Appendix A
Canvas Reference

The Canvas Element

See the official W3C specification for full details on the `canvas` element (`www.w3.org/TR/html5/the-canvas-element.html`).

Table A-1 and Table A-2 list the attributes and methods available on the canvas element.

Table A-1 Canvas element attributes

Attribute	Type	Default	Description
`width`	`number`	`300`	The width of the `canvas` coordinate space.
`height`	`number`	`150`	The height of the `canvas` coordinate space.

Table A-2 Canvas element methods

Method	Description
`canvas.getContext(` `contextId,` `[...]` `)`	Returns a `context` object that exposes an API for drawing content on the `canvas` element. The `contextId` argument determines the type of `context object` returned. Whether any extra parameters are allowed depends on the `context` type. Example: `var ctx = canvas.getContext("2d"); // create 2d context`
`canvas.toDataURL(` `[string type],` `[...]` `)`	Returns a `string` containing a `data:` URL created from the canvas image. The image format can be specified in the optional `type` parameter in the form of a mime type. The default value for `type` is `image/png`. Some formats allow extra optional parameters to specify, for example, image quality. Example: `var uri = canvas.toDataURL("image/jpeg", 0.7);`
`canvas.toBlob(` `function callback,` `[string type],` `[...]` `)`	Creates a `Blob` object with the `canvas` image and calls the `callback` function, passing the created `Blob`. See `canvas.toDataURL()` for a description of the type parameter and optional extra parameters. See the W3C File API specification for details on the `Blob` object: http://dev.w3.org/2006/webapi/FileAPI/

The 2D Context API

This section describes the drawing API available through the 2D canvas context. See the official W3C specification for full details (`http://dev.w3.org/html5/2dcontext/`).

State management

Table A-3 lists the methods for managing the drawing state stack.

Table A-3 State management methods

Method	Description
`ctx.save()`	Pushes the current drawing state onto the drawing state stack.
`ctx.restore()`	Pops the top drawing state from the drawing state stack and restores the drawing state.

The drawing state consists of the clipping region, the transformation matrix, and the values of the following properties:

- `strokeStyle`
- `fillStyle`
- `lineWidth`
- `lineCap`
- `lineJoin`
- `miterLimit`
- `shadowOffsetX`
- `shadowOffsetY`
- `shadowBlur`
- `shadowColor`
- `font`
- `textAlign`
- `textBaseline`
- `globalAlpha`
- `globalCompositeOperation`

Transformations

When you are drawing shapes, paths, and images, all coordinates are transformed by the current transformation matrix. Table A-4 lists the methods available for modifying this matrix.

Table A-4 Transformation methods

Method	Description
`ctx.scale( number x, number y )`	Adds a scaling transformation that scales the coordinate space by a factor of `x` in the horizontal direction and a factor of `y` in the vertical direction.
`ctx.rotate( number angle )`	Adds a rotation transformation that rotates the coordinate space `angle` radians around the origin.
`ctx.translate( number x, number y )`	Adds a translation transformation that translates the coordinate space `x` units in the horizontal direction and `y` units in the vertical direction.
`ctx.transform( number a, number b, number c, number d, number e, number f )`	Multiplies the current transformation matrix with the matrix described by `a b c` `d e f` `0 0 0`
`ctx.setTransform( number a, number b, number c, number d, number e, number f )`	Resets the transformation matrix to the identity matrix and multiplies it with the matrix described by `a b c` `d e f` `0 0 0`

Shapes and Paths

The 2D Context has a rich API for drawing paths that can be stroked, filled, or used as clipping regions. Table A-5 lists the methods related to drawing paths.

Table A-5 Path methods

Method	Description
`ctx.beginPath()`	Resets the path.

`ctx.moveTo(`
`  number x,`
`  number y`
`)`
	Creates a new subpath and adds the point (`x`, `y`).

`ctx.closePath()`	Closes the current subpath and creates a new subpath starting at the endpoint of the now closed subpath.

`ctx.lineTo(`
`  number x,`
`  number y`
`)`
	Adds the point (`x`, `y`) to the current subpath and connects it to the previous point with a straight line.
	If there is no active subpath, one is created with the starting point (`x`, `y`).

`ctx.arcTo(`
`  number x0,`
`  number y0,`
`  number x1,`
`  number y1,`
`  number radius`
`)`
	Adds an arc segment to the current subpath.
	The arc is the shortest arc given by the circumference of a circle that has a tangent to the line from the last point in the subpath to (`x0`, `y0`), a tangent to the line from (`x0`, `y0`) to (`x1`, `y1`), and the specified `radius`.
	If there is no active subpath, one is created with the starting point (`x0`, `y0`).

`ctx.quadraticCurveTo(`
`  number cpx,`
`  number cpy,`
`  number x,`
`  number y`
`)`
	Adds the point (`x`, `y`) to the current subpath and connects it to the previous point using a quadratic Bézier curve with the control point (`cpx`, `cpy`).
	If there is no active subpath, one is created with the starting point (`cpx`, `cpy`).

continued

Table A-5 continued

Method	Description

`ctx.bezierCurveTo(`
`  number cp0x,`
`  number cp0y,`
`  number cp1x,`
`  number cp1y,`
`  number x,`
`  number y`
`)`
	Adds the point (`x`, `y`) to the current subpath and connects it to the previous point using a cubic Bézier curve with the control points (`cp0x`, `cp0y`) and (`cp1x`, `cp1y`).
	
	If there is no active subpath, one is created with the starting point (`cpx0`, `cpy0`).

`ctx.rect(`
`  number x,`
`  number y,`
`  number width,`
`  number height`
`)`
	Creates a new subpath with the points (`x`, `y`), (`x + width`, `y`), (`x + width`, `y + height`), and (`x`, `y + height`) connected by straight lines to form a rectangle. The subpath is then closed.

`ctx.arc(`
`  number x,`
`  number y,`
`  number radius,`
`  number startAngle,`
`  number endAngle,`
`  boolean anticlockwise`
`)`
	Adds an arc segment along the circumference of the circle centered at (`x`, `y`) with the specified `radius`. The start and end points of the arc segment are described by the angles `startAngle` and `endAngle`, specified in radians. The optional `anticlockwise` parameter determines the direction used to get from the start to the end. The default value is `false`.
	
	If there is no active subpath, one is created with the starting point of the arc. If there is an active subpath, the starting point of the arc is added to the subpath and connected to the last point by a straight line.

`ctx.fill()`	Fills all the subpaths in the current path using the current `fillStyle`.
`ctx.stroke()`	Fills all the subpaths in the current path using the current `strokeStyle`, `lineWidth`, `lineCap`, `lineJoin`, and `miterLimit` styles.
`ctx.clip()`	Sets the clipping region to the intersection of the current clipping region and all the subpaths in the current drawing path.

`ctx.isPointInPath(`
`  number x,`
`  number y`
`)`
	Returns `true` if the `point` (`x`, `y`) is inside the current path; otherwise returns `false`.

In addition to the path drawing API, the 2D Context also provides a few methods that work strictly with rectangles. Table A-6 lists these methods.

Table A-6 Rectangle methods

Method	Description
`ctx.clearRect( number x, number y, number width, number height )`	Clears a rectangle with upper-left corner (`x`, `y`) and dimensions `width` x `height`. All pixels in the region are set to transparent black.
`ctx.strokeRect( number x, number y, number width, number height )`	Strokes a rectangle with upper-left corner (`x`, `y`) and dimensions `width` x `height` using the current `strokeStyle`, `lineWidth`, and `lineJoin` style.
`ctx.fillRect( number x, number y, number width, number height )`	Fills a rectangle with upper-left corner (`x`, `y`) and dimensions `width` x `height` using the current `fillStyle`.

Fills and Strokes

The style of stroked and filled paths is determined by the properties listed in Table A-7.

Table A-7 Style properties

Property	Type	Default	Description
`ctx.fillStyle`	`any`	`#000000`	The current style used to fill shapes. Valid values are a `string` containing a CSS color, a `CanvasGradient` object, and a `CanvasPattern` object.
`ctx.strokeStyle`	`any`	`#000000`	The current style used to stroke shapes. Valid values are a `string` containing a CSS color, a `CanvasGradient` object, and a `CanvasPattern` object.

continued

Table A-7 continued

Property	Type	Default	Description
`ctx.lineWidth`	`number`	`1.0`	The current line width used to stroke shapes, specified in coordinate space units.
`ctx.lineCap`	`string`	`butt`	The current line cap style. Valid values are `butt`, `round`, and `square`.
`ctx.lineJoin`	`string`	`miter`	The current line join style used where two lines meet. Valid values are `bevel`, `round`, and `miter`.
`ctx.miterLimit`	`number`	`10.0`	The current miter limit ratio.

Besides CSS colors, the `ctx.strokeStyle` and `ctx.fillStyle` properties can take a `CanvasGradient` or `CanvasPattern` object created with the methods listed in Table A-8.

Table A-8 Gradients and patterns

Function	Description

`ctx.createLinearGradient(`
`  number x0,`
`  number y0,`
`  number x1,`
`  number y1`
`)`
	Returns a `CanvasGradient` object that represents a linear gradient from the point (`x0`, `y0`) to the point (`x1`, `y1`).
	Use the `gradient.addColorStop()` method on the `CanvasGradient` object to add stops to the gradient. Example:
	`var gradient = ctx.createLinearGradient(0,0,1,1);`
	`gradient.addColorStop(0.0, "red");`
	`gradient.addColorStop(0.5, "green");`
	`gradient.addColorStop(1.0, "blue");`

`ctx.createRadialGradient(`
`  number x0,`
`  number y0,`
`  number r0,`
`  number x1,`
`  number y1,`
`  number r1`
`)`
	Returns a `CanvasGradient` object that represents a radial gradient described by a circle centered in the point (`x0`, `y0`) with a radius `r0` and a circle centered in the point (`x1`, `y1`) with a radius `r1`.
	See `ctx.createLinearGradient()` for details about the `CanvasGradient` object.

Function	Description
`ctx.createPattern(` `Object image,` `[string repetition]` `)`	Returns a `CanvasPattern` object created from `image` repeating in the directions specified by `repetition`. The `image` parameter can be `HTMLImageElement`, `HTMLCanvasElement`, or `HTMLVideoElement`. If `image` is a `HTMLVideoElement`, the frame at the current playback position is used. The `repetition` parameter is optional and can have these values: `repeat`, `repeat-x`, `repeat-y`, or `no-repeat`. The default value is `repeat`.

Shadows

You can add a shadow effect to any stroked or filled path by using the properties listed in Table A-9.

Table A-9 Shadow properties

Property	Type	Default	Description
`ctx.shadowOffsetX`	`number`	`0.0`	The offset distance of the shadow in the horizontal direction.
`ctx.shadowOffsetY`	`number`	`0.0`	The offset distance of the shadow in the vertical direction.
`ctx.shadowBlur`	`number`	`0.0`	The strength of the blur effect applied to the shadow. Must be a non-negative number.
`ctx.shadowColor`	`string`	`transparent black`	The current shadow color. Must be a valid CSS color.

Images

There is only a single method related to drawing images, but it can be invoked in two different ways as shown in Table A-10.

Table A-10 Image methods

Method	Description

`ctx.drawImage(`
`  Object image,`
`  number x,`
`  number y,`
`  [number width,`
`  number height]`
`)`
	Draws the specified image onto the canvas, positioning it with the upper-left corner at (`x`, `y`).
	If the optional `width` and `height` arguments are given, the image is stretched to the specified dimensions.
	The `image` parameter can be `HTMLImageElement`, `HTMLCanvasElement`, or `HTMLVideoElement`.
	If `image` is `HTMLVideoElement`, the frame at the current playback position is used.

`ctx.drawImage(`
`  Object image,`
`  number sx,`
`  number sy,`
`  number swidth,`
`  number sheight,`
`  number x,`
`  number y,`
`  [number width,`
`  number height]`
`)`
	Draws a subregion of the specified `image` onto the canvas.
	The subregion is a rectangle with the upper-left corner in (`sx`, `sy`) and dimensions `swidth` x `sheight`.

Text

Table A-11 and Table A-12 list the properties and methods related to text drawing.

Table A-11 Text properties

Property	Type	Default	Description
`ctx.font`	`string`	`10px sans-serif`	The current font. Must be a valid CSS font setting.
`ctx.textAlign`	`string`	`start`	The current text alignment. Valid values are `start`, `end`, `left`, and `right`.
`ctx.textBaseline`	`string`	`alphabetic`	The current text baseline setting. Valid values are `top`, `hanging`, `middle`, `alphabetic`, `ideographic`, and `bottom`.

Table A-12 **Text methods**

Method	Description

`ctx.fillText(`
`  string text,`
`  number x,`
`  number y,`
`  number maxWidth`
`)`
	Draws the specified `text` on the canvas using the current `font`, `textAlign`, and `textBaseline` values, filling the area using the current `fillStyle` value. The text is anchored at the point (`x`, `y`).
	The optional `maxWidth` parameter adds a width constraint specified in CSS pixels.

`ctx.strokeText(`
`  string text,`
`  number x,`
`  number y,`
`  number maxWidth`
`)`
	As `ctx.fillText()` but strokes the text using the current `ctx.strokeStyle`, `ctx.lineWidth`, `ctx.lineJoin`, and `ctx.miterLimit` values.

`ctx.measureText(`
`  string text`
`)`
	Calculates the width required to draw the specified `text` using the current `ctx.font` value. Returns a `TextMetrics` object with a `width` property holding the result, given in CSS pixels.

Compositing

Whenever new shapes, paths, and images are drawn on the canvas, they are composited with the existing content using the compositing properties listed in Table A-13.

Table A-13 **Compositing properties**

Property	Type	Default	Description
`ctx.globalAlpha`	`number`	`1.0`	The alpha value applied to shapes and images drawn on the `canvas` element.
`ctx.globalCompositeOperation`	`string`	`source-over`	The operation used to composite shapes and images with the existing content of the `canvas` element. All compositing operations are described by Porter-Duff (PD) operations (`http://keithp.com/~keithp/porterduff/p253-porter.pdf`) See Table A-14 for valid operator names.

Table A-14 lists the valid values for the `ctx.globalCompositeOperation` property. In the descriptive text, A refers to the new shape or image, and B refers to the existing content of the `canvas` element.

Table A-14 Composite operations

Value	Description
`source-atop`	Renders A on top of B but only where B is not transparent.
`source-in`	Renders only A and only where B is not transparent.
`source-out`	Renders only A and only where B is transparent.
`source-over`	Renders A on top of B where A is not transparent.
`destination-atop`	Renders B on top of A but only where B is not transparent.
`destination-in`	Renders only B and only where A is not transparent.
`destination-out`	Renders only B and only where A is transparent.
`destination-over`	Renders B on top of A where A is not transparent.
`lighter`	Renders the sum of A and B.
`copy`	Disregards B and renders only A.
`xor`	Renders A where B is transparent and B where A is transparent. Renders transparent where neither A nor B is transparent.

Pixel manipulation

Access to individual pixel values is possible using the image data methods listed in Table A-15. Note that the compositing rules are ignored when replacing pixel data.

Table A-15 Image data methods

Method	Description
`ctx.getImageData( number x, number y, number width, number height )`	Returns an `ImageData` object with pixel data from the region described by the rectangle with upper-left corner at (`x`, `y`) and dimensions `width` x `height`. See Table A-16 for further details on the `ImageData` object.
`ctx.createImageData( number width, number height )`	Creates a new `ImageData` object with the dimensions `width` x `height` in CSS pixels. All values in the new `ImageData` object are set to 0; that is, all pixels are transparent black.

Method	Description

`ctx.createImageData(`
`ImageData imagedata`
`)`
	Creates a new `ImageData` object with the same dimensions as the specified `imagedata` object. All values in the new `ImageData` object are set to 0; that is, all pixels are transparent black.

`ctx.putImageData(`
`ImageData imagedata,`
`number x,`
`number y,`
`[number dirtyX,`
`number dirtyY,`
`number dirtyWidth,`
`number dirtyHeight]`
`)`
	Copies pixel data from the specified `imagedata` object onto the canvas. The data is positioned with the upper-left corner at (`x`, `y`).
	If the optional `dirtyX`, `dirtyY`, `dirtyWidth`, and `dirtyHeight` parameters are given, only data from that rectangular region is copied.
	This operation is not subject to `ctx.globalCompositeOperation`, `ctx.globalAlpha`, or any shadow effect.

Table A-16 lists the properties of the `ImageData` object returned by the `ctx.getImageData()` and `ctx.createImageData()` methods.

Table A-16 ImageData properties

Property	Type	Description
`imagedata.width`	`number`	The width of the data, given in device pixels.
`imagedata.height`	`number`	The height of the data, given in device pixels.
`imagedata.data`	`CanvasPixelArray`	An array with `(width*height*4)` elements containing the RGBA values of the image data. Each value is an integer between 0 and 255.

Accessibility

A working group is currently seeking to improve the accessibility of the `canvas` element, for example, with regard to screen readers. Because this is an ongoing effort, the methods listed in Table A-17 may not be fully implemented in your target browser(s) yet.

Table A-17 Accessibility methods

Method	Description

`ctx.drawFocusRing(`
`  Element element,`
`  [boolean canDrawCustom]`
`)`
	Draws a native focus ring around the current drawing path if `element` is focused or if `element` is a descendant of an element that is focused. The specified `element` must be a child of the `canvas` element.
	
	If the optional `canDrawCustom` argument is `true`, the focus ring is drawn only if the user's system is configured to draw custom focus rings.
	
	Returns `true` if `element` is focused and `canDrawCustom` is `true` but the user's system is not set up to draw custom focus rings; otherwise returns `false`.

`ctx.setCaretSelectionRect(`
`  Element element,`
`  number x,`
`  number y,`
`  number width,`
`  number height`
`)`
	If the specified `element` is focused, provides the rectangular region described by the point (`x`, `y`) and the dimensions `width` x `height` to any accessibility API supported by the user agent. The specified `element` must be a child of the `canvas` element.
	
	The `x`, `y`, `width`, and `height` values are all transformed by the current transformation matrix.
	
	Returns `true` if `element` is focused and a child of the `canvas` element; otherwise returns `false`.

`ctx.caretBlinkRate()`	Returns the system's blink rate in milliseconds. Returns -1 if the system does not support a caret blink rate.