     * Returns object of given namespace
     * @memberOf fabric.util
     * @param {String} namespace Namespace string e.g. 'fabric.Image.filter' or
'fabric'
     * @return {Object} Object for given namespace (default fabric)
     */
    resolveNamespace: function(namespace) {
      if (!namespace) {
        return fabric;
      }

      var parts = namespace.split('.'),
          len = parts.length, i,
          obj = global || fabric.window;

      for (i = 0; i < len; ++i) {
        obj = obj[parts[i]];
      }

      return obj;
    },

    /**
     * Loads image element from given url and passes it to a callback
     * @memberOf fabric.util
     * @param {String} url URL representing an image
     * @param {Function} callback Callback; invoked with loaded image
     * @param {*} [context] Context to invoke callback in
     * @param {Object} [crossOrigin] crossOrigin value to set image element to
     */
    loadImage: function(url, callback, context, crossOrigin) {
      if (!url) {
        callback && callback.call(context, url);
        return;
      }

      var img = fabric.util.createImage();

      /** @ignore */
      img.onload = function () {
        callback && callback.call(context, img);
        img = img.onload = img.onerror = null;
      };

      /** @ignore */
      img.onerror = function() {
        fabric.log('Error loading ' + img.src);
        callback && callback.call(context, null, true);
        img = img.onload = img.onerror = null;
      };

      // data-urls appear to be buggy with crossOrigin
      //
https://github.com/kangax/fabric.js/commit/d0abb90f1cd5c5ef9d2a94d3fb21a22330da3
e0a#commitcomment-4513767
      // see https://code.google.com/p/chromium/issues/detail?id=315152
      //     https://bugzilla.mozilla.org/show_bug.cgi?id=935069
      if (url.indexOf('data') !== 0 && crossOrigin) {
        img.crossOrigin = crossOrigin;
      }
```

```
      img.src = url;
    },

    /**
     * Creates corresponding fabric instances from their object representations
     * @static
     * @memberOf fabric.util
     * @param {Array} objects Objects to enliven
     * @param {Function} callback Callback to invoke when all objects are
created
     * @param {String} namespace Namespace to get klass "Class" object from
     * @param {Function} reviver Method for further parsing of object elements,
     * called after each fabric object created.
     */
    enlivenObjects: function(objects, callback, namespace, reviver) {
      objects = objects || [];

      function onLoaded() {
        if (++numLoadedObjects === numTotalObjects) {
          callback && callback(enlivenedObjects);
        }
      }

      var enlivenedObjects = [],
          numLoadedObjects = 0,
          numTotalObjects = objects.length,
          forceAsync = true;

      if (!numTotalObjects) {
        callback && callback(enlivenedObjects);
        return;
      }

      objects.forEach(function (o, index) {
        // if sparse array
        if (!o || !o.type) {
          onLoaded();
          return;
        }
        var klass = fabric.util.getKlass(o.type, namespace);
        klass.fromObject(o, function (obj, error) {
          error || (enlivenedObjects[index] = obj);
          reviver && reviver(o, obj, error);
          onLoaded();
        }, forceAsync);
      });
    },

    /**
     * Create and wait for loading of patterns
     * @static
     * @memberOf fabric.util
     * @param {Array} objects Objects to enliven
     * @param {Function} callback Callback to invoke when all objects are
created
     * @param {String} namespace Namespace to get klass "Class" object from
     * @param {Function} reviver Method for further parsing of object elements,
     * called after each fabric object created.
     */
    enlivenPatterns: function(patterns, callback) {
      patterns = patterns || [];

      function onLoaded() {
```

```
        if (++numLoadedPatterns === numPatterns) {
          callback && callback(enlivenedPatterns);
        }
      }

      var enlivenedPatterns = [],
          numLoadedPatterns = 0,
          numPatterns = patterns.length;

      if (!numPatterns) {
        callback && callback(enlivenedPatterns);
        return;
      }

      patterns.forEach(function (p, index) {
        if (p && p.source) {
          new fabric.Pattern(p, function(pattern) {
            enlivenedPatterns[index] = pattern;
            onLoaded();
          });
        }
        else {
          enlivenedPatterns[index] = p;
          onLoaded();
        }
      });
    },

    /**
     * Groups SVG elements (usually those retrieved from SVG document)
     * @static
     * @memberOf fabric.util
     * @param {Array} elements SVG elements to group
     * @param {Object} [options] Options object
     * @param {String} path Value to set sourcePath to
     * @return {fabric.Object|fabric.PathGroup}
     */
    groupSVGElements: function(elements, options, path) {
      var object;

      object = new fabric.PathGroup(elements, options);

      if (typeof path !== 'undefined') {
        object.sourcePath = path;
      }
      return object;
    },

    /**
     * Populates an object with properties of another object
     * @static
     * @memberOf fabric.util
     * @param {Object} source Source object
     * @param {Object} destination Destination object
     * @return {Array} properties Propertie names to include
     */
    populateWithProperties: function(source, destination, properties) {
      if (properties && Object.prototype.toString.call(properties) === '[object
Array]') {
        for (var i = 0, len = properties.length; i < len; i++) {
          if (properties[i] in source) {
            destination[properties[i]] = source[properties[i]];
          }
        }
```

```
      }
    },

    /**
     * Draws a dashed line between two points
     *
     * This method is used to draw dashed line around selection area.
     * See <a href="http://stackoverflow.com/questions/4576724/dotted-stroke-in-
canvas">dotted stroke in canvas</a>
     *
     * @param {CanvasRenderingContext2D} ctx context
     * @param {Number} x  start x coordinate
     * @param {Number} y start y coordinate
     * @param {Number} x2 end x coordinate
     * @param {Number} y2 end y coordinate
     * @param {Array} da dash array pattern
     */
    drawDashedLine: function(ctx, x, y, x2, y2, da) {
      var dx = x2 - x,
          dy = y2 - y,
          len = sqrt(dx * dx + dy * dy),
          rot = atan2(dy, dx),
          dc = da.length,
          di = 0,
          draw = true;

      ctx.save();
      ctx.translate(x, y);
      ctx.moveTo(0, 0);
      ctx.rotate(rot);

      x = 0;
      while (len > x) {
        x += da[di++ % dc];
        if (x > len) {
          x = len;
        }
        ctx[draw ? 'lineTo' : 'moveTo'](x, 0);
        draw = !draw;
      }

      ctx.restore();
    },

    /**
     * Creates canvas element and initializes it via excanvas if necessary
     * @static
     * @memberOf fabric.util
     * @param {CanvasElement} [canvasEl] optional canvas element to initialize;
     * when not given, element is created implicitly
     * @return {CanvasElement} initialized canvas element
     */
    createCanvasElement: function(canvasEl) {
      canvasEl || (canvasEl = fabric.document.createElement('canvas'));
      /* eslint-disable camelcase */
      if (!canvasEl.getContext && typeof G_vmlCanvasManager !== 'undefined') {
        G_vmlCanvasManager.initElement(canvasEl);
      }
      /* eslint-enable camelcase */
      return canvasEl;
    },

    /**
     * Creates image element (works on client and node)
```

```
     * @static
     * @memberOf fabric.util
     * @return {HTMLImageElement} HTML image element
     */
    createImage: function() {
      return fabric.isLikelyNode
        ? new (require('canvas').Image)()
        : fabric.document.createElement('img');
    },

    /**
     * Creates accessors (getXXX, setXXX) for a "class", based on
"stateProperties" array
     * @static
     * @memberOf fabric.util
     * @param {Object} klass "Class" to create accessors for
     */
    createAccessors: function(klass) {
      var proto = klass.prototype, i, propName,
          capitalizedPropName, setterName, getterName;

      for (i = proto.stateProperties.length; i--; ) {

        propName = proto.stateProperties[i];
        capitalizedPropName = propName.charAt(0).toUpperCase() +
propName.slice(1);
        setterName = 'set' + capitalizedPropName;
        getterName = 'get' + capitalizedPropName;

        // using `new Function` for better introspection
        if (!proto[getterName]) {
          proto[getterName] = (function(property) {
            return new Function('return this.get("' + property + '")');
          })(propName);
        }
        if (!proto[setterName]) {
          proto[setterName] = (function(property) {
            return new Function('value', 'return this.set("' + property + '",
value)');
          })(propName);
        }
      }
    },

    /**
     * @static
     * @memberOf fabric.util
     * @param {fabric.Object} receiver Object implementing `clipTo` method
     * @param {CanvasRenderingContext2D} ctx Context to clip
     */
    clipContext: function(receiver, ctx) {
      ctx.save();
      ctx.beginPath();
      receiver.clipTo(ctx);
      ctx.clip();
    },

    /**
     * Multiply matrix A by matrix B to nest transformations
     * @static
     * @memberOf fabric.util
     * @param  {Array} a First transformMatrix
     * @param  {Array} b Second transformMatrix
     * @param  {Boolean} is2x2 flag to multiply matrices as 2x2 matrices
```

```
     * @return {Array} The product of the two transform matrices
     */
    multiplyTransformMatrices: function(a, b, is2x2) {
      // Matrix multiply a * b
      return [
        a[0] * b[0] + a[2] * b[1],
        a[1] * b[0] + a[3] * b[1],
        a[0] * b[2] + a[2] * b[3],
        a[1] * b[2] + a[3] * b[3],
        is2x2 ? 0 : a[0] * b[4] + a[2] * b[5] + a[4],
        is2x2 ? 0 : a[1] * b[4] + a[3] * b[5] + a[5]
      ];
    },

    /**
     * Decomposes standard 2x2 matrix into transform componentes
     * @static
     * @memberOf fabric.util
     * @param  {Array} a transformMatrix
     * @return {Object} Components of transform
     */
    qrDecompose: function(a) {
      var angle = atan2(a[1], a[0]),
          denom = pow(a[0], 2) + pow(a[1], 2),
          scaleX = sqrt(denom),
          scaleY = (a[0] * a[3] - a[2] * a [1]) / scaleX,
          skewX = atan2(a[0] * a[2] + a[1] * a [3], denom);
      return {
        angle: angle  / PiBy180,
        scaleX: scaleX,
        scaleY: scaleY,
        skewX: skewX / PiBy180,
        skewY: 0,
        translateX: a[4],
        translateY: a[5]
      };
    },

    customTransformMatrix: function(scaleX, scaleY, skewX) {
      var skewMatrixX = [1, 0, abs(Math.tan(skewX * PiBy180)), 1],
          scaleMatrix = [abs(scaleX), 0, 0, abs(scaleY)];
      return fabric.util.multiplyTransformMatrices(scaleMatrix, skewMatrixX,
true);
    },

    resetObjectTransform: function (target) {
      target.scaleX = 1;
      target.scaleY = 1;
      target.skewX = 0;
      target.skewY = 0;
      target.flipX = false;
      target.flipY = false;
      target.setAngle(0);
    },

    /**
     * Returns string representation of function body
     * @param {Function} fn Function to get body of
     * @return {String} Function body
     */
    getFunctionBody: function(fn) {
      return (String(fn).match(/function[^{]*\{([\s\S]*)\}/) || {})[1];
    },
```

```
    /**
     * Returns true if context has transparent pixel
     * at specified location (taking tolerance into account)
     * @param {CanvasRenderingContext2D} ctx context
     * @param {Number} x x coordinate
     * @param {Number} y y coordinate
     * @param {Number} tolerance Tolerance
     */
    isTransparent: function(ctx, x, y, tolerance) {

      // If tolerance is > 0 adjust start coords to take into account.
      // If moves off Canvas fix to 0
      if (tolerance > 0) {
        if (x > tolerance) {
          x -= tolerance;
        }
        else {
          x = 0;
        }
        if (y > tolerance) {
          y -= tolerance;
        }
        else {
          y = 0;
        }
      }

      var _isTransparent = true, i, temp,
          imageData = ctx.getImageData(x, y, (tolerance * 2) || 1, (tolerance *
2) || 1),
          l = imageData.data.length;

      // Split image data - for tolerance > 1, pixelDataSize = 4;
      for (i = 3; i < l; i += 4) {
        temp = imageData.data[i];
        _isTransparent = temp <= 0;
        if (_isTransparent === false) {
          break; // Stop if colour found
        }
      }

      imageData = null;

      return _isTransparent;
    },

    /**
     * Parse preserveAspectRatio attribute from element
     * @param {string} attribute to be parsed
     * @return {Object} an object containing align and meetOrSlice attribute
     */
    parsePreserveAspectRatioAttribute: function(attribute) {
      var meetOrSlice = 'meet', alignX = 'Mid', alignY = 'Mid',
          aspectRatioAttrs = attribute.split(' '), align;

      if (aspectRatioAttrs && aspectRatioAttrs.length) {
        meetOrSlice = aspectRatioAttrs.pop();
        if (meetOrSlice !== 'meet' && meetOrSlice !== 'slice') {
          align = meetOrSlice;
          meetOrSlice = 'meet';
        }
        else if (aspectRatioAttrs.length) {
          align = aspectRatioAttrs.pop();
        }
```

```
      }
      //divide align in alignX and alignY
      alignX = align !== 'none' ? align.slice(1, 4) : 'none';
      alignY = align !== 'none' ? align.slice(5, 8) : 'none';
      return {
        meetOrSlice: meetOrSlice,
        alignX: alignX,
        alignY: alignY
      };
    },

    /**
     * Clear char widths cache for a font family.
     * @memberOf fabric.util
     * @param {String} [fontFamily] font family to clear
     */
    clearFabricFontCache: function(fontFamily) {
      if (!fontFamily) {
        fabric.charWidthsCache = { };
      }
      else if (fabric.charWidthsCache[fontFamily]) {
        delete fabric.charWidthsCache[fontFamily];
      }
    },

    /**
     * Clear char widths cache for a font family.
     * @memberOf fabric.util
     * @param {Number} ar aspect ratio
     * @param {Number} maximumArea Maximum area you want to achieve
     * @param {Number} maximumSide biggest side allowed
     * @return {Object.x} Limited dimensions by X
     * @return {Object.y} Limited dimensions by Y
     */
    limitDimsByArea: function(ar, maximumArea) {
      var roughWidth = Math.sqrt(maximumArea * ar),
          perfLimitSizeY = Math.floor(maximumArea / roughWidth);
      return { x: Math.floor(roughWidth), y: perfLimitSizeY };
    },

    capValue: function(min, value, max) {
      return Math.max(min, Math.min(value, max));
    }
  };

})(typeof exports !== 'undefined' ? exports : this);


(function() {

  var arcToSegmentsCache = { },
      segmentToBezierCache = { },
      boundsOfCurveCache = { },
      _join = Array.prototype.join;

  /* Adapted from
http://dxr.mozilla.org/mozilla-central/source/content/svg/content/src/
nsSVGPathDataParser.cpp
   * by Andrea Bogazzi code is under MPL. if you don't have a copy of the
license you can take it here
   * http://mozilla.org/MPL/2.0/
   */
  function arcToSegments(toX, toY, rx, ry, large, sweep, rotateX) {
    var argsString = _join.call(arguments);
```

```
    if (arcToSegmentsCache[argsString]) {
      return arcToSegmentsCache[argsString];
    }

    var PI = Math.PI, th = rotateX * PI / 180,
        sinTh = Math.sin(th),
        cosTh = Math.cos(th),
        fromX = 0, fromY = 0;

    rx = Math.abs(rx);
    ry = Math.abs(ry);

    var px = -cosTh * toX * 0.5 - sinTh * toY * 0.5,
        py = -cosTh * toY * 0.5 + sinTh * toX * 0.5,
        rx2 = rx * rx, ry2 = ry * ry, py2 = py * py, px2 = px * px,
        pl = rx2 * ry2 - rx2 * py2 - ry2 * px2,
        root = 0;

    if (pl < 0) {
      var s = Math.sqrt(1 - pl / (rx2 * ry2));
      rx *= s;
      ry *= s;
    }
    else {
      root = (large === sweep ? -1.0 : 1.0) *
              Math.sqrt( pl / (rx2 * py2 + ry2 * px2));
    }

    var cx = root * rx * py / ry,
        cy = -root * ry * px / rx,
        cx1 = cosTh * cx - sinTh * cy + toX * 0.5,
        cy1 = sinTh * cx + cosTh * cy + toY * 0.5,
        mTheta = calcVectorAngle(1, 0, (px - cx) / rx, (py - cy) / ry),
        dtheta = calcVectorAngle((px - cx) / rx, (py - cy) / ry, (-px - cx) /
rx, (-py - cy) / ry);

    if (sweep === 0 && dtheta > 0) {
      dtheta -= 2 * PI;
    }
    else if (sweep === 1 && dtheta < 0) {
      dtheta += 2 * PI;
    }

    // Convert into cubic bezier segments <= 90deg
    var segments = Math.ceil(Math.abs(dtheta / PI * 2)),
        result = [], mDelta = dtheta / segments,
        mT = 8 / 3 * Math.sin(mDelta / 4) * Math.sin(mDelta / 4) /
Math.sin(mDelta / 2),
        th3 = mTheta + mDelta;

    for (var i = 0; i < segments; i++) {
      result[i] = segmentToBezier(mTheta, th3, cosTh, sinTh, rx, ry, cx1, cy1,
mT, fromX, fromY);
      fromX = result[i][4];
      fromY = result[i][5];
      mTheta = th3;
      th3 += mDelta;
    }
    arcToSegmentsCache[argsString] = result;
    return result;
  }

  function segmentToBezier(th2, th3, cosTh, sinTh, rx, ry, cx1, cy1, mT, fromX,
fromY) {
```

```
  var argsString2 = _join.call(arguments);
  if (segmentToBezierCache[argsString2]) {
    return segmentToBezierCache[argsString2];
  }

  var costh2 = Math.cos(th2),
      sinth2 = Math.sin(th2),
      costh3 = Math.cos(th3),
      sinth3 = Math.sin(th3),
      toX = cosTh * rx * costh3 - sinTh * ry * sinth3 + cx1,
      toY = sinTh * rx * costh3 + cosTh * ry * sinth3 + cy1,
      cp1X = fromX + mT * ( -cosTh * rx * sinth2 - sinTh * ry * costh2),
      cp1Y = fromY + mT * ( -sinTh * rx * sinth2 + cosTh * ry * costh2),
      cp2X = toX + mT * ( cosTh * rx * sinth3 + sinTh * ry * costh3),
      cp2Y = toY + mT * ( sinTh * rx * sinth3 - cosTh * ry * costh3);

  segmentToBezierCache[argsString2] = [
    cp1X, cp1Y,
    cp2X, cp2Y,
    toX, toY
  ];
  return segmentToBezierCache[argsString2];
}

/*
 * Private
 */
function calcVectorAngle(ux, uy, vx, vy) {
  var ta = Math.atan2(uy, ux),
      tb = Math.atan2(vy, vx);
  if (tb >= ta) {
    return tb - ta;
  }
  else {
    return 2 * Math.PI - (ta - tb);
  }
}

/**
 * Draws arc
 * @param {CanvasRenderingContext2D} ctx
 * @param {Number} fx
 * @param {Number} fy
 * @param {Array} coords
 */
fabric.util.drawArc = function(ctx, fx, fy, coords) {
  var rx = coords[0],
      ry = coords[1],
      rot = coords[2],
      large = coords[3],
      sweep = coords[4],
      tx = coords[5],
      ty = coords[6],
      segs = [[], [], [], []],
      segsNorm = arcToSegments(tx - fx, ty - fy, rx, ry, large, sweep, rot);

  for (var i = 0, len = segsNorm.length; i < len; i++) {
    segs[i][0] = segsNorm[i][0] + fx;
    segs[i][1] = segsNorm[i][1] + fy;
    segs[i][2] = segsNorm[i][2] + fx;
    segs[i][3] = segsNorm[i][3] + fy;
    segs[i][4] = segsNorm[i][4] + fx;
    segs[i][5] = segsNorm[i][5] + fy;
    ctx.bezierCurveTo.apply(ctx, segs[i]);
```

```
    }
  };

  /**
   * Calculate bounding box of a elliptic-arc
   * @param {Number} fx start point of arc
   * @param {Number} fy
   * @param {Number} rx horizontal radius
   * @param {Number} ry vertical radius
   * @param {Number} rot angle of horizontal axe
   * @param {Number} large 1 or 0, whatever the arc is the big or the small on
the 2 points
   * @param {Number} sweep 1 or 0, 1 clockwise or counterclockwise direction
   * @param {Number} tx end point of arc
   * @param {Number} ty
   */
  fabric.util.getBoundsOfArc = function(fx, fy, rx, ry, rot, large, sweep, tx,
ty) {

    var fromX = 0, fromY = 0, bound, bounds = [],
        segs = arcToSegments(tx - fx, ty - fy, rx, ry, large, sweep, rot);

    for (var i = 0, len = segs.length; i < len; i++) {
      bound = getBoundsOfCurve(fromX, fromY, segs[i][0], segs[i][1], segs[i][2],
segs[i][3], segs[i][4], segs[i][5]);
      bounds.push({ x: bound[0].x + fx, y: bound[0].y + fy });
      bounds.push({ x: bound[1].x + fx, y: bound[1].y + fy });
      fromX = segs[i][4];
      fromY = segs[i][5];
    }
    return bounds;
  };

  /**
   * Calculate bounding box of a beziercurve
   * @param {Number} x0 starting point
   * @param {Number} y0
   * @param {Number} x1 first control point
   * @param {Number} y1
   * @param {Number} x2 secondo control point
   * @param {Number} y2
   * @param {Number} x3 end of beizer
   * @param {Number} y3
   */
  // taken from http://jsbin.com/ivomiq/56/edit  no credits available for that.
  function getBoundsOfCurve(x0, y0, x1, y1, x2, y2, x3, y3) {
    var argsString = _join.call(arguments);
    if (boundsOfCurveCache[argsString]) {
      return boundsOfCurveCache[argsString];
    }

    var sqrt = Math.sqrt,
        min = Math.min, max = Math.max,
        abs = Math.abs, tvalues = [],
        bounds = [[], []],
        a, b, c, t, t1, t2, b2ac, sqrtb2ac;

    b = 6 * x0 - 12 * x1 + 6 * x2;
    a = -3 * x0 + 9 * x1 - 9 * x2 + 3 * x3;
    c = 3 * x1 - 3 * x0;

    for (var i = 0; i < 2; ++i) {
      if (i > 0) {
        b = 6 * y0 - 12 * y1 + 6 * y2;
```

```
        a = -3 * y0 + 9 * y1 - 9 * y2 + 3 * y3;
        c = 3 * y1 - 3 * y0;
      }

      if (abs(a) < 1e-12) {
        if (abs(b) < 1e-12) {
          continue;
        }
        t = -c / b;
        if (0 < t && t < 1) {
          tvalues.push(t);
        }
        continue;
      }
      b2ac = b * b - 4 * c * a;
      if (b2ac < 0) {
        continue;
      }
      sqrtb2ac = sqrt(b2ac);
      t1 = (-b + sqrtb2ac) / (2 * a);
      if (0 < t1 && t1 < 1) {
        tvalues.push(t1);
      }
      t2 = (-b - sqrtb2ac) / (2 * a);
      if (0 < t2 && t2 < 1) {
        tvalues.push(t2);
      }
    }

    var x, y, j = tvalues.length, jlen = j, mt;
    while (j--) {
      t = tvalues[j];
      mt = 1 - t;
      x = (mt * mt * mt * x0) + (3 * mt * mt * t * x1) + (3 * mt * t * t * x2) +
(t * t * t * x3);
      bounds[0][j] = x;

      y = (mt * mt * mt * y0) + (3 * mt * mt * t * y1) + (3 * mt * t * t * y2) +
(t * t * t * y3);
      bounds[1][j] = y;
    }

    bounds[0][jlen] = x0;
    bounds[1][jlen] = y0;
    bounds[0][jlen + 1] = x3;
    bounds[1][jlen + 1] = y3;
    var result = [
      {
        x: min.apply(null, bounds[0]),
        y: min.apply(null, bounds[1])
      },
      {
        x: max.apply(null, bounds[0]),
        y: max.apply(null, bounds[1])
      }
    ];
    boundsOfCurveCache[argsString] = result;
    return result;
  }

  fabric.util.getBoundsOfCurve = getBoundsOfCurve;

})();
```

```
(function() {

  var slice = Array.prototype.slice;

  /* _ES5_COMPAT_START_ */

  if (!Array.prototype.indexOf) {
    /**
     * Finds index of an element in an array
     * @param {*} searchElement
     * @return {Number}
     */
    Array.prototype.indexOf = function (searchElement /*, fromIndex */ ) {
      if (this === void 0 || this === null) {
        throw new TypeError();
      }
      var t = Object(this), len = t.length >>> 0;
      if (len === 0) {
        return -1;
      }
      var n = 0;
      if (arguments.length > 0) {
        n = Number(arguments[1]);
        if (n !== n) { // shortcut for verifying if it's NaN
          n = 0;
        }
        else if (n !== 0 && n !== Number.POSITIVE_INFINITY && n !==
Number.NEGATIVE_INFINITY) {
          n = (n > 0 || -1) * Math.floor(Math.abs(n));
        }
      }
      if (n >= len) {
        return -1;
      }
      var k = n >= 0 ? n : Math.max(len - Math.abs(n), 0);
      for (; k < len; k++) {
        if (k in t && t[k] === searchElement) {
          return k;
        }
      }
      return -1;
    };
  }

  if (!Array.prototype.forEach) {
    /**
     * Iterates an array, invoking callback for each element
     * @param {Function} fn Callback to invoke for each element
     * @param {Object} [context] Context to invoke callback in
     * @return {Array}
     */
    Array.prototype.forEach = function(fn, context) {
      for (var i = 0, len = this.length >>> 0; i < len; i++) {
        if (i in this) {
          fn.call(context, this[i], i, this);
        }
      }
    };
  }

  if (!Array.prototype.map) {
    /**
     * Returns a result of iterating over an array, invoking callback for each
```

```
element
     * @param {Function} fn Callback to invoke for each element
     * @param {Object} [context] Context to invoke callback in
     * @return {Array}
     */
    Array.prototype.map = function(fn, context) {
      var result = [];
      for (var i = 0, len = this.length >>> 0; i < len; i++) {
        if (i in this) {
          result[i] = fn.call(context, this[i], i, this);
        }
      }
      return result;
    };
  }

  if (!Array.prototype.every) {
    /**
     * Returns true if a callback returns truthy value for all elements in an
array
     * @param {Function} fn Callback to invoke for each element
     * @param {Object} [context] Context to invoke callback in
     * @return {Boolean}
     */
    Array.prototype.every = function(fn, context) {
      for (var i = 0, len = this.length >>> 0; i < len; i++) {
        if (i in this && !fn.call(context, this[i], i, this)) {
          return false;
        }
      }
      return true;
    };
  }

  if (!Array.prototype.some) {
    /**
     * Returns true if a callback returns truthy value for at least one element
in an array
     * @param {Function} fn Callback to invoke for each element
     * @param {Object} [context] Context to invoke callback in
     * @return {Boolean}
     */
    Array.prototype.some = function(fn, context) {
      for (var i = 0, len = this.length >>> 0; i < len; i++) {
        if (i in this && fn.call(context, this[i], i, this)) {
          return true;
        }
      }
      return false;
    };
  }

  if (!Array.prototype.filter) {
    /**
     * Returns the result of iterating over elements in an array
     * @param {Function} fn Callback to invoke for each element
     * @param {Object} [context] Context to invoke callback in
     * @return {Array}
     */
    Array.prototype.filter = function(fn, context) {
      var result = [], val;
      for (var i = 0, len = this.length >>> 0; i < len; i++) {
        if (i in this) {
          val = this[i]; // in case fn mutates this
```

```
          if (fn.call(context, val, i, this)) {
            result.push(val);
          }
        }
      }
      return result;
    };
  }

  if (!Array.prototype.reduce) {
    /**
     * Returns "folded" (reduced) result of iterating over elements in an array
     * @param {Function} fn Callback to invoke for each element
     * @return {*}
     */
    Array.prototype.reduce = function(fn /*, initial*/) {
      var len = this.length >>> 0,
          i = 0,
          rv;

      if (arguments.length > 1) {
        rv = arguments[1];
      }
      else {
        do {
          if (i in this) {
            rv = this[i++];
            break;
          }
          // if array contains no values, no initial value to return
          if (++i >= len) {
            throw new TypeError();
          }
        }
        while (true);
      }
      for (; i < len; i++) {
        if (i in this) {
          rv = fn.call(null, rv, this[i], i, this);
        }
      }
      return rv;
    };
  }

  /* _ES5_COMPAT_END_ */

  /**
   * Invokes method on all items in a given array
   * @memberOf fabric.util.array
   * @param {Array} array Array to iterate over
   * @param {String} method Name of a method to invoke
   * @return {Array}
   */
  function invoke(array, method) {
    var args = slice.call(arguments, 2), result = [];
    for (var i = 0, len = array.length; i < len; i++) {
      result[i] = args.length ? array[i][method].apply(array[i], args) :
array[i][method].call(array[i]);
    }
    return result;
  }

  /**
```

```
 * Finds maximum value in array (not necessarily "first" one)
 * @memberOf fabric.util.array
 * @param {Array} array Array to iterate over
 * @param {String} byProperty
 * @return {*}
 */
function max(array, byProperty) {
  return find(array, byProperty, function(value1, value2) {
    return value1 >= value2;
  });
}

/**
 * Finds minimum value in array (not necessarily "first" one)
 * @memberOf fabric.util.array
 * @param {Array} array Array to iterate over
 * @param {String} byProperty
 * @return {*}
 */
function min(array, byProperty) {
  return find(array, byProperty, function(value1, value2) {
    return value1 < value2;
  });
}

/**
 * @private
 */
function fill(array, value) {
  var k = array.length;
  while (k--) {
    array[k] = value;
  }
  return array;
}

/**
 * @private
 */
function find(array, byProperty, condition) {
  if (!array || array.length === 0) {
    return;
  }

  var i = array.length - 1,
      result = byProperty ? array[i][byProperty] : array[i];
  if (byProperty) {
    while (i--) {
      if (condition(array[i][byProperty], result)) {
        result = array[i][byProperty];
      }
    }
  }
  else {
    while (i--) {
      if (condition(array[i], result)) {
        result = array[i];
      }
    }
  }
  return result;
}

/**
```

```
   * @namespace fabric.util.array
   */
  fabric.util.array = {
    fill: fill,
    invoke: invoke,
    min: min,
    max: max
  };

})();


(function() {
  /**
   * Copies all enumerable properties of one js object to another
   * Does not clone or extend fabric.Object subclasses.
   * @memberOf fabric.util.object
   * @param {Object} destination Where to copy to
   * @param {Object} source Where to copy from
   * @return {Object}
   */

  function extend(destination, source, deep) {
    // JScript DontEnum bug is not taken care of
    // the deep clone is for internal use, is not meant to avoid
    // javascript traps or cloning html element or self referenced objects.
    if (deep) {
      if (!fabric.isLikelyNode && source instanceof Element) {
        // avoid cloning deep images, canvases,
        destination = source;
      }
      else if (source instanceof Array) {
        destination = [];
        for (var i = 0, len = source.length; i < len; i++) {
          destination[i] = extend({ }, source[i], deep);
        }
      }
      else if (source && typeof source === 'object') {
        for (var property in source) {
          if (source.hasOwnProperty(property)) {
            destination[property] = extend({ }, source[property], deep);
          }
        }
      }
      else {
        // this sounds odd for an extend but is ok for recursive use
        destination = source;
      }
    }
    else {
      for (var property in source) {
        destination[property] = source[property];
      }
    }
    return destination;
  }

  /**
   * Creates an empty object and copies all enumerable properties of another
object to it
   * @memberOf fabric.util.object
   * @param {Object} object Object to clone
   * @return {Object}
   */
```

```
  function clone(object, deep) {
    return extend({ }, object, deep);
  }

  /** @namespace fabric.util.object */
  fabric.util.object = {
    extend: extend,
    clone: clone
  };

})();


(function() {

  /* _ES5_COMPAT_START_ */
  if (!String.prototype.trim) {
    /**
     * Trims a string (removing whitespace from the beginning and the end)
     * @function external:String#trim
     * @see <a
href="https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/
String/Trim">String#trim on MDN</a>
     */
    String.prototype.trim = function () {
      // this trim is not fully ES3 or ES5 compliant, but it should cover most
cases for now
      return this.replace(/^[\s\xA0]+/, '').replace(/[\s\xA0]+$/, '');
    };
  }
  /* _ES5_COMPAT_END_ */

  /**
   * Camelizes a string
   * @memberOf fabric.util.string
   * @param {String} string String to camelize
   * @return {String} Camelized version of a string
   */
  function camelize(string) {
    return string.replace(/-+(.)?/g, function(match, character) {
      return character ? character.toUpperCase() : '';
    });
  }

  /**
   * Capitalizes a string
   * @memberOf fabric.util.string
   * @param {String} string String to capitalize
   * @param {Boolean} [firstLetterOnly] If true only first letter is capitalized
   * and other letters stay untouched, if false first letter is capitalized
   * and other letters are converted to lowercase.
   * @return {String} Capitalized version of a string
   */
  function capitalize(string, firstLetterOnly) {
    return string.charAt(0).toUpperCase() +
      (firstLetterOnly ? string.slice(1) : string.slice(1).toLowerCase());
  }

  /**
   * Escapes XML in a string
   * @memberOf fabric.util.string
   * @param {String} string String to escape
   * @return {String} Escaped version of a string
   */
```

```
  function escapeXml(string) {
    return string.replace(/&/g, '&amp;')
       .replace(/"/g, '&quot;')
       .replace(/'/g, '&apos;')
       .replace(/</g, '&lt;')
       .replace(/>/g, '&gt;');
  }

  /**
   * String utilities
   * @namespace fabric.util.string
   */
  fabric.util.string = {
    camelize: camelize,
    capitalize: capitalize,
    escapeXml: escapeXml
  };
})();


/* _ES5_COMPAT_START_ */
(function() {

  var slice = Array.prototype.slice,
      apply = Function.prototype.apply,
      Dummy = function() { };

  if (!Function.prototype.bind) {
    /**
     * Cross-browser approximation of ES5 Function.prototype.bind (not fully
spec conforming)
     * @see <a
href="https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/
Function/bind">Function#bind on MDN</a>
     * @param {Object} thisArg Object to bind function to
     * @param {Any[]} Values to pass to a bound function
     * @return {Function}
     */
    Function.prototype.bind = function(thisArg) {
      var _this = this, args = slice.call(arguments, 1), bound;
      if (args.length) {
        bound = function() {
          return apply.call(_this, this instanceof Dummy ? this : thisArg,
args.concat(slice.call(arguments)));
        };
      }
      else {
        /** @ignore */
        bound = function() {
          return apply.call(_this, this instanceof Dummy ? this : thisArg,
arguments);
        };
      }
      Dummy.prototype = this.prototype;
      bound.prototype = new Dummy();

      return bound;
    };
  }

})();
/* _ES5_COMPAT_END_ */
```

```
(function() {

  var slice = Array.prototype.slice, emptyFunction = function() { },

      IS_DONTENUM_BUGGY = (function() {
        for (var p in { toString: 1 }) {
          if (p === 'toString') {
            return false;
          }
        }
        return true;
      })(),

      /** @ignore */
      addMethods = function(klass, source, parent) {
        for (var property in source) {

          if (property in klass.prototype &&
              typeof klass.prototype[property] === 'function' &&
              (source[property] + '').indexOf('callSuper') > -1) {

            klass.prototype[property] = (function(property) {
              return function() {

                var superclass = this.constructor.superclass;
                this.constructor.superclass = parent;
                var returnValue = source[property].apply(this, arguments);
                this.constructor.superclass = superclass;

                if (property !== 'initialize') {
                  return returnValue;
                }
              };
            })(property);
          }
          else {
            klass.prototype[property] = source[property];
          }

          if (IS_DONTENUM_BUGGY) {
            if (source.toString !== Object.prototype.toString) {
              klass.prototype.toString = source.toString;
            }
            if (source.valueOf !== Object.prototype.valueOf) {
              klass.prototype.valueOf = source.valueOf;
            }
          }
        }
      };

  function Subclass() { }

  function callSuper(methodName) {
    var parentMethod = null,
        _this = this;

    // climb prototype chain to find method not equal to callee's method
    while (_this.constructor.superclass) {
      var superClassMethod = _this.constructor.superclass.prototype[methodName];
      if (_this[methodName] !== superClassMethod) {
        parentMethod = superClassMethod;
        break;
      }
      // eslint-disable-next-line
```

```
      _this = _this.constructor.superclass.prototype;
    }

    if (!parentMethod) {
      return console.log('tried to callSuper ' + methodName + ', method not
found in prototype chain', this);
    }

    return (arguments.length > 1)
      ? parentMethod.apply(this, slice.call(arguments, 1))
      : parentMethod.call(this);
  }

  /**
   * Helper for creation of "classes".
   * @memberOf fabric.util
   * @param {Function} [parent] optional "Class" to inherit from
   * @param {Object} [properties] Properties shared by all instances of this
class
   *                  (be careful modifying objects defined here as this would
affect all instances)
   */
  function createClass() {
    var parent = null,
        properties = slice.call(arguments, 0);

    if (typeof properties[0] === 'function') {
      parent = properties.shift();
    }
    function klass() {
      this.initialize.apply(this, arguments);
    }

    klass.superclass = parent;
    klass.subclasses = [];

    if (parent) {
      Subclass.prototype = parent.prototype;
      klass.prototype = new Subclass();
      parent.subclasses.push(klass);
    }
    for (var i = 0, length = properties.length; i < length; i++) {
      addMethods(klass, properties[i], parent);
    }
    if (!klass.prototype.initialize) {
      klass.prototype.initialize = emptyFunction;
    }
    klass.prototype.constructor = klass;
    klass.prototype.callSuper = callSuper;
    return klass;
  }

  fabric.util.createClass = createClass;
})();


(function () {

  var unknown = 'unknown';

  /* EVENT HANDLING */

  function areHostMethods(object) {
    var methodNames = Array.prototype.slice.call(arguments, 1),
```

```
        t, i, len = methodNames.length;
    for (i = 0; i < len; i++) {
      t = typeof object[methodNames[i]];
      if (!(/^(?:function|object|unknown)$/).test(t)) {
        return false;
      }
    }
    return true;
  }

  /** @ignore */
  var getElement,
      setElement,
      getUniqueId = (function () {
        var uid = 0;
        return function (element) {
          return element.__uniqueID || (element.__uniqueID = 'uniqueID__' + uid+
+);
        };
      })();

  (function () {
    var elements = { };
    /** @ignore */
    getElement = function (uid) {
      return elements[uid];
    };
    /** @ignore */
    setElement = function (uid, element) {
      elements[uid] = element;
    };
  })();

  function createListener(uid, handler) {
    return {
      handler: handler,
      wrappedHandler: createWrappedHandler(uid, handler)
    };
  }

  function createWrappedHandler(uid, handler) {
    return function (e) {
      handler.call(getElement(uid), e || fabric.window.event);
    };
  }

  function createDispatcher(uid, eventName) {
    return function (e) {
      if (handlers[uid] && handlers[uid][eventName]) {
        var handlersForEvent = handlers[uid][eventName];
        for (var i = 0, len = handlersForEvent.length; i < len; i++) {
          handlersForEvent[i].call(this, e || fabric.window.event);
        }
      }
    };
  }

  var shouldUseAddListenerRemoveListener = (
        areHostMethods(fabric.document.documentElement, 'addEventListener',
'removeEventListener') &&
        areHostMethods(fabric.window, 'addEventListener',
'removeEventListener')),

      shouldUseAttachEventDetachEvent = (
```

```
        areHostMethods(fabric.document.documentElement, 'attachEvent',
'detachEvent') &&
        areHostMethods(fabric.window, 'attachEvent', 'detachEvent')),

      // IE branch
      listeners = { },

      // DOM L0 branch
      handlers = { },

      addListener, removeListener;

  if (shouldUseAddListenerRemoveListener) {
    /** @ignore */
    addListener = function (element, eventName, handler, options) {
      // since ie10 or ie9 can use addEventListener but they do not support
options, i need to check
      element && element.addEventListener(eventName, handler,
shouldUseAttachEventDetachEvent ? false : options);
    };
    /** @ignore */
    removeListener = function (element, eventName, handler, options) {
      element && element.removeEventListener(eventName, handler,
shouldUseAttachEventDetachEvent ? false : options);
    };
  }

  else if (shouldUseAttachEventDetachEvent) {
    /** @ignore */
    addListener = function (element, eventName, handler) {
      if (!element) {
        return;
      }
      var uid = getUniqueId(element);
      setElement(uid, element);
      if (!listeners[uid]) {
        listeners[uid] = { };
      }
      if (!listeners[uid][eventName]) {
        listeners[uid][eventName] = [];

      }
      var listener = createListener(uid, handler);
      listeners[uid][eventName].push(listener);
      element.attachEvent('on' + eventName, listener.wrappedHandler);
    };
    /** @ignore */
    removeListener = function (element, eventName, handler) {
      if (!element) {
        return;
      }
      var uid = getUniqueId(element), listener;
      if (listeners[uid] && listeners[uid][eventName]) {
        for (var i = 0, len = listeners[uid][eventName].length; i < len; i++) {
          listener = listeners[uid][eventName][i];
          if (listener && listener.handler === handler) {
            element.detachEvent('on' + eventName, listener.wrappedHandler);
            listeners[uid][eventName][i] = null;
          }
        }
      }
    };
  }
  else {
```

```
  /** @ignore */
  addListener = function (element, eventName, handler) {
    if (!element) {
      return;
    }
    var uid = getUniqueId(element);
    if (!handlers[uid]) {
      handlers[uid] = { };
    }
    if (!handlers[uid][eventName]) {
      handlers[uid][eventName] = [];
      var existingHandler = element['on' + eventName];
      if (existingHandler) {
        handlers[uid][eventName].push(existingHandler);
      }
      element['on' + eventName] = createDispatcher(uid, eventName);
    }
    handlers[uid][eventName].push(handler);
  };
  /** @ignore */
  removeListener = function (element, eventName, handler) {
    if (!element) {
      return;
    }
    var uid = getUniqueId(element);
    if (handlers[uid] && handlers[uid][eventName]) {
      var handlersForEvent = handlers[uid][eventName];
      for (var i = 0, len = handlersForEvent.length; i < len; i++) {
        if (handlersForEvent[i] === handler) {
          handlersForEvent.splice(i, 1);
        }
      }
    }
  };
}

/**
 * Adds an event listener to an element
 * @function
 * @memberOf fabric.util
 * @param {HTMLElement} element
 * @param {String} eventName
 * @param {Function} handler
 */
fabric.util.addListener = addListener;

/**
 * Removes an event listener from an element
 * @function
 * @memberOf fabric.util
 * @param {HTMLElement} element
 * @param {String} eventName
 * @param {Function} handler
 */
fabric.util.removeListener = removeListener;

/**
 * Cross-browser wrapper for getting event's coordinates
 * @memberOf fabric.util
 * @param {Event} event Event object
 */
function getPointer(event) {
  event || (event = fabric.window.event);
```

```
    var element = event.target ||
                  (typeof event.srcElement !== unknown ? event.srcElement :
null),

        scroll = fabric.util.getScrollLeftTop(element);

    return {
      x: pointerX(event) + scroll.left,
      y: pointerY(event) + scroll.top
    };
  }

  var pointerX = function(event) {
    // looks like in IE (<9) clientX at certain point (apparently when mouseup
fires on VML element)
    // is represented as COM object, with all the consequences, like "unknown"
type and error on [[Get]]
    // need to investigate later
        return (typeof event.clientX !== unknown ? event.clientX : 0);
      },

      pointerY = function(event) {
        return (typeof event.clientY !== unknown ? event.clientY : 0);
      };

  function _getPointer(event, pageProp, clientProp) {
    var touchProp = event.type === 'touchend' ? 'changedTouches' : 'touches';

    return (event[touchProp] && event[touchProp][0]
      ? (event[touchProp][0][pageProp] - (event[touchProp][0][pageProp] -
event[touchProp][0][clientProp]))
        || event[clientProp]
      : event[clientProp]);
  }

  if (fabric.isTouchSupported) {
    pointerX = function(event) {
      return _getPointer(event, 'pageX', 'clientX');
    };
    pointerY = function(event) {
      return _getPointer(event, 'pageY', 'clientY');
    };
  }

  fabric.util.getPointer = getPointer;

  fabric.util.object.extend(fabric.util, fabric.Observable);

})();


(function () {

  /**
   * Cross-browser wrapper for setting element's style
   * @memberOf fabric.util
   * @param {HTMLElement} element
   * @param {Object} styles
   * @return {HTMLElement} Element that was passed as a first argument
   */
  function setStyle(element, styles) {
    var elementStyle = element.style;
    if (!elementStyle) {
      return element;
```

```
    }
    if (typeof styles === 'string') {
      element.style.cssText += ';' + styles;
      return styles.indexOf('opacity') > -1
        ? setOpacity(element, styles.match(/opacity:\s*(\d?\.?\d*)/)[1])
        : element;
    }
    for (var property in styles) {
      if (property === 'opacity') {
        setOpacity(element, styles[property]);
      }
      else {
        var normalizedProperty = (property === 'float' || property ===
'cssFloat')
          ? (typeof elementStyle.styleFloat === 'undefined' ? 'cssFloat' :
'styleFloat')
          : property;
        elementStyle[normalizedProperty] = styles[property];
      }
    }
    return element;
  }

  var parseEl = fabric.document.createElement('div'),
      supportsOpacity = typeof parseEl.style.opacity === 'string',
      supportsFilters = typeof parseEl.style.filter === 'string',
      reOpacity = /alpha\s*\(\s*opacity\s*=\s*([^\)]+)\)/,

      /** @ignore */
      setOpacity = function (element) { return element; };

  if (supportsOpacity) {
    /** @ignore */
    setOpacity = function(element, value) {
      element.style.opacity = value;
      return element;
    };
  }
  else if (supportsFilters) {
    /** @ignore */
    setOpacity = function(element, value) {
      var es = element.style;
      if (element.currentStyle && !element.currentStyle.hasLayout) {
        es.zoom = 1;
      }
      if (reOpacity.test(es.filter)) {
        value = value >= 0.9999 ? '' : ('alpha(opacity=' + (value * 100) + ')');
        es.filter = es.filter.replace(reOpacity, value);
      }
      else {
        es.filter += ' alpha(opacity=' + (value * 100) + ')';
      }
      return element;
    };
  }

  fabric.util.setStyle = setStyle;

})();


(function() {

  var _slice = Array.prototype.slice;
```

```
  /**
   * Takes id and returns an element with that id (if one exists in a document)
   * @memberOf fabric.util
   * @param {String|HTMLElement} id
   * @return {HTMLElement|null}
   */
  function getById(id) {
    return typeof id === 'string' ? fabric.document.getElementById(id) : id;
  }

  var sliceCanConvertNodelists,
      /**
       * Converts an array-like object (e.g. arguments or NodeList) to an array
       * @memberOf fabric.util
       * @param {Object} arrayLike
       * @return {Array}
       */
      toArray = function(arrayLike) {
        return _slice.call(arrayLike, 0);
      };

  try {
    sliceCanConvertNodelists = toArray(fabric.document.childNodes) instanceof
Array;
  }
  catch (err) { }

  if (!sliceCanConvertNodelists) {
    toArray = function(arrayLike) {
      var arr = new Array(arrayLike.length), i = arrayLike.length;
      while (i--) {
        arr[i] = arrayLike[i];
      }
      return arr;
    };
  }

  /**
   * Creates specified element with specified attributes
   * @memberOf fabric.util
   * @param {String} tagName Type of an element to create
   * @param {Object} [attributes] Attributes to set on an element
   * @return {HTMLElement} Newly created element
   */
  function makeElement(tagName, attributes) {
    var el = fabric.document.createElement(tagName);
    for (var prop in attributes) {
      if (prop === 'class') {
        el.className = attributes[prop];
      }
      else if (prop === 'for') {
        el.htmlFor = attributes[prop];
      }
      else {
        el.setAttribute(prop, attributes[prop]);
      }
    }
    return el;
  }

  /**
   * Adds class to an element
   * @memberOf fabric.util
```

```
   * @param {HTMLElement} element Element to add class to
   * @param {String} className Class to add to an element
   */
  function addClass(element, className) {
    if (element && (' ' + element.className + ' ').indexOf(' ' + className + '
') === -1) {
      element.className += (element.className ? ' ' : '') + className;
    }
  }

  /**
   * Wraps element with another element
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to wrap
   * @param {HTMLElement|String} wrapper Element to wrap with
   * @param {Object} [attributes] Attributes to set on a wrapper
   * @return {HTMLElement} wrapper
   */
  function wrapElement(element, wrapper, attributes) {
    if (typeof wrapper === 'string') {
      wrapper = makeElement(wrapper, attributes);
    }
    if (element.parentNode) {
      element.parentNode.replaceChild(wrapper, element);
    }
    wrapper.appendChild(element);
    return wrapper;
  }

  /**
   * Returns element scroll offsets
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to operate on
   * @return {Object} Object with left/top values
   */
  function getScrollLeftTop(element) {

    var left = 0,
        top = 0,
        docElement = fabric.document.documentElement,
        body = fabric.document.body || {
          scrollLeft: 0, scrollTop: 0
        };

    // While loop checks (and then sets element to) .parentNode OR .host
    //  to account for ShadowDOM. We still want to traverse up out of ShadowDOM,
    //  but the .parentNode of a root ShadowDOM node will always be null,
instead
    //  it should be accessed through .host. See
http://stackoverflow.com/a/24765528/4383938
    while (element && (element.parentNode || element.host)) {

      // Set element to element parent, or 'host' in case of ShadowDOM
      element = element.parentNode || element.host;

      if (element === fabric.document) {
        left = body.scrollLeft || docElement.scrollLeft || 0;
        top = body.scrollTop ||  docElement.scrollTop || 0;
      }
      else {
        left += element.scrollLeft || 0;
        top += element.scrollTop || 0;
      }
```

```
      if (element.nodeType === 1 &&
          fabric.util.getElementStyle(element, 'position') === 'fixed') {
        break;
      }
    }

    return { left: left, top: top };
  }

  /**
   * Returns offset for a given element
   * @function
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to get offset for
   * @return {Object} Object with "left" and "top" properties
   */
  function getElementOffset(element) {
    var docElem,
        doc = element && element.ownerDocument,
        box = { left: 0, top: 0 },
        offset = { left: 0, top: 0 },
        scrollLeftTop,
        offsetAttributes = {
          borderLeftWidth: 'left',
          borderTopWidth:  'top',
          paddingLeft:     'left',
          paddingTop:      'top'
        };

    if (!doc) {
      return offset;
    }

    for (var attr in offsetAttributes) {
      offset[offsetAttributes[attr]] += parseInt(getElementStyle(element, attr),
10) || 0;
    }

    docElem = doc.documentElement;
    if ( typeof element.getBoundingClientRect !== 'undefined' ) {
      box = element.getBoundingClientRect();
    }

    scrollLeftTop = getScrollLeftTop(element);

    return {
      left: box.left + scrollLeftTop.left - (docElem.clientLeft || 0) +
offset.left,
      top: box.top + scrollLeftTop.top - (docElem.clientTop || 0)  + offset.top
    };
  }

  /**
   * Returns style attribute value of a given element
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to get style attribute for
   * @param {String} attr Style attribute to get for element
   * @return {String} Style attribute value of the given element.
   */
  var getElementStyle;
  if (fabric.document.defaultView &&
fabric.document.defaultView.getComputedStyle) {
    getElementStyle = function(element, attr) {
      var style = fabric.document.defaultView.getComputedStyle(element, null);
```

```
    return style ? style[attr] : undefined;
  };
}
else {
  getElementStyle = function(element, attr) {
    var value = element.style[attr];
    if (!value && element.currentStyle) {
      value = element.currentStyle[attr];
    }
    return value;
  };
}

(function () {
  var style = fabric.document.documentElement.style,
      selectProp = 'userSelect' in style
        ? 'userSelect'
        : 'MozUserSelect' in style
          ? 'MozUserSelect'
          : 'WebkitUserSelect' in style
            ? 'WebkitUserSelect'
            : 'KhtmlUserSelect' in style
              ? 'KhtmlUserSelect'
              : '';

  /**
   * Makes element unselectable
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to make unselectable
   * @return {HTMLElement} Element that was passed in
   */
  function makeElementUnselectable(element) {
    if (typeof element.onselectstart !== 'undefined') {
      element.onselectstart = fabric.util.falseFunction;
    }
    if (selectProp) {
      element.style[selectProp] = 'none';
    }
    else if (typeof element.unselectable === 'string') {
      element.unselectable = 'on';
    }
    return element;
  }

  /**
   * Makes element selectable
   * @memberOf fabric.util
   * @param {HTMLElement} element Element to make selectable
   * @return {HTMLElement} Element that was passed in
   */
  function makeElementSelectable(element) {
    if (typeof element.onselectstart !== 'undefined') {
      element.onselectstart = null;
    }
    if (selectProp) {
      element.style[selectProp] = '';
    }
    else if (typeof element.unselectable === 'string') {
      element.unselectable = '';
    }
    return element;
  }

  fabric.util.makeElementUnselectable = makeElementUnselectable;
```

```
    fabric.util.makeElementSelectable = makeElementSelectable;
  })();

  (function() {

    /**
     * Inserts a script element with a given url into a document; invokes
callback, when that script is finished loading
     * @memberOf fabric.util
     * @param {String} url URL of a script to load
     * @param {Function} callback Callback to execute when script is finished
loading
     */
    function getScript(url, callback) {
      var headEl = fabric.document.getElementsByTagName('head')[0],
          scriptEl = fabric.document.createElement('script'),
          loading = true;

      /** @ignore */
      scriptEl.onload = /** @ignore */ scriptEl.onreadystatechange = function(e)
{
        if (loading) {
          if (typeof this.readyState === 'string' &&
              this.readyState !== 'loaded' &&
              this.readyState !== 'complete') {
            return;
          }
          loading = false;
          callback(e || fabric.window.event);
          scriptEl = scriptEl.onload = scriptEl.onreadystatechange = null;
        }
      };
      scriptEl.src = url;
      headEl.appendChild(scriptEl);
      // causes issue in Opera
      // headEl.removeChild(scriptEl);
    }

    fabric.util.getScript = getScript;
  })();

  fabric.util.getById = getById;
  fabric.util.toArray = toArray;
  fabric.util.makeElement = makeElement;
  fabric.util.addClass = addClass;
  fabric.util.wrapElement = wrapElement;
  fabric.util.getScrollLeftTop = getScrollLeftTop;
  fabric.util.getElementOffset = getElementOffset;
  fabric.util.getElementStyle = getElementStyle;

})();


(function() {

  function addParamToUrl(url, param) {
    return url + (/\?/.test(url) ? '&' : '?') + param;
  }

  var makeXHR = (function() {
    var factories = [
      function() { return new ActiveXObject('Microsoft.XMLHTTP'); },
      function() { return new ActiveXObject('Msxml2.XMLHTTP'); },
      function() { return new ActiveXObject('Msxml2.XMLHTTP.3.0'); },
```

```
      function() { return new XMLHttpRequest(); }
    ];
    for (var i = factories.length; i--; ) {
      try {
        var req = factories[i]();
        if (req) {
          return factories[i];
        }
      }
      catch (err) { }
    }
  })();

  function emptyFn() { }

  /**
   * Cross-browser abstraction for sending XMLHttpRequest
   * @memberOf fabric.util
   * @param {String} url URL to send XMLHttpRequest to
   * @param {Object} [options] Options object
   * @param {String} [options.method="GET"]
   * @param {String} [options.parameters] parameters to append to url in GET or
in body
   * @param {String} [options.body] body to send with POST or PUT request
   * @param {Function} options.onComplete Callback to invoke when request is
completed
   * @return {XMLHttpRequest} request
   */
  function request(url, options) {

    options || (options = { });

    var method = options.method ? options.method.toUpperCase() : 'GET',
        onComplete = options.onComplete || function() { },
        xhr = makeXHR(),
        body = options.body || options.parameters;

    /** @ignore */
    xhr.onreadystatechange = function() {
      if (xhr.readyState === 4) {
        onComplete(xhr);
        xhr.onreadystatechange = emptyFn;
      }
    };

    if (method === 'GET') {
      body = null;
      if (typeof options.parameters === 'string') {
        url = addParamToUrl(url, options.parameters);
      }
    }

    xhr.open(method, url, true);

    if (method === 'POST' || method === 'PUT') {
      xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    }

    xhr.send(body);
    return xhr;
  }

  fabric.util.request = request;
})();
```

```
/**
 * Wrapper around `console.log` (when available)
 * @param {*} [values] Values to log
 */
fabric.log = function() { };

/**
 * Wrapper around `console.warn` (when available)
 * @param {*} [values] Values to log as a warning
 */
fabric.warn = function() { };

/* eslint-disable */
if (typeof console !== 'undefined') {

  ['log', 'warn'].forEach(function(methodName) {

    if (typeof console[methodName] !== 'undefined' &&
        typeof console[methodName].apply === 'function') {

      fabric[methodName] = function() {
        return console[methodName].apply(console, arguments);
      };
    }
  });
}
/* eslint-enable */


(function() {

  function noop() {
    return false;
  }

  /**
   * Changes value from one to another within certain period of time, invoking
callbacks as value is being changed.
   * @memberOf fabric.util
   * @param {Object} [options] Animation options
   * @param {Function} [options.onChange] Callback; invoked on every value
change
   * @param {Function} [options.onComplete] Callback; invoked when value change
is completed
   * @param {Number} [options.startValue=0] Starting value
   * @param {Number} [options.endValue=100] Ending value
   * @param {Number} [options.byValue=100] Value to modify the property by
   * @param {Function} [options.easing] Easing function
   * @param {Number} [options.duration=500] Duration of change (in ms)
   */
  function animate(options) {

    requestAnimFrame(function(timestamp) {
      options || (options = { });

      var start = timestamp || +new Date(),
          duration = options.duration || 500,
          finish = start + duration, time,
          onChange = options.onChange || noop,
          abort = options.abort || noop,
          onComplete = options.onComplete || noop,
          easing = options.easing || function(t, b, c, d) {return -c *
```

```
Math.cos(t / d * (Math.PI / 2)) + c + b;},
          startValue = 'startValue' in options ? options.startValue : 0,
          endValue = 'endValue' in options ? options.endValue : 100,
          byValue = options.byValue || endValue - startValue;

      options.onStart && options.onStart();

      (function tick(ticktime) {
        if (abort()) {
          onComplete(endValue, 1, 1);
          return;
        }
        time = ticktime || +new Date();
        var currentTime = time > finish ? duration : (time - start),
            timePerc = currentTime / duration,
            current = easing(currentTime, startValue, byValue, duration),
            valuePerc = Math.abs((current - startValue) / byValue);
        onChange(current, valuePerc, timePerc);
        if (time > finish) {
          options.onComplete && options.onComplete();
          return;
        }
        requestAnimFrame(tick);
      })(start);
    });

  }

  var _requestAnimFrame = fabric.window.requestAnimationFrame       ||
                          fabric.window.webkitRequestAnimationFrame ||
                          fabric.window.mozRequestAnimationFrame    ||
                          fabric.window.oRequestAnimationFrame      ||
                          fabric.window.msRequestAnimationFrame     ||
                          function(callback) {
                            fabric.window.setTimeout(callback, 1000 / 60);
                          };

  /**
   * requestAnimationFrame polyfill based on
http://paulirish.com/2011/requestanimationframe-for-smart-animating/
   * In order to get a precise start time, `requestAnimFrame` should be called
as an entry into the method
   * @memberOf fabric.util
   * @param {Function} callback Callback to invoke
   * @param {DOMElement} element optional Element to associate with animation
   */
  function requestAnimFrame() {
    return _requestAnimFrame.apply(fabric.window, arguments);
  }

  fabric.util.animate = animate;
  fabric.util.requestAnimFrame = requestAnimFrame;

})();


(function() {
  // Calculate an in-between color. Returns a "rgba()" string.
  // Credit: Edwin Martin <edwin@bitstorm.org>
  //         http://www.bitstorm.org/jquery/color-animation/jquery.animate-
colors.js
  function calculateColor(begin, end, pos) {
    var color = 'rgba('
        + parseInt((begin[0] + pos * (end[0] - begin[0])), 10) + ','
```

```
        + parseInt((begin[1] + pos * (end[1] - begin[1])), 10) + ','
        + parseInt((begin[2] + pos * (end[2] - begin[2])), 10);

    color += ',' + (begin && end ? parseFloat(begin[3] + pos * (end[3] -
begin[3])) : 1);
    color += ')';
    return color;
  }

  /**
   * Changes the color from one to another within certain period of time,
invoking callbacks as value is being changed.
   * @memberOf fabric.util
   * @param {String} fromColor The starting color in hex or rgb(a) format.
   * @param {String} toColor The starting color in hex or rgb(a) format.
   * @param {Number} [duration] Duration of change (in ms).
   * @param {Object} [options] Animation options
   * @param {Function} [options.onChange] Callback; invoked on every value
change
   * @param {Function} [options.onComplete] Callback; invoked when value change
is completed
   * @param {Function} [options.colorEasing] Easing function. Note that this
function only take two arguments (currentTime, duration). Thus the regular
animation easing functions cannot be used.
   */
  function animateColor(fromColor, toColor, duration, options) {
    var startColor = new fabric.Color(fromColor).getSource(),
        endColor = new fabric.Color(toColor).getSource();

    options = options || {};

    fabric.util.animate(fabric.util.object.extend(options, {
      duration: duration || 500,
      startValue: startColor,
      endValue: endColor,
      byValue: endColor,
      easing: function (currentTime, startValue, byValue, duration) {
        var posValue = options.colorEasing
              ? options.colorEasing(currentTime, duration)
              : 1 - Math.cos(currentTime / duration * (Math.PI / 2));
        return calculateColor(startValue, byValue, posValue);
      }
    }));
  }

  fabric.util.animateColor = animateColor;

})();


(function() {

  function normalize(a, c, p, s) {
    if (a < Math.abs(c)) {
      a = c;
      s = p / 4;
    }
    else {
      //handle the 0/0 case:
      if (c === 0 && a === 0) {
        s = p / (2 * Math.PI) * Math.asin(1);
      }
      else {
        s = p / (2 * Math.PI) * Math.asin(c / a);
```

```
    }
  }
  return { a: a, c: c, p: p, s: s };
}

function elastic(opts, t, d) {
  return opts.a *
    Math.pow(2, 10 * (t -= 1)) *
    Math.sin( (t * d - opts.s) * (2 * Math.PI) / opts.p );
}

/**
 * Cubic easing out
 * @memberOf fabric.util.ease
 */
function easeOutCubic(t, b, c, d) {
  return c * ((t = t / d - 1) * t * t + 1) + b;
}

/**
 * Cubic easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutCubic(t, b, c, d) {
  t /= d / 2;
  if (t < 1) {
    return c / 2 * t * t * t + b;
  }
  return c / 2 * ((t -= 2) * t * t + 2) + b;
}

/**
 * Quartic easing in
 * @memberOf fabric.util.ease
 */
function easeInQuart(t, b, c, d) {
  return c * (t /= d) * t * t * t + b;
}

/**
 * Quartic easing out
 * @memberOf fabric.util.ease
 */
function easeOutQuart(t, b, c, d) {
  return -c * ((t = t / d - 1) * t * t * t - 1) + b;
}

/**
 * Quartic easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutQuart(t, b, c, d) {
  t /= d / 2;
  if (t < 1) {
    return c / 2 * t * t * t * t + b;
  }
  return -c / 2 * ((t -= 2) * t * t * t - 2) + b;
}

/**
 * Quintic easing in
 * @memberOf fabric.util.ease
 */
function easeInQuint(t, b, c, d) {
```

```
  return c * (t /= d) * t * t * t * t + b;
}

/**
 * Quintic easing out
 * @memberOf fabric.util.ease
 */
function easeOutQuint(t, b, c, d) {
  return c * ((t = t / d - 1) * t * t * t * t + 1) + b;
}

/**
 * Quintic easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutQuint(t, b, c, d) {
  t /= d / 2;
  if (t < 1) {
    return c / 2 * t * t * t * t * t + b;
  }
  return c / 2 * ((t -= 2) * t * t * t * t + 2) + b;
}

/**
 * Sinusoidal easing in
 * @memberOf fabric.util.ease
 */
function easeInSine(t, b, c, d) {
  return -c * Math.cos(t / d * (Math.PI / 2)) + c + b;
}

/**
 * Sinusoidal easing out
 * @memberOf fabric.util.ease
 */
function easeOutSine(t, b, c, d) {
  return c * Math.sin(t / d * (Math.PI / 2)) + b;
}

/**
 * Sinusoidal easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutSine(t, b, c, d) {
  return -c / 2 * (Math.cos(Math.PI * t / d) - 1) + b;
}

/**
 * Exponential easing in
 * @memberOf fabric.util.ease
 */
function easeInExpo(t, b, c, d) {
  return (t === 0) ? b : c * Math.pow(2, 10 * (t / d - 1)) + b;
}

/**
 * Exponential easing out
 * @memberOf fabric.util.ease
 */
function easeOutExpo(t, b, c, d) {
  return (t === d) ? b + c : c * (-Math.pow(2, -10 * t / d) + 1) + b;
}

/**
```

```
 * Exponential easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutExpo(t, b, c, d) {
  if (t === 0) {
    return b;
  }
  if (t === d) {
    return b + c;
  }
  t /= d / 2;
  if (t < 1) {
    return c / 2 * Math.pow(2, 10 * (t - 1)) + b;
  }
  return c / 2 * (-Math.pow(2, -10 * --t) + 2) + b;
}

/**
 * Circular easing in
 * @memberOf fabric.util.ease
 */
function easeInCirc(t, b, c, d) {
  return -c * (Math.sqrt(1 - (t /= d) * t) - 1) + b;
}

/**
 * Circular easing out
 * @memberOf fabric.util.ease
 */
function easeOutCirc(t, b, c, d) {
  return c * Math.sqrt(1 - (t = t / d - 1) * t) + b;
}

/**
 * Circular easing in and out
 * @memberOf fabric.util.ease
 */
function easeInOutCirc(t, b, c, d) {
  t /= d / 2;
  if (t < 1) {
    return -c / 2 * (Math.sqrt(1 - t * t) - 1) + b;
  }
  return c / 2 * (Math.sqrt(1 - (t -= 2) * t) + 1) + b;
}

/**
 * Elastic easing in
 * @memberOf fabric.util.ease
 */
function easeInElastic(t, b, c, d) {
  var s = 1.70158, p = 0, a = c;
  if (t === 0) {
    return b;
  }
  t /= d;
  if (t === 1) {
    return b + c;
  }
  if (!p) {
    p = d * 0.3;
  }
  var opts = normalize(a, c, p, s);
  return -elastic(opts, t, d) + b;
}
```

```
  /**
   * Elastic easing out
   * @memberOf fabric.util.ease
   */
  function easeOutElastic(t, b, c, d) {
    var s = 1.70158, p = 0, a = c;
    if (t === 0) {
      return b;
    }
    t /= d;
    if (t === 1) {
      return b + c;
    }
    if (!p) {
      p = d * 0.3;
    }
    var opts = normalize(a, c, p, s);
    return opts.a * Math.pow(2, -10 * t) * Math.sin((t * d - opts.s) * (2 *
Math.PI) / opts.p ) + opts.c + b;
  }

  /**
   * Elastic easing in and out
   * @memberOf fabric.util.ease
   */
  function easeInOutElastic(t, b, c, d) {
    var s = 1.70158, p = 0, a = c;
    if (t === 0) {
      return b;
    }
    t /= d / 2;
    if (t === 2) {
      return b + c;
    }
    if (!p) {
      p = d * (0.3 * 1.5);
    }
    var opts = normalize(a, c, p, s);
    if (t < 1) {
      return -0.5 * elastic(opts, t, d) + b;
    }
    return opts.a * Math.pow(2, -10 * (t -= 1)) *
      Math.sin((t * d - opts.s) * (2 * Math.PI) / opts.p ) * 0.5 + opts.c + b;
  }

  /**
   * Backwards easing in
   * @memberOf fabric.util.ease
   */
  function easeInBack(t, b, c, d, s) {
    if (s === undefined) {
      s = 1.70158;
    }
    return c * (t /= d) * t * ((s + 1) * t - s) + b;
  }

  /**
   * Backwards easing out
   * @memberOf fabric.util.ease
   */
  function easeOutBack(t, b, c, d, s) {
    if (s === undefined) {
      s = 1.70158;
```

```
    }
    return c * ((t = t / d - 1) * t * ((s + 1) * t + s) + 1) + b;
  }

  /**
   * Backwards easing in and out
   * @memberOf fabric.util.ease
   */
  function easeInOutBack(t, b, c, d, s) {
    if (s === undefined) {
      s = 1.70158;
    }
    t /= d / 2;
    if (t < 1) {
      return c / 2 * (t * t * (((s *= (1.525)) + 1) * t - s)) + b;
    }
    return c / 2 * ((t -= 2) * t * (((s *= (1.525)) + 1) * t + s) + 2) + b;
  }

  /**
   * Bouncing easing in
   * @memberOf fabric.util.ease
   */
  function easeInBounce(t, b, c, d) {
    return c - easeOutBounce (d - t, 0, c, d) + b;
  }

  /**
   * Bouncing easing out
   * @memberOf fabric.util.ease
   */
  function easeOutBounce(t, b, c, d) {
    if ((t /= d) < (1 / 2.75)) {
      return c * (7.5625 * t * t) + b;
    }
    else if (t < (2 / 2.75)) {
      return c * (7.5625 * (t -= (1.5 / 2.75)) * t + 0.75) + b;
    }
    else if (t < (2.5 / 2.75)) {
      return c * (7.5625 * (t -= (2.25 / 2.75)) * t + 0.9375) + b;
    }
    else {
      return c * (7.5625 * (t -= (2.625 / 2.75)) * t + 0.984375) + b;
    }
  }

  /**
   * Bouncing easing in and out
   * @memberOf fabric.util.ease
   */
  function easeInOutBounce(t, b, c, d) {
    if (t < d / 2) {
      return easeInBounce (t * 2, 0, c, d) * 0.5 + b;
    }
    return easeOutBounce(t * 2 - d, 0, c, d) * 0.5 + c * 0.5 + b;
  }

  /**
   * Easing functions
   * See <a href="http://gizma.com/easing/">Easing Equations by Robert
Penner</a>
   * @namespace fabric.util.ease
   */
  fabric.util.ease = {
```

```
  /**
   * Quadratic easing in
   * @memberOf fabric.util.ease
   */
  easeInQuad: function(t, b, c, d) {
    return c * (t /= d) * t + b;
  },

  /**
   * Quadratic easing out
   * @memberOf fabric.util.ease
   */
  easeOutQuad: function(t, b, c, d) {
    return -c * (t /= d) * (t - 2) + b;
  },

  /**
   * Quadratic easing in and out
   * @memberOf fabric.util.ease
   */
  easeInOutQuad: function(t, b, c, d) {
    t /= (d / 2);
    if (t < 1) {
      return c / 2 * t * t + b;
    }
    return -c / 2 * ((--t) * (t - 2) - 1) + b;
  },

  /**
   * Cubic easing in
   * @memberOf fabric.util.ease
   */
  easeInCubic: function(t, b, c, d) {
    return c * (t /= d) * t * t + b;
  },

  easeOutCubic: easeOutCubic,
  easeInOutCubic: easeInOutCubic,
  easeInQuart: easeInQuart,
  easeOutQuart: easeOutQuart,
  easeInOutQuart: easeInOutQuart,
  easeInQuint: easeInQuint,
  easeOutQuint: easeOutQuint,
  easeInOutQuint: easeInOutQuint,
  easeInSine: easeInSine,
  easeOutSine: easeOutSine,
  easeInOutSine: easeInOutSine,
  easeInExpo: easeInExpo,
  easeOutExpo: easeOutExpo,
  easeInOutExpo: easeInOutExpo,
  easeInCirc: easeInCirc,
  easeOutCirc: easeOutCirc,
  easeInOutCirc: easeInOutCirc,
  easeInElastic: easeInElastic,
  easeOutElastic: easeOutElastic,
  easeInOutElastic: easeInOutElastic,
  easeInBack: easeInBack,
  easeOutBack: easeOutBack,
  easeInOutBack: easeInOutBack,
  easeInBounce: easeInBounce,
  easeOutBounce: easeOutBounce,
  easeInOutBounce: easeInOutBounce
};
```

```
})();


(function(global) {

  'use strict';

  /**
   * @name fabric
   * @namespace
   */

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      clone = fabric.util.object.clone,
      toFixed = fabric.util.toFixed,
      parseUnit = fabric.util.parseUnit,
      multiplyTransformMatrices = fabric.util.multiplyTransformMatrices,

      reAllowedSVGTagNames = /^(path|circle|polygon|polyline|ellipse|rect|line|
image|text)$/i,
      reViewBoxTagNames = /^(symbol|image|marker|pattern|view|svg)$/i,
      reNotAllowedAncestors = /^(?:pattern|defs|symbol|metadata|clipPath|mask)
$/i,
      reAllowedParents = /^(symbol|g|a|svg)$/i,

      attributesMap = {
        cx:                   'left',
        x:                    'left',
        r:                    'radius',
        cy:                   'top',
        y:                    'top',
        display:              'visible',
        visibility:           'visible',
        transform:            'transformMatrix',
        'fill-opacity':       'fillOpacity',
        'fill-rule':          'fillRule',
        'font-family':        'fontFamily',
        'font-size':          'fontSize',
        'font-style':         'fontStyle',
        'font-weight':        'fontWeight',
        'stroke-dasharray':   'strokeDashArray',
        'stroke-linecap':     'strokeLineCap',
        'stroke-linejoin':    'strokeLineJoin',
        'stroke-miterlimit':  'strokeMiterLimit',
        'stroke-opacity':     'strokeOpacity',
        'stroke-width':       'strokeWidth',
        'text-decoration':    'textDecoration',
        'text-anchor':        'originX',
        opacity:              'opacity'
      },

      colorAttributes = {
        stroke: 'strokeOpacity',
        fill:   'fillOpacity'
      };

  fabric.cssRules = { };
  fabric.gradientDefs = { };

  function normalizeAttr(attr) {
    // transform attribute names
    if (attr in attributesMap) {
```

```
      return attributesMap[attr];
    }
    return attr;
  }

  function normalizeValue(attr, value, parentAttributes, fontSize) {
    var isArray = Object.prototype.toString.call(value) === '[object Array]',
        parsed;

    if ((attr === 'fill' || attr === 'stroke') && value === 'none') {
      value = '';
    }
    else if (attr === 'strokeDashArray') {
      if (value === 'none') {
        value = null;
      }
      else {
        value = value.replace(/,/g, ' ').split(/\s+/).map(function(n) {
          return parseFloat(n);
        });
      }
    }
    else if (attr === 'transformMatrix') {
      if (parentAttributes && parentAttributes.transformMatrix) {
        value = multiplyTransformMatrices(
          parentAttributes.transformMatrix,
fabric.parseTransformAttribute(value));
      }
      else {
        value = fabric.parseTransformAttribute(value);
      }
    }
    else if (attr === 'visible') {
      value = (value === 'none' || value === 'hidden') ? false : true;
      // display=none on parent element always takes precedence over child
element
      if (parentAttributes && parentAttributes.visible === false) {
        value = false;
      }
    }
    else if (attr === 'opacity') {
      value = parseFloat(value);
      if (parentAttributes && typeof parentAttributes.opacity !== 'undefined') {
        value *= parentAttributes.opacity;
      }
    }
    else if (attr === 'originX' /* text-anchor */) {
      value = value === 'start' ? 'left' : value === 'end' ? 'right' : 'center';
    }
    else {
      parsed = isArray ? value.map(parseUnit) : parseUnit(value, fontSize);
    }

    return (!isArray && isNaN(parsed) ? value : parsed);
  }

  /**
   * @private
   * @param {Object} attributes Array of attributes to parse
   */
  function _setStrokeFillOpacity(attributes) {
    for (var attr in colorAttributes) {

      if (typeof attributes[colorAttributes[attr]] === 'undefined' ||
```

```
attributes[attr] === '') {
        continue;
      }

      if (typeof attributes[attr] === 'undefined') {
        if (!fabric.Object.prototype[attr]) {
          continue;
        }
        attributes[attr] = fabric.Object.prototype[attr];
      }

      if (attributes[attr].indexOf('url(') === 0) {
        continue;
      }

      var color = new fabric.Color(attributes[attr]);
      attributes[attr] = color.setAlpha(toFixed(color.getAlpha() *
attributes[colorAttributes[attr]], 2)).toRgba();
    }
    return attributes;
  }

  /**
   * @private
   */
  function _getMultipleNodes(doc, nodeNames) {
    var nodeName, nodeArray = [], nodeList;
    for (var i = 0; i < nodeNames.length; i++) {
      nodeName = nodeNames[i];
      nodeList = doc.getElementsByTagName(nodeName);
      nodeArray = nodeArray.concat(Array.prototype.slice.call(nodeList));
    }
    return nodeArray;
  }

  /**
   * Parses "transform" attribute, returning an array of values
   * @static
   * @function
   * @memberOf fabric
   * @param {String} attributeValue String containing attribute value
   * @return {Array} Array of 6 elements representing transformation matrix
   */
  fabric.parseTransformAttribute = (function() {
    function rotateMatrix(matrix, args) {
      var cos = Math.cos(args[0]), sin = Math.sin(args[0]),
          x = 0, y = 0;
      if (args.length === 3) {
        x = args[1];
        y = args[2];
      }

      matrix[0] = cos;
      matrix[1] = sin;
      matrix[2] = -sin;
      matrix[3] = cos;
      matrix[4] = x - (cos * x - sin * y);
      matrix[5] = y - (sin * x + cos * y);
    }

    function scaleMatrix(matrix, args) {
      var multiplierX = args[0],
          multiplierY = (args.length === 2) ? args[1] : args[0];
```

```
      matrix[0] = multiplierX;
      matrix[3] = multiplierY;
    }

    function skewMatrix(matrix, args, pos) {
      matrix[pos] = Math.tan(fabric.util.degreesToRadians(args[0]));
    }

    function translateMatrix(matrix, args) {
      matrix[4] = args[0];
      if (args.length === 2) {
        matrix[5] = args[1];
      }
    }

    // identity matrix
    var iMatrix = [
          1, // a
          0, // b
          0, // c
          1, // d
          0, // e
          0  // f
        ],

        // == begin transform regexp
        number = fabric.reNum,

        commaWsp = '(?:\\s+,?\\s*|,\\s*)',

        skewX = '(?:(skewX)\\s*\\(\\s*(' + number + ')\\s*\\))',

        skewY = '(?:(skewY)\\s*\\(\\s*(' + number + ')\\s*\\))',

        rotate = '(?:(rotate)\\s*\\(\\s*(' + number + ')(?:' +
                    commaWsp + '(' + number + ')' +
                    commaWsp + '(' + number + '))?\\s*\\))',

        scale = '(?:(scale)\\s*\\(\\s*(' + number + ')(?:' +
                    commaWsp + '(' + number + '))?\\s*\\))',

        translate = '(?:(translate)\\s*\\(\\s*(' + number + ')(?:' +
                    commaWsp + '(' + number + '))?\\s*\\))',

        matrix = '(?:(matrix)\\s*\\(\\s*' +
                  '(' + number + ')' + commaWsp +
                  '(' + number + ')' + commaWsp +
                  '(' + number + ')' + commaWsp +
                  '(' + number + ')' + commaWsp +
                  '(' + number + ')' + commaWsp +
                  '(' + number + ')' +
                  '\\s*\\))',

        transform = '(?:' +
                    matrix + '|' +
                    translate + '|' +
                    scale + '|' +
                    rotate + '|' +
                    skewX + '|' +
                    skewY +
                    ')',

        transforms = '(?:' + transform + '(?:' + commaWsp + '*' + transform +
')*' + ')',
```

```
        transformList = '^\\s*(?:' + transforms + '?)\\s*$',

        // http://www.w3.org/TR/SVG/coords.html#TransformAttribute
        reTransformList = new RegExp(transformList),
        // == end transform regexp

        reTransform = new RegExp(transform, 'g');

    return function(attributeValue) {

      // start with identity matrix
      var matrix = iMatrix.concat(),
          matrices = [];

      // return if no argument was given or
      // an argument does not match transform attribute regexp
      if (!attributeValue || (attributeValue && !
reTransformList.test(attributeValue))) {
        return matrix;
      }

      attributeValue.replace(reTransform, function(match) {

        var m = new RegExp(transform).exec(match).filter(function (match) {
              // match !== '' && match != null
              return (!!match);
            }),
            operation = m[1],
            args = m.slice(2).map(parseFloat);

        switch (operation) {
          case 'translate':
            translateMatrix(matrix, args);
            break;
          case 'rotate':
            args[0] = fabric.util.degreesToRadians(args[0]);
            rotateMatrix(matrix, args);
            break;
          case 'scale':
            scaleMatrix(matrix, args);
            break;
          case 'skewX':
            skewMatrix(matrix, args, 2);
            break;
          case 'skewY':
            skewMatrix(matrix, args, 1);
            break;
          case 'matrix':
            matrix = args;
            break;
        }

        // snapshot current matrix into matrices array
        matrices.push(matrix.concat());
        // reset
        matrix = iMatrix.concat();
      });

      var combinedMatrix = matrices[0];
      while (matrices.length > 1) {
        matrices.shift();
        combinedMatrix = fabric.util.multiplyTransformMatrices(combinedMatrix,
matrices[0]);
```

```
    }
    return combinedMatrix;
  };
})();

/**
 * @private
 */
function parseStyleString(style, oStyle) {
  var attr, value;
  style.replace(/;\s*$/, '').split(';').forEach(function (chunk) {
    var pair = chunk.split(':');

    attr = pair[0].trim().toLowerCase();
    value =  pair[1].trim();

    oStyle[attr] = value;
  });
}

/**
 * @private
 */
function parseStyleObject(style, oStyle) {
  var attr, value;
  for (var prop in style) {
    if (typeof style[prop] === 'undefined') {
      continue;
    }

    attr = prop.toLowerCase();
    value = style[prop];

    oStyle[attr] = value;
  }
}

/**
 * @private
 */
function getGlobalStylesForElement(element, svgUid) {
  var styles = { };
  for (var rule in fabric.cssRules[svgUid]) {
    if (elementMatchesRule(element, rule.split(' '))) {
      for (var property in fabric.cssRules[svgUid][rule]) {
        styles[property] = fabric.cssRules[svgUid][rule][property];
      }
    }
  }
  return styles;
}

/**
 * @private
 */
function elementMatchesRule(element, selectors) {
  var firstMatching, parentMatching = true;
  //start from rightmost selector.
  firstMatching = selectorMatches(element, selectors.pop());
  if (firstMatching && selectors.length) {
    parentMatching = doesSomeParentMatch(element, selectors);
  }
  return firstMatching && parentMatching && (selectors.length === 0);
}
```

```
  function doesSomeParentMatch(element, selectors) {
    var selector, parentMatching = true;
    while (element.parentNode && element.parentNode.nodeType === 1 &&
selectors.length) {
      if (parentMatching) {
        selector = selectors.pop();
      }
      element = element.parentNode;
      parentMatching = selectorMatches(element, selector);
    }
    return selectors.length === 0;
  }

  /**
   * @private
   */
  function selectorMatches(element, selector) {
    var nodeName = element.nodeName,
        classNames = element.getAttribute('class'),
        id = element.getAttribute('id'), matcher;
    // i check if a selector matches slicing away part from it.
    // if i get empty string i should match
    matcher = new RegExp('^' + nodeName, 'i');
    selector = selector.replace(matcher, '');
    if (id && selector.length) {
      matcher = new RegExp('#' + id + '(?![a-zA-Z\\-]+)', 'i');
      selector = selector.replace(matcher, '');
    }
    if (classNames && selector.length) {
      classNames = classNames.split(' ');
      for (var i = classNames.length; i--;) {
        matcher = new RegExp('\\.' + classNames[i] + '(?![a-zA-Z\\-]+)', 'i');
        selector = selector.replace(matcher, '');
      }
    }
    return selector.length === 0;
  }

  /**
   * @private
   * to support IE8 missing getElementById on SVGdocument
   */
  function elementById(doc, id) {
    var el;
    doc.getElementById && (el = doc.getElementById(id));
    if (el) {
      return el;
    }
    var node, i, nodelist = doc.getElementsByTagName('*');
    for (i = 0; i < nodelist.length; i++) {
      node = nodelist[i];
      if (id === node.getAttribute('id')) {
        return node;
      }
    }
  }

  /**
   * @private
   */
  function parseUseDirectives(doc) {
    var nodelist = _getMultipleNodes(doc, ['use', 'svg:use']), i = 0;
```

```
    while (nodelist.length && i < nodelist.length) {
      var el = nodelist[i],
          xlink = el.getAttribute('xlink:href').substr(1),
          x = el.getAttribute('x') || 0,
          y = el.getAttribute('y') || 0,
          el2 = elementById(doc, xlink).cloneNode(true),
          currentTrans = (el2.getAttribute('transform') || '') + ' translate(' +
x + ', ' + y + ')',
          parentNode, oldLength = nodelist.length, attr, j, attrs, l;

      applyViewboxTransform(el2);
      if (/^svg$/i.test(el2.nodeName)) {
        var el3 = el2.ownerDocument.createElement('g');
        for (j = 0, attrs = el2.attributes, l = attrs.length; j < l; j++) {
          attr = attrs.item(j);
          el3.setAttribute(attr.nodeName, attr.nodeValue);
        }
        // el2.firstChild != null
        while (el2.firstChild) {
          el3.appendChild(el2.firstChild);
        }
        el2 = el3;
      }

      for (j = 0, attrs = el.attributes, l = attrs.length; j < l; j++) {
        attr = attrs.item(j);
        if (attr.nodeName === 'x' || attr.nodeName === 'y' || attr.nodeName ===
'xlink:href') {
          continue;
        }

        if (attr.nodeName === 'transform') {
          currentTrans = attr.nodeValue + ' ' + currentTrans;
        }
        else {
          el2.setAttribute(attr.nodeName, attr.nodeValue);
        }
      }

      el2.setAttribute('transform', currentTrans);
      el2.setAttribute('instantiated_by_use', '1');
      el2.removeAttribute('id');
      parentNode = el.parentNode;
      parentNode.replaceChild(el2, el);
      // some browsers do not shorten nodelist after replaceChild (IE8)
      if (nodelist.length === oldLength) {
        i++;
      }
    }
  }

  // http://www.w3.org/TR/SVG/coords.html#ViewBoxAttribute
  // matches, e.g.: +14.56e-12, etc.
  var reViewBoxAttrValue = new RegExp(
    '^' +
    '\\s*(' + fabric.reNum + '+)\\s*,?' +
    '\\s*(' + fabric.reNum + '+)\\s*,?' +
    '\\s*(' + fabric.reNum + '+)\\s*,?' +
    '\\s*(' + fabric.reNum + '+)\\s*' +
    '$'
  );

  /**
   * Add a <g> element that envelop all child elements and makes the viewbox
```

```
transformMatrix descend on all elements
   */
  function applyViewboxTransform(element) {

    var viewBoxAttr = element.getAttribute('viewBox'),
        scaleX = 1,
        scaleY = 1,
        minX = 0,
        minY = 0,
        viewBoxWidth, viewBoxHeight, matrix, el,
        widthAttr = element.getAttribute('width'),
        heightAttr = element.getAttribute('height'),
        x = element.getAttribute('x') || 0,
        y = element.getAttribute('y') || 0,
        preserveAspectRatio = element.getAttribute('preserveAspectRatio') || '',
        missingViewBox = (!viewBoxAttr || !
reViewBoxTagNames.test(element.nodeName)
                          || !(viewBoxAttr =
viewBoxAttr.match(reViewBoxAttrValue))),
        missingDimAttr = (!widthAttr || !heightAttr || widthAttr === '100%' ||
heightAttr === '100%'),
        toBeParsed = missingViewBox && missingDimAttr,
        parsedDim = { }, translateMatrix = '';

    parsedDim.width = 0;
    parsedDim.height = 0;
    parsedDim.toBeParsed = toBeParsed;

    if (toBeParsed) {
      return parsedDim;
    }

    if (missingViewBox) {
      parsedDim.width = parseUnit(widthAttr);
      parsedDim.height = parseUnit(heightAttr);
      return parsedDim;
    }

    minX = -parseFloat(viewBoxAttr[1]);
    minY = -parseFloat(viewBoxAttr[2]);
    viewBoxWidth = parseFloat(viewBoxAttr[3]);
    viewBoxHeight = parseFloat(viewBoxAttr[4]);

    if (!missingDimAttr) {
      parsedDim.width = parseUnit(widthAttr);
      parsedDim.height = parseUnit(heightAttr);
      scaleX = parsedDim.width / viewBoxWidth;
      scaleY = parsedDim.height / viewBoxHeight;
    }
    else {
      parsedDim.width = viewBoxWidth;
      parsedDim.height = viewBoxHeight;
    }

    // default is to preserve aspect ratio
    preserveAspectRatio =
fabric.util.parsePreserveAspectRatioAttribute(preserveAspectRatio);
    if (preserveAspectRatio.alignX !== 'none') {
      //translate all container for the effect of Mid, Min, Max
      scaleY = scaleX = (scaleX > scaleY ? scaleY : scaleX);
    }

    if (scaleX === 1 && scaleY === 1 && minX === 0 && minY === 0 && x === 0 && y
=== 0) {
```

```
      return parsedDim;
    }

    if (x || y) {
      translateMatrix = ' translate(' + parseUnit(x) + ' ' + parseUnit(y) + ')
';
    }

    matrix = translateMatrix + ' matrix(' + scaleX +
                  ' 0' +
                  ' 0 ' +
                  scaleY + ' ' +
                  (minX * scaleX) + ' ' +
                  (minY * scaleY) + ') ';

    if (element.nodeName === 'svg') {
      el = element.ownerDocument.createElement('g');
      // element.firstChild != null
      while (element.firstChild) {
        el.appendChild(element.firstChild);
      }
      element.appendChild(el);
    }
    else {
      el = element;
      matrix = el.getAttribute('transform') + matrix;
    }

    el.setAttribute('transform', matrix);
    return parsedDim;
  }

  function hasAncestorWithNodeName(element, nodeName) {
    while (element && (element = element.parentNode)) {
      if (element.nodeName && nodeName.test(element.nodeName.replace('svg:',
''))
        && !element.getAttribute('instantiated_by_use')) {
        return true;
      }
    }
    return false;
  }

  /**
   * Parses an SVG document, converts it to an array of corresponding fabric.*
instances and passes them to a callback
   * @static
   * @function
   * @memberOf fabric
   * @param {SVGDocument} doc SVG document to parse
   * @param {Function} callback Callback to call when parsing is finished;
   * It's being passed an array of elements (parsed from a document).
   * @param {Function} [reviver] Method for further parsing of SVG elements,
called after each fabric object created.
   * @param {Object} [parsingOptions] options for parsing document
   * @param {String} [parsingOptions.crossOrigin] crossOrigin settings
   */
  fabric.parseSVGDocument = function(doc, callback, reviver, parsingOptions) {
    if (!doc) {
      return;
    }

    parseUseDirectives(doc);
```

```
    var svgUid =  fabric.Object.__uid++,
        options = applyViewboxTransform(doc),
        descendants = fabric.util.toArray(doc.getElementsByTagName('*'));
    options.crossOrigin = parsingOptions && parsingOptions.crossOrigin;
    options.svgUid = svgUid;

    if (descendants.length === 0 && fabric.isLikelyNode) {
      // we're likely in node, where "o3-xml" library fails to gEBTN("*")
      // https://github.com/ajaxorg/node-o3-xml/issues/21
      descendants = doc.selectNodes('//*[name(.)!="svg"]');
      var arr = [];
      for (var i = 0, len = descendants.length; i < len; i++) {
        arr[i] = descendants[i];
      }
      descendants = arr;
    }

    var elements = descendants.filter(function(el) {
      applyViewboxTransform(el);
      return reAllowedSVGTagNames.test(el.nodeName.replace('svg:', '')) &&
            !hasAncestorWithNodeName(el, reNotAllowedAncestors); //
http://www.w3.org/TR/SVG/struct.html#DefsElement
    });

    if (!elements || (elements && !elements.length)) {
      callback && callback([], {});
      return;
    }

    fabric.gradientDefs[svgUid] = fabric.getGradientDefs(doc);
    fabric.cssRules[svgUid] = fabric.getCSSRules(doc);
    // Precedence of rules:   style > class > attribute
    fabric.parseElements(elements, function(instances) {
      if (callback) {
        callback(instances, options);
      }
    }, clone(options), reviver, parsingOptions);
  };

  var reFontDeclaration = new RegExp(
    '(normal|italic)?\\s*(normal|small-caps)?\\s*' +
    '(normal|bold|bolder|lighter|100|200|300|400|500|600|700|800|900)?\\s*(' +
      fabric.reNum +
    '(?:px|cm|mm|em|pt|pc|in)*)(?:\\/(normal|' + fabric.reNum + '))?\\s+(.*)');

  extend(fabric, {
    /**
     * Parses a short font declaration, building adding its properties to a
style object
     * @static
     * @function
     * @memberOf fabric
     * @param {String} value font declaration
     * @param {Object} oStyle definition
     */
    parseFontDeclaration: function(value, oStyle) {
      var match = value.match(reFontDeclaration);

      if (!match) {
        return;
      }
      var fontStyle = match[1],
          // font variant is not used
          // fontVariant = match[2],
```

```
          fontWeight = match[3],
          fontSize = match[4],
          lineHeight = match[5],
          fontFamily = match[6];

      if (fontStyle) {
        oStyle.fontStyle = fontStyle;
      }
      if (fontWeight) {
        oStyle.fontWeight = isNaN(parseFloat(fontWeight)) ? fontWeight :
parseFloat(fontWeight);
      }
      if (fontSize) {
        oStyle.fontSize = parseUnit(fontSize);
      }
      if (fontFamily) {
        oStyle.fontFamily = fontFamily;
      }
      if (lineHeight) {
        oStyle.lineHeight = lineHeight === 'normal' ? 1 : lineHeight;
      }
    },

    /**
     * Parses an SVG document, returning all of the gradient declarations found
in it
     * @static
     * @function
     * @memberOf fabric
     * @param {SVGDocument} doc SVG document to parse
     * @return {Object} Gradient definitions; key corresponds to element id,
value -- to gradient definition element
     */
    getGradientDefs: function(doc) {
      var tagArray = [
            'linearGradient',
            'radialGradient',
            'svg:linearGradient',
            'svg:radialGradient'],
          elList = _getMultipleNodes(doc, tagArray),
          el, j = 0, id, xlink,
          gradientDefs = { }, idsToXlinkMap = { };

      j = elList.length;

      while (j--) {
        el = elList[j];
        xlink = el.getAttribute('xlink:href');
        id = el.getAttribute('id');
        if (xlink) {
          idsToXlinkMap[id] = xlink.substr(1);
        }
        gradientDefs[id] = el;
      }

      for (id in idsToXlinkMap) {
        var el2 = gradientDefs[idsToXlinkMap[id]].cloneNode(true);
        el = gradientDefs[id];
        while (el2.firstChild) {
          el.appendChild(el2.firstChild);
        }
      }
      return gradientDefs;
    },
```

```
    /**
     * Returns an object of attributes' name/value, given element and an array
of attribute names;
     * Parses parent "g" nodes recursively upwards.
     * @static
     * @memberOf fabric
     * @param {DOMElement} element Element to parse
     * @param {Array} attributes Array of attributes to parse
     * @return {Object} object containing parsed attributes' names/values
     */
    parseAttributes: function(element, attributes, svgUid) {

      if (!element) {
        return;
      }

      var value,
          parentAttributes = { },
          fontSize;

      if (typeof svgUid === 'undefined') {
        svgUid = element.getAttribute('svgUid');
      }
      // if there's a parent container (`g` or `a` or `symbol` node), parse its
attributes recursively upwards
      if (element.parentNode &&
reAllowedParents.test(element.parentNode.nodeName)) {
        parentAttributes = fabric.parseAttributes(element.parentNode,
attributes, svgUid);
      }
      fontSize = (parentAttributes && parentAttributes.fontSize ) ||
                 element.getAttribute('font-size') ||
fabric.Text.DEFAULT_SVG_FONT_SIZE;

      var ownAttributes = attributes.reduce(function(memo, attr) {
        value = element.getAttribute(attr);
        if (value) { // eslint-disable-line
          memo[attr] = value;
        }
        return memo;
      }, { });
      // add values parsed from style, which take precedence over attributes
      // (see:
http://www.w3.org/TR/SVG/styling.html#UsingPresentationAttributes)
      ownAttributes = extend(ownAttributes,
        extend(getGlobalStylesForElement(element, svgUid),
fabric.parseStyleAttribute(element)));

      var normalizedAttr, normalizedValue, normalizedStyle = {};
      for (var attr in ownAttributes) {
        normalizedAttr = normalizeAttr(attr);
        normalizedValue = normalizeValue(normalizedAttr, ownAttributes[attr],
parentAttributes, fontSize);
        normalizedStyle[normalizedAttr] = normalizedValue;
      }
      if (normalizedStyle && normalizedStyle.font) {
        fabric.parseFontDeclaration(normalizedStyle.font, normalizedStyle);
      }
      var mergedAttrs = extend(parentAttributes, normalizedStyle);
      return reAllowedParents.test(element.nodeName) ? mergedAttrs :
_setStrokeFillOpacity(mergedAttrs);
    },
```

```
    /**
     * Transforms an array of svg elements to corresponding fabric.* instances
     * @static
     * @memberOf fabric
     * @param {Array} elements Array of elements to parse
     * @param {Function} callback Being passed an array of fabric instances
(transformed from SVG elements)
     * @param {Object} [options] Options object
     * @param {Function} [reviver] Method for further parsing of SVG elements,
called after each fabric object created.
     */
    parseElements: function(elements, callback, options, reviver,
parsingOptions) {
      new fabric.ElementsParser(elements, callback, options, reviver,
parsingOptions).parse();
    },

    /**
     * Parses "style" attribute, retuning an object with values
     * @static
     * @memberOf fabric
     * @param {SVGElement} element Element to parse
     * @return {Object} Objects with values parsed from style attribute of an
element
     */
    parseStyleAttribute: function(element) {
      var oStyle = { },
          style = element.getAttribute('style');

      if (!style) {
        return oStyle;
      }

      if (typeof style === 'string') {
        parseStyleString(style, oStyle);
      }
      else {
        parseStyleObject(style, oStyle);
      }

      return oStyle;
    },

    /**
     * Parses "points" attribute, returning an array of values
     * @static
     * @memberOf fabric
     * @param {String} points points attribute string
     * @return {Array} array of points
     */
    parsePointsAttribute: function(points) {

      // points attribute is required and must not be empty
      if (!points) {
        return null;
      }

      // replace commas with whitespace and remove bookending whitespace
      points = points.replace(/,/g, ' ').trim();

      points = points.split(/\s+/);
      var parsedPoints = [], i, len;

      i = 0;
```

```
      len = points.length;
      for (; i < len; i += 2) {
        parsedPoints.push({
          x: parseFloat(points[i]),
          y: parseFloat(points[i + 1])
        });
      }

      // odd number of points is an error
      // if (parsedPoints.length % 2 !== 0) {
      //   return null;
      // }

      return parsedPoints;
    },

    /**
     * Returns CSS rules for a given SVG document
     * @static
     * @function
     * @memberOf fabric
     * @param {SVGDocument} doc SVG document to parse
     * @return {Object} CSS rules of this document
     */
    getCSSRules: function(doc) {
      var styles = doc.getElementsByTagName('style'),
          allRules = { }, rules;

      // very crude parsing of style contents
      for (var i = 0, len = styles.length; i < len; i++) {
        // IE9 doesn't support textContent, but provides text instead.
        var styleContents = styles[i].textContent || styles[i].text;

        // remove comments
        styleContents = styleContents.replace(/\/\*[\s\S]*?\*\//g, '');
        if (styleContents.trim() === '') {
          continue;
        }
        rules = styleContents.match(/[^{]*\{[\s\S]*?\}/g);
        rules = rules.map(function(rule) { return rule.trim(); });
        rules.forEach(function(rule) {

          var match = rule.match(/([\s\S]*?)\s*\{([^}]*)\}/),
              ruleObj = { }, declaration = match[2].trim(),
              propertyValuePairs = declaration.replace(/;$/, '').split(/\s*;\
s*/);

          for (var i = 0, len = propertyValuePairs.length; i < len; i++) {
            var pair = propertyValuePairs[i].split(/\s*:\s*/),
                property = pair[0],
                value = pair[1];
            ruleObj[property] = value;
          }
          rule = match[1];
          rule.split(',').forEach(function(_rule) {
            _rule = _rule.replace(/^svg/i, '').trim();
            if (_rule === '') {
              return;
            }
            if (allRules[_rule]) {
              fabric.util.object.extend(allRules[_rule], ruleObj);
            }
            else {
              allRules[_rule] = fabric.util.object.clone(ruleObj);
```

```
          }
        });
      });
    }
    return allRules;
  },

  /**
   * Takes url corresponding to an SVG document, and parses it into a set of
fabric objects.
   * Note that SVG is fetched via XMLHttpRequest, so it needs to conform to
SOP (Same Origin Policy)
   * @memberOf fabric
   * @param {String} url
   * @param {Function} callback
   * @param {Function} [reviver] Method for further parsing of SVG elements,
called after each fabric object created.
   * @param {Object} [options] Object containing options for parsing
   * @param {String} [options.crossOrigin] crossOrigin crossOrigin setting to
use for external resources
   */
  loadSVGFromURL: function(url, callback, reviver, options) {

    url = url.replace(/^\n\s*/, '').trim();
    new fabric.util.request(url, {
      method: 'get',
      onComplete: onComplete
    });

    function onComplete(r) {

      var xml = r.responseXML;
      if (xml && !xml.documentElement && fabric.window.ActiveXObject &&
r.responseText) {
        xml = new ActiveXObject('Microsoft.XMLDOM');
        xml.async = 'false';
        //IE chokes on DOCTYPE
        xml.loadXML(r.responseText.replace(/<!DOCTYPE[\s\S]*?(\[[\s\S]*\])*?
>/i, ''));
      }
      if (!xml || !xml.documentElement) {
        callback && callback(null);
      }

      fabric.parseSVGDocument(xml.documentElement, function (results,
_options) {
        callback && callback(results, _options);
      }, reviver, options);
    }
  },

  /**
   * Takes string corresponding to an SVG document, and parses it into a set
of fabric objects
   * @memberOf fabric
   * @param {String} string
   * @param {Function} callback
   * @param {Function} [reviver] Method for further parsing of SVG elements,
called after each fabric object created.
   * @param {Object} [options] Object containing options for parsing
   * @param {String} [options.crossOrigin] crossOrigin crossOrigin setting to
use for external resources
   */
  loadSVGFromString: function(string, callback, reviver, options) {
```

```
      string = string.trim();
      var doc;
      if (typeof DOMParser !== 'undefined') {
        var parser = new DOMParser();
        if (parser && parser.parseFromString) {
          doc = parser.parseFromString(string, 'text/xml');
        }
      }
      else if (fabric.window.ActiveXObject) {
        doc = new ActiveXObject('Microsoft.XMLDOM');
        doc.async = 'false';
        // IE chokes on DOCTYPE
        doc.loadXML(string.replace(/<!DOCTYPE[\s\S]*?(\[[\s\S]*\])*?>/i, ''));
      }

      fabric.parseSVGDocument(doc.documentElement, function (results, _options)
{
        callback(results, _options);
      }, reviver, options);
    }
  });

})(typeof exports !== 'undefined' ? exports : this);


fabric.ElementsParser = function(elements, callback, options, reviver,
parsingOptions) {
  this.elements = elements;
  this.callback = callback;
  this.options = options;
  this.reviver = reviver;
  this.svgUid = (options && options.svgUid) || 0;
  this.parsingOptions = parsingOptions;
};

fabric.ElementsParser.prototype.parse = function() {
  this.instances = new Array(this.elements.length);
  this.numElements = this.elements.length;

  this.createObjects();
};

fabric.ElementsParser.prototype.createObjects = function() {
  for (var i = 0, len = this.elements.length; i < len; i++) {
    this.elements[i].setAttribute('svgUid', this.svgUid);
    (function(_obj, i) {
      setTimeout(function() {
        _obj.createObject(_obj.elements[i], i);
      }, 0);
    })(this, i);
  }
};

fabric.ElementsParser.prototype.createObject = function(el, index) {
  var klass = fabric[fabric.util.string.capitalize(el.tagName.replace('svg:',
''))];
  if (klass && klass.fromElement) {
    try {
      this._createObject(klass, el, index);
    }
    catch (err) {
      fabric.log(err);
    }
  }
```

```
  else {
    this.checkIfDone();
  }
};

fabric.ElementsParser.prototype._createObject = function(klass, el, index) {
  if (klass.async) {
    klass.fromElement(el, this.createCallback(index, el), this.options);
  }
  else {
    var obj = klass.fromElement(el, this.options);
    this.resolveGradient(obj, 'fill');
    this.resolveGradient(obj, 'stroke');
    this.reviver && this.reviver(el, obj);
    this.instances[index] = obj;
    this.checkIfDone();
  }
};

fabric.ElementsParser.prototype.createCallback = function(index, el) {
  var _this = this;
  return function(obj) {
    _this.resolveGradient(obj, 'fill');
    _this.resolveGradient(obj, 'stroke');
    _this.reviver && _this.reviver(el, obj);
    _this.instances[index] = obj;
    _this.checkIfDone();
  };
};

fabric.ElementsParser.prototype.resolveGradient = function(obj, property) {

  var instanceFillValue = obj.get(property);
  if (!(/^url\(/).test(instanceFillValue)) {
    return;
  }
  var gradientId = instanceFillValue.slice(5, instanceFillValue.length - 1);
  if (fabric.gradientDefs[this.svgUid][gradientId]) {
    obj.set(property,
      fabric.Gradient.fromElement(fabric.gradientDefs[this.svgUid][gradientId],
obj));
  }
};

fabric.ElementsParser.prototype.checkIfDone = function() {
  if (--this.numElements === 0) {
    this.instances = this.instances.filter(function(el) {
      // eslint-disable-next-line no-eq-null, eqeqeq
      return el != null;
    });
    this.callback(this.instances);
  }
};


(function(global) {

  'use strict';

  /* Adaptation of work of Kevin Lindsey (kevin@kevlindev.com) */

  var fabric = global.fabric || (global.fabric = { });

  if (fabric.Point) {
```

```
  fabric.warn('fabric.Point is already defined');
  return;
}

fabric.Point = Point;

/**
 * Point class
 * @class fabric.Point
 * @memberOf fabric
 * @constructor
 * @param {Number} x
 * @param {Number} y
 * @return {fabric.Point} thisArg
 */
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = /** @lends fabric.Point.prototype */ {

  type: 'point',

  constructor: Point,

  /**
   * Adds another point to this one and returns another one
   * @param {fabric.Point} that
   * @return {fabric.Point} new Point instance with added values
   */
  add: function (that) {
    return new Point(this.x + that.x, this.y + that.y);
  },

  /**
   * Adds another point to this one
   * @param {fabric.Point} that
   * @return {fabric.Point} thisArg
   * @chainable
   */
  addEquals: function (that) {
    this.x += that.x;
    this.y += that.y;
    return this;
  },

  /**
   * Adds value to this point and returns a new one
   * @param {Number} scalar
   * @return {fabric.Point} new Point with added value
   */
  scalarAdd: function (scalar) {
    return new Point(this.x + scalar, this.y + scalar);
  },

  /**
   * Adds value to this point
   * @param {Number} scalar
   * @return {fabric.Point} thisArg
   * @chainable
   */
  scalarAddEquals: function (scalar) {
    this.x += scalar;
```

```
    this.y += scalar;
    return this;
  },

  /**
   * Subtracts another point from this point and returns a new one
   * @param {fabric.Point} that
   * @return {fabric.Point} new Point object with subtracted values
   */
  subtract: function (that) {
    return new Point(this.x - that.x, this.y - that.y);
  },

  /**
   * Subtracts another point from this point
   * @param {fabric.Point} that
   * @return {fabric.Point} thisArg
   * @chainable
   */
  subtractEquals: function (that) {
    this.x -= that.x;
    this.y -= that.y;
    return this;
  },

  /**
   * Subtracts value from this point and returns a new one
   * @param {Number} scalar
   * @return {fabric.Point}
   */
  scalarSubtract: function (scalar) {
    return new Point(this.x - scalar, this.y - scalar);
  },

  /**
   * Subtracts value from this point
   * @param {Number} scalar
   * @return {fabric.Point} thisArg
   * @chainable
   */
  scalarSubtractEquals: function (scalar) {
    this.x -= scalar;
    this.y -= scalar;
    return this;
  },

  /**
   * Miltiplies this point by a value and returns a new one
   * TODO: rename in scalarMultiply in 2.0
   * @param {Number} scalar
   * @return {fabric.Point}
   */
  multiply: function (scalar) {
    return new Point(this.x * scalar, this.y * scalar);
  },

  /**
   * Miltiplies this point by a value
   * TODO: rename in scalarMultiplyEquals in 2.0
   * @param {Number} scalar
   * @return {fabric.Point} thisArg
   * @chainable
   */
  multiplyEquals: function (scalar) {
```

```
  this.x *= scalar;
  this.y *= scalar;
  return this;
},

/**
 * Divides this point by a value and returns a new one
 * TODO: rename in scalarDivide in 2.0
 * @param {Number} scalar
 * @return {fabric.Point}
 */
divide: function (scalar) {
  return new Point(this.x / scalar, this.y / scalar);
},

/**
 * Divides this point by a value
 * TODO: rename in scalarDivideEquals in 2.0
 * @param {Number} scalar
 * @return {fabric.Point} thisArg
 * @chainable
 */
divideEquals: function (scalar) {
  this.x /= scalar;
  this.y /= scalar;
  return this;
},

/**
 * Returns true if this point is equal to another one
 * @param {fabric.Point} that
 * @return {Boolean}
 */
eq: function (that) {
  return (this.x === that.x && this.y === that.y);
},

/**
 * Returns true if this point is less than another one
 * @param {fabric.Point} that
 * @return {Boolean}
 */
lt: function (that) {
  return (this.x < that.x && this.y < that.y);
},

/**
 * Returns true if this point is less than or equal to another one
 * @param {fabric.Point} that
 * @return {Boolean}
 */
lte: function (that) {
  return (this.x <= that.x && this.y <= that.y);
},

/**

 * Returns true if this point is greater another one
 * @param {fabric.Point} that
 * @return {Boolean}
 */
gt: function (that) {
  return (this.x > that.x && this.y > that.y);
},
```

```
    /**
     * Returns true if this point is greater than or equal to another one
     * @param {fabric.Point} that
     * @return {Boolean}
     */
    gte: function (that) {
      return (this.x >= that.x && this.y >= that.y);
    },

    /**
     * Returns new point which is the result of linear interpolation with this
one and another one
     * @param {fabric.Point} that
     * @param {Number} t , position of interpolation, between 0 and 1 default
0.5
     * @return {fabric.Point}
     */
    lerp: function (that, t) {
      if (typeof t === 'undefined') {
        t = 0.5;
      }
      t = Math.max(Math.min(1, t), 0);
      return new Point(this.x + (that.x - this.x) * t, this.y + (that.y -
this.y) * t);
    },

    /**
     * Returns distance from this point and another one
     * @param {fabric.Point} that
     * @return {Number}
     */
    distanceFrom: function (that) {
      var dx = this.x - that.x,
          dy = this.y - that.y;
      return Math.sqrt(dx * dx + dy * dy);
    },

    /**
     * Returns the point between this point and another one
     * @param {fabric.Point} that
     * @return {fabric.Point}
     */
    midPointFrom: function (that) {
      return this.lerp(that);
    },

    /**
     * Returns a new point which is the min of this and another one
     * @param {fabric.Point} that
     * @return {fabric.Point}
     */
    min: function (that) {
      return new Point(Math.min(this.x, that.x), Math.min(this.y, that.y));
    },

    /**
     * Returns a new point which is the max of this and another one
     * @param {fabric.Point} that
     * @return {fabric.Point}
     */
    max: function (that) {
      return new Point(Math.max(this.x, that.x), Math.max(this.y, that.y));
    },
```

```
/**
 * Returns string representation of this point
 * @return {String}
 */
toString: function () {
  return this.x + ',' + this.y;
},

/**
 * Sets x/y of this point
 * @param {Number} x
 * @param {Number} y
 * @chainable
 */
setXY: function (x, y) {
  this.x = x;
  this.y = y;
  return this;
},

/**
 * Sets x of this point
 * @param {Number} x
 * @chainable
 */
setX: function (x) {
  this.x = x;
  return this;
},

/**
 * Sets y of this point
 * @param {Number} y
 * @chainable
 */
setY: function (y) {
  this.y = y;
  return this;
},

/**
 * Sets x/y of this point from another point
 * @param {fabric.Point} that
 * @chainable
 */
setFromPoint: function (that) {
  this.x = that.x;
  this.y = that.y;
  return this;
},

/**
 * Swaps x/y of this point and another point
 * @param {fabric.Point} that
 */
swap: function (that) {
  var x = this.x,
      y = this.y;
  this.x = that.x;
  this.y = that.y;
  that.x = x;
  that.y = y;
},
```

```
    /**
     * return a cloned instance of the point
     * @return {fabric.Point}
     */
    clone: function () {
      return new Point(this.x, this.y);
    }
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  /* Adaptation of work of Kevin Lindsey (kevin@kevlindev.com) */
  var fabric = global.fabric || (global.fabric = { });

  if (fabric.Intersection) {
    fabric.warn('fabric.Intersection is already defined');
    return;
  }

  /**
   * Intersection class
   * @class fabric.Intersection
   * @memberOf fabric
   * @constructor
   */
  function Intersection(status) {
    this.status = status;
    this.points = [];
  }

  fabric.Intersection = Intersection;

  fabric.Intersection.prototype = /** @lends fabric.Intersection.prototype */ {

    constructor: Intersection,

    /**
     * Appends a point to intersection
     * @param {fabric.Point} point
     * @return {fabric.Intersection} thisArg
     * @chainable
     */
    appendPoint: function (point) {
      this.points.push(point);
      return this;
    },

    /**
     * Appends points to intersection
     * @param {Array} points
     * @return {fabric.Intersection} thisArg
     * @chainable
     */
    appendPoints: function (points) {
      this.points = this.points.concat(points);
      return this;
    }
  };
```

```
  /**
   * Checks if one line intersects another
   * TODO: rename in intersectSegmentSegment
   * @static
   * @param {fabric.Point} a1
   * @param {fabric.Point} a2
   * @param {fabric.Point} b1
   * @param {fabric.Point} b2
   * @return {fabric.Intersection}
   */
  fabric.Intersection.intersectLineLine = function (a1, a2, b1, b2) {
    var result,
        uaT = (b2.x - b1.x) * (a1.y - b1.y) - (b2.y - b1.y) * (a1.x - b1.x),
        ubT = (a2.x - a1.x) * (a1.y - b1.y) - (a2.y - a1.y) * (a1.x - b1.x),
        uB = (b2.y - b1.y) * (a2.x - a1.x) - (b2.x - b1.x) * (a2.y - a1.y);
    if (uB !== 0) {
      var ua = uaT / uB,
          ub = ubT / uB;
      if (0 <= ua && ua <= 1 && 0 <= ub && ub <= 1) {
        result = new Intersection('Intersection');
        result.appendPoint(new fabric.Point(a1.x + ua * (a2.x - a1.x), a1.y + ua
* (a2.y - a1.y)));
      }
      else {
        result = new Intersection();
      }
    }
    else {
      if (uaT === 0 || ubT === 0) {
        result = new Intersection('Coincident');
      }
      else {
        result = new Intersection('Parallel');
      }
    }
    return result;
  };

  /**
   * Checks if line intersects polygon
   * TODO: rename in intersectSegmentPolygon
   * fix detection of coincident
   * @static
   * @param {fabric.Point} a1
   * @param {fabric.Point} a2
   * @param {Array} points
   * @return {fabric.Intersection}
   */
  fabric.Intersection.intersectLinePolygon = function(a1, a2, points) {
    var result = new Intersection(),
        length = points.length,
        b1, b2, inter;

    for (var i = 0; i < length; i++) {
      b1 = points[i];
      b2 = points[(i + 1) % length];
      inter = Intersection.intersectLineLine(a1, a2, b1, b2);

      result.appendPoints(inter.points);
    }
    if (result.points.length > 0) {
      result.status = 'Intersection';
    }
```

```
    return result;
  };

  /**
   * Checks if polygon intersects another polygon
   * @static
   * @param {Array} points1
   * @param {Array} points2
   * @return {fabric.Intersection}
   */
  fabric.Intersection.intersectPolygonPolygon = function (points1, points2) {
    var result = new Intersection(),
        length = points1.length;

    for (var i = 0; i < length; i++) {
      var a1 = points1[i],
          a2 = points1[(i + 1) % length],
          inter = Intersection.intersectLinePolygon(a1, a2, points2);

      result.appendPoints(inter.points);
    }
    if (result.points.length > 0) {
      result.status = 'Intersection';
    }
    return result;
  };

  /**
   * Checks if polygon intersects rectangle
   * @static
   * @param {Array} points
   * @param {fabric.Point} r1
   * @param {fabric.Point} r2
   * @return {fabric.Intersection}
   */
  fabric.Intersection.intersectPolygonRectangle = function (points, r1, r2) {
    var min = r1.min(r2),
        max = r1.max(r2),
        topRight = new fabric.Point(max.x, min.y),
        bottomLeft = new fabric.Point(min.x, max.y),
        inter1 = Intersection.intersectLinePolygon(min, topRight, points),
        inter2 = Intersection.intersectLinePolygon(topRight, max, points),
        inter3 = Intersection.intersectLinePolygon(max, bottomLeft, points),
        inter4 = Intersection.intersectLinePolygon(bottomLeft, min, points),
        result = new Intersection();

    result.appendPoints(inter1.points);
    result.appendPoints(inter2.points);
    result.appendPoints(inter3.points);
    result.appendPoints(inter4.points);

    if (result.points.length > 0) {
      result.status = 'Intersection';
    }
    return result;
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';
```

```
  var fabric = global.fabric || (global.fabric = { });

  if (fabric.Color) {
    fabric.warn('fabric.Color is already defined.');
    return;
  }

  /**
   * Color class
   * The purpose of {@link fabric.Color} is to abstract and encapsulate common
color operations;
   * {@link fabric.Color} is a constructor and creates instances of {@link
fabric.Color} objects.
   *
   * @class fabric.Color
   * @param {String} color optional in hex or rgb(a) or hsl format or from known
color list
   * @return {fabric.Color} thisArg
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-2/#colors}
   */
  function Color(color) {
    if (!color) {
      this.setSource([0, 0, 0, 1]);
    }
    else {
      this._tryParsingColor(color);
    }
  }

  fabric.Color = Color;

  fabric.Color.prototype = /** @lends fabric.Color.prototype */ {

    /**
     * @private
     * @param {String|Array} color Color value to parse
     */
    _tryParsingColor: function(color) {
      var source;

      if (color in Color.colorNameMap) {
        color = Color.colorNameMap[color];
      }

      if (color === 'transparent') {
        source = [255, 255, 255, 0];
      }

      if (!source) {
        source = Color.sourceFromHex(color);
      }
      if (!source) {
        source = Color.sourceFromRgb(color);
      }
      if (!source) {
        source = Color.sourceFromHsl(color);
      }
      if (!source) {
        //if color is not recognize let's make black as canvas does
        source = [0, 0, 0, 1];
      }
      if (source) {
        this.setSource(source);
      }
```

```
    },

    /**
     * Adapted from <a
href="https://rawgithub.com/mjijackson/mjijackson.github.com/master/2008/02/rgb-
to-hsl-and-rgb-to-hsv-color-model-conversion-algorithms-in-
javascript.html">https://github.com/mjijackson</a>
     * @private
     * @param {Number} r Red color value
     * @param {Number} g Green color value
     * @param {Number} b Blue color value
     * @return {Array} Hsl color
     */
    _rgbToHsl: function(r, g, b) {
      r /= 255; g /= 255; b /= 255;

      var h, s, l,
          max = fabric.util.array.max([r, g, b]),
          min = fabric.util.array.min([r, g, b]);

      l = (max + min) / 2;

      if (max === min) {
        h = s = 0; // achromatic
      }
      else {
        var d = max - min;
        s = l > 0.5 ? d / (2 - max - min) : d / (max + min);
        switch (max) {
          case r:
            h = (g - b) / d + (g < b ? 6 : 0);
            break;
          case g:
            h = (b - r) / d + 2;
            break;
          case b:
            h = (r - g) / d + 4;
            break;
        }
        h /= 6;
      }

      return [
        Math.round(h * 360),
        Math.round(s * 100),
        Math.round(l * 100)
      ];
    },

    /**
     * Returns source of this color (where source is an array representation;
ex: [200, 200, 100, 1])
     * @return {Array}
     */
    getSource: function() {
      return this._source;
    },

    /**
     * Sets source of this color (where source is an array representation; ex:
[200, 200, 100, 1])
     * @param {Array} source
     */
    setSource: function(source) {
```

```
      this._source = source;
    },

    /**
     * Returns color representation in RGB format
     * @return {String} ex: rgb(0-255,0-255,0-255)
     */
    toRgb: function() {
      var source = this.getSource();
      return 'rgb(' + source[0] + ',' + source[1] + ',' + source[2] + ')';
    },

    /**
     * Returns color representation in RGBA format
     * @return {String} ex: rgba(0-255,0-255,0-255,0-1)
     */
    toRgba: function() {
      var source = this.getSource();
      return 'rgba(' + source[0] + ',' + source[1] + ',' + source[2] + ',' +
source[3] + ')';
    },

    /**
     * Returns color representation in HSL format
     * @return {String} ex: hsl(0-360,0%-100%,0%-100%)
     */
    toHsl: function() {
      var source = this.getSource(),
          hsl = this._rgbToHsl(source[0], source[1], source[2]);

      return 'hsl(' + hsl[0] + ',' + hsl[1] + '%,' + hsl[2] + '%)';
    },

    /**
     * Returns color representation in HSLA format
     * @return {String} ex: hsla(0-360,0%-100%,0%-100%,0-1)
     */
    toHsla: function() {
      var source = this.getSource(),
          hsl = this._rgbToHsl(source[0], source[1], source[2]);

      return 'hsla(' + hsl[0] + ',' + hsl[1] + '%,' + hsl[2] + '%,' + source[3]
+ ')';
    },

    /**
     * Returns color representation in HEX format
     * @return {String} ex: FF5555
     */
    toHex: function() {
      var source = this.getSource(), r, g, b;

      r = source[0].toString(16);
      r = (r.length === 1) ? ('0' + r) : r;

      g = source[1].toString(16);
      g = (g.length === 1) ? ('0' + g) : g;

      b = source[2].toString(16);
      b = (b.length === 1) ? ('0' + b) : b;

      return r.toUpperCase() + g.toUpperCase() + b.toUpperCase();
    },
```

```
    /**
     * Returns color representation in HEXA format
     * @return {String} ex: FF5555CC
     */
    toHexa: function() {
      var source = this.getSource(), a;

      a = source[3] * 255;
      a = a.toString(16);
      a = (a.length === 1) ? ('0' + a) : a;

      return this.toHex() + a.toUpperCase();
    },

    /**
     * Gets value of alpha channel for this color
     * @return {Number} 0-1
     */
    getAlpha: function() {
      return this.getSource()[3];
    },

    /**
     * Sets value of alpha channel for this color
     * @param {Number} alpha Alpha value 0-1
     * @return {fabric.Color} thisArg
     */
    setAlpha: function(alpha) {
      var source = this.getSource();
      source[3] = alpha;
      this.setSource(source);
      return this;
    },

    /**
     * Transforms color to its grayscale representation
     * @return {fabric.Color} thisArg
     */
    toGrayscale: function() {
      var source = this.getSource(),
          average = parseInt((source[0] * 0.3 + source[1] * 0.59 + source[2] *
0.11).toFixed(0), 10),
          currentAlpha = source[3];
      this.setSource([average, average, average, currentAlpha]);
      return this;
    },

    /**
     * Transforms color to its black and white representation
     * @param {Number} threshold
     * @return {fabric.Color} thisArg
     */
    toBlackWhite: function(threshold) {
      var source = this.getSource(),
          average = (source[0] * 0.3 + source[1] * 0.59 + source[2] *
0.11).toFixed(0),
          currentAlpha = source[3];

      threshold = threshold || 127;

      average = (Number(average) < Number(threshold)) ? 0 : 255;
      this.setSource([average, average, average, currentAlpha]);
      return this;
    },
```

```
    /**
     * Overlays color with another color
     * @param {String|fabric.Color} otherColor
     * @return {fabric.Color} thisArg
     */
    overlayWith: function(otherColor) {
      if (!(otherColor instanceof Color)) {
        otherColor = new Color(otherColor);
      }

      var result = [],
          alpha = this.getAlpha(),
          otherAlpha = 0.5,
          source = this.getSource(),
          otherSource = otherColor.getSource();

      for (var i = 0; i < 3; i++) {
        result.push(Math.round((source[i] * (1 - otherAlpha)) + (otherSource[i]
* otherAlpha)));
      }

      result[3] = alpha;
      this.setSource(result);
      return this;
    }
  };

  /**
   * Regex matching color in RGB or RGBA formats (ex: rgb(0, 0, 0), rgba(255,
100, 10, 0.5), rgba( 255 , 100 , 10 , 0.5 ), rgb(1,1,1), rgba(100%, 60%, 10%,
0.5))
   * @static
   * @field
   * @memberOf fabric.Color
   */
   // eslint-disable-next-line max-len
  fabric.Color.reRGBa = /^rgba?\(\s*(\d{1,3}(?:\.\d+)?\%?)\s*,\s*(\d{1,3}(?:\.\
d+)?\%?)\s*,\s*(\d{1,3}(?:\.\d+)?\%?)\s*(?:\s*,\s*((?:\d*\.?\d+)?)\s*)?\)$/;

  /**
   * Regex matching color in HSL or HSLA formats (ex: hsl(200, 80%, 10%),
hsla(300, 50%, 80%, 0.5), hsla( 300 , 50% , 80% , 0.5 ))
   * @static
   * @field
   * @memberOf fabric.Color
   */
  fabric.Color.reHSLa = /^hsla?\(\s*(\d{1,3})\s*,\s*(\d{1,3}\%)\s*,\s*(\d{1,3}\
%)\s*(?:\s*,\s*(\d+(?:\.\d+)?)\s*)?\)$/;

  /**
   * Regex matching color in HEX format (ex: #FF5544CC, #FF5555, 010155, aff)
   * @static
   * @field
   * @memberOf fabric.Color
   */
  fabric.Color.reHex = /^#?([0-9a-f]{8}|[0-9a-f]{6}|[0-9a-f]{4}|[0-9a-f]{3})$/i;

  /**
   * Map of the 17 basic color names with HEX code
   * @static
   * @field
   * @memberOf fabric.Color
   * @see: http://www.w3.org/TR/CSS2/syndata.html#color-units
```

```
   */
  fabric.Color.colorNameMap = {
    aqua:    '#00FFFF',
    black:   '#000000',
    blue:    '#0000FF',
    fuchsia: '#FF00FF',
    gray:    '#808080',
    grey:    '#808080',
    green:   '#008000',
    lime:    '#00FF00',
    maroon:  '#800000',
    navy:    '#000080',
    olive:   '#808000',
    orange:  '#FFA500',
    purple:  '#800080',
    red:     '#FF0000',
    silver:  '#C0C0C0',
    teal:    '#008080',
    white:   '#FFFFFF',
    yellow:  '#FFFF00'
  };

  /**
   * @private
   * @param {Number} p
   * @param {Number} q
   * @param {Number} t
   * @return {Number}
   */
  function hue2rgb(p, q, t) {
    if (t < 0) {
      t += 1;
    }
    if (t > 1) {
      t -= 1;
    }
    if (t < 1 / 6) {
      return p + (q - p) * 6 * t;
    }
    if (t < 1 / 2) {
      return q;
    }
    if (t < 2 / 3) {
      return p + (q - p) * (2 / 3 - t) * 6;
    }
    return p;
  }

  /**
   * Returns new color object, when given a color in RGB format
   * @memberOf fabric.Color
   * @param {String} color Color value ex: rgb(0-255,0-255,0-255)
   * @return {fabric.Color}
   */
  fabric.Color.fromRgb = function(color) {
    return Color.fromSource(Color.sourceFromRgb(color));
  };

  /**
   * Returns array representation (ex: [100, 100, 200, 1]) of a color that's in
RGB or RGBA format
   * @memberOf fabric.Color
   * @param {String} color Color value ex: rgb(0-255,0-255,0-255), rgb(0%-
100%,0%-100%,0%-100%)
```

```
   * @return {Array} source
   */
  fabric.Color.sourceFromRgb = function(color) {
    var match = color.match(Color.reRGBa);
    if (match) {
      var r = parseInt(match[1], 10) / (/%$/.test(match[1]) ? 100 : 1) * (/%
$/.test(match[1]) ? 255 : 1),
          g = parseInt(match[2], 10) / (/%$/.test(match[2]) ? 100 : 1) * (/%
$/.test(match[2]) ? 255 : 1),
          b = parseInt(match[3], 10) / (/%$/.test(match[3]) ? 100 : 1) * (/%
$/.test(match[3]) ? 255 : 1);

      return [
        parseInt(r, 10),
        parseInt(g, 10),
        parseInt(b, 10),
        match[4] ? parseFloat(match[4]) : 1
      ];
    }
  };

  /**
   * Returns new color object, when given a color in RGBA format
   * @static
   * @function
   * @memberOf fabric.Color
   * @param {String} color
   * @return {fabric.Color}
   */
  fabric.Color.fromRgba = Color.fromRgb;

  /**
   * Returns new color object, when given a color in HSL format
   * @param {String} color Color value ex: hsl(0-260,0%-100%,0%-100%)
   * @memberOf fabric.Color
   * @return {fabric.Color}
   */
  fabric.Color.fromHsl = function(color) {
    return Color.fromSource(Color.sourceFromHsl(color));
  };

  /**
   * Returns array representation (ex: [100, 100, 200, 1]) of a color that's in
HSL or HSLA format.
   * Adapted from <a
href="https://rawgithub.com/mjijackson/mjijackson.github.com/master/2008/02/rgb-
to-hsl-and-rgb-to-hsv-color-model-conversion-algorithms-in-
javascript.html">https://github.com/mjijackson</a>
   * @memberOf fabric.Color
   * @param {String} color Color value ex: hsl(0-360,0%-100%,0%-100%) or hsla(0-
360,0%-100%,0%-100%, 0-1)
   * @return {Array} source
   * @see http://http://www.w3.org/TR/css3-color/#hsl-color
   */
  fabric.Color.sourceFromHsl = function(color) {
    var match = color.match(Color.reHSLa);
    if (!match) {
      return;
    }

    var h = (((parseFloat(match[1]) % 360) + 360) % 360) / 360,
        s = parseFloat(match[2]) / (/%$/.test(match[2]) ? 100 : 1),
        l = parseFloat(match[3]) / (/%$/.test(match[3]) ? 100 : 1),
        r, g, b;
```

```
    if (s === 0) {
      r = g = b = l;
    }
    else {
      var q = l <= 0.5 ? l * (s + 1) : l + s - l * s,
          p = l * 2 - q;

      r = hue2rgb(p, q, h + 1 / 3);
      g = hue2rgb(p, q, h);
      b = hue2rgb(p, q, h - 1 / 3);
    }

    return [
      Math.round(r * 255),
      Math.round(g * 255),
      Math.round(b * 255),
      match[4] ? parseFloat(match[4]) : 1
    ];
  };

  /**
   * Returns new color object, when given a color in HSLA format
   * @static
   * @function
   * @memberOf fabric.Color
   * @param {String} color
   * @return {fabric.Color}
   */
  fabric.Color.fromHsla = Color.fromHsl;

  /**
   * Returns new color object, when given a color in HEX format
   * @static
   * @memberOf fabric.Color
   * @param {String} color Color value ex: FF5555
   * @return {fabric.Color}
   */
  fabric.Color.fromHex = function(color) {
    return Color.fromSource(Color.sourceFromHex(color));
  };

  /**
   * Returns array representation (ex: [100, 100, 200, 1]) of a color that's in
HEX format
   * @static
   * @memberOf fabric.Color
   * @param {String} color ex: FF5555 or FF5544CC (RGBa)
   * @return {Array} source
   */
  fabric.Color.sourceFromHex = function(color) {
    if (color.match(Color.reHex)) {
      var value = color.slice(color.indexOf('#') + 1),
          isShortNotation = (value.length === 3 || value.length === 4),
          isRGBa = (value.length === 8 || value.length === 4),
          r = isShortNotation ? (value.charAt(0) + value.charAt(0)) :
value.substring(0, 2),
          g = isShortNotation ? (value.charAt(1) + value.charAt(1)) :
value.substring(2, 4),
          b = isShortNotation ? (value.charAt(2) + value.charAt(2)) :
value.substring(4, 6),
          a = isRGBa ? (isShortNotation ? (value.charAt(3) + value.charAt(3)) :
value.substring(6, 8)) : 'FF';
```

```
      return [
        parseInt(r, 16),
        parseInt(g, 16),
        parseInt(b, 16),
        parseFloat((parseInt(a, 16) / 255).toFixed(2))
      ];
    }
  };

  /**
   * Returns new color object, when given color in array representation (ex:
[200, 100, 100, 0.5])
   * @static
   * @memberOf fabric.Color
   * @param {Array} source
   * @return {fabric.Color}
   */
  fabric.Color.fromSource = function(source) {
    var oColor = new Color();
    oColor.setSource(source);
    return oColor;
  };

})(typeof exports !== 'undefined' ? exports : this);


(function() {

  /* _FROM_SVG_START_ */
  function getColorStop(el) {
    var style = el.getAttribute('style'),
        offset = el.getAttribute('offset') || 0,
        color, colorAlpha, opacity;

    // convert percents to absolute values
    offset = parseFloat(offset) / (/%$/.test(offset) ? 100 : 1);
    offset = offset < 0 ? 0 : offset > 1 ? 1 : offset;
    if (style) {
      var keyValuePairs = style.split(/\s*;\s*/);

      if (keyValuePairs[keyValuePairs.length - 1] === '') {
        keyValuePairs.pop();
      }

      for (var i = keyValuePairs.length; i--; ) {

        var split = keyValuePairs[i].split(/\s*:\s*/),
            key = split[0].trim(),
            value = split[1].trim();

        if (key === 'stop-color') {
          color = value;
        }
        else if (key === 'stop-opacity') {
          opacity = value;
        }
      }
    }

    if (!color) {
      color = el.getAttribute('stop-color') || 'rgb(0,0,0)';
    }
    if (!opacity) {
      opacity = el.getAttribute('stop-opacity');
```

```
    }

    color = new fabric.Color(color);
    colorAlpha = color.getAlpha();
    opacity = isNaN(parseFloat(opacity)) ? 1 : parseFloat(opacity);
    opacity *= colorAlpha;

    return {
      offset: offset,
      color: color.toRgb(),
      opacity: opacity
    };
  }

  function getLinearCoords(el) {
    return {
      x1: el.getAttribute('x1') || 0,
      y1: el.getAttribute('y1') || 0,
      x2: el.getAttribute('x2') || '100%',
      y2: el.getAttribute('y2') || 0
    };
  }

  function getRadialCoords(el) {
    return {
      x1: el.getAttribute('fx') || el.getAttribute('cx') || '50%',
      y1: el.getAttribute('fy') || el.getAttribute('cy') || '50%',
      r1: 0,
      x2: el.getAttribute('cx') || '50%',
      y2: el.getAttribute('cy') || '50%',
      r2: el.getAttribute('r') || '50%'
    };
  }
  /* _FROM_SVG_END_ */

  var clone = fabric.util.object.clone;

  /**
   * Gradient class
   * @class fabric.Gradient
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-2#gradients}
   * @see {@link fabric.Gradient#initialize} for constructor definition
   */
  fabric.Gradient = fabric.util.createClass(/** @lends fabric.Gradient.prototype
*/ {

    /**
     * Horizontal offset for aligning gradients coming from SVG when outside
pathgroups
     * @type Number
     * @default 0
     */
    offsetX: 0,

    /**
     * Vertical offset for aligning gradients coming from SVG when outside
pathgroups
     * @type Number
     * @default 0
     */
    offsetY: 0,

    /**
     * Constructor
```

```
     * @param {Object} [options] Options object with type, coords, gradientUnits
and colorStops
     * @return {fabric.Gradient} thisArg
     */
    initialize: function(options) {
      options || (options = { });

      var coords = { };

      this.id = fabric.Object.__uid++;
      this.type = options.type || 'linear';

      coords = {
        x1: options.coords.x1 || 0,
        y1: options.coords.y1 || 0,
        x2: options.coords.x2 || 0,
        y2: options.coords.y2 || 0
      };

      if (this.type === 'radial') {
        coords.r1 = options.coords.r1 || 0;
        coords.r2 = options.coords.r2 || 0;
      }
      this.coords = coords;
      this.colorStops = options.colorStops.slice();
      if (options.gradientTransform) {
        this.gradientTransform = options.gradientTransform;
      }
      this.offsetX = options.offsetX || this.offsetX;
      this.offsetY = options.offsetY || this.offsetY;
    },

    /**
     * Adds another colorStop
     * @param {Object} colorStop Object with offset and color
     * @return {fabric.Gradient} thisArg
     */
    addColorStop: function(colorStops) {
      for (var position in colorStops) {
        var color = new fabric.Color(colorStops[position]);
        this.colorStops.push({
          offset: parseFloat(position),
          color: color.toRgb(),
          opacity: color.getAlpha()
        });
      }
      return this;
    },

    /**
     * Returns object representation of a gradient
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object}
     */
    toObject: function(propertiesToInclude) {
      var object = {
        type: this.type,
        coords: this.coords,
        colorStops: this.colorStops,
        offsetX: this.offsetX,
        offsetY: this.offsetY,
        gradientTransform: this.gradientTransform ?
this.gradientTransform.concat() : this.gradientTransform
```

```
      };
      fabric.util.populateWithProperties(this, object, propertiesToInclude);

      return object;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of an gradient
     * @param {Object} object Object to create a gradient for
     * @return {String} SVG representation of an gradient (linear/radial)
     */
    toSVG: function(object) {
      var coords = clone(this.coords, true),
          markup, commonAttributes, colorStops = clone(this.colorStops, true),
          needsSwap = coords.r1 > coords.r2;
      // colorStops must be sorted ascending
      colorStops.sort(function(a, b) {
        return a.offset - b.offset;
      });

      if (!(object.group && object.group.type === 'path-group')) {
        for (var prop in coords) {
          if (prop === 'x1' || prop === 'x2') {
            coords[prop] += this.offsetX - object.width / 2;
          }
          else if (prop === 'y1' || prop === 'y2') {
            coords[prop] += this.offsetY - object.height / 2;
          }
        }
      }

      commonAttributes = 'id="SVGID_' + this.id +
                    '" gradientUnits="userSpaceOnUse"';
      if (this.gradientTransform) {
        commonAttributes += ' gradientTransform="matrix(' +
this.gradientTransform.join(' ') + ')" ';
      }
      if (this.type === 'linear') {
        markup = [
          '<linearGradient ',
          commonAttributes,
          ' x1="', coords.x1,
          '" y1="', coords.y1,
          '" x2="', coords.x2,
          '" y2="', coords.y2,
          '">\n'
        ];
      }
      else if (this.type === 'radial') {
        // svg radial gradient has just 1 radius. the biggest.
        markup = [
          '<radialGradient ',
          commonAttributes,
          ' cx="', needsSwap ? coords.x1 : coords.x2,
          '" cy="', needsSwap ? coords.y1 : coords.y2,
          '" r="', needsSwap ? coords.r1 : coords.r2,
          '" fx="', needsSwap ? coords.x2 : coords.x1,
          '" fy="', needsSwap ? coords.y2 : coords.y1,
          '">\n'
        ];
      }

      if (this.type === 'radial') {
```

```
        if (needsSwap) {
          // svg goes from internal to external radius. if radius are inverted,
swap color stops.
          colorStops = colorStops.concat();
          colorStops.reverse();
          for (var i = 0; i < colorStops.length; i++) {
            colorStops[i].offset = 1 - colorStops[i].offset;
          }
        }
        var minRadius = Math.min(coords.r1, coords.r2);
        if (minRadius > 0) {
          // i have to shift all colorStops and add new one in 0.
          var maxRadius = Math.max(coords.r1, coords.r2),
              percentageShift = minRadius / maxRadius;
          for (var i = 0; i < colorStops.length; i++) {
            colorStops[i].offset += percentageShift * (1 -
colorStops[i].offset);
          }
        }
      }

      for (var i = 0; i < colorStops.length; i++) {
        var colorStop = colorStops[i];
        markup.push(
          '<stop ',
            'offset="', (colorStop.offset * 100) + '%',
            '" style="stop-color:', colorStop.color,
            (colorStop.opacity !== null ? ';stop-opacity: ' +
colorStop.opacity : ';'),
          '"/>\n'
        );
      }

      markup.push((this.type === 'linear' ? '</linearGradient>\n' :
'</radialGradient>\n'));

      return markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns an instance of CanvasGradient
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Object} object
     * @return {CanvasGradient}
     */
    toLive: function(ctx, object) {
      var gradient, prop, coords = fabric.util.object.clone(this.coords);

      if (!this.type) {
        return;
      }

      if (object.group && object.group.type === 'path-group') {
        for (prop in coords) {
          if (prop === 'x1' || prop === 'x2') {
            coords[prop] += -this.offsetX + object.width / 2;
          }
          else if (prop === 'y1' || prop === 'y2') {
            coords[prop] += -this.offsetY + object.height / 2;
          }
        }
      }
```

```
    if (this.type === 'linear') {
      gradient = ctx.createLinearGradient(
        coords.x1, coords.y1, coords.x2, coords.y2);
    }
    else if (this.type === 'radial') {
      gradient = ctx.createRadialGradient(
        coords.x1, coords.y1, coords.r1, coords.x2, coords.y2, coords.r2);
    }

    for (var i = 0, len = this.colorStops.length; i < len; i++) {
      var color = this.colorStops[i].color,
          opacity = this.colorStops[i].opacity,
          offset = this.colorStops[i].offset;

      if (typeof opacity !== 'undefined') {
        color = new fabric.Color(color).setAlpha(opacity).toRgba();
      }
      gradient.addColorStop(offset, color);
    }

    return gradient;
  }
});

fabric.util.object.extend(fabric.Gradient, {

  /* _FROM_SVG_START_ */
  /**
   * Returns {@link fabric.Gradient} instance from an SVG element
   * @static
   * @memberOf fabric.Gradient
   * @param {SVGGradientElement} el SVG gradient element
   * @param {fabric.Object} instance
   * @return {fabric.Gradient} Gradient instance
   * @see http://www.w3.org/TR/SVG/pservers.html#LinearGradientElement
   * @see http://www.w3.org/TR/SVG/pservers.html#RadialGradientElement
   */
  fromElement: function(el, instance) {

    /**
     *  @example:
     *
     *  <linearGradient id="linearGrad1">
     *    <stop offset="0%" stop-color="white"/>
     *    <stop offset="100%" stop-color="black"/>
     *  </linearGradient>
     *
     *  OR
     *
     *  <linearGradient id="linearGrad2">
     *    <stop offset="0" style="stop-color:rgb(255,255,255)"/>
     *    <stop offset="1" style="stop-color:rgb(0,0,0)"/>
     *  </linearGradient>
     *
     *  OR
     *
     *  <radialGradient id="radialGrad1">
     *    <stop offset="0%" stop-color="white" stop-opacity="1" />
     *    <stop offset="50%" stop-color="black" stop-opacity="0.5" />
     *    <stop offset="100%" stop-color="white" stop-opacity="1" />
     *  </radialGradient>
     *
     *  OR
     *
```

```
       *  <radialGradient id="radialGrad2">
       *    <stop offset="0" stop-color="rgb(255,255,255)" />
       *    <stop offset="0.5" stop-color="rgb(0,0,0)" />
       *    <stop offset="1" stop-color="rgb(255,255,255)" />
       *  </radialGradient>
       *
       */

      var colorStopEls = el.getElementsByTagName('stop'),
          type,
          gradientUnits = el.getAttribute('gradientUnits') ||
'objectBoundingBox',
          gradientTransform = el.getAttribute('gradientTransform'),
          colorStops = [],
          coords, ellipseMatrix;

      if (el.nodeName === 'linearGradient' || el.nodeName === 'LINEARGRADIENT')
{
        type = 'linear';
      }
      else {
        type = 'radial';
      }

      if (type === 'linear') {
        coords = getLinearCoords(el);
      }
      else if (type === 'radial') {
        coords = getRadialCoords(el);
      }

      for (var i = colorStopEls.length; i--; ) {
        colorStops.push(getColorStop(colorStopEls[i]));
      }

      ellipseMatrix = _convertPercentUnitsToValues(instance, coords,
gradientUnits);

      var gradient = new fabric.Gradient({
        type: type,
        coords: coords,
        colorStops: colorStops,
        offsetX: -instance.left,
        offsetY: -instance.top
      });

      if (gradientTransform || ellipseMatrix !== '') {
        gradient.gradientTransform =
fabric.parseTransformAttribute((gradientTransform || '') + ellipseMatrix);
      }
      return gradient;
    },
    /* _FROM_SVG_END_ */

    /**
     * Returns {@link fabric.Gradient} instance from its object representation
     * @static
     * @memberOf fabric.Gradient
     * @param {Object} obj
     * @param {Object} [options] Options object
     */
    forObject: function(obj, options) {
      options || (options = { });
      _convertPercentUnitsToValues(obj, options.coords, 'userSpaceOnUse');
```

```
      return new fabric.Gradient(options);
    }
  });

  /**
   * @private
   */
  function _convertPercentUnitsToValues(object, options, gradientUnits) {
    var propValue, addFactor = 0, multFactor = 1, ellipseMatrix = '';
    for (var prop in options) {
      if (options[prop] === 'Infinity') {
        options[prop] = 1;
      }
      else if (options[prop] === '-Infinity') {
        options[prop] = 0;
      }
      propValue = parseFloat(options[prop], 10);
      if (typeof options[prop] === 'string' && /^\d+%$/.test(options[prop])) {
        multFactor = 0.01;
      }
      else {
        multFactor = 1;
      }
      if (prop === 'x1' || prop === 'x2' || prop === 'r2') {
        multFactor *= gradientUnits === 'objectBoundingBox' ? object.width : 1;
        addFactor = gradientUnits === 'objectBoundingBox' ? object.left || 0 :
0;
      }
      else if (prop === 'y1' || prop === 'y2') {
        multFactor *= gradientUnits === 'objectBoundingBox' ? object.height : 1;
        addFactor = gradientUnits === 'objectBoundingBox' ? object.top || 0 : 0;
      }
      options[prop] = propValue * multFactor + addFactor;
    }
    if (object.type === 'ellipse' &&
        options.r2 !== null &&
        gradientUnits === 'objectBoundingBox' &&
        object.rx !== object.ry) {

      var scaleFactor = object.ry / object.rx;
      ellipseMatrix = ' scale(1, ' + scaleFactor + ')';
      if (options.y1) {
        options.y1 /= scaleFactor;
      }
      if (options.y2) {
        options.y2 /= scaleFactor;
      }
    }
    return ellipseMatrix;
  }
})();


(function() {

  'use strict';

  var toFixed = fabric.util.toFixed;

  /**
   * Pattern class
   * @class fabric.Pattern
   * @see {@link http://fabricjs.com/patterns|Pattern demo}
   * @see {@link http://fabricjs.com/dynamic-patterns|DynamicPattern demo}
```

```
   * @see {@link fabric.Pattern#initialize} for constructor definition
   */


  fabric.Pattern = fabric.util.createClass(/** @lends fabric.Pattern.prototype
*/ {

    /**
     * Repeat property of a pattern (one of repeat, repeat-x, repeat-y or no-
repeat)
     * @type String
     * @default
     */
    repeat: 'repeat',

    /**
     * Pattern horizontal offset from object's left/top corner
     * @type Number
     * @default
     */
    offsetX: 0,

    /**
     * Pattern vertical offset from object's left/top corner
     * @type Number
     * @default
     */
    offsetY: 0,

    /**
     * Constructor
     * @param {Object} [options] Options object
     * @param {Function} [callback] function to invoke after callback init.
     * @return {fabric.Pattern} thisArg
     */
    initialize: function(options, callback) {
      options || (options = { });

      this.id = fabric.Object.__uid++;
      this.setOptions(options);
      if (!options.source || (options.source && typeof options.source !==
'string')) {
        callback && callback(this);
        return;
      }
      // function string
      if (typeof fabric.util.getFunctionBody(options.source) !== 'undefined') {
        this.source = new Function(fabric.util.getFunctionBody(options.source));
        callback && callback(this);
      }
      else {
        // img src string
        var _this = this;
        this.source = fabric.util.createImage();
        fabric.util.loadImage(options.source, function(img) {
          _this.source = img;
          callback && callback(_this);
        });
      }
    },

    /**
     * Returns object representation of a pattern
     * @param {Array} [propertiesToInclude] Any properties that you might want
```

```
to additionally include in the output
     * @return {Object} Object representation of a pattern instance
     */
    toObject: function(propertiesToInclude) {
      var NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS,
          source, object;

      // callback
      if (typeof this.source === 'function') {
        source = String(this.source);
      }
      // <img> element
      else if (typeof this.source.src === 'string') {
        source = this.source.src;
      }
      // <canvas> element
      else if (typeof this.source === 'object' && this.source.toDataURL) {
        source = this.source.toDataURL();
      }

      object = {
        type: 'pattern',
        source: source,
        repeat: this.repeat,
        offsetX: toFixed(this.offsetX, NUM_FRACTION_DIGITS),
        offsetY: toFixed(this.offsetY, NUM_FRACTION_DIGITS),
      };
      fabric.util.populateWithProperties(this, object, propertiesToInclude);

      return object;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of a pattern
     * @param {fabric.Object} object
     * @return {String} SVG representation of a pattern
     */
    toSVG: function(object) {
      var patternSource = typeof this.source === 'function' ? this.source() :
this.source,
          patternWidth = patternSource.width / object.width,
          patternHeight = patternSource.height / object.height,
          patternOffsetX = this.offsetX / object.width,
          patternOffsetY = this.offsetY / object.height,
          patternImgSrc = '';
      if (this.repeat === 'repeat-x' || this.repeat === 'no-repeat') {
        patternHeight = 1;
      }
      if (this.repeat === 'repeat-y' || this.repeat === 'no-repeat') {
        patternWidth = 1;
      }
      if (patternSource.src) {
        patternImgSrc = patternSource.src;
      }
      else if (patternSource.toDataURL) {
        patternImgSrc = patternSource.toDataURL();
      }

      return '<pattern id="SVGID_' + this.id +
                    '" x="' + patternOffsetX +
                    '" y="' + patternOffsetY +
                    '" width="' + patternWidth +
                    '" height="' + patternHeight + '">\n' +
```

```
              '<image x="0" y="0"' +
                     ' width="' + patternSource.width +
                     '" height="' + patternSource.height +
                     '" xlink:href="' + patternImgSrc +
              '"></image>\n' +
            '</pattern>\n';
    },
    /* _TO_SVG_END_ */

    setOptions: function(options) {
      for (var prop in options) {
        this[prop] = options[prop];
      }
    },

    /**
     * Returns an instance of CanvasPattern
     * @param {CanvasRenderingContext2D} ctx Context to create pattern
     * @return {CanvasPattern}
     */
    toLive: function(ctx) {
      var source = typeof this.source === 'function' ? this.source() :
this.source;

      // if the image failed to load, return, and allow rest to continue loading
      if (!source) {
        return '';
      }

      // if an image
      if (typeof source.src !== 'undefined') {
        if (!source.complete) {
          return '';
        }
        if (source.naturalWidth === 0 || source.naturalHeight === 0) {
          return '';
        }
      }
      return ctx.createPattern(source, this.repeat);
    }
  });
})();


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      toFixed = fabric.util.toFixed;

  if (fabric.Shadow) {
    fabric.warn('fabric.Shadow is already defined.');
    return;
  }

  /**
   * Shadow class
   * @class fabric.Shadow
   * @see {@link http://fabricjs.com/shadows|Shadow demo}
   * @see {@link fabric.Shadow#initialize} for constructor definition
   */
  fabric.Shadow = fabric.util.createClass(/** @lends fabric.Shadow.prototype */
{
```

```
    /**
     * Shadow color
     * @type String
     * @default
     */
    color: 'rgb(0,0,0)',

    /**
     * Shadow blur
     * @type Number
     */
    blur: 0,

    /**
     * Shadow horizontal offset
     * @type Number
     * @default
     */
    offsetX: 0,

    /**
     * Shadow vertical offset
     * @type Number
     * @default
     */
    offsetY: 0,

    /**
     * Whether the shadow should affect stroke operations
     * @type Boolean
     * @default
     */
    affectStroke: false,

    /**
     * Indicates whether toObject should include default values
     * @type Boolean
     * @default
     */
    includeDefaultValues: true,

    /**
     * Constructor
     * @param {Object|String} [options] Options object with any of color, blur,
offsetX, offsetX properties or string (e.g. "rgba(0,0,0,0.2) 2px 2px 10px, "2px
2px 10px rgba(0,0,0,0.2)")
     * @return {fabric.Shadow} thisArg
     */
    initialize: function(options) {

      if (typeof options === 'string') {
        options = this._parseShadow(options);
      }

      for (var prop in options) {
        this[prop] = options[prop];
      }

      this.id = fabric.Object.__uid++;
    },

    /**
     * @private
```

```
     * @param {String} shadow Shadow value to parse
     * @return {Object} Shadow object with color, offsetX, offsetY and blur
     */
    _parseShadow: function(shadow) {
      var shadowStr = shadow.trim(),
          offsetsAndBlur = fabric.Shadow.reOffsetsAndBlur.exec(shadowStr) || [],
          color = shadowStr.replace(fabric.Shadow.reOffsetsAndBlur, '') ||
'rgb(0,0,0)';

      return {
        color: color.trim(),
        offsetX: parseInt(offsetsAndBlur[1], 10) || 0,
        offsetY: parseInt(offsetsAndBlur[2], 10) || 0,
        blur: parseInt(offsetsAndBlur[3], 10) || 0
      };
    },

    /**
     * Returns a string representation of an instance
     * @see http://www.w3.org/TR/css-text-decor-3/#text-shadow
     * @return {String} Returns CSS3 text-shadow declaration
     */
    toString: function() {
      return [this.offsetX, this.offsetY, this.blur, this.color].join('px ');
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of a shadow
     * @param {fabric.Object} object
     * @return {String} SVG representation of a shadow
     */
    toSVG: function(object) {
      var fBoxX = 40, fBoxY = 40, NUM_FRACTION_DIGITS =
fabric.Object.NUM_FRACTION_DIGITS,
          offset = fabric.util.rotateVector(
            { x: this.offsetX, y: this.offsetY },
            fabric.util.degreesToRadians(-object.angle)),
          BLUR_BOX = 20;

      if (object.width && object.height) {
        //http://www.w3.org/TR/SVG/filters.html#FilterEffectsRegion
        // we add some extra space to filter box to contain the blur ( 20 )
        fBoxX = toFixed((Math.abs(offset.x) + this.blur) / object.width,
NUM_FRACTION_DIGITS) * 100 + BLUR_BOX;
        fBoxY = toFixed((Math.abs(offset.y) + this.blur) / object.height,
NUM_FRACTION_DIGITS) * 100 + BLUR_BOX;
      }
      if (object.flipX) {
        offset.x *= -1;
      }
      if (object.flipY) {
        offset.y *= -1;
      }
      return (
        '<filter id="SVGID_' + this.id + '" y="-' + fBoxY + '%" height="' + (100
+ 2 * fBoxY) + '%" ' +
          'x="-' + fBoxX + '%" width="' + (100 + 2 * fBoxX) + '%" ' + '>\n' +
          '\t<feGaussianBlur in="SourceAlpha" stdDeviation="' +
            toFixed(this.blur ? this.blur / 2 : 0, NUM_FRACTION_DIGITS) +
'"></feGaussianBlur>\n' +
          '\t<feOffset dx="' + toFixed(offset.x, NUM_FRACTION_DIGITS) +
          '" dy="' + toFixed(offset.y, NUM_FRACTION_DIGITS) + '" result="oBlur"
></feOffset>\n' +
```

```
          '\t<feFlood flood-color="' + this.color + '"/>\n' +
          '\t<feComposite in2="oBlur" operator="in" />\n' +
          '\t<feMerge>\n' +
            '\t\t<feMergeNode></feMergeNode>\n' +
            '\t\t<feMergeNode in="SourceGraphic"></feMergeNode>\n' +
          '\t</feMerge>\n' +
        '</filter>\n');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns object representation of a shadow
     * @return {Object} Object representation of a shadow instance
     */
    toObject: function() {
      if (this.includeDefaultValues) {
        return {
          color: this.color,
          blur: this.blur,
          offsetX: this.offsetX,
          offsetY: this.offsetY,
          affectStroke: this.affectStroke
        };
      }
      var obj = { }, proto = fabric.Shadow.prototype;

      ['color', 'blur', 'offsetX', 'offsetY',
'affectStroke'].forEach(function(prop) {
        if (this[prop] !== proto[prop]) {
          obj[prop] = this[prop];
        }
      }, this);

      return obj;
    }
  });

  /**
   * Regex matching shadow offsetX, offsetY and blur (ex: "2px 2px 10px
rgba(0,0,0,0.2)", "rgb(0,255,0) 2px 2px")
   * @static
   * @field
   * @memberOf fabric.Shadow
   */
  // eslint-disable-next-line max-len
  fabric.Shadow.reOffsetsAndBlur = /(?:\s|^)(-?\d+(?:px)?(?:\s?|$))?(-?\d+
(?:px)?(?:\s?|$))?(\d+(?:px)?)?(?:\s?|$)(?:$|\s)/;

})(typeof exports !== 'undefined' ? exports : this);


(function () {

  'use strict';

  if (fabric.StaticCanvas) {
    fabric.warn('fabric.StaticCanvas is already defined.');
    return;
  }

  // aliases for faster resolution
  var extend = fabric.util.object.extend,
      getElementOffset = fabric.util.getElementOffset,
      removeFromArray = fabric.util.removeFromArray,
```

```
      toFixed = fabric.util.toFixed,
      transformPoint = fabric.util.transformPoint,
      invertTransform = fabric.util.invertTransform,

      CANVAS_INIT_ERROR = new Error('Could not initialize `canvas` element');

  /**
   * Static canvas class
   * @class fabric.StaticCanvas
   * @mixes fabric.Collection
   * @mixes fabric.Observable
   * @see {@link http://fabricjs.com/static_canvas|StaticCanvas demo}
   * @see {@link fabric.StaticCanvas#initialize} for constructor definition
   * @fires before:render
   * @fires after:render
   * @fires canvas:cleared
   * @fires object:added
   * @fires object:removed
   */
  fabric.StaticCanvas = fabric.util.createClass(fabric.CommonMethods, /** @lends
fabric.StaticCanvas.prototype */ {

    /**
     * Constructor
     * @param {HTMLElement | String} el &lt;canvas> element to initialize
instance on
     * @param {Object} [options] Options object
     * @return {Object} thisArg
     */
    initialize: function(el, options) {
      options || (options = { });

      this._initStatic(el, options);
    },

    /**
     * Background color of canvas instance.
     * Should be set via {@link fabric.StaticCanvas#setBackgroundColor}.
     * @type {(String|fabric.Pattern)}
     * @default
     */
    backgroundColor: '',

    /**
     * Background image of canvas instance.
     * Should be set via {@link fabric.StaticCanvas#setBackgroundImage}.
     * <b>Backwards incompatibility note:</b> The "backgroundImageOpacity"
     * and "backgroundImageStretch" properties are deprecated since 1.3.9.
     * Use {@link fabric.Image#opacity}, {@link fabric.Image#width} and {@link
fabric.Image#height}.
     * @type fabric.Image
     * @default
     */
    backgroundImage: null,

    /**
     * Overlay color of canvas instance.
     * Should be set via {@link fabric.StaticCanvas#setOverlayColor}
     * @since 1.3.9
     * @type {(String|fabric.Pattern)}
     * @default
     */
    overlayColor: '',
```

```
    /**
     * Overlay image of canvas instance.
     * Should be set via {@link fabric.StaticCanvas#setOverlayImage}.
     * <b>Backwards incompatibility note:</b> The "overlayImageLeft"
     * and "overlayImageTop" properties are deprecated since 1.3.9.
     * Use {@link fabric.Image#left} and {@link fabric.Image#top}.
     * @type fabric.Image
     * @default
     */
    overlayImage: null,

    /**
     * Indicates whether toObject/toDatalessObject should include default values
     * @type Boolean
     * @default
     */
    includeDefaultValues: true,

    /**
     * Indicates whether objects' state should be saved
     * @type Boolean
     * @default
     */
    stateful: false,

    /**
     * Indicates whether {@link fabric.Collection.add}, {@link
fabric.Collection.insertAt} and {@link fabric.Collection.remove} should also re-
render canvas.
     * Disabling this option could give a great performance boost when
adding/removing a lot of objects to/from canvas at once
     * (followed by a manual rendering after addition/deletion)
     * @type Boolean
     * @default
     */
    renderOnAddRemove: true,

    /**
     * Function that determines clipping of entire canvas area
     * Being passed context as first argument. See clipping canvas area in
{@link https://github.com/kangax/fabric.js/wiki/FAQ}
     * @type Function
     * @default
     */
    clipTo: null,

    /**
     * Indicates whether object controls (borders/controls) are rendered above
overlay image
     * @type Boolean
     * @default
     */
    controlsAboveOverlay: false,

    /**
     * Indicates whether the browser can be scrolled when using a touchscreen
and dragging on the canvas
     * @type Boolean
     * @default
     */
    allowTouchScrolling: false,

    /**
     * Indicates whether this canvas will use image smoothing, this is on by
```

```
default in browsers
     * @type Boolean
     * @default
     */
    imageSmoothingEnabled: true,

    /**
     * The transformation (in the format of Canvas transform) which focuses the
viewport
     * @type Array
     * @default
     */
    viewportTransform: fabric.iMatrix.concat(),

    /**
     * if set to false background image is not affected by viewport transform
     * @since 1.6.3
     * @type Boolean
     * @default
     */
    backgroundVpt: true,

    /**
     * if set to false overlya image is not affected by viewport transform
     * @since 1.6.3
     * @type Boolean
     * @default
     */
    overlayVpt: true,

    /**
     * Callback; invoked right before object is about to be scaled/rotated
     */
    onBeforeScaleRotate: function () {
      /* NOOP */
    },

    /**
     * When true, canvas is scaled by devicePixelRatio for better rendering on
retina screens
     */
    enableRetinaScaling: true,

    /**
     * Describe canvas element extension over design
     * properties are tl,tr,bl,br.
     * if canvas is not zoomed/panned those points are the four corner of canvas
     * if canvas is viewportTransformed you those points indicate the extension
     * of canvas element in plain untrasformed coordinates
     * The coordinates get updated with @method calcViewportBoundaries.
     * @memberOf fabric.StaticCanvas.prototype
     */
    vptCoords: { },

    /**
     * Based on vptCoords and object.aCoords, skip rendering of objects that
     * are not included in current viewport.
     * May greatly help in applications with crowded canvas and use of zoom/pan
     * If One of the corner of the bounding box of the object is on the canvas
     * the objects get rendered.
     * @memberOf fabric.StaticCanvas.prototype
     */
    skipOffscreen: false,
```

```
    /**
     * @private
     * @param {HTMLElement | String} el &lt;canvas> element to initialize
instance on
     * @param {Object} [options] Options object
     */
    _initStatic: function(el, options) {
      var cb = fabric.StaticCanvas.prototype.renderAll.bind(this);
      this._objects = [];
      this._createLowerCanvas(el);
      this._initOptions(options);
      this._setImageSmoothing();
      // only initialize retina scaling once
      if (!this.interactive) {
        this._initRetinaScaling();
      }

      if (options.overlayImage) {
        this.setOverlayImage(options.overlayImage, cb);
      }
      if (options.backgroundImage) {
        this.setBackgroundImage(options.backgroundImage, cb);
      }
      if (options.backgroundColor) {
        this.setBackgroundColor(options.backgroundColor, cb);
      }
      if (options.overlayColor) {
        this.setOverlayColor(options.overlayColor, cb);
      }
      this.calcOffset();
    },

    /**
     * @private
     */
    _isRetinaScaling: function() {
      return (fabric.devicePixelRatio !== 1 && this.enableRetinaScaling);
    },

    /**
     * @private
     * @return {Number} retinaScaling if applied, otherwise 1;
     */
    getRetinaScaling: function() {
      return this._isRetinaScaling() ? fabric.devicePixelRatio : 1;
    },

    /**
     * @private
     */
    _initRetinaScaling: function() {
      if (!this._isRetinaScaling()) {
        return;
      }
      this.lowerCanvasEl.setAttribute('width', this.width *
fabric.devicePixelRatio);
      this.lowerCanvasEl.setAttribute('height', this.height *
fabric.devicePixelRatio);

      this.contextContainer.scale(fabric.devicePixelRatio,
fabric.devicePixelRatio);
    },

    /**
```

```
     * Calculates canvas element offset relative to the document
     * This method is also attached as "resize" event handler of window
     * @return {fabric.Canvas} instance
     * @chainable
     */
    calcOffset: function () {
      this._offset = getElementOffset(this.lowerCanvasEl);
      return this;
    },

    /**
     * Sets {@link fabric.StaticCanvas#overlayImage|overlay image} for this
canvas
     * @param {(fabric.Image|String)} image fabric.Image instance or URL of an
image to set overlay to
     * @param {Function} callback callback to invoke when image is loaded and
set as an overlay
     * @param {Object} [options] Optional options to set for the {@link
fabric.Image|overlay image}.
     * @return {fabric.Canvas} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/MnzHT/|jsFiddle demo}
     * @example <caption>Normal overlayImage with left/top = 0</caption>
     * canvas.setOverlayImage('http://fabricjs.com/assets/jail_cell_bars.png',
canvas.renderAll.bind(canvas), {
     *   // Needed to position overlayImage at 0/0
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>overlayImage with different properties</caption>
     * canvas.setOverlayImage('http://fabricjs.com/assets/jail_cell_bars.png',
canvas.renderAll.bind(canvas), {
     *   opacity: 0.5,
     *   angle: 45,
     *   left: 400,
     *   top: 400,
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>Stretched overlayImage #1 - width/height correspond to
canvas width/height</caption>
     * fabric.Image.fromURL('http://fabricjs.com/assets/jail_cell_bars.png',
function(img) {
     *    img.set({width: canvas.width, height: canvas.height, originX: 'left',
originY: 'top'});
     *    canvas.setOverlayImage(img, canvas.renderAll.bind(canvas));
     * });
     * @example <caption>Stretched overlayImage #2 - width/height correspond to
canvas width/height</caption>
     * canvas.setOverlayImage('http://fabricjs.com/assets/jail_cell_bars.png',
canvas.renderAll.bind(canvas), {
     *   width: canvas.width,
     *   height: canvas.height,
     *   // Needed to position overlayImage at 0/0
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>overlayImage loaded from cross-origin</caption>
     * canvas.setOverlayImage('http://fabricjs.com/assets/jail_cell_bars.png',
canvas.renderAll.bind(canvas), {
     *   opacity: 0.5,
     *   angle: 45,
     *   left: 400,
     *   top: 400,
```

```
     *   originX: 'left',
     *   originY: 'top',
     *   crossOrigin: 'anonymous'
     * });
     */
    setOverlayImage: function (image, callback, options) {
      return this.__setBgOverlayImage('overlayImage', image, callback, options);
    },

    /**
     * Sets {@link fabric.StaticCanvas#backgroundImage|background image} for
this canvas
     * @param {(fabric.Image|String)} image fabric.Image instance or URL of an
image to set background to
     * @param {Function} callback Callback to invoke when image is loaded and
set as background
     * @param {Object} [options] Optional options to set for the {@link
fabric.Image|background image}.
     * @return {fabric.Canvas} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/YH9yD/|jsFiddle demo}
     * @example <caption>Normal backgroundImage with left/top = 0</caption>
     *
canvas.setBackgroundImage('http://fabricjs.com/assets/honey_im_subtle.png',
canvas.renderAll.bind(canvas), {
     *   // Needed to position backgroundImage at 0/0
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>backgroundImage with different properties</caption>
     *
canvas.setBackgroundImage('http://fabricjs.com/assets/honey_im_subtle.png',
canvas.renderAll.bind(canvas), {
     *   opacity: 0.5,
     *   angle: 45,
     *   left: 400,
     *   top: 400,
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>Stretched backgroundImage #1 - width/height correspond
to canvas width/height</caption>
     * fabric.Image.fromURL('http://fabricjs.com/assets/honey_im_subtle.png',
function(img) {
     *    img.set({width: canvas.width, height: canvas.height, originX: 'left',
originY: 'top'});
     *    canvas.setBackgroundImage(img, canvas.renderAll.bind(canvas));
     * });
     * @example <caption>Stretched backgroundImage #2 - width/height correspond
to canvas width/height</caption>
     *
canvas.setBackgroundImage('http://fabricjs.com/assets/honey_im_subtle.png',
canvas.renderAll.bind(canvas), {
     *   width: canvas.width,
     *   height: canvas.height,
     *   // Needed to position backgroundImage at 0/0
     *   originX: 'left',
     *   originY: 'top'
     * });
     * @example <caption>backgroundImage loaded from cross-origin</caption>
     *
canvas.setBackgroundImage('http://fabricjs.com/assets/honey_im_subtle.png',
canvas.renderAll.bind(canvas), {
     *   opacity: 0.5,
```

```
     *   angle: 45,
     *   left: 400,
     *   top: 400,
     *   originX: 'left',
     *   originY: 'top',
     *   crossOrigin: 'anonymous'
     * });
     */
    setBackgroundImage: function (image, callback, options) {
      return this.__setBgOverlayImage('backgroundImage', image, callback,
options);
    },

    /**
     * Sets {@link fabric.StaticCanvas#overlayColor|background color} for this
canvas
     * @param {(String|fabric.Pattern)} overlayColor Color or pattern to set
background color to
     * @param {Function} callback Callback to invoke when background color is
set
     * @return {fabric.Canvas} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/pB55h/|jsFiddle demo}
     * @example <caption>Normal overlayColor - color value</caption>
     * canvas.setOverlayColor('rgba(255, 73, 64, 0.6)',
canvas.renderAll.bind(canvas));
     * @example <caption>fabric.Pattern used as overlayColor</caption>
     * canvas.setOverlayColor({
     *   source: 'http://fabricjs.com/assets/escheresque_ste.png'
     * }, canvas.renderAll.bind(canvas));
     * @example <caption>fabric.Pattern used as overlayColor with repeat and
offset</caption>
     * canvas.setOverlayColor({
     *   source: 'http://fabricjs.com/assets/escheresque_ste.png',
     *   repeat: 'repeat',
     *   offsetX: 200,
     *   offsetY: 100
     * }, canvas.renderAll.bind(canvas));
     */
    setOverlayColor: function(overlayColor, callback) {
      return this.__setBgOverlayColor('overlayColor', overlayColor, callback);
    },

    /**
     * Sets {@link fabric.StaticCanvas#backgroundColor|background color} for
this canvas
     * @param {(String|fabric.Pattern)} backgroundColor Color or pattern to set
background color to
     * @param {Function} callback Callback to invoke when background color is
set
     * @return {fabric.Canvas} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/hXzvk/|jsFiddle demo}
     * @example <caption>Normal backgroundColor - color value</caption>
     * canvas.setBackgroundColor('rgba(255, 73, 64, 0.6)',
canvas.renderAll.bind(canvas));
     * @example <caption>fabric.Pattern used as backgroundColor</caption>
     * canvas.setBackgroundColor({
     *   source: 'http://fabricjs.com/assets/escheresque_ste.png'
     * }, canvas.renderAll.bind(canvas));
     * @example <caption>fabric.Pattern used as backgroundColor with repeat and
offset</caption>
     * canvas.setBackgroundColor({
     *   source: 'http://fabricjs.com/assets/escheresque_ste.png',
```

```
     *   repeat: 'repeat',
     *   offsetX: 200,
     *   offsetY: 100
     * }, canvas.renderAll.bind(canvas));
     */
    setBackgroundColor: function(backgroundColor, callback) {
      return this.__setBgOverlayColor('backgroundColor', backgroundColor,
callback);
    },

    /**
     * @private
     * @see {@link
http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-
element.html#dom-context-2d-imagesmoothingenabled|WhatWG Canvas Standard}
     */
    _setImageSmoothing: function() {
      var ctx = this.getContext();

      ctx.imageSmoothingEnabled = ctx.imageSmoothingEnabled ||
ctx.webkitImageSmoothingEnabled
        || ctx.mozImageSmoothingEnabled || ctx.msImageSmoothingEnabled ||
ctx.oImageSmoothingEnabled;
      ctx.imageSmoothingEnabled = this.imageSmoothingEnabled;
    },

    /**
     * @private
     * @param {String} property Property to set ({@link
fabric.StaticCanvas#backgroundImage|backgroundImage}
     * or {@link fabric.StaticCanvas#overlayImage|overlayImage})
     * @param {(fabric.Image|String|null)} image fabric.Image instance, URL of
an image or null to set background or overlay to
     * @param {Function} callback Callback to invoke when image is loaded and
set as background or overlay
     * @param {Object} [options] Optional options to set for the {@link
fabric.Image|image}.
     */
    __setBgOverlayImage: function(property, image, callback, options) {
      if (typeof image === 'string') {
        fabric.util.loadImage(image, function(img) {
          img && (this[property] = new fabric.Image(img, options));
          callback && callback(img);
        }, this, options && options.crossOrigin);
      }
      else {
        options && image.setOptions(options);
        this[property] = image;
        callback && callback(image);
      }

      return this;
    },

    /**
     * @private
     * @param {String} property Property to set ({@link
fabric.StaticCanvas#backgroundColor|backgroundColor}
     * or {@link fabric.StaticCanvas#overlayColor|overlayColor})
     * @param {(Object|String|null)} color Object with pattern information,
color value or null
     * @param {Function} [callback] Callback is invoked when color is set
     */
    __setBgOverlayColor: function(property, color, callback) {
```

```
      this[property] = color;
      this._initGradient(color, property);
      this._initPattern(color, property, callback);
      return this;
    },

    /**
     * @private
     */
    _createCanvasElement: function(canvasEl) {
      var element = fabric.util.createCanvasElement(canvasEl);
      if (!element.style) {
        element.style = { };
      }
      if (!element) {
        throw CANVAS_INIT_ERROR;
      }
      if (typeof element.getContext === 'undefined') {
        throw CANVAS_INIT_ERROR;
      }
      return element;
    },

    /**
     * @private
     * @param {Object} [options] Options object
     */
    _initOptions: function (options) {
      this._setOptions(options);

      this.width = this.width || parseInt(this.lowerCanvasEl.width, 10) || 0;
      this.height = this.height || parseInt(this.lowerCanvasEl.height, 10) || 0;

      if (!this.lowerCanvasEl.style) {
        return;
      }

      this.lowerCanvasEl.width = this.width;
      this.lowerCanvasEl.height = this.height;

      this.lowerCanvasEl.style.width = this.width + 'px';
      this.lowerCanvasEl.style.height = this.height + 'px';

      this.viewportTransform = this.viewportTransform.slice();
    },

    /**
     * Creates a bottom canvas
     * @private
     * @param {HTMLElement} [canvasEl]
     */
    _createLowerCanvas: function (canvasEl) {
      this.lowerCanvasEl = fabric.util.getById(canvasEl) ||
this._createCanvasElement(canvasEl);

      fabric.util.addClass(this.lowerCanvasEl, 'lower-canvas');

      if (this.interactive) {
        this._applyCanvasStyle(this.lowerCanvasEl);
      }

      this.contextContainer = this.lowerCanvasEl.getContext('2d');
    },
```

```
    /**
     * Returns canvas width (in px)
     * @return {Number}
     */
    getWidth: function () {
      return this.width;
    },

    /**
     * Returns canvas height (in px)
     * @return {Number}
     */
    getHeight: function () {
      return this.height;
    },

    /**
     * Sets width of this canvas instance
     * @param {Number|String} value                          Value to set width
to
     * @param {Object}        [options]                      Options object
     * @param {Boolean}       [options.backstoreOnly=false] Set the given
dimensions only as canvas backstore dimensions
     * @param {Boolean}       [options.cssOnly=false]       Set the given
dimensions only as css dimensions
     * @return {fabric.Canvas} instance
     * @chainable true
     */
    setWidth: function (value, options) {
      return this.setDimensions({ width: value }, options);
    },

    /**
     * Sets height of this canvas instance
     * @param {Number|String} value                          Value to set height
to
     * @param {Object}        [options]                      Options object
     * @param {Boolean}       [options.backstoreOnly=false] Set the given
dimensions only as canvas backstore dimensions
     * @param {Boolean}       [options.cssOnly=false]       Set the given
dimensions only as css dimensions
     * @return {fabric.Canvas} instance
     * @chainable true
     */
    setHeight: function (value, options) {
      return this.setDimensions({ height: value }, options);
    },

    /**
     * Sets dimensions (width, height) of this canvas instance. when
options.cssOnly flag active you should also supply the unit of measure (px/%/em)
     * @param {Object}        dimensions                     Object with
width/height properties
     * @param {Number|String} [dimensions.width]            Width of canvas
element
     * @param {Number|String} [dimensions.height]           Height of canvas
element
     * @param {Object}        [options]                      Options object
     * @param {Boolean}       [options.backstoreOnly=false] Set the given
dimensions only as canvas backstore dimensions
     * @param {Boolean}       [options.cssOnly=false]       Set the given
dimensions only as css dimensions
     * @return {fabric.Canvas} thisArg
     * @chainable
```

```
 */
setDimensions: function (dimensions, options) {
  var cssValue;

  options = options || {};

  for (var prop in dimensions) {
    cssValue = dimensions[prop];

    if (!options.cssOnly) {
      this._setBackstoreDimension(prop, dimensions[prop]);
      cssValue += 'px';
    }

    if (!options.backstoreOnly) {
      this._setCssDimension(prop, cssValue);
    }
  }
  this._initRetinaScaling();
  this._setImageSmoothing();
  this.calcOffset();

  if (!options.cssOnly) {
    this.renderAll();
  }

  return this;
},

/**
 * Helper for setting width/height
 * @private
 * @param {String} prop property (width|height)
 * @param {Number} value value to set property to
 * @return {fabric.Canvas} instance
 * @chainable true
 */
_setBackstoreDimension: function (prop, value) {
  this.lowerCanvasEl[prop] = value;

  if (this.upperCanvasEl) {
    this.upperCanvasEl[prop] = value;
  }

  if (this.cacheCanvasEl) {
    this.cacheCanvasEl[prop] = value;
  }

  this[prop] = value;

  return this;
},

/**
 * Helper for setting css width/height
 * @private
 * @param {String} prop property (width|height)
 * @param {String} value value to set property to
 * @return {fabric.Canvas} instance
 * @chainable true
 */
_setCssDimension: function (prop, value) {
  this.lowerCanvasEl.style[prop] = value;
```

```
      if (this.upperCanvasEl) {
        this.upperCanvasEl.style[prop] = value;
      }

      if (this.wrapperEl) {
        this.wrapperEl.style[prop] = value;
      }

      return this;
    },

    /**
     * Returns canvas zoom level
     * @return {Number}
     */
    getZoom: function () {
      return this.viewportTransform[0];
    },

    /**
     * Sets viewport transform of this canvas instance
     * @param {Array} vpt the transform in the form of context.transform
     * @return {fabric.Canvas} instance
     * @chainable true
     */
    setViewportTransform: function (vpt) {
      var activeGroup = this._activeGroup, object, ignoreVpt = false,
skipAbsolute = true;
      this.viewportTransform = vpt;
      for (var i = 0, len = this._objects.length; i < len; i++) {
        object = this._objects[i];
        object.group || object.setCoords(ignoreVpt, skipAbsolute);
      }
      if (activeGroup) {
        activeGroup.setCoords(ignoreVpt, skipAbsolute);
      }
      this.calcViewportBoundaries();
      this.renderAll();
      return this;
    },

    /**
     * Sets zoom level of this canvas instance, zoom centered around point
     * @param {fabric.Point} point to zoom with respect to
     * @param {Number} value to set zoom to, less than 1 zooms out
     * @return {fabric.Canvas} instance
     * @chainable true
     */
    zoomToPoint: function (point, value) {
      // TODO: just change the scale, preserve other transformations
      var before = point, vpt = this.viewportTransform.slice(0);
      point = transformPoint(point, invertTransform(this.viewportTransform));
      vpt[0] = value;
      vpt[3] = value;
      var after = transformPoint(point, vpt);
      vpt[4] += before.x - after.x;
      vpt[5] += before.y - after.y;
      return this.setViewportTransform(vpt);
    },

    /**
     * Sets zoom level of this canvas instance
     * @param {Number} value to set zoom to, less than 1 zooms out
     * @return {fabric.Canvas} instance
```

```
 * @chainable true
 */
setZoom: function (value) {
  this.zoomToPoint(new fabric.Point(0, 0), value);
  return this;
},

/**
 * Pan viewport so as to place point at top left corner of canvas
 * @param {fabric.Point} point to move to
 * @return {fabric.Canvas} instance
 * @chainable true
 */
absolutePan: function (point) {
  var vpt = this.viewportTransform.slice(0);
  vpt[4] = -point.x;
  vpt[5] = -point.y;
  return this.setViewportTransform(vpt);
},

/**
 * Pans viewpoint relatively
 * @param {fabric.Point} point (position vector) to move by
 * @return {fabric.Canvas} instance
 * @chainable true
 */
relativePan: function (point) {
  return this.absolutePan(new fabric.Point(
    -point.x - this.viewportTransform[4],
    -point.y - this.viewportTransform[5]
  ));
},

/**
 * Returns &lt;canvas> element corresponding to this instance
 * @return {HTMLCanvasElement}
 */
getElement: function () {
  return this.lowerCanvasEl;
},

/**
 * @private
 * @param {fabric.Object} obj Object that was added
 */
_onObjectAdded: function(obj) {
  this.stateful && obj.setupState();
  obj._set('canvas', this);
  obj.setCoords();
  this.fire('object:added', { target: obj });
  obj.fire('added');
},

/**
 * @private
 * @param {fabric.Object} obj Object that was removed
 */
_onObjectRemoved: function(obj) {
  this.fire('object:removed', { target: obj });
  obj.fire('removed');
  delete obj.canvas;
},

/**
```

```
     * Clears specified context of canvas element
     * @param {CanvasRenderingContext2D} ctx Context to clear
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    clearContext: function(ctx) {
      ctx.clearRect(0, 0, this.width, this.height);
      return this;
    },

    /**
     * Returns context of canvas where objects are drawn
     * @return {CanvasRenderingContext2D}
     */
    getContext: function () {
      return this.contextContainer;
    },

    /**
     * Clears all contexts (background, main, top) of an instance
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    clear: function () {
      this._objects.length = 0;
      this.backgroundImage = null;
      this.overlayImage = null;
      this.backgroundColor = '';
      this.overlayColor = '';
      if (this._hasITextHandlers) {
        this.off('mouse:up', this._mouseUpITextHandler);
        this._iTextInstances = null;
        this._hasITextHandlers = false;
      }
      this.clearContext(this.contextContainer);
      this.fire('canvas:cleared');
      this.renderAll();
      return this;
    },

    /**
     * Renders the canvas
     * @return {fabric.Canvas} instance
     * @chainable
     */
    renderAll: function () {
      var canvasToDrawOn = this.contextContainer;
      this.renderCanvas(canvasToDrawOn, this._objects);
      return this;
    },

    /**
     * Calculate the position of the 4 corner of canvas with current
viewportTransform.
     * helps to determinate when an object is in the current rendering viewport
using
     * object absolute coordinates ( aCoords )
     * @return {Object} points.tl
     * @chainable
     */
    calcViewportBoundaries: function() {
      var points = { }, width = this.getWidth(), height = this.getHeight(),
          iVpt = invertTransform(this.viewportTransform);
      points.tl = transformPoint({ x: 0, y: 0 }, iVpt);
```

```
  points.br = transformPoint({ x: width, y: height }, iVpt);
  points.tr = new fabric.Point(points.br.x, points.tl.y);
  points.bl = new fabric.Point(points.tl.x, points.br.y);
  this.vptCoords = points;
  return points;
},

/**
 * Renders background, objects, overlay and controls.
 * @param {CanvasRenderingContext2D} ctx
 * @param {Array} objects to render
 * @return {fabric.Canvas} instance
 * @chainable
 */
renderCanvas: function(ctx, objects) {
  this.calcViewportBoundaries();
  this.clearContext(ctx);
  this.fire('before:render');
  if (this.clipTo) {
    fabric.util.clipContext(this, ctx);
  }
  this._renderBackground(ctx);

  ctx.save();
  //apply viewport transform once for all rendering process
  ctx.transform.apply(ctx, this.viewportTransform);
  this._renderObjects(ctx, objects);
  ctx.restore();
  if (!this.controlsAboveOverlay && this.interactive) {
    this.drawControls(ctx);
  }
  if (this.clipTo) {
    ctx.restore();
  }
  this._renderOverlay(ctx);
  if (this.controlsAboveOverlay && this.interactive) {
    this.drawControls(ctx);
  }
  this.fire('after:render');
},

/**
 * @private
 * @param {CanvasRenderingContext2D} ctx Context to render on
 * @param {Array} objects to render
 */
_renderObjects: function(ctx, objects) {
  for (var i = 0, length = objects.length; i < length; ++i) {
    objects[i] && objects[i].render(ctx);
  }
},

/**
 * @private
 * @param {CanvasRenderingContext2D} ctx Context to render on
 * @param {string} property 'background' or 'overlay'
 */
_renderBackgroundOrOverlay: function(ctx, property) {
  var object = this[property + 'Color'];
  if (object) {
    ctx.fillStyle = object.toLive
      ? object.toLive(ctx, this)
      : object;
```

```
        ctx.fillRect(
          object.offsetX || 0,
          object.offsetY || 0,
          this.width,
          this.height);
      }
      object = this[property + 'Image'];
      if (object) {
        if (this[property + 'Vpt']) {
          ctx.save();
          ctx.transform.apply(ctx, this.viewportTransform);
        }
        object.render(ctx);
        this[property + 'Vpt'] && ctx.restore();
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderBackground: function(ctx) {
      this._renderBackgroundOrOverlay(ctx, 'background');
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderOverlay: function(ctx) {
      this._renderBackgroundOrOverlay(ctx, 'overlay');
    },

    /**
     * Returns coordinates of a center of canvas.
     * Returned value is an object with top and left properties
     * @return {Object} object with "top" and "left" number values
     */
    getCenter: function () {
      return {
        top: this.getHeight() / 2,
        left: this.getWidth() / 2
      };
    },

    /**
     * Centers object horizontally in the canvas
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center horizontally
     * @return {fabric.Canvas} thisArg
     */
    centerObjectH: function (object) {
      return this._centerObject(object, new fabric.Point(this.getCenter().left,
object.getCenterPoint().y));
    },

    /**
     * Centers object vertically in the canvas
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center vertically
     * @return {fabric.Canvas} thisArg
     * @chainable
```

```
     */
    centerObjectV: function (object) {
      return this._centerObject(object, new
fabric.Point(object.getCenterPoint().x, this.getCenter().top));
    },

    /**
     * Centers object vertically and horizontally in the canvas
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center vertically and
horizontally
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    centerObject: function(object) {
      var center = this.getCenter();

      return this._centerObject(object, new fabric.Point(center.left,
center.top));
    },

    /**
     * Centers object vertically and horizontally in the viewport
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center vertically and
horizontally
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    viewportCenterObject: function(object) {
      var vpCenter = this.getVpCenter();

      return this._centerObject(object, vpCenter);
    },

    /**
     * Centers object horizontally in the viewport, object.top is unchanged
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center vertically and
horizontally
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    viewportCenterObjectH: function(object) {
      var vpCenter = this.getVpCenter();
      this._centerObject(object, new fabric.Point(vpCenter.x,
object.getCenterPoint().y));
      return this;
    },

    /**
     * Centers object Vertically in the viewport, object.top is unchanged
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @param {fabric.Object} object Object to center vertically and
horizontally
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    viewportCenterObjectV: function(object) {
      var vpCenter = this.getVpCenter();
```

```
      return this._centerObject(object, new
fabric.Point(object.getCenterPoint().x, vpCenter.y));
    },

    /**
     * Calculate the point in canvas that correspond to the center of actual
viewport.
     * @return {fabric.Point} vpCenter, viewport center
     * @chainable
     */
    getVpCenter: function() {
      var center = this.getCenter(),
          iVpt = invertTransform(this.viewportTransform);
      return transformPoint({ x: center.left, y: center.top }, iVpt);
    },

    /**
     * @private
     * @param {fabric.Object} object Object to center
     * @param {fabric.Point} center Center point
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    _centerObject: function(object, center) {
      object.setPositionByOrigin(center, 'center', 'center');
      this.renderAll();
      return this;
    },

    /**
     * Returs dataless JSON representation of canvas
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {String} json string
     */
    toDatalessJSON: function (propertiesToInclude) {
      return this.toDatalessObject(propertiesToInclude);
    },

    /**
     * Returns object representation of canvas
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function (propertiesToInclude) {
      return this._toObjectMethod('toObject', propertiesToInclude);
    },

    /**
     * Returns dataless object representation of canvas
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toDatalessObject: function (propertiesToInclude) {
      return this._toObjectMethod('toDatalessObject', propertiesToInclude);
    },

    /**
     * @private
     */
    _toObjectMethod: function (methodName, propertiesToInclude) {
```

```
      var data = {
        objects: this._toObjects(methodName, propertiesToInclude)
      };

      extend(data, this.__serializeBgOverlay(methodName, propertiesToInclude));

      fabric.util.populateWithProperties(this, data, propertiesToInclude);

      return data;
    },

    /**
     * @private
     */
    _toObjects: function(methodName, propertiesToInclude) {
      return this.getObjects().filter(function(object) {
        return !object.excludeFromExport;
      }).map(function(instance) {
        return this._toObject(instance, methodName, propertiesToInclude);
      }, this);
    },

    /**
     * @private
     */
    _toObject: function(instance, methodName, propertiesToInclude) {
      var originalValue;

      if (!this.includeDefaultValues) {
        originalValue = instance.includeDefaultValues;
        instance.includeDefaultValues = false;
      }

      var object = instance[methodName](propertiesToInclude);
      if (!this.includeDefaultValues) {
        instance.includeDefaultValues = originalValue;
      }
      return object;
    },

    /**
     * @private
     */
    __serializeBgOverlay: function(methodName, propertiesToInclude) {
      var data = { }, bgImage = this.backgroundImage, overlay =
this.overlayImage;

      if (this.backgroundColor) {
        data.background = this.backgroundColor.toObject
          ? this.backgroundColor.toObject(propertiesToInclude)
          : this.backgroundColor;
      }

      if (this.overlayColor) {
        data.overlay = this.overlayColor.toObject
          ? this.overlayColor.toObject(propertiesToInclude)
          : this.overlayColor;
      }
      if (bgImage && !bgImage.excludeFromExport) {
        data.backgroundImage = this._toObject(bgImage, methodName,
propertiesToInclude);
      }
      if (overlay && !overlay.excludeFromExport) {
```

```
        data.overlayImage = this._toObject(overlay, methodName,
propertiesToInclude);
      }

      return data;
    },

    /* _TO_SVG_START_ */
    /**
     * When true, getSvgTransform() will apply the
StaticCanvas.viewportTransform to the SVG transformation. When true,
     * a zoomed canvas will then produce zoomed SVG output.
     * @type Boolean
     * @default
     */
    svgViewportTransformation: true,

    /**
     * Returns SVG representation of canvas
     * @function
     * @param {Object} [options] Options object for SVG output
     * @param {Boolean} [options.suppressPreamble=false] If true xml tag is not
included
     * @param {Object} [options.viewBox] SVG viewbox object
     * @param {Number} [options.viewBox.x] x-cooridnate of viewbox
     * @param {Number} [options.viewBox.y] y-coordinate of viewbox
     * @param {Number} [options.viewBox.width] Width of viewbox
     * @param {Number} [options.viewBox.height] Height of viewbox
     * @param {String} [options.encoding=UTF-8] Encoding of SVG output
     * @param {String} [options.width] desired width of svg with or without
units
     * @param {String} [options.height] desired height of svg with or without
units
     * @param {Function} [reviver] Method for further parsing of svg elements,
called after each fabric object converted into svg representation.
     * @return {String} SVG string
     * @tutorial {@link http://fabricjs.com/fabric-intro-part-3#serialization}
     * @see {@link http://jsfiddle.net/fabricjs/jQ3ZZ/|jsFiddle demo}
     * @example <caption>Normal SVG output</caption>
     * var svg = canvas.toSVG();
     * @example <caption>SVG output without preamble (without &lt;?xml
../>)</caption>
     * var svg = canvas.toSVG({suppressPreamble: true});
     * @example <caption>SVG output with viewBox attribute</caption>
     * var svg = canvas.toSVG({
     *   viewBox: {
     *     x: 100,
     *     y: 100,
     *     width: 200,
     *     height: 300
     *   }
     * });
     * @example <caption>SVG output with different encoding (default:
UTF-8)</caption>
     * var svg = canvas.toSVG({encoding: 'ISO-8859-1'});
     * @example <caption>Modify SVG output with reviver function</caption>
     * var svg = canvas.toSVG(null, function(svg) {
     *   return svg.replace('stroke-dasharray: ; stroke-linecap: butt; stroke-
linejoin: miter; stroke-miterlimit: 10; ', '');
     * });
     */
    toSVG: function(options, reviver) {
      options || (options = { });
```

```
      var markup = [];

      this._setSVGPreamble(markup, options);
      this._setSVGHeader(markup, options);

      this._setSVGBgOverlayColor(markup, 'backgroundColor');
      this._setSVGBgOverlayImage(markup, 'backgroundImage', reviver);

      this._setSVGObjects(markup, reviver);

      this._setSVGBgOverlayColor(markup, 'overlayColor');
      this._setSVGBgOverlayImage(markup, 'overlayImage', reviver);

      markup.push('</svg>');

      return markup.join('');
    },

    /**
     * @private
     */
    _setSVGPreamble: function(markup, options) {
      if (options.suppressPreamble) {
        return;
      }
      markup.push(
        '<?xml version="1.0" encoding="', (options.encoding || 'UTF-8'), '"
standalone="no" ?>\n',
          '<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" ',
            '"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">\n'
      );
    },

    /**
     * @private
     */
    _setSVGHeader: function(markup, options) {
      var width = options.width || this.width,
          height = options.height || this.height,
          vpt, viewBox = 'viewBox="0 0 ' + this.width + ' ' + this.height + '"
',
          NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS;

      if (options.viewBox) {
        viewBox = 'viewBox="' +
                options.viewBox.x + ' ' +
                options.viewBox.y + ' ' +
                options.viewBox.width + ' ' +
                options.viewBox.height + '" ';
      }
      else {
        if (this.svgViewportTransformation) {
          vpt = this.viewportTransform;
          viewBox = 'viewBox="' +
                  toFixed(-vpt[4] / vpt[0], NUM_FRACTION_DIGITS) + ' ' +
                  toFixed(-vpt[5] / vpt[3], NUM_FRACTION_DIGITS) + ' ' +
                  toFixed(this.width / vpt[0], NUM_FRACTION_DIGITS) + ' ' +
                  toFixed(this.height / vpt[3], NUM_FRACTION_DIGITS) + '" ';
        }
      }

      markup.push(
        '<svg ',
          'xmlns="http://www.w3.org/2000/svg" ',
```

```
          'xmlns:xlink="http://www.w3.org/1999/xlink" ',
          'version="1.1" ',
          'width="', width, '" ',
          'height="', height, '" ',
          viewBox,
          'xml:space="preserve">\n',
        '<desc>Created with Fabric.js ', fabric.version, '</desc>\n',
        '<defs>\n',
          this.createSVGFontFacesMarkup(),
          this.createSVGRefElementsMarkup(),
        '</defs>\n'
      );
    },

    /**
     * Creates markup containing SVG referenced elements like patterns,
gradients etc.
     * @return {String}
     */
    createSVGRefElementsMarkup: function() {
      var _this = this,
          markup = ['backgroundColor', 'overlayColor'].map(function(prop) {
            var fill = _this[prop];
            if (fill && fill.toLive) {
              return fill.toSVG(_this, false);
            }
          });
      return markup.join('');
    },

    /**
     * Creates markup containing SVG font faces,
     * font URLs for font faces must be collected by developers
     * and are not extracted from the DOM by fabricjs
     * @param {Array} objects Array of fabric objects
     * @return {String}
     */
    createSVGFontFacesMarkup: function() {
      var markup = '', fontList = { }, obj, fontFamily,
          style, row, rowIndex, _char, charIndex,
          fontPaths = fabric.fontPaths, objects = this.getObjects();

      for (var i = 0, len = objects.length; i < len; i++) {
        obj = objects[i];
        fontFamily = obj.fontFamily;
        if (obj.type.indexOf('text') === -1 || fontList[fontFamily] || !
fontPaths[fontFamily]) {
          continue;
        }
        fontList[fontFamily] = true;
        if (!obj.styles) {
          continue;
        }
        style = obj.styles;
        for (rowIndex in style) {
          row = style[rowIndex];
          for (charIndex in row) {
            _char = row[charIndex];
            fontFamily = _char.fontFamily;
            if (!fontList[fontFamily] && fontPaths[fontFamily]) {
              fontList[fontFamily] = true;
            }
          }
        }
```

```
      }

      for (var j in fontList) {
        markup += [
          '\t\t@font-face {\n',
          '\t\t\tfont-family: \'', j, '\';\n',
          '\t\t\tsrc: url(\'', fontPaths[j], '\');\n',
          '\t\t}\n'
        ].join('');
      }

      if (markup) {
        markup = [
          '\t<style type="text/css">',
          '<![CDATA[\n',
          markup,
          ']]>',
          '</style>\n'
        ].join('');
      }

      return markup;
    },

    /**
     * @private
     */
    _setSVGObjects: function(markup, reviver) {
      var instance;
      for (var i = 0, objects = this.getObjects(), len = objects.length; i <
len; i++) {
        instance = objects[i];
        if (instance.excludeFromExport) {
          continue;
        }
        this._setSVGObject(markup, instance, reviver);
      }
    },

    /**
     * push single object svg representation in the markup
     * @private
     */
    _setSVGObject: function(markup, instance, reviver) {
      markup.push(instance.toSVG(reviver));
    },

    /**
     * @private
     */
    _setSVGBgOverlayImage: function(markup, property, reviver) {
      if (this[property] && this[property].toSVG) {
        markup.push(this[property].toSVG(reviver));
      }
    },

    /**
     * @private
     */
    _setSVGBgOverlayColor: function(markup, property) {
      var filler = this[property];
      if (!filler) {
        return;
      }
```

```
      if (filler.toLive) {
        var repeat = filler.repeat;
        markup.push(
          '<rect transform="translate(', this.width / 2, ',', this.height / 2,
')"',
            ' x="', filler.offsetX - this.width / 2, '" y="', filler.offsetY -
this.height / 2, '" ',
            'width="',
              (repeat === 'repeat-y' || repeat === 'no-repeat'
                ? filler.source.width
                : this.width),
            '" height="',
              (repeat === 'repeat-x' || repeat === 'no-repeat'
                ? filler.source.height
                : this.height),
            '" fill="url(#SVGID_' + filler.id + ')"',
          '></rect>\n'
        );
      }
      else {
        markup.push(
          '<rect x="0" y="0" ',
            'width="', this.width,
            '" height="', this.height,
            '" fill="', this[property], '"',
          '></rect>\n'
        );
      }
    },
    /* _TO_SVG_END_ */

    /**
     * Moves an object or the objects of a multiple selection
     * to the bottom of the stack of drawn objects
     * @param {fabric.Object} object Object to send to back
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    sendToBack: function (object) {
      if (!object) {
        return this;
      }
      var activeGroup = this._activeGroup,
          i, obj, objs;
      if (object === activeGroup) {
        objs = activeGroup._objects;
        for (i = objs.length; i--;) {
          obj = objs[i];
          removeFromArray(this._objects, obj);
          this._objects.unshift(obj);
        }
      }
      else {
        removeFromArray(this._objects, object);
        this._objects.unshift(object);
      }
      return this.renderAll && this.renderAll();
    },

    /**
     * Moves an object or the objects of a multiple selection
     * to the top of the stack of drawn objects
     * @param {fabric.Object} object Object to send
     * @return {fabric.Canvas} thisArg
```

```
     * @chainable
     */
    bringToFront: function (object) {
      if (!object) {
        return this;
      }
      var activeGroup = this._activeGroup,
          i, obj, objs;
      if (object === activeGroup) {
        objs = activeGroup._objects;
        for (i = 0; i < objs.length; i++) {
          obj = objs[i];
          removeFromArray(this._objects, obj);
          this._objects.push(obj);
        }
      }
      else {
        removeFromArray(this._objects, object);
        this._objects.push(object);
      }
      return this.renderAll && this.renderAll();
    },

    /**
     * Moves an object or a selection down in stack of drawn objects
     * @param {fabric.Object} object Object to send
     * @param {Boolean} [intersecting] If `true`, send object behind next lower
intersecting object
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    sendBackwards: function (object, intersecting) {
      if (!object) {
        return this;
      }

      var activeGroup = this._activeGroup,
          i, obj, idx, newIdx, objs, objsMoved = 0;

      if (object === activeGroup) {
        objs = activeGroup._objects;
        for (i = 0; i < objs.length; i++) {
          obj = objs[i];
          idx = this._objects.indexOf(obj);
          if (idx > 0 + objsMoved) {
            newIdx = idx - 1;
            removeFromArray(this._objects, obj);
            this._objects.splice(newIdx, 0, obj);
          }
          objsMoved++;
        }
      }
      else {
        idx = this._objects.indexOf(object);
        if (idx !== 0) {
          // if object is not on the bottom of stack
          newIdx = this._findNewLowerIndex(object, idx, intersecting);
          removeFromArray(this._objects, object);
          this._objects.splice(newIdx, 0, object);
        }
      }
      this.renderAll && this.renderAll();
      return this;
    },
```

```
    /**
     * @private
     */
    _findNewLowerIndex: function(object, idx, intersecting) {
      var newIdx;

      if (intersecting) {
        newIdx = idx;

        // traverse down the stack looking for the nearest intersecting object
        for (var i = idx - 1; i >= 0; --i) {

          var isIntersecting = object.intersectsWithObject(this._objects[i]) ||
                               object.isContainedWithinObject(this._objects[i])
||
                               this._objects[i].isContainedWithinObject(object);

          if (isIntersecting) {
            newIdx = i;
            break;
          }
        }
      }
      else {
        newIdx = idx - 1;
      }

      return newIdx;
    },

    /**
     * Moves an object or a selection up in stack of drawn objects
     * @param {fabric.Object} object Object to send
     * @param {Boolean} [intersecting] If `true`, send object in front of next
upper intersecting object
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    bringForward: function (object, intersecting) {
      if (!object) {
        return this;
      }

      var activeGroup = this._activeGroup,
          i, obj, idx, newIdx, objs, objsMoved = 0;

      if (object === activeGroup) {
        objs = activeGroup._objects;
        for (i = objs.length; i--;) {
          obj = objs[i];
          idx = this._objects.indexOf(obj);
          if (idx < this._objects.length - 1 - objsMoved) {
            newIdx = idx + 1;
            removeFromArray(this._objects, obj);
            this._objects.splice(newIdx, 0, obj);
          }
          objsMoved++;
        }
      }
      else {
        idx = this._objects.indexOf(object);
        if (idx !== this._objects.length - 1) {
          // if object is not on top of stack (last item in an array)
```

```
          newIdx = this._findNewUpperIndex(object, idx, intersecting);
          removeFromArray(this._objects, object);
          this._objects.splice(newIdx, 0, object);
        }
      }
      this.renderAll && this.renderAll();
      return this;
    },

    /**
     * @private
     */
    _findNewUpperIndex: function(object, idx, intersecting) {
      var newIdx;

      if (intersecting) {
        newIdx = idx;

        // traverse up the stack looking for the nearest intersecting object
        for (var i = idx + 1; i < this._objects.length; ++i) {

          var isIntersecting = object.intersectsWithObject(this._objects[i]) ||
                               object.isContainedWithinObject(this._objects[i])
||
                               this._objects[i].isContainedWithinObject(object);

          if (isIntersecting) {
            newIdx = i;
            break;
          }
        }
      }
      else {
        newIdx = idx + 1;
      }

      return newIdx;
    },

    /**
     * Moves an object to specified level in stack of drawn objects
     * @param {fabric.Object} object Object to send
     * @param {Number} index Position to move to
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    moveTo: function (object, index) {
      removeFromArray(this._objects, object);
      this._objects.splice(index, 0, object);
      return this.renderAll && this.renderAll();
    },

    /**
     * Clears a canvas element and removes all event listeners
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    dispose: function () {
      this.clear();
      return this;
    },

    /**
     * Returns a string representation of an instance
```

```
     * @return {String} string representation of an instance
     */
    toString: function () {
      return '#<fabric.Canvas (' + this.complexity() + '): ' +
               '{ objects: ' + this.getObjects().length + ' }>';
    }
  });

  extend(fabric.StaticCanvas.prototype, fabric.Observable);
  extend(fabric.StaticCanvas.prototype, fabric.Collection);
  extend(fabric.StaticCanvas.prototype, fabric.DataURLExporter);

  extend(fabric.StaticCanvas, /** @lends fabric.StaticCanvas */ {

    /**
     * @static
     * @type String
     * @default
     */
    EMPTY_JSON: '{"objects": [], "background": "white"}',

    /**
     * Provides a way to check support of some of the canvas methods
     * (either those of HTMLCanvasElement itself, or rendering context)
     *
     * @param {String} methodName Method to check support for;
     *                            Could be one of "getImageData", "toDataURL",
"toDataURLWithQuality" or "setLineDash"
     * @return {Boolean | null} `true` if method is supported (or at least
exists),
     *                          `null` if canvas element or context can not be
initialized
     */
    supports: function (methodName) {
      var el = fabric.util.createCanvasElement();

      if (!el || !el.getContext) {
        return null;
      }

      var ctx = el.getContext('2d');
      if (!ctx) {
        return null;
      }

      switch (methodName) {

        case 'getImageData':
          return typeof ctx.getImageData !== 'undefined';

        case 'setLineDash':
          return typeof ctx.setLineDash !== 'undefined';

        case 'toDataURL':
          return typeof el.toDataURL !== 'undefined';

        case 'toDataURLWithQuality':
          try {
            el.toDataURL('image/jpeg', 0);
            return true;
          }
          catch (e) { }
          return false;
```

```
        default:
          return null;
      }
    }
  });

  /**
   * Returns JSON representation of canvas
   * @function
   * @param {Array} [propertiesToInclude] Any properties that you might want to
additionally include in the output
   * @return {String} JSON string
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-3#serialization}
   * @see {@link http://jsfiddle.net/fabricjs/pec86/|jsFiddle demo}
   * @example <caption>JSON without additional properties</caption>
   * var json = canvas.toJSON();
   * @example <caption>JSON with additional properties included</caption>
   * var json = canvas.toJSON(['lockMovementX', 'lockMovementY', 'lockRotation',
'lockScalingX', 'lockScalingY', 'lockUniScaling']);
   * @example <caption>JSON without default values</caption>
   * canvas.includeDefaultValues = false;
   * var json = canvas.toJSON();
   */
  fabric.StaticCanvas.prototype.toJSON = fabric.StaticCanvas.prototype.toObject;

})();


/**
 * BaseBrush class
 * @class fabric.BaseBrush
 * @see {@link http://fabricjs.com/freedrawing|Freedrawing demo}
 */
fabric.BaseBrush = fabric.util.createClass(/** @lends fabric.BaseBrush.prototype
*/ {

  /**
   * Color of a brush
   * @type String
   * @default
   */
  color: 'rgb(0, 0, 0)',

  /**
   * Width of a brush
   * @type Number
   * @default
   */
  width: 1,

  /**
   * Shadow object representing shadow of this shape.
   * <b>Backwards incompatibility note:</b> This property replaces "shadowColor"
(String), "shadowOffsetX" (Number),
   * "shadowOffsetY" (Number) and "shadowBlur" (Number) since v1.2.12
   * @type fabric.Shadow
   * @default
   */
  shadow: null,

  /**
   * Line endings style of a brush (one of "butt", "round", "square")
   * @type String
   * @default
```

```
   */
  strokeLineCap: 'round',

  /**
   * Corner style of a brush (one of "bevil", "round", "miter")
   * @type String
   * @default
   */
  strokeLineJoin: 'round',

  /**
   * Stroke Dash Array.
   * @type Array
   * @default
   */
  strokeDashArray: null,

  /**
   * Sets shadow of an object
   * @param {Object|String} [options] Options object or string (e.g. "2px 2px
10px rgba(0,0,0,0.2)")
   * @return {fabric.Object} thisArg
   * @chainable
   */
  setShadow: function(options) {
    this.shadow = new fabric.Shadow(options);
    return this;
  },

  /**
   * Sets brush styles
   * @private
   */
  _setBrushStyles: function() {
    var ctx = this.canvas.contextTop;

    ctx.strokeStyle = this.color;
    ctx.lineWidth = this.width;
    ctx.lineCap = this.strokeLineCap;
    ctx.lineJoin = this.strokeLineJoin;
    if (this.strokeDashArray && fabric.StaticCanvas.supports('setLineDash')) {
      ctx.setLineDash(this.strokeDashArray);
    }
  },

  /**
   * Sets brush shadow styles
   * @private
   */
  _setShadow: function() {
    if (!this.shadow) {
      return;
    }

    var ctx = this.canvas.contextTop,
        zoom = this.canvas.getZoom();

    ctx.shadowColor = this.shadow.color;
    ctx.shadowBlur = this.shadow.blur * zoom;
    ctx.shadowOffsetX = this.shadow.offsetX * zoom;
    ctx.shadowOffsetY = this.shadow.offsetY * zoom;
  },

  /**
```

```
   * Removes brush shadow styles
   * @private
   */
  _resetShadow: function() {
    var ctx = this.canvas.contextTop;

    ctx.shadowColor = '';
    ctx.shadowBlur = ctx.shadowOffsetX = ctx.shadowOffsetY = 0;
  }
});


(function() {

  /**
   * PencilBrush class
   * @class fabric.PencilBrush
   * @extends fabric.BaseBrush
   */
  fabric.PencilBrush = fabric.util.createClass(fabric.BaseBrush, /** @lends
fabric.PencilBrush.prototype */ {

    /**
     * Constructor
     * @param {fabric.Canvas} canvas
     * @return {fabric.PencilBrush} Instance of a pencil brush
     */
    initialize: function(canvas) {
      this.canvas = canvas;
      this._points = [];
    },

    /**
     * Inovoked on mouse down
     * @param {Object} pointer
     */
    onMouseDown: function(pointer) {
      this._prepareForDrawing(pointer);
      // capture coordinates immediately
      // this allows to draw dots (when movement never occurs)
      this._captureDrawingPath(pointer);
      this._render();
    },

    /**
     * Inovoked on mouse move
     * @param {Object} pointer
     */
    onMouseMove: function(pointer) {
      this._captureDrawingPath(pointer);
      // redraw curve
      // clear top canvas
      this.canvas.clearContext(this.canvas.contextTop);
      this._render();
    },

    /**
     * Invoked on mouse up
     */
    onMouseUp: function() {
      this._finalizeAndAddPath();
    },

    /**
```

```
     * @private
     * @param {Object} pointer Actual mouse position related to the canvas.
     */
    _prepareForDrawing: function(pointer) {

      var p = new fabric.Point(pointer.x, pointer.y);

      this._reset();
      this._addPoint(p);

      this.canvas.contextTop.moveTo(p.x, p.y);
    },

    /**
     * @private
     * @param {fabric.Point} point Point to be added to points array
     */
    _addPoint: function(point) {
      this._points.push(point);
    },

    /**
     * Clear points array and set contextTop canvas style.
     * @private
     */
    _reset: function() {
      this._points.length = 0;

      this._setBrushStyles();
      this._setShadow();
    },

    /**
     * @private
     * @param {Object} pointer Actual mouse position related to the canvas.
     */
    _captureDrawingPath: function(pointer) {
      var pointerPoint = new fabric.Point(pointer.x, pointer.y);
      this._addPoint(pointerPoint);
    },

    /**
     * Draw a smooth path on the topCanvas using quadraticCurveTo
     * @private
     */
    _render: function() {
      var ctx  = this.canvas.contextTop,
          v = this.canvas.viewportTransform,
          p1 = this._points[0],
          p2 = this._points[1];

      ctx.save();
      ctx.transform(v[0], v[1], v[2], v[3], v[4], v[5]);
      ctx.beginPath();

      //if we only have 2 points in the path and they are the same
      //it means that the user only clicked the canvas without moving the mouse
      //then we should be drawing a dot. A path isn't drawn between two
identical dots
      //that's why we set them apart a bit
      if (this._points.length === 2 && p1.x === p2.x && p1.y === p2.y) {
        p1.x -= 0.5;
        p2.x += 0.5;
      }
```

```
    ctx.moveTo(p1.x, p1.y);

    for (var i = 1, len = this._points.length; i < len; i++) {
      // we pick the point between pi + 1 & pi + 2 as the
      // end point and p1 as our control point.
      var midPoint = p1.midPointFrom(p2);
      ctx.quadraticCurveTo(p1.x, p1.y, midPoint.x, midPoint.y);

      p1 = this._points[i];
      p2 = this._points[i + 1];
    }
    // Draw last line as a straight line while
    // we wait for the next point to be able to calculate
    // the bezier control point
    ctx.lineTo(p1.x, p1.y);
    ctx.stroke();
    ctx.restore();
  },

  /**
   * Converts points to SVG path
   * @param {Array} points Array of points
   * @return {String} SVG path
   */
  convertPointsToSVGPath: function(points) {
    var path = [],
        p1 = new fabric.Point(points[0].x, points[0].y),
        p2 = new fabric.Point(points[1].x, points[1].y);

    path.push('M ', points[0].x, ' ', points[0].y, ' ');
    for (var i = 1, len = points.length; i < len; i++) {
      var midPoint = p1.midPointFrom(p2);
      // p1 is our bezier control point
      // midpoint is our endpoint
      // start point is p(i-1) value.
      path.push('Q ', p1.x, ' ', p1.y, ' ', midPoint.x, ' ', midPoint.y, ' ');
      p1 = new fabric.Point(points[i].x, points[i].y);
      if ((i + 1) < points.length) {
        p2 = new fabric.Point(points[i + 1].x, points[i + 1].y);
      }
    }
    path.push('L ', p1.x, ' ', p1.y, ' ');
    return path;
  },

  /**
   * Creates fabric.Path object to add on canvas
   * @param {String} pathData Path data
   * @return {fabric.Path} Path to add on canvas
   */
  createPath: function(pathData) {
    var path = new fabric.Path(pathData, {
      fill: null,
      stroke: this.color,
      strokeWidth: this.width,
      strokeLineCap: this.strokeLineCap,
      strokeLineJoin: this.strokeLineJoin,
      strokeDashArray: this.strokeDashArray,
      originX: 'center',
      originY: 'center'
    });

    if (this.shadow) {
      this.shadow.affectStroke = true;
```

```
        path.setShadow(this.shadow);
      }

      return path;
    },

    /**
     * On mouseup after drawing the path on contextTop canvas
     * we use the points captured to create an new fabric path object
     * and add it to the fabric canvas.
     */
    _finalizeAndAddPath: function() {
      var ctx = this.canvas.contextTop;
      ctx.closePath();

      var pathData = this.convertPointsToSVGPath(this._points).join('');
      if (pathData === 'M 0 0 Q 0 0 0 0 L 0 0') {
        // do not create 0 width/height paths, as they are
        // rendered inconsistently across browsers
        // Firefox 4, for example, renders a dot,
        // whereas Chrome 10 renders nothing
        this.canvas.renderAll();
        return;
      }

      var path = this.createPath(pathData);

      this.canvas.add(path);
      path.setCoords();

      this.canvas.clearContext(this.canvas.contextTop);
      this._resetShadow();
      this.canvas.renderAll();

      // fire event 'path' created
      this.canvas.fire('path:created', { path: path });
    }
  });
})();


/**
 * CircleBrush class
 * @class fabric.CircleBrush
 */
fabric.CircleBrush = fabric.util.createClass(fabric.BaseBrush, /** @lends
fabric.CircleBrush.prototype */ {

  /**
   * Width of a brush
   * @type Number
   * @default
   */
  width: 10,

  /**
   * Constructor
   * @param {fabric.Canvas} canvas
   * @return {fabric.CircleBrush} Instance of a circle brush
   */
  initialize: function(canvas) {
    this.canvas = canvas;
    this.points = [];
  },
```

```
/**
 * Invoked inside on mouse down and mouse move
 * @param {Object} pointer
 */
drawDot: function(pointer) {
  var point = this.addPoint(pointer),
      ctx = this.canvas.contextTop,
      v = this.canvas.viewportTransform;
  ctx.save();
  ctx.transform(v[0], v[1], v[2], v[3], v[4], v[5]);

  ctx.fillStyle = point.fill;
  ctx.beginPath();
  ctx.arc(point.x, point.y, point.radius, 0, Math.PI * 2, false);
  ctx.closePath();
  ctx.fill();

  ctx.restore();
},

/**
 * Invoked on mouse down
 */
onMouseDown: function(pointer) {
  this.points.length = 0;
  this.canvas.clearContext(this.canvas.contextTop);
  this._setShadow();
  this.drawDot(pointer);
},

/**
 * Invoked on mouse move
 * @param {Object} pointer
 */
onMouseMove: function(pointer) {
  this.drawDot(pointer);
},

/**
 * Invoked on mouse up
 */
onMouseUp: function() {
  var originalRenderOnAddRemove = this.canvas.renderOnAddRemove;
  this.canvas.renderOnAddRemove = false;

  var circles = [];

  for (var i = 0, len = this.points.length; i < len; i++) {
    var point = this.points[i],
        circle = new fabric.Circle({
          radius: point.radius,
          left: point.x,
          top: point.y,
          originX: 'center',
          originY: 'center',
          fill: point.fill
        });

    this.shadow && circle.setShadow(this.shadow);

    circles.push(circle);
  }
  var group = new fabric.Group(circles, { originX: 'center', originY: 'center'
```

```
});
    group.canvas = this.canvas;

    this.canvas.add(group);
    this.canvas.fire('path:created', { path: group });

    this.canvas.clearContext(this.canvas.contextTop);
    this._resetShadow();
    this.canvas.renderOnAddRemove = originalRenderOnAddRemove;
    this.canvas.renderAll();
  },

  /**
   * @param {Object} pointer
   * @return {fabric.Point} Just added pointer point
   */
  addPoint: function(pointer) {
    var pointerPoint = new fabric.Point(pointer.x, pointer.y),

        circleRadius = fabric.util.getRandomInt(
                        Math.max(0, this.width - 20), this.width + 20) / 2,

        circleColor = new fabric.Color(this.color)
                        .setAlpha(fabric.util.getRandomInt(0, 100) / 100)
                        .toRgba();

    pointerPoint.radius = circleRadius;
    pointerPoint.fill = circleColor;

    this.points.push(pointerPoint);

    return pointerPoint;
  }
});


/**
 * SprayBrush class
 * @class fabric.SprayBrush
 */
fabric.SprayBrush = fabric.util.createClass( fabric.BaseBrush, /** @lends
fabric.SprayBrush.prototype */ {

  /**
   * Width of a spray
   * @type Number
   * @default
   */
  width:              10,

  /**
   * Density of a spray (number of dots per chunk)
   * @type Number
   * @default
   */
  density:            20,

  /**
   * Width of spray dots
   * @type Number
   * @default
   */
  dotWidth:           1,
```

```
  /**
   * Width variance of spray dots
   * @type Number
   * @default
   */
  dotWidthVariance:   1,

  /**
   * Whether opacity of a dot should be random
   * @type Boolean
   * @default
   */
  randomOpacity:        false,

  /**
   * Whether overlapping dots (rectangles) should be removed (for performance
reasons)
   * @type Boolean
   * @default
   */
  optimizeOverlapping:  true,

  /**
   * Constructor
   * @param {fabric.Canvas} canvas
   * @return {fabric.SprayBrush} Instance of a spray brush
   */
  initialize: function(canvas) {
    this.canvas = canvas;
    this.sprayChunks = [];
  },

  /**
   * Invoked on mouse down
   * @param {Object} pointer
   */
  onMouseDown: function(pointer) {
    this.sprayChunks.length = 0;
    this.canvas.clearContext(this.canvas.contextTop);
    this._setShadow();

    this.addSprayChunk(pointer);
    this.render();
  },

  /**
   * Invoked on mouse move
   * @param {Object} pointer
   */
  onMouseMove: function(pointer) {
    this.addSprayChunk(pointer);
    this.render();
  },

  /**
   * Invoked on mouse up
   */
  onMouseUp: function() {
    var originalRenderOnAddRemove = this.canvas.renderOnAddRemove;
    this.canvas.renderOnAddRemove = false;

    var rects = [];

    for (var i = 0, ilen = this.sprayChunks.length; i < ilen; i++) {
```

```
      var sprayChunk = this.sprayChunks[i];

      for (var j = 0, jlen = sprayChunk.length; j < jlen; j++) {

        var rect = new fabric.Rect({
          width: sprayChunk[j].width,
          height: sprayChunk[j].width,
          left: sprayChunk[j].x + 1,
          top: sprayChunk[j].y + 1,
          originX: 'center',
          originY: 'center',
          fill: this.color
        });

        this.shadow && rect.setShadow(this.shadow);
        rects.push(rect);
      }
    }

    if (this.optimizeOverlapping) {
      rects = this._getOptimizedRects(rects);
    }

    var group = new fabric.Group(rects, { originX: 'center', originY:
'center' });
    group.canvas = this.canvas;

    this.canvas.add(group);
    this.canvas.fire('path:created', { path: group });

    this.canvas.clearContext(this.canvas.contextTop);
    this._resetShadow();
    this.canvas.renderOnAddRemove = originalRenderOnAddRemove;
    this.canvas.renderAll();
  },

  /**
   * @private
   * @param {Array} rects
   */
  _getOptimizedRects: function(rects) {

    // avoid creating duplicate rects at the same coordinates
    var uniqueRects = { }, key;

    for (var i = 0, len = rects.length; i < len; i++) {
      key = rects[i].left + '' + rects[i].top;
      if (!uniqueRects[key]) {
        uniqueRects[key] = rects[i];
      }
    }
    var uniqueRectsArray = [];
    for (key in uniqueRects) {
      uniqueRectsArray.push(uniqueRects[key]);
    }

    return uniqueRectsArray;
  },

  /**
   * Renders brush
   */
  render: function() {
    var ctx = this.canvas.contextTop;
```

```
    ctx.fillStyle = this.color;

    var v = this.canvas.viewportTransform;
    ctx.save();
    ctx.transform(v[0], v[1], v[2], v[3], v[4], v[5]);

    for (var i = 0, len = this.sprayChunkPoints.length; i < len; i++) {
      var point = this.sprayChunkPoints[i];
      if (typeof point.opacity !== 'undefined') {
        ctx.globalAlpha = point.opacity;
      }
      ctx.fillRect(point.x, point.y, point.width, point.width);
    }
    ctx.restore();
  },

  /**
   * @param {Object} pointer
   */
  addSprayChunk: function(pointer) {
    this.sprayChunkPoints = [];

    var x, y, width, radius = this.width / 2;

    for (var i = 0; i < this.density; i++) {

      x = fabric.util.getRandomInt(pointer.x - radius, pointer.x + radius);
      y = fabric.util.getRandomInt(pointer.y - radius, pointer.y + radius);

      if (this.dotWidthVariance) {
        width = fabric.util.getRandomInt(
          // bottom clamp width to 1
          Math.max(1, this.dotWidth - this.dotWidthVariance),
          this.dotWidth + this.dotWidthVariance);
      }
      else {
        width = this.dotWidth;
      }

      var point = new fabric.Point(x, y);
      point.width = width;

      if (this.randomOpacity) {
        point.opacity = fabric.util.getRandomInt(0, 100) / 100;
      }

      this.sprayChunkPoints.push(point);
    }

    this.sprayChunks.push(this.sprayChunkPoints);
  }
});


/**
 * PatternBrush class
 * @class fabric.PatternBrush
 * @extends fabric.BaseBrush
 */
fabric.PatternBrush = fabric.util.createClass(fabric.PencilBrush, /** @lends
fabric.PatternBrush.prototype */ {

  getPatternSrc: function() {
```

```
    var dotWidth = 20,
        dotDistance = 5,
        patternCanvas = fabric.document.createElement('canvas'),
        patternCtx = patternCanvas.getContext('2d');

    patternCanvas.width = patternCanvas.height = dotWidth + dotDistance;

    patternCtx.fillStyle = this.color;
    patternCtx.beginPath();
    patternCtx.arc(dotWidth / 2, dotWidth / 2, dotWidth / 2, 0, Math.PI * 2,
false);
    patternCtx.closePath();
    patternCtx.fill();

    return patternCanvas;
  },

  getPatternSrcFunction: function() {
    return String(this.getPatternSrc).replace('this.color', '"' + this.color +
'"');
  },

  /**
   * Creates "pattern" instance property
   */
  getPattern: function() {
    return this.canvas.contextTop.createPattern(this.source ||
this.getPatternSrc(), 'repeat');
  },

  /**
   * Sets brush styles
   */
  _setBrushStyles: function() {
    this.callSuper('_setBrushStyles');
    this.canvas.contextTop.strokeStyle = this.getPattern();
  },

  /**
   * Creates path
   */
  createPath: function(pathData) {
    var path = this.callSuper('createPath', pathData),
        topLeft = path._getLeftTopCoords().scalarAdd(path.strokeWidth / 2);

    path.stroke = new fabric.Pattern({
      source: this.source || this.getPatternSrcFunction(),
      offsetX: -topLeft.x,
      offsetY: -topLeft.y
    });
    return path;
  }
});


(function() {

  var getPointer = fabric.util.getPointer,
      degreesToRadians = fabric.util.degreesToRadians,
      radiansToDegrees = fabric.util.radiansToDegrees,
      atan2 = Math.atan2,
      abs = Math.abs,
      supportLineDash = fabric.StaticCanvas.supports('setLineDash'),
```

```
      STROKE_OFFSET = 0.5;

  /**
   * Canvas class
   * @class fabric.Canvas
   * @extends fabric.StaticCanvas
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-1#canvas}
   * @see {@link fabric.Canvas#initialize} for constructor definition
   *
   * @fires object:added
   * @fires object:modified
   * @fires object:rotating
   * @fires object:scaling
   * @fires object:moving
   * @fires object:selected
   *
   * @fires before:selection:cleared
   * @fires selection:cleared
   * @fires selection:created
   *
   * @fires path:created
   * @fires mouse:down
   * @fires mouse:move
   * @fires mouse:up
   * @fires mouse:over
   * @fires mouse:out
   *
   */
  fabric.Canvas = fabric.util.createClass(fabric.StaticCanvas, /** @lends
fabric.Canvas.prototype */ {

    /**
     * Constructor
     * @param {HTMLElement | String} el &lt;canvas> element to initialize
instance on
     * @param {Object} [options] Options object
     * @return {Object} thisArg
     */
    initialize: function(el, options) {
      options || (options = { });

      this._initStatic(el, options);
      this._initInteractive();
      this._createCacheCanvas();
    },

    /**
     * When true, objects can be transformed by one side (unproportionally)
     * @type Boolean
     * @default
     */
    uniScaleTransform:      false,

    /**
     * Indicates which key enable unproportional scaling
     * values: 'altKey', 'shiftKey', 'ctrlKey'.
     * If `null` or 'none' or any other string that is not a modifier key
     * feature is disabled feature disabled.
     * @since 1.6.2
     * @type String
     * @default
     */
    uniScaleKey:           'shiftKey',
```

```
    /**
     * When true, objects use center point as the origin of scale
transformation.
     * <b>Backwards incompatibility note:</b> This property replaces
"centerTransform" (Boolean).
     * @since 1.3.4
     * @type Boolean
     * @default
     */
    centeredScaling:        false,

    /**
     * When true, objects use center point as the origin of rotate
transformation.
     * <b>Backwards incompatibility note:</b> This property replaces
"centerTransform" (Boolean).
     * @since 1.3.4
     * @type Boolean
     * @default
     */
    centeredRotation:       false,

    /**
     * Indicates which key enable centered Transfrom
     * values: 'altKey', 'shiftKey', 'ctrlKey'.
     * If `null` or 'none' or any other string that is not a modifier key
     * feature is disabled feature disabled.
     * @since 1.6.2
     * @type String
     * @default
     */
    centeredKey:           'altKey',

    /**
     * Indicates which key enable alternate action on corner
     * values: 'altKey', 'shiftKey', 'ctrlKey'.
     * If `null` or 'none' or any other string that is not a modifier key
     * feature is disabled feature disabled.
     * @since 1.6.2
     * @type String
     * @default
     */
    altActionKey:           'shiftKey',

    /**
     * Indicates that canvas is interactive. This property should not be
changed.
     * @type Boolean
     * @default
     */
    interactive:            true,

    /**
     * Indicates whether group selection should be enabled
     * @type Boolean
     * @default
     */
    selection:              true,

    /**
     * Indicates which key enable multiple click selection
     * values: 'altKey', 'shiftKey', 'ctrlKey'.
     * If `null` or 'none' or any other string that is not a modifier key
     * feature is disabled feature disabled.
```

```
     * @since 1.6.2
     * @type String
     * @default
     */
    selectionKey:           'shiftKey',

    /**
     * Indicates which key enable alternative selection
     * in case of target overlapping with active object
     * values: 'altKey', 'shiftKey', 'ctrlKey'.
     * If `null` or 'none' or any other string that is not a modifier key
     * feature is disabled feature disabled.
     * @since 1.6.5
     * @type null|String
     * @default
     */
    altSelectionKey:           null,

    /**
     * Color of selection
     * @type String
     * @default
     */
    selectionColor:         'rgba(100, 100, 255, 0.3)', // blue

    /**
     * Default dash array pattern
     * If not empty the selection border is dashed
     * @type Array
     */
    selectionDashArray:     [],

    /**
     * Color of the border of selection (usually slightly darker than color of
selection itself)
     * @type String
     * @default
     */
    selectionBorderColor:   'rgba(255, 255, 255, 0.3)',

    /**
     * Width of a line used in object/group selection
     * @type Number
     * @default
     */
    selectionLineWidth:     1,

    /**
     * Default cursor value used when hovering over an object on canvas
     * @type String
     * @default
     */
    hoverCursor:            'move',

    /**
     * Default cursor value used when moving an object on canvas
     * @type String
     * @default
     */
    moveCursor:             'move',

    /**
     * Default cursor value used for the entire canvas
     * @type String
```

```
     * @default
     */
    defaultCursor:          'default',

    /**
     * Cursor value used during free drawing
     * @type String
     * @default
     */
    freeDrawingCursor:      'crosshair',

    /**
     * Cursor value used for rotation point
     * @type String
     * @default
     */
    rotationCursor:         'crosshair',

    /**
     * Default element class that's given to wrapper (div) element of canvas
     * @type String
     * @default
     */
    containerClass:         'canvas-container',

    /**
     * When true, object detection happens on per-pixel basis rather than on
per-bounding-box
     * @type Boolean
     * @default
     */
    perPixelTargetFind:     false,

    /**
     * Number of pixels around target pixel to tolerate (consider active) during
object detection
     * @type Number
     * @default
     */
    targetFindTolerance:    0,

    /**
     * When true, target detection is skipped when hovering over canvas. This
can be used to improve performance.
     * @type Boolean
     * @default
     */
    skipTargetFind:         false,

    /**
     * When true, mouse events on canvas (mousedown/mousemove/mouseup) result in
free drawing.
     * After mousedown, mousemove creates a shape,
     * and then mouseup finalizes it and adds an instance of `fabric.Path` onto
canvas.
     * @tutorial {@link http://fabricjs.com/fabric-intro-part-4#free_drawing}
     * @type Boolean
     * @default
     */
    isDrawingMode:          false,

    /**
     * Indicates whether objects should remain in current stack position when
selected.
```

```
     * When false objects are brought to top and rendered as part of the
selection group
     * @type Boolean
     * @default
     */
    preserveObjectStacking: false,

    /**
     * Indicates the angle that an object will lock to while rotating.
     * @type Number
     * @since 1.6.7
     * @default
     */
    snapAngle: 0,

    /**
     * Indicates the distance from the snapAngle the rotation will lock to the
snapAngle.
     * When `null`, the snapThreshold will default to the snapAngle.
     * @type null|Number
     * @since 1.6.7
     * @default
     */
    snapThreshold: null,

    /**
     * Indicates if the right click on canvas can output the context menu or not
     * @type Boolean
     * @since 1.6.5
     * @default
     */
    stopContextMenu: false,

    /**
     * Indicates if the canvas can fire right click events
     * @type Boolean
     * @since 1.6.5
     * @default
     */
    fireRightClick: false,

    /**
     * Indicates if the canvas can fire middle click events
     * @type Boolean
     * @since 1.7.8
     * @default
     */
    fireMiddleClick: false,

    /**
     * @private
     */
    _initInteractive: function() {
      this._currentTransform = null;
      this._groupSelector = null;
      this._initWrapperElement();
      this._createUpperCanvas();
      this._initEventListeners();

      this._initRetinaScaling();

      this.freeDrawingBrush = fabric.PencilBrush && new
fabric.PencilBrush(this);
```

```
      this.calcOffset();
    },

    /**
     * Divides objects in two groups, one to render immediately
     * and one to render as activeGroup.
     * @return {Array} objects to render immediately and pushes the other in the
activeGroup.
     */
    _chooseObjectsToRender: function() {
      var activeGroup = this.getActiveGroup(),
          activeObject = this.getActiveObject(),
          object, objsToRender = [], activeGroupObjects = [];

      if ((activeGroup || activeObject) && !this.preserveObjectStacking) {
        for (var i = 0, length = this._objects.length; i < length; i++) {
          object = this._objects[i];
          if ((!activeGroup || !activeGroup.contains(object)) && object !==
activeObject) {
            objsToRender.push(object);
          }
          else {
            activeGroupObjects.push(object);
          }
        }
        if (activeGroup) {
          activeGroup._set('_objects', activeGroupObjects);
          objsToRender.push(activeGroup);
        }
        activeObject && objsToRender.push(activeObject);
      }
      else {
        objsToRender = this._objects;
      }
      return objsToRender;
    },

    /**
     * Renders both the top canvas and the secondary container canvas.
     * @return {fabric.Canvas} instance
     * @chainable
     */
    renderAll: function () {
      if (this.contextTopDirty && !this._groupSelector && !this.isDrawingMode) {
        this.clearContext(this.contextTop);
        this.contextTopDirty = false;
      }
      var canvasToDrawOn = this.contextContainer;
      this.renderCanvas(canvasToDrawOn, this._chooseObjectsToRender());
      return this;
    },

    /**
     * Method to render only the top canvas.
     * Also used to render the group selection box.
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    renderTop: function () {
      var ctx = this.contextTop;
      this.clearContext(ctx);

      // we render the top context - last object
      if (this.selection && this._groupSelector) {
```

```
        this._drawSelection(ctx);
      }

      this.fire('after:render');
      this.contextTopDirty = true;
      return this;
    },

    /**
     * Resets the current transform to its original values and chooses the type
of resizing based on the event
     * @private
     */
    _resetCurrentTransform: function() {
      var t = this._currentTransform;

      t.target.set({
        scaleX: t.original.scaleX,
        scaleY: t.original.scaleY,
        skewX: t.original.skewX,
        skewY: t.original.skewY,
        left: t.original.left,
        top: t.original.top
      });

      if (this._shouldCenterTransform(t.target)) {
        if (t.action === 'rotate') {
          this._setOriginToCenter(t.target);
        }
        else {
          if (t.originX !== 'center') {
            if (t.originX === 'right') {
              t.mouseXSign = -1;
            }
            else {
              t.mouseXSign = 1;
            }
          }
          if (t.originY !== 'center') {
            if (t.originY === 'bottom') {
              t.mouseYSign = -1;
            }
            else {
              t.mouseYSign = 1;
            }
          }

          t.originX = 'center';
          t.originY = 'center';
        }
      }
      else {
        t.originX = t.original.originX;
        t.originY = t.original.originY;
      }
    },

    /**
     * Checks if point is contained within an area of given object
     * @param {Event} e Event object
     * @param {fabric.Object} target Object to test against
     * @param {Object} [point] x,y object of point coordinates we want to check.
     * @return {Boolean} true if point is contained within an area of given
object
```

```
 */
containsPoint: function (e, target, point) {
  var ignoreZoom = true,
      pointer = point || this.getPointer(e, ignoreZoom),
      xy;

  if (target.group && target.group === this.getActiveGroup()) {
    xy = this._normalizePointer(target.group, pointer);
  }
  else {
    xy = { x: pointer.x, y: pointer.y };
  }
  // http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/u32.html
  // http://idav.ucdavis.edu/~okreylos/TAship/Spring2000/PointInPolygon.html
  return (target.containsPoint(xy) || target._findTargetCorner(pointer));
},

/**
 * @private
 */
_normalizePointer: function (object, pointer) {
  var m = object.calcTransformMatrix(),
      invertedM = fabric.util.invertTransform(m),
      vptPointer = this.restorePointerVpt(pointer);
  return fabric.util.transformPoint(vptPointer, invertedM);
},

/**
 * Returns true if object is transparent at a certain location
 * @param {fabric.Object} target Object to check
 * @param {Number} x Left coordinate
 * @param {Number} y Top coordinate
 * @return {Boolean}
 */
isTargetTransparent: function (target, x, y) {
  var hasBorders = target.hasBorders,
      transparentCorners = target.transparentCorners,
      ctx = this.contextCache,
      originalColor = target.selectionBackgroundColor;

  target.hasBorders = target.transparentCorners = false;
  target.selectionBackgroundColor = '';

  ctx.save();
  ctx.transform.apply(ctx, this.viewportTransform);
  target.render(ctx);
  ctx.restore();

  target.active && target._renderControls(ctx);

  target.hasBorders = hasBorders;
  target.transparentCorners = transparentCorners;
  target.selectionBackgroundColor = originalColor;

  var isTransparent = fabric.util.isTransparent(
    ctx, x, y, this.targetFindTolerance);

  this.clearContext(ctx);

  return isTransparent;
},

/**
 * @private
```

```
     * @param {Event} e Event object
     * @param {fabric.Object} target
     */
    _shouldClearSelection: function (e, target) {
      var activeGroup = this.getActiveGroup(),
          activeObject = this.getActiveObject();

      return (
        !target
        ||
        (target &&
          activeGroup &&
          !activeGroup.contains(target) &&
          activeGroup !== target &&
          !e[this.selectionKey])
        ||
        (target && !target.evented)
        ||
        (target &&
          !target.selectable &&
          activeObject &&
          activeObject !== target)
      );
    },

    /**
     * @private
     * @param {fabric.Object} target
     */
    _shouldCenterTransform: function (target) {
      if (!target) {
        return;
      }

      var t = this._currentTransform,
          centerTransform;

      if (t.action === 'scale' || t.action === 'scaleX' || t.action ===
'scaleY') {
        centerTransform = this.centeredScaling || target.centeredScaling;
      }
      else if (t.action === 'rotate') {
        centerTransform = this.centeredRotation || target.centeredRotation;
      }

      return centerTransform ? !t.altKey : t.altKey;
    },

    /**
     * @private
     */
    _getOriginFromCorner: function(target, corner) {
      var origin = {
        x: target.originX,
        y: target.originY
      };

      if (corner === 'ml' || corner === 'tl' || corner === 'bl') {
        origin.x = 'right';
      }
      else if (corner === 'mr' || corner === 'tr' || corner === 'br') {
        origin.x = 'left';
      }
```

```
  if (corner === 'tl' || corner === 'mt' || corner === 'tr') {
    origin.y = 'bottom';
  }
  else if (corner === 'bl' || corner === 'mb' || corner === 'br') {
    origin.y = 'top';
  }

  return origin;
},

/**
 * @private
 */
_getActionFromCorner: function(target, corner, e) {
  if (!corner) {
    return 'drag';
  }

  switch (corner) {
    case 'mtr':
      return 'rotate';
    case 'ml':
    case 'mr':
      return e[this.altActionKey] ? 'skewY' : 'scaleX';
    case 'mt':
    case 'mb':
      return e[this.altActionKey] ? 'skewX' : 'scaleY';
    default:
      return 'scale';
  }
},

/**
 * @private
 * @param {Event} e Event object
 * @param {fabric.Object} target
 */
_setupCurrentTransform: function (e, target) {
  if (!target) {
    return;
  }

  var pointer = this.getPointer(e),
      corner = target._findTargetCorner(this.getPointer(e, true)),
      action = this._getActionFromCorner(target, corner, e),
      origin = this._getOriginFromCorner(target, corner);

  this._currentTransform = {
    target: target,
    action: action,
    corner: corner,
    scaleX: target.scaleX,
    scaleY: target.scaleY,
    skewX: target.skewX,
    skewY: target.skewY,
    offsetX: pointer.x - target.left,
    offsetY: pointer.y - target.top,
    originX: origin.x,
    originY: origin.y,
    ex: pointer.x,
    ey: pointer.y,
    lastX: pointer.x,
    lastY: pointer.y,
    left: target.left,
```

```
    top: target.top,
    theta: degreesToRadians(target.angle),
    width: target.width * target.scaleX,
    mouseXSign: 1,
    mouseYSign: 1,
    shiftKey: e.shiftKey,
    altKey: e[this.centeredKey]
  };

  this._currentTransform.original = {
    left: target.left,
    top: target.top,
    scaleX: target.scaleX,
    scaleY: target.scaleY,
    skewX: target.skewX,
    skewY: target.skewY,
    originX: origin.x,
    originY: origin.y
  };

  this._resetCurrentTransform();
},

/**
 * Translates object by "setting" its left/top
 * @private
 * @param {Number} x pointer's x coordinate
 * @param {Number} y pointer's y coordinate
 * @return {Boolean} true if the translation occurred
 */
_translateObject: function (x, y) {
  var transform = this._currentTransform,
      target = transform.target,
      newLeft = x - transform.offsetX,
      newTop = y - transform.offsetY,
      moveX = !target.get('lockMovementX') && target.left !== newLeft,
      moveY = !target.get('lockMovementY') && target.top !== newTop;

  moveX && target.set('left', newLeft);
  moveY && target.set('top', newTop);
  return moveX || moveY;
},

/**
 * Check if we are increasing a positive skew or lower it,
 * checking mouse direction and pressed corner.
 * @private
 */
_changeSkewTransformOrigin: function(mouseMove, t, by) {
  var property = 'originX', origins = { 0: 'center' },
      skew = t.target.skewX, originA = 'left', originB = 'right',
      corner = t.corner === 'mt' || t.corner === 'ml' ? 1 : -1,
      flipSign = 1;

  mouseMove = mouseMove > 0 ? 1 : -1;
  if (by === 'y') {
    skew = t.target.skewY;
    originA = 'top';
    originB = 'bottom';
    property = 'originY';
  }
  origins[-1] = originA;
  origins[1] = originB;
```

```
      t.target.flipX && (flipSign *= -1);
      t.target.flipY && (flipSign *= -1);

      if (skew === 0) {
        t.skewSign = -corner * mouseMove * flipSign;
        t[property] = origins[-mouseMove];
      }
      else {
        skew = skew > 0 ? 1 : -1;
        t.skewSign = skew;
        t[property] = origins[skew * corner * flipSign];
      }
    },

    /**
     * Skew object by mouse events
     * @private
     * @param {Number} x pointer's x coordinate
     * @param {Number} y pointer's y coordinate
     * @param {String} by Either 'x' or 'y'
     * @return {Boolean} true if the skewing occurred
     */
    _skewObject: function (x, y, by) {
      var t = this._currentTransform,
          target = t.target, skewed = false,
          lockSkewingX = target.get('lockSkewingX'),
          lockSkewingY = target.get('lockSkewingY');

      if ((lockSkewingX && by === 'x') || (lockSkewingY && by === 'y')) {
        return false;
      }

      // Get the constraint point
      var center = target.getCenterPoint(),
          actualMouseByCenter = target.toLocalPoint(new fabric.Point(x, y),
'center', 'center')[by],
          lastMouseByCenter = target.toLocalPoint(new fabric.Point(t.lastX,
t.lastY), 'center', 'center')[by],
          actualMouseByOrigin, constraintPosition, dim =
target._getTransformedDimensions();

      this._changeSkewTransformOrigin(actualMouseByCenter - lastMouseByCenter,
t, by);
      actualMouseByOrigin = target.toLocalPoint(new fabric.Point(x, y),
t.originX, t.originY)[by];
      constraintPosition = target.translateToOriginPoint(center, t.originX,
t.originY);
      // Actually skew the object
      skewed = this._setObjectSkew(actualMouseByOrigin, t, by, dim);
      t.lastX = x;
      t.lastY = y;
      // Make sure the constraints apply
      target.setPositionByOrigin(constraintPosition, t.originX, t.originY);
      return skewed;
    },

    /**
     * Set object skew
     * @private
     * @return {Boolean} true if the skewing occurred
     */
    _setObjectSkew: function(localMouse, transform, by, _dim) {
      var target = transform.target, newValue, skewed = false,
          skewSign = transform.skewSign, newDim, dimNoSkew,
```

```
          otherBy, _otherBy, _by, newDimMouse, skewX, skewY;

      if (by === 'x') {
        otherBy = 'y';
        _otherBy = 'Y';
        _by = 'X';
        skewX = 0;
        skewY = target.skewY;
      }
      else {
        otherBy = 'x';
        _otherBy = 'X';
        _by = 'Y';
        skewX = target.skewX;
        skewY = 0;
      }

      dimNoSkew = target._getTransformedDimensions(skewX, skewY);
      newDimMouse = 2 * Math.abs(localMouse) - dimNoSkew[by];
      if (newDimMouse <= 2) {
        newValue = 0;
      }
      else {
        newValue = skewSign * Math.atan((newDimMouse / target['scale' + _by]) /
                                        (dimNoSkew[otherBy] / target['scale' +
_otherBy]));
        newValue = fabric.util.radiansToDegrees(newValue);
      }
      skewed = target['skew' + _by] !== newValue;
      target.set('skew' + _by, newValue);
      if (target['skew' + _otherBy] !== 0) {
        newDim = target._getTransformedDimensions();
        newValue = (_dim[otherBy] / newDim[otherBy]) * target['scale' +
_otherBy];
        target.set('scale' + _otherBy, newValue);
      }
      return skewed;
    },

    /**
     * Scales object by invoking its scaleX/scaleY methods
     * @private
     * @param {Number} x pointer's x coordinate
     * @param {Number} y pointer's y coordinate
     * @param {String} by Either 'x' or 'y' - specifies dimension constraint by
which to scale an object.
     *                    When not provided, an object is scaled by both
dimensions equally
     * @return {Boolean} true if the scaling occurred
     */
    _scaleObject: function (x, y, by) {
      var t = this._currentTransform,
          target = t.target,
          lockScalingX = target.get('lockScalingX'),
          lockScalingY = target.get('lockScalingY'),
          lockScalingFlip = target.get('lockScalingFlip');

      if (lockScalingX && lockScalingY) {
        return false;
      }

      // Get the constraint point
      var constraintPosition =
target.translateToOriginPoint(target.getCenterPoint(), t.originX, t.originY),
```

```
          localMouse = target.toLocalPoint(new fabric.Point(x, y), t.originX,
t.originY),
          dim = target._getTransformedDimensions(), scaled = false;

      this._setLocalMouse(localMouse, t);

      // Actually scale the object
      scaled = this._setObjectScale(localMouse, t, lockScalingX, lockScalingY,
by, lockScalingFlip, dim);

      // Make sure the constraints apply
      target.setPositionByOrigin(constraintPosition, t.originX, t.originY);
      return scaled;
    },

    /**
     * @private
     * @return {Boolean} true if the scaling occurred
     */
    _setObjectScale: function(localMouse, transform, lockScalingX, lockScalingY,
by, lockScalingFlip, _dim) {
      var target = transform.target, forbidScalingX = false, forbidScalingY =
false, scaled = false,
          changeX, changeY, scaleX, scaleY;

      scaleX = localMouse.x * target.scaleX / _dim.x;
      scaleY = localMouse.y * target.scaleY / _dim.y;
      changeX = target.scaleX !== scaleX;
      changeY = target.scaleY !== scaleY;

      if (lockScalingFlip && scaleX <= 0 && scaleX < target.scaleX) {
        forbidScalingX = true;
      }

      if (lockScalingFlip && scaleY <= 0 && scaleY < target.scaleY) {
        forbidScalingY = true;
      }

      if (by === 'equally' && !lockScalingX && !lockScalingY) {
        forbidScalingX || forbidScalingY || (scaled =
this._scaleObjectEqually(localMouse, target, transform, _dim));
      }
      else if (!by) {
        forbidScalingX || lockScalingX || (target.set('scaleX', scaleX) &&
(scaled = scaled || changeX));
        forbidScalingY || lockScalingY || (target.set('scaleY', scaleY) &&
(scaled = scaled || changeY));
      }
      else if (by === 'x' && !target.get('lockUniScaling')) {
        forbidScalingX || lockScalingX || (target.set('scaleX', scaleX) &&
(scaled = scaled || changeX));
      }
      else if (by === 'y' && !target.get('lockUniScaling')) {
        forbidScalingY || lockScalingY || (target.set('scaleY', scaleY) &&
(scaled = scaled || changeY));
      }
      transform.newScaleX = scaleX;
      transform.newScaleY = scaleY;
      forbidScalingX || forbidScalingY || this._flipObject(transform, by);
      return scaled;
    },

    /**
     * @private
```

```
     * @return {Boolean} true if the scaling occurred
     */
    _scaleObjectEqually: function(localMouse, target, transform, _dim) {

      var dist = localMouse.y + localMouse.x,
          lastDist = _dim.y * transform.original.scaleY / target.scaleY +
                     _dim.x * transform.original.scaleX / target.scaleX,
          scaled;

      // We use transform.scaleX/Y instead of target.scaleX/Y
      // because the object may have a min scale and we'll loose the proportions
      transform.newScaleX = transform.original.scaleX * dist / lastDist;
      transform.newScaleY = transform.original.scaleY * dist / lastDist;
      scaled = transform.newScaleX !== target.scaleX || transform.newScaleY !==
target.scaleY;
      target.set('scaleX', transform.newScaleX);
      target.set('scaleY', transform.newScaleY);
      return scaled;
    },

    /**
     * @private
     */
    _flipObject: function(transform, by) {
      if (transform.newScaleX < 0 && by !== 'y') {
        if (transform.originX === 'left') {
          transform.originX = 'right';
        }
        else if (transform.originX === 'right') {
          transform.originX = 'left';
        }
      }

      if (transform.newScaleY < 0 && by !== 'x') {
        if (transform.originY === 'top') {
          transform.originY = 'bottom';
        }
        else if (transform.originY === 'bottom') {
          transform.originY = 'top';
        }
      }
    },

    /**
     * @private
     */
    _setLocalMouse: function(localMouse, t) {
      var target = t.target, zoom = this.getZoom(),
          padding = target.padding / zoom;

      if (t.originX === 'right') {
        localMouse.x *= -1;
      }
      else if (t.originX === 'center') {
        localMouse.x *= t.mouseXSign * 2;
        if (localMouse.x < 0) {
          t.mouseXSign = -t.mouseXSign;
        }
      }

      if (t.originY === 'bottom') {
        localMouse.y *= -1;
      }
      else if (t.originY === 'center') {
```

```
    localMouse.y *= t.mouseYSign * 2;
    if (localMouse.y < 0) {
      t.mouseYSign = -t.mouseYSign;
    }
  }

  // adjust the mouse coordinates when dealing with padding
  if (abs(localMouse.x) > padding) {
    if (localMouse.x < 0) {
      localMouse.x += padding;
    }
    else {
      localMouse.x -= padding;
    }
  }
  else { // mouse is within the padding, set to 0
    localMouse.x = 0;
  }

  if (abs(localMouse.y) > padding) {
    if (localMouse.y < 0) {
      localMouse.y += padding;
    }
    else {
      localMouse.y -= padding;
    }
  }
  else {
    localMouse.y = 0;
  }
},

/**
 * Rotates object by invoking its rotate method
 * @private
 * @param {Number} x pointer's x coordinate
 * @param {Number} y pointer's y coordinate
 * @return {Boolean} true if the rotation occurred
 */
_rotateObject: function (x, y) {

  var t = this._currentTransform;

  if (t.target.get('lockRotation')) {
    return false;
  }

  var lastAngle = atan2(t.ey - t.top, t.ex - t.left),
      curAngle = atan2(y - t.top, x - t.left),
      angle = radiansToDegrees(curAngle - lastAngle + t.theta),
      hasRoated = true;

  if (t.target.snapAngle > 0) {
    var snapAngle  = t.target.snapAngle,
        snapThreshold  = t.target.snapThreshold || snapAngle,
        rightAngleLocked = Math.ceil(angle / snapAngle) * snapAngle,
        leftAngleLocked = Math.floor(angle / snapAngle) * snapAngle;

    if (Math.abs(angle - leftAngleLocked) < snapThreshold) {
      angle = leftAngleLocked;
    }
    else if (Math.abs(angle - rightAngleLocked) < snapThreshold) {
      angle = rightAngleLocked;
    }
```

```
  }

  // normalize angle to positive value
  if (angle < 0) {
    angle = 360 + angle;
  }
  angle %= 360;

  if (t.target.angle === angle) {
    hasRoated = false;
  }
  else {
    t.target.angle = angle;
  }

  return hasRoated;
},

/**
 * Set the cursor type of the canvas element
 * @param {String} value Cursor type of the canvas element.
 * @see http://www.w3.org/TR/css3-ui/#cursor
 */
setCursor: function (value) {
  this.upperCanvasEl.style.cursor = value;
},

/**
 * @param {fabric.Object} target to reset transform
 * @private
 */
_resetObjectTransform: function (target) {
  target.scaleX = 1;
  target.scaleY = 1;
  target.skewX = 0;
  target.skewY = 0;
  target.setAngle(0);
},

/**
 * @private
 * @param {CanvasRenderingContext2D} ctx to draw the selection on
 */
_drawSelection: function (ctx) {
  var groupSelector = this._groupSelector,
      left = groupSelector.left,
      top = groupSelector.top,
      aleft = abs(left),
      atop = abs(top);

  if (this.selectionColor) {
    ctx.fillStyle = this.selectionColor;

    ctx.fillRect(
      groupSelector.ex - ((left > 0) ? 0 : -left),
      groupSelector.ey - ((top > 0) ? 0 : -top),
      aleft,
      atop
    );
  }

  if (!this.selectionLineWidth || !this.selectionBorderColor) {
    return;
  }
```

```
      ctx.lineWidth = this.selectionLineWidth;
      ctx.strokeStyle = this.selectionBorderColor;

      // selection border
      if (this.selectionDashArray.length > 1 && !supportLineDash) {

        var px = groupSelector.ex + STROKE_OFFSET - ((left > 0) ? 0 : aleft),
            py = groupSelector.ey + STROKE_OFFSET - ((top > 0) ? 0 : atop);

        ctx.beginPath();

        fabric.util.drawDashedLine(ctx, px, py, px + aleft, py,
this.selectionDashArray);
        fabric.util.drawDashedLine(ctx, px, py + atop - 1, px + aleft, py + atop
- 1, this.selectionDashArray);
        fabric.util.drawDashedLine(ctx, px, py, px, py + atop,
this.selectionDashArray);
        fabric.util.drawDashedLine(ctx, px + aleft - 1, py, px + aleft - 1, py +
atop, this.selectionDashArray);

        ctx.closePath();
        ctx.stroke();
      }
      else {
        fabric.Object.prototype._setLineDash.call(this, ctx,
this.selectionDashArray);
        ctx.strokeRect(
          groupSelector.ex + STROKE_OFFSET - ((left > 0) ? 0 : aleft),
          groupSelector.ey + STROKE_OFFSET - ((top > 0) ? 0 : atop),
          aleft,
          atop
        );
      }
    },

    /**
     * Method that determines what object we are clicking on
     * the skipGroup parameter is for internal use, is needed for shift+click
action
     * @param {Event} e mouse event
     * @param {Boolean} skipGroup when true, activeGroup is skipped and only
objects are traversed through
     */
    findTarget: function (e, skipGroup) {
      if (this.skipTargetFind) {
        return;
      }

      var ignoreZoom = true,
          pointer = this.getPointer(e, ignoreZoom),
          activeGroup = this.getActiveGroup(),
          activeObject = this.getActiveObject(),
          activeTarget, activeTargetSubs;
      // first check current group (if one exists)
      // active group does not check sub targets like normal groups.
      // if active group just exits.
      this.targets = [];
      if (activeGroup && !skipGroup && activeGroup ===
this._searchPossibleTargets([activeGroup], pointer)) {
        this._fireOverOutEvents(activeGroup, e);
        return activeGroup;
      }
      // if we hit the corner of an activeObject, let's return that.
      if (activeObject && activeObject._findTargetCorner(pointer)) {
```

```
        this._fireOverOutEvents(activeObject, e);
        return activeObject;
      }
      if (activeObject && activeObject ===
this._searchPossibleTargets([activeObject], pointer)) {
        if (!this.preserveObjectStacking) {
          this._fireOverOutEvents(activeObject, e);
          return activeObject;
        }
        else {
          activeTarget = activeObject;
          activeTargetSubs = this.targets;
          this.targets = [];
        }
      }

      var target = this._searchPossibleTargets(this._objects, pointer);
      if (e[this.altSelectionKey] && target && activeTarget && target !==
activeTarget) {
        target = activeTarget;
        this.targets = activeTargetSubs;
      }
      this._fireOverOutEvents(target, e);
      return target;
    },

    /**
     * @private
     */
    _fireOverOutEvents: function(target, e) {
      var overOpt, outOpt, hoveredTarget = this._hoveredTarget;
      if (hoveredTarget !== target) {
        overOpt = { e: e, target: target, previousTarget: this._hoveredTarget };
        outOpt = { e: e, target: this._hoveredTarget, nextTarget: target };
        this._hoveredTarget = target;
      }
      if (target) {
        if (hoveredTarget !== target) {
          if (hoveredTarget) {
            this.fire('mouse:out', outOpt);
            hoveredTarget.fire('mouseout', outOpt);
          }
          this.fire('mouse:over', overOpt);
          target.fire('mouseover', overOpt);
        }
      }
      else if (hoveredTarget) {
        this.fire('mouse:out', outOpt);
        hoveredTarget.fire('mouseout', outOpt);
      }
    },

    /**
     * @private
     */
    _checkTarget: function(pointer, obj) {
      if (obj &&
          obj.visible &&
          obj.evented &&
          this.containsPoint(null, obj, pointer)){
        if ((this.perPixelTargetFind || obj.perPixelTargetFind) && !
obj.isEditing) {
          var isTransparent = this.isTargetTransparent(obj, pointer.x,
pointer.y);
```

```
          if (!isTransparent) {
            return true;
          }
        }
        else {
          return true;
        }
      }
    },

    /**
     * @private
     */
    _searchPossibleTargets: function(objects, pointer) {

      // Cache all targets where their bounding box contains point.
      var target, i = objects.length, normalizedPointer, subTarget;
      // Do not check for currently grouped objects, since we check the parent
group itself.
      // untill we call this function specifically to search inside the
activeGroup
      while (i--) {
        if (this._checkTarget(pointer, objects[i])) {
          target = objects[i];
          if (target.type === 'group' && target.subTargetCheck) {
            normalizedPointer = this._normalizePointer(target, pointer);
            subTarget = this._searchPossibleTargets(target._objects,
normalizedPointer);
            subTarget && this.targets.push(subTarget);
          }
          break;
        }
      }
      return target;
    },

    /**
     * Returns pointer coordinates without the effect of the viewport
     * @param {Object} pointer with "x" and "y" number values
     * @return {Object} object with "x" and "y" number values
     */
    restorePointerVpt: function(pointer) {
      return fabric.util.transformPoint(
        pointer,
        fabric.util.invertTransform(this.viewportTransform)
      );
    },

    /**
     * Returns pointer coordinates relative to canvas.
     * Can return coordinates with or without viewportTransform.
     * ignoreZoom false gives back coordinates that represent
     * the point clicked on canvas element.
     * ignoreZoom true gives back coordinates after being processed
     * by the viewportTransform ( sort of coordinates of what is displayed
     * on the canvas where you are clicking.
     * To interact with your shapes top and left you want to use ignoreZoom true
     * most of the time, while ignoreZoom false will give you coordinates
     * compatible with the object.oCoords system.
     * of the time.
     * @param {Event} e
     * @param {Boolean} ignoreZoom
     * @return {Object} object with "x" and "y" number values
     */
```

```
    getPointer: function (e, ignoreZoom, upperCanvasEl) {
      if (!upperCanvasEl) {
        upperCanvasEl = this.upperCanvasEl;
      }
      var pointer = getPointer(e),
          bounds = upperCanvasEl.getBoundingClientRect(),
          boundsWidth = bounds.width || 0,
          boundsHeight = bounds.height || 0,
          cssScale;

      if (!boundsWidth || !boundsHeight ) {
        if ('top' in bounds && 'bottom' in bounds) {
          boundsHeight = Math.abs( bounds.top - bounds.bottom );
        }
        if ('right' in bounds && 'left' in bounds) {
          boundsWidth = Math.abs( bounds.right - bounds.left );
        }
      }

      this.calcOffset();

      pointer.x = pointer.x - this._offset.left;
      pointer.y = pointer.y - this._offset.top;
      if (!ignoreZoom) {
        pointer = this.restorePointerVpt(pointer);
      }

      if (boundsWidth === 0 || boundsHeight === 0) {
        // If bounds are not available (i.e. not visible), do not apply scale.
        cssScale = { width: 1, height: 1 };
      }
      else {
        cssScale = {
          width: upperCanvasEl.width / boundsWidth,
          height: upperCanvasEl.height / boundsHeight
        };
      }

      return {
        x: pointer.x * cssScale.width,
        y: pointer.y * cssScale.height
      };
    },

    /**
     * @private
     * @throws {CANVAS_INIT_ERROR} If canvas can not be initialized
     */
    _createUpperCanvas: function () {
      var lowerCanvasClass = this.lowerCanvasEl.className.replace(/\s*lower-
canvas\s*/, '');

      if (this.upperCanvasEl) {
        this.upperCanvasEl.className = '';
      }
      else {
        this.upperCanvasEl = this._createCanvasElement();
      }
      fabric.util.addClass(this.upperCanvasEl, 'upper-canvas ' +
lowerCanvasClass);

      this.wrapperEl.appendChild(this.upperCanvasEl);

      this._copyCanvasStyle(this.lowerCanvasEl, this.upperCanvasEl);
```

```
      this._applyCanvasStyle(this.upperCanvasEl);
      this.contextTop = this.upperCanvasEl.getContext('2d');
    },

    /**
     * @private
     */
    _createCacheCanvas: function () {
      this.cacheCanvasEl = this._createCanvasElement();
      this.cacheCanvasEl.setAttribute('width', this.width);
      this.cacheCanvasEl.setAttribute('height', this.height);
      this.contextCache = this.cacheCanvasEl.getContext('2d');
    },

    /**
     * @private
     */
    _initWrapperElement: function () {
      this.wrapperEl = fabric.util.wrapElement(this.lowerCanvasEl, 'div', {
        'class': this.containerClass
      });
      fabric.util.setStyle(this.wrapperEl, {
        width: this.getWidth() + 'px',
        height: this.getHeight() + 'px',
        position: 'relative'
      });
      fabric.util.makeElementUnselectable(this.wrapperEl);
    },

    /**
     * @private
     * @param {HTMLElement} element canvas element to apply styles on
     */
    _applyCanvasStyle: function (element) {
      var width = this.getWidth() || element.width,
          height = this.getHeight() || element.height;

      fabric.util.setStyle(element, {
        position: 'absolute',
        width: width + 'px',
        height: height + 'px',
        left: 0,
        top: 0,
        'touch-action': 'none'
      });
      element.width = width;
      element.height = height;
      fabric.util.makeElementUnselectable(element);
    },

    /**
     * Copys the the entire inline style from one element (fromEl) to another
(toEl)
     * @private
     * @param {Element} fromEl Element style is copied from
     * @param {Element} toEl Element copied style is applied to
     */
    _copyCanvasStyle: function (fromEl, toEl) {
      toEl.style.cssText = fromEl.style.cssText;
    },

    /**
     * Returns context of canvas where object selection is drawn
     * @return {CanvasRenderingContext2D}
```

```
     */
    getSelectionContext: function() {
      return this.contextTop;
    },

    /**
     * Returns &lt;canvas> element on which object selection is drawn
     * @return {HTMLCanvasElement}
     */
    getSelectionElement: function () {
      return this.upperCanvasEl;
    },

    /**
     * @private
     * @param {Object} object
     */
    _setActiveObject: function(object) {
      var obj = this._activeObject;
      if (obj) {
        obj.set('active', false);
        if (object !== obj && obj.onDeselect && typeof obj.onDeselect ===
'function') {
          obj.onDeselect();
        }
      }
      this._activeObject = object;
      object.set('active', true);
    },

    /**
     * Sets given object as the only active object on canvas
     * @param {fabric.Object} object Object to set as an active one
     * @param {Event} [e] Event (passed along when firing "object:selected")
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    setActiveObject: function (object, e) {
      var currentActiveObject = this.getActiveObject();
      if (currentActiveObject && currentActiveObject !== object) {
        currentActiveObject.fire('deselected', { e: e });
      }
      this._setActiveObject(object);
      this.fire('object:selected', { target: object, e: e });
      object.fire('selected', { e: e });
      this.renderAll();
      return this;
    },

    /**
     * Returns currently active object
     * @return {fabric.Object} active object
     */
    getActiveObject: function () {
      return this._activeObject;
    },

    /**
     * @private
     * @param {fabric.Object} obj Object that was removed
     */
    _onObjectRemoved: function(obj) {
      // removing active object should fire "selection:cleared" events
      if (this.getActiveObject() === obj) {
```

```
        this.fire('before:selection:cleared', { target: obj });
        this._discardActiveObject();
        this.fire('selection:cleared', { target: obj });
        obj.fire('deselected');
      }
      if (this._hoveredTarget === obj) {
        this._hoveredTarget = null;
      }
      this.callSuper('_onObjectRemoved', obj);
    },

    /**
     * @private
     */
    _discardActiveObject: function() {
      var obj = this._activeObject;
      if (obj) {
        obj.set('active', false);
        if (obj.onDeselect && typeof obj.onDeselect === 'function') {
          obj.onDeselect();
        }
      }
      this._activeObject = null;
    },

    /**
     * Discards currently active object and fire events. If the function is
called by fabric
     * as a consequence of a mouse event, the event is passed as a parmater and
     * sent to the fire function for the custom events. When used as a method
the
     * e param does not have any application.
     * @param {event} e
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    discardActiveObject: function (e) {
      var activeObject = this._activeObject;
      if (activeObject) {
        this.fire('before:selection:cleared', { target: activeObject, e: e });
        this._discardActiveObject();
        this.fire('selection:cleared', { e: e });
        activeObject.fire('deselected', { e: e });
      }
      return this;
    },

    /**
     * @private
     * @param {fabric.Group} group
     */
    _setActiveGroup: function(group) {
      this._activeGroup = group;
      if (group) {
        group.set('active', true);
      }
    },

    /**
     * Sets active group to a specified one. If the function is called by fabric
     * as a consequence of a mouse event, the event is passed as a parmater and
     * sent to the fire function for the custom events. When used as a method
the
     * e param does not have any application.
```

```
     * @param {fabric.Group} group Group to set as a current one
     * @param {Event} e Event object
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    setActiveGroup: function (group, e) {
      this._setActiveGroup(group);
      if (group) {
        this.fire('object:selected', { target: group, e: e });
        group.fire('selected', { e: e });
      }
      return this;
    },

    /**
     * Returns currently active group
     * @return {fabric.Group} Current group
     */
    getActiveGroup: function () {
      return this._activeGroup;
    },

    /**
     * @private
     */
    _discardActiveGroup: function() {
      var g = this.getActiveGroup();
      if (g) {
        g.destroy();
      }
      this.setActiveGroup(null);
    },

    /**
     * Discards currently active group and fire events If the function is called
by fabric
     * as a consequence of a mouse event, the event is passed as a parmater and
     * sent to the fire function for the custom events. When used as a method
the
     * e param does not have any application.
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    discardActiveGroup: function (e) {
      var g = this.getActiveGroup();
      if (g) {
        this.fire('before:selection:cleared', { e: e, target: g });
        this._discardActiveGroup();
        this.fire('selection:cleared', { e: e });
      }
      return this;
    },

    /**
     * Deactivates all objects on canvas, removing any active group or object
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    deactivateAll: function () {
      var allObjects = this.getObjects(),
          i = 0,
          len = allObjects.length,
          obj;
      for ( ; i < len; i++) {
```

```
        obj = allObjects[i];
        obj && obj.set('active', false);
      }
      this._discardActiveGroup();
      this._discardActiveObject();
      return this;
    },

    /**
     * Deactivates all objects and dispatches appropriate events If the function
is called by fabric
     * as a consequence of a mouse event, the event is passed as a parmater and
     * sent to the fire function for the custom events. When used as a method
the
     * e param does not have any application.
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    deactivateAllWithDispatch: function (e) {
      var allObjects = this.getObjects(),
          i = 0,
          len = allObjects.length,
          obj;
      for ( ; i < len; i++) {
        obj = allObjects[i];
        obj && obj.set('active', false);
      }
      this.discardActiveGroup(e);
      this.discardActiveObject(e);
      return this;
    },

    /**
     * Clears a canvas element and removes all event listeners
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    dispose: function () {
      fabric.StaticCanvas.prototype.dispose.call(this);
      var wrapper = this.wrapperEl;
      this.removeListeners();
      wrapper.removeChild(this.upperCanvasEl);
      wrapper.removeChild(this.lowerCanvasEl);
      delete this.upperCanvasEl;
      if (wrapper.parentNode) {
        wrapper.parentNode.replaceChild(this.lowerCanvasEl, this.wrapperEl);
      }
      delete this.wrapperEl;
      return this;
    },

    /**
     * Clears all contexts (background, main, top) of an instance
     * @return {fabric.Canvas} thisArg
     * @chainable
     */
    clear: function () {
      this.discardActiveGroup();
      this.discardActiveObject();
      this.clearContext(this.contextTop);
      return this.callSuper('clear');
    },

    /**
```

```
     * Draws objects' controls (borders/controls)
     * @param {CanvasRenderingContext2D} ctx Context to render controls on
     */
    drawControls: function(ctx) {
      var activeGroup = this.getActiveGroup();

      if (activeGroup) {
        activeGroup._renderControls(ctx);
      }
      else {
        this._drawObjectsControls(ctx);
      }
    },

    /**
     * @private
     */
    _drawObjectsControls: function(ctx) {
      for (var i = 0, len = this._objects.length; i < len; ++i) {
        if (!this._objects[i] || !this._objects[i].active) {
          continue;
        }
        this._objects[i]._renderControls(ctx);
      }
    },

    /**
     * @private
     */
    _toObject: function(instance, methodName, propertiesToInclude) {
      //If the object is part of the current selection group, it should
      //be transformed appropriately
      //i.e. it should be serialised as it would appear if the selection group
      //were to be destroyed.
      var originalProperties = this._realizeGroupTransformOnObject(instance),
          object = this.callSuper('_toObject', instance, methodName,
propertiesToInclude);
      //Undo the damage we did by changing all of its properties
      this._unwindGroupTransformOnObject(instance, originalProperties);
      return object;
    },

    /**
     * Realises an object's group transformation on it
     * @private
     * @param {fabric.Object} [instance] the object to transform (gets mutated)
     * @returns the original values of instance which were changed
     */
    _realizeGroupTransformOnObject: function(instance) {
      if (instance.group && instance.group === this.getActiveGroup()) {
        //Copy all the positionally relevant properties across now
        var originalValues = {},
            layoutProps = ['angle', 'flipX', 'flipY', 'left', 'scaleX',
'scaleY', 'skewX', 'skewY', 'top'];
        layoutProps.forEach(function(prop) {
          originalValues[prop] = instance[prop];
        });
        this.getActiveGroup().realizeTransform(instance);
        return originalValues;
      }
      else {
        return null;
      }
    },
```

```
    /**
     * Restores the changed properties of instance
     * @private
     * @param {fabric.Object} [instance] the object to un-transform (gets
mutated)
     * @param {Object} [originalValues] the original values of instance, as
returned by _realizeGroupTransformOnObject
     */
    _unwindGroupTransformOnObject: function(instance, originalValues) {
      if (originalValues) {
        instance.set(originalValues);
      }
    },

    /**
     * @private
     */
    _setSVGObject: function(markup, instance, reviver) {
      var originalProperties;
      //If the object is in a selection group, simulate what would happen to
that
      //object when the group is deselected
      originalProperties = this._realizeGroupTransformOnObject(instance);
      this.callSuper('_setSVGObject', markup, instance, reviver);
      this._unwindGroupTransformOnObject(instance, originalProperties);
    },
  });

  // copying static properties manually to work around Opera's bug,
  // where "prototype" property is enumerable and overrides existing prototype
  for (var prop in fabric.StaticCanvas) {
    if (prop !== 'prototype') {
      fabric.Canvas[prop] = fabric.StaticCanvas[prop];
    }
  }

  if (fabric.isTouchSupported) {
    /** @ignore */
    fabric.Canvas.prototype._setCursorFromEvent = function() { };
  }

  /**
   * @ignore
   * @class fabric.Element
   * @alias fabric.Canvas
   * @deprecated Use {@link fabric.Canvas} instead.
   * @constructor
   */
  fabric.Element = fabric.Canvas;
})();


(function() {

  var cursorOffset = {
        mt: 0, // n
        tr: 1, // ne
        mr: 2, // e
        br: 3, // se
        mb: 4, // s
        bl: 5, // sw
        ml: 6, // w
        tl: 7 // nw
```

```
      },
      addListener = fabric.util.addListener,
      removeListener = fabric.util.removeListener,
      RIGHT_CLICK = 3, MIDDLE_CLICK = 2, LEFT_CLICK = 1;

  function checkClick(e, value) {
    return 'which' in e ? e.which === value : e.button === value - 1;
  }

  fabric.util.object.extend(fabric.Canvas.prototype, /** @lends
fabric.Canvas.prototype */ {

    /**
     * Map of cursor style values for each of the object controls
     * @private
     */
    cursorMap: [
      'n-resize',
      'ne-resize',
      'e-resize',
      'se-resize',
      's-resize',
      'sw-resize',
      'w-resize',
      'nw-resize'
    ],

    /**
     * Adds mouse listeners to canvas
     * @private
     */
    _initEventListeners: function () {
      // in case we initialized the class twice. This should not happen normally
      // but in some kind of applications where the canvas element may be
changed
      // this is a workaround to having double listeners.
      this.removeListeners();
      this._bindEvents();

      addListener(fabric.window, 'resize', this._onResize);

      // mouse events
      addListener(this.upperCanvasEl, 'mousedown', this._onMouseDown);
      addListener(this.upperCanvasEl, 'mousemove', this._onMouseMove);
      addListener(this.upperCanvasEl, 'mouseout', this._onMouseOut);
      addListener(this.upperCanvasEl, 'mouseenter', this._onMouseEnter);
      addListener(this.upperCanvasEl, 'wheel', this._onMouseWheel);
      addListener(this.upperCanvasEl, 'contextmenu', this._onContextMenu);

      // touch events
      addListener(this.upperCanvasEl, 'touchstart', this._onMouseDown,
{ passive: false });
      addListener(this.upperCanvasEl, 'touchmove', this._onMouseMove, { passive:
false });

      if (typeof eventjs !== 'undefined' && 'add' in eventjs) {
        eventjs.add(this.upperCanvasEl, 'gesture', this._onGesture);
        eventjs.add(this.upperCanvasEl, 'drag', this._onDrag);
        eventjs.add(this.upperCanvasEl, 'orientation',
this._onOrientationChange);
        eventjs.add(this.upperCanvasEl, 'shake', this._onShake);
        eventjs.add(this.upperCanvasEl, 'longpress', this._onLongPress);
      }
    },
```

```
    /**
     * @private
     */
    _bindEvents: function() {
      if (this.eventsBinded) {
        // for any reason we pass here twice we do not want to bind events
twice.
        return;
      }
      this._onMouseDown = this._onMouseDown.bind(this);
      this._onMouseMove = this._onMouseMove.bind(this);
      this._onMouseUp = this._onMouseUp.bind(this);
      this._onResize = this._onResize.bind(this);
      this._onGesture = this._onGesture.bind(this);
      this._onDrag = this._onDrag.bind(this);
      this._onShake = this._onShake.bind(this);
      this._onLongPress = this._onLongPress.bind(this);
      this._onOrientationChange = this._onOrientationChange.bind(this);
      this._onMouseWheel = this._onMouseWheel.bind(this);
      this._onMouseOut = this._onMouseOut.bind(this);
      this._onMouseEnter = this._onMouseEnter.bind(this);
      this._onContextMenu = this._onContextMenu.bind(this);
      this.eventsBinded = true;
    },

    /**
     * Removes all event listeners
     */
    removeListeners: function() {
      removeListener(fabric.window, 'resize', this._onResize);

      removeListener(this.upperCanvasEl, 'mousedown', this._onMouseDown);
      removeListener(this.upperCanvasEl, 'mousemove', this._onMouseMove);
      removeListener(this.upperCanvasEl, 'mouseout', this._onMouseOut);
      removeListener(this.upperCanvasEl, 'mouseenter', this._onMouseEnter);
      removeListener(this.upperCanvasEl, 'wheel', this._onMouseWheel);
      removeListener(this.upperCanvasEl, 'contextmenu', this._onContextMenu);

      removeListener(this.upperCanvasEl, 'touchstart', this._onMouseDown);
      removeListener(this.upperCanvasEl, 'touchmove', this._onMouseMove);

      if (typeof eventjs !== 'undefined' && 'remove' in eventjs) {
        eventjs.remove(this.upperCanvasEl, 'gesture', this._onGesture);
        eventjs.remove(this.upperCanvasEl, 'drag', this._onDrag);
        eventjs.remove(this.upperCanvasEl, 'orientation',
this._onOrientationChange);
        eventjs.remove(this.upperCanvasEl, 'shake', this._onShake);
        eventjs.remove(this.upperCanvasEl, 'longpress', this._onLongPress);
      }
    },

    /**
     * @private
     * @param {Event} [e] Event object fired on Event.js gesture
     * @param {Event} [self] Inner Event object
     */
    _onGesture: function(e, self) {
      this.__onTransformGesture && this.__onTransformGesture(e, self);
    },

    /**
     * @private
     * @param {Event} [e] Event object fired on Event.js drag
```

```
 * @param {Event} [self] Inner Event object
 */
_onDrag: function(e, self) {
  this.__onDrag && this.__onDrag(e, self);
},

/**
 * @private
 * @param {Event} [e] Event object fired on wheel event
 */
_onMouseWheel: function(e) {
  this.__onMouseWheel(e);
},

/**
 * @private
 * @param {Event} e Event object fired on mousedown
 */
_onMouseOut: function(e) {
  var target = this._hoveredTarget;
  this.fire('mouse:out', { target: target, e: e });
  this._hoveredTarget = null;
  target && target.fire('mouseout', { e: e });
  if (this._iTextInstances) {
    this._iTextInstances.forEach(function(obj) {
      if (obj.isEditing) {
        obj.hiddenTextarea.focus();
      }
    });
  }
},

/**
 * @private
 * @param {Event} e Event object fired on mouseenter
 */
_onMouseEnter: function(e) {
  if (!this.findTarget(e)) {
    this.fire('mouse:over', { target: null, e: e });
    this._hoveredTarget = null;
  }
},

/**
 * @private
 * @param {Event} [e] Event object fired on Event.js orientation change
 * @param {Event} [self] Inner Event object
 */
_onOrientationChange: function(e, self) {
  this.__onOrientationChange && this.__onOrientationChange(e, self);
},

/**
 * @private
 * @param {Event} [e] Event object fired on Event.js shake
 * @param {Event} [self] Inner Event object
 */
_onShake: function(e, self) {
  this.__onShake && this.__onShake(e, self);
},

/**
 * @private
 * @param {Event} [e] Event object fired on Event.js shake
```

```
     * @param {Event} [self] Inner Event object
     */
    _onLongPress: function(e, self) {
      this.__onLongPress && this.__onLongPress(e, self);
    },

    /**
     * @private
     * @param {Event} e Event object fired on mousedown
     */
    _onContextMenu: function (e) {
      if (this.stopContextMenu) {
        e.stopPropagation();
        e.preventDefault();
      }
      return false;
    },

    /**
     * @private
     * @param {Event} e Event object fired on mousedown
     */
    _onMouseDown: function (e) {
      this.__onMouseDown(e);

      addListener(fabric.document, 'touchend', this._onMouseUp, { passive: false
});
      addListener(fabric.document, 'touchmove', this._onMouseMove, { passive:
false });

      removeListener(this.upperCanvasEl, 'mousemove', this._onMouseMove);
      removeListener(this.upperCanvasEl, 'touchmove', this._onMouseMove);

      if (e.type === 'touchstart') {
        // Unbind mousedown to prevent double triggers from touch devices
        removeListener(this.upperCanvasEl, 'mousedown', this._onMouseDown);
      }
      else {
        addListener(fabric.document, 'mouseup', this._onMouseUp);
        addListener(fabric.document, 'mousemove', this._onMouseMove);
      }
    },

    /**
     * @private
     * @param {Event} e Event object fired on mouseup
     */
    _onMouseUp: function (e) {
      this.__onMouseUp(e);

      removeListener(fabric.document, 'mouseup', this._onMouseUp);
      removeListener(fabric.document, 'touchend', this._onMouseUp);

      removeListener(fabric.document, 'mousemove', this._onMouseMove);
      removeListener(fabric.document, 'touchmove', this._onMouseMove);

      addListener(this.upperCanvasEl, 'mousemove', this._onMouseMove);
      addListener(this.upperCanvasEl, 'touchmove', this._onMouseMove, { passive:
false });

      if (e.type === 'touchend') {
        // Wait 400ms before rebinding mousedown to prevent double triggers
        // from touch devices
        var _this = this;
```

```
        setTimeout(function() {
          addListener(_this.upperCanvasEl, 'mousedown', _this._onMouseDown);
        }, 400);
      }
    },

    /**
     * @private
     * @param {Event} e Event object fired on mousemove
     */
    _onMouseMove: function (e) {
      !this.allowTouchScrolling && e.preventDefault && e.preventDefault();
      this.__onMouseMove(e);
    },

    /**
     * @private
     */
    _onResize: function () {
      this.calcOffset();
    },

    /**
     * Decides whether the canvas should be redrawn in mouseup and mousedown
events.
     * @private
     * @param {Object} target
     * @param {Object} pointer
     */
    _shouldRender: function(target, pointer) {
      var activeObject = this.getActiveGroup() || this.getActiveObject();

      if (activeObject && activeObject.isEditing && target === activeObject) {
        // if we mouse up/down over a editing textbox a cursor change,
        // there is no need to re render
        return false;
      }
      return !!(
        (target && (
          target.isMoving ||
          target !== activeObject))
        ||
        (!target && !!activeObject)
        ||
        (!target && !activeObject && !this._groupSelector)
        ||
        (pointer &&
          this._previousPointer &&
          this.selection && (
          pointer.x !== this._previousPointer.x ||
          pointer.y !== this._previousPointer.y))
      );
    },

    /**
     * Method that defines the actions when mouse is released on canvas.
     * The method resets the currentTransform parameters, store the image corner
     * position in the image object and render the canvas on top.
     * @private
     * @param {Event} e Event object fired on mouseup
     */
    __onMouseUp: function (e) {

      var target;
```

```
      // if right/middle click just fire events and return
      // target undefined will make the _handleEvent search the target
      if (checkClick(e, RIGHT_CLICK)) {
        if (this.fireRightClick) {
          this._handleEvent(e, 'up', target, RIGHT_CLICK);
        }
        return;
      }

      if (checkClick(e, MIDDLE_CLICK)) {
        if (this.fireMiddleClick) {
          this._handleEvent(e, 'up', target, MIDDLE_CLICK);
        }
        return;
      }

      if (this.isDrawingMode && this._isCurrentlyDrawing) {
        this._onMouseUpInDrawingMode(e);
        return;
      }

      var searchTarget = true, transform = this._currentTransform,
          groupSelector = this._groupSelector,
          isClick = (!groupSelector || (groupSelector.left === 0 &&
groupSelector.top === 0));

      if (transform) {
        this._finalizeCurrentTransform(e);
        searchTarget = !transform.actionPerformed;
      }

      target = searchTarget ? this.findTarget(e, true) : transform.target;

      var shouldRender = this._shouldRender(target, this.getPointer(e));

      if (target || !isClick) {
        this._maybeGroupObjects(e);
      }
      else {
        // those are done by default on mouse up
        // by _maybeGroupObjects, we are skipping it in case of no target find
        this._groupSelector = null;
        this._currentTransform = null;
      }

      if (target) {
        target.isMoving = false;
      }
      this._setCursorFromEvent(e, target);
      this._handleEvent(e, 'up', target ? target : null, LEFT_CLICK, isClick);
      target && (target.__corner = 0);
      shouldRender && this.renderAll();
    },

    /**
     * @private
     * Handle event firing for target and subtargets
     * @param {Event} e event from mouse
     * @param {String} eventType event to fire (up, down or move)
     * @param {fabric.Object} targetObj receiving event
     * @param {Number} [button] button used in the event 1 = left, 2 = middle, 3
= right
     * @param {Boolean} isClick for left button only, indicates that the mouse
up happened without move.
```

```
     */
    _handleEvent: function(e, eventType, targetObj, button, isClick) {
      var target = typeof targetObj === 'undefined' ? this.findTarget(e) :
targetObj,
          targets = this.targets || [],
          options = {
            e: e,
            target: target,
            subTargets: targets,
            button: button || LEFT_CLICK,
            isClick: isClick || false
          };
      this.fire('mouse:' + eventType, options);
      target && target.fire('mouse' + eventType, options);
      for (var i = 0; i < targets.length; i++) {
        targets[i].fire('mouse' + eventType, options);
      }
    },

    /**
     * @private
     * @param {Event} e send the mouse event that generate the finalize down, so
it can be used in the event
     */
    _finalizeCurrentTransform: function(e) {

      var transform = this._currentTransform,
          target = transform.target;

      if (target._scaling) {
        target._scaling = false;
      }

      target.setCoords();
      this._restoreOriginXY(target);

      if (transform.actionPerformed || (this.stateful &&
target.hasStateChanged())) {
        this.fire('object:modified', { target: target, e: e });
        target.fire('modified', { e: e });
      }
    },

    /**
     * @private
     * @param {Object} target Object to restore
     */
    _restoreOriginXY: function(target) {
      if (this._previousOriginX && this._previousOriginY) {

        var originPoint = target.translateToOriginPoint(
          target.getCenterPoint(),
          this._previousOriginX,
          this._previousOriginY);

        target.originX = this._previousOriginX;
        target.originY = this._previousOriginY;

        target.left = originPoint.x;
        target.top = originPoint.y;

        this._previousOriginX = null;
        this._previousOriginY = null;
      }
```

```
},

/**
 * @private
 * @param {Event} e Event object fired on mousedown
 */
_onMouseDownInDrawingMode: function(e) {
  this._isCurrentlyDrawing = true;
  this.discardActiveObject(e).renderAll();
  if (this.clipTo) {
    fabric.util.clipContext(this, this.contextTop);
  }
  var pointer = this.getPointer(e);
  this.freeDrawingBrush.onMouseDown(pointer);
  this._handleEvent(e, 'down');
},

/**
 * @private
 * @param {Event} e Event object fired on mousemove
 */
_onMouseMoveInDrawingMode: function(e) {
  if (this._isCurrentlyDrawing) {
    var pointer = this.getPointer(e);
    this.freeDrawingBrush.onMouseMove(pointer);
  }
  this.setCursor(this.freeDrawingCursor);
  this._handleEvent(e, 'move');
},

/**
 * @private
 * @param {Event} e Event object fired on mouseup
 */
_onMouseUpInDrawingMode: function(e) {
  this._isCurrentlyDrawing = false;
  if (this.clipTo) {
    this.contextTop.restore();
  }
  this.freeDrawingBrush.onMouseUp();
  this._handleEvent(e, 'up');
},

/**
 * Method that defines the actions when mouse is clicked on canvas.
 * The method inits the currentTransform parameters and renders all the
 * canvas so the current image can be placed on the top canvas and the rest
 * in on the container one.
 * @private
 * @param {Event} e Event object fired on mousedown
 */
__onMouseDown: function (e) {

  var target = this.findTarget(e);

  // if right click just fire events
  if (checkClick(e, RIGHT_CLICK)) {
    if (this.fireRightClick) {
      this._handleEvent(e, 'down', target ? target : null, RIGHT_CLICK);
    }
    return;
  }

  if (checkClick(e, MIDDLE_CLICK)) {
```

```
        if (this.fireMiddleClick) {
          this._handleEvent(e, 'down', target ? target : null, MIDDLE_CLICK);
        }
        return;
      }

      if (this.isDrawingMode) {
        this._onMouseDownInDrawingMode(e);
        return;
      }

      // ignore if some object is being transformed at this moment
      if (this._currentTransform) {
        return;
      }

      // save pointer for check in __onMouseUp event
      var pointer = this.getPointer(e, true);
      this._previousPointer = pointer;

      var shouldRender = this._shouldRender(target, pointer),
          shouldGroup = this._shouldGroup(e, target);

      if (this._shouldClearSelection(e, target)) {
        this.deactivateAllWithDispatch(e);
      }
      else if (shouldGroup) {
        this._handleGrouping(e, target);
        target = this.getActiveGroup();
      }

      if (this.selection && (!target || (!target.selectable && !
target.isEditing))) {
        this._groupSelector = {
          ex: pointer.x,
          ey: pointer.y,
          top: 0,
          left: 0
        };
      }

      if (target) {
        if (target.selectable && (target.__corner || !shouldGroup)) {
          this._beforeTransform(e, target);
          this._setupCurrentTransform(e, target);
        }
        var activeObject = this.getActiveObject();
        if (target !== this.getActiveGroup() && target !== activeObject) {
          this.deactivateAll();
          if (target.selectable) {
            activeObject && activeObject.fire('deselected', { e: e });
            this.setActiveObject(target, e);
          }
        }
      }
      this._handleEvent(e, 'down', target ? target : null);
      // we must renderAll so that we update the visuals
      shouldRender && this.renderAll();
    },

    /**
     * @private
     */
    _beforeTransform: function(e, target) {
```

```
      this.stateful && target.saveState();

      // determine if it's a drag or rotate case
      if (target._findTargetCorner(this.getPointer(e))) {
        this.onBeforeScaleRotate(target);
      }

    },

    /**
     * @private
     * @param {Object} target Object for that origin is set to center
     */
    _setOriginToCenter: function(target) {
      this._previousOriginX = this._currentTransform.target.originX;
      this._previousOriginY = this._currentTransform.target.originY;

      var center = target.getCenterPoint();

      target.originX = 'center';
      target.originY = 'center';

      target.left = center.x;
      target.top = center.y;

      this._currentTransform.left = target.left;
      this._currentTransform.top = target.top;
    },

    /**
     * @private
     * @param {Object} target Object for that center is set to origin
     */
    _setCenterToOrigin: function(target) {
      var originPoint = target.translateToOriginPoint(
        target.getCenterPoint(),
        this._previousOriginX,
        this._previousOriginY);

      target.originX = this._previousOriginX;
      target.originY = this._previousOriginY;

      target.left = originPoint.x;
      target.top = originPoint.y;

      this._previousOriginX = null;
      this._previousOriginY = null;
    },

    /**
     * Method that defines the actions when mouse is hovering the canvas.
     * The currentTransform parameter will definde whether the user is
rotating/scaling/translating
     * an image or neither of them (only hovering). A group selection is also
possible and would cancel
     * all any other type of action.
     * In case of an image transformation only the top canvas will be rendered.
     * @private
     * @param {Event} e Event object fired on mousemove
     */
    __onMouseMove: function (e) {

      var target, pointer;
```

```
      if (this.isDrawingMode) {
        this._onMouseMoveInDrawingMode(e);
        return;
      }
      if (typeof e.touches !== 'undefined' && e.touches.length > 1) {
        return;
      }

      var groupSelector = this._groupSelector;

      // We initially clicked in an empty area, so we draw a box for multiple
selection
      if (groupSelector) {
        pointer = this.getPointer(e, true);

        groupSelector.left = pointer.x - groupSelector.ex;
        groupSelector.top = pointer.y - groupSelector.ey;

        this.renderTop();
      }
      else if (!this._currentTransform) {
        target = this.findTarget(e);
        this._setCursorFromEvent(e, target);
      }
      else {
        this._transformObject(e);
      }
      this._handleEvent(e, 'move', target ? target : null);
    },

    /**
     * Method that defines actions when an Event Mouse Wheel
     * @param {Event} e Event object fired on mouseup
     */
    __onMouseWheel: function(e) {
      this._handleEvent(e, 'wheel');
    },

    /**
     * @private
     * @param {Event} e Event fired on mousemove
     */
    _transformObject: function(e) {
      var pointer = this.getPointer(e),
          transform = this._currentTransform;

      transform.reset = false;
      transform.target.isMoving = true;
      transform.shiftKey = e.shiftKey;
      transform.altKey = e[this.centeredKey];

      this._beforeScaleTransform(e, transform);
      this._performTransformAction(e, transform, pointer);

      transform.actionPerformed && this.renderAll();
    },

    /**
     * @private
     */
    _performTransformAction: function(e, transform, pointer) {
      var x = pointer.x,
          y = pointer.y,
          target = transform.target,
```

```
          action = transform.action,
          actionPerformed = false;

      if (action === 'rotate') {
        (actionPerformed = this._rotateObject(x, y)) && this._fire('rotating',
target, e);
      }
      else if (action === 'scale') {
        (actionPerformed = this._onScale(e, transform, x, y)) &&
this._fire('scaling', target, e);
      }
      else if (action === 'scaleX') {
        (actionPerformed = this._scaleObject(x, y, 'x')) &&
this._fire('scaling', target, e);
      }
      else if (action === 'scaleY') {
        (actionPerformed = this._scaleObject(x, y, 'y')) &&
this._fire('scaling', target, e);
      }
      else if (action === 'skewX') {
        (actionPerformed = this._skewObject(x, y, 'x')) && this._fire('skewing',
target, e);
      }
      else if (action === 'skewY') {
        (actionPerformed = this._skewObject(x, y, 'y')) && this._fire('skewing',
target, e);
      }
      else {
        actionPerformed = this._translateObject(x, y);
        if (actionPerformed) {
          this._fire('moving', target, e);
          this.setCursor(target.moveCursor || this.moveCursor);
        }
      }
      transform.actionPerformed = transform.actionPerformed || actionPerformed;
    },

    /**
     * @private
     */
    _fire: function(eventName, target, e) {
      this.fire('object:' + eventName, { target: target, e: e });
      target.fire(eventName, { e: e });
    },

    /**
     * @private
     */
    _beforeScaleTransform: function(e, transform) {
      if (transform.action === 'scale' || transform.action === 'scaleX' ||
transform.action === 'scaleY') {
        var centerTransform = this._shouldCenterTransform(transform.target);

        // Switch from a normal resize to center-based
        if ((centerTransform && (transform.originX !== 'center' ||
transform.originY !== 'center')) ||
            // Switch from center-based resize to normal one
            (!centerTransform && transform.originX === 'center' &&
transform.originY === 'center')
        ) {
          this._resetCurrentTransform();
          transform.reset = true;
        }
      }
```

```
    },

    /**
     * @private
     * @param {Event} e Event object
     * @param {Object} transform current tranform
     * @param {Number} x mouse position x from origin
     * @param {Number} y mouse poistion y from origin
     * @return {Boolean} true if the scaling occurred
     */
    _onScale: function(e, transform, x, y) {
      if ((e[this.uniScaleKey] || this.uniScaleTransform) && !
transform.target.get('lockUniScaling')) {
        transform.currentAction = 'scale';
        return this._scaleObject(x, y);
      }
      else {
        // Switch from a normal resize to proportional
        if (!transform.reset && transform.currentAction === 'scale') {
          this._resetCurrentTransform();
        }

        transform.currentAction = 'scaleEqually';
        return this._scaleObject(x, y, 'equally');
      }
    },

    /**
     * Sets the cursor depending on where the canvas is being hovered.
     * Note: very buggy in Opera
     * @param {Event} e Event object
     * @param {Object} target Object that the mouse is hovering, if so.
     */
    _setCursorFromEvent: function (e, target) {
      if (!target) {
        this.setCursor(this.defaultCursor);
        return false;
      }

      var hoverCursor = target.hoverCursor || this.hoverCursor,
          activeGroup = this.getActiveGroup(),
          // only show proper corner when group selection is not active
          corner = target._findTargetCorner
                    && (!activeGroup || !activeGroup.contains(target))
                    && target._findTargetCorner(this.getPointer(e, true));

      if (!corner) {
        this.setCursor(hoverCursor);
      }
      else {
        this._setCornerCursor(corner, target, e);
      }
      //actually unclear why it should return something
      //is never evaluated
      return true;
    },

    /**
     * @private
     */
    _setCornerCursor: function(corner, target, e) {
      if (corner in cursorOffset) {
        this.setCursor(this._getRotatedCornerCursor(corner, target, e));
      }
```

```
      else if (corner === 'mtr' && target.hasRotatingPoint) {
        this.setCursor(this.rotationCursor);
      }
      else {
        this.setCursor(this.defaultCursor);
        return false;
      }
    },

    /**
     * @private
     */
    _getRotatedCornerCursor: function(corner, target, e) {
      var n = Math.round((target.getAngle() % 360) / 45);

      if (n < 0) {
        n += 8; // full circle ahead
      }
      n += cursorOffset[corner];
      if (e[this.altActionKey] && cursorOffset[corner] % 2 === 0) {
        //if we are holding shift and we are on a mx corner...
        n += 2;
      }
      // normalize n to be from 0 to 7
      n %= 8;

      return this.cursorMap[n];
    }
  });
})();


(function() {

  var min = Math.min,
      max = Math.max;

  fabric.util.object.extend(fabric.Canvas.prototype, /** @lends
fabric.Canvas.prototype */ {

    /**
     * @private
     * @param {Event} e Event object
     * @param {fabric.Object} target
     * @return {Boolean}
     */
    _shouldGroup: function(e, target) {
      var activeObject = this.getActiveObject();
      return e[this.selectionKey] && target && target.selectable &&
            (this.getActiveGroup() || (activeObject && activeObject !== target))
            && this.selection;
    },

    /**
     * @private
     * @param {Event} e Event object
     * @param {fabric.Object} target
     */
    _handleGrouping: function (e, target) {
      var activeGroup = this.getActiveGroup();

      if (target === activeGroup) {
        // if it's a group, find target again, using activeGroup objects
        target = this.findTarget(e, true);
```

```
    // if even object is not found, bail out
    if (!target) {
      return;
    }
  }
  if (activeGroup) {
    this._updateActiveGroup(target, e);
  }
  else {
    this._createActiveGroup(target, e);
  }

  if (this._activeGroup) {
    this._activeGroup.saveCoords();
  }
},

/**
 * @private
 */
_updateActiveGroup: function(target, e) {
  var activeGroup = this.getActiveGroup();

  if (activeGroup.contains(target)) {

    activeGroup.removeWithUpdate(target);
    target.set('active', false);

    if (activeGroup.size() === 1) {
      // remove group alltogether if after removal it only contains 1 object
      this.discardActiveGroup(e);
      // activate last remaining object
      this.setActiveObject(activeGroup.item(0), e);
      return;
    }
  }
  else {
    activeGroup.addWithUpdate(target);
  }
  this.fire('selection:created', { target: activeGroup, e: e });
  activeGroup.set('active', true);
},

/**
 * @private
 */
_createActiveGroup: function(target, e) {

  if (this._activeObject && target !== this._activeObject) {

    var group = this._createGroup(target);
    group.addWithUpdate();

    this.setActiveGroup(group, e);
    this._activeObject = null;

    this.fire('selection:created', { target: group, e: e });
  }

  target.set('active', true);
},

/**
 * @private
```

```
     * @param {Object} target
     */
    _createGroup: function(target) {

      var objects = this.getObjects(),
          isActiveLower = objects.indexOf(this._activeObject) <
objects.indexOf(target),
          groupObjects = isActiveLower
            ? [this._activeObject, target]
            : [target, this._activeObject];
      this._activeObject.isEditing && this._activeObject.exitEditing();
      return new fabric.Group(groupObjects, {
        canvas: this
      });
    },

    /**
     * @private
     * @param {Event} e mouse event
     */
    _groupSelectedObjects: function (e) {

      var group = this._collectObjects();

      // do not create group for 1 element only
      if (group.length === 1) {
        this.setActiveObject(group[0], e);
      }
      else if (group.length > 1) {
        group = new fabric.Group(group.reverse(), {
          canvas: this
        });
        group.addWithUpdate();
        this.setActiveGroup(group, e);
        group.saveCoords();
        this.fire('selection:created', { target: group, e: e });
        this.renderAll();
      }
    },

    /**
     * @private
     */
    _collectObjects: function() {
      var group = [],
          currentObject,
          x1 = this._groupSelector.ex,
          y1 = this._groupSelector.ey,
          x2 = x1 + this._groupSelector.left,
          y2 = y1 + this._groupSelector.top,
          selectionX1Y1 = new fabric.Point(min(x1, x2), min(y1, y2)),
          selectionX2Y2 = new fabric.Point(max(x1, x2), max(y1, y2)),
          isClick = x1 === x2 && y1 === y2;

      for (var i = this._objects.length; i--; ) {
        currentObject = this._objects[i];

        if (!currentObject || !currentObject.selectable || !
currentObject.visible) {
          continue;
        }

        if (currentObject.intersectsWithRect(selectionX1Y1, selectionX2Y2) ||
            currentObject.isContainedWithinRect(selectionX1Y1, selectionX2Y2) ||
```

```
            currentObject.containsPoint(selectionX1Y1) ||
            currentObject.containsPoint(selectionX2Y2)
          ) {
            currentObject.set('active', true);
            group.push(currentObject);

            // only add one object if it's a click
            if (isClick) {
              break;
            }
          }
        }

        return group;
      },

      /**
       * @private
       */
      _maybeGroupObjects: function(e) {
        if (this.selection && this._groupSelector) {
          this._groupSelectedObjects(e);
        }

        var activeGroup = this.getActiveGroup();
        if (activeGroup) {
          activeGroup.setObjectsCoords().setCoords();
          activeGroup.isMoving = false;
          this.setCursor(this.defaultCursor);
        }

        // clear selection and current transformation
        this._groupSelector = null;
        this._currentTransform = null;
      }
    });

})();


(function () {

  var supportQuality = fabric.StaticCanvas.supports('toDataURLWithQuality');

  fabric.util.object.extend(fabric.StaticCanvas.prototype, /** @lends
fabric.StaticCanvas.prototype */ {

    /**
     * Exports canvas element to a dataurl image. Note that when multiplier is
used, cropping is scaled appropriately
     * @param {Object} [options] Options object
     * @param {String} [options.format=png] The format of the output image.
Either "jpeg" or "png"
     * @param {Number} [options.quality=1] Quality level (0..1). Only used for
jpeg.
     * @param {Number} [options.multiplier=1] Multiplier to scale by
     * @param {Number} [options.left] Cropping left offset. Introduced in
v1.2.14
     * @param {Number} [options.top] Cropping top offset. Introduced in v1.2.14
     * @param {Number} [options.width] Cropping width. Introduced in v1.2.14
     * @param {Number} [options.height] Cropping height. Introduced in v1.2.14
     * @return {String} Returns a data: URL containing a representation of the
object in the format specified by options.format
     * @see {@link http://jsfiddle.net/fabricjs/NfZVb/|jsFiddle demo}
```

```
     * @example <caption>Generate jpeg dataURL with lower quality</caption>
     * var dataURL = canvas.toDataURL({
     *   format: 'jpeg',
     *   quality: 0.8
     * });
     * @example <caption>Generate cropped png dataURL (clipping of
canvas)</caption>
     * var dataURL = canvas.toDataURL({
     *   format: 'png',
     *   left: 100,
     *   top: 100,
     *   width: 200,
     *   height: 200
     * });
     * @example <caption>Generate double scaled png dataURL</caption>
     * var dataURL = canvas.toDataURL({
     *   format: 'png',
     *   multiplier: 2
     * });
     */
    toDataURL: function (options) {
      options || (options = { });

      var format = options.format || 'png',
          quality = options.quality || 1,
          multiplier = options.multiplier || 1,
          cropping = {
            left: options.left || 0,
            top: options.top || 0,
            width: options.width || 0,
            height: options.height || 0,
          };
      return this.__toDataURLWithMultiplier(format, quality, cropping,
multiplier);
    },

    /**
     * @private
     */
    __toDataURLWithMultiplier: function(format, quality, cropping, multiplier) {

      var origWidth = this.getWidth(),
          origHeight = this.getHeight(),
          scaledWidth = (cropping.width || this.getWidth()) * multiplier,
          scaledHeight = (cropping.height || this.getHeight()) * multiplier,
          zoom = this.getZoom(),
          newZoom = zoom * multiplier,
          vp = this.viewportTransform,
          translateX = (vp[4] - cropping.left) * multiplier,
          translateY = (vp[5] - cropping.top) * multiplier,
          newVp = [newZoom, 0, 0, newZoom, translateX, translateY],
          originalInteractive = this.interactive;

      this.viewportTransform = newVp;
      // setting interactive to false avoid exporting controls
      this.interactive && (this.interactive = false);
      if (origWidth !== scaledWidth || origHeight !== scaledHeight) {
        // this.setDimensions is going to renderAll also;
        this.setDimensions({ width: scaledWidth, height: scaledHeight });
      }
      else {
        this.renderAll();
      }
      var data = this.__toDataURL(format, quality, cropping);
```

```
      originalInteractive && (this.interactive = originalInteractive);
      this.viewportTransform = vp;
      //setDimensions with no option object is taking care of:
      //this.width, this.height, this.renderAll()
      this.setDimensions({ width: origWidth, height: origHeight });
      return data;
    },

    /**
     * @private
     */
    __toDataURL: function(format, quality) {

      var canvasEl = this.contextContainer.canvas;
      // to avoid common confusion
https://github.com/kangax/fabric.js/issues/806
      if (format === 'jpg') {
        format = 'jpeg';
      }

      var data = supportQuality
                ? canvasEl.toDataURL('image/' + format, quality)
                : canvasEl.toDataURL('image/' + format);

      return data;
    },

    /**
     * Exports canvas element to a dataurl image (allowing to change image size
via multiplier).
     * @deprecated since 1.0.13
     * @param {String} format (png|jpeg)
     * @param {Number} multiplier
     * @param {Number} quality (0..1)
     * @return {String}
     */
    toDataURLWithMultiplier: function (format, multiplier, quality) {
      return this.toDataURL({
        format: format,
        multiplier: multiplier,
        quality: quality
      });
    },
  });

})();


fabric.util.object.extend(fabric.StaticCanvas.prototype, /** @lends
fabric.StaticCanvas.prototype */ {

  /**
   * Populates canvas with data from the specified dataless JSON.
   * JSON format must conform to the one of {@link fabric.Canvas#toDatalessJSON}
   * @deprecated since 1.2.2
   * @param {String|Object} json JSON string or object
   * @param {Function} callback Callback, invoked when json is parsed
   *                            and corresponding objects (e.g: {@link
fabric.Image})
   *                            are initialized
   * @param {Function} [reviver] Method for further parsing of JSON elements,
called after each fabric object created.
   * @return {fabric.Canvas} instance
   * @chainable
```

```
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-3#deserialization}
   */
  loadFromDatalessJSON: function (json, callback, reviver) {
    return this.loadFromJSON(json, callback, reviver);
  },

  /**
   * Populates canvas with data from the specified JSON.
   * JSON format must conform to the one of {@link fabric.Canvas#toJSON}
   * @param {String|Object} json JSON string or object
   * @param {Function} callback Callback, invoked when json is parsed
   *                            and corresponding objects (e.g: {@link
fabric.Image})
   *                            are initialized
   * @param {Function} [reviver] Method for further parsing of JSON elements,
called after each fabric object created.
   * @return {fabric.Canvas} instance
   * @chainable
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-3#deserialization}
   * @see {@link http://jsfiddle.net/fabricjs/fmgXt/|jsFiddle demo}
   * @example <caption>loadFromJSON</caption>
   * canvas.loadFromJSON(json, canvas.renderAll.bind(canvas));
   * @example <caption>loadFromJSON with reviver</caption>
   * canvas.loadFromJSON(json, canvas.renderAll.bind(canvas), function(o,
object) {
   *   // `o` = json object
   *   // `object` = fabric.Object instance
   *   // ... do some stuff ...
   * });
   */
  loadFromJSON: function (json, callback, reviver) {
    if (!json) {
      return;
    }

    // serialize if it wasn't already
    var serialized = (typeof json === 'string')
      ? JSON.parse(json)
      : fabric.util.object.clone(json);

    var _this = this,
        renderOnAddRemove = this.renderOnAddRemove;
    this.renderOnAddRemove = false;

    this._enlivenObjects(serialized.objects, function (enlivenedObjects) {
      _this.clear();
      _this._setBgOverlay(serialized, function () {
        enlivenedObjects.forEach(function(obj, index) {
          // we splice the array just in case some custom classes restored from
JSON
          // will add more object to canvas at canvas init.
          _this.insertAt(obj, index);
        });
        _this.renderOnAddRemove = renderOnAddRemove;
        // remove parts i cannot set as options
        delete serialized.objects;
        delete serialized.backgroundImage;
        delete serialized.overlayImage;
        delete serialized.background;
        delete serialized.overlay;
        // this._initOptions does too many things to just
        // call it. Normally loading an Object from JSON
        // create the Object instance. Here the Canvas is
        // already an instance and we are just loading things over it
```

```
        _this._setOptions(serialized);
        _this.renderAll();
        callback && callback();
      });
    }, reviver);
    return this;
  },

  /**
   * @private
   * @param {Object} serialized Object with background and overlay information
   * @param {Function} callback Invoked after all background and overlay
images/patterns loaded
   */
  _setBgOverlay: function(serialized, callback) {
    var loaded = {
      backgroundColor: false,
      overlayColor: false,
      backgroundImage: false,
      overlayImage: false
    };

    if (!serialized.backgroundImage && !serialized.overlayImage && !
serialized.background && !serialized.overlay) {
      callback && callback();
      return;
    }

    var cbIfLoaded = function () {
      if (loaded.backgroundImage && loaded.overlayImage &&
loaded.backgroundColor && loaded.overlayColor) {
        callback && callback();
      }
    };

    this.__setBgOverlay('backgroundImage', serialized.backgroundImage, loaded,
cbIfLoaded);
    this.__setBgOverlay('overlayImage', serialized.overlayImage, loaded,
cbIfLoaded);
    this.__setBgOverlay('backgroundColor', serialized.background, loaded,
cbIfLoaded);
    this.__setBgOverlay('overlayColor', serialized.overlay, loaded, cbIfLoaded);
  },

  /**
   * @private
   * @param {String} property Property to set (backgroundImage, overlayImage,
backgroundColor, overlayColor)
   * @param {(Object|String)} value Value to set
   * @param {Object} loaded Set loaded property to true if property is set
   * @param {Object} callback Callback function to invoke after property is set
   */
  __setBgOverlay: function(property, value, loaded, callback) {
    var _this = this;

    if (!value) {
      loaded[property] = true;
      callback && callback();
      return;
    }

    if (property === 'backgroundImage' || property === 'overlayImage') {
      fabric.util.enlivenObjects([value], function(enlivedObject){
        _this[property] = enlivedObject[0];
```

```
        loaded[property] = true;
        callback && callback();
      });
    }
    else {
      this['set' + fabric.util.string.capitalize(property, true)](value,
function() {
        loaded[property] = true;
        callback && callback();
      });
    }
  },

  /**
   * @private
   * @param {Array} objects
   * @param {Function} callback
   * @param {Function} [reviver]
   */
  _enlivenObjects: function (objects, callback, reviver) {
    if (!objects || objects.length === 0) {
      callback && callback([]);
      return;
    }

    fabric.util.enlivenObjects(objects, function(enlivenedObjects) {
      callback && callback(enlivenedObjects);
    }, null, reviver);
  },

  /**
   * @private
   * @param {String} format
   * @param {Function} callback
   */
  _toDataURL: function (format, callback) {
    this.clone(function (clone) {
      callback(clone.toDataURL(format));
    });
  },

  /**
   * @private
   * @param {String} format
   * @param {Number} multiplier
   * @param {Function} callback
   */
  _toDataURLWithMultiplier: function (format, multiplier, callback) {
    this.clone(function (clone) {
      callback(clone.toDataURLWithMultiplier(format, multiplier));
    });
  },

  /**
   * Clones canvas instance
   * @param {Object} [callback] Receives cloned instance as a first argument
   * @param {Array} [properties] Array of properties to include in the cloned
canvas and children
   */
  clone: function (callback, properties) {
    var data = JSON.stringify(this.toJSON(properties));
    this.cloneWithoutData(function(clone) {
      clone.loadFromJSON(data, function() {
        callback && callback(clone);
```

```
      });
    });
  },

  /**
   * Clones canvas instance without cloning existing data.
   * This essentially copies canvas dimensions, clipping properties, etc.
   * but leaves data empty (so that you can populate it with your own)
   * @param {Object} [callback] Receives cloned instance as a first argument
   */
  cloneWithoutData: function(callback) {
    var el = fabric.document.createElement('canvas');

    el.width = this.getWidth();
    el.height = this.getHeight();

    var clone = new fabric.Canvas(el);
    clone.clipTo = this.clipTo;
    if (this.backgroundImage) {
      clone.setBackgroundImage(this.backgroundImage.src, function() {
        clone.renderAll();
        callback && callback(clone);
      });
      clone.backgroundImageOpacity = this.backgroundImageOpacity;
      clone.backgroundImageStretch = this.backgroundImageStretch;
    }
    else {
      callback && callback(clone);
    }
  }
});


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      clone = fabric.util.object.clone,
      toFixed = fabric.util.toFixed,
      capitalize = fabric.util.string.capitalize,
      degreesToRadians = fabric.util.degreesToRadians,
      supportsLineDash = fabric.StaticCanvas.supports('setLineDash'),
      objectCaching = !fabric.isLikelyNode,
      ALIASING_LIMIT = 2;

  if (fabric.Object) {
    return;
  }

  /**
   * Root object class from which all 2d shape classes inherit from
   * @class fabric.Object
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-1#objects}
   * @see {@link fabric.Object#initialize} for constructor definition
   *
   * @fires added
   * @fires removed
   *
   * @fires selected
   * @fires deselected
   * @fires modified
   * @fires rotating
```

```
   * @fires scaling
   * @fires moving
   * @fires skewing
   *
   * @fires mousedown
   * @fires mouseup
   * @fires mouseover
   * @fires mouseout
   * @fires mousewheel
   */
  fabric.Object = fabric.util.createClass(fabric.CommonMethods, /** @lends
fabric.Object.prototype */ {

    /**
     * Retrieves object's {@link fabric.Object#clipTo|clipping function}
     * @method getClipTo
     * @memberOf fabric.Object.prototype
     * @return {Function}
     */

    /**
     * Sets object's {@link fabric.Object#clipTo|clipping function}
     * @method setClipTo
     * @memberOf fabric.Object.prototype
     * @param {Function} clipTo Clipping function
     * @return {fabric.Object} thisArg
     * @chainable
     */

    /**
     * Retrieves object's {@link fabric.Object#transformMatrix|transformMatrix}
     * @method getTransformMatrix
     * @memberOf fabric.Object.prototype
     * @return {Array} transformMatrix
     */

    /**
     * Sets object's {@link fabric.Object#transformMatrix|transformMatrix}
     * @method setTransformMatrix
     * @memberOf fabric.Object.prototype
     * @param {Array} transformMatrix
     * @return {fabric.Object} thisArg
     * @chainable
     */

    /**
     * Retrieves object's {@link fabric.Object#visible|visible} state
     * @method getVisible
     * @memberOf fabric.Object.prototype
     * @return {Boolean} True if visible
     */

    /**
     * Sets object's {@link fabric.Object#visible|visible} state
     * @method setVisible
     * @memberOf fabric.Object.prototype
     * @param {Boolean} value visible value
     * @return {fabric.Object} thisArg
     * @chainable
     */

    /**
     * Retrieves object's {@link fabric.Object#shadow|shadow}
     * @method getShadow
```

```
 * @memberOf fabric.Object.prototype
 * @return {Object} Shadow instance
 */

/**
 * Retrieves object's {@link fabric.Object#stroke|stroke}
 * @method getStroke
 * @memberOf fabric.Object.prototype
 * @return {String} stroke value
 */

/**
 * Sets object's {@link fabric.Object#stroke|stroke}
 * @method setStroke
 * @memberOf fabric.Object.prototype
 * @param {String} value stroke value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#strokeWidth|strokeWidth}
 * @method getStrokeWidth
 * @memberOf fabric.Object.prototype
 * @return {Number} strokeWidth value
 */

/**
 * Sets object's {@link fabric.Object#strokeWidth|strokeWidth}
 * @method setStrokeWidth
 * @memberOf fabric.Object.prototype
 * @param {Number} value strokeWidth value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#originX|originX}
 * @method getOriginX
 * @memberOf fabric.Object.prototype
 * @return {String} originX value
 */

/**
 * Sets object's {@link fabric.Object#originX|originX}
 * @method setOriginX
 * @memberOf fabric.Object.prototype
 * @param {String} value originX value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#originY|originY}
 * @method getOriginY
 * @memberOf fabric.Object.prototype
 * @return {String} originY value
 */

/**
 * Sets object's {@link fabric.Object#originY|originY}
 * @method setOriginY
 * @memberOf fabric.Object.prototype
 * @param {String} value originY value
```

```
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#fill|fill}
 * @method getFill
 * @memberOf fabric.Object.prototype
 * @return {String} Fill value
 */

/**
 * Sets object's {@link fabric.Object#fill|fill}
 * @method setFill
 * @memberOf fabric.Object.prototype
 * @param {String} value Fill value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#opacity|opacity}
 * @method getOpacity
 * @memberOf fabric.Object.prototype
 * @return {Number} Opacity value (0-1)
 */

/**
 * Sets object's {@link fabric.Object#opacity|opacity}
 * @method setOpacity
 * @memberOf fabric.Object.prototype
 * @param {Number} value Opacity value (0-1)
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#angle|angle} (in degrees)
 * @method getAngle
 * @memberOf fabric.Object.prototype
 * @return {Number}
 */

/**
 * Retrieves object's {@link fabric.Object#top|top position}
 * @method getTop
 * @memberOf fabric.Object.prototype
 * @return {Number} Top value (in pixels)
 */

/**
 * Sets object's {@link fabric.Object#top|top position}
 * @method setTop
 * @memberOf fabric.Object.prototype
 * @param {Number} value Top value (in pixels)
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#left|left position}
 * @method getLeft
 * @memberOf fabric.Object.prototype
 * @return {Number} Left value (in pixels)
```

```
 */

/**
 * Sets object's {@link fabric.Object#left|left position}
 * @method setLeft
 * @memberOf fabric.Object.prototype
 * @param {Number} value Left value (in pixels)
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#scaleX|scaleX} value
 * @method getScaleX
 * @memberOf fabric.Object.prototype
 * @return {Number} scaleX value
 */

/**
 * Sets object's {@link fabric.Object#scaleX|scaleX} value
 * @method setScaleX
 * @memberOf fabric.Object.prototype
 * @param {Number} value scaleX value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#scaleY|scaleY} value
 * @method getScaleY
 * @memberOf fabric.Object.prototype
 * @return {Number} scaleY value
 */

/**
 * Sets object's {@link fabric.Object#scaleY|scaleY} value
 * @method setScaleY
 * @memberOf fabric.Object.prototype
 * @param {Number} value scaleY value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#flipX|flipX} value
 * @method getFlipX
 * @memberOf fabric.Object.prototype
 * @return {Boolean} flipX value
 */

/**
 * Sets object's {@link fabric.Object#flipX|flipX} value
 * @method setFlipX
 * @memberOf fabric.Object.prototype
 * @param {Boolean} value flipX value
 * @return {fabric.Object} thisArg
 * @chainable
 */

/**
 * Retrieves object's {@link fabric.Object#flipY|flipY} value
 * @method getFlipY
 * @memberOf fabric.Object.prototype
 * @return {Boolean} flipY value
```

```
     */

    /**
     * Sets object's {@link fabric.Object#flipY|flipY} value
     * @method setFlipY
     * @memberOf fabric.Object.prototype
     * @param {Boolean} value flipY value
     * @return {fabric.Object} thisArg
     * @chainable
     */

    /**
     * Type of an object (rect, circle, path, etc.).
     * Note that this property is meant to be read-only and not meant to be
modified.
     * If you modify, certain parts of Fabric (such as JSON loading) won't work
correctly.
     * @type String
     * @default
     */
    type:                     'object',

    /**
     * Horizontal origin of transformation of an object (one of "left", "right",
"center")
     * See http://jsfiddle.net/1ow02gea/40/ on how originX/originY affect
objects in groups
     * @type String
     * @default
     */
    originX:                  'left',

    /**
     * Vertical origin of transformation of an object (one of "top", "bottom",
"center")
     * See http://jsfiddle.net/1ow02gea/40/ on how originX/originY affect
objects in groups
     * @type String
     * @default
     */
    originY:                  'top',

    /**
     * Top position of an object. Note that by default it's relative to object
top. You can change this by setting originY={top/center/bottom}
     * @type Number
     * @default
     */
    top:                      0,

    /**
     * Left position of an object. Note that by default it's relative to object
left. You can change this by setting originX={left/center/right}
     * @type Number
     * @default
     */
    left:                     0,

    /**
     * Object width
     * @type Number
     * @default
     */
    width:                    0,
```

```
/**
 * Object height
 * @type Number
 * @default
 */
height:                   0,

/**
 * Object scale factor (horizontal)
 * @type Number
 * @default
 */
scaleX:                   1,

/**
 * Object scale factor (vertical)
 * @type Number
 * @default
 */
scaleY:                   1,

/**
 * When true, an object is rendered as flipped horizontally
 * @type Boolean
 * @default
 */
flipX:                    false,

/**
 * When true, an object is rendered as flipped vertically
 * @type Boolean
 * @default
 */
flipY:                    false,

/**
 * Opacity of an object
 * @type Number
 * @default
 */
opacity:                  1,

/**
 * Angle of rotation of an object (in degrees)
 * @type Number
 * @default
 */
angle:                    0,

/**
 * Angle of skew on x axes of an object (in degrees)
 * @type Number
 * @default
 */
skewX:                    0,

/**
 * Angle of skew on y axes of an object (in degrees)
 * @type Number
 * @default
 */
skewY:                    0,
```

```
    /**
     * Size of object's controlling corners (in pixels)
     * @type Number
     * @default
     */
    cornerSize:               13,

    /**
     * When true, object's controlling corners are rendered as transparent
inside (i.e. stroke instead of fill)
     * @type Boolean
     * @default
     */
    transparentCorners:       true,

    /**
     * Default cursor value used when hovering over this object on canvas
     * @type String
     * @default
     */
    hoverCursor:              null,

    /**
     * Default cursor value used when moving this object on canvas
     * @type String
     * @default
     */
    moveCursor:               null,

    /**
     * Padding between object and its controlling borders (in pixels)
     * @type Number
     * @default
     */
    padding:                  0,

    /**
     * Color of controlling borders of an object (when it's active)
     * @type String
     * @default
     */
    borderColor:              'rgba(102,153,255,0.75)',

    /**
     * Array specifying dash pattern of an object's borders (hasBorder must be
true)
     * @since 1.6.2
     * @type Array
     */
    borderDashArray:          null,

    /**
     * Color of controlling corners of an object (when it's active)
     * @type String
     * @default
     */
    cornerColor:              'rgba(102,153,255,0.5)',

    /**
     * Color of controlling corners of an object (when it's active and
transparentCorners false)
     * @since 1.6.2
     * @type String
     * @default
```

```
     */
    cornerStrokeColor:        null,

    /**
     * Specify style of control, 'rect' or 'circle'
     * @since 1.6.2
     * @type String
     */
    cornerStyle:          'rect',

    /**
     * Array specifying dash pattern of an object's control (hasBorder must be
true)
     * @since 1.6.2
     * @type Array
     */
    cornerDashArray:          null,

    /**
     * When true, this object will use center point as the origin of
transformation
     * when being scaled via the controls.
     * <b>Backwards incompatibility note:</b> This property replaces
"centerTransform" (Boolean).
     * @since 1.3.4
     * @type Boolean
     * @default
     */
    centeredScaling:          false,

    /**
     * When true, this object will use center point as the origin of
transformation
     * when being rotated via the controls.
     * <b>Backwards incompatibility note:</b> This property replaces
"centerTransform" (Boolean).
     * @since 1.3.4
     * @type Boolean
     * @default
     */
    centeredRotation:         true,

    /**
     * Color of object's fill
     * @type String
     * @default
     */
    fill:                     'rgb(0,0,0)',

    /**
     * Fill rule used to fill an object
     * accepted values are nonzero, evenodd
     * <b>Backwards incompatibility note:</b> This property was used for setting
globalCompositeOperation until v1.4.12 (use
`fabric.Object#globalCompositeOperation` instead)
     * @type String
     * @default
     */
    fillRule:                 'nonzero',

    /**
     * Composite rule used for canvas globalCompositeOperation
     * @type String
     * @default
```

```
     */
    globalCompositeOperation: 'source-over',

    /**
     * Background color of an object.
     * @type String
     * @default
     */
    backgroundColor:          '',

    /**
     * Selection Background color of an object. colored layer behind the object
when it is active.
     * does not mix good with globalCompositeOperation methods.
     * @type String
     * @default
     */
    selectionBackgroundColor:          '',

    /**
     * When defined, an object is rendered via stroke and this property
specifies its color
     * @type String
     * @default
     */
    stroke:                   null,

    /**
     * Width of a stroke used to render this object
     * @type Number
     * @default
     */
    strokeWidth:              1,

    /**
     * Array specifying dash pattern of an object's stroke (stroke must be
defined)
     * @type Array
     */
    strokeDashArray:          null,

    /**
     * Line endings style of an object's stroke (one of "butt", "round",
"square")
     * @type String
     * @default
     */
    strokeLineCap:            'butt',

    /**
     * Corner style of an object's stroke (one of "bevil", "round", "miter")
     * @type String
     * @default
     */
    strokeLineJoin:           'miter',

    /**
     * Maximum miter length (used for strokeLineJoin = "miter") of an object's
stroke
     * @type Number
     * @default
     */
    strokeMiterLimit:         10,
```

```
    /**
     * Shadow object representing shadow of this shape
     * @type fabric.Shadow
     * @default
     */
    shadow:                   null,

    /**
     * Opacity of object's controlling borders when object is active and moving
     * @type Number
     * @default
     */
    borderOpacityWhenMoving:  0.4,

    /**
     * Scale factor of object's controlling borders
     * @type Number
     * @default
     */
    borderScaleFactor:        1,

    /**
     * Transform matrix (similar to SVG's transform matrix)
     * @type Array
     */
    transformMatrix:          null,

    /**
     * Minimum allowed scale value of an object
     * @type Number
     * @default
     */
    minScaleLimit:            0.01,

    /**
     * When set to `false`, an object can not be selected for modification
(using either point-click-based or group-based selection).
     * But events still fire on it.
     * @type Boolean
     * @default
     */
    selectable:               true,

    /**
     * When set to `false`, an object can not be a target of events. All events
propagate through it. Introduced in v1.3.4
     * @type Boolean
     * @default
     */
    evented:                  true,

    /**
     * When set to `false`, an object is not rendered on canvas
     * @type Boolean
     * @default
     */
    visible:                  true,

    /**
     * When set to `false`, object's controls are not displayed and can not be
used to manipulate object
     * @type Boolean
     * @default
     */
```

```
    hasControls:              true,

    /**
     * When set to `false`, object's controlling borders are not rendered
     * @type Boolean
     * @default
     */
    hasBorders:               true,

    /**
     * When set to `false`, object's controlling rotating point will not be
visible or selectable
     * @type Boolean
     * @default
     */
    hasRotatingPoint:         true,

    /**
     * Offset for object's controlling rotating point (when enabled via
`hasRotatingPoint`)
     * @type Number
     * @default
     */
    rotatingPointOffset:      40,

    /**
     * When set to `true`, objects are "found" on canvas on per-pixel basis
rather than according to bounding box
     * @type Boolean
     * @default
     */
    perPixelTargetFind:       false,

    /**
     * When `false`, default object's values are not included in its
serialization
     * @type Boolean
     * @default
     */
    includeDefaultValues:     true,

    /**
     * Function that determines clipping of an object (context is passed as a
first argument)
     * Note that context origin is at the object's center point (not left/top
corner)
     * @type Function
     */
    clipTo:                   null,

    /**
     * When `true`, object horizontal movement is locked
     * @type Boolean
     * @default
     */
    lockMovementX:            false,

    /**
     * When `true`, object vertical movement is locked
     * @type Boolean
     * @default
     */
    lockMovementY:            false,
```

```
/**
 * When `true`, object rotation is locked
 * @type Boolean
 * @default
 */
lockRotation:             false,

/**
 * When `true`, object horizontal scaling is locked
 * @type Boolean
 * @default
 */
lockScalingX:             false,

/**
 * When `true`, object vertical scaling is locked
 * @type Boolean
 * @default
 */
lockScalingY:             false,

/**
 * When `true`, object non-uniform scaling is locked
 * @type Boolean
 * @default
 */
lockUniScaling:           false,

/**
 * When `true`, object horizontal skewing is locked
 * @type Boolean
 * @default
 */
lockSkewingX:             false,

/**
 * When `true`, object vertical skewing is locked
 * @type Boolean
 * @default
 */
lockSkewingY:             false,

/**
 * When `true`, object cannot be flipped by scaling into negative values
 * @type Boolean
 * @default
 */
lockScalingFlip:          false,

/**
 * When `true`, object is not exported in SVG or OBJECT/JSON
 * since 1.6.3
 * @type Boolean
 * @default
 */
excludeFromExport:        false,

/**
 * When `true`, object is cached on an additional canvas.
 * default to true
 * since 1.7.0
 * @type Boolean
 * @default true
 */
```

```
    objectCaching:           objectCaching,

    /**
     * When `true`, object properties are checked for cache invalidation. In
some particular
     * situation you may want this to be disabled ( spray brush, very big
pathgroups, groups)
     * or if your application does not allow you to modify properties for groups
child you want
     * to disable it for groups.
     * default to false
     * since 1.7.0
     * @type Boolean
     * @default false
     */
    statefullCache:            false,

    /**
     * When `true`, cache does not get updated during scaling. The picture will
get blocky if scaled
     * too much and will be redrawn with correct details at the end of scaling.
     * this setting is performance and application dependant.
     * default to true
     * since 1.7.0
     * @type Boolean
     * @default true
     */
    noScaleCache:              true,

    /**
     * When set to `true`, object's cache will be rerendered next render call.
     * since 1.7.0
     * @type Boolean
     * @default true
     */
    dirty:               true,

    /**
     * List of properties to consider when checking if state
     * of an object is changed (fabric.Object#hasStateChanged)
     * as well as for history (undo/redo) purposes
     * @type Array
     */
    stateProperties: (
      'top left width height scaleX scaleY flipX flipY originX originY
transformMatrix ' +
      'stroke strokeWidth strokeDashArray strokeLineCap strokeLineJoin
strokeMiterLimit ' +
      'angle opacity fill globalCompositeOperation shadow clipTo visible
backgroundColor ' +
      'skewX skewY fillRule'
    ).split(' '),

    /**
     * List of properties to consider when checking if cache needs refresh
     * @type Array
     */
    cacheProperties: (
      'fill stroke strokeWidth strokeDashArray width height' +
      ' strokeLineCap strokeLineJoin strokeMiterLimit backgroundColor'
    ).split(' '),

    /**
     * Constructor
```

```
     * @param {Object} [options] Options object
     */
    initialize: function(options) {
      options = options || { };
      if (options) {
        this.setOptions(options);
      }
    },

    /**
     * Create a the canvas used to keep the cached copy of the object
     * @private
     */
    _createCacheCanvas: function() {
      this._cacheProperties = {};
      this._cacheCanvas = fabric.document.createElement('canvas');
      this._cacheContext = this._cacheCanvas.getContext('2d');
      this._updateCacheCanvas();
    },

    /**
     * Limit the cache dimensions so that X * Y do not cross
fabric.perfLimitSizeTotal
     * and each side do not cross fabric.cacheSideLimit
     * those numbers are configurable so that you can get as much detail as you
want
     * making bargain with performances.
     * @param {Object} dims
     * @param {Object} dims.width width of canvas
     * @param {Object} dims.height height of canvas
     * @param {Object} dims.zoomX zoomX zoom value to unscale the canvas before
drawing cache
     * @param {Object} dims.zoomY zoomY zoom value to unscale the canvas before
drawing cache
     * @return {Object}.width width of canvas
     * @return {Object}.height height of canvas
     * @return {Object}.zoomX zoomX zoom value to unscale the canvas before
drawing cache
     * @return {Object}.zoomY zoomY zoom value to unscale the canvas before
drawing cache
     */
    _limitCacheSize: function(dims) {
      var perfLimitSizeTotal = fabric.perfLimitSizeTotal,
          maximumSide = fabric.cacheSideLimit,
          width = dims.width, height = dims.height,
          ar = width / height, limitedDims = fabric.util.limitDimsByArea(ar,
perfLimitSizeTotal, maximumSide),
          capValue = fabric.util.capValue, max = fabric.maxCacheSideLimit, min =
fabric.minCacheSideLimit,
          x = capValue(min, limitedDims.x, max),
          y = capValue(min, limitedDims.y, max);
      if (width > x) {
        dims.zoomX /= width / x;
        dims.width = x;
      }
      else if (width < min) {
        dims.width = min;
      }
      if (height > y) {
        dims.zoomY /= height / y;
        dims.height = y;
      }
      else if (height < min) {
        dims.height = min;
```

```
      }
      return dims;
    },

    /**
     * Return the dimension and the zoom level needed to create a cache canvas
     * big enough to host the object to be cached.
     * @private
     * @return {Object}.width width of canvas
     * @return {Object}.height height of canvas
     * @return {Object}.zoomX zoomX zoom value to unscale the canvas before
drawing cache
     * @return {Object}.zoomY zoomY zoom value to unscale the canvas before
drawing cache
     */
    _getCacheCanvasDimensions: function() {
      var zoom = this.canvas && this.canvas.getZoom() || 1,
          objectScale = this.getObjectScaling(),
          dim = this._getNonTransformedDimensions(),
          retina = this.canvas && this.canvas._isRetinaScaling() ?
fabric.devicePixelRatio : 1,
          zoomX = objectScale.scaleX * zoom * retina,
          zoomY = objectScale.scaleY * zoom * retina,
          width = dim.x * zoomX,
          height = dim.y * zoomY;
      return {
        width: width + ALIASING_LIMIT,
        height: height + ALIASING_LIMIT,
        zoomX: zoomX,
        zoomY: zoomY
      };
    },

    /**
     * Update width and height of the canvas for cache
     * returns true or false if canvas needed resize.
     * @private
     * @return {Boolean} true if the canvas has been resized
     */
    _updateCacheCanvas: function() {
      if (this.noScaleCache && this.canvas && this.canvas._currentTransform) {
        var action = this.canvas._currentTransform.action;
        if (action.slice && action.slice(0, 5) === 'scale') {
          return false;
        }
      }
      var dims = this._limitCacheSize(this._getCacheCanvasDimensions()),
          minCacheSize = fabric.minCacheSideLimit,
          width = dims.width, height = dims.height,
          zoomX = dims.zoomX, zoomY = dims.zoomY,
          dimensionsChanged = width !== this.cacheWidth || height !==
this.cacheHeight,
          zoomChanged = this.zoomX !== zoomX || this.zoomY !== zoomY,
          shouldRedraw = dimensionsChanged || zoomChanged,
          additionalWidth = 0, additionalHeight = 0, shouldResizeCanvas = false;
      if (dimensionsChanged) {
        var canvasWidth = this._cacheCanvas.width,
            canvasHeight = this._cacheCanvas.height,
            sizeGrowing = width > canvasWidth || height > canvasHeight,
            sizeShrinking = (width < canvasWidth * 0.9 || height < canvasHeight
* 0.9) &&
              canvasWidth > minCacheSize && canvasHeight > minCacheSize;
        shouldResizeCanvas = sizeGrowing || sizeShrinking;
        if (sizeGrowing) {
```

```
          additionalWidth = (width * 0.1) & ~1;
          additionalHeight = (height * 0.1) & ~1;
        }
      }
      if (shouldRedraw) {
        if (shouldResizeCanvas) {
          this._cacheCanvas.width = Math.max(Math.ceil(width) + additionalWidth,
minCacheSize);
          this._cacheCanvas.height = Math.max(Math.ceil(height) +
additionalHeight, minCacheSize);
          this.cacheTranslationX = (width + additionalWidth) / 2;
          this.cacheTranslationY = (height + additionalHeight) / 2;
        }
        else {
          this._cacheContext.setTransform(1, 0, 0, 1, 0, 0);
          this._cacheContext.clearRect(0, 0, this._cacheCanvas.width,
this._cacheCanvas.height);
        }
        this.cacheWidth = width;
        this.cacheHeight = height;
        this._cacheContext.translate(this.cacheTranslationX,
this.cacheTranslationY);
        this._cacheContext.scale(zoomX, zoomY);
        this.zoomX = zoomX;
        this.zoomY = zoomY;
        return true;
      }
      return false;
    },

    /**
     * Sets object's properties from options
     * @param {Object} [options] Options object
     */
    setOptions: function(options) {
      this._setOptions(options);
      this._initGradient(options.fill, 'fill');
      this._initGradient(options.stroke, 'stroke');
      this._initClipping(options);
      this._initPattern(options.fill, 'fill');
      this._initPattern(options.stroke, 'stroke');
    },

    /**
     * Transforms context when rendering an object
     * @param {CanvasRenderingContext2D} ctx Context
     * @param {Boolean} fromLeft When true, context is transformed to object's
top/left corner. This is used when rendering text on Node
     */
    transform: function(ctx, fromLeft) {
      if (this.group && !this.group._transformDone && this.group ===
this.canvas._activeGroup) {
        this.group.transform(ctx);
      }
      var center = fromLeft ? this._getLeftTopCoords() : this.getCenterPoint();
      ctx.translate(center.x, center.y);
      this.angle && ctx.rotate(degreesToRadians(this.angle));
      ctx.scale(
        this.scaleX * (this.flipX ? -1 : 1),
        this.scaleY * (this.flipY ? -1 : 1)
      );
      this.skewX && ctx.transform(1, 0, Math.tan(degreesToRadians(this.skewX)),
1, 0, 0);
      this.skewY && ctx.transform(1, Math.tan(degreesToRadians(this.skewY)), 0,
```

```
1, 0, 0);
    },

    /**
     * Returns an object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} Object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      var NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS,

          object = {
            type:                    this.type,
            originX:                 this.originX,
            originY:                 this.originY,
            left:                    toFixed(this.left, NUM_FRACTION_DIGITS),
            top:                     toFixed(this.top, NUM_FRACTION_DIGITS),
            width:                   toFixed(this.width, NUM_FRACTION_DIGITS),
            height:                  toFixed(this.height, NUM_FRACTION_DIGITS),
            fill:                    (this.fill && this.fill.toObject) ?
this.fill.toObject() : this.fill,
            stroke:                  (this.stroke && this.stroke.toObject) ?
this.stroke.toObject() : this.stroke,
            strokeWidth:             toFixed(this.strokeWidth,
NUM_FRACTION_DIGITS),
            strokeDashArray:         this.strokeDashArray ?
this.strokeDashArray.concat() : this.strokeDashArray,
            strokeLineCap:           this.strokeLineCap,
            strokeLineJoin:          this.strokeLineJoin,
            strokeMiterLimit:        toFixed(this.strokeMiterLimit,
NUM_FRACTION_DIGITS),
            scaleX:                  toFixed(this.scaleX, NUM_FRACTION_DIGITS),
            scaleY:                  toFixed(this.scaleY, NUM_FRACTION_DIGITS),
            angle:                   toFixed(this.getAngle(),
NUM_FRACTION_DIGITS),
            flipX:                   this.flipX,
            flipY:                   this.flipY,
            opacity:                 toFixed(this.opacity,
NUM_FRACTION_DIGITS),
            shadow:                  (this.shadow && this.shadow.toObject) ?
this.shadow.toObject() : this.shadow,
            visible:                 this.visible,
            clipTo:                  this.clipTo && String(this.clipTo),
            backgroundColor:         this.backgroundColor,
            fillRule:                this.fillRule,
            globalCompositeOperation: this.globalCompositeOperation,
            transformMatrix:         this.transformMatrix ?
this.transformMatrix.concat() : null,
            skewX:                   toFixed(this.skewX, NUM_FRACTION_DIGITS),
            skewY:                   toFixed(this.skewY, NUM_FRACTION_DIGITS)
          };

      fabric.util.populateWithProperties(this, object, propertiesToInclude);

      if (!this.includeDefaultValues) {
        object = this._removeDefaultValues(object);
      }

      return object;
    },

    /**
     * Returns (dataless) object representation of an instance
```

```
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} Object representation of an instance
     */
    toDatalessObject: function(propertiesToInclude) {
      // will be overwritten by subclasses
      return this.toObject(propertiesToInclude);
    },

    /**
     * @private
     * @param {Object} object
     */
    _removeDefaultValues: function(object) {
      var prototype = fabric.util.getKlass(object.type).prototype,
          stateProperties = prototype.stateProperties;
      stateProperties.forEach(function(prop) {
        if (object[prop] === prototype[prop]) {
          delete object[prop];
        }
        var isArray = Object.prototype.toString.call(object[prop]) === '[object
Array]' &&
                      Object.prototype.toString.call(prototype[prop]) ===
'[object Array]';

        // basically a check for [] === []
        if (isArray && object[prop].length === 0 && prototype[prop].length ===
0) {
          delete object[prop];
        }
      });

      return object;
    },

    /**
     * Returns a string representation of an instance
     * @return {String}
     */
    toString: function() {
      return '#<fabric.' + capitalize(this.type) + '>';
    },

    /**
     * Return the object scale factor counting also the group scaling
     * @return {Object} object with scaleX and scaleY properties
     */
    getObjectScaling: function() {
      var scaleX = this.scaleX, scaleY = this.scaleY;
      if (this.group) {
        var scaling = this.group.getObjectScaling();
        scaleX *= scaling.scaleX;
        scaleY *= scaling.scaleY;
      }
      return { scaleX: scaleX, scaleY: scaleY };
    },

    /**
     * @private
     * @param {String} key
     * @param {*} value
     * @return {fabric.Object} thisArg
     */
    _set: function(key, value) {
```

```
      var shouldConstrainValue = (key === 'scaleX' || key === 'scaleY');

      if (shouldConstrainValue) {
        value = this._constrainScale(value);
      }
      if (key === 'scaleX' && value < 0) {
        this.flipX = !this.flipX;
        value *= -1;
      }
      else if (key === 'scaleY' && value < 0) {
        this.flipY = !this.flipY;
        value *= -1;
      }
      else if (key === 'shadow' && value && !(value instanceof fabric.Shadow)) {
        value = new fabric.Shadow(value);
      }
      else if (key === 'dirty' && this.group) {
        this.group.set('dirty', value);
      }

      this[key] = value;

      if (this.cacheProperties.indexOf(key) > -1) {
        if (this.group) {
          this.group.set('dirty', true);
        }
        this.dirty = true;
      }

      if (this.group && this.stateProperties.indexOf(key) > -1) {
        this.group.set('dirty', true);
      }

      if (key === 'width' || key === 'height') {
        this.minScaleLimit = Math.min(0.1, 1 / Math.max(this.width,
this.height));
      }

      return this;
    },

    /**
     * This callback function is called by the parent group of an object every
     * time a non-delegated property changes on the group. It is passed the key
     * and value as parameters. Not adding in this function's signature to avoid
     * Travis build error about unused variables.
     */
    setOnGroup: function() {
      // implemented by sub-classes, as needed.
    },

    /**
     * Sets sourcePath of an object
     * @param {String} value Value to set sourcePath to
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setSourcePath: function(value) {
      this.sourcePath = value;
      return this;
    },

    /**
     * Retrieves viewportTransform from Object's canvas if possible
```

```
     * @method getViewportTransform
     * @memberOf fabric.Object.prototype
     * @return {Boolean}
     */
    getViewportTransform: function() {
      if (this.canvas && this.canvas.viewportTransform) {
        return this.canvas.viewportTransform;
      }
      return fabric.iMatrix.concat();
    },

    /*
     * @private
     * return if the object would be visible in rendering
     * @memberOf fabric.Object.prototype
     * @return {Boolean}
     */
    isNotVisible: function() {
      return this.opacity === 0 || (this.width === 0 && this.height === 0) || !
this.visible;
    },

    /**
     * Renders an object on a specified context
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    render: function(ctx, noTransform) {
      // do not render if width/height are zeros or object is not visible
      if (this.isNotVisible()) {
        return;
      }
      if (this.canvas && this.canvas.skipOffscreen && !this.group && !
this.isOnScreen()) {
        return;
      }
      ctx.save();
      //setup fill rule for current object
      this._setupCompositeOperation(ctx);
      this.drawSelectionBackground(ctx);
      if (!noTransform) {
        this.transform(ctx);
      }
      this._setOpacity(ctx);
      this._setShadow(ctx);
      if (this.transformMatrix) {
        ctx.transform.apply(ctx, this.transformMatrix);
      }
      this.clipTo && fabric.util.clipContext(this, ctx);
      if (this.shouldCache(noTransform)) {
        if (!this._cacheCanvas) {
          this._createCacheCanvas();
        }
        if (this.isCacheDirty(noTransform)) {
          this.statefullCache && this.saveState({ propertySet: 'cacheProperties'
});
          this.drawObject(this._cacheContext, noTransform);
          this.dirty = false;
        }
        this.drawCacheOnCanvas(ctx);
      }
      else {
        this.dirty = false;
        this.drawObject(ctx, noTransform);
```

```
        if (noTransform && this.objectCaching && this.statefullCache) {
          this.saveState({ propertySet: 'cacheProperties' });
        }
      }
      this.clipTo && ctx.restore();
      ctx.restore();
    },

    /**
     * When returns `true`, force the object to have its own cache, even if it
is inside a group
     * it may be needed when your object behave in a particular way on the cache
and always needs
     * its own isolated canvas to render correctly.
     * This function is created to be subclassed by custom classes.
     * since 1.7.12
     * @type function
     * @return false
     */
    needsItsOwnCache: function() {
      return false;
    },

    /**
     * Decide if the object should cache or not.
     * objectCaching is a global flag, wins over everything
     * needsItsOwnCache should be used when the object drawing method requires
     * a cache step. None of the fabric classes requires it.
     * Generally you do not cache objects in groups because the group outside is
cached.
     * @param {Boolean} noTransform if rendereing in pathGroup, caching is not
supported at object level
     * @return {Boolean}
     */
    shouldCache: function(noTransform) {
      return !noTransform && this.objectCaching &&
      (!this.group || this.needsItsOwnCache() || !this.group.isCaching());
    },

    /**
     * Check if this object or a child object will cast a shadow
     * used by Group.shouldCache to know if child has a shadow recursively
     * @return {Boolean}
     */
    willDrawShadow: function() {
      return !!this.shadow && (this.shadow.offsetX !== 0 ||
this.shadow.offsetY !== 0);
    },

    /**
     * Execute the drawing operation for an object on a specified context
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    drawObject: function(ctx, noTransform) {
      this._renderBackground(ctx);
      this._setStrokeStyles(ctx);
      this._setFillStyles(ctx);
      this._render(ctx, noTransform);
    },

    /**
     * Paint the cached copy of the object on the target context.
     * @param {CanvasRenderingContext2D} ctx Context to render on
```

```
     */
    drawCacheOnCanvas: function(ctx) {
      ctx.scale(1 / this.zoomX, 1 / this.zoomY);
      ctx.drawImage(this._cacheCanvas, -this.cacheTranslationX, -
this.cacheTranslationY);
    },

    /**
     * Check if cache is dirty
     * @param {Boolean} skipCanvas skip canvas checks because this object is
painted
     * on parent canvas.
     */
    isCacheDirty: function(skipCanvas) {
      if (this.isNotVisible()) {
        return false;
      }
      if (this._cacheCanvas && !skipCanvas && this._updateCacheCanvas()) {
        // in this case the context is already cleared.
        return true;
      }
      else {
        if (this.dirty || (this.statefullCache &&
this.hasStateChanged('cacheProperties'))) {
          if (this._cacheCanvas && !skipCanvas) {
            var width = this.cacheWidth / this.zoomX;
            var height = this.cacheHeight / this.zoomY;
            this._cacheContext.clearRect(-width / 2, -height / 2, width,
height);
          }
          return true;
        }
      }
      return false;
    },

    /**
     * Draws a background for the object big as its untrasformed dimensions
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderBackground: function(ctx) {
      if (!this.backgroundColor) {
        return;
      }
      var dim = this._getNonTransformedDimensions();
      ctx.fillStyle = this.backgroundColor;

      ctx.fillRect(
        -dim.x / 2,
        -dim.y / 2,
        dim.x,
        dim.y
      );
      // if there is background color no other shadows
      // should be casted
      this._removeShadow(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _setOpacity: function(ctx) {
```

```
      ctx.globalAlpha *= this.opacity;
    },

    _setStrokeStyles: function(ctx) {
      if (this.stroke) {
        ctx.lineWidth = this.strokeWidth;
        ctx.lineCap = this.strokeLineCap;
        ctx.lineJoin = this.strokeLineJoin;
        ctx.miterLimit = this.strokeMiterLimit;
        ctx.strokeStyle = this.stroke.toLive
          ? this.stroke.toLive(ctx, this)
          : this.stroke;
      }
    },

    _setFillStyles: function(ctx) {
      if (this.fill) {
        ctx.fillStyle = this.fill.toLive
          ? this.fill.toLive(ctx, this)
          : this.fill;
      }
    },

    /**
     * @private
     * Sets line dash
     * @param {CanvasRenderingContext2D} ctx Context to set the dash line on
     * @param {Array} dashArray array representing dashes
     * @param {Function} alternative function to call if browaser does not
support lineDash
     */
    _setLineDash: function(ctx, dashArray, alternative) {
      if (!dashArray) {
        return;
      }
      // Spec requires the concatenation of two copies the dash list when the
number of elements is odd
      if (1 & dashArray.length) {
        dashArray.push.apply(dashArray, dashArray);
      }
      if (supportsLineDash) {
        ctx.setLineDash(dashArray);
      }
      else {
        alternative && alternative(ctx);
      }
    },

    /**
     * Renders controls and borders for the object
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderControls: function(ctx) {
      if (!this.active || (this.group && this.group !==
this.canvas.getActiveGroup())) {
        return;
      }

      var vpt = this.getViewportTransform(),
          matrix = this.calcTransformMatrix(),
          options;
      matrix = fabric.util.multiplyTransformMatrices(vpt, matrix);
      options = fabric.util.qrDecompose(matrix);
```

```
      ctx.save();
      ctx.translate(options.translateX, options.translateY);
      ctx.lineWidth = 1 * this.borderScaleFactor;
      if (!this.group) {
        ctx.globalAlpha = this.isMoving ? this.borderOpacityWhenMoving : 1;
      }
      if (this.group && this.group === this.canvas.getActiveGroup()) {
        ctx.rotate(degreesToRadians(options.angle));
        this.drawBordersInGroup(ctx, options);
      }
      else {
        ctx.rotate(degreesToRadians(this.angle));
        this.drawBorders(ctx);
      }
      this.drawControls(ctx);
      ctx.restore();
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _setShadow: function(ctx) {
      if (!this.shadow) {
        return;
      }

      var multX = (this.canvas && this.canvas.viewportTransform[0]) || 1,
          multY = (this.canvas && this.canvas.viewportTransform[3]) || 1,
          scaling = this.getObjectScaling();
      if (this.canvas && this.canvas._isRetinaScaling()) {
        multX *= fabric.devicePixelRatio;
        multY *= fabric.devicePixelRatio;
      }
      ctx.shadowColor = this.shadow.color;
      ctx.shadowBlur = this.shadow.blur * (multX + multY) * (scaling.scaleX +
scaling.scaleY) / 4;
      ctx.shadowOffsetX = this.shadow.offsetX * multX * scaling.scaleX;
      ctx.shadowOffsetY = this.shadow.offsetY * multY * scaling.scaleY;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _removeShadow: function(ctx) {
      if (!this.shadow) {
        return;
      }

      ctx.shadowColor = '';
      ctx.shadowBlur = ctx.shadowOffsetX = ctx.shadowOffsetY = 0;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Object} filler fabric.Pattern or fabric.Gradient
     */
    _applyPatternGradientTransform: function(ctx, filler) {
      if (!filler.toLive) {
        return;
      }
      var transform = filler.gradientTransform || filler.patternTransform;
```

```
      if (transform) {
        ctx.transform.apply(ctx, transform);
      }
      var offsetX = -this.width / 2 + filler.offsetX || 0,
          offsetY = -this.height / 2 + filler.offsetY || 0;
      ctx.translate(offsetX, offsetY);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderFill: function(ctx) {
      if (!this.fill) {
        return;
      }

      ctx.save();
      this._applyPatternGradientTransform(ctx, this.fill);
      if (this.fillRule === 'evenodd') {
        ctx.fill('evenodd');
      }
      else {
        ctx.fill();
      }
      ctx.restore();
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderStroke: function(ctx) {
      if (!this.stroke || this.strokeWidth === 0) {
        return;
      }

      if (this.shadow && !this.shadow.affectStroke) {
        this._removeShadow(ctx);
      }

      ctx.save();
      this._setLineDash(ctx, this.strokeDashArray, this._renderDashedStroke);
      this._applyPatternGradientTransform(ctx, this.stroke);
      ctx.stroke();
      ctx.restore();
    },

    /**
     * Clones an instance, some objects are async, so using callback method will
work for every object.
     * Using the direct return does not work for images and groups.
     * @param {Function} callback Callback is invoked with a clone as a first
argument
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {fabric.Object} clone of an instance
     */
    clone: function(callback, propertiesToInclude) {
      if (this.constructor.fromObject) {
        return this.constructor.fromObject(this.toObject(propertiesToInclude),
callback);
      }
      return new fabric.Object(this.toObject(propertiesToInclude));
```

```
    },

    /**
     * Creates an instance of fabric.Image out of an object
     * @param {Function} callback callback, invoked with an instance as a first
argument
     * @param {Object} [options] for clone as image, passed to toDataURL
     * @param {Boolean} [options.enableRetinaScaling] enable retina scaling for
the cloned image
     * @return {fabric.Object} thisArg
     */
    cloneAsImage: function(callback, options) {
      var dataUrl = this.toDataURL(options);
      fabric.util.loadImage(dataUrl, function(img) {
        if (callback) {
          callback(new fabric.Image(img));
        }
      });
      return this;
    },

    /**
     * Converts an object into a data-url-like string
     * @param {Object} options Options object
     * @param {String} [options.format=png] The format of the output image.
Either "jpeg" or "png"
     * @param {Number} [options.quality=1] Quality level (0..1). Only used for
jpeg.
     * @param {Number} [options.multiplier=1] Multiplier to scale by
     * @param {Number} [options.left] Cropping left offset. Introduced in
v1.2.14
     * @param {Number} [options.top] Cropping top offset. Introduced in v1.2.14
     * @param {Number} [options.width] Cropping width. Introduced in v1.2.14
     * @param {Number} [options.height] Cropping height. Introduced in v1.2.14
     * @param {Boolean} [options.enableRetina] Enable retina scaling for clone
image. Introduce in 1.6.4
     * @return {String} Returns a data: URL containing a representation of the
object in the format specified by options.format
     */
    toDataURL: function(options) {
      options || (options = { });

      var el = fabric.util.createCanvasElement(),
          boundingRect = this.getBoundingRect();

      el.width = boundingRect.width;
      el.height = boundingRect.height;
      fabric.util.wrapElement(el, 'div');
      var canvas = new fabric.StaticCanvas(el, { enableRetinaScaling:
options.enableRetinaScaling });
      // to avoid common confusion
https://github.com/kangax/fabric.js/issues/806
      if (options.format === 'jpg') {
        options.format = 'jpeg';
      }

      if (options.format === 'jpeg') {
        canvas.backgroundColor = '#fff';
      }

      var origParams = {
        active: this.get('active'),
        left: this.getLeft(),
        top: this.getTop()
```

```
      };

      this.set('active', false);
      this.setPositionByOrigin(new fabric.Point(canvas.getWidth() / 2,
canvas.getHeight() / 2), 'center', 'center');

      var originalCanvas = this.canvas;
      canvas.add(this);
      var data = canvas.toDataURL(options);

      this.set(origParams).setCoords();
      this.canvas = originalCanvas;

      canvas.dispose();
      canvas = null;

      return data;
    },

    /**
     * Returns true if specified type is identical to the type of an instance
     * @param {String} type Type to check against
     * @return {Boolean}
     */
    isType: function(type) {
      return this.type === type;
    },

    /**
     * Returns complexity of an instance
     * @return {Number} complexity of this instance (is 1 unless subclassed)
     */
    complexity: function() {
      return 1;
    },

    /**
     * Returns a JSON representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} JSON
     */
    toJSON: function(propertiesToInclude) {
      // delegate, not alias
      return this.toObject(propertiesToInclude);
    },

    /**
     * Sets gradient (fill or stroke) of an object
     * <b>Backwards incompatibility note:</b> This method was named
"setGradientFill" until v1.1.0
     * @param {String} property Property name 'stroke' or 'fill'
     * @param {Object} [options] Options object
     * @param {String} [options.type] Type of gradient 'radial' or 'linear'
     * @param {Number} [options.x1=0] x-coordinate of start point
     * @param {Number} [options.y1=0] y-coordinate of start point
     * @param {Number} [options.x2=0] x-coordinate of end point
     * @param {Number} [options.y2=0] y-coordinate of end point
     * @param {Number} [options.r1=0] Radius of start point (only for radial
gradients)
     * @param {Number} [options.r2=0] Radius of end point (only for radial
gradients)
     * @param {Object} [options.colorStops] Color stops object eg. {0: 'ff0000',
1: '000000'}
```

```
     * @param {Object} [options.gradientTransform] transforMatrix for gradient
     * @return {fabric.Object} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/58y8b/|jsFiddle demo}
     * @example <caption>Set linear gradient</caption>
     * object.setGradient('fill', {
     *   type: 'linear',
     *   x1: -object.width / 2,
     *   y1: 0,
     *   x2: object.width / 2,
     *   y2: 0,
     *   colorStops: {
     *     0: 'red',
     *     0.5: '#005555',
     *     1: 'rgba(0,0,255,0.5)'
     *   }
     * });
     * canvas.renderAll();
     * @example <caption>Set radial gradient</caption>
     * object.setGradient('fill', {
     *   type: 'radial',
     *   x1: 0,
     *   y1: 0,
     *   x2: 0,
     *   y2: 0,
     *   r1: object.width / 2,
     *   r2: 10,
     *   colorStops: {
     *     0: 'red',
     *     0.5: '#005555',
     *     1: 'rgba(0,0,255,0.5)'
     *   }
     * });
     * canvas.renderAll();
     */
    setGradient: function(property, options) {
      options || (options = { });

      var gradient = { colorStops: [] };

      gradient.type = options.type || (options.r1 || options.r2 ? 'radial' :
'linear');
      gradient.coords = {
        x1: options.x1,
        y1: options.y1,
        x2: options.x2,
        y2: options.y2
      };

      if (options.r1 || options.r2) {
        gradient.coords.r1 = options.r1;
        gradient.coords.r2 = options.r2;
      }

      gradient.gradientTransform = options.gradientTransform;
      fabric.Gradient.prototype.addColorStop.call(gradient, options.colorStops);

      return this.set(property, fabric.Gradient.forObject(this, gradient));
    },

    /**
     * Sets pattern fill of an object
     * @param {Object} options Options object
     * @param {(String|HTMLImageElement)} options.source Pattern source
```

```
     * @param {String} [options.repeat=repeat] Repeat property of a pattern (one
of repeat, repeat-x, repeat-y or no-repeat)
     * @param {Number} [options.offsetX=0] Pattern horizontal offset from
object's left/top corner
     * @param {Number} [options.offsetY=0] Pattern vertical offset from object's
left/top corner
     * @return {fabric.Object} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/QT3pa/|jsFiddle demo}
     * @example <caption>Set pattern</caption>
     * fabric.util.loadImage('http://fabricjs.com/assets/escheresque_ste.png',
function(img) {
     *   object.setPatternFill({
     *     source: img,
     *     repeat: 'repeat'
     *   });
     *   canvas.renderAll();
     * });
     */
    setPatternFill: function(options) {
      return this.set('fill', new fabric.Pattern(options));
    },

    /**
     * Sets {@link fabric.Object#shadow|shadow} of an object
     * @param {Object|String} [options] Options object or string (e.g. "2px 2px
10px rgba(0,0,0,0.2)")
     * @param {String} [options.color=rgb(0,0,0)] Shadow color
     * @param {Number} [options.blur=0] Shadow blur
     * @param {Number} [options.offsetX=0] Shadow horizontal offset
     * @param {Number} [options.offsetY=0] Shadow vertical offset
     * @return {fabric.Object} thisArg
     * @chainable
     * @see {@link http://jsfiddle.net/fabricjs/7gvJG/|jsFiddle demo}
     * @example <caption>Set shadow with string notation</caption>
     * object.setShadow('2px 2px 10px rgba(0,0,0,0.2)');
     * canvas.renderAll();
     * @example <caption>Set shadow with object notation</caption>
     * object.setShadow({
     *   color: 'red',
     *   blur: 10,
     *   offsetX: 20,
     *   offsetY: 20
     * });
     * canvas.renderAll();
     */
    setShadow: function(options) {
      return this.set('shadow', options ? new fabric.Shadow(options) : null);
    },

    /**
     * Sets "color" of an instance (alias of `set('fill', &hellip;)`)
     * @param {String} color Color value
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setColor: function(color) {
      this.set('fill', color);
      return this;
    },

    /**
     * Sets "angle" of an instance
     * @param {Number} angle Angle value (in degrees)
```

```
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setAngle: function(angle) {
      var shouldCenterOrigin = (this.originX !== 'center' || this.originY !==
'center') && this.centeredRotation;

      if (shouldCenterOrigin) {
        this._setOriginToCenter();
      }

      this.set('angle', angle);

      if (shouldCenterOrigin) {
        this._resetOrigin();
      }

      return this;
    },

    /**
     * Centers object horizontally on canvas to which it was added last.
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
     */
    centerH: function () {
      this.canvas && this.canvas.centerObjectH(this);
      return this;
    },

    /**
     * Centers object horizontally on current viewport of canvas to which it was
added last.
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
     */
    viewportCenterH: function () {
      this.canvas && this.canvas.viewportCenterObjectH(this);
      return this;
    },

    /**
     * Centers object vertically on canvas to which it was added last.
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
     */
    centerV: function () {
      this.canvas && this.canvas.centerObjectV(this);
      return this;
    },

    /**
     * Centers object vertically on current viewport of canvas to which it was
added last.
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
```

```
     */
    viewportCenterV: function () {
      this.canvas && this.canvas.viewportCenterObjectV(this);
      return this;
    },

    /**
     * Centers object vertically and horizontally on canvas to which is was
added last
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
     */
    center: function () {
      this.canvas && this.canvas.centerObject(this);
      return this;
    },

    /**
     * Centers object on current viewport of canvas to which it was added last.
     * You might need to call `setCoords` on an object after centering, to
update controls area.
     * @return {fabric.Object} thisArg
     * @chainable
     */
    viewportCenter: function () {
      this.canvas && this.canvas.viewportCenterObject(this);
      return this;
    },

    /**
     * Removes object from canvas to which it was added last
     * @return {fabric.Object} thisArg
     * @chainable
     */
    remove: function() {
      if (this.canvas) {
        if (this.group && this.group === this.canvas._activeGroup) {
          this.group.remove(this);
        }
        this.canvas.remove(this);
      }
      return this;
    },

    /**
     * Returns coordinates of a pointer relative to an object
     * @param {Event} e Event to operate upon
     * @param {Object} [pointer] Pointer to operate upon (instead of event)
     * @return {Object} Coordinates of a pointer (x, y)
     */
    getLocalPointer: function(e, pointer) {
      pointer = pointer || this.canvas.getPointer(e);
      var pClicked = new fabric.Point(pointer.x, pointer.y),
          objectLeftTop = this._getLeftTopCoords();
      if (this.angle) {
        pClicked = fabric.util.rotatePoint(
          pClicked, objectLeftTop, degreesToRadians(-this.angle));
      }
      return {
        x: pClicked.x - objectLeftTop.x,
        y: pClicked.y - objectLeftTop.y
      };
```

```
    },

    /**
     * Sets canvas globalCompositeOperation for specific object
     * custom composition operation for the particular object can be specifed
using globalCompositeOperation property
     * @param {CanvasRenderingContext2D} ctx Rendering canvas context
     */
    _setupCompositeOperation: function (ctx) {
      if (this.globalCompositeOperation) {
        ctx.globalCompositeOperation = this.globalCompositeOperation;
      }
    }
  });

  fabric.util.createAccessors(fabric.Object);

  /**
   * Alias for {@link fabric.Object.prototype.setAngle}
   * @alias rotate -> setAngle
   * @memberOf fabric.Object
   */
  fabric.Object.prototype.rotate = fabric.Object.prototype.setAngle;

  extend(fabric.Object.prototype, fabric.Observable);

  /**
   * Defines the number of fraction digits to use when serializing object
values.
   * You can use it to increase/decrease precision of such values like left,
top, scaleX, scaleY, etc.
   * @static
   * @memberOf fabric.Object
   * @constant
   * @type Number
   */
  fabric.Object.NUM_FRACTION_DIGITS = 2;

  fabric.Object._fromObject = function(className, object, callback, forceAsync,
extraParam) {
    var klass = fabric[className];
    object = clone(object, true);
    if (forceAsync) {
      fabric.util.enlivenPatterns([object.fill, object.stroke],
function(patterns) {
        if (typeof patterns[0] !== 'undefined') {
          object.fill = patterns[0];
        }
        if (typeof patterns[1] !== 'undefined') {
          object.stroke = patterns[1];
        }
        var instance = extraParam ? new klass(object[extraParam], object) : new
klass(object);
        callback && callback(instance);
      });
    }
    else {
      var instance = extraParam ? new klass(object[extraParam], object) : new
klass(object);
      callback && callback(instance);
      return instance;
    }
  };
```

```
  /**
   * Unique id used internally when creating SVG elements
   * @static
   * @memberOf fabric.Object
   * @type Number
   */
  fabric.Object.__uid = 0;

})(typeof exports !== 'undefined' ? exports : this);


(function() {

  var degreesToRadians = fabric.util.degreesToRadians,
      originXOffset = {
        left: -0.5,
        center: 0,
        right: 0.5
      },
      originYOffset = {
        top: -0.5,
        center: 0,
        bottom: 0.5
      };

  fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

    /**
     * Translates the coordinates from origin to center coordinates (based on
the object's dimensions)
     * @param {fabric.Point} point The point which corresponds to the originX
and originY params
     * @param {String} fromOriginX Horizontal origin: 'left', 'center' or
'right'
     * @param {String} fromOriginY Vertical origin: 'top', 'center' or 'bottom'
     * @param {String} toOriginX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} toOriginY Vertical origin: 'top', 'center' or 'bottom'
     * @return {fabric.Point}
     */
    translateToGivenOrigin: function(point, fromOriginX, fromOriginY, toOriginX,
toOriginY) {
      var x = point.x,
          y = point.y,
          offsetX, offsetY, dim;

      if (typeof fromOriginX === 'string') {
        fromOriginX = originXOffset[fromOriginX];
      }
      else {
        fromOriginX -= 0.5;
      }

      if (typeof toOriginX === 'string') {
        toOriginX = originXOffset[toOriginX];
      }
      else {
        toOriginX -= 0.5;
      }

      offsetX = toOriginX - fromOriginX;

      if (typeof fromOriginY === 'string') {
        fromOriginY = originYOffset[fromOriginY];
```

```
      }
      else {
        fromOriginY -= 0.5;
      }

      if (typeof toOriginY === 'string') {
        toOriginY = originYOffset[toOriginY];
      }
      else {
        toOriginY -= 0.5;
      }

      offsetY = toOriginY - fromOriginY;

      if (offsetX || offsetY) {
        dim = this._getTransformedDimensions();
        x = point.x + offsetX * dim.x;
        y = point.y + offsetY * dim.y;
      }

      return new fabric.Point(x, y);
    },

    /**
     * Translates the coordinates from origin to center coordinates (based on
the object's dimensions)
     * @param {fabric.Point} point The point which corresponds to the originX
and originY params
     * @param {String} originX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} originY Vertical origin: 'top', 'center' or 'bottom'
     * @return {fabric.Point}
     */
    translateToCenterPoint: function(point, originX, originY) {
      var p = this.translateToGivenOrigin(point, originX, originY, 'center',
'center');
      if (this.angle) {
        return fabric.util.rotatePoint(p, point, degreesToRadians(this.angle));
      }
      return p;
    },

    /**
     * Translates the coordinates from center to origin coordinates (based on
the object's dimensions)
     * @param {fabric.Point} center The point which corresponds to center of the
object
     * @param {String} originX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} originY Vertical origin: 'top', 'center' or 'bottom'
     * @return {fabric.Point}
     */
    translateToOriginPoint: function(center, originX, originY) {
      var p = this.translateToGivenOrigin(center, 'center', 'center', originX,
originY);
      if (this.angle) {
        return fabric.util.rotatePoint(p, center, degreesToRadians(this.angle));
      }
      return p;
    },

    /**
     * Returns the real center coordinates of the object
     * @return {fabric.Point}
     */
    getCenterPoint: function() {
```

```
      var leftTop = new fabric.Point(this.left, this.top);
      return this.translateToCenterPoint(leftTop, this.originX, this.originY);
    },

    /**
     * Returns the coordinates of the object based on center coordinates
     * @param {fabric.Point} point The point which corresponds to the originX
and originY params
     * @return {fabric.Point}
     */
    // getOriginPoint: function(center) {
    //   return this.translateToOriginPoint(center, this.originX, this.originY);
    // },

    /**
     * Returns the coordinates of the object as if it has a different origin
     * @param {String} originX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} originY Vertical origin: 'top', 'center' or 'bottom'
     * @return {fabric.Point}
     */
    getPointByOrigin: function(originX, originY) {
      var center = this.getCenterPoint();
      return this.translateToOriginPoint(center, originX, originY);
    },

    /**
     * Returns the point in local coordinates
     * @param {fabric.Point} point The point relative to the global coordinate
system
     * @param {String} originX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} originY Vertical origin: 'top', 'center' or 'bottom'
     * @return {fabric.Point}
     */
    toLocalPoint: function(point, originX, originY) {
      var center = this.getCenterPoint(),
          p, p2;

      if (typeof originX !== 'undefined' && typeof originY !== 'undefined' ) {
        p = this.translateToGivenOrigin(center, 'center', 'center', originX,
originY);
      }
      else {
        p = new fabric.Point(this.left, this.top);
      }

      p2 = new fabric.Point(point.x, point.y);
      if (this.angle) {
        p2 = fabric.util.rotatePoint(p2, center, -degreesToRadians(this.angle));
      }
      return p2.subtractEquals(p);
    },

    /**
     * Returns the point in global coordinates
     * @param {fabric.Point} The point relative to the local coordinate system
     * @return {fabric.Point}
     */
    // toGlobalPoint: function(point) {
    //   return fabric.util.rotatePoint(point, this.getCenterPoint(),
degreesToRadians(this.angle)).addEquals(new fabric.Point(this.left, this.top));
    // },

    /**
     * Sets the position of the object taking into consideration the object's
```

```
origin
     * @param {fabric.Point} pos The new position of the object
     * @param {String} originX Horizontal origin: 'left', 'center' or 'right'
     * @param {String} originY Vertical origin: 'top', 'center' or 'bottom'
     * @return {void}
     */
    setPositionByOrigin: function(pos, originX, originY) {
      var center = this.translateToCenterPoint(pos, originX, originY),
          position = this.translateToOriginPoint(center, this.originX,
this.originY);

      this.set('left', position.x);
      this.set('top', position.y);
    },

    /**
     * @param {String} to One of 'left', 'center', 'right'
     */
    adjustPosition: function(to) {
      var angle = degreesToRadians(this.angle),
          hypotFull = this.getWidth(),
          xFull = Math.cos(angle) * hypotFull,
          yFull = Math.sin(angle) * hypotFull,
          offsetFrom, offsetTo;

      //TODO: this function does not consider mixed situation like top, center.
      if (typeof this.originX === 'string') {
        offsetFrom = originXOffset[this.originX];
      }
      else {
        offsetFrom = this.originX - 0.5;
      }
      if (typeof to === 'string') {
        offsetTo = originXOffset[to];
      }
      else {
        offsetTo = to - 0.5;
      }
      this.left += xFull * (offsetTo - offsetFrom);
      this.top += yFull * (offsetTo - offsetFrom);
      this.setCoords();
      this.originX = to;
    },

    /**
     * Sets the origin/position of the object to it's center point
     * @private
     * @return {void}
     */
    _setOriginToCenter: function() {
      this._originalOriginX = this.originX;
      this._originalOriginY = this.originY;

      var center = this.getCenterPoint();

      this.originX = 'center';
      this.originY = 'center';

      this.left = center.x;
      this.top = center.y;
    },

    /**
     * Resets the origin/position of the object to it's original origin
```

```
     * @private
     * @return {void}
     */
    _resetOrigin: function() {
      var originPoint = this.translateToOriginPoint(
        this.getCenterPoint(),
        this._originalOriginX,
        this._originalOriginY);

      this.originX = this._originalOriginX;
      this.originY = this._originalOriginY;

      this.left = originPoint.x;
      this.top = originPoint.y;

      this._originalOriginX = null;
      this._originalOriginY = null;
    },

    /**
     * @private
     */
    _getLeftTopCoords: function() {
      return this.translateToOriginPoint(this.getCenterPoint(), 'left', 'top');
    },

    /**
    * Callback; invoked right before object is about to go from active to
inactive
    */
    onDeselect: function() {
      /* NOOP */
    }
  });

})();


(function() {

  function getCoords(coords) {
    return [
      new fabric.Point(coords.tl.x, coords.tl.y),
      new fabric.Point(coords.tr.x, coords.tr.y),
      new fabric.Point(coords.br.x, coords.br.y),
      new fabric.Point(coords.bl.x, coords.bl.y)
    ];
  }

  var degreesToRadians = fabric.util.degreesToRadians,
      multiplyMatrices = fabric.util.multiplyTransformMatrices;

  fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

    /**
     * Describe object's corner position in canvas element coordinates.
     * properties are tl,mt,tr,ml,mr,bl,mb,br,mtr for the main controls.
     * each property is an object with x, y and corner.
     * The `corner` property contains in a similar manner the 4 points of the
     * interactive area of the corner.
     * The coordinates depends from this properties: width, height, scaleX,
scaleY
     * skewX, skewY, angle, strokeWidth, viewportTransform, top, left, padding.
```

```
     * The coordinates get updated with @method setCoords.
     * You can calculate them without updating with @method calcCoords;
     * @memberOf fabric.Object.prototype
     */
    oCoords: null,

    /**
     * Describe object's corner position in canvas object absolute coordinates
     * properties are tl,tr,bl,br and describe the four main corner.
     * each property is an object with x, y, instance of Fabric.Point.
     * The coordinates depends from this properties: width, height, scaleX,
scaleY
     * skewX, skewY, angle, strokeWidth, top, left.
     * Those coordinates are usefull to understand where an object is. They get
updated
     * with oCoords but they do not need to be updated when zoom or panning
change.
     * The coordinates get updated with @method setCoords.
     * You can calculate them without updating with @method calcCoords(true);
     * @memberOf fabric.Object.prototype
     */
    aCoords: null,

    /**
     * return correct set of coordinates for intersection
     */
    getCoords: function(absolute, calculate) {
      if (!this.oCoords) {
        this.setCoords();
      }
      var coords = absolute ? this.aCoords : this.oCoords;
      return getCoords(calculate ? this.calcCoords(absolute) : coords);
    },

    /**
     * Checks if object intersects with an area formed by 2 points
     * @param {Object} pointTL top-left point of area
     * @param {Object} pointBR bottom-right point of area
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if object intersects with an area formed by 2
points
     */
    intersectsWithRect: function(pointTL, pointBR, absolute, calculate) {
      var coords = this.getCoords(absolute, calculate),
          intersection = fabric.Intersection.intersectPolygonRectangle(
            coords,
            pointTL,
            pointBR
          );
      return intersection.status === 'Intersection';
    },

    /**
     * Checks if object intersects with another object
     * @param {Object} other Object to test
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if object intersects with another object
     */
    intersectsWithObject: function(other, absolute, calculate) {
      var intersection = fabric.Intersection.intersectPolygonPolygon(
```

```
            this.getCoords(absolute, calculate),
            other.getCoords(absolute, calculate)
          );

      return intersection.status === 'Intersection'
        || other.isContainedWithinObject(this, absolute, calculate)
        || this.isContainedWithinObject(other, absolute, calculate);
    },

    /**
     * Checks if object is fully contained within area of another object
     * @param {Object} other Object to test
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if object is fully contained within area of
another object
     */
    isContainedWithinObject: function(other, absolute, calculate) {
      var points = this.getCoords(absolute, calculate),
          i = 0, lines = other._getImageLines(
            calculate ? other.calcCoords(absolute) : absolute ? other.aCoords :
other.oCoords
          );
      for (; i < 4; i++) {
        if (!other.containsPoint(points[i], lines)) {
          return false;
        }
      }
      return true;
    },

    /**
     * Checks if object is fully contained within area formed by 2 points
     * @param {Object} pointTL top-left point of area
     * @param {Object} pointBR bottom-right point of area
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if object is fully contained within area formed by
2 points
     */
    isContainedWithinRect: function(pointTL, pointBR, absolute, calculate) {
      var boundingRect = this.getBoundingRect(absolute, calculate);

      return (
        boundingRect.left >= pointTL.x &&
        boundingRect.left + boundingRect.width <= pointBR.x &&
        boundingRect.top >= pointTL.y &&
        boundingRect.top + boundingRect.height <= pointBR.y
      );
    },

    /**
     * Checks if point is inside the object
     * @param {fabric.Point} point Point to check against
     * @param {Object} [lines] object returned from @method _getImageLines
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if point is inside the object
     */
    containsPoint: function(point, lines, absolute, calculate) {
      var lines = lines || this._getImageLines(
```

```
        calculate ? this.calcCoords(absolute) : absolute ? this.aCoords :
this.oCoords
      ),
          xPoints = this._findCrossPoints(point, lines);

      // if xPoints is odd then point is inside the object
      return (xPoints !== 0 && xPoints % 2 === 1);
    },

    /**
     * Checks if object is contained within the canvas with current
viewportTransform
     * the check is done stopping at first point that appear on screen
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Boolean} true if object is fully contained within canvas
     */
    isOnScreen: function(calculate) {
      if (!this.canvas) {
        return false;
      }
      var pointTL = this.canvas.vptCoords.tl, pointBR =
this.canvas.vptCoords.br;
      var points = this.getCoords(true, calculate), point;
      for (var i = 0; i < 4; i++) {
        point = points[i];
        if (point.x <= pointBR.x && point.x >= pointTL.x && point.y <= pointBR.y
&& point.y >= pointTL.y) {
          return true;
        }
      }
      // no points on screen, check intersection with absolute coordinates
      if (this.intersectsWithRect(pointTL, pointBR, true)) {
        return true;
      }
      // worst case scenario the object is so big that contanins the screen
      var centerPoint = { x: (pointTL.x + pointBR.x) / 2, y: (pointTL.y +
pointBR.y) / 2 };
      if (this.containsPoint(centerPoint, null, true)) {
        return true;
      }
      return false;
    },

    /**
     * Method that returns an object with the object edges in it, given the
coordinates of the corners
     * @private
     * @param {Object} oCoords Coordinates of the object corners
     */
    _getImageLines: function(oCoords) {
      return {
        topline: {
          o: oCoords.tl,
          d: oCoords.tr
        },
        rightline: {
          o: oCoords.tr,
          d: oCoords.br
        },
        bottomline: {
          o: oCoords.br,
          d: oCoords.bl
        },
```

```
        leftline: {
          o: oCoords.bl,
          d: oCoords.tl
        }
      };
    },

    /**
     * Helper method to determine how many cross points are between the 4 object
edges
     * and the horizontal line determined by a point on canvas
     * @private
     * @param {fabric.Point} point Point to check
     * @param {Object} lines Coordinates of the object being evaluated
     */
     // remove yi, not used but left code here just in case.
    _findCrossPoints: function(point, lines) {
      var b1, b2, a1, a2, xi, // yi,
          xcount = 0,
          iLine;

      for (var lineKey in lines) {
        iLine = lines[lineKey];
        // optimisation 1: line below point. no cross
        if ((iLine.o.y < point.y) && (iLine.d.y < point.y)) {
          continue;
        }
        // optimisation 2: line above point. no cross
        if ((iLine.o.y >= point.y) && (iLine.d.y >= point.y)) {
          continue;
        }
        // optimisation 3: vertical line case
        if ((iLine.o.x === iLine.d.x) && (iLine.o.x >= point.x)) {
          xi = iLine.o.x;
          // yi = point.y;
        }
        // calculate the intersection point
        else {
          b1 = 0;
          b2 = (iLine.d.y - iLine.o.y) / (iLine.d.x - iLine.o.x);
          a1 = point.y - b1 * point.x;
          a2 = iLine.o.y - b2 * iLine.o.x;

          xi = -(a1 - a2) / (b1 - b2);
          // yi = a1 + b1 * xi;
        }
        // dont count xi < point.x cases
        if (xi >= point.x) {
          xcount += 1;
        }
        // optimisation 4: specific for square images
        if (xcount === 2) {
          break;
        }
      }
      return xcount;
    },

    /**
     * Returns width of an object's bounding rectangle
     * @deprecated since 1.0.4
     * @return {Number} width value
     */
    getBoundingRectWidth: function() {
```

```
      return this.getBoundingRect().width;
    },

    /**
     * Returns height of an object's bounding rectangle
     * @deprecated since 1.0.4
     * @return {Number} height value
     */
    getBoundingRectHeight: function() {
      return this.getBoundingRect().height;
    },

    /**
     * Returns coordinates of object's bounding rectangle (left, top, width,
height)
     * the box is intented as aligned to axis of canvas.
     * @param {Boolean} [absolute] use coordinates without viewportTransform
     * @param {Boolean} [calculate] use coordinates of current position instead
of .oCoords
     * @return {Object} Object with left, top, width, height properties
     */
    getBoundingRect: function(absolute, calculate) {
      var coords = this.getCoords(absolute, calculate);
      return fabric.util.makeBoundingBoxFromPoints(coords);
    },

    /**
     * Returns width of an object bounding box counting transformations
     * @return {Number} width value
     */
    getWidth: function() {
      return this._getTransformedDimensions().x;
    },

    /**
     * Returns height of an object bounding box counting transformations
     * to be renamed in 2.0
     * @return {Number} height value
     */
    getHeight: function() {
      return this._getTransformedDimensions().y;
    },

    /**
     * Makes sure the scale is valid and modifies it if necessary
     * @private
     * @param {Number} value
     * @return {Number}
     */
    _constrainScale: function(value) {
      if (Math.abs(value) < this.minScaleLimit) {
        if (value < 0) {
          return -this.minScaleLimit;
        }
        else {
          return this.minScaleLimit;
        }
      }
      return value;
    },

    /**
     * Scales an object (equally by x and y)
     * @param {Number} value Scale factor
```

```
     * @return {fabric.Object} thisArg
     * @chainable
     */
    scale: function(value) {
      value = this._constrainScale(value);

      if (value < 0) {
        this.flipX = !this.flipX;
        this.flipY = !this.flipY;
        value *= -1;
      }

      this.scaleX = value;
      this.scaleY = value;
      return this.setCoords();
    },

    /**
     * Scales an object to a given width, with respect to bounding box (scaling
by x/y equally)
     * @param {Number} value New width value
     * @return {fabric.Object} thisArg
     * @chainable
     */
    scaleToWidth: function(value) {
      // adjust to bounding rect factor so that rotated shapes would fit as well
      var boundingRectFactor = this.getBoundingRect().width / this.getWidth();
      return this.scale(value / this.width / boundingRectFactor);
    },

    /**
     * Scales an object to a given height, with respect to bounding box (scaling
by x/y equally)
     * @param {Number} value New height value
     * @return {fabric.Object} thisArg
     * @chainable
     */
    scaleToHeight: function(value) {
      // adjust to bounding rect factor so that rotated shapes would fit as well
      var boundingRectFactor = this.getBoundingRect().height / this.getHeight();
      return this.scale(value / this.height / boundingRectFactor);
    },

    /**
     * Calculate and returns the .coords of an object.
     * @return {Object} Object with tl, tr, br, bl ....
     * @chainable
     */
    calcCoords: function(absolute) {
      var theta = degreesToRadians(this.angle),
          vpt = this.getViewportTransform(),
          dim = absolute ? this._getTransformedDimensions() :
this._calculateCurrentDimensions(),
          currentWidth = dim.x, currentHeight = dim.y,
          sinTh = Math.sin(theta),
          cosTh = Math.cos(theta),
          _angle = currentWidth > 0 ? Math.atan(currentHeight / currentWidth) :
0,
          _hypotenuse = (currentWidth / Math.cos(_angle)) / 2,
          offsetX = Math.cos(_angle + theta) * _hypotenuse,
          offsetY = Math.sin(_angle + theta) * _hypotenuse,
          center = this.getCenterPoint(),
          // offset added for rotate and scale actions
          coords = absolute ? center : fabric.util.transformPoint(center, vpt),
```

```
          tl  = new fabric.Point(coords.x - offsetX, coords.y - offsetY),
          tr  = new fabric.Point(tl.x + (currentWidth * cosTh), tl.y +
(currentWidth * sinTh)),
          bl  = new fabric.Point(tl.x - (currentHeight * sinTh), tl.y +
(currentHeight * cosTh)),
          br  = new fabric.Point(coords.x + offsetX, coords.y + offsetY);
      if (!absolute) {
        var ml  = new fabric.Point((tl.x + bl.x) / 2, (tl.y + bl.y) / 2),
            mt  = new fabric.Point((tr.x + tl.x) / 2, (tr.y + tl.y) / 2),
            mr  = new fabric.Point((br.x + tr.x) / 2, (br.y + tr.y) / 2),
            mb  = new fabric.Point((br.x + bl.x) / 2, (br.y + bl.y) / 2),
            mtr = new fabric.Point(mt.x + sinTh * this.rotatingPointOffset, mt.y
- cosTh * this.rotatingPointOffset);
      }

      // debugging

      /* setTimeout(function() {
         canvas.contextTop.fillStyle = 'green';
         canvas.contextTop.fillRect(mb.x, mb.y, 3, 3);
         canvas.contextTop.fillRect(bl.x, bl.y, 3, 3);
         canvas.contextTop.fillRect(br.x, br.y, 3, 3);
         canvas.contextTop.fillRect(tl.x, tl.y, 3, 3);
         canvas.contextTop.fillRect(tr.x, tr.y, 3, 3);
         canvas.contextTop.fillRect(ml.x, ml.y, 3, 3);
         canvas.contextTop.fillRect(mr.x, mr.y, 3, 3);
         canvas.contextTop.fillRect(mt.x, mt.y, 3, 3);
         canvas.contextTop.fillRect(mtr.x, mtr.y, 3, 3);
       }, 50); */

      var coords = {
        // corners
        tl: tl, tr: tr, br: br, bl: bl,
      };
      if (!absolute) {
        // middle
        coords.ml = ml;
        coords.mt = mt;
        coords.mr = mr;
        coords.mb = mb;
        // rotating point
        coords.mtr = mtr;
      }
      return coords;
    },

    /**
     * Sets corner position coordinates based on current angle, width and height
     * See https://github.com/kangax/fabric.js/wiki/When-to-call-setCoords
     * @param {Boolean} [ignoreZoom] set oCoords with or without the viewport
transform.
     * @param {Boolean} [skipAbsolute] skip calculation of aCoords, usefull in
setViewportTransform
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setCoords: function(ignoreZoom, skipAbsolute) {
      this.oCoords = this.calcCoords(ignoreZoom);
      if (!skipAbsolute) {
        this.aCoords = this.calcCoords(true);
      }

      // set coordinates of the draggable boxes in the corners used to
scale/rotate the image
```

```
      ignoreZoom || (this._setCornerCoords && this._setCornerCoords());

      return this;
    },

    /**
     * calculate rotation matrix of an object
     * @return {Array} rotation matrix for the object
     */
    _calcRotateMatrix: function() {
      if (this.angle) {
        var theta = degreesToRadians(this.angle), cos = Math.cos(theta), sin =
Math.sin(theta);
        // trying to keep rounding error small, ugly but it works.
        if (cos === 6.123233995736766e-17 || cos === -1.8369701987210297e-16) {
          cos = 0;
        }
        return [cos, sin, -sin, cos, 0, 0];
      }
      return fabric.iMatrix.concat();
    },

    /**
     * calculate trasform Matrix that represent current transformation from
     * object properties.
     * @param {Boolean} [skipGroup] return transformMatrix for object and not go
upward with parents
     * @return {Array} matrix Transform Matrix for the object
     */
    calcTransformMatrix: function(skipGroup) {
      var center = this.getCenterPoint(),
          translateMatrix = [1, 0, 0, 1, center.x, center.y],
          rotateMatrix,
          dimensionMatrix = this._calcDimensionsTransformMatrix(this.skewX,
this.skewY, true),
          matrix;
      if (this.group && !skipGroup) {
        matrix = multiplyMatrices(this.group.calcTransformMatrix(),
translateMatrix);
      }
      else {
        matrix = translateMatrix;
      }
      if (this.angle) {
        rotateMatrix = this._calcRotateMatrix();
        matrix = multiplyMatrices(matrix, rotateMatrix);
      }
      matrix = multiplyMatrices(matrix, dimensionMatrix);
      return matrix;
    },

    _calcDimensionsTransformMatrix: function(skewX, skewY, flipping) {
      var skewMatrix,
          scaleX = this.scaleX * (flipping && this.flipX ? -1 : 1),
          scaleY = this.scaleY * (flipping && this.flipY ? -1 : 1),
          scaleMatrix = [scaleX, 0, 0, scaleY, 0, 0];
      if (skewX) {
        skewMatrix = [1, 0, Math.tan(degreesToRadians(skewX)), 1];
        scaleMatrix = multiplyMatrices(scaleMatrix, skewMatrix, true);
      }
      if (skewY) {
        skewMatrix = [1, Math.tan(degreesToRadians(skewY)), 0, 1];
        scaleMatrix = multiplyMatrices(scaleMatrix, skewMatrix, true);
      }
```

```
      return scaleMatrix;
    },

    /*
     * Calculate object dimensions from its properties
     * @private
     * @return {Object} .x width dimension
     * @return {Object} .y height dimension
     */
    _getNonTransformedDimensions: function() {
      var strokeWidth = this.strokeWidth,
          w = this.width + strokeWidth,
          h = this.height + strokeWidth;
      return { x: w, y: h };
    },

    /*
     * Calculate object bounding boxdimensions from its properties scale, skew.
     * @private
     * @return {Object} .x width dimension
     * @return {Object} .y height dimension
     */
    _getTransformedDimensions: function(skewX, skewY) {
      if (typeof skewX === 'undefined') {
        skewX = this.skewX;
      }
      if (typeof skewY === 'undefined') {
        skewY = this.skewY;
      }
      var dimensions = this._getNonTransformedDimensions(),
          dimX = dimensions.x / 2, dimY = dimensions.y / 2,
          points = [
            {
              x: -dimX,
              y: -dimY
            },
            {
              x: dimX,
              y: -dimY
            },
            {
              x: -dimX,
              y: dimY
            },
            {
              x: dimX,
              y: dimY
            }],
          i, transformMatrix = this._calcDimensionsTransformMatrix(skewX, skewY,
false),
          bbox;
      for (i = 0; i < points.length; i++) {
        points[i] = fabric.util.transformPoint(points[i], transformMatrix);
      }
      bbox = fabric.util.makeBoundingBoxFromPoints(points);
      return { x: bbox.width, y: bbox.height };
    },

    /*
     * Calculate object dimensions for controls. include padding and canvas zoom
     * private
     */
    _calculateCurrentDimensions: function()  {
      var vpt = this.getViewportTransform(),
```

```
          dim = this._getTransformedDimensions(),
          p = fabric.util.transformPoint(dim, vpt, true);

      return p.scalarAdd(2 * this.padding);
    },
  });
})();


fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

  /**
   * Moves an object to the bottom of the stack of drawn objects
   * @return {fabric.Object} thisArg
   * @chainable
   */
  sendToBack: function() {
    if (this.group) {
      fabric.StaticCanvas.prototype.sendToBack.call(this.group, this);
    }
    else {
      this.canvas.sendToBack(this);
    }
    return this;
  },

  /**
   * Moves an object to the top of the stack of drawn objects
   * @return {fabric.Object} thisArg
   * @chainable
   */
  bringToFront: function() {
    if (this.group) {
      fabric.StaticCanvas.prototype.bringToFront.call(this.group, this);
    }
    else {
      this.canvas.bringToFront(this);
    }
    return this;
  },

  /**
   * Moves an object down in stack of drawn objects
   * @param {Boolean} [intersecting] If `true`, send object behind next lower
intersecting object
   * @return {fabric.Object} thisArg
   * @chainable
   */
  sendBackwards: function(intersecting) {
    if (this.group) {
      fabric.StaticCanvas.prototype.sendBackwards.call(this.group, this,
intersecting);
    }
    else {
      this.canvas.sendBackwards(this, intersecting);
    }
    return this;
  },

  /**
   * Moves an object up in stack of drawn objects
   * @param {Boolean} [intersecting] If `true`, send object in front of next
upper intersecting object
```

```
   * @return {fabric.Object} thisArg
   * @chainable
   */
  bringForward: function(intersecting) {
    if (this.group) {
      fabric.StaticCanvas.prototype.bringForward.call(this.group, this,
intersecting);
    }
    else {
      this.canvas.bringForward(this, intersecting);
    }
    return this;
  },

  /**
   * Moves an object to specified level in stack of drawn objects
   * @param {Number} index New position of object
   * @return {fabric.Object} thisArg
   * @chainable
   */
  moveTo: function(index) {
    if (this.group) {
      fabric.StaticCanvas.prototype.moveTo.call(this.group, this, index);
    }
    else {
      this.canvas.moveTo(this, index);
    }
    return this;
  }
});


/* _TO_SVG_START_ */
(function() {

  function getSvgColorString(prop, value) {
    if (!value) {
      return prop + ': none; ';
    }
    else if (value.toLive) {
      return prop + ': url(#SVGID_' + value.id + '); ';
    }
    else {
      var color = new fabric.Color(value),
          str = prop + ': ' + color.toRgb() + '; ',
          opacity = color.getAlpha();
      if (opacity !== 1) {
        //change the color in rgb + opacity
        str += prop + '-opacity: ' + opacity.toString() + '; ';
      }
      return str;
    }
  }

  fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {
    /**
     * Returns styles-string for svg-export
     * @param {Boolean} skipShadow a boolean to skip shadow filter output
     * @return {String}
     */
    getSvgStyles: function(skipShadow) {

      var fillRule = this.fillRule,
```

```
          strokeWidth = this.strokeWidth ? this.strokeWidth : '0',
          strokeDashArray = this.strokeDashArray ? this.strokeDashArray.join('
') : 'none',
          strokeLineCap = this.strokeLineCap ? this.strokeLineCap : 'butt',
          strokeLineJoin = this.strokeLineJoin ? this.strokeLineJoin : 'miter',
          strokeMiterLimit = this.strokeMiterLimit ? this.strokeMiterLimit :
'4',
          opacity = typeof this.opacity !== 'undefined' ? this.opacity : '1',
          visibility = this.visible ? '' : ' visibility: hidden;',
          filter = skipShadow ? '' : this.getSvgFilter(),
          fill = getSvgColorString('fill', this.fill),
          stroke = getSvgColorString('stroke', this.stroke);

      return [
        stroke,
        'stroke-width: ', strokeWidth, '; ',
        'stroke-dasharray: ', strokeDashArray, '; ',
        'stroke-linecap: ', strokeLineCap, '; ',
        'stroke-linejoin: ', strokeLineJoin, '; ',
        'stroke-miterlimit: ', strokeMiterLimit, '; ',
        fill,
        'fill-rule: ', fillRule, '; ',
        'opacity: ', opacity, ';',
        filter,
        visibility
      ].join('');
    },

    /**
     * Returns filter for svg shadow
     * @return {String}
     */
    getSvgFilter: function() {
      return this.shadow ? 'filter: url(#SVGID_' + this.shadow.id + ');' : '';
    },

    /**
     * Returns id attribute for svg output
     * @return {String}
     */
    getSvgId: function() {
      return this.id ? 'id="' + this.id + '" ' : '';
    },

    /**
     * Returns transform-string for svg-export
     * @return {String}
     */
    getSvgTransform: function() {
      if (this.group && this.group.type === 'path-group') {
        return '';
      }
      var toFixed = fabric.util.toFixed,
          angle = this.getAngle(),
          skewX = (this.getSkewX() % 360),
          skewY = (this.getSkewY() % 360),
          center = this.getCenterPoint(),

          NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS,

          translatePart = this.type === 'path-group' ? '' : 'translate(' +
                          toFixed(center.x, NUM_FRACTION_DIGITS) +
                          ' ' +
                          toFixed(center.y, NUM_FRACTION_DIGITS) +
```

```
                        ')',

          anglePart = angle !== 0
            ? (' rotate(' + toFixed(angle, NUM_FRACTION_DIGITS) + ')')
            : '',

          scalePart = (this.scaleX === 1 && this.scaleY === 1)
            ? '' :
            (' scale(' +
              toFixed(this.scaleX, NUM_FRACTION_DIGITS) +
              ' ' +
              toFixed(this.scaleY, NUM_FRACTION_DIGITS) +
            ')'),

          skewXPart = skewX !== 0 ? ' skewX(' + toFixed(skewX,
NUM_FRACTION_DIGITS) + ')' : '',

          skewYPart = skewY !== 0 ? ' skewY(' + toFixed(skewY,
NUM_FRACTION_DIGITS) + ')' : '',

          addTranslateX = this.type === 'path-group' ? this.width : 0,

          flipXPart = this.flipX ? ' matrix(-1 0 0 1 ' + addTranslateX + ' 0)
' : '',

          addTranslateY = this.type === 'path-group' ? this.height : 0,

          flipYPart = this.flipY ? ' matrix(1 0 0 -1 0 ' + addTranslateY + ')' :
'';

      return [
        translatePart, anglePart, scalePart, flipXPart, flipYPart, skewXPart,
skewYPart
      ].join('');
    },

    /**
     * Returns transform-string for svg-export from the transform matrix of
single elements
     * @return {String}
     */
    getSvgTransformMatrix: function() {
      return this.transformMatrix ? ' matrix(' + this.transformMatrix.join(' ')
+ ') ' : '';
    },

    /**
     * @private
     */
    _createBaseSVGMarkup: function() {
      var markup = [];

      if (this.fill && this.fill.toLive) {
        markup.push(this.fill.toSVG(this, false));
      }
      if (this.stroke && this.stroke.toLive) {
        markup.push(this.stroke.toSVG(this, false));
      }
      if (this.shadow) {
        markup.push(this.shadow.toSVG(this));
      }
      return markup;
    }
  });
```

```
})();
/* _TO_SVG_END_ */


(function() {

  var extend = fabric.util.object.extend,
      originalSet = 'stateProperties';

  /*
    Depends on `stateProperties`
  */
  function saveProps(origin, destination, props) {
    var tmpObj = { }, deep = true;
    props.forEach(function(prop) {
      tmpObj[prop] = origin[prop];
    });
    extend(origin[destination], tmpObj, deep);
  }

  function _isEqual(origValue, currentValue, firstPass) {
    if (origValue === currentValue) {
      // if the objects are identical, return
      return true;
    }
    else if (Array.isArray(origValue)) {
      if (origValue.length !== currentValue.length) {
        return false;
      }
      for (var i = 0, len = origValue.length; i < len; i++) {
        if (!_isEqual(origValue[i], currentValue[i])) {
          return false;
        }
      }
      return true;
    }
    else if (origValue && typeof origValue === 'object') {
      var keys = Object.keys(origValue), key;
      if (!firstPass && keys.length !== Object.keys(currentValue).length) {
        return false;
      }
      for (var i = 0, len = keys.length; i < len; i++) {
        key = keys[i];
        if (!_isEqual(origValue[key], currentValue[key])) {
          return false;
        }
      }
      return true;
    }
  }


  fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

    /**
     * Returns true if object state (one of its state properties) was changed
     * @param {String} [propertySet] optional name for the set of property we
want to save
     * @return {Boolean} true if instance' state has changed since `{@link
fabric.Object#saveState}` was called
     */
    hasStateChanged: function(propertySet) {
      propertySet = propertySet || originalSet;
```

```
      var dashedPropertySet = '_' + propertySet;
      if (Object.keys(this[dashedPropertySet]).length <
this[propertySet].length) {
        return true;
      }
      return !_isEqual(this[dashedPropertySet], this, true);
    },

    /**
     * Saves state of an object
     * @param {Object} [options] Object with additional `stateProperties` array
to include when saving state
     * @return {fabric.Object} thisArg
     */
    saveState: function(options) {
      var propertySet = options && options.propertySet || originalSet,
          destination = '_' + propertySet;
      if (!this[destination]) {
        return this.setupState(options);
      }
      saveProps(this, destination, this[propertySet]);
      if (options && options.stateProperties) {
        saveProps(this, destination, options.stateProperties);
      }
      return this;
    },

    /**
     * Setups state of an object
     * @param {Object} [options] Object with additional `stateProperties` array
to include when saving state
     * @return {fabric.Object} thisArg
     */
    setupState: function(options) {
      options = options || { };
      var propertySet = options.propertySet || originalSet;
      options.propertySet = propertySet;
      this['_' + propertySet] = { };
      this.saveState(options);
      return this;
    }
  });
})();


(function() {

  var degreesToRadians = fabric.util.degreesToRadians,
      /* eslint-disable camelcase */
      isVML = function() { return typeof G_vmlCanvasManager !== 'undefined'; };
      /* eslint-enable camelcase */
  fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

    /**
     * The object interactivity controls.
     * @private
     */
    _controlsVisibility: null,

    /**
     * Determines which corner has been clicked
     * @private
     * @param {Object} pointer The pointer indicating the mouse position
```

```
     * @return {String|Boolean} corner code (tl, tr, bl, br, etc.), or false if
nothing is found
     */
    _findTargetCorner: function(pointer) {
      if (!this.hasControls || !this.active) {
        return false;
      }

      var ex = pointer.x,
          ey = pointer.y,
          xPoints,
          lines;
      this.__corner = 0;
      for (var i in this.oCoords) {

        if (!this.isControlVisible(i)) {
          continue;
        }

        if (i === 'mtr' && !this.hasRotatingPoint) {
          continue;
        }

        if (this.get('lockUniScaling') &&
           (i === 'mt' || i === 'mr' || i === 'mb' || i === 'ml')) {
          continue;
        }

        lines = this._getImageLines(this.oCoords[i].corner);

        // debugging

        // canvas.contextTop.fillRect(lines.bottomline.d.x,
lines.bottomline.d.y, 2, 2);
        // canvas.contextTop.fillRect(lines.bottomline.o.x,
lines.bottomline.o.y, 2, 2);

        // canvas.contextTop.fillRect(lines.leftline.d.x, lines.leftline.d.y, 2,
2);
        // canvas.contextTop.fillRect(lines.leftline.o.x, lines.leftline.o.y, 2,
2);

        // canvas.contextTop.fillRect(lines.topline.d.x, lines.topline.d.y, 2,
2);
        // canvas.contextTop.fillRect(lines.topline.o.x, lines.topline.o.y, 2,
2);

        // canvas.contextTop.fillRect(lines.rightline.d.x, lines.rightline.d.y,
2, 2);
        // canvas.contextTop.fillRect(lines.rightline.o.x, lines.rightline.o.y,
2, 2);

        xPoints = this._findCrossPoints({ x: ex, y: ey }, lines);
        if (xPoints !== 0 && xPoints % 2 === 1) {
          this.__corner = i;
          return i;
        }
      }
      return false;
    },

    /**
     * Sets the coordinates of the draggable boxes in the corners of
     * the image used to scale/rotate it.
```

```
     * @private
     */
    _setCornerCoords: function() {
      var coords = this.oCoords,
          newTheta = degreesToRadians(45 - this.angle),
          /* Math.sqrt(2 * Math.pow(this.cornerSize, 2)) / 2, */
          /* 0.707106 stands for sqrt(2)/2 */
          cornerHypotenuse = this.cornerSize * 0.707106,
          cosHalfOffset = cornerHypotenuse * Math.cos(newTheta),
          sinHalfOffset = cornerHypotenuse * Math.sin(newTheta),
          x, y;

      for (var point in coords) {
        x = coords[point].x;
        y = coords[point].y;
        coords[point].corner = {
          tl: {
            x: x - sinHalfOffset,
            y: y - cosHalfOffset
          },
          tr: {
            x: x + cosHalfOffset,
            y: y - sinHalfOffset
          },
          bl: {
            x: x - cosHalfOffset,
            y: y + sinHalfOffset
          },
          br: {
            x: x + sinHalfOffset,
            y: y + cosHalfOffset
          }
        };
      }
    },

    /**
     * Draws a colored layer behind the object, inside its selection borders.
     * Requires public options: padding, selectionBackgroundColor
     * this function is called when the context is transformed
     * has checks to be skipped when the object is on a staticCanvas
     * @param {CanvasRenderingContext2D} ctx Context to draw on
     * @return {fabric.Object} thisArg
     * @chainable
     */
    drawSelectionBackground: function(ctx) {
      if (!this.selectionBackgroundColor || this.group || !this.active ||
        (this.canvas && !this.canvas.interactive)) {
        return this;
      }
      ctx.save();
      var center = this.getCenterPoint(), wh =
this._calculateCurrentDimensions(),
          vpt = this.canvas.viewportTransform;
      ctx.translate(center.x, center.y);
      ctx.scale(1 / vpt[0], 1 / vpt[3]);
      ctx.rotate(degreesToRadians(this.angle));
      ctx.fillStyle = this.selectionBackgroundColor;
      ctx.fillRect(-wh.x / 2, -wh.y / 2, wh.x, wh.y);
      ctx.restore();
      return this;
    },

    /**
```

```
     * Draws borders of an object's bounding box.
     * Requires public properties: width, height
     * Requires public options: padding, borderColor
     * @param {CanvasRenderingContext2D} ctx Context to draw on
     * @return {fabric.Object} thisArg
     * @chainable
     */
    drawBorders: function(ctx) {
      if (!this.hasBorders) {
        return this;
      }

      var wh = this._calculateCurrentDimensions(),
          strokeWidth = 1 / this.borderScaleFactor,
          width = wh.x + strokeWidth,
          height = wh.y + strokeWidth;

      ctx.save();
      ctx.strokeStyle = this.borderColor;
      this._setLineDash(ctx, this.borderDashArray, null);

      ctx.strokeRect(
        -width / 2,
        -height / 2,
        width,
        height
      );

      if (this.hasRotatingPoint && this.isControlVisible('mtr') && !
this.get('lockRotation') && this.hasControls) {

        var rotateHeight = -height / 2;

        ctx.beginPath();
        ctx.moveTo(0, rotateHeight);
        ctx.lineTo(0, rotateHeight - this.rotatingPointOffset);
        ctx.closePath();
        ctx.stroke();
      }

      ctx.restore();
      return this;
    },

    /**
     * Draws borders of an object's bounding box when it is inside a group.
     * Requires public properties: width, height
     * Requires public options: padding, borderColor
     * @param {CanvasRenderingContext2D} ctx Context to draw on
     * @param {object} options object representing current object parameters
     * @return {fabric.Object} thisArg
     * @chainable
     */
    drawBordersInGroup: function(ctx, options) {
      if (!this.hasBorders) {
        return this;
      }

      var p = this._getNonTransformedDimensions(),
          matrix = fabric.util.customTransformMatrix(options.scaleX,
options.scaleY, options.skewX),
          wh = fabric.util.transformPoint(p, matrix),
          strokeWidth = 1 / this.borderScaleFactor,
          width = wh.x + strokeWidth,
```

```
      height = wh.y + strokeWidth;

  ctx.save();
  this._setLineDash(ctx, this.borderDashArray, null);
  ctx.strokeStyle = this.borderColor;

  ctx.strokeRect(
    -width / 2,
    -height / 2,
    width,
    height
  );

  ctx.restore();
  return this;
},

/**
 * Draws corners of an object's bounding box.
 * Requires public properties: width, height
 * Requires public options: cornerSize, padding
 * @param {CanvasRenderingContext2D} ctx Context to draw on
 * @return {fabric.Object} thisArg
 * @chainable
 */
drawControls: function(ctx) {
  if (!this.hasControls) {
    return this;
  }

  var wh = this._calculateCurrentDimensions(),
      width = wh.x,
      height = wh.y,
      scaleOffset = this.cornerSize,
      left = -(width + scaleOffset) / 2,
      top = -(height + scaleOffset) / 2,
      methodName = this.transparentCorners ? 'stroke' : 'fill';

  ctx.save();
  ctx.strokeStyle = ctx.fillStyle = this.cornerColor;
  if (!this.transparentCorners) {
    ctx.strokeStyle = this.cornerStrokeColor;
  }
  this._setLineDash(ctx, this.cornerDashArray, null);

  // top-left
  this._drawControl('tl', ctx, methodName,
    left,
    top);

  // top-right
  this._drawControl('tr', ctx, methodName,
    left + width,
    top);

  // bottom-left
  this._drawControl('bl', ctx, methodName,
    left,
    top + height);

  // bottom-right
  this._drawControl('br', ctx, methodName,
    left + width,
    top + height);
```

```
      if (!this.get('lockUniScaling')) {

        // middle-top
        this._drawControl('mt', ctx, methodName,
          left + width / 2,
          top);

        // middle-bottom
        this._drawControl('mb', ctx, methodName,
          left + width / 2,
          top + height);

        // middle-right
        this._drawControl('mr', ctx, methodName,
          left + width,
          top + height / 2);

        // middle-left
        this._drawControl('ml', ctx, methodName,
          left,
          top + height / 2);
      }

      // middle-top-rotate
      if (this.hasRotatingPoint) {
        this._drawControl('mtr', ctx, methodName,
          left + width / 2,
          top - this.rotatingPointOffset);
      }

      ctx.restore();

      return this;
    },

    /**
     * @private
     */
    _drawControl: function(control, ctx, methodName, left, top) {
      if (!this.isControlVisible(control)) {
        return;
      }
      var size = this.cornerSize, stroke = !this.transparentCorners &&
this.cornerStrokeColor;
      switch (this.cornerStyle) {
        case 'circle':
          ctx.beginPath();
          ctx.arc(left + size / 2, top + size / 2, size / 2, 0, 2 * Math.PI,
false);
          ctx[methodName]();
          if (stroke) {
            ctx.stroke();
          }
          break;
        default:
          isVML() || this.transparentCorners || ctx.clearRect(left, top, size,
size);
          ctx[methodName + 'Rect'](left, top, size, size);
          if (stroke) {
            ctx.strokeRect(left, top, size, size);
          }
      }
    },
```

```
    /**
     * Returns true if the specified control is visible, false otherwise.
     * @param {String} controlName The name of the control. Possible values are
'tl', 'tr', 'br', 'bl', 'ml', 'mt', 'mr', 'mb', 'mtr'.
     * @returns {Boolean} true if the specified control is visible, false
otherwise
     */
    isControlVisible: function(controlName) {
      return this._getControlsVisibility()[controlName];
    },

    /**
     * Sets the visibility of the specified control.
     * @param {String} controlName The name of the control. Possible values are
'tl', 'tr', 'br', 'bl', 'ml', 'mt', 'mr', 'mb', 'mtr'.
     * @param {Boolean} visible true to set the specified control visible, false
otherwise
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setControlVisible: function(controlName, visible) {
      this._getControlsVisibility()[controlName] = visible;
      return this;
    },

    /**
     * Sets the visibility state of object controls.
     * @param {Object} [options] Options object
     * @param {Boolean} [options.bl] true to enable the bottom-left control,
false to disable it
     * @param {Boolean} [options.br] true to enable the bottom-right control,
false to disable it
     * @param {Boolean} [options.mb] true to enable the middle-bottom control,
false to disable it
     * @param {Boolean} [options.ml] true to enable the middle-left control,
false to disable it
     * @param {Boolean} [options.mr] true to enable the middle-right control,
false to disable it
     * @param {Boolean} [options.mt] true to enable the middle-top control,
false to disable it
     * @param {Boolean} [options.tl] true to enable the top-left control, false
to disable it
     * @param {Boolean} [options.tr] true to enable the top-right control, false
to disable it
     * @param {Boolean} [options.mtr] true to enable the middle-top-rotate
control, false to disable it
     * @return {fabric.Object} thisArg
     * @chainable
     */
    setControlsVisibility: function(options) {
      options || (options = { });

      for (var p in options) {
        this.setControlVisible(p, options[p]);
      }
      return this;
    },

    /**
     * Returns the instance of the control visibility set for this object.
     * @private
     * @returns {Object}
     */
```

```
    _getControlsVisibility: function() {
      if (!this._controlsVisibility) {
        this._controlsVisibility = {
          tl: true,
          tr: true,
          br: true,
          bl: true,
          ml: true,
          mt: true,
          mr: true,
          mb: true,
          mtr: true
        };
      }
      return this._controlsVisibility;
    }
  });
})();


fabric.util.object.extend(fabric.StaticCanvas.prototype, /** @lends
fabric.StaticCanvas.prototype */ {

  /**
   * Animation duration (in ms) for fx* methods
   * @type Number
   * @default
   */
  FX_DURATION: 500,

  /**
   * Centers object horizontally with animation.
   * @param {fabric.Object} object Object to center
   * @param {Object} [callbacks] Callbacks object with optional "onComplete"
and/or "onChange" properties
   * @param {Function} [callbacks.onComplete] Invoked on completion
   * @param {Function} [callbacks.onChange] Invoked on every step of animation
   * @return {fabric.Canvas} thisArg
   * @chainable
   */
  fxCenterObjectH: function (object, callbacks) {
    callbacks = callbacks || { };

    var empty = function() { },
        onComplete = callbacks.onComplete || empty,
        onChange = callbacks.onChange || empty,
        _this = this;

    fabric.util.animate({
      startValue: object.get('left'),
      endValue: this.getCenter().left,
      duration: this.FX_DURATION,
      onChange: function(value) {
        object.set('left', value);
        _this.renderAll();
        onChange();
      },
      onComplete: function() {
        object.setCoords();
        onComplete();
      }
    });

    return this;
```

```
  },

  /**
   * Centers object vertically with animation.
   * @param {fabric.Object} object Object to center
   * @param {Object} [callbacks] Callbacks object with optional "onComplete"
and/or "onChange" properties
   * @param {Function} [callbacks.onComplete] Invoked on completion
   * @param {Function} [callbacks.onChange] Invoked on every step of animation
   * @return {fabric.Canvas} thisArg
   * @chainable
   */
  fxCenterObjectV: function (object, callbacks) {
    callbacks = callbacks || { };

    var empty = function() { },
        onComplete = callbacks.onComplete || empty,
        onChange = callbacks.onChange || empty,
        _this = this;

    fabric.util.animate({
      startValue: object.get('top'),
      endValue: this.getCenter().top,
      duration: this.FX_DURATION,
      onChange: function(value) {
        object.set('top', value);
        _this.renderAll();
        onChange();
      },
      onComplete: function() {
        object.setCoords();
        onComplete();
      }
    });

    return this;
  },

  /**
   * Same as `fabric.Canvas#remove` but animated
   * @param {fabric.Object} object Object to remove
   * @param {Object} [callbacks] Callbacks object with optional "onComplete"
and/or "onChange" properties
   * @param {Function} [callbacks.onComplete] Invoked on completion
   * @param {Function} [callbacks.onChange] Invoked on every step of animation
   * @return {fabric.Canvas} thisArg
   * @chainable
   */
  fxRemove: function (object, callbacks) {
    callbacks = callbacks || { };

    var empty = function() { },
        onComplete = callbacks.onComplete || empty,
        onChange = callbacks.onChange || empty,
        _this = this;

    fabric.util.animate({
      startValue: object.get('opacity'),
      endValue: 0,
      duration: this.FX_DURATION,
      onStart: function() {
        object.set('active', false);
      },
      onChange: function(value) {
```

```
        object.set('opacity', value);
        _this.renderAll();
        onChange();
      },
      onComplete: function () {
        _this.remove(object);
        onComplete();
      }
    });

    return this;
  }
});

fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {
  /**
   * Animates object's properties
   * @param {String|Object} property Property to animate (if string) or
properties to animate (if object)
   * @param {Number|Object} value Value to animate property to (if string was
given first) or options object
   * @return {fabric.Object} thisArg
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-2#animation}
   * @chainable
   *
   * As object — multiple properties
   *
   * object.animate({ left: ..., top: ... });
   * object.animate({ left: ..., top: ... }, { duration: ... });
   *
   * As string — one property
   *
   * object.animate('left', ...);
   * object.animate('left', { duration: ... });
   *
   */
  animate: function() {
    if (arguments[0] && typeof arguments[0] === 'object') {
      var propsToAnimate = [], prop, skipCallbacks;
      for (prop in arguments[0]) {
        propsToAnimate.push(prop);
      }
      for (var i = 0, len = propsToAnimate.length; i < len; i++) {
        prop = propsToAnimate[i];
        skipCallbacks = i !== len - 1;
        this._animate(prop, arguments[0][prop], arguments[1], skipCallbacks);
      }
    }
    else {
      this._animate.apply(this, arguments);
    }
    return this;
  },

  /**
   * @private
   * @param {String} property Property to animate
   * @param {String} to Value to animate to
   * @param {Object} [options] Options object
   * @param {Boolean} [skipCallbacks] When true, callbacks like onchange and
oncomplete are not invoked
   */
  _animate: function(property, to, options, skipCallbacks) {
```

```
    var _this = this, propPair;

    to = to.toString();

    if (!options) {
      options = { };
    }
    else {
      options = fabric.util.object.clone(options);
    }

    if (~property.indexOf('.')) {
      propPair = property.split('.');
    }

    var currentValue = propPair
      ? this.get(propPair[0])[propPair[1]]
      : this.get(property);

    if (!('from' in options)) {
      options.from = currentValue;
    }

    if (~to.indexOf('=')) {
      to = currentValue + parseFloat(to.replace('=', ''));
    }
    else {
      to = parseFloat(to);
    }

    fabric.util.animate({
      startValue: options.from,
      endValue: to,
      byValue: options.by,
      easing: options.easing,
      duration: options.duration,
      abort: options.abort && function() {
        return options.abort.call(_this);
      },
      onChange: function(value, valueProgress, timeProgress) {
        if (propPair) {
          _this[propPair[0]][propPair[1]] = value;
        }
        else {
          _this.set(property, value);
        }
        if (skipCallbacks) {
          return;
        }
        options.onChange && options.onChange(value, valueProgress,
timeProgress);
      },
      onComplete: function(value, valueProgress, timeProgress) {
        if (skipCallbacks) {
          return;
        }

        _this.setCoords();
        options.onComplete && options.onComplete(value, valueProgress,
timeProgress);
      }
    });
  }
});
```

```
(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      clone = fabric.util.object.clone,
      coordProps = { x1: 1, x2: 1, y1: 1, y2: 1 },
      supportsLineDash = fabric.StaticCanvas.supports('setLineDash');

  if (fabric.Line) {
    fabric.warn('fabric.Line is already defined');
    return;
  }

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push(
    'x1',
    'x2',
    'y1',
    'y2'
  );

  /**
   * Line class
   * @class fabric.Line
   * @extends fabric.Object
   * @see {@link fabric.Line#initialize} for constructor definition
   */
  fabric.Line = fabric.util.createClass(fabric.Object, /** @lends
fabric.Line.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'line',

    /**
     * x value or first line edge
     * @type Number
     * @default
     */
    x1: 0,

    /**
     * y value or first line edge
     * @type Number
     * @default
     */
    y1: 0,

    /**
     * x value or second line edge
     * @type Number
     * @default
     */
    x2: 0,

    /**
     * y value or second line edge
```

```
     * @type Number
     * @default
     */
    y2: 0,

    cacheProperties: cacheProperties,

    /**
     * Constructor
     * @param {Array} [points] Array of points
     * @param {Object} [options] Options object
     * @return {fabric.Line} thisArg
     */
    initialize: function(points, options) {
      if (!points) {
        points = [0, 0, 0, 0];
      }

      this.callSuper('initialize', options);

      this.set('x1', points[0]);
      this.set('y1', points[1]);
      this.set('x2', points[2]);
      this.set('y2', points[3]);

      this._setWidthHeight(options);
    },

    /**
     * @private
     * @param {Object} [options] Options
     */
    _setWidthHeight: function(options) {
      options || (options = { });

      this.width = Math.abs(this.x2 - this.x1);
      this.height = Math.abs(this.y2 - this.y1);

      this.left = 'left' in options
        ? options.left
        : this._getLeftToOriginX();

      this.top = 'top' in options
        ? options.top
        : this._getTopToOriginY();
    },

    /**
     * @private
     * @param {String} key
     * @param {*} value
     */
    _set: function(key, value) {
      this.callSuper('_set', key, value);
      if (typeof coordProps[key] !== 'undefined') {
        this._setWidthHeight();
      }
      return this;
    },

    /**
     * @private
     * @return {Number} leftToOriginX Distance from left edge of canvas to
originX of Line.
```

```
     */
    _getLeftToOriginX: makeEdgeToOriginGetter(
      { // property names
        origin: 'originX',
        axis1: 'x1',
        axis2: 'x2',
        dimension: 'width'
      },
      { // possible values of origin
        nearest: 'left',
        center: 'center',
        farthest: 'right'
      }
    ),

    /**
     * @private
     * @return {Number} topToOriginY Distance from top edge of canvas to originY
of Line.
     */
    _getTopToOriginY: makeEdgeToOriginGetter(
      { // property names
        origin: 'originY',
        axis1: 'y1',
        axis2: 'y2',
        dimension: 'height'
      },
      { // possible values of origin
        nearest: 'top',
        center: 'center',
        farthest: 'bottom'
      }
    ),

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    _render: function(ctx, noTransform) {
      ctx.beginPath();

      if (noTransform) {
        //  Line coords are distances from left-top of canvas to origin of line.
        //  To render line in a path-group, we need to translate them to
        //  distances from center of path-group to center of line.
        var cp = this.getCenterPoint(),
            offset = this.strokeWidth / 2;
        ctx.translate(
          cp.x - (this.strokeLineCap === 'butt' && this.height === 0 ? 0 :
offset),
          cp.y - (this.strokeLineCap === 'butt' && this.width === 0 ? 0 :
offset)
        );
      }

      if (!this.strokeDashArray || this.strokeDashArray && supportsLineDash) {
        // move from center (of virtual box) to its left/top corner
        // we can't assume x1, y1 is top left and x2, y2 is bottom right
        var p = this.calcLinePoints();
        ctx.moveTo(p.x1, p.y1);
        ctx.lineTo(p.x2, p.y2);
      }
```

```
      ctx.lineWidth = this.strokeWidth;

      // TODO: test this
      // make sure setting "fill" changes color of a line
      // (by copying fillStyle to strokeStyle, since line is stroked, not
filled)
      var origStrokeStyle = ctx.strokeStyle;
      ctx.strokeStyle = this.stroke || ctx.fillStyle;
      this.stroke && this._renderStroke(ctx);
      ctx.strokeStyle = origStrokeStyle;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      var p = this.calcLinePoints();

      ctx.beginPath();
      fabric.util.drawDashedLine(ctx, p.x1, p.y1, p.x2, p.y2,
this.strokeDashArray);
      ctx.closePath();
    },

    /**
     * Returns object representation of an instance
     * @methd toObject
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      return extend(this.callSuper('toObject', propertiesToInclude),
this.calcLinePoints());
    },

    /*
     * Calculate object dimensions from its properties
     * @private
     */
    _getNonTransformedDimensions: function() {
      var dim = this.callSuper('_getNonTransformedDimensions');
      if (this.strokeLineCap === 'butt') {
        if (this.width === 0) {
          dim.y -= this.strokeWidth;
        }
        if (this.height === 0) {
          dim.x -= this.strokeWidth;
        }
      }
      return dim;
    },

    /**
     * Recalculates line points given width and height
     * @private
     */
    calcLinePoints: function() {
      var xMult = this.x1 <= this.x2 ? -1 : 1,
          yMult = this.y1 <= this.y2 ? -1 : 1,
          x1 = (xMult * this.width * 0.5),
          y1 = (yMult * this.height * 0.5),
          x2 = (xMult * this.width * -0.5),
```

```
          y2 = (yMult * this.height * -0.5);

      return {
        x1: x1,
        x2: x2,
        y1: y1,
        y2: y2
      };
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(),
          p = { x1: this.x1, x2: this.x2, y1: this.y1, y2: this.y2 };

      if (!(this.group && this.group.type === 'path-group')) {
        p = this.calcLinePoints();
      }
      markup.push(
        '<line ', this.getSvgId(),
          'x1="', p.x1,
          '" y1="', p.y1,
          '" x2="', p.x2,
          '" y2="', p.y2,
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(),
          this.getSvgTransformMatrix(),
        '"/>\n'
      );

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
{@link fabric.Line.fromElement})
   * @static
   * @memberOf fabric.Line
   * @see http://www.w3.org/TR/SVG/shapes.html#LineElement
   */
  fabric.Line.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat('x1 y1 x2
y2'.split(' '));

  /**
   * Returns fabric.Line instance from an SVG element
   * @static
   * @memberOf fabric.Line
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Line} instance of fabric.Line
   */
  fabric.Line.fromElement = function(element, options) {
    options = options || { };
    var parsedAttributes = fabric.parseAttributes(element,
fabric.Line.ATTRIBUTE_NAMES),
```

```
        points = [
          parsedAttributes.x1 || 0,
          parsedAttributes.y1 || 0,
          parsedAttributes.x2 || 0,
          parsedAttributes.y2 || 0
        ];
    options.originX = 'left';
    options.originY = 'top';
    return new fabric.Line(points, extend(parsedAttributes, options));
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns fabric.Line instance from an object representation
   * @static
   * @memberOf fabric.Line
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] invoked with new instance as first argument
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Line} instance of fabric.Line
   */
  fabric.Line.fromObject = function(object, callback, forceAsync) {
    function _callback(instance) {
      delete instance.points;
      callback && callback(instance);
    };
    var options = clone(object, true);
    options.points = [object.x1, object.y1, object.x2, object.y2];
    var line = fabric.Object._fromObject('Line', options, _callback, forceAsync,
'points');
    if (line) {
      delete line.points;
    }
    return line;
  };

  /**
   * Produces a function that calculates distance from canvas edge to Line
origin.
   */
  function makeEdgeToOriginGetter(propertyNames, originValues) {
    var origin = propertyNames.origin,
        axis1 = propertyNames.axis1,
        axis2 = propertyNames.axis2,
        dimension = propertyNames.dimension,
        nearest = originValues.nearest,
        center = originValues.center,
        farthest = originValues.farthest;

    return function() {
      switch (this.get(origin)) {
        case nearest:
          return Math.min(this.get(axis1), this.get(axis2));
        case center:
          return Math.min(this.get(axis1), this.get(axis2)) + (0.5 *
this.get(dimension));
        case farthest:
          return Math.max(this.get(axis1), this.get(axis2));
      }
    };

  }
```

```
})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      pi = Math.PI,
      extend = fabric.util.object.extend;

  if (fabric.Circle) {
    fabric.warn('fabric.Circle is already defined.');
    return;
  }

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push(
    'radius'
  );

  /**
   * Circle class
   * @class fabric.Circle
   * @extends fabric.Object
   * @see {@link fabric.Circle#initialize} for constructor definition
   */
  fabric.Circle = fabric.util.createClass(fabric.Object, /** @lends
fabric.Circle.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'circle',

    /**
     * Radius of this circle
     * @type Number
     * @default
     */
    radius: 0,

    /**
     * Start angle of the circle, moving clockwise
     * @type Number
     * @default 0
     */
    startAngle: 0,

    /**
     * End angle of the circle
     * @type Number
     * @default 2Pi
     */
    endAngle: pi * 2,

    cacheProperties: cacheProperties,

    /**
     * Constructor
     * @param {Object} [options] Options object
     * @return {fabric.Circle} thisArg
```

```
     */
    initialize: function(options) {
      this.callSuper('initialize', options);
      this.set('radius', options && options.radius || 0);
    },

    /**
     * @private
     * @param {String} key
     * @param {*} value
     * @return {fabric.Circle} thisArg
     */
    _set: function(key, value) {
      this.callSuper('_set', key, value);

      if (key === 'radius') {
        this.setRadius(value);
      }

      return this;
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      return this.callSuper('toObject', ['radius', 'startAngle',
'endAngle'].concat(propertiesToInclude));
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(), x = 0, y = 0,
          angle = (this.endAngle - this.startAngle) % ( 2 * pi);

      if (angle === 0) {
        if (this.group && this.group.type === 'path-group') {
          x = this.left + this.radius;
          y = this.top + this.radius;
        }
        markup.push(
          '<circle ', this.getSvgId(),
            'cx="' + x + '" cy="' + y + '" ',
            'r="', this.radius,
            '" style="', this.getSvgStyles(),
            '" transform="', this.getSvgTransform(),
            ' ', this.getSvgTransformMatrix(),
          '"/>\n'
        );
      }
      else {
        var startX = Math.cos(this.startAngle) * this.radius,
            startY = Math.sin(this.startAngle) * this.radius,
            endX = Math.cos(this.endAngle) * this.radius,
            endY = Math.sin(this.endAngle) * this.radius,
```

```
            largeFlag = angle > pi ? '1' : '0';

          markup.push(
            '<path d="M ' + startX + ' ' + startY,
            ' A ' + this.radius + ' ' + this.radius,
            ' 0 ', +largeFlag + ' 1', ' ' + endX + ' ' + endY,
            '" style="', this.getSvgStyles(),
            '" transform="', this.getSvgTransform(),
            ' ', this.getSvgTransformMatrix(),
            '"/>\n'
          );
        }

        return reviver ? reviver(markup.join('')) : markup.join('');
      },
      /* _TO_SVG_END_ */

      /**
       * @private
       * @param {CanvasRenderingContext2D} ctx context to render on
       * @param {Boolean} [noTransform] When true, context is not transformed
       */
      _render: function(ctx, noTransform) {
        ctx.beginPath();
        ctx.arc(noTransform ? this.left + this.radius : 0,
                noTransform ? this.top + this.radius : 0,
                this.radius,
                this.startAngle,
                this.endAngle, false);
        this._renderFill(ctx);
        this._renderStroke(ctx);
      },

      /**
       * Returns horizontal radius of an object (according to how an object is
scaled)
       * @return {Number}
       */
      getRadiusX: function() {
        return this.get('radius') * this.get('scaleX');
      },

      /**
       * Returns vertical radius of an object (according to how an object is
scaled)
       * @return {Number}
       */
      getRadiusY: function() {
        return this.get('radius') * this.get('scaleY');
      },

      /**
       * Sets radius of an object (and updates width accordingly)
       * @return {fabric.Circle} thisArg
       */
      setRadius: function(value) {
        this.radius = value;
        return this.set('width', value * 2).set('height', value * 2);
      },
    });

    /* _FROM_SVG_START_ */
    /**
     * List of attribute names to account for when parsing SVG element (used by
```

```
{@link fabric.Circle.fromElement})
   * @static
   * @memberOf fabric.Circle
   * @see: http://www.w3.org/TR/SVG/shapes.html#CircleElement
   */
  fabric.Circle.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat('cx cy
r'.split(' '));

  /**
   * Returns {@link fabric.Circle} instance from an SVG element
   * @static
   * @memberOf fabric.Circle
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @throws {Error} If value of `r` attribute is missing or invalid
   * @return {fabric.Circle} Instance of fabric.Circle
   */
  fabric.Circle.fromElement = function(element, options) {
    options || (options = { });

    var parsedAttributes = fabric.parseAttributes(element,
fabric.Circle.ATTRIBUTE_NAMES);

    if (!isValidRadius(parsedAttributes)) {
      throw new Error('value of `r` attribute is required and can not be
negative');
    }

    parsedAttributes.left = parsedAttributes.left || 0;
    parsedAttributes.top = parsedAttributes.top || 0;

    var obj = new fabric.Circle(extend(parsedAttributes, options));

    obj.left -= obj.radius;
    obj.top -= obj.radius;
    return obj;
  };

  /**
   * @private
   */
  function isValidRadius(attributes) {
    return (('radius' in attributes) && (attributes.radius >= 0));
  }
  /* _FROM_SVG_END_ */

  /**
   * Returns {@link fabric.Circle} instance from an object representation
   * @static
   * @memberOf fabric.Circle
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] invoked with new instance as first argument
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {Object} Instance of fabric.Circle
   */
  fabric.Circle.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Circle', object, callback, forceAsync);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {
```

```
  'use strict';

  var fabric = global.fabric || (global.fabric = { });

  if (fabric.Triangle) {
    fabric.warn('fabric.Triangle is already defined');
    return;
  }

  /**
   * Triangle class
   * @class fabric.Triangle
   * @extends fabric.Object
   * @return {fabric.Triangle} thisArg
   * @see {@link fabric.Triangle#initialize} for constructor definition
   */
  fabric.Triangle = fabric.util.createClass(fabric.Object, /** @lends
fabric.Triangle.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'triangle',

    /**
     * Constructor
     * @param {Object} [options] Options object
     * @return {Object} thisArg
     */
    initialize: function(options) {
      this.callSuper('initialize', options);
      this.set('width', options && options.width || 100)
          .set('height', options && options.height || 100);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _render: function(ctx) {
      var widthBy2 = this.width / 2,
          heightBy2 = this.height / 2;

      ctx.beginPath();
      ctx.moveTo(-widthBy2, heightBy2);
      ctx.lineTo(0, -heightBy2);
      ctx.lineTo(widthBy2, heightBy2);
      ctx.closePath();

      this._renderFill(ctx);
      this._renderStroke(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      var widthBy2 = this.width / 2,
          heightBy2 = this.height / 2;
```

```
      ctx.beginPath();
      fabric.util.drawDashedLine(ctx, -widthBy2, heightBy2, 0, -heightBy2,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, 0, -heightBy2, widthBy2, heightBy2,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, widthBy2, heightBy2, -widthBy2, heightBy2,
this.strokeDashArray);
      ctx.closePath();
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(),
          widthBy2 = this.width / 2,
          heightBy2 = this.height / 2,
          points = [
            -widthBy2 + ' ' + heightBy2,
            '0 ' + -heightBy2,
            widthBy2 + ' ' + heightBy2
          ]
          .join(',');

      markup.push(
        '<polygon ', this.getSvgId(),
          'points="', points,
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(),
        '"/>'
      );

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */
  });

  /**
   * Returns {@link fabric.Triangle} instance from an object representation
   * @static
   * @memberOf fabric.Triangle
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] invoked with new instance as first argument
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Triangle}
   */
  fabric.Triangle.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Triangle', object, callback, forceAsync);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      piBy2   = Math.PI * 2,
```

```
      extend = fabric.util.object.extend;

  if (fabric.Ellipse) {
    fabric.warn('fabric.Ellipse is already defined.');
    return;
  }

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push(
    'rx',
    'ry'
  );

  /**
   * Ellipse class
   * @class fabric.Ellipse
   * @extends fabric.Object
   * @return {fabric.Ellipse} thisArg
   * @see {@link fabric.Ellipse#initialize} for constructor definition
   */
  fabric.Ellipse = fabric.util.createClass(fabric.Object, /** @lends
fabric.Ellipse.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'ellipse',

    /**
     * Horizontal radius
     * @type Number
     * @default
     */
    rx:   0,

    /**
     * Vertical radius
     * @type Number
     * @default
     */
    ry:   0,

    cacheProperties: cacheProperties,

    /**
     * Constructor
     * @param {Object} [options] Options object
     * @return {fabric.Ellipse} thisArg
     */
    initialize: function(options) {
      this.callSuper('initialize', options);
      this.set('rx', options && options.rx || 0);
      this.set('ry', options && options.ry || 0);
    },

    /**
     * @private
     * @param {String} key
     * @param {*} value
     * @return {fabric.Ellipse} thisArg
     */
    _set: function(key, value) {
```

```
      this.callSuper('_set', key, value);
      switch (key) {

        case 'rx':
          this.rx = value;
          this.set('width', value * 2);
          break;

        case 'ry':
          this.ry = value;
          this.set('height', value * 2);
          break;

      }
      return this;
    },

    /**
     * Returns horizontal radius of an object (according to how an object is
scaled)
     * @return {Number}
     */
    getRx: function() {
      return this.get('rx') * this.get('scaleX');
    },

    /**
     * Returns Vertical radius of an object (according to how an object is
scaled)
     * @return {Number}
     */
    getRy: function() {
      return this.get('ry') * this.get('scaleY');
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      return this.callSuper('toObject', ['rx',
'ry'].concat(propertiesToInclude));
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(), x = 0, y = 0;
      if (this.group && this.group.type === 'path-group') {
        x = this.left + this.rx;
        y = this.top + this.ry;
      }
      markup.push(
        '<ellipse ', this.getSvgId(),
          'cx="', x, '" cy="', y, '" ',
          'rx="', this.rx,
          '" ry="', this.ry,
```

```
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(),
          this.getSvgTransformMatrix(),
        '"/>\n'
      );

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    _render: function(ctx, noTransform) {
      ctx.beginPath();
      ctx.save();
      ctx.transform(1, 0, 0, this.ry / this.rx, 0, 0);
      ctx.arc(
        noTransform ? this.left + this.rx : 0,
        noTransform ? (this.top + this.ry) * this.rx / this.ry : 0,
        this.rx,
        0,
        piBy2,
        false);
      ctx.restore();
      this._renderFill(ctx);
      this._renderStroke(ctx);
    },
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
{@link fabric.Ellipse.fromElement})
   * @static
   * @memberOf fabric.Ellipse
   * @see http://www.w3.org/TR/SVG/shapes.html#EllipseElement
   */
  fabric.Ellipse.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat('cx cy rx
ry'.split(' '));

  /**
   * Returns {@link fabric.Ellipse} instance from an SVG element
   * @static
   * @memberOf fabric.Ellipse
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Ellipse}
   */
  fabric.Ellipse.fromElement = function(element, options) {
    options || (options = { });

    var parsedAttributes = fabric.parseAttributes(element,
fabric.Ellipse.ATTRIBUTE_NAMES);

    parsedAttributes.left = parsedAttributes.left || 0;
    parsedAttributes.top = parsedAttributes.top || 0;

    var ellipse = new fabric.Ellipse(extend(parsedAttributes, options));

    ellipse.top -= ellipse.ry;
    ellipse.left -= ellipse.rx;
```

```
    return ellipse;
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns {@link fabric.Ellipse} instance from an object representation
   * @static
   * @memberOf fabric.Ellipse
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] invoked with new instance as first argument
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Ellipse}
   */
  fabric.Ellipse.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Ellipse', object, callback, forceAsync);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend;

  if (fabric.Rect) {
    fabric.warn('fabric.Rect is already defined');
    return;
  }

  var stateProperties = fabric.Object.prototype.stateProperties.concat();
  stateProperties.push('rx', 'ry');

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push('rx', 'ry');

  /**
   * Rectangle class
   * @class fabric.Rect
   * @extends fabric.Object
   * @return {fabric.Rect} thisArg
   * @see {@link fabric.Rect#initialize} for constructor definition
   */
  fabric.Rect = fabric.util.createClass(fabric.Object, /** @lends
fabric.Rect.prototype */ {

    /**
     * List of properties to consider when checking if state of an object is
changed ({@link fabric.Object#hasStateChanged})
     * as well as for history (undo/redo) purposes
     * @type Array
     */
    stateProperties: stateProperties,

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'rect',
```

```
    /**
     * Horizontal border radius
     * @type Number
     * @default
     */
    rx:   0,

    /**
     * Vertical border radius
     * @type Number
     * @default
     */
    ry:   0,

    cacheProperties: cacheProperties,

    /**
     * Constructor
     * @param {Object} [options] Options object
     * @return {Object} thisArg
     */
    initialize: function(options) {
      this.callSuper('initialize', options);
      this._initRxRy();
    },

    /**
     * Initializes rx/ry attributes
     * @private
     */
    _initRxRy: function() {
      if (this.rx && !this.ry) {
        this.ry = this.rx;
      }
      else if (this.ry && !this.rx) {
        this.rx = this.ry;
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    _render: function(ctx, noTransform) {

      // optimize 1x1 case (used in spray brush)
      if (this.width === 1 && this.height === 1) {
        ctx.fillRect(-0.5, -0.5, 1, 1);
        return;
      }

      var rx = this.rx ? Math.min(this.rx, this.width / 2) : 0,
          ry = this.ry ? Math.min(this.ry, this.height / 2) : 0,
          w = this.width,
          h = this.height,
          x = noTransform ? this.left : -this.width / 2,
          y = noTransform ? this.top : -this.height / 2,
          isRounded = rx !== 0 || ry !== 0,
          /* "magic number" for bezier approximations of arcs
(http://itc.ktu.lt/itc354/Riskus354.pdf) */
          k = 1 - 0.5522847498;
      ctx.beginPath();
```

```
      ctx.moveTo(x + rx, y);

      ctx.lineTo(x + w - rx, y);
      isRounded && ctx.bezierCurveTo(x + w - k * rx, y, x + w, y + k * ry, x +
w, y + ry);

      ctx.lineTo(x + w, y + h - ry);
      isRounded && ctx.bezierCurveTo(x + w, y + h - k * ry, x + w - k * rx, y +
h, x + w - rx, y + h);

      ctx.lineTo(x + rx, y + h);
      isRounded && ctx.bezierCurveTo(x + k * rx, y + h, x, y + h - k * ry, x, y
+ h - ry);

      ctx.lineTo(x, y + ry);
      isRounded && ctx.bezierCurveTo(x, y + k * ry, x + k * rx, y, x + rx, y);

      ctx.closePath();

      this._renderFill(ctx);
      this._renderStroke(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      var x = -this.width / 2,
          y = -this.height / 2,
          w = this.width,
          h = this.height;

      ctx.beginPath();
      fabric.util.drawDashedLine(ctx, x, y, x + w, y, this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x + w, y, x + w, y + h,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x + w, y + h, x, y + h,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x, y + h, x, y, this.strokeDashArray);
      ctx.closePath();
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      return this.callSuper('toObject', ['rx',
'ry'].concat(propertiesToInclude));
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(), x = this.left, y = this.top;
      if (!(this.group && this.group.type === 'path-group')) {
```

```
        x = -this.width / 2;
        y = -this.height / 2;
      }
      markup.push(
        '<rect ', this.getSvgId(),
          'x="', x, '" y="', y,
          '" rx="', this.get('rx'), '" ry="', this.get('ry'),
          '" width="', this.width, '" height="', this.height,
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(),
          this.getSvgTransformMatrix(),
        '"/>\n');

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
`fabric.Rect.fromElement`)
   * @static
   * @memberOf fabric.Rect
   * @see: http://www.w3.org/TR/SVG/shapes.html#RectElement
   */
  fabric.Rect.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat('x y rx ry width
height'.split(' '));

  /**
   * Returns {@link fabric.Rect} instance from an SVG element
   * @static
   * @memberOf fabric.Rect
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Rect} Instance of fabric.Rect
   */
  fabric.Rect.fromElement = function(element, options) {
    if (!element) {
      return null;
    }
    options = options || { };

    var parsedAttributes = fabric.parseAttributes(element,
fabric.Rect.ATTRIBUTE_NAMES);

    parsedAttributes.left = parsedAttributes.left || 0;
    parsedAttributes.top  = parsedAttributes.top  || 0;
    var rect = new fabric.Rect(extend((options ?
fabric.util.object.clone(options) : { }), parsedAttributes));
    rect.visible = rect.visible && rect.width > 0 && rect.height > 0;
    return rect;
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns {@link fabric.Rect} instance from an object representation
   * @static
   * @memberOf fabric.Rect
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.Rect
instance is created
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
```

```
   * @return {Object} instance of fabric.Rect
   */
  fabric.Rect.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Rect', object, callback, forceAsync);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      min = fabric.util.array.min,
      max = fabric.util.array.max,
      toFixed = fabric.util.toFixed,
      NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS;

  if (fabric.Polyline) {
    fabric.warn('fabric.Polyline is already defined');
    return;
  }

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push('points');

  /**
   * Polyline class
   * @class fabric.Polyline
   * @extends fabric.Object
   * @see {@link fabric.Polyline#initialize} for constructor definition
   */
  fabric.Polyline = fabric.util.createClass(fabric.Object, /** @lends
fabric.Polyline.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'polyline',

    /**
     * Points array
     * @type Array
     * @default
     */
    points: null,

    /**
     * Minimum X from points values, necessary to offset points
     * @type Number
     * @default
     */
    minX: 0,

    /**
     * Minimum Y from points values, necessary to offset points
     * @type Number
     * @default
     */
    minY: 0,
```

```
    cacheProperties: cacheProperties,

    /**
     * Constructor
     * @param {Array} points Array of points (where each point is an object with
x and y)
     * @param {Object} [options] Options object
     * @return {fabric.Polyline} thisArg
     * @example
     * var poly = new fabric.Polyline([
     *     { x: 10, y: 10 },
     *     { x: 50, y: 30 },
     *     { x: 40, y: 70 },
     *     { x: 60, y: 50 },
     *     { x: 100, y: 150 },
     *     { x: 40, y: 100 }
     *   ], {
     *   stroke: 'red',
     *   left: 100,
     *   top: 100
     * });
     */
    initialize: function(points, options) {
      options = options || {};
      this.points = points || [];
      this.callSuper('initialize', options);
      this._calcDimensions();
      if (!('top' in options)) {
        this.top = this.minY;
      }
      if (!('left' in options)) {
        this.left = this.minX;
      }
      this.pathOffset = {
        x: this.minX + this.width / 2,
        y: this.minY + this.height / 2
      };
    },

    /**
     * @private
     */
    _calcDimensions: function() {

      var points = this.points,
          minX = min(points, 'x'),
          minY = min(points, 'y'),
          maxX = max(points, 'x'),
          maxY = max(points, 'y');

      this.width = (maxX - minX) || 0;
      this.height = (maxY - minY) || 0;
      this.minX = minX || 0;
      this.minY = minY || 0;
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} Object representation of an instance
     */
    toObject: function(propertiesToInclude) {
```

```
      return extend(this.callSuper('toObject', propertiesToInclude), {
        points: this.points.concat()
      });
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var points = [],
          diffX = 0,
          diffY = 0,
          markup = this._createBaseSVGMarkup();

      if (!(this.group && this.group.type === 'path-group')) {
        diffX = this.pathOffset.x;
        diffY = this.pathOffset.y;
      }

      for (var i = 0, len = this.points.length; i < len; i++) {
        points.push(
          toFixed(this.points[i].x - diffX, NUM_FRACTION_DIGITS), ',',
          toFixed(this.points[i].y - diffY, NUM_FRACTION_DIGITS), ' '
        );
      }
      markup.push(
        '<', this.type, ' ', this.getSvgId(),
          'points="', points.join(''),
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(),
          ' ', this.getSvgTransformMatrix(),
        '"/>\n'
      );

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */


    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    commonRender: function(ctx, noTransform) {
      var point, len = this.points.length,
          x = noTransform ? 0 : this.pathOffset.x,
          y = noTransform ? 0 : this.pathOffset.y;

      if (!len || isNaN(this.points[len - 1].y)) {
        // do not draw if no points or odd points
        // NaN comes from parseFloat of a empty string in parser
        return false;
      }
      ctx.beginPath();
      ctx.moveTo(this.points[0].x - x, this.points[0].y - y);
      for (var i = 0; i < len; i++) {
        point = this.points[i];
        ctx.lineTo(point.x - x, point.y - y);
      }
```

```
      return true;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    _render: function(ctx, noTransform) {
      if (!this.commonRender(ctx, noTransform)) {
        return;
      }
      this._renderFill(ctx);
      this._renderStroke(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      var p1, p2;

      ctx.beginPath();
      for (var i = 0, len = this.points.length; i < len; i++) {
        p1 = this.points[i];
        p2 = this.points[i + 1] || p1;
        fabric.util.drawDashedLine(ctx, p1.x, p1.y, p2.x, p2.y,
this.strokeDashArray);
      }
    },

    /**
     * Returns complexity of an instance
     * @return {Number} complexity of this instance
     */
    complexity: function() {
      return this.get('points').length;
    }
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
{@link fabric.Polyline.fromElement})
   * @static
   * @memberOf fabric.Polyline
   * @see: http://www.w3.org/TR/SVG/shapes.html#PolylineElement
   */
  fabric.Polyline.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat();

  /**
   * Returns fabric.Polyline instance from an SVG element
   * @static
   * @memberOf fabric.Polyline
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Polyline} Instance of fabric.Polyline
   */
  fabric.Polyline.fromElement = function(element, options) {
    if (!element) {
      return null;
    }
    options || (options = { });
```

```
    var points = fabric.parsePointsAttribute(element.getAttribute('points')),
        parsedAttributes = fabric.parseAttributes(element,
fabric.Polyline.ATTRIBUTE_NAMES);

    return new fabric.Polyline(points,
fabric.util.object.extend(parsedAttributes, options));
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns fabric.Polyline instance from an object representation
   * @static
   * @memberOf fabric.Polyline
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.Path
instance is created
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Polyline} Instance of fabric.Polyline
   */
  fabric.Polyline.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Polyline', object, callback, forceAsync,
'points');
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend;

  if (fabric.Polygon) {
    fabric.warn('fabric.Polygon is already defined');
    return;
  }

  /**
   * Polygon class
   * @class fabric.Polygon
   * @extends fabric.Polyline
   * @see {@link fabric.Polygon#initialize} for constructor definition
   */
  fabric.Polygon = fabric.util.createClass(fabric.Polyline, /** @lends
fabric.Polygon.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'polygon',

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    _render: function(ctx, noTransform) {
      if (!this.commonRender(ctx, noTransform)) {
```

```
        return;
      }
      ctx.closePath();
      this._renderFill(ctx);
      this._renderStroke(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      this.callSuper('_renderDashedStroke', ctx);
      ctx.closePath();
    },
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
`fabric.Polygon.fromElement`)
   * @static
   * @memberOf fabric.Polygon
   * @see: http://www.w3.org/TR/SVG/shapes.html#PolygonElement
   */
  fabric.Polygon.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat();

  /**
   * Returns {@link fabric.Polygon} instance from an SVG element
   * @static
   * @memberOf fabric.Polygon
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Polygon} Instance of fabric.Polygon
   */
  fabric.Polygon.fromElement = function(element, options) {
    if (!element) {
      return null;
    }

    options || (options = { });

    var points = fabric.parsePointsAttribute(element.getAttribute('points')),
        parsedAttributes = fabric.parseAttributes(element,
fabric.Polygon.ATTRIBUTE_NAMES);

    return new fabric.Polygon(points, extend(parsedAttributes, options));
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns fabric.Polygon instance from an object representation
   * @static
   * @memberOf fabric.Polygon
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.Path
instance is created
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Polygon} Instance of fabric.Polygon
   */
  fabric.Polygon.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Polygon', object, callback, forceAsync,
'points');
```

```
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      min = fabric.util.array.min,
      max = fabric.util.array.max,
      extend = fabric.util.object.extend,
      _toString = Object.prototype.toString,
      drawArc = fabric.util.drawArc,
      commandLengths = {
        m: 2,
        l: 2,
        h: 1,
        v: 1,
        c: 6,
        s: 4,
        q: 4,
        t: 2,
        a: 7
      },
      repeatedCommands = {
        m: 'l',
        M: 'L'
      };

  if (fabric.Path) {
    fabric.warn('fabric.Path is already defined');
    return;
  }

  var stateProperties = fabric.Object.prototype.stateProperties.concat();
  stateProperties.push('path');

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push('path', 'fillRule');

  /**
   * Path class
   * @class fabric.Path
   * @extends fabric.Object
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-
1#path_and_pathgroup}
   * @see {@link fabric.Path#initialize} for constructor definition
   */
  fabric.Path = fabric.util.createClass(fabric.Object, /** @lends
fabric.Path.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'path',

    /**
     * Array of path points
     * @type Array
     * @default
```

```
     */
    path: null,

    /**
     * Minimum X from points values, necessary to offset points
     * @type Number
     * @default
     */
    minX: 0,

    /**
     * Minimum Y from points values, necessary to offset points
     * @type Number
     * @default
     */
    minY: 0,

    cacheProperties: cacheProperties,

    stateProperties: stateProperties,

    /**
     * Constructor
     * @param {Array|String} path Path data (sequence of coordinates and
corresponding "command" tokens)
     * @param {Object} [options] Options object
     * @return {fabric.Path} thisArg
     */
    initialize: function(path, options) {
      options = options || { };
      this.callSuper('initialize', options);

      if (!path) {
        path = [];
      }

      var fromArray = _toString.call(path) === '[object Array]';

      this.path = fromArray
        ? path
        // one of commands (m,M,l,L,q,Q,c,C,etc.) followed by non-command
characters (i.e. command values)
        : path.match && path.match(/[mzlhvcsqta][^mzlhvcsqta]*/gi);

      if (!this.path) {
        return;
      }

      if (!fromArray) {
        this.path = this._parsePath();
      }

      this._setPositionDimensions(options);
    },

    /**
     * @private
     * @param {Object} options Options object
     */
    _setPositionDimensions: function(options) {
      var calcDim = this._parseDimensions();

      this.minX = calcDim.left;
      this.minY = calcDim.top;
```

```
  this.width = calcDim.width;
  this.height = calcDim.height;

  if (typeof options.left === 'undefined') {
    this.left = calcDim.left + (this.originX === 'center'
      ? this.width / 2
      : this.originX === 'right'
        ? this.width
        : 0);
  }

  if (typeof options.top === 'undefined') {
    this.top = calcDim.top + (this.originY === 'center'
      ? this.height / 2
      : this.originY === 'bottom'
        ? this.height
        : 0);
  }

  this.pathOffset = this.pathOffset || {
    x: this.minX + this.width / 2,
    y: this.minY + this.height / 2
  };
},

/**
 * @private
 * @param {CanvasRenderingContext2D} ctx context to render path on
 */
_renderPathCommands: function(ctx) {
  var current, // current instruction
      previous = null,
      subpathStartX = 0,
      subpathStartY = 0,
      x = 0, // current x
      y = 0, // current y
      controlX = 0, // current control point x
      controlY = 0, // current control point y
      tempX,
      tempY,
      l = -this.pathOffset.x,
      t = -this.pathOffset.y;

  if (this.group && this.group.type === 'path-group') {
    l = 0;
    t = 0;
  }

  ctx.beginPath();

  for (var i = 0, len = this.path.length; i < len; ++i) {

    current = this.path[i];

    switch (current[0]) { // first letter

      case 'l': // lineto, relative
        x += current[1];
        y += current[2];
        ctx.lineTo(x + l, y + t);
        break;

      case 'L': // lineto, absolute
        x = current[1];
```

```
      y = current[2];
      ctx.lineTo(x + l, y + t);
      break;

    case 'h': // horizontal lineto, relative
      x += current[1];
      ctx.lineTo(x + l, y + t);
      break;

    case 'H': // horizontal lineto, absolute
      x = current[1];
      ctx.lineTo(x + l, y + t);
      break;

    case 'v': // vertical lineto, relative
      y += current[1];
      ctx.lineTo(x + l, y + t);
      break;

    case 'V': // verical lineto, absolute
      y = current[1];
      ctx.lineTo(x + l, y + t);
      break;

    case 'm': // moveTo, relative
      x += current[1];
      y += current[2];
      subpathStartX = x;
      subpathStartY = y;
      ctx.moveTo(x + l, y + t);
      break;

    case 'M': // moveTo, absolute
      x = current[1];
      y = current[2];
      subpathStartX = x;
      subpathStartY = y;
      ctx.moveTo(x + l, y + t);
      break;

    case 'c': // bezierCurveTo, relative
      tempX = x + current[5];
      tempY = y + current[6];
      controlX = x + current[3];
      controlY = y + current[4];
      ctx.bezierCurveTo(
        x + current[1] + l, // x1
        y + current[2] + t, // y1
        controlX + l, // x2
        controlY + t, // y2
        tempX + l,
        tempY + t
      );
      x = tempX;
      y = tempY;
      break;

    case 'C': // bezierCurveTo, absolute
      x = current[5];
      y = current[6];
      controlX = current[3];
      controlY = current[4];
      ctx.bezierCurveTo(
        current[1] + l,
```

```
              current[2] + t,
              controlX + l,
              controlY + t,
              x + l,
              y + t
            );
            break;

          case 's': // shorthand cubic bezierCurveTo, relative

            // transform to absolute x,y
            tempX = x + current[3];
            tempY = y + current[4];

            if (previous[0].match(/[CcSs]/) === null) {
              // If there is no previous command or if the previous command was
not a C, c, S, or s,
              // the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control points
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }

            ctx.bezierCurveTo(
              controlX + l,
              controlY + t,
              x + current[1] + l,
              y + current[2] + t,
              tempX + l,
              tempY + t
            );
            // set control point to 2nd one of this command
            // "... the first control point is assumed to be
            // the reflection of the second control point on
            // the previous command relative to the current point."
            controlX = x + current[1];
            controlY = y + current[2];

            x = tempX;
            y = tempY;
            break;

          case 'S': // shorthand cubic bezierCurveTo, absolute
            tempX = current[3];
            tempY = current[4];
            if (previous[0].match(/[CcSs]/) === null) {
              // If there is no previous command or if the previous command was
not a C, c, S, or s,
              // the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control points
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }
            ctx.bezierCurveTo(
              controlX + l,
              controlY + t,
```

```
              current[1] + l,
              current[2] + t,
              tempX + l,
              tempY + t
            );
            x = tempX;
            y = tempY;

            // set control point to 2nd one of this command
            // "... the first control point is assumed to be
            // the reflection of the second control point on
            // the previous command relative to the current point."
            controlX = current[1];
            controlY = current[2];

            break;

          case 'q': // quadraticCurveTo, relative
            // transform to absolute x,y
            tempX = x + current[3];
            tempY = y + current[4];

            controlX = x + current[1];
            controlY = y + current[2];

            ctx.quadraticCurveTo(
              controlX + l,
              controlY + t,
              tempX + l,
              tempY + t
            );
            x = tempX;
            y = tempY;
            break;

          case 'Q': // quadraticCurveTo, absolute
            tempX = current[3];
            tempY = current[4];

            ctx.quadraticCurveTo(
              current[1] + l,
              current[2] + t,
              tempX + l,
              tempY + t
            );
            x = tempX;
            y = tempY;
            controlX = current[1];
            controlY = current[2];
            break;

          case 't': // shorthand quadraticCurveTo, relative

            // transform to absolute x,y
            tempX = x + current[1];
            tempY = y + current[2];

            if (previous[0].match(/[QqTt]/) === null) {
              // If there is no previous command or if the previous command was
not a Q, q, T or t,
              // assume the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
```

```
            else {
              // calculate reflection of previous control point
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }

            ctx.quadraticCurveTo(
              controlX + l,
              controlY + t,
              tempX + l,
              tempY + t
            );
            x = tempX;
            y = tempY;

            break;

          case 'T':
            tempX = current[1];
            tempY = current[2];

            if (previous[0].match(/[QqTt]/) === null) {
              // If there is no previous command or if the previous command was
not a Q, q, T or t,
              // assume the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control point
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }
            ctx.quadraticCurveTo(
              controlX + l,
              controlY + t,
              tempX + l,
              tempY + t
            );
            x = tempX;
            y = tempY;
            break;

          case 'a':
            // TODO: optimize this
            drawArc(ctx, x + l, y + t, [
              current[1],
              current[2],
              current[3],
              current[4],
              current[5],
              current[6] + x + l,
              current[7] + y + t
            ]);
            x += current[6];
            y += current[7];
            break;

          case 'A':
            // TODO: optimize this
            drawArc(ctx, x + l, y + t, [
              current[1],
              current[2],
              current[3],
```

```
              current[4],
              current[5],
              current[6] + l,
              current[7] + t
            ]);
            x = current[6];
            y = current[7];
            break;

          case 'z':
          case 'Z':
            x = subpathStartX;
            y = subpathStartY;
            ctx.closePath();
            break;
        }
        previous = current;
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx context to render path on
     */
    _render: function(ctx) {
      this._renderPathCommands(ctx);
      this._renderFill(ctx);
      this._renderStroke(ctx);
    },

    /**
     * Returns string representation of an instance
     * @return {String} string representation of an instance
     */
    toString: function() {
      return '#<fabric.Path (' + this.complexity() +
        '): { "top": ' + this.top + ', "left": ' + this.left + ' }>';
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      var o = extend(this.callSuper('toObject', ['sourcePath',
'pathOffset'].concat(propertiesToInclude)), {
        path: this.path.map(function(item) { return item.slice(); }),
        top: this.top,
        left: this.left,
      });
      return o;
    },

    /**
     * Returns dataless object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toDatalessObject: function(propertiesToInclude) {
      var o = this.toObject(propertiesToInclude);
      if (this.sourcePath) {
```

```
        o.path = this.sourcePath;
      }
      delete o.sourcePath;
      return o;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var chunks = [],
          markup = this._createBaseSVGMarkup(), addTransform = '';

      for (var i = 0, len = this.path.length; i < len; i++) {
        chunks.push(this.path[i].join(' '));
      }
      var path = chunks.join(' ');
      if (!(this.group && this.group.type === 'path-group')) {
        addTransform = ' translate(' + (-this.pathOffset.x) + ', ' + (-
this.pathOffset.y) + ') ';
      }
      markup.push(
        '<path ', this.getSvgId(),
          'd="', path,
          '" style="', this.getSvgStyles(),
          '" transform="', this.getSvgTransform(), addTransform,
          this.getSvgTransformMatrix(), '" stroke-linecap="round" ',
        '/>\n'
      );

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns number representation of an instance complexity
     * @return {Number} complexity of this instance
     */
    complexity: function() {
      return this.path.length;
    },

    /**
     * @private
     */
    _parsePath: function() {
      var result = [],
          coords = [],
          currentPath,
          parsed,
          re = /([-+]?((\d+\.\d+)|((\d+)|(\.\d+)))(?:e[-+]?\d+)?)/ig,
          match,
          coordsStr;

      for (var i = 0, coordsParsed, len = this.path.length; i < len; i++) {
        currentPath = this.path[i];

        coordsStr = currentPath.slice(1).trim();
        coords.length = 0;
```

```
        while ((match = re.exec(coordsStr))) {
          coords.push(match[0]);
        }

        coordsParsed = [currentPath.charAt(0)];

        for (var j = 0, jlen = coords.length; j < jlen; j++) {
          parsed = parseFloat(coords[j]);
          if (!isNaN(parsed)) {
            coordsParsed.push(parsed);
          }
        }

        var command = coordsParsed[0],
            commandLength = commandLengths[command.toLowerCase()],
            repeatedCommand = repeatedCommands[command] || command;

        if (coordsParsed.length - 1 > commandLength) {
          for (var k = 1, klen = coordsParsed.length; k < klen; k +=
commandLength) {
            result.push([command].concat(coordsParsed.slice(k, k +
commandLength)));
            command = repeatedCommand;
          }
        }
        else {
          result.push(coordsParsed);
        }
      }

      return result;
    },

    /**
     * @private
     */
    _parseDimensions: function() {

      var aX = [],
          aY = [],
          current, // current instruction
          previous = null,
          subpathStartX = 0,
          subpathStartY = 0,
          x = 0, // current x
          y = 0, // current y
          controlX = 0, // current control point x
          controlY = 0, // current control point y
          tempX,
          tempY,
          bounds;

      for (var i = 0, len = this.path.length; i < len; ++i) {

        current = this.path[i];

        switch (current[0]) { // first letter

          case 'l': // lineto, relative
            x += current[1];
            y += current[2];
            bounds = [];
            break;
```

```
case 'L': // lineto, absolute
  x = current[1];
  y = current[2];
  bounds = [];
  break;

case 'h': // horizontal lineto, relative
  x += current[1];
  bounds = [];
  break;

case 'H': // horizontal lineto, absolute
  x = current[1];
  bounds = [];
  break;

case 'v': // vertical lineto, relative
  y += current[1];
  bounds = [];
  break;

case 'V': // verical lineto, absolute
  y = current[1];
  bounds = [];
  break;

case 'm': // moveTo, relative
  x += current[1];
  y += current[2];
  subpathStartX = x;
  subpathStartY = y;
  bounds = [];
  break;

case 'M': // moveTo, absolute
  x = current[1];
  y = current[2];
  subpathStartX = x;
  subpathStartY = y;
  bounds = [];
  break;

case 'c': // bezierCurveTo, relative
  tempX = x + current[5];
  tempY = y + current[6];
  controlX = x + current[3];
  controlY = y + current[4];
  bounds = fabric.util.getBoundsOfCurve(x, y,
    x + current[1], // x1
    y + current[2], // y1
    controlX, // x2
    controlY, // y2
    tempX,
    tempY
  );
  x = tempX;
  y = tempY;
  break;

case 'C': // bezierCurveTo, absolute
  controlX = current[3];
  controlY = current[4];
  bounds = fabric.util.getBoundsOfCurve(x, y,
    current[1],
```

```
              current[2],
              controlX,
              controlY,
              current[5],
              current[6]
            );
            x = current[5];
            y = current[6];
            break;

          case 's': // shorthand cubic bezierCurveTo, relative

            // transform to absolute x,y
            tempX = x + current[3];
            tempY = y + current[4];

            if (previous[0].match(/[CcSs]/) === null) {
              // If there is no previous command or if the previous command was
not a C, c, S, or s,
              // the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control points
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }

            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
              controlY,
              x + current[1],
              y + current[2],
              tempX,
              tempY
            );
            // set control point to 2nd one of this command
            // "... the first control point is assumed to be
            // the reflection of the second control point on
            // the previous command relative to the current point."
            controlX = x + current[1];
            controlY = y + current[2];
            x = tempX;
            y = tempY;
            break;

          case 'S': // shorthand cubic bezierCurveTo, absolute
            tempX = current[3];
            tempY = current[4];
            if (previous[0].match(/[CcSs]/) === null) {
              // If there is no previous command or if the previous command was
not a C, c, S, or s,
              // the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control points
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }
            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
```

```
              controlY,
              current[1],
              current[2],
              tempX,
              tempY
            );
            x = tempX;
            y = tempY;
            // set control point to 2nd one of this command
            // "... the first control point is assumed to be
            // the reflection of the second control point on
            // the previous command relative to the current point."
            controlX = current[1];
            controlY = current[2];
            break;

          case 'q': // quadraticCurveTo, relative
            // transform to absolute x,y
            tempX = x + current[3];
            tempY = y + current[4];
            controlX = x + current[1];
            controlY = y + current[2];
            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
              controlY,
              controlX,
              controlY,
              tempX,
              tempY
            );
            x = tempX;
            y = tempY;
            break;

          case 'Q': // quadraticCurveTo, absolute
            controlX = current[1];
            controlY = current[2];
            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
              controlY,
              controlX,
              controlY,
              current[3],
              current[4]
            );
            x = current[3];
            y = current[4];
            break;

          case 't': // shorthand quadraticCurveTo, relative
            // transform to absolute x,y
            tempX = x + current[1];
            tempY = y + current[2];
            if (previous[0].match(/[QqTt]/) === null) {
              // If there is no previous command or if the previous command was
not a Q, q, T or t,
              // assume the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control point
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
```

```
            }

            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
              controlY,
              controlX,
              controlY,
              tempX,
              tempY
            );
            x = tempX;
            y = tempY;

            break;

          case 'T':
            tempX = current[1];
            tempY = current[2];

            if (previous[0].match(/[QqTt]/) === null) {
              // If there is no previous command or if the previous command was
not a Q, q, T or t,
              // assume the control point is coincident with the current point
              controlX = x;
              controlY = y;
            }
            else {
              // calculate reflection of previous control point
              controlX = 2 * x - controlX;
              controlY = 2 * y - controlY;
            }
            bounds = fabric.util.getBoundsOfCurve(x, y,
              controlX,
              controlY,
              controlX,
              controlY,
              tempX,
              tempY
            );
            x = tempX;
            y = tempY;
            break;

          case 'a':
            // TODO: optimize this
            bounds = fabric.util.getBoundsOfArc(x, y,
              current[1],
              current[2],
              current[3],
              current[4],
              current[5],
              current[6] + x,
              current[7] + y
            );
            x += current[6];
            y += current[7];
            break;

          case 'A':
            // TODO: optimize this
            bounds = fabric.util.getBoundsOfArc(x, y,
              current[1],
              current[2],
              current[3],
```

```
              current[4],
              current[5],
              current[6],
              current[7]
            );
            x = current[6];
            y = current[7];
            break;

          case 'z':
          case 'Z':
            x = subpathStartX;
            y = subpathStartY;
            break;
        }
        previous = current;
        bounds.forEach(function (point) {
          aX.push(point.x);
          aY.push(point.y);
        });
        aX.push(x);
        aY.push(y);
      }

      var minX = min(aX) || 0,
          minY = min(aY) || 0,
          maxX = max(aX) || 0,
          maxY = max(aY) || 0,
          deltaX = maxX - minX,
          deltaY = maxY - minY,

          o = {
            left: minX,
            top: minY,
            width: deltaX,
            height: deltaY
          };

      return o;
    }
  });

  /**
   * Creates an instance of fabric.Path from an object
   * @static
   * @memberOf fabric.Path
   * @param {Object} object
   * @param {Function} [callback] Callback to invoke when an fabric.Path
instance is created
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   */
  fabric.Path.fromObject = function(object, callback, forceAsync) {
    // remove this pattern rom 2.0, accept just object.
    var path;
    if (typeof object.path === 'string') {
      fabric.loadSVGFromURL(object.path, function (elements) {
        var pathUrl = object.path;
        path = elements[0];
        delete object.path;

        path.setOptions(object);
        path.setSourcePath(pathUrl);
```

```
        callback && callback(path);
      });
    }
    else {
      return fabric.Object._fromObject('Path', object, callback, forceAsync,
'path');
    }
  };

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
`fabric.Path.fromElement`)
   * @static
   * @memberOf fabric.Path
   * @see http://www.w3.org/TR/SVG/paths.html#PathElement
   */
  fabric.Path.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat(['d']);

  /**
   * Creates an instance of fabric.Path from an SVG <path> element
   * @static
   * @memberOf fabric.Path
   * @param {SVGElement} element to parse
   * @param {Function} callback Callback to invoke when an fabric.Path instance
is created
   * @param {Object} [options] Options object
   */
  fabric.Path.fromElement = function(element, callback, options) {
    var parsedAttributes = fabric.parseAttributes(element,
fabric.Path.ATTRIBUTE_NAMES);
    callback && callback(new fabric.Path(parsedAttributes.d,
extend(parsedAttributes, options)));
  };
  /* _FROM_SVG_END_ */

  /**
   * Indicates that instances of this type are async
   * @static
   * @memberOf fabric.Path
   * @type Boolean
   * @default
   */
  fabric.Path.async = true;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend;

  if (fabric.PathGroup) {
    fabric.warn('fabric.PathGroup is already defined');
    return;
  }

  /**
   * Path group class
   * @class fabric.PathGroup
   * @extends fabric.Path
```

```
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-
1#path_and_pathgroup}
   * @see {@link fabric.PathGroup#initialize} for constructor definition
   */
  fabric.PathGroup = fabric.util.createClass(fabric.Object, /** @lends
fabric.PathGroup.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'path-group',

    /**
     * Fill value
     * @type String
     * @default
     */
    fill: '',

    /**
     * Pathgroups are container, do not render anything on theyr own, ence no
cache properties
     * @type Boolean
     * @default
     */
    cacheProperties: [],

    /**
     * Constructor
     * @param {Array} paths
     * @param {Object} [options] Options object
     * @return {fabric.PathGroup} thisArg
     */
    initialize: function(paths, options) {

      options = options || { };
      this.paths = paths || [];

      for (var i = this.paths.length; i--;) {
        this.paths[i].group = this;
      }

      if (options.toBeParsed) {
        this.parseDimensionsFromPaths(options);
        delete options.toBeParsed;
      }
      this.setOptions(options);
      this.setCoords();
    },

    /**
     * Calculate width and height based on paths contained
     */
    parseDimensionsFromPaths: function(options) {
      var points, p, xC = [], yC = [], path, height, width,
          m;
      for (var j = this.paths.length; j--;) {
        path = this.paths[j];
        height = path.height + path.strokeWidth;
        width = path.width + path.strokeWidth;
        points = [
          { x: path.left, y: path.top },
```

```
          { x: path.left + width, y: path.top },
          { x: path.left, y: path.top + height },
          { x: path.left + width, y: path.top + height }
        ];
        m = this.paths[j].transformMatrix;
        for (var i = 0; i < points.length; i++) {
          p = points[i];
          if (m) {
            p = fabric.util.transformPoint(p, m, false);
          }
          xC.push(p.x);
          yC.push(p.y);
        }
      }
      options.width = Math.max.apply(null, xC);
      options.height = Math.max.apply(null, yC);
    },

    /**
     * Execute the drawing operation for an object on a specified context
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    drawObject: function(ctx) {
      ctx.save();
      ctx.translate(-this.width / 2, -this.height / 2);
      for (var i = 0, l = this.paths.length; i < l; ++i) {
        this.paths[i].render(ctx, true);
      }
      ctx.restore();
    },

    /**
     * Decide if the object should cache or not.
     * objectCaching is a global flag, wins over everything
     * needsItsOwnCache should be used when the object drawing method requires
     * a cache step. None of the fabric classes requires it.
     * Generally you do not cache objects in groups because the group outside is
cached.
     * @return {Boolean}
     */
    shouldCache: function() {
      var parentCache = this.objectCaching && (!this.group ||
this.needsItsOwnCache() || !this.group.isCaching());
      this.caching = parentCache;
      if (parentCache) {
        for (var i = 0, len = this.paths.length; i < len; i++) {
          if (this.paths[i].willDrawShadow()) {
            this.caching = false;
            return false;
          }
        }
      }
      return parentCache;
    },

    /**
     * Check if this object or a child object will cast a shadow
     * @return {Boolean}
     */
    willDrawShadow: function() {
      if (this.shadow) {
        return true;
      }
```

```
      for (var i = 0, len = this.paths.length; i < len; i++) {
        if (this.paths[i].willDrawShadow()) {
          return true;
        }
      }
      return false;
    },

    /**
     * Check if this group or its parent group are caching, recursively up
     * @return {Boolean}
     */
    isCaching: function() {
      return this.caching || this.group && this.group.isCaching();
    },

    /**
     * Check if cache is dirty
     */
    isCacheDirty: function() {
      if (this.callSuper('isCacheDirty')) {
        return true;
      }
      if (!this.statefullCache) {
        return false;
      }
      for (var i = 0, len = this.paths.length; i < len; i++) {
        if (this.paths[i].isCacheDirty(true)) {
          if (this._cacheCanvas) {
            var x = this.cacheWidth / this.zoomX, y = this.cacheHeight /
this.zoomY;
            this._cacheContext.clearRect(-x / 2, -y / 2, x, y);
          }
          return true;
        }
      }
      return false;
    },

    /**
     * Sets certain property to a certain value
     * @param {String} prop
     * @param {*} value
     * @return {fabric.PathGroup} thisArg
     */
    _set: function(prop, value) {

      if (prop === 'fill' && value && this.isSameColor()) {
        var i = this.paths.length;
        while (i--) {
          this.paths[i]._set(prop, value);
        }
      }

      return this.callSuper('_set', prop, value);
    },

    /**
     * Returns object representation of this path group
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
```

```
      var pathsToObject = this.paths.map(function(path) {
        var originalDefaults = path.includeDefaultValues;
        path.includeDefaultValues = path.group.includeDefaultValues;
        var obj = path.toObject(propertiesToInclude);
        path.includeDefaultValues = originalDefaults;
        return obj;
      });
      var o = extend(this.callSuper('toObject',
['sourcePath'].concat(propertiesToInclude)), {
        paths: pathsToObject
      });
      return o;
    },

    /**
     * Returns dataless object representation of this path group
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} dataless object representation of an instance
     */
    toDatalessObject: function(propertiesToInclude) {
      var o = this.toObject(propertiesToInclude);
      if (this.sourcePath) {
        o.paths = this.sourcePath;
      }
      return o;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var objects = this.getObjects(),
          p = this.getPointByOrigin('left', 'top'),
          translatePart = 'translate(' + p.x + ' ' + p.y + ')',
          markup = this._createBaseSVGMarkup();
      markup.push(
        '<g ', this.getSvgId(),
        'style="', this.getSvgStyles(), '" ',
        'transform="', this.getSvgTransformMatrix(), translatePart,
this.getSvgTransform(), '" ',
        '>\n'
      );

      for (var i = 0, len = objects.length; i < len; i++) {
        markup.push('\t', objects[i].toSVG(reviver));
      }
      markup.push('</g>\n');

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns a string representation of this path group
     * @return {String} string representation of an object
     */
    toString: function() {
      return '#<fabric.PathGroup (' + this.complexity() +
        '): { top: ' + this.top + ', left: ' + this.left + ' }>';
```

```
    },

    /**
     * Returns true if all paths in this group are of same color
     * @return {Boolean} true if all paths are of the same color (`fill`)
     */
    isSameColor: function() {
      var firstPathFill = this.getObjects()[0].get('fill') || '';
      if (typeof firstPathFill !== 'string') {
        return false;
      }
      firstPathFill = firstPathFill.toLowerCase();
      return this.getObjects().every(function(path) {
        var pathFill = path.get('fill') || '';
        return typeof pathFill === 'string' && (pathFill).toLowerCase() ===
firstPathFill;
      });
    },

    /**
     * Returns number representation of object's complexity
     * @return {Number} complexity
     */
    complexity: function() {
      return this.paths.reduce(function(total, path) {
        return total + ((path && path.complexity) ? path.complexity() : 0);
      }, 0);
    },

    /**
     * Returns all paths in this path group
     * @return {Array} array of path objects included in this path group
     */
    getObjects: function() {
      return this.paths;
    }
  });

  /**
   * Creates fabric.PathGroup instance from an object representation
   * @static
   * @memberOf fabric.PathGroup
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.PathGroup
instance is created
   */
  fabric.PathGroup.fromObject = function(object, callback) {
    var originalPaths = object.paths;
    delete object.paths;
    if (typeof originalPaths === 'string') {
      fabric.loadSVGFromURL(originalPaths, function (elements) {
        var pathUrl = originalPaths;
        var pathGroup = fabric.util.groupSVGElements(elements, object, pathUrl);
        object.paths = originalPaths;
        callback(pathGroup);
      });
    }
    else {
      fabric.util.enlivenObjects(originalPaths, function(enlivenedObjects) {
        var pathGroup = new fabric.PathGroup(enlivenedObjects, object);
        object.paths = originalPaths;
        callback(pathGroup);
      });
    }
```

```
  };

  /**
   * Indicates that instances of this type are async
   * @static
   * @memberOf fabric.PathGroup
   * @type Boolean
   * @default
   */
  fabric.PathGroup.async = true;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      min = fabric.util.array.min,
      max = fabric.util.array.max;

  if (fabric.Group) {
    return;
  }

  // lock-related properties, for use in fabric.Group#get
  // to enable locking behavior on group
  // when one of its objects has lock-related properties set
  var _lockProperties = {
    lockMovementX:  true,
    lockMovementY:  true,
    lockRotation:   true,
    lockScalingX:   true,
    lockScalingY:   true,
    lockUniScaling: true
  };

  /**
   * Group class
   * @class fabric.Group
   * @extends fabric.Object
   * @mixes fabric.Collection
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-3#groups}
   * @see {@link fabric.Group#initialize} for constructor definition
   */
  fabric.Group = fabric.util.createClass(fabric.Object, fabric.Collection, /**
@lends fabric.Group.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'group',

    /**
     * Width of stroke
     * @type Number
     * @default
     */
    strokeWidth: 0,
```

```
    /**
     * Indicates if click events should also check for subtargets
     * @type Boolean
     * @default
     */
    subTargetCheck: false,

    /**
     * Groups are container, do not render anything on theyr own, ence no cache
properties
     * @type Boolean
     * @default
     */
    cacheProperties: [],

    /**
     * Constructor
     * @param {Object} objects Group objects
     * @param {Object} [options] Options object
     * @param {Boolean} [isAlreadyGrouped] if true, objects have been grouped
already.
     * @return {Object} thisArg
     */
    initialize: function(objects, options, isAlreadyGrouped) {
      options = options || { };

      this._objects = [];
      // if objects enclosed in a group have been grouped already,
      // we cannot change properties of objects.
      // Thus we need to set options to group without objects,
      // because delegatedProperties propagate to objects.
      isAlreadyGrouped && this.callSuper('initialize', options);

      this._objects = objects || [];
      for (var i = this._objects.length; i--; ) {
        this._objects[i].group = this;
      }

      if (options.originX) {
        this.originX = options.originX;
      }
      if (options.originY) {
        this.originY = options.originY;
      }

      if (isAlreadyGrouped) {
        // do not change coordinate of objects enclosed in a group,
        // because objects coordinate system have been group coodinate system
already.
        this._updateObjectsCoords(true);
      }
      else {
        this._calcBounds();
        this._updateObjectsCoords();
        this.callSuper('initialize', options);
      }

      this.setCoords();
      this.saveCoords();
    },

    /**
     * @private
     * @param {Boolean} [skipCoordsChange] if true, coordinates of objects
```

```
enclosed in a group do not change
     */
    _updateObjectsCoords: function(skipCoordsChange) {
      var center = this.getCenterPoint();
      for (var i = this._objects.length; i--; ){
        this._updateObjectCoords(this._objects[i], center, skipCoordsChange);
      }
    },

    /**
     * @private
     * @param {Object} object
     * @param {fabric.Point} center, current center of group.
     * @param {Boolean} [skipCoordsChange] if true, coordinates of object dose
not change
     */
    _updateObjectCoords: function(object, center, skipCoordsChange) {
      // do not display corners of objects enclosed in a group
      object.__origHasControls = object.hasControls;
      object.hasControls = false;

      if (skipCoordsChange) {
        return;
      }

      var objectLeft = object.getLeft(),
          objectTop = object.getTop(),
          ignoreZoom = true, skipAbsolute = true;

      object.set({
        left: objectLeft - center.x,
        top: objectTop - center.y
      });
      object.setCoords(ignoreZoom, skipAbsolute);
    },

    /**
     * Returns string represenation of a group
     * @return {String}
     */
    toString: function() {
      return '#<fabric.Group: (' + this.complexity() + ')>';
    },

    /**
     * Adds an object to a group; Then recalculates group's dimension, position.
     * @param {Object} object
     * @return {fabric.Group} thisArg
     * @chainable
     */
    addWithUpdate: function(object) {
      this._restoreObjectsState();
      fabric.util.resetObjectTransform(this);
      if (object) {
        this._objects.push(object);
        object.group = this;
        object._set('canvas', this.canvas);
      }
      // since _restoreObjectsState set objects inactive
      this.forEachObject(this._setObjectActive, this);
      this._calcBounds();
      this._updateObjectsCoords();
      this.setCoords();
      this.dirty = true;
```

```
      return this;
    },

    /**
     * @private
     */
    _setObjectActive: function(object) {
      object.set('active', true);
      object.group = this;
    },

    /**
     * Removes an object from a group; Then recalculates group's dimension,
position.
     * @param {Object} object
     * @return {fabric.Group} thisArg
     * @chainable
     */
    removeWithUpdate: function(object) {
      this._restoreObjectsState();
      fabric.util.resetObjectTransform(this);
      // since _restoreObjectsState set objects inactive
      this.forEachObject(this._setObjectActive, this);

      this.remove(object);
      this._calcBounds();
      this._updateObjectsCoords();
      this.setCoords();
      this.dirty = true;
      return this;
    },

    /**
     * @private
     */
    _onObjectAdded: function(object) {
      this.dirty = true;
      object.group = this;
      object._set('canvas', this.canvas);
    },

    /**
     * @private
     */
    _onObjectRemoved: function(object) {
      this.dirty = true;
      delete object.group;
      object.set('active', false);
    },

    /**
     * Properties that are delegated to group objects when reading/writing
     * @param {Object} delegatedProperties
     */
    delegatedProperties: {
      fill:             true,
      stroke:           true,
      strokeWidth:      true,
      fontFamily:       true,
      fontWeight:       true,
      fontSize:         true,
      fontStyle:        true,
      lineHeight:       true,
      textDecoration:   true,
```

```
      textAlign:        true,
      backgroundColor:  true
    },

    /**
     * @private
     */
    _set: function(key, value) {
      var i = this._objects.length;

      if (this.delegatedProperties[key] || key === 'canvas') {
        while (i--) {
          this._objects[i].set(key, value);
        }
      }
      else {
        while (i--) {
          this._objects[i].setOnGroup(key, value);
        }
      }

      this.callSuper('_set', key, value);
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      var objsToObject = this.getObjects().map(function(obj) {
        var originalDefaults = obj.includeDefaultValues;
        obj.includeDefaultValues = obj.group.includeDefaultValues;
        var _obj = obj.toObject(propertiesToInclude);
        obj.includeDefaultValues = originalDefaults;
        return _obj;
      });
      return extend(this.callSuper('toObject', propertiesToInclude), {
        objects: objsToObject
      });
    },

    /**
     * Returns object representation of an instance, in dataless mode.
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toDatalessObject: function(propertiesToInclude) {
      var objsToObject = this.getObjects().map(function(obj) {
        var originalDefaults = obj.includeDefaultValues;
        obj.includeDefaultValues = obj.group.includeDefaultValues;
        var _obj = obj.toDatalessObject(propertiesToInclude);
        obj.includeDefaultValues = originalDefaults;
        return _obj;
      });
      return extend(this.callSuper('toDatalessObject', propertiesToInclude), {
        objects: objsToObject
      });
    },

    /**
     * Renders instance on a given context
```

```
     * @param {CanvasRenderingContext2D} ctx context to render instance on
     */
    render: function(ctx) {
      this._transformDone = true;
      this.callSuper('render', ctx);
      this._transformDone = false;
    },

    /**
     * Decide if the object should cache or not.
     * objectCaching is a global flag, wins over everything
     * needsItsOwnCache should be used when the object drawing method requires
     * a cache step. None of the fabric classes requires it.
     * Generally you do not cache objects in groups because the group outside is
cached.
     * @return {Boolean}
     */
    shouldCache: function() {
      var parentCache = this.objectCaching && (!this.group ||
this.needsItsOwnCache() || !this.group.isCaching());
      this.caching = parentCache;
      if (parentCache) {
        for (var i = 0, len = this._objects.length; i < len; i++) {
          if (this._objects[i].willDrawShadow()) {
            this.caching = false;
            return false;
          }
        }
      }
      return parentCache;
    },

    /**
     * Check if this object or a child object will cast a shadow
     * @return {Boolean}
     */
    willDrawShadow: function() {
      if (this.callSuper('willDrawShadow')) {
        return true;
      }
      for (var i = 0, len = this._objects.length; i < len; i++) {
        if (this._objects[i].willDrawShadow()) {
          return true;
        }
      }
      return false;
    },

    /**
     * Check if this group or its parent group are caching, recursively up
     * @return {Boolean}
     */
    isCaching: function() {
      return this.caching || this.group && this.group.isCaching();
    },

    /**
     * Execute the drawing operation for an object on a specified context
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    drawObject: function(ctx) {
      for (var i = 0, len = this._objects.length; i < len; i++) {
        this._renderObject(this._objects[i], ctx);
```

```
      }
    },

    /**
     * Check if cache is dirty
     */
    isCacheDirty: function() {
      if (this.callSuper('isCacheDirty')) {
        return true;
      }
      if (!this.statefullCache) {
        return false;
      }
      for (var i = 0, len = this._objects.length; i < len; i++) {
        if (this._objects[i].isCacheDirty(true)) {
          if (this._cacheCanvas) {
            // if this group has not a cache canvas there is nothing to clean
            var x = this.cacheWidth / this.zoomX, y = this.cacheHeight /
this.zoomY;
            this._cacheContext.clearRect(-x / 2, -y / 2, x, y);
          }
          return true;
        }
      }
      return false;
    },

    /**
     * Renders controls and borders for the object
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} [noTransform] When true, context is not transformed
     */
    _renderControls: function(ctx, noTransform) {
      ctx.save();
      ctx.globalAlpha = this.isMoving ? this.borderOpacityWhenMoving : 1;
      this.callSuper('_renderControls', ctx, noTransform);
      for (var i = 0, len = this._objects.length; i < len; i++) {
        this._objects[i]._renderControls(ctx);
      }
      ctx.restore();
    },

    /**
     * @private
     */
    _renderObject: function(object, ctx) {
      // do not render if object is not visible
      if (!object.visible) {
        return;
      }

      var originalHasRotatingPoint = object.hasRotatingPoint;
      object.hasRotatingPoint = false;
      object.render(ctx);
      object.hasRotatingPoint = originalHasRotatingPoint;
    },

    /**
     * Retores original state of each of group objects (original state is that
which was before group was created).
     * @private
     * @return {fabric.Group} thisArg
     * @chainable
     */
```

```
_restoreObjectsState: function() {
  this._objects.forEach(this._restoreObjectState, this);
  return this;
},

/**
 * Realises the transform from this group onto the supplied object
 * i.e. it tells you what would happen if the supplied object was in
 * the group, and then the group was destroyed. It mutates the supplied
 * object.
 * @param {fabric.Object} object
 * @return {fabric.Object} transformedObject
 */
realizeTransform: function(object) {
  var matrix = object.calcTransformMatrix(),
      options = fabric.util.qrDecompose(matrix),
      center = new fabric.Point(options.translateX, options.translateY);
  object.flipX = false;
  object.flipY = false;
  object.set('scaleX', options.scaleX);
  object.set('scaleY', options.scaleY);
  object.skewX = options.skewX;
  object.skewY = options.skewY;
  object.angle = options.angle;
  object.setPositionByOrigin(center, 'center', 'center');
  return object;
},

/**
 * Restores original state of a specified object in group
 * @private
 * @param {fabric.Object} object
 * @return {fabric.Group} thisArg
 */
_restoreObjectState: function(object) {
  this.realizeTransform(object);
  object.setCoords();
  object.hasControls = object.__origHasControls;
  delete object.__origHasControls;
  object.set('active', false);
  delete object.group;

  return this;
},

/**
 * Destroys a group (restoring state of its objects)
 * @return {fabric.Group} thisArg
 * @chainable
 */
destroy: function() {
  // when group is destroyed objects needs to get a repaint to be eventually
  // displayed on canvas.
  this._objects.forEach(function(object) {
    object.set('dirty', true);
  });
  return this._restoreObjectsState();
},

/**
 * Saves coordinates of this instance (to be used together with `hasMoved`)
 * @saveCoords
 * @return {fabric.Group} thisArg
 * @chainable
```

```
     */
    saveCoords: function() {
      this._originalLeft = this.get('left');
      this._originalTop = this.get('top');
      return this;
    },

    /**
     * Checks whether this group was moved (since `saveCoords` was called last)
     * @return {Boolean} true if an object was moved (since
fabric.Group#saveCoords was called)
     */
    hasMoved: function() {
      return this._originalLeft !== this.get('left') ||
             this._originalTop !== this.get('top');
    },

    /**
     * Sets coordinates of all objects inside group
     * @return {fabric.Group} thisArg
     * @chainable
     */
    setObjectsCoords: function() {
      var ignoreZoom = true, skipAbsolute = true;
      this.forEachObject(function(object) {
        object.setCoords(ignoreZoom, skipAbsolute);
      });
      return this;
    },

    /**
     * @private
     */
    _calcBounds: function(onlyWidthHeight) {
      var aX = [],
          aY = [],
          o, prop,
          props = ['tr', 'br', 'bl', 'tl'],
          i = 0, iLen = this._objects.length,
          j, jLen = props.length,
          ignoreZoom = true;

      for ( ; i < iLen; ++i) {
        o = this._objects[i];
        o.setCoords(ignoreZoom);
        for (j = 0; j < jLen; j++) {
          prop = props[j];
          aX.push(o.oCoords[prop].x);
          aY.push(o.oCoords[prop].y);
        }
      }

      this.set(this._getBounds(aX, aY, onlyWidthHeight));
    },

    /**
     * @private
     */
    _getBounds: function(aX, aY, onlyWidthHeight) {
      var minXY = new fabric.Point(min(aX), min(aY)),
          maxXY = new fabric.Point(max(aX), max(aY)),
          obj = {
            width: (maxXY.x - minXY.x) || 0,
            height: (maxXY.y - minXY.y) || 0
```

```
        };

      if (!onlyWidthHeight) {
        obj.left = minXY.x || 0;
        obj.top = minXY.y || 0;
        if (this.originX === 'center') {
          obj.left += obj.width / 2;
        }
        if (this.originX === 'right') {
          obj.left += obj.width;
        }
        if (this.originY === 'center') {
          obj.top += obj.height / 2;
        }
        if (this.originY === 'bottom') {
          obj.top += obj.height;
        }
      }
      return obj;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns svg representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup();
      markup.push(
        '<g ', this.getSvgId(), 'transform="',
        /* avoiding styles intentionally */
        this.getSvgTransform(),
        this.getSvgTransformMatrix(),
        '" style="',
        this.getSvgFilter(),
        '">\n'
      );

      for (var i = 0, len = this._objects.length; i < len; i++) {
        markup.push('\t', this._objects[i].toSVG(reviver));
      }

      markup.push('</g>\n');

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns requested property
     * @param {String} prop Property to get
     * @return {*}
     */
    get: function(prop) {
      if (prop in _lockProperties) {
        if (this[prop]) {
          return this[prop];
        }
        else {
          for (var i = 0, len = this._objects.length; i < len; i++) {
            if (this._objects[i][prop]) {
              return true;
```

```
            }
          }
          return false;
        }
      }
      else {
        if (prop in this.delegatedProperties) {
          return this._objects[0] && this._objects[0].get(prop);
        }
        return this[prop];
      }
    }
  });

  /**
   * Returns {@link fabric.Group} instance from an object representation
   * @static
   * @memberOf fabric.Group
   * @param {Object} object Object to create a group from
   * @param {Function} [callback] Callback to invoke when an group instance is
created
   */
  fabric.Group.fromObject = function(object, callback) {
    fabric.util.enlivenObjects(object.objects, function(enlivenedObjects) {
      var options = fabric.util.object.clone(object, true);
      delete options.objects;
      callback && callback(new fabric.Group(enlivenedObjects, options, true));
    });
  };

  /**
   * Indicates that instances of this type are async
   * @static
   * @memberOf fabric.Group
   * @type Boolean
   * @default
   */
  fabric.Group.async = true;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var extend = fabric.util.object.extend;

  if (!global.fabric) {
    global.fabric = { };
  }

  if (global.fabric.Image) {
    fabric.warn('fabric.Image is already defined.');
    return;
  }

  var stateProperties = fabric.Object.prototype.stateProperties.concat();
  stateProperties.push(
    'alignX',
    'alignY',
    'meetOrSlice'
  );
```

```
  /**
   * Image class
   * @class fabric.Image
   * @extends fabric.Object
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-1#images}
   * @see {@link fabric.Image#initialize} for constructor definition
   */
  fabric.Image = fabric.util.createClass(fabric.Object, /** @lends
fabric.Image.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'image',

    /**
     * crossOrigin value (one of "", "anonymous", "use-credentials")
     * @see
https://developer.mozilla.org/en-US/docs/HTML/CORS_settings_attributes
     * @type String
     * @default
     */
    crossOrigin: '',

    /**
     * AlignX value, part of preserveAspectRatio (one of "none", "mid", "min",
"max")
     * @see http://www.w3.org/TR/SVG/coords.html#PreserveAspectRatioAttribute
     * This parameter defines how the picture is aligned to its viewport when
image element width differs from image width.
     * @type String
     * @default
     */
    alignX: 'none',

    /**
     * AlignY value, part of preserveAspectRatio (one of "none", "mid", "min",
"max")
     * @see http://www.w3.org/TR/SVG/coords.html#PreserveAspectRatioAttribute
     * This parameter defines how the picture is aligned to its viewport when
image element height differs from image height.
     * @type String
     * @default
     */
    alignY: 'none',

    /**
     * meetOrSlice value, part of preserveAspectRatio  (one of "meet", "slice").
     * if meet the image is always fully visibile, if slice the viewport is
always filled with image.
     * @see http://www.w3.org/TR/SVG/coords.html#PreserveAspectRatioAttribute
     * @type String
     * @default
     */
    meetOrSlice: 'meet',

    /**
     * Width of a stroke.
     * For image quality a stroke multiple of 2 gives better results.
     * @type Number
     * @default
     */
```

```
    strokeWidth: 0,

    /**
     * private
     * contains last value of scaleX to detect
     * if the Image got resized after the last Render
     * @type Number
     */
    _lastScaleX: 1,

    /**
     * private
     * contains last value of scaleY to detect
     * if the Image got resized after the last Render
     * @type Number
     */
    _lastScaleY: 1,

    /**
     * minimum scale factor under which any resizeFilter is triggered to resize
the image
     * 0 will disable the automatic resize. 1 will trigger automatically always.
     * number bigger than 1 can be used in case we want to scale with some
filter above
     * the natural image dimensions
     * @type Number
     */
    minimumScaleTrigger: 0.5,

    /**
     * List of properties to consider when checking if
     * state of an object is changed ({@link fabric.Object#hasStateChanged})
     * as well as for history (undo/redo) purposes
     * @type Array
     */
    stateProperties: stateProperties,

    /**
     * When `true`, object is cached on an additional canvas.
     * default to false for images
     * since 1.7.0
     * @type Boolean
     * @default
     */
    objectCaching: false,

    /**
     * Constructor
     * @param {HTMLImageElement | String} element Image element
     * @param {Object} [options] Options object
     * @param {function} [callback] callback function to call after eventual
filters applied.
     * @return {fabric.Image} thisArg
     */
    initialize: function(element, options, callback) {
      options || (options = { });
      this.filters = [];
      this.resizeFilters = [];
      this.callSuper('initialize', options);
      this._initElement(element, options, callback);
    },

    /**
     * Returns image element which this instance if based on
```

```
     * @return {HTMLImageElement} Image element
     */
    getElement: function() {
      return this._element;
    },

    /**
     * Sets image element for this instance to a specified one.
     * If filters defined they are applied to new image.
     * You might need to call `canvas.renderAll` and `object.setCoords` after
replacing, to render new image and update controls area.
     * @param {HTMLImageElement} element
     * @param {Function} [callback] Callback is invoked when all filters have
been applied and new image is generated
     * @param {Object} [options] Options object
     * @return {fabric.Image} thisArg
     * @chainable
     */
    setElement: function(element, callback, options) {

      var _callback, _this;

      this._element = element;
      this._originalElement = element;
      this._initConfig(options);

      if (this.resizeFilters.length === 0) {
        _callback = callback;
      }
      else {
        _this = this;
        _callback = function() {
          _this.applyFilters(callback, _this.resizeFilters, _this._filteredEl ||
_this._originalElement, true);
        };
      }

      if (this.filters.length !== 0) {
        this.applyFilters(_callback);
      }
      else if (_callback) {
        _callback(this);
      }

      return this;
    },

    /**
     * Sets crossOrigin value (on an instance and corresponding image element)
     * @return {fabric.Image} thisArg
     * @chainable
     */
    setCrossOrigin: function(value) {
      this.crossOrigin = value;
      this._element.crossOrigin = value;

      return this;
    },

    /**
     * Returns original size of an image
     * @return {Object} Object with "width" and "height" properties
     */
    getOriginalSize: function() {
```

```
      var element = this.getElement();
      return {
        width: element.width,
        height: element.height
      };
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _stroke: function(ctx) {
      if (!this.stroke || this.strokeWidth === 0) {
        return;
      }
      var w = this.width / 2, h = this.height / 2;
      ctx.beginPath();
      ctx.moveTo(-w, -h);
      ctx.lineTo(w, -h);
      ctx.lineTo(w, h);
      ctx.lineTo(-w, h);
      ctx.lineTo(-w, -h);
      ctx.closePath();
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderDashedStroke: function(ctx) {
      var x = -this.width / 2,
          y = -this.height / 2,
          w = this.width,
          h = this.height;

      ctx.save();
      this._setStrokeStyles(ctx);

      ctx.beginPath();
      fabric.util.drawDashedLine(ctx, x, y, x + w, y, this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x + w, y, x + w, y + h,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x + w, y + h, x, y + h,
this.strokeDashArray);
      fabric.util.drawDashedLine(ctx, x, y + h, x, y, this.strokeDashArray);
      ctx.closePath();
      ctx.restore();
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} Object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      var filters = [], resizeFilters = [],
          scaleX = 1, scaleY = 1;

      this.filters.forEach(function(filterObj) {
        if (filterObj) {
          if (filterObj.type === 'Resize') {
            scaleX *= filterObj.scaleX;
            scaleY *= filterObj.scaleY;
```

```
          }
          filters.push(filterObj.toObject());
        }
      });

      this.resizeFilters.forEach(function(filterObj) {
        filterObj && resizeFilters.push(filterObj.toObject());
      });
      var object = extend(
        this.callSuper(
          'toObject',
          ['crossOrigin', 'alignX', 'alignY',
'meetOrSlice'].concat(propertiesToInclude)
        ), {
          src: this.getSrc(),
          filters: filters,
          resizeFilters: resizeFilters,
        });

      object.width /= scaleX;
      object.height /= scaleY;

      return object;
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      var markup = this._createBaseSVGMarkup(), x = -this.width / 2, y = -
this.height / 2,
          preserveAspectRatio = 'none', filtered = true;
      if (this.group && this.group.type === 'path-group') {
        x = this.left;
        y = this.top;
      }
      if (this.alignX !== 'none' && this.alignY !== 'none') {
        preserveAspectRatio = 'x' + this.alignX + 'Y' + this.alignY + ' ' +
this.meetOrSlice;
      }
      markup.push(
        '<g transform="', this.getSvgTransform(), this.getSvgTransformMatrix(),
'">\n',
          '<image ', this.getSvgId(), 'xlink:href="', this.getSvgSrc(filtered),
            '" x="', x, '" y="', y,
            '" style="', this.getSvgStyles(),
            // we're essentially moving origin of transformation from top/left
corner to the center of the shape
            // by wrapping it in container <g> element with actual
transformation, then offsetting object to the top/left
            // so that object's center aligns with container's left/top
            '" width="', this.width,
            '" height="', this.height,
            '" preserveAspectRatio="', preserveAspectRatio, '"',
          '></image>\n'
      );

      if (this.stroke || this.strokeDashArray) {
        var origFill = this.fill;
        this.fill = null;
```

```
        markup.push(
          '<rect ',
            'x="', x, '" y="', y,
            '" width="', this.width, '" height="', this.height,
            '" style="', this.getSvgStyles(),
          '"/>\n'
        );
        this.fill = origFill;
      }

      markup.push('</g>\n');

      return reviver ? reviver(markup.join('')) : markup.join('');
    },
    /* _TO_SVG_END_ */

    /**
     * Returns source of an image
     * @param {Boolean} filtered indicates if the src is needed for svg
     * @return {String} Source of an image
     */
    getSrc: function(filtered) {
      var element = filtered ? this._element : this._originalElement;
      if (element) {
        return fabric.isLikelyNode ? element._src : element.src;
      }
      else {
        return this.src || '';
      }
    },

    /**
     * Sets source of an image
     * @param {String} src Source string (URL)
     * @param {Function} [callback] Callback is invoked when image has been
loaded (and all filters have been applied)
     * @param {Object} [options] Options object
     * @return {fabric.Image} thisArg
     * @chainable
     */
    setSrc: function(src, callback, options) {
      fabric.util.loadImage(src, function(img) {
        return this.setElement(img, callback, options);
      }, this, options && options.crossOrigin);
    },

    /**
     * Returns string representation of an instance
     * @return {String} String representation of an instance
     */
    toString: function() {
      return '#<fabric.Image: { src: "' + this.getSrc() + '" }>';
    },

    /**
     * Applies filters assigned to this image (from "filters" array)
     * @method applyFilters
     * @param {Function} callback Callback is invoked when all filters have been
applied and new image is generated
     * @param {Array} filters to be applied
     * @param {fabric.Image} imgElement image to filter ( default to
this._element )
     * @param {Boolean} forResizing
     * @return {CanvasElement} canvasEl to be drawn immediately
```

```
     * @chainable
     */
    applyFilters: function(callback, filters, imgElement, forResizing) {

      filters = filters || this.filters;
      imgElement = imgElement || this._originalElement;

      if (!imgElement) {
        return;
      }

      var replacement = fabric.util.createImage(),
          retinaScaling = this.canvas ? this.canvas.getRetinaScaling() :
fabric.devicePixelRatio,
          minimumScale = this.minimumScaleTrigger / retinaScaling,
          _this = this, scaleX, scaleY;

      if (filters.length === 0) {
        this._element = imgElement;
        callback && callback(this);
        return imgElement;
      }

      var canvasEl = fabric.util.createCanvasElement();
      canvasEl.width = imgElement.width;
      canvasEl.height = imgElement.height;
      canvasEl.getContext('2d').drawImage(imgElement, 0, 0, imgElement.width,
imgElement.height);

      filters.forEach(function(filter) {
        if (!filter) {
          return;
        }
        if (forResizing) {
          scaleX = _this.scaleX < minimumScale ? _this.scaleX : 1;
          scaleY = _this.scaleY < minimumScale ? _this.scaleY : 1;
          if (scaleX * retinaScaling < 1) {
            scaleX *= retinaScaling;
          }
          if (scaleY * retinaScaling < 1) {
            scaleY *= retinaScaling;
          }
        }
        else {
          scaleX = filter.scaleX;
          scaleY = filter.scaleY;
        }
        filter.applyTo(canvasEl, scaleX, scaleY);
        if (!forResizing && filter.type === 'Resize') {
          _this.width *= filter.scaleX;
          _this.height *= filter.scaleY;
        }
      });

      /** @ignore */
      replacement.width = canvasEl.width;
      replacement.height = canvasEl.height;
      if (fabric.isLikelyNode) {
        replacement.src = canvasEl.toBuffer(undefined,
fabric.Image.pngCompression);
        // onload doesn't fire in some node versions, so we invoke callback
manually
        _this._element = replacement;
        !forResizing && (_this._filteredEl = replacement);
```

```
        callback && callback(_this);
      }
      else {
        replacement.onload = function() {
          _this._element = replacement;
          !forResizing && (_this._filteredEl = replacement);
          callback && callback(_this);
          replacement.onload = canvasEl = null;
        };
        replacement.src = canvasEl.toDataURL('image/png');
      }
      return canvasEl;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    _render: function(ctx, noTransform) {
      var x, y, imageMargins = this._findMargins(), elementToDraw;

      x = (noTransform ? this.left : -this.width / 2);
      y = (noTransform ? this.top : -this.height / 2);

      if (this.meetOrSlice === 'slice') {
        ctx.beginPath();
        ctx.rect(x, y, this.width, this.height);
        ctx.clip();
      }

      if (this.isMoving === false && this.resizeFilters.length &&
this._needsResize()) {
        this._lastScaleX = this.scaleX;
        this._lastScaleY = this.scaleY;
        elementToDraw = this.applyFilters(null, this.resizeFilters,
this._filteredEl || this._originalElement, true);
      }
      else {
        elementToDraw = this._element;
      }
      elementToDraw && ctx.drawImage(elementToDraw,
                                     x + imageMargins.marginX,
                                     y + imageMargins.marginY,
                                     imageMargins.width,
                                     imageMargins.height
                                    );

      this._stroke(ctx);
      this._renderStroke(ctx);
    },

    /**
     * @private, needed to check if image needs resize
     */
    _needsResize: function() {
      return (this.scaleX !== this._lastScaleX || this.scaleY !==
this._lastScaleY);
    },

    /**
     * @private
     */
    _findMargins: function() {
```

```
      var width = this.width, height = this.height, scales,
          scale, marginX = 0, marginY = 0;

      if (this.alignX !== 'none' || this.alignY !== 'none') {
        scales = [this.width / this._element.width, this.height /
this._element.height];
        scale = this.meetOrSlice === 'meet'
                ? Math.min.apply(null, scales) : Math.max.apply(null, scales);
        width = this._element.width * scale;
        height = this._element.height * scale;
        if (this.alignX === 'Mid') {
          marginX = (this.width - width) / 2;
        }
        if (this.alignX === 'Max') {
          marginX = this.width - width;
        }
        if (this.alignY === 'Mid') {
          marginY = (this.height - height) / 2;
        }
        if (this.alignY === 'Max') {
          marginY = this.height - height;
        }
      }
      return {
        width:  width,
        height: height,
        marginX: marginX,
        marginY: marginY
      };
    },

    /**
     * @private
     */
    _resetWidthHeight: function() {
      var element = this.getElement();

      this.set('width', element.width);
      this.set('height', element.height);
    },

    /**
     * The Image class's initialization method. This method is automatically
     * called by the constructor.
     * @private
     * @param {HTMLImageElement|String} element The element representing the
image
     * @param {Object} [options] Options object
     */
    _initElement: function(element, options, callback) {
      this.setElement(fabric.util.getById(element), callback, options);
      fabric.util.addClass(this.getElement(), fabric.Image.CSS_CANVAS);
    },

    /**
     * @private
     * @param {Object} [options] Options object
     */
    _initConfig: function(options) {
      options || (options = { });
      this.setOptions(options);
      this._setWidthHeight(options);
      if (this._element && this.crossOrigin) {
        this._element.crossOrigin = this.crossOrigin;
```

```
      }
    },

    /**
     * @private
     * @param {Array} filters to be initialized
     * @param {Function} callback Callback to invoke when all
fabric.Image.filters instances are created
     */
    _initFilters: function(filters, callback) {
      if (filters && filters.length) {
        fabric.util.enlivenObjects(filters, function(enlivenedObjects) {
          callback && callback(enlivenedObjects);
        }, 'fabric.Image.filters');
      }
      else {
        callback && callback();
      }
    },

    /**
     * @private
     * @param {Object} [options] Object with width/height properties
     */
    _setWidthHeight: function(options) {
      this.width = 'width' in options
        ? options.width
        : (this.getElement()
            ? this.getElement().width || 0
            : 0);

      this.height = 'height' in options
        ? options.height
        : (this.getElement()
            ? this.getElement().height || 0
            : 0);
    },
  });

  /**
   * Default CSS class name for canvas
   * @static
   * @type String
   * @default
   */
  fabric.Image.CSS_CANVAS = 'canvas-img';

  /**
   * Alias for getSrc
   * @static
   */
  fabric.Image.prototype.getSvgSrc = fabric.Image.prototype.getSrc;

  /**
   * Creates an instance of fabric.Image from its object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} callback Callback to invoke when an image instance is
created
   */
  fabric.Image.fromObject = function(object, callback) {
    fabric.util.loadImage(object.src, function(img, error) {
      if (error) {
        callback && callback(null, error);
```

```
        return;
      }
      fabric.Image.prototype._initFilters.call(object, object.filters,
function(filters) {
        object.filters = filters || [];
        fabric.Image.prototype._initFilters.call(object, object.resizeFilters,
function(resizeFilters) {
          object.resizeFilters = resizeFilters || [];
          return new fabric.Image(img, object, callback);
        });
      });
    }, null, object.crossOrigin);
  };

  /**
   * Creates an instance of fabric.Image from an URL string
   * @static
   * @param {String} url URL to create an image from
   * @param {Function} [callback] Callback to invoke when image is created
(newly created image is passed as a first argument)
   * @param {Object} [imgOptions] Options object
   */
  fabric.Image.fromURL = function(url, callback, imgOptions) {
    fabric.util.loadImage(url, function(img) {
      callback && callback(new fabric.Image(img, imgOptions));
    }, null, imgOptions && imgOptions.crossOrigin);
  };

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
{@link fabric.Image.fromElement})
   * @static
   * @see {@link http://www.w3.org/TR/SVG/struct.html#ImageElement}
   */
  fabric.Image.ATTRIBUTE_NAMES =
    fabric.SHARED_ATTRIBUTES.concat('x y width height preserveAspectRatio
xlink:href crossOrigin'.split(' '));

  /**
   * Returns {@link fabric.Image} instance from an SVG element
   * @static
   * @param {SVGElement} element Element to parse
   * @param {Function} callback Callback to execute when fabric.Image object is
created
   * @param {Object} [options] Options object
   * @return {fabric.Image} Instance of fabric.Image
   */
  fabric.Image.fromElement = function(element, callback, options) {
    var parsedAttributes = fabric.parseAttributes(element,
fabric.Image.ATTRIBUTE_NAMES),
        preserveAR;

    if (parsedAttributes.preserveAspectRatio) {
      preserveAR =
fabric.util.parsePreserveAspectRatioAttribute(parsedAttributes.preserveAspectRat
io);
      extend(parsedAttributes, preserveAR);
    }

    fabric.Image.fromURL(parsedAttributes['xlink:href'], callback,
      extend((options ? fabric.util.object.clone(options) : { }),
parsedAttributes));
  };
```

```
  /* _FROM_SVG_END_ */

  /**
   * Indicates that instances of this type are async
   * @static
   * @type Boolean
   * @default
   */
  fabric.Image.async = true;

  /**
   * Indicates compression level used when generating PNG under Node (in
applyFilters). Any of 0-9
   * @static
   * @type Number
   * @default
   */
  fabric.Image.pngCompression = 1;

})(typeof exports !== 'undefined' ? exports : this);


fabric.util.object.extend(fabric.Object.prototype, /** @lends
fabric.Object.prototype */ {

  /**
   * @private
   * @return {Number} angle value
   */
  _getAngleValueForStraighten: function() {
    var angle = this.getAngle() % 360;
    if (angle > 0) {
      return Math.round((angle - 1) / 90) * 90;
    }
    return Math.round(angle / 90) * 90;
  },

  /**
   * Straightens an object (rotating it from current angle to one of 0, 90, 180,
270, etc. depending on which is closer)
   * @return {fabric.Object} thisArg
   * @chainable
   */
  straighten: function() {
    this.setAngle(this._getAngleValueForStraighten());
    return this;
  },

  /**
   * Same as {@link fabric.Object.prototype.straighten} but with animation
   * @param {Object} callbacks Object with callback functions
   * @param {Function} [callbacks.onComplete] Invoked on completion
   * @param {Function} [callbacks.onChange] Invoked on every step of animation
   * @return {fabric.Object} thisArg
   * @chainable
   */
  fxStraighten: function(callbacks) {
    callbacks = callbacks || { };

    var empty = function() { },
        onComplete = callbacks.onComplete || empty,
        onChange = callbacks.onChange || empty,
        _this = this;
```

```
    fabric.util.animate({
      startValue: this.get('angle'),
      endValue: this._getAngleValueForStraighten(),
      duration: this.FX_DURATION,
      onChange: function(value) {
        _this.setAngle(value);
        onChange();
      },
      onComplete: function() {
        _this.setCoords();
        onComplete();
      },
      onStart: function() {
        _this.set('active', false);
      }
    });

    return this;
  }
});

fabric.util.object.extend(fabric.StaticCanvas.prototype, /** @lends
fabric.StaticCanvas.prototype */ {

  /**
   * Straightens object, then rerenders canvas
   * @param {fabric.Object} object Object to straighten
   * @return {fabric.Canvas} thisArg
   * @chainable
   */
  straightenObject: function (object) {
    object.straighten();
    this.renderAll();
    return this;
  },

  /**
   * Same as {@link fabric.Canvas.prototype.straightenObject}, but animated
   * @param {fabric.Object} object Object to straighten
   * @return {fabric.Canvas} thisArg
   * @chainable
   */
  fxStraightenObject: function (object) {
    object.fxStraighten({
      onChange: this.renderAll.bind(this)
    });
    return this;
  }
});


/**
 * @namespace fabric.Image.filters
 * @memberOf fabric.Image
 * @tutorial {@link http://fabricjs.com/fabric-intro-part-2#image_filters}
 * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
 */
fabric.Image.filters = fabric.Image.filters || { };

/**
 * Root filter class from which all filter classes inherit from
 * @class fabric.Image.filters.BaseFilter
 * @memberOf fabric.Image.filters
 */
```

```
fabric.Image.filters.BaseFilter = fabric.util.createClass(/** @lends
fabric.Image.filters.BaseFilter.prototype */ {

  /**
   * Filter type
   * @param {String} type
   * @default
   */
  type: 'BaseFilter',

  /**
   * Constructor
   * @param {Object} [options] Options object
   */
  initialize: function(options) {
    if (options) {
      this.setOptions(options);
    }
  },

  /**
   * Sets filter's properties from options
   * @param {Object} [options] Options object
   */
  setOptions: function(options) {
    for (var prop in options) {
      this[prop] = options[prop];
    }
  },

  /**
   * Returns object representation of an instance
   * @return {Object} Object representation of an instance
   */
  toObject: function() {
    return { type: this.type };
  },

  /**
   * Returns a JSON representation of an instance
   * @return {Object} JSON
   */
  toJSON: function() {
    // delegate, not alias
    return this.toObject();
  }
});

fabric.Image.filters.BaseFilter.fromObject = function(object, callback) {
  var filter = new fabric.Image.filters[object.type](object);
  callback && callback(filter);
  return filter;
};


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;
```

```
  /**
   * Brightness filter class
   * @class fabric.Image.filters.Brightness
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Brightness#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Brightness({
   *   brightness: 200
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Brightness = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Brightness.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Brightness',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Brightness.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.brightness=0] Value to brighten the image up (-
255..255)
     */
    initialize: function(options) {
      options = options || { };
      this.brightness = options.brightness || 0;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          brightness = this.brightness;

      for (var i = 0, len = data.length; i < len; i += 4) {
        data[i] += brightness;
        data[i + 1] += brightness;
        data[i + 2] += brightness;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        brightness: this.brightness
```

```
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Brightness} Instance of
fabric.Image.filters.Brightness
   */
  fabric.Image.filters.Brightness.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Adapted from <a
href="http://www.html5rocks.com/en/tutorials/canvas/imagefilters/">html5rocks
article</a>
   * @class fabric.Image.filters.Convolute
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Convolute#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example <caption>Sharpen filter</caption>
   * var filter = new fabric.Image.filters.Convolute({
   *   matrix: [ 0, -1,  0,
   *            -1,  5, -1,
   *             0, -1,  0 ]
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   * @example <caption>Blur filter</caption>
   * var filter = new fabric.Image.filters.Convolute({
   *   matrix: [ 1/9, 1/9, 1/9,
   *             1/9, 1/9, 1/9,
   *             1/9, 1/9, 1/9 ]
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   * @example <caption>Emboss filter</caption>
   * var filter = new fabric.Image.filters.Convolute({
   *   matrix: [ 1,   1,  1,
   *             1, 0.7, -1,
   *            -1,  -1, -1 ]
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   * @example <caption>Emboss filter with opaqueness</caption>
   * var filter = new fabric.Image.filters.Convolute({
   *   opaque: true,
```

```
   *   matrix: [ 1,   1,  1,
   *             1, 0.7, -1,
   *            -1,  -1, -1 ]
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Convolute = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Convolute.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Convolute',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Convolute.prototype
     * @param {Object} [options] Options object
     * @param {Boolean} [options.opaque=false] Opaque value (true/false)
     * @param {Array} [options.matrix] Filter matrix
     */
    initialize: function(options) {
      options = options || { };

      this.opaque = options.opaque;
      this.matrix = options.matrix || [
        0, 0, 0,
        0, 1, 0,
        0, 0, 0
      ];
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {

      var weights = this.matrix,
          context = canvasEl.getContext('2d'),
          pixels = context.getImageData(0, 0, canvasEl.width, canvasEl.height),

          side = Math.round(Math.sqrt(weights.length)),
          halfSide = Math.floor(side / 2),
          src = pixels.data,
          sw = pixels.width,
          sh = pixels.height,
          output = context.createImageData(sw, sh),
          dst = output.data,
          // go through the destination image pixels
          alphaFac = this.opaque ? 1 : 0,
          r, g, b, a, dstOff,
          scx, scy, srcOff, wt;

      for (var y = 0; y < sh; y++) {
        for (var x = 0; x < sw; x++) {
          dstOff = (y * sw + x) * 4;
          // calculate the weighed sum of the source image pixels that
          // fall under the convolution matrix
          r = 0; g = 0; b = 0; a = 0;
```

```
          for (var cy = 0; cy < side; cy++) {
            for (var cx = 0; cx < side; cx++) {
              scy = y + cy - halfSide;
              scx = x + cx - halfSide;

              // eslint-disable-next-line max-depth
              if (scy < 0 || scy > sh || scx < 0 || scx > sw) {
                continue;
              }

              srcOff = (scy * sw + scx) * 4;
              wt = weights[cy * side + cx];

              r += src[srcOff] * wt;
              g += src[srcOff + 1] * wt;
              b += src[srcOff + 2] * wt;
              a += src[srcOff + 3] * wt;
            }
          }
          dst[dstOff] = r;
          dst[dstOff + 1] = g;
          dst[dstOff + 2] = b;
          dst[dstOff + 3] = a + alphaFac * (255 - a);
        }
      }

      context.putImageData(output, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        opaque: this.opaque,
        matrix: this.matrix
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Convolute} Instance of
fabric.Image.filters.Convolute
   */
  fabric.Image.filters.Convolute.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;
```

```
  /**
   * GradientTransparency filter class
   * @class fabric.Image.filters.GradientTransparency
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.GradientTransparency#initialize} for
constructor definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.GradientTransparency({
   *   threshold: 200
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
   // eslint-disable-next-line max-len
  filters.GradientTransparency = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.GradientTransparency.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'GradientTransparency',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.GradientTransparency.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.threshold=100] Threshold value
     */
    initialize: function(options) {
      options = options || { };
      this.threshold = options.threshold || 100;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          threshold = this.threshold,
          total = data.length;

      for (var i = 0, len = data.length; i < len; i += 4) {
        data[i + 3] = threshold + 255 * (total - i) / total;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        threshold: this.threshold
      });
```

```
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.GradientTransparency} Instance of
fabric.Image.filters.GradientTransparency
   */
  fabric.Image.filters.GradientTransparency.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Grayscale image filter class
   * @class fabric.Image.filters.Grayscale
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Grayscale();
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Grayscale = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Grayscale.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Grayscale',

    /**
     * Applies filter to canvas element
     * @memberOf fabric.Image.filters.Grayscale.prototype
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          len = imageData.width * imageData.height * 4,
          index = 0,
          average;

      while (index < len) {
        average = (data[index] + data[index + 1] + data[index + 2]) / 3;
        data[index]     = average;
        data[index + 1] = average;
```

```
        data[index + 2] = average;
        index += 4;
      }

      context.putImageData(imageData, 0, 0);
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Grayscale} Instance of
fabric.Image.filters.Grayscale
   */
  fabric.Image.filters.Grayscale.fromObject = function(object, callback) {
    object = object || { };
    object.type = 'Grayscale';
    return fabric.Image.filters.BaseFilter.fromObject(object, callback);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Invert filter class
   * @class fabric.Image.filters.Invert
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Invert();
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Invert = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Invert.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Invert',

    /**
     * Applies filter to canvas element
     * @memberOf fabric.Image.filters.Invert.prototype
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
```

```
          iLen = data.length, i;

      for (i = 0; i < iLen; i += 4) {
        data[i] = 255 - data[i];
        data[i + 1] = 255 - data[i + 1];
        data[i + 2] = 255 - data[i + 2];
      }

      context.putImageData(imageData, 0, 0);
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Invert} Instance of
fabric.Image.filters.Invert
   */
  fabric.Image.filters.Invert.fromObject = function(object, callback) {
    object = object || { };
    object.type = 'Invert';
    return fabric.Image.filters.BaseFilter.fromObject(object, callback);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Mask filter class
   * See http://resources.aleph-1.com/mask/
   * @class fabric.Image.filters.Mask
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Mask#initialize} for constructor
definition
   */
  filters.Mask = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Mask.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Mask',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Mask.prototype
     * @param {Object} [options] Options object
     * @param {fabric.Image} [options.mask] Mask image object
     * @param {Number} [options.channel=0] Rgb channel (0, 1, 2 or 3)
     */
```

```
    initialize: function(options) {
      options = options || { };

      this.mask = options.mask;
      this.channel = [0, 1, 2, 3].indexOf(options.channel) > -1 ?
options.channel : 0;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      if (!this.mask) {
        return;
      }

      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          maskEl = this.mask.getElement(),
          maskCanvasEl = fabric.util.createCanvasElement(),
          channel = this.channel,
          i,
          iLen = imageData.width * imageData.height * 4;

      maskCanvasEl.width = canvasEl.width;
      maskCanvasEl.height = canvasEl.height;

      maskCanvasEl.getContext('2d').drawImage(maskEl, 0, 0, canvasEl.width,
canvasEl.height);

      var maskImageData = maskCanvasEl.getContext('2d').getImageData(0, 0,
canvasEl.width, canvasEl.height),
          maskData = maskImageData.data;

      for (i = 0; i < iLen; i += 4) {
        data[i + 3] = maskData[i + channel];
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        mask: this.mask.toObject(),
        channel: this.channel
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when a mask filter instance
is created
   */
  fabric.Image.filters.Mask.fromObject = function(object, callback) {
```

```
    fabric.util.loadImage(object.mask.src, function(img) {
      object.mask = new fabric.Image(img, object.mask);
      return fabric.Image.filters.BaseFilter.fromObject(object, callback);
    });
  };

  /**
   * Indicates that instances of this type are async
   * @static
   * @type Boolean
   * @default
   */
  fabric.Image.filters.Mask.async = true;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Noise filter class
   * @class fabric.Image.filters.Noise
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Noise#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Noise({
   *   noise: 700
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Noise = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Noise.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Noise',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Noise.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.noise=0] Noise value
     */
    initialize: function(options) {
      options = options || { };
      this.noise = options.noise || 0;
    },

    /**
     * Applies filter to canvas element
```

```
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          noise = this.noise, rand;

      for (var i = 0, len = data.length; i < len; i += 4) {

        rand = (0.5 - Math.random()) * noise;

        data[i] += rand;
        data[i + 1] += rand;
        data[i + 2] += rand;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        noise: this.noise
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Noise} Instance of fabric.Image.filters.Noise
   */
  fabric.Image.filters.Noise.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Pixelate filter class
   * @class fabric.Image.filters.Pixelate
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Pixelate#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Pixelate({
```

```
   *   blocksize: 8
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Pixelate = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Pixelate.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Pixelate',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Pixelate.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.blocksize=4] Blocksize for pixelate
     */
    initialize: function(options) {
      options = options || { };
      this.blocksize = options.blocksize || 4;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          iLen = imageData.height,
          jLen = imageData.width,
          index, i, j, r, g, b, a;

      for (i = 0; i < iLen; i += this.blocksize) {
        for (j = 0; j < jLen; j += this.blocksize) {

          index = (i * 4) * jLen + (j * 4);

          r = data[index];
          g = data[index + 1];
          b = data[index + 2];
          a = data[index + 3];

          /*
           blocksize: 4

           [1,x,x,x,1]
           [x,x,x,x,1]
           [x,x,x,x,1]
           [x,x,x,x,1]
           [1,1,1,1,1]
           */

          for (var _i = i, _ilen = i + this.blocksize; _i < _ilen; _i++) {
            for (var _j = j, _jlen = j + this.blocksize; _j < _jlen; _j++) {
              index = (_i * 4) * jLen + (_j * 4);
              data[index] = r;
              data[index + 1] = g;
```

```
              data[index + 2] = b;
              data[index + 3] = a;
            }
          }
        }
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        blocksize: this.blocksize
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Pixelate} Instance of
fabric.Image.filters.Pixelate
   */
  fabric.Image.filters.Pixelate.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Remove white filter class
   * @class fabric.Image.filters.RemoveWhite
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.RemoveWhite#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.RemoveWhite({
   *   threshold: 40,
   *   distance: 140
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.RemoveWhite = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.RemoveWhite.prototype */ {

    /**
```

```
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'RemoveWhite',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.RemoveWhite.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.threshold=30] Threshold value
     * @param {Number} [options.distance=20] Distance value
     */
    initialize: function(options) {
      options = options || { };
      this.threshold = options.threshold || 30;
      this.distance = options.distance || 20;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          threshold = this.threshold,
          distance = this.distance,
          limit = 255 - threshold,
          abs = Math.abs,
          r, g, b;

      for (var i = 0, len = data.length; i < len; i += 4) {
        r = data[i];
        g = data[i + 1];
        b = data[i + 2];

        if (r > limit &&
            g > limit &&
            b > limit &&
            abs(r - g) < distance &&
            abs(r - b) < distance &&
            abs(g - b) < distance
        ) {
          data[i + 3] = 0;
        }
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        threshold: this.threshold,
        distance: this.distance
      });
    }
  });
```

```
  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.RemoveWhite} Instance of
fabric.Image.filters.RemoveWhite
   */
  fabric.Image.filters.RemoveWhite.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Sepia filter class
   * @class fabric.Image.filters.Sepia
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Sepia();
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Sepia = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Sepia.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Sepia',

    /**
     * Applies filter to canvas element
     * @memberOf fabric.Image.filters.Sepia.prototype
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          iLen = data.length, i, avg;

      for (i = 0; i < iLen; i += 4) {
        avg = 0.3  * data[i] + 0.59 * data[i + 1] + 0.11 * data[i + 2];
        data[i] = avg + 100;
        data[i + 1] = avg + 50;
        data[i + 2] = avg + 255;
      }

      context.putImageData(imageData, 0, 0);
```

```
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Sepia} Instance of fabric.Image.filters.Sepia
   */
  fabric.Image.filters.Sepia.fromObject = function(object, callback) {
    object = object || { };
    object.type = 'Sepia';
    return new fabric.Image.filters.BaseFilter.fromObject(object, callback);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Sepia2 filter class
   * @class fabric.Image.filters.Sepia2
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Sepia2();
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Sepia2 = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Sepia2.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Sepia2',

    /**
     * Applies filter to canvas element
     * @memberOf fabric.Image.filters.Sepia.prototype
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          iLen = data.length, i, r, g, b;

      for (i = 0; i < iLen; i += 4) {
        r = data[i];
        g = data[i + 1];
        b = data[i + 2];
```

```
        data[i] = (r * 0.393 + g * 0.769 + b * 0.189 ) / 1.351;
        data[i + 1] = (r * 0.349 + g * 0.686 + b * 0.168 ) / 1.203;
        data[i + 2] = (r * 0.272 + g * 0.534 + b * 0.131 ) / 2.140;
      }

      context.putImageData(imageData, 0, 0);
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Sepia2} Instance of
fabric.Image.filters.Sepia2
   */
  fabric.Image.filters.Sepia2.fromObject = function(object, callback) {
    object = object || { };
    object.type = 'Sepia2';
    return new fabric.Image.filters.BaseFilter.fromObject(object, callback);
  };

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Tint filter class
   * Adapted from <a
href="https://github.com/mezzoblue/PaintbrushJS">https://github.com/mezzoblue/
PaintbrushJS</a>
   * @class fabric.Image.filters.Tint
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Tint#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example <caption>Tint filter with hex color and opacity</caption>
   * var filter = new fabric.Image.filters.Tint({
   *   color: '#3513B0',
   *   opacity: 0.5
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   * @example <caption>Tint filter with rgba color</caption>
   * var filter = new fabric.Image.filters.Tint({
   *   color: 'rgba(53, 21, 176, 0.5)'
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Tint = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Tint.prototype */ {
```

```
    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Tint',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Tint.prototype
     * @param {Object} [options] Options object
     * @param {String} [options.color=#000000] Color to tint the image with
     * @param {Number} [options.opacity] Opacity value that controls the tint
effect's transparency (0..1)
     */
    initialize: function(options) {
      options = options || { };

      this.color = options.color || '#000000';
      this.opacity = typeof options.opacity !== 'undefined'
                     ? options.opacity
                     : new fabric.Color(this.color).getAlpha();
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          iLen = data.length, i,
          tintR, tintG, tintB,
          r, g, b, alpha1,
          source;

      source = new fabric.Color(this.color).getSource();

      tintR = source[0] * this.opacity;
      tintG = source[1] * this.opacity;
      tintB = source[2] * this.opacity;

      alpha1 = 1 - this.opacity;

      for (i = 0; i < iLen; i += 4) {
        r = data[i];
        g = data[i + 1];
        b = data[i + 2];

        // alpha compositing
        data[i] = tintR + r * alpha1;
        data[i + 1] = tintG + g * alpha1;
        data[i + 2] = tintB + b * alpha1;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
```

```
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        color: this.color,
        opacity: this.opacity
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Tint} Instance of fabric.Image.filters.Tint
   */
  fabric.Image.filters.Tint.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Multiply filter class
   * Adapted from <a href="http://www.laurenscorijn.com/articles/colormath-
basics">http://www.laurenscorijn.com/articles/colormath-basics</a>
   * @class fabric.Image.filters.Multiply
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @example <caption>Multiply filter with hex color</caption>
   * var filter = new fabric.Image.filters.Multiply({
   *   color: '#F0F'
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   * @example <caption>Multiply filter with rgb color</caption>
   * var filter = new fabric.Image.filters.Multiply({
   *   color: 'rgb(53, 21, 176)'
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Multiply = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Multiply.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Multiply',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Multiply.prototype
     * @param {Object} [options] Options object
```

```
     * @param {String} [options.color=#000000] Color to multiply the image
pixels with
     */
    initialize: function(options) {
      options = options || { };

      this.color = options.color || '#000000';
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          iLen = data.length, i,
          source;

      source = new fabric.Color(this.color).getSource();

      for (i = 0; i < iLen; i += 4) {
        data[i] *= source[0] / 255;
        data[i + 1] *= source[1] / 255;
        data[i + 2] *= source[2] / 255;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        color: this.color
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Multiply} Instance of
fabric.Image.filters.Multiply
   */
  fabric.Image.filters.Multiply.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {
  'use strict';

  var fabric = global.fabric,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;
```

```
  /**
   * Color Blend filter class
   * @class fabric.Image.filter.Blend
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @example
   * var filter = new fabric.Image.filters.Blend({
   *  color: '#000',
   *  mode: 'multiply'
   * });
   *
   * var filter = new fabric.Image.filters.Blend({
   *  image: fabricImageObject,
   *  mode: 'multiply',
   *  alpha: 0.5
   * });

   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */

  filters.Blend = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Blend.prototype */ {
    type: 'Blend',

    initialize: function(options) {
      options = options || {};
      this.color = options.color || '#000';
      this.image = options.image || false;
      this.mode = options.mode || 'multiply';
      this.alpha = options.alpha || 1;
    },

    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          tr, tg, tb,
          r, g, b,
          _r, _g, _b,
          source,
          isImage = false;

      if (this.image) {
        // Blend images
        isImage = true;

        var _el = fabric.util.createCanvasElement();
        _el.width = this.image.width;
        _el.height = this.image.height;

        var tmpCanvas = new fabric.StaticCanvas(_el);
        tmpCanvas.add(this.image);
        var context2 =  tmpCanvas.getContext('2d');
        source = context2.getImageData(0, 0, tmpCanvas.width,
tmpCanvas.height).data;
      }
      else {
        // Blend color
        source = new fabric.Color(this.color).getSource();

        tr = source[0] * this.alpha;
        tg = source[1] * this.alpha;
```

```
    tb = source[2] * this.alpha;
  }

  for (var i = 0, len = data.length; i < len; i += 4) {

    r = data[i];
    g = data[i + 1];
    b = data[i + 2];

    if (isImage) {
      tr = source[i] * this.alpha;
      tg = source[i + 1] * this.alpha;
      tb = source[i + 2] * this.alpha;
    }

    switch (this.mode) {
      case 'multiply':
        data[i] = r * tr / 255;
        data[i + 1] = g * tg / 255;
        data[i + 2] = b * tb / 255;
        break;
      case 'screen':
        data[i] = 1 - (1 - r) * (1 - tr);
        data[i + 1] = 1 - (1 - g) * (1 - tg);
        data[i + 2] = 1 - (1 - b) * (1 - tb);
        break;
      case 'add':
        data[i] = Math.min(255, r + tr);
        data[i + 1] = Math.min(255, g + tg);
        data[i + 2] = Math.min(255, b + tb);
        break;
      case 'diff':
      case 'difference':
        data[i] = Math.abs(r - tr);
        data[i + 1] = Math.abs(g - tg);
        data[i + 2] = Math.abs(b - tb);
        break;
      case 'subtract':
        _r = r - tr;
        _g = g - tg;
        _b = b - tb;

        data[i] = (_r < 0) ? 0 : _r;
        data[i + 1] = (_g < 0) ? 0 : _g;
        data[i + 2] = (_b < 0) ? 0 : _b;
        break;
      case 'darken':
        data[i] = Math.min(r, tr);
        data[i + 1] = Math.min(g, tg);
        data[i + 2] = Math.min(b, tb);
        break;
      case 'lighten':
        data[i] = Math.max(r, tr);
        data[i + 1] = Math.max(g, tg);
        data[i + 2] = Math.max(b, tb);
        break;
    }
  }

  context.putImageData(imageData, 0, 0);
},

/**
 * Returns object representation of an instance
```

```
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return {
        color: this.color,
        image: this.image,
        mode: this.mode,
        alpha: this.alpha
      };
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Blend} Instance of fabric.Image.filters.Blend
   */
  fabric.Image.filters.Blend.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }), pow = Math.pow, floor =
Math.floor,
      sqrt = Math.sqrt, abs = Math.abs, max = Math.max, round = Math.round, sin
= Math.sin,
      ceil = Math.ceil,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Resize image filter class
   * @class fabric.Image.filters.Resize
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Resize();
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Resize = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Resize.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Resize',

    /**
     * Resize type
     * @param {String} resizeType
     * @default
     */
    resizeType: 'hermite',
```

```
/**
 * Scale factor for resizing, x axis
 * @param {Number} scaleX
 * @default
 */
scaleX: 0,

/**
 * Scale factor for resizing, y axis
 * @param {Number} scaleY
 * @default
 */
scaleY: 0,

/**
 * LanczosLobes parameter for lanczos filter
 * @param {Number} lanczosLobes
 * @default
 */
lanczosLobes: 3,

/**
 * Applies filter to canvas element
 * @memberOf fabric.Image.filters.Resize.prototype
 * @param {Object} canvasEl Canvas element to apply filter to
 * @param {Number} scaleX
 * @param {Number} scaleY
 */
applyTo: function(canvasEl, scaleX, scaleY) {
  if (scaleX === 1 && scaleY === 1) {
    return;
  }

  this.rcpScaleX = 1 / scaleX;
  this.rcpScaleY = 1 / scaleY;

  var oW = canvasEl.width, oH = canvasEl.height,
      dW = round(oW * scaleX), dH = round(oH * scaleY),
      imageData;

  if (this.resizeType === 'sliceHack') {
    imageData = this.sliceByTwo(canvasEl, oW, oH, dW, dH);
  }
  if (this.resizeType === 'hermite') {
    imageData = this.hermiteFastResize(canvasEl, oW, oH, dW, dH);
  }
  if (this.resizeType === 'bilinear') {
    imageData = this.bilinearFiltering(canvasEl, oW, oH, dW, dH);
  }
  if (this.resizeType === 'lanczos') {
    imageData = this.lanczosResize(canvasEl, oW, oH, dW, dH);
  }
  canvasEl.width = dW;
  canvasEl.height = dH;
  canvasEl.getContext('2d').putImageData(imageData, 0, 0);
},

/**
 * Filter sliceByTwo
 * @param {Object} canvasEl Canvas element to apply filter to
 * @param {Number} oW Original Width
 * @param {Number} oH Original Height
 * @param {Number} dW Destination Width
```

```
 * @param {Number} dH Destination Height
 * @returns {ImageData}
 */
sliceByTwo: function(canvasEl, oW, oH, dW, dH) {
  var context = canvasEl.getContext('2d'), imageData,
      multW = 0.5, multH = 0.5, signW = 1, signH = 1,
      doneW = false, doneH = false, stepW = oW, stepH = oH,
      tmpCanvas = fabric.util.createCanvasElement(),
      tmpCtx = tmpCanvas.getContext('2d');
  dW = floor(dW);
  dH = floor(dH);
  tmpCanvas.width = max(dW, oW);
  tmpCanvas.height = max(dH, oH);

  if (dW > oW) {
    multW = 2;
    signW = -1;
  }
  if (dH > oH) {
    multH = 2;
    signH = -1;
  }
  imageData = context.getImageData(0, 0, oW, oH);
  canvasEl.width = max(dW, oW);
  canvasEl.height = max(dH, oH);
  context.putImageData(imageData, 0, 0);

  while (!doneW || !doneH) {
    oW = stepW;
    oH = stepH;
    if (dW * signW < floor(stepW * multW * signW)) {
      stepW = floor(stepW * multW);
    }
    else {
      stepW = dW;
      doneW = true;
    }
    if (dH * signH < floor(stepH * multH * signH)) {
      stepH = floor(stepH * multH);
    }
    else {
      stepH = dH;
      doneH = true;
    }
    imageData = context.getImageData(0, 0, oW, oH);
    tmpCtx.putImageData(imageData, 0, 0);
    context.clearRect(0, 0, stepW, stepH);
    context.drawImage(tmpCanvas, 0, 0, oW, oH, 0, 0, stepW, stepH);
  }
  return context.getImageData(0, 0, dW, dH);
},

/**
 * Filter lanczosResize
 * @param {Object} canvasEl Canvas element to apply filter to
 * @param {Number} oW Original Width
 * @param {Number} oH Original Height
 * @param {Number} dW Destination Width
 * @param {Number} dH Destination Height
 * @returns {ImageData}
 */
lanczosResize: function(canvasEl, oW, oH, dW, dH) {

  function lanczosCreate(lobes) {
```

```
        return function(x) {
          if (x > lobes) {
            return 0;
          }
          x *= Math.PI;
          if (abs(x) < 1e-16) {
            return 1;
          }
          var xx = x / lobes;
          return sin(x) * sin(xx) / x / xx;
        };
      }

      function process(u) {
        var v, i, weight, idx, a, red, green,
            blue, alpha, fX, fY;
        center.x = (u + 0.5) * ratioX;
        icenter.x = floor(center.x);
        for (v = 0; v < dH; v++) {
          center.y = (v + 0.5) * ratioY;
          icenter.y = floor(center.y);
          a = 0; red = 0; green = 0; blue = 0; alpha = 0;
          for (i = icenter.x - range2X; i <= icenter.x + range2X; i++) {
            if (i < 0 || i >= oW) {
              continue;
            }
            fX = floor(1000 * abs(i - center.x));
            if (!cacheLanc[fX]) {
              cacheLanc[fX] = { };
            }
            for (var j = icenter.y - range2Y; j <= icenter.y + range2Y; j++) {
              if (j < 0 || j >= oH) {
                continue;
              }
              fY = floor(1000 * abs(j - center.y));
              if (!cacheLanc[fX][fY]) {
                cacheLanc[fX][fY] = lanczos(sqrt(pow(fX * rcpRatioX, 2) + pow(fY
* rcpRatioY, 2)) / 1000);
              }
              weight = cacheLanc[fX][fY];
              if (weight > 0) {
                idx = (j * oW + i) * 4;
                a += weight;
                red += weight * srcData[idx];
                green += weight * srcData[idx + 1];
                blue += weight * srcData[idx + 2];
                alpha += weight * srcData[idx + 3];
              }
            }
          }
          idx = (v * dW + u) * 4;
          destData[idx] = red / a;
          destData[idx + 1] = green / a;
          destData[idx + 2] = blue / a;
          destData[idx + 3] = alpha / a;
        }

        if (++u < dW) {
          return process(u);
        }
        else {
          return destImg;
        }
      }
```

```
  var context = canvasEl.getContext('2d'),
      srcImg = context.getImageData(0, 0, oW, oH),
      destImg = context.getImageData(0, 0, dW, dH),
      srcData = srcImg.data, destData = destImg.data,
      lanczos = lanczosCreate(this.lanczosLobes),
      ratioX = this.rcpScaleX, ratioY = this.rcpScaleY,
      rcpRatioX = 2 / this.rcpScaleX, rcpRatioY = 2 / this.rcpScaleY,
      range2X = ceil(ratioX * this.lanczosLobes / 2),
      range2Y = ceil(ratioY * this.lanczosLobes / 2),
      cacheLanc = { }, center = { }, icenter = { };

  return process(0);
},

/**
 * bilinearFiltering
 * @param {Object} canvasEl Canvas element to apply filter to
 * @param {Number} oW Original Width
 * @param {Number} oH Original Height
 * @param {Number} dW Destination Width
 * @param {Number} dH Destination Height
 * @returns {ImageData}
 */
bilinearFiltering: function(canvasEl, oW, oH, dW, dH) {
  var a, b, c, d, x, y, i, j, xDiff, yDiff, chnl,
      color, offset = 0, origPix, ratioX = this.rcpScaleX,
      ratioY = this.rcpScaleY, context = canvasEl.getContext('2d'),
      w4 = 4 * (oW - 1), img = context.getImageData(0, 0, oW, oH),
      pixels = img.data, destImage = context.getImageData(0, 0, dW, dH),
      destPixels = destImage.data;
  for (i = 0; i < dH; i++) {
    for (j = 0; j < dW; j++) {
      x = floor(ratioX * j);
      y = floor(ratioY * i);
      xDiff = ratioX * j - x;
      yDiff = ratioY * i - y;
      origPix = 4 * (y * oW + x);

      for (chnl = 0; chnl < 4; chnl++) {
        a = pixels[origPix + chnl];
        b = pixels[origPix + 4 + chnl];
        c = pixels[origPix + w4 + chnl];
        d = pixels[origPix + w4 + 4 + chnl];
        color = a * (1 - xDiff) * (1 - yDiff) + b * xDiff * (1 - yDiff) +
                c * yDiff * (1 - xDiff) + d * xDiff * yDiff;
        destPixels[offset++] = color;
      }
    }
  }
  return destImage;
},

/**
 * hermiteFastResize
 * @param {Object} canvasEl Canvas element to apply filter to
 * @param {Number} oW Original Width
 * @param {Number} oH Original Height
 * @param {Number} dW Destination Width
 * @param {Number} dH Destination Height
 * @returns {ImageData}
 */
hermiteFastResize: function(canvasEl, oW, oH, dW, dH) {
  var ratioW = this.rcpScaleX, ratioH = this.rcpScaleY,
```

```
        ratioWHalf = ceil(ratioW / 2),
        ratioHHalf = ceil(ratioH / 2),
        context = canvasEl.getContext('2d'),
        img = context.getImageData(0, 0, oW, oH), data = img.data,
        img2 = context.getImageData(0, 0, dW, dH), data2 = img2.data;
    for (var j = 0; j < dH; j++) {
      for (var i = 0; i < dW; i++) {
        var x2 = (i + j * dW) * 4, weight = 0, weights = 0, weightsAlpha = 0,
            gxR = 0, gxG = 0, gxB = 0, gxA = 0, centerY = (j + 0.5) * ratioH;
        for (var yy = floor(j * ratioH); yy < (j + 1) * ratioH; yy++) {
          var dy = abs(centerY - (yy + 0.5)) / ratioHHalf,
              centerX = (i + 0.5) * ratioW, w0 = dy * dy;
          for (var xx = floor(i * ratioW); xx < (i + 1) * ratioW; xx++) {
            var dx = abs(centerX - (xx + 0.5)) / ratioWHalf,
                w = sqrt(w0 + dx * dx);
            /* eslint-disable max-depth */
            if (w > 1 && w < -1) {
              continue;
            }
            //hermite filter
            weight = 2 * w * w * w - 3 * w * w + 1;
            if (weight > 0) {
              dx = 4 * (xx + yy * oW);
              //alpha
              gxA += weight * data[dx + 3];
              weightsAlpha += weight;
              //colors
              if (data[dx + 3] < 255) {
                weight = weight * data[dx + 3] / 250;
              }
              gxR += weight * data[dx];
              gxG += weight * data[dx + 1];
              gxB += weight * data[dx + 2];
              weights += weight;
            }
            /* eslint-enable max-depth */
          }
        }
        data2[x2] = gxR / weights;
        data2[x2 + 1] = gxG / weights;
        data2[x2 + 2] = gxB / weights;
        data2[x2 + 3] = gxA / weightsAlpha;
      }
    }
    return img2;
  },

  /**
   * Returns object representation of an instance
   * @return {Object} Object representation of an instance
   */
  toObject: function() {
    return {
      type: this.type,
      scaleX: this.scaleX,
      scaleY: this.scaleY,
      resizeType: this.resizeType,
      lanczosLobes: this.lanczosLobes
    };
  }
});

/**
 * Returns filter instance from an object representation
```

```
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Resize} Instance of
fabric.Image.filters.Resize
   */
  fabric.Image.filters.Resize.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Color Matrix filter class
   * @class fabric.Image.filters.ColorMatrix
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.ColorMatrix#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @see {@Link http://www.webwasp.co.uk/tutorials/219/Color_Matrix_Filter.php}
   * @see {@Link http://phoboslab.org/log/2013/11/fast-image-filters-with-webgl}
   * @example <caption>Kodachrome filter</caption>
   * var filter = new fabric.Image.filters.ColorMatrix({
   *  matrix: [
       1.1285582396593525, -0.3967382283601348, -0.03992559172921793, 0,
63.72958762196502,
       -0.16404339962244616, 1.0835251566291304, -0.05498805115633132, 0,
24.732407896706203,
       -0.16786010706155763, -0.5603416277695248, 1.6014850761964943, 0,
35.62982807460946,
       0, 0, 0, 1, 0
      ]
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.ColorMatrix = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.ColorMatrix.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'ColorMatrix',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.ColorMatrix.prototype
     * @param {Object} [options] Options object
     * @param {Array} [options.matrix] Color Matrix to modify the image data
with
     */
    initialize: function( options ) {
```

```
      options || ( options = {} );
      this.matrix = options.matrix || [
        1, 0, 0, 0, 0,
        0, 1, 0, 0, 0,
        0, 0, 1, 0, 0,
        0, 0, 0, 1, 0
      ];
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function( canvasEl ) {
      var context = canvasEl.getContext( '2d' ),
          imageData = context.getImageData( 0, 0, canvasEl.width,
canvasEl.height ),
          data = imageData.data,
          iLen = data.length,
          i,
          r,
          g,
          b,
          a,
          m = this.matrix;

      for ( i = 0; i < iLen; i += 4 ) {
        r = data[ i ];
        g = data[ i + 1 ];
        b = data[ i + 2 ];
        a = data[ i + 3 ];

        data[ i ] = r * m[ 0 ] + g * m[ 1 ] + b * m[ 2 ] + a * m[ 3 ] + m[ 4 ];
        data[ i + 1 ] = r * m[ 5 ] + g * m[ 6 ] + b * m[ 7 ] + a * m[ 8 ] + m[ 9
];
        data[ i + 2 ] = r * m[ 10 ] + g * m[ 11 ] + b * m[ 12 ] + a * m[ 13 ] +
m[ 14 ];
        data[ i + 3 ] = r * m[ 15 ] + g * m[ 16 ] + b * m[ 17 ] + a * m[ 18 ] +
m[ 19 ];
      }

      context.putImageData( imageData, 0, 0 );
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        type: this.type,
        matrix: this.matrix
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] function to invoke after filter creation
   * @return {fabric.Image.filters.ColorMatrix} Instance of
fabric.Image.filters.ColorMatrix
   */
```

```
  fabric.Image.filters.ColorMatrix.fromObject =
fabric.Image.filters.BaseFilter.fromObject;
})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Contrast filter class
   * @class fabric.Image.filters.Contrast
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Contrast#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Contrast({
   *   contrast: 40
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Contrast = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Contrast.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
     */
    type: 'Contrast',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Contrast.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.contrast=0] Value to contrast the image up (-
255...255)
     */
    initialize: function(options) {
      options = options || { };
      this.contrast = options.contrast || 0;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          contrastF = 259 * (this.contrast + 255) / (255 * (259 -
this.contrast));

      for (var i = 0, len = data.length; i < len; i += 4) {
```

```
        data[i] = contrastF * (data[i] - 128) + 128;
        data[i + 1] = contrastF * (data[i + 1] - 128) + 128;
        data[i + 2] = contrastF * (data[i + 2] - 128) + 128;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        contrast: this.contrast
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Contrast} Instance of
fabric.Image.filters.Contrast
   */
  fabric.Image.filters.Contrast.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';

  var fabric  = global.fabric || (global.fabric = { }),
      extend = fabric.util.object.extend,
      filters = fabric.Image.filters,
      createClass = fabric.util.createClass;

  /**
   * Saturate filter class
   * @class fabric.Image.filters.Saturate
   * @memberOf fabric.Image.filters
   * @extends fabric.Image.filters.BaseFilter
   * @see {@link fabric.Image.filters.Saturate#initialize} for constructor
definition
   * @see {@link http://fabricjs.com/image-filters|ImageFilters demo}
   * @example
   * var filter = new fabric.Image.filters.Saturate({
   *   saturate: 100
   * });
   * object.filters.push(filter);
   * object.applyFilters(canvas.renderAll.bind(canvas));
   */
  filters.Saturate = createClass(filters.BaseFilter, /** @lends
fabric.Image.filters.Saturate.prototype */ {

    /**
     * Filter type
     * @param {String} type
     * @default
```

```
     */
    type: 'Saturate',

    /**
     * Constructor
     * @memberOf fabric.Image.filters.Saturate.prototype
     * @param {Object} [options] Options object
     * @param {Number} [options.saturate=0] Value to saturate the image (-
100...100)
     */
    initialize: function(options) {
      options = options || { };
      this.saturate = options.saturate || 0;
    },

    /**
     * Applies filter to canvas element
     * @param {Object} canvasEl Canvas element to apply filter to
     */
    applyTo: function(canvasEl) {
      var context = canvasEl.getContext('2d'),
          imageData = context.getImageData(0, 0, canvasEl.width,
canvasEl.height),
          data = imageData.data,
          max, adjust = -this.saturate * 0.01;

      for (var i = 0, len = data.length; i < len; i += 4) {
        max = Math.max(data[i], data[i + 1], data[i + 2]);
        data[i] += max !== data[i] ? (max - data[i]) * adjust : 0;
        data[i + 1] += max !== data[i + 1] ? (max - data[i + 1]) * adjust : 0;
        data[i + 2] += max !== data[i + 2] ? (max - data[i + 2]) * adjust : 0;
      }

      context.putImageData(imageData, 0, 0);
    },

    /**
     * Returns object representation of an instance
     * @return {Object} Object representation of an instance
     */
    toObject: function() {
      return extend(this.callSuper('toObject'), {
        saturate: this.saturate
      });
    }
  });

  /**
   * Returns filter instance from an object representation
   * @static
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] to be invoked after filter creation
   * @return {fabric.Image.filters.Saturate} Instance of
fabric.Image.filters.Saturate
   */
  fabric.Image.filters.Saturate.fromObject =
fabric.Image.filters.BaseFilter.fromObject;

})(typeof exports !== 'undefined' ? exports : this);


(function(global) {

  'use strict';
```

```
  var fabric = global.fabric || (global.fabric = { }),
      toFixed = fabric.util.toFixed,
      NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS,
      MIN_TEXT_WIDTH = 2;

  if (fabric.Text) {
    fabric.warn('fabric.Text is already defined');
    return;
  }

  var stateProperties = fabric.Object.prototype.stateProperties.concat();
  stateProperties.push(
    'fontFamily',
    'fontWeight',
    'fontSize',
    'text',
    'textDecoration',
    'textAlign',
    'fontStyle',
    'lineHeight',
    'textBackgroundColor',
    'charSpacing'
  );

  var cacheProperties = fabric.Object.prototype.cacheProperties.concat();
  cacheProperties.push(
    'fontFamily',
    'fontWeight',
    'fontSize',
    'text',
    'textDecoration',
    'textAlign',
    'fontStyle',
    'lineHeight',
    'textBackgroundColor',
    'charSpacing',
    'styles'
  );
  /**
   * Text class
   * @class fabric.Text
   * @extends fabric.Object
   * @return {fabric.Text} thisArg
   * @tutorial {@link http://fabricjs.com/fabric-intro-part-2#text}
   * @see {@link fabric.Text#initialize} for constructor definition
   */
  fabric.Text = fabric.util.createClass(fabric.Object, /** @lends
fabric.Text.prototype */ {

    /**
     * Properties which when set cause object to change dimensions
     * @type Object
     * @private
     */
    _dimensionAffectingProps: [
      'fontSize',
      'fontWeight',
      'fontFamily',
      'fontStyle',
      'lineHeight',
      'text',
      'charSpacing',
      'textAlign'
```

```
    ],

    /**
     * @private
     */
    _reNewline: /\r?\n/,

    /**
     * Use this regular expression to filter for whitespace that is not a new
line.
     * Mostly used when text is 'justify' aligned.
     * @private
     */
    _reSpacesAndTabs: /[ \t\r]+/g,

    /**
     * Retrieves object's fontSize
     * @method getFontSize
     * @memberOf fabric.Text.prototype
     * @return {String} Font size (in pixels)
     */

    /**
     * Sets object's fontSize
     * Does not update the object .width and .height,
     * call ._initDimensions() to update the values.
     * @method setFontSize
     * @memberOf fabric.Text.prototype
     * @param {Number} fontSize Font size (in pixels)
     * @return {fabric.Text}
     * @chainable
     */

    /**
     * Retrieves object's fontWeight
     * @method getFontWeight
     * @memberOf fabric.Text.prototype
     * @return {(String|Number)} Font weight
     */

    /**
     * Sets object's fontWeight
     * Does not update the object .width and .height,
     * call ._initDimensions() to update the values.
     * @method setFontWeight
     * @memberOf fabric.Text.prototype
     * @param {(Number|String)} fontWeight Font weight
     * @return {fabric.Text}
     * @chainable
     */

    /**
     * Retrieves object's fontFamily
     * @method getFontFamily
     * @memberOf fabric.Text.prototype
     * @return {String} Font family
     */

    /**
     * Sets object's fontFamily
     * Does not update the object .width and .height,
     * call ._initDimensions() to update the values.
     * @method setFontFamily
     * @memberOf fabric.Text.prototype
```

```
 * @param {String} fontFamily Font family
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's text
 * @method getText
 * @memberOf fabric.Text.prototype
 * @return {String} text
 */

/**
 * Sets object's text
 * Does not update the object .width and .height,
 * call ._initDimensions() to update the values.
 * @method setText
 * @memberOf fabric.Text.prototype
 * @param {String} text Text
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's textDecoration
 * @method getTextDecoration
 * @memberOf fabric.Text.prototype
 * @return {String} Text decoration
 */

/**
 * Sets object's textDecoration
 * @method setTextDecoration
 * @memberOf fabric.Text.prototype
 * @param {String} textDecoration Text decoration
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's fontStyle
 * @method getFontStyle
 * @memberOf fabric.Text.prototype
 * @return {String} Font style
 */

/**
 * Sets object's fontStyle
 * Does not update the object .width and .height,
 * call ._initDimensions() to update the values.
 * @method setFontStyle
 * @memberOf fabric.Text.prototype
 * @param {String} fontStyle Font style
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's lineHeight
 * @method getLineHeight
 * @memberOf fabric.Text.prototype
 * @return {Number} Line height
 */
```

```
/**
 * Sets object's lineHeight
 * @method setLineHeight
 * @memberOf fabric.Text.prototype
 * @param {Number} lineHeight Line height
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's textAlign
 * @method getTextAlign
 * @memberOf fabric.Text.prototype
 * @return {String} Text alignment
 */

/**
 * Sets object's textAlign
 * @method setTextAlign
 * @memberOf fabric.Text.prototype
 * @param {String} textAlign Text alignment
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Retrieves object's textBackgroundColor
 * @method getTextBackgroundColor
 * @memberOf fabric.Text.prototype
 * @return {String} Text background color
 */

/**
 * Sets object's textBackgroundColor
 * @method setTextBackgroundColor
 * @memberOf fabric.Text.prototype
 * @param {String} textBackgroundColor Text background color
 * @return {fabric.Text}
 * @chainable
 */

/**
 * Type of an object
 * @type String
 * @default
 */
type:                 'text',

/**
 * Font size (in pixels)
 * @type Number
 * @default
 */
fontSize:             40,

/**
 * Font weight (e.g. bold, normal, 400, 600, 800)
 * @type {(Number|String)}
 * @default
 */
fontWeight:           'normal',

/**
 * Font family
```

```
     * @type String
     * @default
     */
    fontFamily:           'Times New Roman',

    /**
     * Text decoration Possible values: "", "underline", "overline" or "line-
through".
     * @type String
     * @default
     */
    textDecoration:       '',

    /**
     * Text alignment. Possible values: "left", "center", "right" or "justify".
     * @type String
     * @default
     */
    textAlign:            'left',

    /**
     * Font style . Possible values: "", "normal", "italic" or "oblique".
     * @type String
     * @default
     */
    fontStyle:            '',

    /**
     * Line height
     * @type Number
     * @default
     */
    lineHeight:           1.16,

    /**
     * Background color of text lines
     * @type String
     * @default
     */
    textBackgroundColor:  '',

    /**
     * List of properties to consider when checking if
     * state of an object is changed ({@link fabric.Object#hasStateChanged})
     * as well as for history (undo/redo) purposes
     * @type Array
     */
    stateProperties:      stateProperties,

    /**
     * List of properties to consider when checking if cache needs refresh
     * @type Array
     */
    cacheProperties:      cacheProperties,

    /**
     * When defined, an object is rendered via stroke and this property
specifies its color.
     * <b>Backwards incompatibility note:</b> This property was named
"strokeStyle" until v1.1.6
     * @type String
     * @default
     */
    stroke:               null,
```

```
    /**
     * Shadow object representing shadow of this shape.
     * <b>Backwards incompatibility note:</b> This property was named
"textShadow" (String) until v1.2.11
     * @type fabric.Shadow
     * @default
     */
    shadow:               null,

    /**
     * @private
     */
    _fontSizeFraction: 0.25,

    /**
     * Text Line proportion to font Size (in pixels)
     * @type Number
     * @default
     */
    _fontSizeMult:             1.13,

    /**
     * additional space between characters
     * expressed in thousands of em unit
     * @type Number
     * @default
     */
    charSpacing:             0,

    /**
     * Constructor
     * @param {String} text Text string
     * @param {Object} [options] Options object
     * @return {fabric.Text} thisArg
     */
    initialize: function(text, options) {
      options = options || { };
      this.text = text;
      this.__skipDimension = true;
      this.callSuper('initialize', options);
      this.__skipDimension = false;
      this._initDimensions();
      this.setCoords();
      this.setupState({ propertySet: '_dimensionAffectingProps' });
    },

    /**
     * Initialize text dimensions. Render all text on given context
     * or on a offscreen canvas to get the text width with measureText.
     * Updates this.width and this.height with the proper values.
     * Does not return dimensions.
     * @param {CanvasRenderingContext2D} [ctx] Context to render on
     * @private
     */
    _initDimensions: function(ctx) {
      if (this.__skipDimension) {
        return;
      }
      if (!ctx) {
        ctx = fabric.util.createCanvasElement().getContext('2d');
        this._setTextStyles(ctx);
      }
      this._textLines = this._splitTextIntoLines();
```

```
      this._clearCache();
      this.width = this._getTextWidth(ctx) || this.cursorWidth ||
MIN_TEXT_WIDTH;
      this.height = this._getTextHeight(ctx);
    },

    /**
     * Returns string representation of an instance
     * @return {String} String representation of text object
     */
    toString: function() {
      return '#<fabric.Text (' + this.complexity() +
        '): { "text": "' + this.text + '", "fontFamily": "' + this.fontFamily +
'" }>';
    },

    /**
     * Return the dimension and the zoom level needed to create a cache canvas
     * big enough to host the object to be cached.
     * @private
     * @return {Object}.width width of canvas
     * @return {Object}.height height of canvas
     * @return {Object}.zoomX zoomX zoom value to unscale the canvas before
drawing cache
     * @return {Object}.zoomY zoomY zoom value to unscale the canvas before
drawing cache
     */
    _getCacheCanvasDimensions: function() {
      var dim = this.callSuper('_getCacheCanvasDimensions');
      var fontSize = this.fontSize;
      dim.width += fontSize * dim.zoomX;
      dim.height += fontSize * dim.zoomY;
      return dim;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _render: function(ctx) {
      this._setTextStyles(ctx);
      if (this.group && this.group.type === 'path-group') {
        ctx.translate(this.left, this.top);
      }
      this._renderTextLinesBackground(ctx);
      this._renderText(ctx);
      this._renderTextDecoration(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderText: function(ctx) {
      this._renderTextFill(ctx);
      this._renderTextStroke(ctx);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _setTextStyles: function(ctx) {
      ctx.textBaseline = 'alphabetic';
```

```
      ctx.font = this._getFontDeclaration();
    },

    /**
     * @private
     * @return {Number} Height of fabric.Text object
     */
    _getTextHeight: function() {
      return this._getHeightOfSingleLine() + (this._textLines.length - 1) *
this._getHeightOfLine();
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @return {Number} Maximum width of fabric.Text object
     */
    _getTextWidth: function(ctx) {
      var maxWidth = this._getLineWidth(ctx, 0);

      for (var i = 1, len = this._textLines.length; i < len; i++) {
        var currentLineWidth = this._getLineWidth(ctx, i);
        if (currentLineWidth > maxWidth) {
          maxWidth = currentLineWidth;
        }
      }
      return maxWidth;
    },

    /**
     * @private
     * @param {String} method Method name ("fillText" or "strokeText")
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} chars Chars to render
     * @param {Number} left Left position of text
     * @param {Number} top Top position of text
     */
    _renderChars: function(method, ctx, chars, left, top) {
      // remove Text word from method var
      var shortM = method.slice(0, -4), _char, width;
      if (this[shortM].toLive) {
        var offsetX = -this.width / 2 + this[shortM].offsetX || 0,
            offsetY = -this.height / 2 + this[shortM].offsetY || 0;
        ctx.save();
        ctx.translate(offsetX, offsetY);
        left -= offsetX;
        top -= offsetY;
      }
      if (this.charSpacing !== 0) {
        var additionalSpace = this._getWidthOfCharSpacing();
        chars = chars.split('');
        for (var i = 0, len = chars.length; i < len; i++) {
          _char = chars[i];
          width = ctx.measureText(_char).width + additionalSpace;
          ctx[method](_char, left, top);
          left += width > 0 ? width : 0;
        }
      }
      else {
        ctx[method](chars, left, top);
      }
      this[shortM].toLive && ctx.restore();
    },
```

```
    /**
     * @private
     * @param {String} method Method name ("fillText" or "strokeText")
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} line Text to render
     * @param {Number} left Left position of text
     * @param {Number} top Top position of text
     * @param {Number} lineIndex Index of a line in a text
     */
    _renderTextLine: function(method, ctx, line, left, top, lineIndex) {
      // lift the line by quarter of fontSize
      top -= this.fontSize * this._fontSizeFraction;

      // short-circuit
      var lineWidth = this._getLineWidth(ctx, lineIndex);
      if (this.textAlign !== 'justify' || this.width < lineWidth) {
        this._renderChars(method, ctx, line, left, top, lineIndex);
        return;
      }

      // stretch the line
      var words = line.split(/\s+/),
          charOffset = 0,
          wordsWidth = this._getWidthOfWords(ctx, words.join(' '), lineIndex,
0),
          widthDiff = this.width - wordsWidth,
          numSpaces = words.length - 1,
          spaceWidth = numSpaces > 0 ? widthDiff / numSpaces : 0,
          leftOffset = 0, word;

      for (var i = 0, len = words.length; i < len; i++) {
        while (line[charOffset] === ' ' && charOffset < line.length) {
          charOffset++;
        }
        word = words[i];
        this._renderChars(method, ctx, word, left + leftOffset, top, lineIndex,
charOffset);
        leftOffset += this._getWidthOfWords(ctx, word, lineIndex, charOffset) +
spaceWidth;
        charOffset += word.length;
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} word
     */
    _getWidthOfWords: function (ctx, word) {
      var width = ctx.measureText(word).width, charCount, additionalSpace;
      if (this.charSpacing !== 0) {
        charCount = word.split('').length;
        additionalSpace = charCount * this._getWidthOfCharSpacing();
        width += additionalSpace;
      }
      return width > 0 ? width : 0;
    },

    /**
     * @private
     * @return {Number} Left offset
     */
    _getLeftOffset: function() {
      return -this.width / 2;
```

```
    },

    /**
     * @private
     * @return {Number} Top offset
     */
    _getTopOffset: function() {
      return -this.height / 2;
    },

    /**
     * Returns true because text has no style
     */
    isEmptyStyles: function() {
      return true;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} method Method name ("fillText" or "strokeText")
     */
    _renderTextCommon: function(ctx, method) {

      var lineHeights = 0, left = this._getLeftOffset(), top =
this._getTopOffset();

      for (var i = 0, len = this._textLines.length; i < len; i++) {
        var heightOfLine = this._getHeightOfLine(ctx, i),
            maxHeight = heightOfLine / this.lineHeight,
            lineWidth = this._getLineWidth(ctx, i),
            leftOffset = this._getLineLeftOffset(lineWidth);
        this._renderTextLine(
          method,
          ctx,
          this._textLines[i],
          left + leftOffset,
          top + lineHeights + maxHeight,
          i
        );
        lineHeights += heightOfLine;
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderTextFill: function(ctx) {
      if (!this.fill && this.isEmptyStyles()) {
        return;
      }

      this._renderTextCommon(ctx, 'fillText');
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderTextStroke: function(ctx) {
      if ((!this.stroke || this.strokeWidth === 0) && this.isEmptyStyles()) {
        return;
      }
```

```
  if (this.shadow && !this.shadow.affectStroke) {
    this._removeShadow(ctx);
  }

  ctx.save();
  this._setLineDash(ctx, this.strokeDashArray);
  ctx.beginPath();
  this._renderTextCommon(ctx, 'strokeText');
  ctx.closePath();
  ctx.restore();
},

/**
 * @private
 * @return {Number} height of line
 */
_getHeightOfLine: function() {
  return this._getHeightOfSingleLine() * this.lineHeight;
},

/**
 * @private
 * @return {Number} height of line without lineHeight
 */
_getHeightOfSingleLine: function() {
  return this.fontSize * this._fontSizeMult;
},

/**
 * @private
 * @param {CanvasRenderingContext2D} ctx Context to render on
 */
_renderTextLinesBackground: function(ctx) {
  if (!this.textBackgroundColor) {
    return;
  }
  var lineTopOffset = 0, heightOfLine,
      lineWidth, lineLeftOffset, originalFill = ctx.fillStyle;

  ctx.fillStyle = this.textBackgroundColor;
  for (var i = 0, len = this._textLines.length; i < len; i++) {
    heightOfLine = this._getHeightOfLine(ctx, i);
    lineWidth = this._getLineWidth(ctx, i);
    if (lineWidth > 0) {
      lineLeftOffset = this._getLineLeftOffset(lineWidth);
      ctx.fillRect(
        this._getLeftOffset() + lineLeftOffset,
        this._getTopOffset() + lineTopOffset,
        lineWidth,
        heightOfLine / this.lineHeight
      );
    }
    lineTopOffset += heightOfLine;
  }
  ctx.fillStyle = originalFill;
  // if there is text background color no
  // other shadows should be casted
  this._removeShadow(ctx);
},

/**
 * @private
 * @param {Number} lineWidth Width of text line
```

```
     * @return {Number} Line left offset
     */
    _getLineLeftOffset: function(lineWidth) {
      if (this.textAlign === 'center') {
        return (this.width - lineWidth) / 2;
      }
      if (this.textAlign === 'right') {
        return this.width - lineWidth;
      }
      return 0;
    },

    /**
     * @private
     */
    _clearCache: function() {
      this.__lineWidths = [];
      this.__lineHeights = [];
    },

    /**
     * @private
     */
    _shouldClearDimensionCache: function() {
      var shouldClear = this._forceClearCache;
      shouldClear || (shouldClear =
this.hasStateChanged('_dimensionAffectingProps'));
      if (shouldClear) {
        this.saveState({ propertySet: '_dimensionAffectingProps' });
        this.dirty = true;
      }
      return shouldClear;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex line number
     * @return {Number} Line width
     */
    _getLineWidth: function(ctx, lineIndex) {
      if (this.__lineWidths[lineIndex]) {
        return this.__lineWidths[lineIndex] === -1 ? this.width :
this.__lineWidths[lineIndex];
      }

      var width, wordCount, line = this._textLines[lineIndex];

      if (line === '') {
        width = 0;
      }
      else {
        width = this._measureLine(ctx, lineIndex);
      }
      this.__lineWidths[lineIndex] = width;

      if (width && this.textAlign === 'justify') {
        wordCount = line.split(/\s+/);
        if (wordCount.length > 1) {
          this.__lineWidths[lineIndex] = -1;
        }
      }
      return width;
    },
```

```
    _getWidthOfCharSpacing: function() {
      if (this.charSpacing !== 0) {
        return this.fontSize * this.charSpacing / 1000;
      }
      return 0;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex line number
     * @return {Number} Line width
     */
    _measureLine: function(ctx, lineIndex) {
      var line = this._textLines[lineIndex],
          width = ctx.measureText(line).width,
          additionalSpace = 0, charCount, finalWidth;
      if (this.charSpacing !== 0) {
        charCount = line.split('').length;
        additionalSpace = (charCount - 1) * this._getWidthOfCharSpacing();
      }
      finalWidth = width + additionalSpace;
      return finalWidth > 0 ? finalWidth : 0;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderTextDecoration: function(ctx) {
      if (!this.textDecoration) {
        return;
      }
      var halfOfVerticalBox = this.height / 2,
          _this = this, offsets = [];

      /** @ignore */
      function renderLinesAtOffset(offsets) {
        var i, lineHeight = 0, len, j, oLen, lineWidth,
            lineLeftOffset, heightOfLine;

        for (i = 0, len = _this._textLines.length; i < len; i++) {

          lineWidth = _this._getLineWidth(ctx, i);
          lineLeftOffset = _this._getLineLeftOffset(lineWidth);
          heightOfLine = _this._getHeightOfLine(ctx, i);

          for (j = 0, oLen = offsets.length; j < oLen; j++) {
            ctx.fillRect(
              _this._getLeftOffset() + lineLeftOffset,
              lineHeight + (_this._fontSizeMult - 1 + offsets[j] ) *
_this.fontSize - halfOfVerticalBox,
              lineWidth,
              _this.fontSize / 15);
          }
          lineHeight += heightOfLine;
        }
      }

      if (this.textDecoration.indexOf('underline') > -1) {
        offsets.push(0.85); // 1 - 3/16
      }
      if (this.textDecoration.indexOf('line-through') > -1) {
```

```
        offsets.push(0.43);
      }
      if (this.textDecoration.indexOf('overline') > -1) {
        offsets.push(-0.12);
      }
      if (offsets.length > 0) {
        renderLinesAtOffset(offsets);
      }
    },

    /**
     * return font declaration string for canvas context
     * @returns {String} font declaration formatted for canvas context.
     */
    _getFontDeclaration: function() {
      return [
        // node-canvas needs "weight style", while browsers need "style weight"
        (fabric.isLikelyNode ? this.fontWeight : this.fontStyle),
        (fabric.isLikelyNode ? this.fontStyle : this.fontWeight),
        this.fontSize + 'px',
        (fabric.isLikelyNode ? ('"' + this.fontFamily + '"') : this.fontFamily)
      ].join(' ');
    },

    /**
     * Renders text instance on a specified context
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    render: function(ctx, noTransform) {
      // do not render if object is not visible
      if (!this.visible) {
        return;
      }
      if (this.canvas && this.canvas.skipOffscreen && !this.group && !
this.isOnScreen()) {
        return;
      }
      if (this._shouldClearDimensionCache()) {
        this._setTextStyles(ctx);
        this._initDimensions(ctx);
      }
      this.callSuper('render', ctx, noTransform);
    },

    /**
     * Returns the text as an array of lines.
     * @returns {Array} Lines in the text
     */
    _splitTextIntoLines: function() {
      return this.text.split(this._reNewline);
    },

    /**
     * Returns object representation of an instance
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} Object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      var additionalProperties = [
        'text',
        'fontSize',
        'fontWeight',
```

```
        'fontFamily',
        'fontStyle',
        'lineHeight',
        'textDecoration',
        'textAlign',
        'textBackgroundColor',
        'charSpacing'
      ].concat(propertiesToInclude);
      return this.callSuper('toObject', additionalProperties);
    },

    /* _TO_SVG_START_ */
    /**
     * Returns SVG representation of an instance
     * @param {Function} [reviver] Method for further parsing of svg
representation.
     * @return {String} svg representation of an instance
     */
    toSVG: function(reviver) {
      if (!this.ctx) {
        this.ctx = fabric.util.createCanvasElement().getContext('2d');
      }
      var markup = this._createBaseSVGMarkup(),
          offsets = this._getSVGLeftTopOffsets(this.ctx),
          textAndBg = this._getSVGTextAndBg(offsets.textTop, offsets.textLeft);
      this._wrapSVGTextAndBg(markup, textAndBg);

      return reviver ? reviver(markup.join('')) : markup.join('');
    },

    /**
     * @private
     */
    _getSVGLeftTopOffsets: function(ctx) {
      var lineTop = this._getHeightOfLine(ctx, 0),
          textLeft = -this.width / 2,
          textTop = 0;

      return {
        textLeft: textLeft + (this.group && this.group.type === 'path-group' ?
this.left : 0),
        textTop: textTop + (this.group && this.group.type === 'path-group' ? -
this.top : 0),
        lineTop: lineTop
      };
    },

    /**
     * @private
     */
    _wrapSVGTextAndBg: function(markup, textAndBg) {
      var noShadow = true, filter = this.getSvgFilter(),
          style = filter === '' ? '' : ' style="' + filter + '"';

      markup.push(
        '\t<g ', this.getSvgId(), 'transform="', this.getSvgTransform(),
this.getSvgTransformMatrix(), '"',
          style, '>\n',
          textAndBg.textBgRects.join(''),
          '\t\t<text xml:space="preserve" ',
            (this.fontFamily ? 'font-family="' + this.fontFamily.replace(/"/g,
'\'') + '" ' : ''),
            (this.fontSize ? 'font-size="' + this.fontSize + '" ' : ''),
            (this.fontStyle ? 'font-style="' + this.fontStyle + '" ' : ''),
```

```
            (this.fontWeight ? 'font-weight="' + this.fontWeight + '" ' : ''),
            (this.textDecoration ? 'text-decoration="' + this.textDecoration +
'" ' : ''),
            'style="', this.getSvgStyles(noShadow), '" >\n',
            textAndBg.textSpans.join(''),
          '\t\t</text>\n',
        '\t</g>\n'
      );
    },

    getSvgStyles: function(skipShadow) {
      var svgStyle = fabric.Object.prototype.getSvgStyles.call(this,
skipShadow);
      return svgStyle + ' white-space: pre;';
    },

    /**
     * @private
     * @param {Number} textTopOffset Text top offset
     * @param {Number} textLeftOffset Text left offset
     * @return {Object}
     */
    _getSVGTextAndBg: function(textTopOffset, textLeftOffset) {
      var textSpans = [],
          textBgRects = [],
          height = 0;
      // bounding-box background
      this._setSVGBg(textBgRects);

      // text and text-background
      for (var i = 0, len = this._textLines.length; i < len; i++) {
        if (this.textBackgroundColor) {
          this._setSVGTextLineBg(textBgRects, i, textLeftOffset, textTopOffset,
height);
        }
        this._setSVGTextLineText(i, textSpans, height, textLeftOffset,
textTopOffset, textBgRects);
        height += this._getHeightOfLine(this.ctx, i);
      }

      return {
        textSpans: textSpans,
        textBgRects: textBgRects
      };
    },

    _setSVGTextLineText: function(i, textSpans, height, textLeftOffset,
textTopOffset) {
      var yPos = this.fontSize * (this._fontSizeMult - this._fontSizeFraction)
        - textTopOffset + height - this.height / 2;
      if (this.textAlign === 'justify') {
        // i call from here to do not intefere with IText
        this._setSVGTextLineJustifed(i, textSpans, yPos, textLeftOffset);
        return;
      }
      textSpans.push(
        '\t\t\t<tspan x="',
          toFixed(textLeftOffset +
this._getLineLeftOffset(this._getLineWidth(this.ctx, i)), NUM_FRACTION_DIGITS),
'" ',
          'y="',
          toFixed(yPos, NUM_FRACTION_DIGITS),
          '" ',
          // doing this on <tspan> elements since setting opacity
```

```
        // on containing <text> one doesn't work in Illustrator
        this._getFillAttributes(this.fill), '>',
        fabric.util.string.escapeXml(this._textLines[i]),
      '</tspan>\n'
    );
  },

  _setSVGTextLineJustifed: function(i, textSpans, yPos, textLeftOffset) {
    var ctx = fabric.util.createCanvasElement().getContext('2d');

    this._setTextStyles(ctx);

    var line = this._textLines[i],
        words = line.split(/\s+/),
        wordsWidth = this._getWidthOfWords(ctx, words.join('')),
        widthDiff = this.width - wordsWidth,
        numSpaces = words.length - 1,
        spaceWidth = numSpaces > 0 ? widthDiff / numSpaces : 0,
        word, attributes = this._getFillAttributes(this.fill),
        len;

    textLeftOffset += this._getLineLeftOffset(this._getLineWidth(ctx, i));

    for (i = 0, len = words.length; i < len; i++) {
      word = words[i];
      textSpans.push(
        '\t\t\t<tspan x="',
          toFixed(textLeftOffset, NUM_FRACTION_DIGITS), '" ',
          'y="',
          toFixed(yPos, NUM_FRACTION_DIGITS),
          '" ',
          // doing this on <tspan> elements since setting opacity
          // on containing <text> one doesn't work in Illustrator
          attributes, '>',
          fabric.util.string.escapeXml(word),
        '</tspan>\n'
      );
      textLeftOffset += this._getWidthOfWords(ctx, word) + spaceWidth;
    }
  },

  _setSVGTextLineBg: function(textBgRects, i, textLeftOffset, textTopOffset,
height) {
    textBgRects.push(
      '\t\t<rect ',
        this._getFillAttributes(this.textBackgroundColor),
        ' x="',
        toFixed(textLeftOffset +
this._getLineLeftOffset(this._getLineWidth(this.ctx, i)), NUM_FRACTION_DIGITS),
        '" y="',
        toFixed(height - this.height / 2, NUM_FRACTION_DIGITS),
        '" width="',
        toFixed(this._getLineWidth(this.ctx, i), NUM_FRACTION_DIGITS),
        '" height="',
        toFixed(this._getHeightOfLine(this.ctx, i) / this.lineHeight,
NUM_FRACTION_DIGITS),
      '"></rect>\n');
  },

  _setSVGBg: function(textBgRects) {
    if (this.backgroundColor) {
      textBgRects.push(
        '\t\t<rect ',
          this._getFillAttributes(this.backgroundColor),
```

```
            ' x="',
            toFixed(-this.width / 2, NUM_FRACTION_DIGITS),
            '" y="',
            toFixed(-this.height / 2, NUM_FRACTION_DIGITS),
            '" width="',
            toFixed(this.width, NUM_FRACTION_DIGITS),
            '" height="',
            toFixed(this.height, NUM_FRACTION_DIGITS),
          '"></rect>\n');
      }
    },

    /**
     * Adobe Illustrator (at least CS5) is unable to render rgba()-based fill
values
     * we work around it by "moving" alpha channel into opacity attribute and
setting fill's alpha to 1
     *
     * @private
     * @param {*} value
     * @return {String}
     */
    _getFillAttributes: function(value) {
      var fillColor = (value && typeof value === 'string') ? new
fabric.Color(value) : '';
      if (!fillColor || !fillColor.getSource() || fillColor.getAlpha() === 1) {
        return 'fill="' + value + '"';
      }
      return 'opacity="' + fillColor.getAlpha() + '" fill="' +
fillColor.setAlpha(1).toRgb() + '"';
    },
    /* _TO_SVG_END_ */

    /**
     * Sets specified property to a specified value
     * @param {String} key
     * @param {*} value
     * @return {fabric.Text} thisArg
     * @chainable
     */
    _set: function(key, value) {
      this.callSuper('_set', key, value);

      if (this._dimensionAffectingProps.indexOf(key) > -1) {
        this._initDimensions();
        this.setCoords();
      }
    },

    /**
     * Returns complexity of an instance
     * @return {Number} complexity
     */
    complexity: function() {
      return 1;
    }
  });

  /* _FROM_SVG_START_ */
  /**
   * List of attribute names to account for when parsing SVG element (used by
{@link fabric.Text.fromElement})
   * @static
   * @memberOf fabric.Text
```

```
   * @see: http://www.w3.org/TR/SVG/text.html#TextElement
   */
  fabric.Text.ATTRIBUTE_NAMES = fabric.SHARED_ATTRIBUTES.concat(
    'x y dx dy font-family font-style font-weight font-size text-decoration
text-anchor'.split(' '));

  /**
   * Default SVG font size
   * @static
   * @memberOf fabric.Text
   */
  fabric.Text.DEFAULT_SVG_FONT_SIZE = 16;

  /**
   * Returns fabric.Text instance from an SVG element (<b>not yet
implemented</b>)
   * @static
   * @memberOf fabric.Text
   * @param {SVGElement} element Element to parse
   * @param {Object} [options] Options object
   * @return {fabric.Text} Instance of fabric.Text
   */
  fabric.Text.fromElement = function(element, options) {
    if (!element) {
      return null;
    }

    var parsedAttributes = fabric.parseAttributes(element,
fabric.Text.ATTRIBUTE_NAMES);
    options = fabric.util.object.extend((options ?
fabric.util.object.clone(options) : { }), parsedAttributes);

    options.top = options.top || 0;
    options.left = options.left || 0;
    if ('dx' in parsedAttributes) {
      options.left += parsedAttributes.dx;
    }
    if ('dy' in parsedAttributes) {
      options.top += parsedAttributes.dy;
    }
    if (!('fontSize' in options)) {
      options.fontSize = fabric.Text.DEFAULT_SVG_FONT_SIZE;
    }

    if (!options.originX) {
      options.originX = 'left';
    }

    var textContent = '';

    // The XML is not properly parsed in IE9 so a workaround to get
    // textContent is through firstChild.data. Another workaround would be
    // to convert XML loaded from a file to be converted using DOMParser (same
way loadSVGFromString() does)
    if (!('textContent' in element)) {
      if ('firstChild' in element && element.firstChild !== null) {
        if ('data' in element.firstChild && element.firstChild.data !== null) {
          textContent = element.firstChild.data;
        }
      }
    }
    else {
      textContent = element.textContent;
    }
```

```
    textContent = textContent.replace(/^\s+|\s+$|\n+/g, '').replace(/\s+/g, '
');

    var text = new fabric.Text(textContent, options),
        textHeightScaleFactor = text.getHeight() / text.height,
        lineHeightDiff = (text.height + text.strokeWidth) * text.lineHeight -
text.height,
        scaledDiff = lineHeightDiff * textHeightScaleFactor,
        textHeight = text.getHeight() + scaledDiff,
        offX = 0;
    /*
      Adjust positioning:
        x/y attributes in SVG correspond to the bottom-left corner of text
bounding box
        top/left properties in Fabric correspond to center point of text
bounding box
    */
    if (text.originX === 'left') {
      offX = text.getWidth() / 2;
    }
    if (text.originX === 'right') {
      offX = -text.getWidth() / 2;
    }
    text.set({
      left: text.getLeft() + offX,
      top: text.getTop() - textHeight / 2 + text.fontSize * (0.18 +
text._fontSizeFraction) / text.lineHeight /* 0.3 is the old lineHeight */
    });

    return text;
  };
  /* _FROM_SVG_END_ */

  /**
   * Returns fabric.Text instance from an object representation
   * @static
   * @memberOf fabric.Text
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.Text
instance is created
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Text} Instance of fabric.Text
   */
  fabric.Text.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Text', object, callback, forceAsync,
'text');
  };

  fabric.util.createAccessors(fabric.Text);

})(typeof exports !== 'undefined' ? exports : this);


(function() {

  var clone = fabric.util.object.clone;

  /**
   * IText class (introduced in <b>v1.4</b>) Events are also fired with "text:"
   * prefix when observing canvas.
   * @class fabric.IText
   * @extends fabric.Text
```

```
   * @mixes fabric.Observable
   *
   * @fires changed
   * @fires selection:changed
   * @fires editing:entered
   * @fires editing:exited
   *
   * @return {fabric.IText} thisArg
   * @see {@link fabric.IText#initialize} for constructor definition
   *
   * <p>Supported key combinations:</p>
   * <pre>
   *   Move cursor:                    left, right, up, down
   *   Select character:               shift + left, shift + right
   *   Select text vertically:         shift + up, shift + down
   *   Move cursor by word:            alt + left, alt + right
   *   Select words:                   shift + alt + left, shift + alt + right
   *   Move cursor to line start/end:  cmd + left, cmd + right or home, end
   *   Select till start/end of line:  cmd + shift + left, cmd + shift + right
or shift + home, shift + end
   *   Jump to start/end of text:      cmd + up, cmd + down
   *   Select till start/end of text:  cmd + shift + up, cmd + shift + down or
shift + pgUp, shift + pgDown
   *   Delete character:               backspace
   *   Delete word:                    alt + backspace
   *   Delete line:                    cmd + backspace
   *   Forward delete:                 delete
   *   Copy text:                      ctrl/cmd + c
   *   Paste text:                     ctrl/cmd + v
   *   Cut text:                       ctrl/cmd + x
   *   Select entire text:             ctrl/cmd + a
   *   Quit editing                    tab or esc
   * </pre>
   *
   * <p>Supported mouse/touch combination</p>
   * <pre>
   *   Position cursor:                click/touch
   *   Create selection:               click/touch & drag
   *   Create selection:               click & shift + click
   *   Select word:                    double click
   *   Select line:                    triple click
   * </pre>
   */
  fabric.IText = fabric.util.createClass(fabric.Text, fabric.Observable, /**
@lends fabric.IText.prototype */ {

    /**
     * Type of an object
     * @type String
     * @default
     */
    type: 'i-text',

    /**
     * Index where text selection starts (or where cursor is when there is no
selection)
     * @type Number
     * @default
     */
    selectionStart: 0,

    /**
     * Index where text selection ends
     * @type Number
```

```
     * @default
     */
    selectionEnd: 0,

    /**
     * Color of text selection
     * @type String
     * @default
     */
    selectionColor: 'rgba(17,119,255,0.3)',

    /**
     * Indicates whether text is in editing mode
     * @type Boolean
     * @default
     */
    isEditing: false,

    /**
     * Indicates whether a text can be edited
     * @type Boolean
     * @default
     */
    editable: true,

    /**
     * Border color of text object while it's in editing mode
     * @type String
     * @default
     */
    editingBorderColor: 'rgba(102,153,255,0.25)',

    /**
     * Width of cursor (in px)
     * @type Number
     * @default
     */
    cursorWidth: 2,

    /**
     * Color of default cursor (when not overwritten by character style)
     * @type String
     * @default
     */
    cursorColor: '#333',

    /**
     * Delay between cursor blink (in ms)
     * @type Number
     * @default
     */
    cursorDelay: 1000,

    /**
     * Duration of cursor fadein (in ms)
     * @type Number
     * @default
     */
    cursorDuration: 600,

    /**
     * Object containing character styles
     * (where top-level properties corresponds to line number and 2nd-level
properties -- to char number in a line)
```

```
 * @type Object
 * @default
 */
styles: null,

/**
 * Indicates whether internal text char widths can be cached
 * @type Boolean
 * @default
 */
caching: true,

/**
 * @private
 */
_reSpace: /\s|\n/,

/**
 * @private
 */
_currentCursorOpacity: 0,

/**
 * @private
 */
_selectionDirection: null,

/**
 * @private
 */
_abortCursorAnimation: false,

/**
 * @private
 */
__widthOfSpace: [],

/**
 * Constructor
 * @param {String} text Text string
 * @param {Object} [options] Options object
 * @return {fabric.IText} thisArg
 */
initialize: function(text, options) {
  this.styles = options ? (options.styles || { }) : { };
  this.callSuper('initialize', text, options);
  this.initBehavior();
},

/**
 * @private
 */
_clearCache: function() {
  this.callSuper('_clearCache');
  this.__widthOfSpace = [];
},

/**
 * Returns true if object has no styling
 */
isEmptyStyles: function() {
  if (!this.styles) {
    return true;
  }
```

```
      var obj = this.styles;

      for (var p1 in obj) {
        for (var p2 in obj[p1]) {
          // eslint-disable-next-line no-unused-vars
          for (var p3 in obj[p1][p2]) {
            return false;
          }
        }
      }
      return true;
    },

    /**
     * Sets selection start (left boundary of a selection)
     * @param {Number} index Index to set selection start to
     */
    setSelectionStart: function(index) {
      index = Math.max(index, 0);
      this._updateAndFire('selectionStart', index);
    },

    /**
     * Sets selection end (right boundary of a selection)
     * @param {Number} index Index to set selection end to
     */
    setSelectionEnd: function(index) {
      index = Math.min(index, this.text.length);
      this._updateAndFire('selectionEnd', index);
    },

    /**
     * @private
     * @param {String} property 'selectionStart' or 'selectionEnd'
     * @param {Number} index new position of property
     */
    _updateAndFire: function(property, index) {
      if (this[property] !== index) {
        this._fireSelectionChanged();
        this[property] = index;
      }
      this._updateTextarea();
    },

    /**
     * Fires the even of selection changed
     * @private
     */
    _fireSelectionChanged: function() {
      this.fire('selection:changed');
      this.canvas && this.canvas.fire('text:selection:changed', { target:
this });
    },

    /**
     * Gets style of a current selection/cursor (at the start position)
     * @param {Number} [startIndex] Start index to get styles at
     * @param {Number} [endIndex] End index to get styles at
     * @return {Object} styles Style object at a specified (or current) index
     */
    getSelectionStyles: function(startIndex, endIndex) {

      if (arguments.length === 2) {
        var styles = [];
```

```
        for (var i = startIndex; i < endIndex; i++) {
          styles.push(this.getSelectionStyles(i));
        }
        return styles;
      }

      var loc = this.get2DCursorLocation(startIndex),
          style = this._getStyleDeclaration(loc.lineIndex, loc.charIndex);

      return style || {};
    },

    /**
     * Sets style of a current selection
     * @param {Object} [styles] Styles object
     * @return {fabric.IText} thisArg
     * @chainable
     */
    setSelectionStyles: function(styles) {
      if (this.selectionStart === this.selectionEnd) {
        this._extendStyles(this.selectionStart, styles);
      }
      else {
        for (var i = this.selectionStart; i < this.selectionEnd; i++) {
          this._extendStyles(i, styles);
        }
      }
      /* not included in _extendStyles to avoid clearing cache more than once */
      this._forceClearCache = true;
      return this;
    },

    /**
     * @private
     */
    _extendStyles: function(index, styles) {
      var loc = this.get2DCursorLocation(index);

      if (!this._getLineStyle(loc.lineIndex)) {
        this._setLineStyle(loc.lineIndex, {});
      }

      if (!this._getStyleDeclaration(loc.lineIndex, loc.charIndex)) {
        this._setStyleDeclaration(loc.lineIndex, loc.charIndex, {});
      }

      fabric.util.object.extend(this._getStyleDeclaration(loc.lineIndex,
loc.charIndex), styles);
    },

    /**
     * Initialize text dimensions. Render all text on given context
     * or on a offscreen canvas to get the text width with measureText.
     * Updates this.width and this.height with the proper values.
     * Does not return dimensions.
     * @param {CanvasRenderingContext2D} [ctx] Context to render on
     * @private
     */
    _initDimensions: function(ctx) {
      if (!ctx) {
        this.clearContextTop();
      }
      this.callSuper('_initDimensions', ctx);
    },
```

```
    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Boolean} noTransform
     */
    render: function(ctx, noTransform) {
      this.clearContextTop();
      this.callSuper('render', ctx, noTransform);
      // clear the cursorOffsetCache, so we ensure to calculate once per
renderCursor
      // the correct position but not at every cursor animation.
      this.cursorOffsetCache = { };
      this.renderCursorOrSelection();
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _render: function(ctx) {
      this.callSuper('_render', ctx);
      this.ctx = ctx;
    },

    /**
     * Prepare and clean the contextTop
     */
    clearContextTop: function() {
      if (!this.active || !this.isEditing) {
        return;
      }
      if (this.canvas && this.canvas.contextTop) {
        var ctx = this.canvas.contextTop;
        ctx.save();
        ctx.transform.apply(ctx, this.canvas.viewportTransform);
        this.transform(ctx);
        this.transformMatrix && ctx.transform.apply(ctx, this.transformMatrix);
        this._clearTextArea(ctx);
        ctx.restore();
      }
    },

    /**
     * Renders cursor or selection (depending on what exists)
     */
    renderCursorOrSelection: function() {
      if (!this.active || !this.isEditing) {
        return;
      }
      var chars = this.text.split(''),
          boundaries, ctx;
      if (this.canvas && this.canvas.contextTop) {
        ctx = this.canvas.contextTop;
        ctx.save();
        ctx.transform.apply(ctx, this.canvas.viewportTransform);
        this.transform(ctx);
        this.transformMatrix && ctx.transform.apply(ctx, this.transformMatrix);
        this._clearTextArea(ctx);
      }
      else {
        ctx = this.ctx;
        ctx.save();
      }
```

```
      if (this.selectionStart === this.selectionEnd) {
        boundaries = this._getCursorBoundaries(chars, 'cursor');
        this.renderCursor(boundaries, ctx);
      }
      else {
        boundaries = this._getCursorBoundaries(chars, 'selection');
        this.renderSelection(chars, boundaries, ctx);
      }
      ctx.restore();
    },

    _clearTextArea: function(ctx) {
      // we add 4 pixel, to be sure to do not leave any pixel out
      var width = this.width + 4, height = this.height + 4;
      ctx.clearRect(-width / 2, -height / 2, width, height);
    },
    /**
     * Returns 2d representation (lineIndex and charIndex) of cursor (or
selection start)
     * @param {Number} [selectionStart] Optional index. When not given, current
selectionStart is used.
     */
    get2DCursorLocation: function(selectionStart) {
      if (typeof selectionStart === 'undefined') {
        selectionStart = this.selectionStart;
      }
      var len = this._textLines.length;
      for (var i = 0; i < len; i++) {
        if (selectionStart <= this._textLines[i].length) {
          return {
            lineIndex: i,
            charIndex: selectionStart
          };
        }
        selectionStart -= this._textLines[i].length + 1;
      }
      return {
        lineIndex: i - 1,
        charIndex: this._textLines[i - 1].length < selectionStart ?
this._textLines[i - 1].length : selectionStart
      };
    },

    /**
     * Returns complete style of char at the current cursor
     * @param {Number} lineIndex Line index
     * @param {Number} charIndex Char index
     * @return {Object} Character style
     */
    getCurrentCharStyle: function(lineIndex, charIndex) {
      var style = this._getStyleDeclaration(lineIndex, charIndex === 0 ? 0 :
charIndex - 1);

      return {
        fontSize: style && style.fontSize || this.fontSize,
        fill: style && style.fill || this.fill,
        textBackgroundColor: style && style.textBackgroundColor ||
this.textBackgroundColor,
        textDecoration: style && style.textDecoration || this.textDecoration,
        fontFamily: style && style.fontFamily || this.fontFamily,
        fontWeight: style && style.fontWeight || this.fontWeight,
        fontStyle: style && style.fontStyle || this.fontStyle,
        stroke: style && style.stroke || this.stroke,
        strokeWidth: style && style.strokeWidth || this.strokeWidth
```

```
      };
    },

    /**
     * Returns fontSize of char at the current cursor
     * @param {Number} lineIndex Line index
     * @param {Number} charIndex Char index
     * @return {Number} Character font size
     */
    getCurrentCharFontSize: function(lineIndex, charIndex) {
      var style = this._getStyleDeclaration(lineIndex, charIndex === 0 ? 0 :
charIndex - 1);
      return style && style.fontSize ? style.fontSize : this.fontSize;
    },

    /**
     * Returns color (fill) of char at the current cursor
     * @param {Number} lineIndex Line index
     * @param {Number} charIndex Char index
     * @return {String} Character color (fill)
     */
    getCurrentCharColor: function(lineIndex, charIndex) {
      var style = this._getStyleDeclaration(lineIndex, charIndex === 0 ? 0 :
charIndex - 1);
      return style && style.fill ? style.fill : this.cursorColor;
    },

    /**
     * Returns cursor boundaries (left, top, leftOffset, topOffset)
     * @private
     * @param {Array} chars Array of characters
     * @param {String} typeOfBoundaries
     */
    _getCursorBoundaries: function(chars, typeOfBoundaries) {

      // left/top are left/top of entire text box
      // leftOffset/topOffset are offset from that left/top point of a text box

      var left = Math.round(this._getLeftOffset()),
          top = this._getTopOffset(),

          offsets = this._getCursorBoundariesOffsets(
                      chars, typeOfBoundaries);

      return {
        left: left,
        top: top,
        leftOffset: offsets.left + offsets.lineLeft,
        topOffset: offsets.top
      };
    },

    /**
     * @private
     */
    _getCursorBoundariesOffsets: function(chars, typeOfBoundaries) {
      if (this.cursorOffsetCache && 'top' in this.cursorOffsetCache) {
        return this.cursorOffsetCache;
      }
      var lineLeftOffset = 0,
          lineIndex = 0,
          charIndex = 0,
          topOffset = 0,
          leftOffset = 0,
```

```
          boundaries;

      for (var i = 0; i < this.selectionStart; i++) {
        if (chars[i] === '\n') {
          leftOffset = 0;
          topOffset += this._getHeightOfLine(this.ctx, lineIndex);

          lineIndex++;
          charIndex = 0;
        }
        else {
          leftOffset += this._getWidthOfChar(this.ctx, chars[i], lineIndex,
charIndex);
          charIndex++;
        }

        lineLeftOffset = this._getLineLeftOffset(this._getLineWidth(this.ctx,
lineIndex));
      }
      if (typeOfBoundaries === 'cursor') {
        topOffset += (1 - this._fontSizeFraction) *
this._getHeightOfLine(this.ctx, lineIndex) / this.lineHeight
          - this.getCurrentCharFontSize(lineIndex, charIndex) * (1 -
this._fontSizeFraction);
      }
      if (this.charSpacing !== 0 && charIndex ===
this._textLines[lineIndex].length) {
        leftOffset -= this._getWidthOfCharSpacing();
      }
      boundaries = {
        top: topOffset,
        left: leftOffset > 0 ? leftOffset : 0,
        lineLeft: lineLeftOffset
      };
      this.cursorOffsetCache = boundaries;
      return this.cursorOffsetCache;
    },

    /**
     * Renders cursor
     * @param {Object} boundaries
     * @param {CanvasRenderingContext2D} ctx transformed context to draw on
     */
    renderCursor: function(boundaries, ctx) {

      var cursorLocation = this.get2DCursorLocation(),
          lineIndex = cursorLocation.lineIndex,
          charIndex = cursorLocation.charIndex,
          charHeight = this.getCurrentCharFontSize(lineIndex, charIndex),
          leftOffset = boundaries.leftOffset,
          multiplier = this.scaleX * this.canvas.getZoom(),
          cursorWidth = this.cursorWidth / multiplier;

      ctx.fillStyle = this.getCurrentCharColor(lineIndex, charIndex);
      ctx.globalAlpha = this.__isMousedown ? 1 : this._currentCursorOpacity;

      ctx.fillRect(
        boundaries.left + leftOffset - cursorWidth / 2,
        boundaries.top + boundaries.topOffset,
        cursorWidth,
        charHeight);
    },

    /**
```

```
     * Renders text selection
     * @param {Array} chars Array of characters
     * @param {Object} boundaries Object with left/top/leftOffset/topOffset
     * @param {CanvasRenderingContext2D} ctx transformed context to draw on
     */
    renderSelection: function(chars, boundaries, ctx) {

      ctx.fillStyle = this.selectionColor;

      var start = this.get2DCursorLocation(this.selectionStart),
          end = this.get2DCursorLocation(this.selectionEnd),
          startLine = start.lineIndex,
          endLine = end.lineIndex;
      for (var i = startLine; i <= endLine; i++) {
        var lineOffset = this._getLineLeftOffset(this._getLineWidth(ctx, i)) ||
0,
            lineHeight = this._getHeightOfLine(this.ctx, i),
            realLineHeight = 0, boxWidth = 0, line = this._textLines[i];

        if (i === startLine) {
          for (var j = 0, len = line.length; j < len; j++) {
            if (j >= start.charIndex && (i !== endLine || j < end.charIndex)) {
              boxWidth += this._getWidthOfChar(ctx, line[j], i, j);
            }
            if (j < start.charIndex) {
              lineOffset += this._getWidthOfChar(ctx, line[j], i, j);
            }
          }
          if (j === line.length) {
            boxWidth -= this._getWidthOfCharSpacing();
          }
        }
        else if (i > startLine && i < endLine) {
          boxWidth += this._getLineWidth(ctx, i) || 5;
        }
        else if (i === endLine) {
          for (var j2 = 0, j2len = end.charIndex; j2 < j2len; j2++) {
            boxWidth += this._getWidthOfChar(ctx, line[j2], i, j2);
          }
          if (end.charIndex === line.length) {
            boxWidth -= this._getWidthOfCharSpacing();
          }
        }
        realLineHeight = lineHeight;
        if (this.lineHeight < 1 || (i === endLine && this.lineHeight > 1)) {
          lineHeight /= this.lineHeight;
        }
        ctx.fillRect(
          boundaries.left + lineOffset,
          boundaries.top + boundaries.topOffset,
          boxWidth > 0 ? boxWidth : 0,
          lineHeight);

        boundaries.topOffset += realLineHeight;
      }
    },

    /**
     * @private
     * @param {String} method
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} line Content of the line
     * @param {Number} left
     * @param {Number} top
```

```
     * @param {Number} lineIndex
     * @param {Number} charOffset
     */
    _renderChars: function(method, ctx, line, left, top, lineIndex, charOffset)
{

      if (this.isEmptyStyles()) {
        return this._renderCharsFast(method, ctx, line, left, top);
      }

      charOffset = charOffset || 0;

      // set proper line offset
      var lineHeight = this._getHeightOfLine(ctx, lineIndex),
          prevStyle,
          thisStyle,
          charsToRender = '';

      ctx.save();
      top -= lineHeight / this.lineHeight * this._fontSizeFraction;
      for (var i = charOffset, len = line.length + charOffset; i <= len; i++) {
        prevStyle = prevStyle || this.getCurrentCharStyle(lineIndex, i);
        thisStyle = this.getCurrentCharStyle(lineIndex, i + 1);

        if (this._hasStyleChanged(prevStyle, thisStyle) || i === len) {
          this._renderChar(method, ctx, lineIndex, i - 1, charsToRender, left,
top, lineHeight);
          charsToRender = '';
          prevStyle = thisStyle;
        }
        charsToRender += line[i - charOffset];
      }
      ctx.restore();
    },

    /**
     * @private
     * @param {String} method
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} line Content of the line
     * @param {Number} left Left coordinate
     * @param {Number} top Top coordinate
     */
    _renderCharsFast: function(method, ctx, line, left, top) {

      if (method === 'fillText' && this.fill) {
        this.callSuper('_renderChars', method, ctx, line, left, top);
      }
      if (method === 'strokeText' && ((this.stroke && this.strokeWidth > 0) ||
this.skipFillStrokeCheck)) {
        this.callSuper('_renderChars', method, ctx, line, left, top);
      }
    },

    /**
     * @private
     * @param {String} method
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex
     * @param {Number} i
     * @param {String} _char
     * @param {Number} left Left coordinate
     * @param {Number} top Top coordinate
     * @param {Number} lineHeight Height of the line
```

```
     */
    _renderChar: function(method, ctx, lineIndex, i, _char, left, top,
lineHeight) {
      var charWidth, charHeight, shouldFill, shouldStroke,
          decl = this._getStyleDeclaration(lineIndex, i),
          offset, textDecoration, chars, additionalSpace, _charWidth;

      if (decl) {
        charHeight = this._getHeightOfChar(ctx, _char, lineIndex, i);
        shouldStroke = decl.stroke;
        shouldFill = decl.fill;
        textDecoration = decl.textDecoration;
      }
      else {
        charHeight = this.fontSize;
      }

      shouldStroke = (shouldStroke || this.stroke) && method === 'strokeText';
      shouldFill = (shouldFill || this.fill) && method === 'fillText';

      decl && ctx.save();

      charWidth = this._applyCharStylesGetWidth(ctx, _char, lineIndex, i, decl
|| null);
      textDecoration = textDecoration || this.textDecoration;

      if (decl && decl.textBackgroundColor) {
        this._removeShadow(ctx);
      }
      if (this.charSpacing !== 0) {
        additionalSpace = this._getWidthOfCharSpacing();
        chars = _char.split('');
        charWidth = 0;
        for (var j = 0, len = chars.length, jChar; j < len; j++) {
          jChar = chars[j];
          shouldFill && ctx.fillText(jChar, left + charWidth, top);
          shouldStroke && ctx.strokeText(jChar, left + charWidth, top);
          _charWidth = ctx.measureText(jChar).width + additionalSpace;
          charWidth += _charWidth > 0 ? _charWidth : 0;
        }
      }
      else {
        shouldFill && ctx.fillText(_char, left, top);
        shouldStroke && ctx.strokeText(_char, left, top);
      }

      if (textDecoration || textDecoration !== '') {
        offset = this._fontSizeFraction * lineHeight / this.lineHeight;
        this._renderCharDecoration(ctx, textDecoration, left, top, offset,
charWidth, charHeight);
      }

      decl && ctx.restore();
      ctx.translate(charWidth, 0);
    },

    /**
     * @private
     * @param {Object} prevStyle
     * @param {Object} thisStyle
     */
    _hasStyleChanged: function(prevStyle, thisStyle) {
      return (prevStyle.fill !== thisStyle.fill ||
              prevStyle.fontSize !== thisStyle.fontSize ||
```

```
              prevStyle.textBackgroundColor !== thisStyle.textBackgroundColor ||
              prevStyle.textDecoration !== thisStyle.textDecoration ||
              prevStyle.fontFamily !== thisStyle.fontFamily ||
              prevStyle.fontWeight !== thisStyle.fontWeight ||
              prevStyle.fontStyle !== thisStyle.fontStyle ||
              prevStyle.stroke !== thisStyle.stroke ||
              prevStyle.strokeWidth !== thisStyle.strokeWidth
      );
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderCharDecoration: function(ctx, textDecoration, left, top, offset,
charWidth, charHeight) {

      if (!textDecoration) {
        return;
      }

      var decorationWeight = charHeight / 15,
          positions = {
            underline: top + charHeight / 10,
            'line-through': top - charHeight * (this._fontSizeFraction +
this._fontSizeMult - 1) + decorationWeight,
            overline: top - (this._fontSizeMult - this._fontSizeFraction) *
charHeight
          },
          decorations = ['underline', 'line-through', 'overline'], i,
decoration;

      for (i = 0; i < decorations.length; i++) {
        decoration = decorations[i];
        if (textDecoration.indexOf(decoration) > -1) {
          ctx.fillRect(left, positions[decoration], charWidth ,
decorationWeight);
        }
      }
    },

    /**
     * @private
     * @param {String} method
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} line
     * @param {Number} left
     * @param {Number} top
     * @param {Number} lineIndex
     */
    _renderTextLine: function(method, ctx, line, left, top, lineIndex) {
      // to "cancel" this.fontSize subtraction in fabric.Text#_renderTextLine
      // the adding 0.03 is just to align text with itext by overlap test
      if (!this.isEmptyStyles()) {
        top += this.fontSize * (this._fontSizeFraction + 0.03);
      }
      this.callSuper('_renderTextLine', method, ctx, line, left, top,
lineIndex);
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
```

```
    _renderTextDecoration: function(ctx) {
      if (this.isEmptyStyles()) {
        return this.callSuper('_renderTextDecoration', ctx);
      }
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _renderTextLinesBackground: function(ctx) {
      this.callSuper('_renderTextLinesBackground', ctx);

      var lineTopOffset = 0, heightOfLine,
          lineWidth, lineLeftOffset,
          leftOffset = this._getLeftOffset(),
          topOffset = this._getTopOffset(),
          colorCache = '',
          line, _char, style, leftCache,
          topCache, widthCache, heightCache;
      ctx.save();
      for (var i = 0, len = this._textLines.length; i < len; i++) {
        heightOfLine = this._getHeightOfLine(ctx, i);
        line = this._textLines[i];

        if (line === '' || !this.styles || !this._getLineStyle(i)) {
          lineTopOffset += heightOfLine;
          continue;
        }

        lineWidth = this._getLineWidth(ctx, i);
        lineLeftOffset = this._getLineLeftOffset(lineWidth);
        leftCache = topCache = widthCache = heightCache = 0;
        for (var j = 0, jlen = line.length; j < jlen; j++) {
          style = this._getStyleDeclaration(i, j) || {};

          if (colorCache !== style.textBackgroundColor) {
            if (heightCache && widthCache) {
              ctx.fillStyle = colorCache;
              ctx.fillRect(leftCache, topCache, widthCache, heightCache);
            }
            leftCache = topCache = widthCache = heightCache = 0;
            colorCache = style.textBackgroundColor || '';
          }

          if (!style.textBackgroundColor) {
            colorCache = '';
            continue;
          }
          _char = line[j];

          if (colorCache === style.textBackgroundColor) {
            colorCache = style.textBackgroundColor;
            if (!leftCache) {
              leftCache = leftOffset + lineLeftOffset +
this._getWidthOfCharsAt(ctx, i, j);
            }
            topCache = topOffset + lineTopOffset;
            widthCache += this._getWidthOfChar(ctx, _char, i, j);
            heightCache = heightOfLine / this.lineHeight;
          }
        }
        // if a textBackgroundColor ends on the last character of a line
        if (heightCache && widthCache) {
```

```
          ctx.fillStyle = colorCache;
          ctx.fillRect(leftCache, topCache, widthCache, heightCache);
          leftCache = topCache = widthCache = heightCache = 0;
        }
        lineTopOffset += heightOfLine;
      }
      ctx.restore();
    },

    /**
     * @private
     */
    _getCacheProp: function(_char, styleDeclaration) {
      return _char +
             styleDeclaration.fontSize +
             styleDeclaration.fontWeight +
             styleDeclaration.fontStyle;
    },

    /**
     * @private
     * @param {String} fontFamily name
     * @return {Object} reference to cache
     */
    _getFontCache: function(fontFamily) {
      if (!fabric.charWidthsCache[fontFamily]) {
        fabric.charWidthsCache[fontFamily] = { };
      }
      return fabric.charWidthsCache[fontFamily];
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} _char
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @param {Object} [decl]
     */
    _applyCharStylesGetWidth: function(ctx, _char, lineIndex, charIndex, decl) {
      var charDecl = decl || this._getStyleDeclaration(lineIndex, charIndex),
          styleDeclaration = clone(charDecl),
          width, cacheProp, charWidthsCache;

      this._applyFontStyles(styleDeclaration);
      charWidthsCache = this._getFontCache(styleDeclaration.fontFamily);
      cacheProp = this._getCacheProp(_char, styleDeclaration);

      // short-circuit if no styles for this char
      // global style from object is always applyed and handled by save and
restore
      if (!charDecl && charWidthsCache[cacheProp] && this.caching) {
        return charWidthsCache[cacheProp];
      }

      if (typeof styleDeclaration.shadow === 'string') {
        styleDeclaration.shadow = new fabric.Shadow(styleDeclaration.shadow);
      }

      var fill = styleDeclaration.fill || this.fill;
      ctx.fillStyle = fill.toLive
        ? fill.toLive(ctx, this)
        : fill;
```

```
      if (styleDeclaration.stroke) {
        ctx.strokeStyle = (styleDeclaration.stroke &&
styleDeclaration.stroke.toLive)
          ? styleDeclaration.stroke.toLive(ctx, this)
          : styleDeclaration.stroke;
      }

      ctx.lineWidth = styleDeclaration.strokeWidth || this.strokeWidth;
      ctx.font = this._getFontDeclaration.call(styleDeclaration);

      //if we want this._setShadow.call to work with styleDeclarion
      //we have to add those references
      if (styleDeclaration.shadow) {
        styleDeclaration.scaleX = this.scaleX;
        styleDeclaration.scaleY = this.scaleY;
        styleDeclaration.canvas = this.canvas;
        styleDeclaration.getObjectScaling = this.getObjectScaling;
        this._setShadow.call(styleDeclaration, ctx);
      }

      if (!this.caching || !charWidthsCache[cacheProp]) {
        width = ctx.measureText(_char).width;
        this.caching && (charWidthsCache[cacheProp] = width);
        return width;
      }

      return charWidthsCache[cacheProp];
    },

    /**
     * @private
     * @param {Object} styleDeclaration
     */
    _applyFontStyles: function(styleDeclaration) {
      if (!styleDeclaration.fontFamily) {
        styleDeclaration.fontFamily = this.fontFamily;
      }
      if (!styleDeclaration.fontSize) {
        styleDeclaration.fontSize = this.fontSize;
      }
      if (!styleDeclaration.fontWeight) {
        styleDeclaration.fontWeight = this.fontWeight;
      }
      if (!styleDeclaration.fontStyle) {
        styleDeclaration.fontStyle = this.fontStyle;
      }
    },

    /**
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @param {Boolean} [returnCloneOrEmpty=false]
     * @private
     */
    _getStyleDeclaration: function(lineIndex, charIndex, returnCloneOrEmpty) {
      if (returnCloneOrEmpty) {
        return (this.styles[lineIndex] && this.styles[lineIndex][charIndex])
          ? clone(this.styles[lineIndex][charIndex])
          : { };
      }

      return this.styles[lineIndex] && this.styles[lineIndex][charIndex] ?
this.styles[lineIndex][charIndex] : null;
    },
```

```
    /**
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @param {Object} style
     * @private
     */
    _setStyleDeclaration: function(lineIndex, charIndex, style) {
      this.styles[lineIndex][charIndex] = style;
    },

    /**
     *
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @private
     */
    _deleteStyleDeclaration: function(lineIndex, charIndex) {
      delete this.styles[lineIndex][charIndex];
    },

    /**
     * @param {Number} lineIndex
     * @private
     */
    _getLineStyle: function(lineIndex) {
      return this.styles[lineIndex];
    },

    /**
     * @param {Number} lineIndex
     * @param {Object} style
     * @private
     */
    _setLineStyle: function(lineIndex, style) {
      this.styles[lineIndex] = style;
    },

    /**
     * @param {Number} lineIndex
     * @private
     */
    _deleteLineStyle: function(lineIndex) {
      delete this.styles[lineIndex];
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _getWidthOfChar: function(ctx, _char, lineIndex, charIndex) {
      if (!this._isMeasuring && this.textAlign === 'justify' &&
this._reSpacesAndTabs.test(_char)) {
        return this._getWidthOfSpace(ctx, lineIndex);
      }
      ctx.save();
      var width = this._applyCharStylesGetWidth(ctx, _char, lineIndex,
charIndex);
      if (this.charSpacing !== 0) {
        width += this._getWidthOfCharSpacing();
      }
      ctx.restore();
      return width > 0 ? width : 0;
    },
```

```
    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex
     * @param {Number} charIndex
     */
    _getHeightOfChar: function(ctx, lineIndex, charIndex) {
      var style = this._getStyleDeclaration(lineIndex, charIndex);
      return style && style.fontSize ? style.fontSize : this.fontSize;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex
     * @param {Number} charIndex
     */
    _getWidthOfCharsAt: function(ctx, lineIndex, charIndex) {
      var width = 0, i, _char;
      for (i = 0; i < charIndex; i++) {
        _char = this._textLines[lineIndex][i];
        width += this._getWidthOfChar(ctx, _char, lineIndex, i);
      }
      return width;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex line number
     * @return {Number} Line width
     */
    _measureLine: function(ctx, lineIndex) {
      this._isMeasuring = true;
      var width = this._getWidthOfCharsAt(ctx, lineIndex,
this._textLines[lineIndex].length);
      if (this.charSpacing !== 0) {
        width -= this._getWidthOfCharSpacing();
      }
      this._isMeasuring = false;
      return width > 0 ? width : 0;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {Number} lineIndex
     */
    _getWidthOfSpace: function (ctx, lineIndex) {
      if (this.__widthOfSpace[lineIndex]) {
        return this.__widthOfSpace[lineIndex];
      }
      var line = this._textLines[lineIndex],
          wordsWidth = this._getWidthOfWords(ctx, line, lineIndex, 0),
          widthDiff = this.width - wordsWidth,
          numSpaces = line.length - line.replace(this._reSpacesAndTabs,
'').length,
          width = Math.max(widthDiff / numSpaces, ctx.measureText(' ').width);
      this.__widthOfSpace[lineIndex] = width;
      return width;
    },

    /**
```

```
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     * @param {String} line
     * @param {Number} lineIndex
     * @param {Number} charOffset
     */
    _getWidthOfWords: function (ctx, line, lineIndex, charOffset) {
      var width = 0;

      for (var charIndex = 0; charIndex < line.length; charIndex++) {
        var _char = line[charIndex];

        if (!_char.match(/\s/)) {
          width += this._getWidthOfChar(ctx, _char, lineIndex, charIndex +
charOffset);
        }
      }

      return width;
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _getHeightOfLine: function(ctx, lineIndex) {
      if (this.__lineHeights[lineIndex]) {
        return this.__lineHeights[lineIndex];
      }

      var line = this._textLines[lineIndex],
          maxHeight = this._getHeightOfChar(ctx, lineIndex, 0);

      for (var i = 1, len = line.length; i < len; i++) {
        var currentCharHeight = this._getHeightOfChar(ctx, lineIndex, i);
        if (currentCharHeight > maxHeight) {
          maxHeight = currentCharHeight;
        }
      }
      this.__lineHeights[lineIndex] = maxHeight * this.lineHeight *
this._fontSizeMult;
      return this.__lineHeights[lineIndex];
    },

    /**
     * @private
     * @param {CanvasRenderingContext2D} ctx Context to render on
     */
    _getTextHeight: function(ctx) {
      var lineHeight, height = 0;
      for (var i = 0, len = this._textLines.length; i < len; i++) {
        lineHeight = this._getHeightOfLine(ctx, i);
        height += (i === len - 1 ? lineHeight / this.lineHeight : lineHeight);
      }
      return height;
    },

    /**
     * Returns object representation of an instance
     * @method toObject
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
```

```
    toObject: function(propertiesToInclude) {
      return fabric.util.object.extend(this.callSuper('toObject',
propertiesToInclude), {
        styles: clone(this.styles, true)
      });
    }
  });

  /**
   * Returns fabric.IText instance from an object representation
   * @static
   * @memberOf fabric.IText
   * @param {Object} object Object to create an instance from
   * @param {function} [callback] invoked with new instance as argument
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.IText} instance of fabric.IText
   */
  fabric.IText.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('IText', object, callback, forceAsync,
'text');
  };
})();


(function() {

  var clone = fabric.util.object.clone;

  fabric.util.object.extend(fabric.IText.prototype, /** @lends
fabric.IText.prototype */ {

    /**
     * Initializes all the interactive behavior of IText
     */
    initBehavior: function() {
      this.initAddedHandler();
      this.initRemovedHandler();
      this.initCursorSelectionHandlers();
      this.initDoubleClickSimulation();
      this.mouseMoveHandler = this.mouseMoveHandler.bind(this);
    },

    onDeselect: function() {
      this.isEditing && this.exitEditing();
      this.selected = false;
      this.callSuper('onDeselect');
    },

    /**
     * Initializes "added" event handler
     */
    initAddedHandler: function() {
      var _this = this;
      this.on('added', function() {
        var canvas = _this.canvas;
        if (canvas) {
          if (!canvas._hasITextHandlers) {
            canvas._hasITextHandlers = true;
            _this._initCanvasHandlers(canvas);
          }
          canvas._iTextInstances = canvas._iTextInstances || [];
          canvas._iTextInstances.push(_this);
        }
```

```
      });
    },

    initRemovedHandler: function() {
      var _this = this;
      this.on('removed', function() {
        var canvas = _this.canvas;
        if (canvas) {
          canvas._iTextInstances = canvas._iTextInstances || [];
          fabric.util.removeFromArray(canvas._iTextInstances, _this);
          if (canvas._iTextInstances.length === 0) {
            canvas._hasITextHandlers = false;
            _this._removeCanvasHandlers(canvas);
          }
        }
      });
    },

    /**
     * register canvas event to manage exiting on other instances
     * @private
     */
    _initCanvasHandlers: function(canvas) {
      canvas._mouseUpITextHandler = (function() {
        if (canvas._iTextInstances) {
          canvas._iTextInstances.forEach(function(obj) {
            obj.__isMousedown = false;
          });
        }
      }).bind(this);
      canvas.on('mouse:up', canvas._mouseUpITextHandler);
    },

    /**
     * remove canvas event to manage exiting on other instances
     * @private
     */
    _removeCanvasHandlers: function(canvas) {
      canvas.off('mouse:up', canvas._mouseUpITextHandler);
    },

    /**
     * @private
     */
    _tick: function() {
      this._currentTickState = this._animateCursor(this, 1, this.cursorDuration,
'_onTickComplete');
    },

    /**
     * @private
     */
    _animateCursor: function(obj, targetOpacity, duration, completeMethod) {

      var tickState;

      tickState = {
        isAborted: false,
        abort: function() {
          this.isAborted = true;
        },
      };

      obj.animate('_currentCursorOpacity', targetOpacity, {
```

```
        duration: duration,
        onComplete: function() {
          if (!tickState.isAborted) {
            obj[completeMethod]();
          }
        },
        onChange: function() {
          // we do not want to animate a selection, only cursor
          if (obj.canvas && obj.selectionStart === obj.selectionEnd) {
            obj.renderCursorOrSelection();
          }
        },
        abort: function() {
          return tickState.isAborted;
        }
      });
      return tickState;
    },

    /**
     * @private
     */
    _onTickComplete: function() {

      var _this = this;

      if (this._cursorTimeout1) {
        clearTimeout(this._cursorTimeout1);
      }
      this._cursorTimeout1 = setTimeout(function() {
        _this._currentTickCompleteState = _this._animateCursor(_this, 0,
this.cursorDuration / 2, '_tick');
      }, 100);
    },

    /**
     * Initializes delayed cursor
     */
    initDelayedCursor: function(restart) {
      var _this = this,
          delay = restart ? 0 : this.cursorDelay;

      this.abortCursorAnimation();
      this._currentCursorOpacity = 1;
      this._cursorTimeout2 = setTimeout(function() {
        _this._tick();
      }, delay);
    },

    /**
     * Aborts cursor animation and clears all timeouts
     */
    abortCursorAnimation: function() {
      var shouldClear = this._currentTickState ||
this._currentTickCompleteState;
      this._currentTickState && this._currentTickState.abort();
      this._currentTickCompleteState && this._currentTickCompleteState.abort();

      clearTimeout(this._cursorTimeout1);
      clearTimeout(this._cursorTimeout2);

      this._currentCursorOpacity = 0;
      // to clear just itext area we need to transform the context
      // it may not be worth it
```

```
      if (shouldClear) {
        this.canvas && this.canvas.clearContext(this.canvas.contextTop ||
this.ctx);
      }

    },

    /**
     * Selects entire text
     */
    selectAll: function() {
      this.selectionStart = 0;
      this.selectionEnd = this.text.length;
      this._fireSelectionChanged();
      this._updateTextarea();
    },

    /**
     * Returns selected text
     * @return {String}
     */
    getSelectedText: function() {
      return this.text.slice(this.selectionStart, this.selectionEnd);
    },

    /**
     * Find new selection index representing start of current word according to
current selection index
     * @param {Number} startFrom Surrent selection index
     * @return {Number} New selection index
     */
    findWordBoundaryLeft: function(startFrom) {
      var offset = 0, index = startFrom - 1;

      // remove space before cursor first
      if (this._reSpace.test(this.text.charAt(index))) {
        while (this._reSpace.test(this.text.charAt(index))) {
          offset++;
          index--;
        }
      }
      while (/\S/.test(this.text.charAt(index)) && index > -1) {
        offset++;
        index--;
      }

      return startFrom - offset;
    },

    /**
     * Find new selection index representing end of current word according to
current selection index
     * @param {Number} startFrom Current selection index
     * @return {Number} New selection index
     */
    findWordBoundaryRight: function(startFrom) {
      var offset = 0, index = startFrom;

      // remove space after cursor first
      if (this._reSpace.test(this.text.charAt(index))) {
        while (this._reSpace.test(this.text.charAt(index))) {
          offset++;
          index++;
        }
```

```
      }
      while (/\S/.test(this.text.charAt(index)) && index < this.text.length) {
        offset++;
        index++;
      }

      return startFrom + offset;
    },

    /**
     * Find new selection index representing start of current line according to
current selection index
     * @param {Number} startFrom Current selection index
     * @return {Number} New selection index
     */
    findLineBoundaryLeft: function(startFrom) {
      var offset = 0, index = startFrom - 1;

      while (!/\n/.test(this.text.charAt(index)) && index > -1) {
        offset++;
        index--;
      }

      return startFrom - offset;
    },

    /**
     * Find new selection index representing end of current line according to
current selection index
     * @param {Number} startFrom Current selection index
     * @return {Number} New selection index
     */
    findLineBoundaryRight: function(startFrom) {
      var offset = 0, index = startFrom;

      while (!/\n/.test(this.text.charAt(index)) && index < this.text.length) {
        offset++;
        index++;
      }

      return startFrom + offset;
    },

    /**
     * Returns number of newlines in selected text
     * @return {Number} Number of newlines in selected text
     */
    getNumNewLinesInSelectedText: function() {
      var selectedText = this.getSelectedText(),
          numNewLines  = 0;

      for (var i = 0, len = selectedText.length; i < len; i++) {
        if (selectedText[i] === '\n') {
          numNewLines++;
        }
      }
      return numNewLines;
    },

    /**
     * Finds index corresponding to beginning or end of a word
     * @param {Number} selectionStart Index of a character
     * @param {Number} direction 1 or -1
     * @return {Number} Index of the beginning or end of a word
```

```
     */
    searchWordBoundary: function(selectionStart, direction) {
      var index     = this._reSpace.test(this.text.charAt(selectionStart)) ?
selectionStart - 1 : selectionStart,
          _char     = this.text.charAt(index),
          reNonWord = /[ \n\.,;!\?\-]/;

      while (!reNonWord.test(_char) && index > 0 && index < this.text.length) {
        index += direction;
        _char = this.text.charAt(index);
      }
      if (reNonWord.test(_char) && _char !== '\n') {
        index += direction === 1 ? 0 : 1;
      }
      return index;
    },

    /**
     * Selects a word based on the index
     * @param {Number} selectionStart Index of a character
     */
    selectWord: function(selectionStart) {
      selectionStart = selectionStart || this.selectionStart;
      var newSelectionStart = this.searchWordBoundary(selectionStart, -1), /*
search backwards */
          newSelectionEnd = this.searchWordBoundary(selectionStart, 1); /*
search forward */

      this.selectionStart = newSelectionStart;
      this.selectionEnd = newSelectionEnd;
      this._fireSelectionChanged();
      this._updateTextarea();
      this.renderCursorOrSelection();
    },

    /**
     * Selects a line based on the index
     * @param {Number} selectionStart Index of a character
     */
    selectLine: function(selectionStart) {
      selectionStart = selectionStart || this.selectionStart;
      var newSelectionStart = this.findLineBoundaryLeft(selectionStart),
          newSelectionEnd = this.findLineBoundaryRight(selectionStart);

      this.selectionStart = newSelectionStart;
      this.selectionEnd = newSelectionEnd;
      this._fireSelectionChanged();
      this._updateTextarea();
    },

    /**
     * Enters editing state
     * @return {fabric.IText} thisArg
     * @chainable
     */
    enterEditing: function(e) {
      if (this.isEditing || !this.editable) {
        return;
      }

      if (this.canvas) {
        this.exitEditingOnOthers(this.canvas);
      }
```

```
      this.isEditing = true;
      this.selected = true;
      this.initHiddenTextarea(e);
      this.hiddenTextarea.focus();
      this._updateTextarea();
      this._saveEditingProps();
      this._setEditingProps();
      this._textBeforeEdit = this.text;

      this._tick();
      this.fire('editing:entered');
      this._fireSelectionChanged();
      if (!this.canvas) {
        return this;
      }
      this.canvas.fire('text:editing:entered', { target: this });
      this.initMouseMoveHandler();
      this.canvas.renderAll();
      return this;
    },

    exitEditingOnOthers: function(canvas) {
      if (canvas._iTextInstances) {
        canvas._iTextInstances.forEach(function(obj) {
          obj.selected = false;
          if (obj.isEditing) {
            obj.exitEditing();
          }
        });
      }
    },

    /**
     * Initializes "mousemove" event handler
     */
    initMouseMoveHandler: function() {
      this.canvas.on('mouse:move', this.mouseMoveHandler);
    },

    /**
     * @private
     */
    mouseMoveHandler: function(options) {
      if (!this.__isMousedown || !this.isEditing) {
        return;
      }

      var newSelectionStart = this.getSelectionStartFromPointer(options.e),
          currentStart = this.selectionStart,
          currentEnd = this.selectionEnd;
      if (
        (newSelectionStart !== this.__selectionStartOnMouseDown || currentStart
=== currentEnd)
          &&
        (currentStart === newSelectionStart || currentEnd === newSelectionStart)
      ) {
        return;
      }
      if (newSelectionStart > this.__selectionStartOnMouseDown) {
        this.selectionStart = this.__selectionStartOnMouseDown;
        this.selectionEnd = newSelectionStart;
      }
      else {
        this.selectionStart = newSelectionStart;
```

```
        this.selectionEnd = this.__selectionStartOnMouseDown;
      }
      if (this.selectionStart !== currentStart || this.selectionEnd !==
currentEnd) {
        this.restartCursorIfNeeded();
        this._fireSelectionChanged();
        this._updateTextarea();
        this.renderCursorOrSelection();
      }
    },

    /**
     * @private
     */
    _setEditingProps: function() {
      this.hoverCursor = 'text';

      if (this.canvas) {
        this.canvas.defaultCursor = this.canvas.moveCursor = 'text';
      }

      this.borderColor = this.editingBorderColor;

      this.hasControls = this.selectable = false;
      this.lockMovementX = this.lockMovementY = true;
    },

    /**
     * @private
     */
    _updateTextarea: function() {
      if (!this.hiddenTextarea || this.inCompositionMode) {
        return;
      }
      this.cursorOffsetCache = { };
      this.hiddenTextarea.value = this.text;
      this.hiddenTextarea.selectionStart = this.selectionStart;
      this.hiddenTextarea.selectionEnd = this.selectionEnd;
      if (this.selectionStart === this.selectionEnd) {
        var style = this._calcTextareaPosition();
        this.hiddenTextarea.style.left = style.left;
        this.hiddenTextarea.style.top = style.top;
        this.hiddenTextarea.style.fontSize = style.fontSize;
      }
    },

    /**
     * @private
     * @return {Object} style contains style for hiddenTextarea
     */
    _calcTextareaPosition: function() {
      if (!this.canvas) {
        return { x: 1, y: 1 };
      }
      var chars = this.text.split(''),
          boundaries = this._getCursorBoundaries(chars, 'cursor'),
          cursorLocation = this.get2DCursorLocation(),
          lineIndex = cursorLocation.lineIndex,
          charIndex = cursorLocation.charIndex,
          charHeight = this.getCurrentCharFontSize(lineIndex, charIndex),
          leftOffset = boundaries.leftOffset,
          m = this.calcTransformMatrix(),
          p = {
            x: boundaries.left + leftOffset,
```

```
        y: boundaries.top + boundaries.topOffset + charHeight
      },
      upperCanvas = this.canvas.upperCanvasEl,
      maxWidth = upperCanvas.width - charHeight,
      maxHeight = upperCanvas.height - charHeight;

  p = fabric.util.transformPoint(p, m);
  p = fabric.util.transformPoint(p, this.canvas.viewportTransform);

  if (p.x < 0) {
    p.x = 0;
  }
  if (p.x > maxWidth) {
    p.x = maxWidth;
  }
  if (p.y < 0) {
    p.y = 0;
  }
  if (p.y > maxHeight) {
    p.y = maxHeight;
  }

  // add canvas offset on document
  p.x += this.canvas._offset.left;
  p.y += this.canvas._offset.top;

  return { left: p.x + 'px', top: p.y + 'px', fontSize: charHeight };
},

/**
 * @private
 */
_saveEditingProps: function() {
  this._savedProps = {
    hasControls: this.hasControls,
    borderColor: this.borderColor,
    lockMovementX: this.lockMovementX,
    lockMovementY: this.lockMovementY,
    hoverCursor: this.hoverCursor,
    defaultCursor: this.canvas && this.canvas.defaultCursor,
    moveCursor: this.canvas && this.canvas.moveCursor
  };
},

/**
 * @private
 */
_restoreEditingProps: function() {
  if (!this._savedProps) {
    return;
  }

  this.hoverCursor = this._savedProps.overCursor;
  this.hasControls = this._savedProps.hasControls;
  this.borderColor = this._savedProps.borderColor;
  this.lockMovementX = this._savedProps.lockMovementX;
  this.lockMovementY = this._savedProps.lockMovementY;

  if (this.canvas) {
    this.canvas.defaultCursor = this._savedProps.defaultCursor;
    this.canvas.moveCursor = this._savedProps.moveCursor;
  }
},
```

```
    /**
     * Exits from editing state
     * @return {fabric.IText} thisArg
     * @chainable
     */
    exitEditing: function() {
      var isTextChanged = (this._textBeforeEdit !== this.text);
      this.selected = false;
      this.isEditing = false;
      this.selectable = true;

      this.selectionEnd = this.selectionStart;

      if (this.hiddenTextarea) {
        this.hiddenTextarea.blur && this.hiddenTextarea.blur();
        this.canvas &&
this.hiddenTextarea.parentNode.removeChild(this.hiddenTextarea);
        this.hiddenTextarea = null;
      }

      this.abortCursorAnimation();
      this._restoreEditingProps();
      this._currentCursorOpacity = 0;

      this.fire('editing:exited');
      isTextChanged && this.fire('modified');
      if (this.canvas) {
        this.canvas.off('mouse:move', this.mouseMoveHandler);
        this.canvas.fire('text:editing:exited', { target: this });
        isTextChanged && this.canvas.fire('object:modified', { target: this });
      }
      return this;
    },

    /**
     * @private
     */
    _removeExtraneousStyles: function() {
      for (var prop in this.styles) {
        if (!this._textLines[prop]) {
          delete this.styles[prop];
        }
      }
    },

    /**
     * @private
     */
    _removeCharsFromTo: function(start, end) {
      while (end !== start) {
        this._removeSingleCharAndStyle(start + 1);
        end--;
      }
      this.selectionStart = start;
      this.selectionEnd = start;
    },

    _removeSingleCharAndStyle: function(index) {
      var isBeginningOfLine = this.text[index - 1] === '\n',
          indexStyle        = isBeginningOfLine ? index : index - 1;
      this.removeStyleObject(isBeginningOfLine, indexStyle);
      this.text = this.text.slice(0, index - 1) +
        this.text.slice(index);
```

```
      this._textLines = this._splitTextIntoLines();
    },

    /**
     * Inserts characters where cursor is (replacing selection if one exists)
     * @param {String} _chars Characters to insert
     * @param {Boolean} useCopiedStyle use fabric.copiedTextStyle
     */
    insertChars: function(_chars, useCopiedStyle) {
      var style;

      if (this.selectionEnd - this.selectionStart > 1) {
        this._removeCharsFromTo(this.selectionStart, this.selectionEnd);
      }
      //short circuit for block paste
      if (!useCopiedStyle && this.isEmptyStyles()) {
        this.insertChar(_chars, false);
        return;
      }
      for (var i = 0, len = _chars.length; i < len; i++) {
        if (useCopiedStyle) {
          style = fabric.util.object.clone(fabric.copiedTextStyle[i], true);
        }
        this.insertChar(_chars[i], i < len - 1, style);
      }
    },

    /**
     * Inserts a character where cursor is
     * @param {String} _char Characters to insert
     * @param {Boolean} skipUpdate trigger rendering and updates at the end of
text insert
     * @param {Object} styleObject Style to be inserted for the new char
     */
    insertChar: function(_char, skipUpdate, styleObject) {
      var isEndOfLine = this.text[this.selectionStart] === '\n';
      this.text = this.text.slice(0, this.selectionStart) +
        _char + this.text.slice(this.selectionEnd);
      this._textLines = this._splitTextIntoLines();
      this.insertStyleObjects(_char, isEndOfLine, styleObject);
      this.selectionStart += _char.length;
      this.selectionEnd = this.selectionStart;
      if (skipUpdate) {
        return;
      }
      this._updateTextarea();
      this.setCoords();
      this._fireSelectionChanged();
      this.fire('changed');
      this.restartCursorIfNeeded();
      if (this.canvas) {
        this.canvas.fire('text:changed', { target: this });
        this.canvas.renderAll();
      }
    },

    restartCursorIfNeeded: function() {
      if (!this._currentTickState || this._currentTickState.isAborted
        || !this._currentTickCompleteState ||
this._currentTickCompleteState.isAborted
      ) {
        this.initDelayedCursor();
      }
    },
```

```
    /**
     * Inserts new style object
     * @param {Number} lineIndex Index of a line
     * @param {Number} charIndex Index of a char
     * @param {Boolean} isEndOfLine True if it's end of line
     */
    insertNewlineStyleObject: function(lineIndex, charIndex, isEndOfLine) {

      this.shiftLineStyles(lineIndex, +1);

      var currentCharStyle = {},
          newLineStyles    = {};

      if (this.styles[lineIndex] && this.styles[lineIndex][charIndex - 1]) {
        currentCharStyle = this.styles[lineIndex][charIndex - 1];
      }

      // if there's nothing after cursor,
      // we clone current char style onto the next (otherwise empty) line
      if (isEndOfLine && currentCharStyle) {
        newLineStyles[0] = clone(currentCharStyle);
        this.styles[lineIndex + 1] = newLineStyles;
      }
      // otherwise we clone styles of all chars
      // after cursor onto the next line, from the beginning
      else {
        var somethingAdded = false;
        for (var index in this.styles[lineIndex]) {
          var numIndex = parseInt(index, 10);
          if (numIndex >= charIndex) {
            somethingAdded = true;
            newLineStyles[numIndex - charIndex] = this.styles[lineIndex][index];
            // remove lines from the previous line since they're on a new line
now
            delete this.styles[lineIndex][index];
          }
        }
        somethingAdded && (this.styles[lineIndex + 1] = newLineStyles);
      }
      this._forceClearCache = true;
    },

    /**
     * Inserts style object for a given line/char index
     * @param {Number} lineIndex Index of a line
     * @param {Number} charIndex Index of a char
     * @param {Object} [style] Style object to insert, if given
     */
    insertCharStyleObject: function(lineIndex, charIndex, style) {

      var currentLineStyles       = this.styles[lineIndex],
          currentLineStylesCloned = clone(currentLineStyles);

      if (charIndex === 0 && !style) {
        charIndex = 1;
      }

      // shift all char styles by 1 forward
      // 0,1,2,3 -> (charIndex=2) -> 0,1,3,4 -> (insert 2) -> 0,1,2,3,4
      for (var index in currentLineStylesCloned) {
        var numericIndex = parseInt(index, 10);

        if (numericIndex >= charIndex) {
```

```
          currentLineStyles[numericIndex + 1] =
currentLineStylesCloned[numericIndex];

          // only delete the style if there was nothing moved there
          if (!currentLineStylesCloned[numericIndex - 1]) {
            delete currentLineStyles[numericIndex];
          }
        }
      }
      var newStyle = style || clone(currentLineStyles[charIndex - 1]);
      newStyle && (this.styles[lineIndex][charIndex] = newStyle);
      this._forceClearCache = true;
    },

    /**
     * Inserts style object(s)
     * @param {String} _chars Characters at the location where style is inserted
     * @param {Boolean} isEndOfLine True if it's end of line
     * @param {Object} [styleObject] Style to insert
     */
    insertStyleObjects: function(_chars, isEndOfLine, styleObject) {
      // removed shortcircuit over isEmptyStyles

      var cursorLocation = this.get2DCursorLocation(),
          lineIndex      = cursorLocation.lineIndex,
          charIndex      = cursorLocation.charIndex;

      if (!this._getLineStyle(lineIndex)) {
        this._setLineStyle(lineIndex, {});
      }

      if (_chars === '\n') {
        this.insertNewlineStyleObject(lineIndex, charIndex, isEndOfLine);
      }
      else {
        this.insertCharStyleObject(lineIndex, charIndex, styleObject);
      }
    },

    /**
     * Shifts line styles up or down
     * @param {Number} lineIndex Index of a line
     * @param {Number} offset Can be -1 or +1
     */
    shiftLineStyles: function(lineIndex, offset) {
      // shift all line styles by 1 upward or downward
      var clonedStyles = clone(this.styles);
      for (var line in clonedStyles) {
        var numericLine = parseInt(line, 10);
        if (numericLine <= lineIndex) {
          delete clonedStyles[numericLine];
        }
      }
      for (var line in this.styles) {
        var numericLine = parseInt(line, 10);
        if (numericLine > lineIndex) {
          this.styles[numericLine + offset] = clonedStyles[numericLine];
          if (!clonedStyles[numericLine - offset]) {
            delete this.styles[numericLine];
          }
        }
      }
      //TODO: evaluate if delete old style lines with offset -1
    },
```

```
    /**
     * Removes style object
     * @param {Boolean} isBeginningOfLine True if cursor is at the beginning of
line
     * @param {Number} [index] Optional index. When not given, current
selectionStart is used.
     */
    removeStyleObject: function(isBeginningOfLine, index) {

      var cursorLocation = this.get2DCursorLocation(index),
          lineIndex      = cursorLocation.lineIndex,
          charIndex      = cursorLocation.charIndex;

      this._removeStyleObject(isBeginningOfLine, cursorLocation, lineIndex,
charIndex);
    },

    _getTextOnPreviousLine: function(lIndex) {
      return this._textLines[lIndex - 1];
    },

    _removeStyleObject: function(isBeginningOfLine, cursorLocation, lineIndex,
charIndex) {

      if (isBeginningOfLine) {
        var textOnPreviousLine     =
this._getTextOnPreviousLine(cursorLocation.lineIndex),
            newCharIndexOnPrevLine = textOnPreviousLine ?
textOnPreviousLine.length : 0;

        if (!this.styles[lineIndex - 1]) {
          this.styles[lineIndex - 1] = {};
        }
        for (charIndex in this.styles[lineIndex]) {
          this.styles[lineIndex - 1][parseInt(charIndex, 10) +
newCharIndexOnPrevLine]
            = this.styles[lineIndex][charIndex];
        }
        this.shiftLineStyles(cursorLocation.lineIndex, -1);
      }
      else {
        var currentLineStyles = this.styles[lineIndex];

        if (currentLineStyles) {
          delete currentLineStyles[charIndex];
        }
        var currentLineStylesCloned = clone(currentLineStyles);
        // shift all styles by 1 backwards
        for (var i in currentLineStylesCloned) {
          var numericIndex = parseInt(i, 10);
          if (numericIndex >= charIndex && numericIndex !== 0) {
            currentLineStyles[numericIndex - 1] =
currentLineStylesCloned[numericIndex];
            delete currentLineStyles[numericIndex];
          }
        }
      }
    },

    /**
     * Inserts new line
     */
    insertNewline: function() {
```

```
      this.insertChars('\n');
    },

    /**
     * Set the selectionStart and selectionEnd according to the ne postion of
cursor
     * mimic the key - mouse navigation when shift is pressed.
     */
    setSelectionStartEndWithShift: function(start, end, newSelection) {
      if (newSelection <= start) {
        if (end === start) {
          this._selectionDirection = 'left';
        }
        else if (this._selectionDirection === 'right') {
          this._selectionDirection = 'left';
          this.selectionEnd = start;
        }
        this.selectionStart = newSelection;
      }
      else if (newSelection > start && newSelection < end) {
        if (this._selectionDirection === 'right') {
          this.selectionEnd = newSelection;
        }
        else {
          this.selectionStart = newSelection;
        }
      }
      else {
        // newSelection is > selection start and end
        if (end === start) {
          this._selectionDirection = 'right';
        }
        else if (this._selectionDirection === 'left') {
          this._selectionDirection = 'right';
          this.selectionStart = end;
        }
        this.selectionEnd = newSelection;
      }
    },

    setSelectionInBoundaries: function() {
      var length = this.text.length;
      if (this.selectionStart > length) {
        this.selectionStart = length;
      }
      else if (this.selectionStart < 0) {
        this.selectionStart = 0;
      }
      if (this.selectionEnd > length) {
        this.selectionEnd = length;
      }
      else if (this.selectionEnd < 0) {
        this.selectionEnd = 0;
      }
    }
  });
})();


fabric.util.object.extend(fabric.IText.prototype, /** @lends
fabric.IText.prototype */ {
  /**
   * Initializes "dbclick" event handler
   */
```

```
initDoubleClickSimulation: function() {

  // for double click
  this.__lastClickTime = +new Date();

  // for triple click
  this.__lastLastClickTime = +new Date();

  this.__lastPointer = { };

  this.on('mousedown', this.onMouseDown.bind(this));
},

onMouseDown: function(options) {

  this.__newClickTime = +new Date();
  var newPointer = this.canvas.getPointer(options.e);

  if (this.isTripleClick(newPointer, options.e)) {
    this.fire('tripleclick', options);
    this._stopEvent(options.e);
  }
  else if (this.isDoubleClick(newPointer)) {
    this.fire('dblclick', options);
    this._stopEvent(options.e);
  }

  this.__lastLastClickTime = this.__lastClickTime;
  this.__lastClickTime = this.__newClickTime;
  this.__lastPointer = newPointer;
  this.__lastIsEditing = this.isEditing;
  this.__lastSelected = this.selected;
},

isDoubleClick: function(newPointer) {
  return this.__newClickTime - this.__lastClickTime < 500 &&
      this.__lastPointer.x === newPointer.x &&
      this.__lastPointer.y === newPointer.y && this.__lastIsEditing;
},

isTripleClick: function(newPointer) {
  return this.__newClickTime - this.__lastClickTime < 500 &&
      this.__lastClickTime - this.__lastLastClickTime < 500 &&
      this.__lastPointer.x === newPointer.x &&
      this.__lastPointer.y === newPointer.y;
},

/**
 * @private
 */
_stopEvent: function(e) {
  e.preventDefault && e.preventDefault();
  e.stopPropagation && e.stopPropagation();
},

/**
 * Initializes event handlers related to cursor or selection
 */
initCursorSelectionHandlers: function() {
  this.initMousedownHandler();
  this.initMouseupHandler();
  this.initClicks();
},
```

```
  /**
   * Initializes double and triple click event handlers
   */
  initClicks: function() {
    this.on('dblclick', function(options) {
      this.selectWord(this.getSelectionStartFromPointer(options.e));
    });
    this.on('tripleclick', function(options) {
      this.selectLine(this.getSelectionStartFromPointer(options.e));
    });
  },

  /**
   * Initializes "mousedown" event handler
   */
  initMousedownHandler: function() {
    this.on('mousedown', function(options) {
      if (!this.editable || (options.e.button && options.e.button !== 1)) {
        return;
      }
      var pointer = this.canvas.getPointer(options.e);
      this.__mousedownX = pointer.x;
      this.__mousedownY = pointer.y;
      this.__isMousedown = true;

      if (this.selected) {
        this.setCursorByClick(options.e);
      }

      if (this.isEditing) {
        this.__selectionStartOnMouseDown = this.selectionStart;
        if (this.selectionStart === this.selectionEnd) {
          this.abortCursorAnimation();
        }
        this.renderCursorOrSelection();
      }
    });
  },

  /**
   * @private
   */
  _isObjectMoved: function(e) {
    var pointer = this.canvas.getPointer(e);

    return this.__mousedownX !== pointer.x ||
           this.__mousedownY !== pointer.y;
  },

  /**
   * Initializes "mouseup" event handler
   */
  initMouseupHandler: function() {
    this.on('mouseup', function(options) {
      this.__isMousedown = false;
      if (!this.editable || this._isObjectMoved(options.e) || (options.e.button
&& options.e.button !== 1)) {
        return;
      }

      if (this.__lastSelected && !this.__corner) {
        this.enterEditing(options.e);
        if (this.selectionStart === this.selectionEnd) {
          this.initDelayedCursor(true);
```

```
        }
        else {
          this.renderCursorOrSelection();
        }
      }
      this.selected = true;
    });
  },

  /**
   * Changes cursor location in a text depending on passed pointer (x/y) object
   * @param {Event} e Event object
   */
  setCursorByClick: function(e) {
    var newSelection = this.getSelectionStartFromPointer(e),
        start = this.selectionStart, end = this.selectionEnd;
    if (e.shiftKey) {
      this.setSelectionStartEndWithShift(start, end, newSelection);
    }
    else {
      this.selectionStart = newSelection;
      this.selectionEnd = newSelection;
    }
    if (this.isEditing) {
      this._fireSelectionChanged();
      this._updateTextarea();
    }
  },

  /**
   * Returns index of a character corresponding to where an object was clicked
   * @param {Event} e Event object
   * @return {Number} Index of a character
   */
  getSelectionStartFromPointer: function(e) {
    var mouseOffset = this.getLocalPointer(e),
        prevWidth = 0,
        width = 0,
        height = 0,
        charIndex = 0,
        newSelectionStart,
        line;

    for (var i = 0, len = this._textLines.length; i < len; i++) {
      line = this._textLines[i];
      height += this._getHeightOfLine(this.ctx, i) * this.scaleY;

      var widthOfLine = this._getLineWidth(this.ctx, i),
          lineLeftOffset = this._getLineLeftOffset(widthOfLine);

      width = lineLeftOffset * this.scaleX;

      for (var j = 0, jlen = line.length; j < jlen; j++) {

        prevWidth = width;

        width += this._getWidthOfChar(this.ctx, line[j], i, this.flipX ? jlen -
j : j) *
                 this.scaleX;

        if (height <= mouseOffset.y || width <= mouseOffset.x) {
          charIndex++;
          continue;
        }
```

```
        return this._getNewSelectionStartFromOffset(
          mouseOffset, prevWidth, width, charIndex + i, jlen);
      }

      if (mouseOffset.y < height) {
        //this happens just on end of lines.
        return this._getNewSelectionStartFromOffset(
          mouseOffset, prevWidth, width, charIndex + i - 1, jlen);
      }
    }

    // clicked somewhere after all chars, so set at the end
    if (typeof newSelectionStart === 'undefined') {
      return this.text.length;
    }
  },

  /**
   * @private
   */
  _getNewSelectionStartFromOffset: function(mouseOffset, prevWidth, width,
index, jlen) {

    var distanceBtwLastCharAndCursor = mouseOffset.x - prevWidth,
        distanceBtwNextCharAndCursor = width - mouseOffset.x,
        offset = distanceBtwNextCharAndCursor > distanceBtwLastCharAndCursor ? 0
: 1,
        newSelectionStart = index + offset;

    // if object is horizontally flipped, mirror cursor location from the end
    if (this.flipX) {
      newSelectionStart = jlen - newSelectionStart;
    }

    if (newSelectionStart > this.text.length) {
      newSelectionStart = this.text.length;
    }

    return newSelectionStart;
  }
});


fabric.util.object.extend(fabric.IText.prototype, /** @lends
fabric.IText.prototype */ {

  /**
   * Initializes hidden textarea (needed to bring up keyboard in iOS)
   */
  initHiddenTextarea: function() {
    this.hiddenTextarea = fabric.document.createElement('textarea');
    this.hiddenTextarea.setAttribute('autocapitalize', 'off');
    this.hiddenTextarea.setAttribute('autocorrect', 'off');
    this.hiddenTextarea.setAttribute('autocomplete', 'off');
    this.hiddenTextarea.setAttribute('spellcheck', 'false');
    this.hiddenTextarea.setAttribute('data-fabric-hiddentextarea', '');
    this.hiddenTextarea.setAttribute('wrap', 'off');
    var style = this._calcTextareaPosition();
    this.hiddenTextarea.style.cssText = 'position: absolute; top: ' + style.top
+
    '; left: ' + style.left + '; z-index: -999; opacity: 0; width: 1px; height:
1px; font-size: 1px;' +
```

```
    ' line-height: 1px; padding-top: ' + style.fontSize + ';';
    fabric.document.body.appendChild(this.hiddenTextarea);

    fabric.util.addListener(this.hiddenTextarea, 'keydown',
this.onKeyDown.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'keyup',
this.onKeyUp.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'input',
this.onInput.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'copy', this.copy.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'cut', this.cut.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'paste',
this.paste.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'compositionstart',
this.onCompositionStart.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'compositionupdate',
this.onCompositionUpdate.bind(this));
    fabric.util.addListener(this.hiddenTextarea, 'compositionend',
this.onCompositionEnd.bind(this));

    if (!this._clickHandlerInitialized && this.canvas) {
      fabric.util.addListener(this.canvas.upperCanvasEl, 'click',
this.onClick.bind(this));
      this._clickHandlerInitialized = true;
    }
  },

  /**
   * For functionalities on keyDown
   * Map a special key to a function of the instance/prototype
   * If you need different behaviour for ESC or TAB or arrows, you have to
change
   * this map setting the name of a function that you build on the fabric.Itext
or
   * your prototype.
   * the map change will affect all Instances unless you need for only some text
Instances
   * in that case you have to clone this object and assign your Instance.
   * this.keysMap = fabric.util.object.clone(this.keysMap);
   * The function must be in fabric.Itext.prototype.myFunction And will receive
event as args[0]
   */
  keysMap: {
    8:  'removeChars',
    9:  'exitEditing',
    27: 'exitEditing',
    13: 'insertNewline',
    33: 'moveCursorUp',
    34: 'moveCursorDown',
    35: 'moveCursorRight',
    36: 'moveCursorLeft',
    37: 'moveCursorLeft',
    38: 'moveCursorUp',
    39: 'moveCursorRight',
    40: 'moveCursorDown',
    46: 'forwardDelete'
  },

  /**
   * For functionalities on keyUp + ctrl || cmd
   */
  ctrlKeysMapUp: {
    67: 'copy',
```

```
    88: 'cut'
  },

  /**
   * For functionalities on keyDown + ctrl || cmd
   */
  ctrlKeysMapDown: {
    65: 'selectAll'
  },

  onClick: function() {
    // No need to trigger click event here, focus is enough to have the keyboard
appear on Android
    this.hiddenTextarea && this.hiddenTextarea.focus();
  },

  /**
   * Handles keyup event
   * @param {Event} e Event object
   */
  onKeyDown: function(e) {
    if (!this.isEditing) {
      return;
    }
    if (e.keyCode in this.keysMap) {
      this[this.keysMap[e.keyCode]](e);
    }
    else if ((e.keyCode in this.ctrlKeysMapDown) && (e.ctrlKey || e.metaKey)) {
      this[this.ctrlKeysMapDown[e.keyCode]](e);
    }
    else {
      return;
    }
    e.stopImmediatePropagation();
    e.preventDefault();
    if (e.keyCode >= 33 && e.keyCode <= 40) {
      // if i press an arrow key just update selection
      this.clearContextTop();
      this.renderCursorOrSelection();
    }
    else {
      this.canvas && this.canvas.renderAll();
    }
  },

  /**
   * Handles keyup event
   * We handle KeyUp because ie11 and edge have difficulties copy/pasting
   * if a copy/cut event fired, keyup is dismissed
   * @param {Event} e Event object
   */
  onKeyUp: function(e) {
    if (!this.isEditing || this._copyDone) {
      this._copyDone = false;
      return;
    }
    if ((e.keyCode in this.ctrlKeysMapUp) && (e.ctrlKey || e.metaKey)) {
      this[this.ctrlKeysMapUp[e.keyCode]](e);
    }
    else {
      return;
    }
    e.stopImmediatePropagation();
    e.preventDefault();
```

```
  this.canvas && this.canvas.renderAll();
},

/**
 * Handles onInput event
 * @param {Event} e Event object
 */
onInput: function(e) {
  if (!this.isEditing || this.inCompositionMode) {
    return;
  }
  var offset = this.selectionStart || 0,
      offsetEnd = this.selectionEnd || 0,
      textLength = this.text.length,
      newTextLength = this.hiddenTextarea.value.length,
      diff, charsToInsert, start;
  if (newTextLength > textLength) {
    //we added some character
    start = this._selectionDirection === 'left' ? offsetEnd : offset;
    diff = newTextLength - textLength;
    charsToInsert = this.hiddenTextarea.value.slice(start, start + diff);
  }
  else {
    //we selected a portion of text and then input something else.
    //Internet explorer does not trigger this else
    diff = newTextLength - textLength + offsetEnd - offset;
    charsToInsert = this.hiddenTextarea.value.slice(offset, offset + diff);
  }
  this.insertChars(charsToInsert);
  e.stopPropagation();
},

/**
 * Composition start
 */
onCompositionStart: function() {
  this.inCompositionMode = true;
  this.prevCompositionLength = 0;
  this.compositionStart = this.selectionStart;
},

/**
 * Composition end
 */
onCompositionEnd: function() {
  this.inCompositionMode = false;
},

/**
 * Composition update
 */
onCompositionUpdate: function(e) {
  var data = e.data;
  this.selectionStart = this.compositionStart;
  this.selectionEnd = this.selectionEnd === this.selectionStart ?
    this.compositionStart + this.prevCompositionLength : this.selectionEnd;
  this.insertChars(data, false);
  this.prevCompositionLength = data.length;
},

/**
 * Forward delete
 */
forwardDelete: function(e) {
```

```
    if (this.selectionStart === this.selectionEnd) {
      if (this.selectionStart === this.text.length) {
        return;
      }
      this.moveCursorRight(e);
    }
    this.removeChars(e);
  },

  /**
   * Copies selected text
   * @param {Event} e Event object
   */
  copy: function(e) {
    if (this.selectionStart === this.selectionEnd) {
      //do not cut-copy if no selection
      return;
    }
    var selectedText = this.getSelectedText(),
        clipboardData = this._getClipboardData(e);

    // Check for backward compatibility with old browsers
    if (clipboardData) {
      clipboardData.setData('text', selectedText);
    }

    fabric.copiedText = selectedText;
    fabric.copiedTextStyle = this.getSelectionStyles(this.selectionStart,
this.selectionEnd);
    e.stopImmediatePropagation();
    e.preventDefault();
    this._copyDone = true;
  },

  /**
   * Pastes text
   * @param {Event} e Event object
   */
  paste: function(e) {
    var copiedText = null,
        clipboardData = this._getClipboardData(e),
        useCopiedStyle = true;

    // Check for backward compatibility with old browsers
    if (clipboardData) {
      copiedText = clipboardData.getData('text').replace(/\r/g, '');
      if (!fabric.copiedTextStyle || fabric.copiedText !== copiedText) {
        useCopiedStyle = false;
      }
    }
    else {
      copiedText = fabric.copiedText;
    }

    if (copiedText) {
      this.insertChars(copiedText, useCopiedStyle);
    }
    e.stopImmediatePropagation();
    e.preventDefault();
  },

  /**
   * Cuts text
   * @param {Event} e Event object
```

```
   */
  cut: function(e) {
    if (this.selectionStart === this.selectionEnd) {
      return;
    }

    this.copy(e);
    this.removeChars(e);
  },

  /**
   * @private
   * @param {Event} e Event object
   * @return {Object} Clipboard data object
   */
  _getClipboardData: function(e) {
    return (e && e.clipboardData) || fabric.window.clipboardData;
  },

  /**
   * Finds the width in pixels before the cursor on the same line
   * @private
   * @param {Number} lineIndex
   * @param {Number} charIndex
   * @return {Number} widthBeforeCursor width before cursor
   */
  _getWidthBeforeCursor: function(lineIndex, charIndex) {
    var textBeforeCursor = this._textLines[lineIndex].slice(0, charIndex),
        widthOfLine = this._getLineWidth(this.ctx, lineIndex),
        widthBeforeCursor = this._getLineLeftOffset(widthOfLine), _char;

    for (var i = 0, len = textBeforeCursor.length; i < len; i++) {
      _char = textBeforeCursor[i];
      widthBeforeCursor += this._getWidthOfChar(this.ctx, _char, lineIndex, i);
    }
    return widthBeforeCursor;
  },

  /**
   * Gets start offset of a selection
   * @param {Event} e Event object
   * @param {Boolean} isRight
   * @return {Number}
   */
  getDownCursorOffset: function(e, isRight) {
    var selectionProp = this._getSelectionForOffset(e, isRight),
        cursorLocation = this.get2DCursorLocation(selectionProp),
        lineIndex = cursorLocation.lineIndex;
    // if on last line, down cursor goes to end of line
    if (lineIndex === this._textLines.length - 1 || e.metaKey || e.keyCode ===
34) {
      // move to the end of a text
      return this.text.length - selectionProp;
    }
    var charIndex = cursorLocation.charIndex,
        widthBeforeCursor = this._getWidthBeforeCursor(lineIndex, charIndex),
        indexOnOtherLine = this._getIndexOnLine(lineIndex + 1,
widthBeforeCursor),
        textAfterCursor = this._textLines[lineIndex].slice(charIndex);

    return textAfterCursor.length + indexOnOtherLine + 2;
  },

  /**
```

```
   * private
   * Helps finding if the offset should be counted from Start or End
   * @param {Event} e Event object
   * @param {Boolean} isRight
   * @return {Number}
   */
  _getSelectionForOffset: function(e, isRight) {
    if (e.shiftKey && this.selectionStart !== this.selectionEnd && isRight) {
      return this.selectionEnd;
    }
    else {
      return this.selectionStart;
    }
  },

  /**
   * @param {Event} e Event object
   * @param {Boolean} isRight
   * @return {Number}
   */
  getUpCursorOffset: function(e, isRight) {
    var selectionProp = this._getSelectionForOffset(e, isRight),
        cursorLocation = this.get2DCursorLocation(selectionProp),
        lineIndex = cursorLocation.lineIndex;
    if (lineIndex === 0 || e.metaKey || e.keyCode === 33) {
      // if on first line, up cursor goes to start of line
      return -selectionProp;
    }
    var charIndex = cursorLocation.charIndex,
        widthBeforeCursor = this._getWidthBeforeCursor(lineIndex, charIndex),
        indexOnOtherLine = this._getIndexOnLine(lineIndex - 1,
widthBeforeCursor),
        textBeforeCursor = this._textLines[lineIndex].slice(0, charIndex);
    // return a negative offset
    return -this._textLines[lineIndex - 1].length + indexOnOtherLine -
textBeforeCursor.length;
  },

  /**
   * find for a given width it founds the matching character.
   * @private
   */
  _getIndexOnLine: function(lineIndex, width) {

    var widthOfLine = this._getLineWidth(this.ctx, lineIndex),
        textOnLine = this._textLines[lineIndex],
        lineLeftOffset = this._getLineLeftOffset(widthOfLine),
        widthOfCharsOnLine = lineLeftOffset,
        indexOnLine = 0,
        foundMatch;

    for (var j = 0, jlen = textOnLine.length; j < jlen; j++) {

      var _char = textOnLine[j],
          widthOfChar = this._getWidthOfChar(this.ctx, _char, lineIndex, j);

      widthOfCharsOnLine += widthOfChar;

      if (widthOfCharsOnLine > width) {

        foundMatch = true;

        var leftEdge = widthOfCharsOnLine - widthOfChar,
            rightEdge = widthOfCharsOnLine,
```

```
            offsetFromLeftEdge = Math.abs(leftEdge - width),
            offsetFromRightEdge = Math.abs(rightEdge - width);

        indexOnLine = offsetFromRightEdge < offsetFromLeftEdge ? j : (j - 1);

        break;
      }
    }

    // reached end
    if (!foundMatch) {
      indexOnLine = textOnLine.length - 1;
    }

    return indexOnLine;
  },


  /**
   * Moves cursor down
   * @param {Event} e Event object
   */
  moveCursorDown: function(e) {
    if (this.selectionStart >= this.text.length && this.selectionEnd >=
this.text.length) {
      return;
    }
    this._moveCursorUpOrDown('Down', e);
  },

  /**
   * Moves cursor up
   * @param {Event} e Event object
   */
  moveCursorUp: function(e) {
    if (this.selectionStart === 0 && this.selectionEnd === 0) {
      return;
    }
    this._moveCursorUpOrDown('Up', e);
  },

  /**
   * Moves cursor up or down, fires the events
   * @param {String} direction 'Up' or 'Down'
   * @param {Event} e Event object
   */
  _moveCursorUpOrDown: function(direction, e) {
    // getUpCursorOffset
    // getDownCursorOffset
    var action = 'get' + direction + 'CursorOffset',
        offset = this[action](e, this._selectionDirection === 'right');
    if (e.shiftKey) {
      this.moveCursorWithShift(offset);
    }
    else {
      this.moveCursorWithoutShift(offset);
    }
    if (offset !== 0) {
      this.setSelectionInBoundaries();
      this.abortCursorAnimation();
      this._currentCursorOpacity = 1;
      this.initDelayedCursor();
      this._fireSelectionChanged();
      this._updateTextarea();
```

```
    }
  },

  /**
   * Moves cursor with shift
   * @param {Number} offset
   */
  moveCursorWithShift: function(offset) {
    var newSelection = this._selectionDirection === 'left'
    ? this.selectionStart + offset
    : this.selectionEnd + offset;
    this.setSelectionStartEndWithShift(this.selectionStart, this.selectionEnd,
newSelection);
    return offset !== 0;
  },

  /**
   * Moves cursor up without shift
   * @param {Number} offset
   */
  moveCursorWithoutShift: function(offset) {
    if (offset < 0) {
      this.selectionStart += offset;
      this.selectionEnd = this.selectionStart;
    }
    else {
      this.selectionEnd += offset;
      this.selectionStart = this.selectionEnd;
    }
    return offset !== 0;
  },

  /**
   * Moves cursor left
   * @param {Event} e Event object
   */
  moveCursorLeft: function(e) {
    if (this.selectionStart === 0 && this.selectionEnd === 0) {
      return;
    }
    this._moveCursorLeftOrRight('Left', e);
  },

  /**
   * @private
   * @return {Boolean} true if a change happened
   */
  _move: function(e, prop, direction) {
    var newValue;
    if (e.altKey) {
      newValue = this['findWordBoundary' + direction](this[prop]);
    }
    else if (e.metaKey || e.keyCode === 35 ||  e.keyCode === 36 ) {
      newValue = this['findLineBoundary' + direction](this[prop]);
    }
    else {
      this[prop] += direction === 'Left' ? -1 : 1;
      return true;
    }
    if (typeof newValue !== undefined && this[prop] !== newValue) {
      this[prop] = newValue;
      return true;
    }
  },
```

```
  /**
   * @private
   */
  _moveLeft: function(e, prop) {
    return this._move(e, prop, 'Left');
  },

  /**
   * @private
   */
  _moveRight: function(e, prop) {
    return this._move(e, prop, 'Right');
  },

  /**
   * Moves cursor left without keeping selection
   * @param {Event} e
   */
  moveCursorLeftWithoutShift: function(e) {
    var change = true;
    this._selectionDirection = 'left';

    // only move cursor when there is no selection,
    // otherwise we discard it, and leave cursor on same place
    if (this.selectionEnd === this.selectionStart && this.selectionStart !== 0)
{
      change = this._moveLeft(e, 'selectionStart');

    }
    this.selectionEnd = this.selectionStart;
    return change;
  },

  /**
   * Moves cursor left while keeping selection
   * @param {Event} e
   */
  moveCursorLeftWithShift: function(e) {
    if (this._selectionDirection === 'right' && this.selectionStart !==
this.selectionEnd) {
      return this._moveLeft(e, 'selectionEnd');
    }
    else if (this.selectionStart !== 0){
      this._selectionDirection = 'left';
      return this._moveLeft(e, 'selectionStart');
    }
  },

  /**
   * Moves cursor right
   * @param {Event} e Event object
   */
  moveCursorRight: function(e) {
    if (this.selectionStart >= this.text.length && this.selectionEnd >=
this.text.length) {
      return;
    }
    this._moveCursorLeftOrRight('Right', e);
  },

  /**
   * Moves cursor right or Left, fires event
   * @param {String} direction 'Left', 'Right'
```

```
   * @param {Event} e Event object
   */
  _moveCursorLeftOrRight: function(direction, e) {
    var actionName = 'moveCursor' + direction + 'With';
    this._currentCursorOpacity = 1;

    if (e.shiftKey) {
      actionName += 'Shift';
    }
    else {
      actionName += 'outShift';
    }
    if (this[actionName](e)) {
      this.abortCursorAnimation();
      this.initDelayedCursor();
      this._fireSelectionChanged();
      this._updateTextarea();
    }
  },

  /**
   * Moves cursor right while keeping selection
   * @param {Event} e
   */
  moveCursorRightWithShift: function(e) {
    if (this._selectionDirection === 'left' && this.selectionStart !==
this.selectionEnd) {
      return this._moveRight(e, 'selectionStart');
    }
    else if (this.selectionEnd !== this.text.length) {
      this._selectionDirection = 'right';
      return this._moveRight(e, 'selectionEnd');
    }
  },

  /**
   * Moves cursor right without keeping selection
   * @param {Event} e Event object
   */
  moveCursorRightWithoutShift: function(e) {
    var changed = true;
    this._selectionDirection = 'right';

    if (this.selectionStart === this.selectionEnd) {
      changed = this._moveRight(e, 'selectionStart');
      this.selectionEnd = this.selectionStart;
    }
    else {
      this.selectionStart = this.selectionEnd;
    }
    return changed;
  },

  /**
   * Removes characters selected by selection
   * @param {Event} e Event object
   */
  removeChars: function(e) {
    if (this.selectionStart === this.selectionEnd) {
      this._removeCharsNearCursor(e);
    }
    else {
      this._removeCharsFromTo(this.selectionStart, this.selectionEnd);
    }
```

```
    this.set('dirty', true);
    this.setSelectionEnd(this.selectionStart);

    this._removeExtraneousStyles();

    this.canvas && this.canvas.renderAll();

    this.setCoords();
    this.fire('changed');
    this.canvas && this.canvas.fire('text:changed', { target: this });
  },

  /**
   * @private
   * @param {Event} e Event object
   */
  _removeCharsNearCursor: function(e) {
    if (this.selectionStart === 0) {
      return;
    }
    if (e.metaKey) {
      // remove all till the start of current line
      var leftLineBoundary = this.findLineBoundaryLeft(this.selectionStart);

      this._removeCharsFromTo(leftLineBoundary, this.selectionStart);
      this.setSelectionStart(leftLineBoundary);
    }
    else if (e.altKey) {
      // remove all till the start of current word
      var leftWordBoundary = this.findWordBoundaryLeft(this.selectionStart);

      this._removeCharsFromTo(leftWordBoundary, this.selectionStart);
      this.setSelectionStart(leftWordBoundary);
    }
    else {
      this._removeSingleCharAndStyle(this.selectionStart);
      this.setSelectionStart(this.selectionStart - 1);
    }
  }
});


/* _TO_SVG_START_ */
(function() {
  var toFixed = fabric.util.toFixed,
      NUM_FRACTION_DIGITS = fabric.Object.NUM_FRACTION_DIGITS;

  fabric.util.object.extend(fabric.IText.prototype, /** @lends
fabric.IText.prototype */ {

    /**
     * @private
     */
    _setSVGTextLineText: function(lineIndex, textSpans, height, textLeftOffset,
textTopOffset, textBgRects) {
      if (!this._getLineStyle(lineIndex)) {
        fabric.Text.prototype._setSVGTextLineText.call(this,
          lineIndex, textSpans, height, textLeftOffset, textTopOffset);
      }
      else {
        this._setSVGTextLineChars(
          lineIndex, textSpans, height, textLeftOffset, textBgRects);
      }
```

```
    },

    /**
     * @private
     */
    _setSVGTextLineChars: function(lineIndex, textSpans, height, textLeftOffset,
textBgRects) {

      var chars = this._textLines[lineIndex],
          charOffset = 0,
          lineLeftOffset = this._getLineLeftOffset(this._getLineWidth(this.ctx,
lineIndex)) - this.width / 2,
          lineOffset = this._getSVGLineTopOffset(lineIndex),
          heightOfLine = this._getHeightOfLine(this.ctx, lineIndex);

      for (var i = 0, len = chars.length; i < len; i++) {
        var styleDecl = this._getStyleDeclaration(lineIndex, i) || { };

        textSpans.push(
          this._createTextCharSpan(
            chars[i], styleDecl, lineLeftOffset, lineOffset.lineTop +
lineOffset.offset, charOffset));

        var charWidth = this._getWidthOfChar(this.ctx, chars[i], lineIndex, i);

        if (styleDecl.textBackgroundColor) {
          textBgRects.push(
            this._createTextCharBg(
              styleDecl, lineLeftOffset, lineOffset.lineTop, heightOfLine,
charWidth, charOffset));
        }

        charOffset += charWidth;
      }
    },

    /**
     * @private
     */
    _getSVGLineTopOffset: function(lineIndex) {
      var lineTopOffset = 0, lastHeight = 0;
      for (var j = 0; j < lineIndex; j++) {
        lineTopOffset += this._getHeightOfLine(this.ctx, j);
      }
      lastHeight = this._getHeightOfLine(this.ctx, j);
      return {
        lineTop: lineTopOffset,
        offset: (this._fontSizeMult - this._fontSizeFraction) * lastHeight /
(this.lineHeight * this._fontSizeMult)
      };
    },

    /**
     * @private
     */
    _createTextCharBg: function(styleDecl, lineLeftOffset, lineTopOffset,
heightOfLine, charWidth, charOffset) {
      return [
        '\t\t<rect fill="', styleDecl.textBackgroundColor,
        '" x="', toFixed(lineLeftOffset + charOffset, NUM_FRACTION_DIGITS),
        '" y="', toFixed(lineTopOffset - this.height / 2, NUM_FRACTION_DIGITS),
        '" width="', toFixed(charWidth, NUM_FRACTION_DIGITS),
        '" height="', toFixed(heightOfLine / this.lineHeight,
NUM_FRACTION_DIGITS),
```

```
        '"></rect>\n'
      ].join('');
    },

    /**
     * @private
     */
    _createTextCharSpan: function(_char, styleDecl, lineLeftOffset,
lineTopOffset, charOffset) {

      var fillStyles = this.getSvgStyles.call(fabric.util.object.extend({
        visible: true,
        fill: this.fill,
        stroke: this.stroke,
        type: 'text',
        getSvgFilter: fabric.Object.prototype.getSvgFilter
      }, styleDecl));

      return [
        '\t\t\t<tspan x="', toFixed(lineLeftOffset + charOffset,
NUM_FRACTION_DIGITS), '" y="',
        toFixed(lineTopOffset - this.height / 2, NUM_FRACTION_DIGITS), '" ',
          (styleDecl.fontFamily ? 'font-family="' +
styleDecl.fontFamily.replace(/"/g, '\'') + '" ' : ''),
          (styleDecl.fontSize ? 'font-size="' + styleDecl.fontSize + '" ' : ''),
          (styleDecl.fontStyle ? 'font-style="' + styleDecl.fontStyle + '" ' :
''),
          (styleDecl.fontWeight ? 'font-weight="' + styleDecl.fontWeight + '"
' : ''),
          (styleDecl.textDecoration ? 'text-decoration="' +
styleDecl.textDecoration + '" ' : ''),
        'style="', fillStyles, '">',
        fabric.util.string.escapeXml(_char),
        '</tspan>\n'
      ].join('');
    },
  });
})();
/* _TO_SVG_END_ */


(function(global) {

  'use strict';

  var fabric = global.fabric || (global.fabric = {});

  /**
   * Textbox class, based on IText, allows the user to resize the text rectangle
   * and wraps lines automatically. Textboxes have their Y scaling locked, the
   * user can only change width. Height is adjusted automatically based on the
   * wrapping of lines.
   * @class fabric.Textbox
   * @extends fabric.IText
   * @mixes fabric.Observable
   * @return {fabric.Textbox} thisArg
   * @see {@link fabric.Textbox#initialize} for constructor definition
   */
  fabric.Textbox = fabric.util.createClass(fabric.IText, fabric.Observable, {

    /**
     * Type of an object
     * @type String
     * @default
```

```
     */
    type: 'textbox',

    /**
     * Minimum width of textbox, in pixels.
     * @type Number
     * @default
     */
    minWidth: 20,

    /**
     * Minimum calculated width of a textbox, in pixels.
     * fixed to 2 so that an empty textbox cannot go to 0
     * and is still selectable without text.
     * @type Number
     * @default
     */
    dynamicMinWidth: 2,

    /**
     * Cached array of text wrapping.
     * @type Array
     */
    __cachedLines: null,

    /**
     * Override standard Object class values
     */
    lockScalingY: true,

    /**
     * Override standard Object class values
     */
    lockScalingFlip: true,

    /**
     * Override standard Object class values
     * Textbox needs this on false
     */
    noScaleCache: false,

    /**
     * Constructor. Some scaling related property values are forced. Visibility
     * of controls is also fixed; only the rotation and width controls are
     * made available.
     * @param {String} text Text string
     * @param {Object} [options] Options object
     * @return {fabric.Textbox} thisArg
     */
    initialize: function(text, options) {

      this.callSuper('initialize', text, options);
      this.setControlsVisibility(fabric.Textbox.getTextboxControlVisibility());
      this.ctx = this.objectCaching ? this._cacheContext :
fabric.util.createCanvasElement().getContext('2d');
      // add width to this list of props that effect line wrapping.
      this._dimensionAffectingProps.push('width');
    },

    /**
     * Unlike superclass's version of this function, Textbox does not update
     * its width.
     * @param {CanvasRenderingContext2D} ctx Context to use for measurements
     * @private
```

```
     * @override
     */
    _initDimensions: function(ctx) {
      if (this.__skipDimension) {
        return;
      }

      if (!ctx) {
        ctx = fabric.util.createCanvasElement().getContext('2d');
        this._setTextStyles(ctx);
        this.clearContextTop();
      }

      // clear dynamicMinWidth as it will be different after we re-wrap line
      this.dynamicMinWidth = 0;

      // wrap lines
      this._textLines = this._splitTextIntoLines(ctx);
      // if after wrapping, the width is smaller than dynamicMinWidth, change
the width and re-wrap
      if (this.dynamicMinWidth > this.width) {
        this._set('width', this.dynamicMinWidth);
      }

      // clear cache and re-calculate height
      this._clearCache();
      this.height = this._getTextHeight(ctx);
    },

    /**
     * Generate an object that translates the style object so that it is
     * broken up by visual lines (new lines and automatic wrapping).
     * The original text styles object is broken up by actual lines (new lines
only),
     * which is only sufficient for Text / IText
     * @private
     */
    _generateStyleMap: function() {
      var realLineCount     = 0,
          realLineCharCount = 0,
          charCount         = 0,
          map               = {};

      for (var i = 0; i < this._textLines.length; i++) {
        if (this.text[charCount] === '\n' && i > 0) {
          realLineCharCount = 0;
          charCount++;
          realLineCount++;
        }
        else if (this.text[charCount] === ' ' && i > 0) {
          // this case deals with space's that are removed from end of lines
when wrapping
          realLineCharCount++;
          charCount++;
        }

        map[i] = { line: realLineCount, offset: realLineCharCount };

        charCount += this._textLines[i].length;
        realLineCharCount += this._textLines[i].length;
      }

      return map;
    },
```

```
    /**
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @param {Boolean} [returnCloneOrEmpty=false]
     * @private
     */
    _getStyleDeclaration: function(lineIndex, charIndex, returnCloneOrEmpty) {
      if (this._styleMap) {
        var map = this._styleMap[lineIndex];
        if (!map) {
          return returnCloneOrEmpty ? { } : null;
        }
        lineIndex = map.line;
        charIndex = map.offset + charIndex;
      }
      return this.callSuper('_getStyleDeclaration', lineIndex, charIndex,
returnCloneOrEmpty);
    },

    /**
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @param {Object} style
     * @private
     */
    _setStyleDeclaration: function(lineIndex, charIndex, style) {
      var map = this._styleMap[lineIndex];
      lineIndex = map.line;
      charIndex = map.offset + charIndex;

      this.styles[lineIndex][charIndex] = style;
    },

    /**
     * @param {Number} lineIndex
     * @param {Number} charIndex
     * @private
     */
    _deleteStyleDeclaration: function(lineIndex, charIndex) {
      var map = this._styleMap[lineIndex];
      lineIndex = map.line;
      charIndex = map.offset + charIndex;

      delete this.styles[lineIndex][charIndex];
    },

    /**
     * @param {Number} lineIndex
     * @private
     */
    _getLineStyle: function(lineIndex) {
      var map = this._styleMap[lineIndex];
      return this.styles[map.line];
    },

    /**
     * @param {Number} lineIndex
     * @param {Object} style
     * @private
     */
    _setLineStyle: function(lineIndex, style) {
      var map = this._styleMap[lineIndex];
      this.styles[map.line] = style;
```

```
    },

    /**
     * @param {Number} lineIndex
     * @private
     */
    _deleteLineStyle: function(lineIndex) {
      var map = this._styleMap[lineIndex];
      delete this.styles[map.line];
    },

    /**
     * Wraps text using the 'width' property of Textbox. First this function
     * splits text on newlines, so we preserve newlines entered by the user.
     * Then it wraps each line using the width of the Textbox by calling
     * _wrapLine().
     * @param {CanvasRenderingContext2D} ctx Context to use for measurements
     * @param {String} text The string of text that is split into lines
     * @returns {Array} Array of lines
     */
    _wrapText: function(ctx, text) {
      var lines = text.split(this._reNewline), wrapped = [], i;

      for (i = 0; i < lines.length; i++) {
        wrapped = wrapped.concat(this._wrapLine(ctx, lines[i], i));
      }

      return wrapped;
    },

    /**
     * Helper function to measure a string of text, given its lineIndex and
charIndex offset
     *
     * @param {CanvasRenderingContext2D} ctx
     * @param {String} text
     * @param {number} lineIndex
     * @param {number} charOffset
     * @returns {number}
     * @private
     */
    _measureText: function(ctx, text, lineIndex, charOffset) {
      var width = 0;
      charOffset = charOffset || 0;
      for (var i = 0, len = text.length; i < len; i++) {
        width += this._getWidthOfChar(ctx, text[i], lineIndex, i + charOffset);
      }
      return width;
    },

    /**
     * Wraps a line of text using the width of the Textbox and a context.
     * @param {CanvasRenderingContext2D} ctx Context to use for measurements
     * @param {String} text The string of text to split into lines
     * @param {Number} lineIndex
     * @returns {Array} Array of line(s) into which the given text is wrapped
     * to.
     */
    _wrapLine: function(ctx, text, lineIndex) {
      var lineWidth        = 0,
          lines            = [],
          line             = '',
          words            = text.split(' '),
          word             = '',
```

```
      offset           = 0,
      infix            = ' ',
      wordWidth        = 0,
      infixWidth       = 0,
      largestWordWidth = 0,
      lineJustStarted = true,
      additionalSpace = this._getWidthOfCharSpacing();

  for (var i = 0; i < words.length; i++) {
    word = words[i];
    wordWidth = this._measureText(ctx, word, lineIndex, offset);

    offset += word.length;

    lineWidth += infixWidth + wordWidth - additionalSpace;

    if (lineWidth >= this.width && !lineJustStarted) {
      lines.push(line);
      line = '';
      lineWidth = wordWidth;
      lineJustStarted = true;
    }
    else {
      lineWidth += additionalSpace;
    }

    if (!lineJustStarted) {
      line += infix;
    }
    line += word;

    infixWidth = this._measureText(ctx, infix, lineIndex, offset);
    offset++;
    lineJustStarted = false;
    // keep track of largest word
    if (wordWidth > largestWordWidth) {
      largestWordWidth = wordWidth;
    }
  }

  i && lines.push(line);

  if (largestWordWidth > this.dynamicMinWidth) {
    this.dynamicMinWidth = largestWordWidth - additionalSpace;
  }

  return lines;
},
/**
 * Gets lines of text to render in the Textbox. This function calculates
 * text wrapping on the fly everytime it is called.
 * @returns {Array} Array of lines in the Textbox.
 * @override
 */
_splitTextIntoLines: function(ctx) {
  ctx = ctx || this.ctx;
  var originalAlign = this.textAlign;
  this._styleMap = null;
  ctx.save();
  this._setTextStyles(ctx);
  this.textAlign = 'left';
  var lines = this._wrapText(ctx, this.text);
  this.textAlign = originalAlign;
  ctx.restore();
```

```
      this._textLines = lines;
      this._styleMap = this._generateStyleMap();
      return lines;
    },

    /**
     * When part of a group, we don't want the Textbox's scale to increase if
     * the group's increases. That's why we reduce the scale of the Textbox by
     * the amount that the group's increases. This is to maintain the effective
     * scale of the Textbox at 1, so that font-size values make sense. Otherwise
     * the same font-size value would result in different actual size depending
     * on the value of the scale.
     * @param {String} key
     * @param {*} value
     */
    setOnGroup: function(key, value) {
      if (key === 'scaleX') {
        this.set('scaleX', Math.abs(1 / value));
        this.set('width', (this.get('width') * value) /
          (typeof this.__oldScaleX === 'undefined' ? 1 : this.__oldScaleX));
        this.__oldScaleX = value;
      }
    },

    /**
     * Returns 2d representation (lineIndex and charIndex) of cursor (or
selection start).
     * Overrides the superclass function to take into account text wrapping.
     *
     * @param {Number} [selectionStart] Optional index. When not given, current
selectionStart is used.
     */
    get2DCursorLocation: function(selectionStart) {
      if (typeof selectionStart === 'undefined') {
        selectionStart = this.selectionStart;
      }

      var numLines = this._textLines.length,
          removed  = 0;

      for (var i = 0; i < numLines; i++) {
        var line    = this._textLines[i],
            lineLen = line.length;

        if (selectionStart <= removed + lineLen) {
          return {
            lineIndex: i,
            charIndex: selectionStart - removed
          };
        }

        removed += lineLen;

        if (this.text[removed] === '\n' || this.text[removed] === ' ') {
          removed++;
        }
      }

      return {
        lineIndex: numLines - 1,
        charIndex: this._textLines[numLines - 1].length
      };
    },
```

```
    /**
     * Overrides superclass function and uses text wrapping data to get cursor
     * boundary offsets instead of the array of chars.
     * @param {Array} chars Unused
     * @param {String} typeOfBoundaries Can be 'cursor' or 'selection'
     * @returns {Object} Object with 'top', 'left', and 'lineLeft' properties
set.
     */
    _getCursorBoundariesOffsets: function(chars, typeOfBoundaries) {
      var topOffset      = 0,
          leftOffset     = 0,
          cursorLocation = this.get2DCursorLocation(),
          lineChars      = this._textLines[cursorLocation.lineIndex].split(''),
          lineLeftOffset = this._getLineLeftOffset(this._getLineWidth(this.ctx,
cursorLocation.lineIndex));

      for (var i = 0; i < cursorLocation.charIndex; i++) {
        leftOffset += this._getWidthOfChar(this.ctx, lineChars[i],
cursorLocation.lineIndex, i);
      }

      for (i = 0; i < cursorLocation.lineIndex; i++) {
        topOffset += this._getHeightOfLine(this.ctx, i);
      }

      if (typeOfBoundaries === 'cursor') {
        topOffset += (1 - this._fontSizeFraction) *
this._getHeightOfLine(this.ctx, cursorLocation.lineIndex)
          / this.lineHeight -
this.getCurrentCharFontSize(cursorLocation.lineIndex, cursorLocation.charIndex)
          * (1 - this._fontSizeFraction);
      }

      return {
        top: topOffset,
        left: leftOffset,
        lineLeft: lineLeftOffset
      };
    },

    getMinWidth: function() {
      return Math.max(this.minWidth, this.dynamicMinWidth);
    },

    /**
     * Returns object representation of an instance
     * @method toObject
     * @param {Array} [propertiesToInclude] Any properties that you might want
to additionally include in the output
     * @return {Object} object representation of an instance
     */
    toObject: function(propertiesToInclude) {
      return this.callSuper('toObject',
['minWidth'].concat(propertiesToInclude));
    }
  });

  /**
   * Returns fabric.Textbox instance from an object representation
   * @static
   * @memberOf fabric.Textbox
   * @param {Object} object Object to create an instance from
   * @param {Function} [callback] Callback to invoke when an fabric.Textbox
instance is created
```

```
   * @param {Boolean} [forceAsync] Force an async behaviour trying to create
pattern first
   * @return {fabric.Textbox} instance of fabric.Textbox
   */
  fabric.Textbox.fromObject = function(object, callback, forceAsync) {
    return fabric.Object._fromObject('Textbox', object, callback, forceAsync,
'text');
  };

  /**
   * Returns the default controls visibility required for Textboxes.
   * @returns {Object}
   */
  fabric.Textbox.getTextboxControlVisibility = function() {
    return {
      tl: false,
      tr: false,
      br: false,
      bl: false,
      ml: true,
      mt: false,
      mr: true,
      mb: false,
      mtr: true
    };
  };

})(typeof exports !== 'undefined' ? exports : this);


(function() {

  /**
   * Override _setObjectScale and add Textbox specific resizing behavior.
Resizing
   * a Textbox doesn't scale text, it only changes width and makes text wrap
automatically.
   */
  var setObjectScaleOverridden = fabric.Canvas.prototype._setObjectScale;

  fabric.Canvas.prototype._setObjectScale = function(localMouse, transform,
                                                     lockScalingX, lockScalingY,
by, lockScalingFlip, _dim) {

    var t = transform.target;
    if (t instanceof fabric.Textbox) {
      var w = t.width * ((localMouse.x / transform.scaleX) / (t.width +
t.strokeWidth));
      if (w >= t.getMinWidth()) {
        t.set('width', w);
        return true;
      }
    }
    else {
      return setObjectScaleOverridden.call(fabric.Canvas.prototype, localMouse,
transform,
        lockScalingX, lockScalingY, by, lockScalingFlip, _dim);
    }
  };

  /**
   * Sets controls of this group to the Textbox's special configuration if
   * one is present in the group. Deletes _controlsVisibility otherwise, so that
   * it gets initialized to default value at runtime.
```

```
   */
  fabric.Group.prototype._refreshControlsVisibility = function() {
    if (typeof fabric.Textbox === 'undefined') {
      return;
    }
    for (var i = this._objects.length; i--;) {
      if (this._objects[i] instanceof fabric.Textbox) {

this.setControlsVisibility(fabric.Textbox.getTextboxControlVisibility());
        return;
      }
    }
  };

  fabric.util.object.extend(fabric.Textbox.prototype, /** @lends
fabric.IText.prototype */ {
    /**
     * @private
     */
    _removeExtraneousStyles: function() {
      for (var prop in this._styleMap) {
        if (!this._textLines[prop]) {
          delete this.styles[this._styleMap[prop].line];
        }
      }
    },

    /**
     * Inserts style object for a given line/char index
     * @param {Number} lineIndex Index of a line
     * @param {Number} charIndex Index of a char
     * @param {Object} [style] Style object to insert, if given
     */
    insertCharStyleObject: function(lineIndex, charIndex, style) {
      // adjust lineIndex and charIndex
      var map = this._styleMap[lineIndex];
      lineIndex = map.line;
      charIndex = map.offset + charIndex;

      fabric.IText.prototype.insertCharStyleObject.apply(this, [lineIndex,
charIndex, style]);
    },

    /**
     * Inserts new style object
     * @param {Number} lineIndex Index of a line
     * @param {Number} charIndex Index of a char
     * @param {Boolean} isEndOfLine True if it's end of line
     */
    insertNewlineStyleObject: function(lineIndex, charIndex, isEndOfLine) {
      // adjust lineIndex and charIndex
      var map = this._styleMap[lineIndex];
      lineIndex = map.line;
      charIndex = map.offset + charIndex;

      fabric.IText.prototype.insertNewlineStyleObject.apply(this, [lineIndex,
charIndex, isEndOfLine]);
    },

    /**
     * Shifts line styles up or down. This function is slightly different than
the one in
     * itext_behaviour as it takes into account the styleMap.
     *
```

```
     * @param {Number} lineIndex Index of a line
     * @param {Number} offset Can be -1 or +1
     */
    shiftLineStyles: function(lineIndex, offset) {
      // shift all line styles by 1 upward
      var map = this._styleMap[lineIndex];
      // adjust line index
      lineIndex = map.line;
      fabric.IText.prototype.shiftLineStyles.call(this, lineIndex, offset);
    },

    /**
     * Figure out programatically the text on previous actual line (actual =
separated by \n);
     *
     * @param {Number} lIndex
     * @returns {String}
     * @private
     */
    _getTextOnPreviousLine: function(lIndex) {
      var textOnPreviousLine = this._textLines[lIndex - 1];

      while (this._styleMap[lIndex - 2] && this._styleMap[lIndex - 2].line ===
this._styleMap[lIndex - 1].line) {
        textOnPreviousLine = this._textLines[lIndex - 2] + textOnPreviousLine;

        lIndex--;
      }

      return textOnPreviousLine;
    },

    /**
     * Removes style object
     * @param {Boolean} isBeginningOfLine True if cursor is at the beginning of
line
     * @param {Number} [index] Optional index. When not given, current
selectionStart is used.
     */
    removeStyleObject: function(isBeginningOfLine, index) {

      var cursorLocation = this.get2DCursorLocation(index),
          map            = this._styleMap[cursorLocation.lineIndex],
          lineIndex      = map.line,
          charIndex      = map.offset + cursorLocation.charIndex;
      this._removeStyleObject(isBeginningOfLine, cursorLocation, lineIndex,
charIndex);
    }
  });
})();


(function() {
  var override = fabric.IText.prototype._getNewSelectionStartFromOffset;
  /**
   * Overrides the IText implementation and adjusts character index as there is
not always a linebreak
   *
   * @param {Number} mouseOffset
   * @param {Number} prevWidth
   * @param {Number} width
   * @param {Number} index
   * @param {Number} jlen
   * @returns {Number}
```

```
   */
  fabric.IText.prototype._getNewSelectionStartFromOffset = function(mouseOffset,
prevWidth, width, index, jlen) {
    index = override.call(this, mouseOffset, prevWidth, width, index, jlen);

    // the index passed into the function is padded by the amount of lines from
_textLines (to account for \n)
    // we need to remove this padding, and pad it by actual lines, and / or
spaces that are meant to be there
    var tmp     = 0,
        removed = 0;

    // account for removed characters
    for (var i = 0; i < this._textLines.length; i++) {
      tmp += this._textLines[i].length;

      if (tmp + removed >= index) {
        break;
      }

      if (this.text[tmp + removed] === '\n' || this.text[tmp + removed] === ' ')
{
        removed++;
      }
    }

    return index - i + removed;
  };
})();


(function() {

  if (typeof document !== 'undefined' && typeof window !== 'undefined') {
    return;
  }

  var DOMParser = require('xmldom').DOMParser,
      URL = require('url'),
      HTTP = require('http'),
      HTTPS = require('https'),

      Canvas = require('canvas'),
      Image = require('canvas').Image;

  /** @private */
  function request(url, encoding, callback) {
    var oURL = URL.parse(url);

    // detect if http or https is used
    if ( !oURL.port ) {
      oURL.port = ( oURL.protocol.indexOf('https:') === 0 ) ? 443 : 80;
    }

    // assign request handler based on protocol
    var reqHandler = (oURL.protocol.indexOf('https:') === 0 ) ? HTTPS : HTTP,
        req = reqHandler.request({
          hostname: oURL.hostname,
          port: oURL.port,
          path: oURL.path,
          method: 'GET'
        }, function(response) {
          var body = '';
          if (encoding) {
```

```
            response.setEncoding(encoding);
          }
          response.on('end', function () {
            callback(body);
          });
          response.on('data', function (chunk) {
            if (response.statusCode === 200) {
              body += chunk;
            }
          });
        });

    req.on('error', function(err) {
      if (err.errno === process.ECONNREFUSED) {
        fabric.log('ECONNREFUSED: connection refused to ' + oURL.hostname + ':'
+ oURL.port);
      }
      else {
        fabric.log(err.message);
      }
      callback(null);
    });

    req.end();
  }

  /** @private */
  function requestFs(path, callback) {
    var fs = require('fs');
    fs.readFile(path, function (err, data) {
      if (err) {
        fabric.log(err);
        throw err;
      }
      else {
        callback(data);
      }
    });
  }

  fabric.util.loadImage = function(url, callback, context) {
    function createImageAndCallBack(data) {
      if (data) {
        img.src = new Buffer(data, 'binary');
        // preserving original url, which seems to be lost in node-canvas
        img._src = url;
        callback && callback.call(context, img);
      }
      else {
        img = null;
        callback && callback.call(context, null, true);
      }
    }
    var img = new Image();
    if (url && (url instanceof Buffer || url.indexOf('data') === 0)) {
      img.src = img._src = url;
      callback && callback.call(context, img);
    }
    else if (url && url.indexOf('http') !== 0) {
      requestFs(url, createImageAndCallBack);
    }
    else if (url) {
      request(url, 'binary', createImageAndCallBack);
    }
```

```
    else {
      callback && callback.call(context, url);
    }
  };

  fabric.loadSVGFromURL = function(url, callback, reviver) {
    url = url.replace(/^\n\s*/, '').replace(/\?.*$/, '').trim();
    if (url.indexOf('http') !== 0) {
      requestFs(url, function(body) {
        fabric.loadSVGFromString(body.toString(), callback, reviver);
      });
    }
    else {
      request(url, '', function(body) {
        fabric.loadSVGFromString(body, callback, reviver);
      });
    }
  };

  fabric.loadSVGFromString = function(string, callback, reviver) {
    var doc = new DOMParser().parseFromString(string);
    fabric.parseSVGDocument(doc.documentElement, function(results, options) {
      callback && callback(results, options);
    }, reviver);
  };

  fabric.util.getScript = function(url, callback) {
    request(url, '', function(body) {
      // eslint-disable-next-line no-eval
      eval(body);
      callback && callback();
    });
  };

  // fabric.util.createCanvasElement = function(_, width, height) {
  //   return new Canvas(width, height);
  // }

  /**
   * Only available when running fabric on node.js
   * @param {Number} width Canvas width
   * @param {Number} height Canvas height
   * @param {Object} [options] Options to pass to FabricCanvas.
   * @param {Object} [nodeCanvasOptions] Options to pass to NodeCanvas.
   * @return {Object} wrapped canvas instance
   */
  fabric.createCanvasForNode = function(width, height, options,
nodeCanvasOptions) {
    nodeCanvasOptions = nodeCanvasOptions || options;

    var canvasEl = fabric.document.createElement('canvas'),
        nodeCanvas = new Canvas(width || 600, height || 600, nodeCanvasOptions),
        nodeCacheCanvas = new Canvas(width || 600, height || 600,
nodeCanvasOptions);

    // jsdom doesn't create style on canvas element, so here be temp. workaround
    canvasEl.style = { };

    canvasEl.width = nodeCanvas.width;
    canvasEl.height = nodeCanvas.height;
    options = options || { };
    options.nodeCanvas = nodeCanvas;
    options.nodeCacheCanvas = nodeCacheCanvas;
    var FabricCanvas = fabric.Canvas || fabric.StaticCanvas,
```

```
        fabricCanvas = new FabricCanvas(canvasEl, options);
    fabricCanvas.nodeCanvas = nodeCanvas;
    fabricCanvas.nodeCacheCanvas = nodeCacheCanvas;
    fabricCanvas.contextContainer = nodeCanvas.getContext('2d');
    fabricCanvas.contextCache = nodeCacheCanvas.getContext('2d');
    fabricCanvas.Font = Canvas.Font;
    return fabricCanvas;
  };

  var originaInitStatic = fabric.StaticCanvas.prototype._initStatic;
  fabric.StaticCanvas.prototype._initStatic = function(el, options) {
    el = el || fabric.document.createElement('canvas');
    this.nodeCanvas = new Canvas(el.width, el.height);
    this.nodeCacheCanvas = new Canvas(el.width, el.height);
    originaInitStatic.call(this, el, options);
    this.contextContainer = this.nodeCanvas.getContext('2d');
    this.contextCache = this.nodeCacheCanvas.getContext('2d');
    this.Font = Canvas.Font;
  };

  /** @ignore */
  fabric.StaticCanvas.prototype.createPNGStream = function() {
    return this.nodeCanvas.createPNGStream();
  };

  fabric.StaticCanvas.prototype.createJPEGStream = function(opts) {
    return this.nodeCanvas.createJPEGStream(opts);
  };

  fabric.StaticCanvas.prototype._initRetinaScaling = function() {
    if (!this._isRetinaScaling()) {
      return;
    }

    this.lowerCanvasEl.setAttribute('width', this.width *
fabric.devicePixelRatio);
    this.lowerCanvasEl.setAttribute('height', this.height *
fabric.devicePixelRatio);
    this.nodeCanvas.width = this.width * fabric.devicePixelRatio;
    this.nodeCanvas.height = this.height * fabric.devicePixelRatio;
    this.contextContainer.scale(fabric.devicePixelRatio,
fabric.devicePixelRatio);
    return this;
  };
  if (fabric.Canvas) {
    fabric.Canvas.prototype._initRetinaScaling =
fabric.StaticCanvas.prototype._initRetinaScaling;
  }

  var origSetBackstoreDimension =
fabric.StaticCanvas.prototype._setBackstoreDimension;
  fabric.StaticCanvas.prototype._setBackstoreDimension = function(prop, value) {
    origSetBackstoreDimension.call(this, prop, value);
    this.nodeCanvas[prop] = value;
    return this;
  };
  if (fabric.Canvas) {
    fabric.Canvas.prototype._setBackstoreDimension =
fabric.StaticCanvas.prototype._setBackstoreDimension;
  }

})();
```