DRAWING CANVAS SCRIPT INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/drawing-canvas

The `IDrawingCanvasInstance` interface derives from IWorldInstance to add APIs specific to the Drawing Canvas plugin.

Examples

See the Drawing Canvas: Generate Image example for a demonstration of programmatically generating an image for Drawing Canvas.

Drawing co-ordinates

The co-ordinate system used in drawing APIs work the same as they do for actions. For more information refer to the Drawing Canvas object documentation.

Colors in drawing methods

In some of the drawing APIs, there are parameters that accept a color. These are represented using an array with either three components e.g. `[r, g, b]`, in which case the alpha is treated as opaque, or four components e.g. `[r, g, b, a]` to specify the alpha as well. Each component is a normalized float value in the range 0-1.

For example `[1, 0, 0]` represents opaque red, and `[0, 0, 1, 0.5]` represents blue at 50% opacity.

Drawing Canvas events

See instance event for standard instance event object properties.

"resolutionchange"

Fired at the same time as the *On resolution changed* trigger. For more details see the section on resizing and resolution in the Drawing Canvas plugin manual entry.

Drawing Canvas APIs

clearCanvas(color)

Clear the entire canvas to a given color.

clearRect(left, top, right, bottom, color)

Clear a rectangular area on the canvas to a given color. This overwrites any existing pixel data in the canvas.

fillRect(left, top, right, bottom, color)

Fill a rectangular area on the canvas with a given color. This draws over any existing pixel data in the canvas (relevant when the opacity is not 1).

outlineRect(left, top, right, bottom, color, thickness)

Draw four lines around a rectangular area on the canvas with a given color and line thickness.

fillLinearGradient(left, top, right, bottom, color1, color2, direction = "horizontal")

Fill a rectangular area on the canvas with a linear gradient from `color1` to `color2`. The `direction` must be one of `"horizontal"` or `"vertical"`.

fillEllipse(x, y, radiusX, radiusY, color, isSmooth = true)

outlineEllipse(x, y, radiusX, radiusY, color, thickness, isSmooth = true)

Fill or outline an elliptical area on the canvas with a given color. The position is the center of the ellipse and the radius parameters determine the shape of the ellipse (set both to the same value to draw a circle). When drawing an outline, the `thickness` parameter is the line thickness. By default the edges of the drawn area are smoothed; for a pixellated style set `isSmooth` to `false`.

line(x1, y1, x2, y2, color, thickness, lineCap = "butt")

lineDashed(x1, y1, x2, y2, color, thickness, dashLength, lineCap = "butt")

Draw either a solid or a dashed line between two points with a given color and line thickness. The dashed variant also takes a `dashLength` parameter to set how long the dashes are. `lineCap` must be one of `"butt"` (which ends the line exactly at the start and end positions) or `"square"` (which squares off the line endings so it extends a little past the start and end positions).

fillPoly(polyPoints, color, isConvex = false)

linePoly(polyPoints, color, thickness, lineCap = "butt")

lineDashedPoly(polyPoints, color, thickness, dashLength, lineCap = "butt")

Fill or outline a polygon area with a given color. The polygon is specified as an array of two-element arrays with co-ordinates for `polyPoints`, e.g. `[[x1, y1], [x2, y2], ...]`. For the line variants, the `thickness`, `dashLength` and `lineCap` parameters are the same as used

for the `line()` and `lineDashed()` methods.
With `fillPoly()` the polygon must provide at least three points, and may be convex or concave. However concave polygons are internally converted in to multiple convex polygons. This process can sometimes fail due to floating point precision issues in the geometric calculations, and result in a glitchy rendering. If you know the shape you are rendering is convex, pass `true` for the `isConvex` parameter, which will bypass the internal conversion; however this will not render correctly if the polygon is in fact concave.

Note that self-intersecting polygons are not supported with `fillPoly()` and will not draw correctly.

---

setDrawBlend(blendMode)

Set the blend mode used for draw operations on to the canvas. This is different to the blend used to draw the canvas itself to the layout. The blend mode is specified as a string and must be one of `"normal"`, `"additive"`, `"copy"`, `"destination-over"`, `"source-in"`, `"destination-in"`, `"source-out"`, `"destination-out"`, `"source-atop"` or `"destination-atop"`.

---

async pasteInstances(instancesArr, includeEffects = true)

Draw a list of instances that are currently overlapping the canvas at their current positions, given as an array of IWorldInstance. By default objects are drawn exactly as they appear, taking in to account any effects added to them; set `includeEffects` to `false` to draw without effects, as if all the object's effects were disabled. Note that the drawing actually happens at the end of the tick, and so this method is `async` so it can be awaited to ensure the paste has completed.

Note if an object is destroyed immediately after pasting without waiting for completion, it will not be drawn, as it will be destroyed before it gets to be drawn.

---

setFixedResolutionMode(fixedWidth, fixedHeight)

setAutoResolutionMode()

Switch between fixed and auto resolution modes. For more information refer to the Drawing Canvas object documentation.

---

surfaceDeviceWidth

surfaceDeviceHeight

getSurfaceDeviceSize()

Read-only values representing the size of the Drawing Canvas rendering surface in device pixels. The method returns both values at the same time.

pixelScale

A read-only value with the size of a single canvas pixel in object co-ordinates. See the section *Co-ordinate systems* in the Drawing Canvas object documentation for more information.

async getImagePixelData()

Takes a snapshot of the drawing canvas pixel state on the GPU, and reads it back to the CPU asynchronously. Resolves with an ImageData representing the pixel data. Note this uses unpremultiplied alpha, whereas the surface on the GPU is premultiplied, so technically this is lossy.

loadImagePixelData(imageData, premultiplyAlpha = false)

Load pixel data in an ImageData in to the Drawing Canvas rendering surface. The ImageData must have a size equal to `surfaceDeviceWidth` and `surfaceDeviceHeight`. If the optional `premultiplyAlpha` parameter is set to `true`, the pixel data will premultiply the alpha (multiplying the RGB components by the A component). This can be left disabled if the pixel data is already premultiplied, which is also faster since the premultiplication step can be skipped.

saveImage(format, quality, areaRect)

Save the current canvas image in a compressed format (e.g. PNG or JPEG). All parameters are optional - if none are specified, the entire canvas is saved in PNG format. The `format` parameter is a string of the MIME type of the image format to use, e.g. "image/png". Where a lossy format is used like "image/jpeg", the `quality` parameter is a number from 0-1 for the compression quality. A subset of the canvas area can be saved (e.g. for saving a cropped image) by specifying a DOMRect for the `areaRect` parameter using units of device pixels. The method returns a Promise that resolves with a Blob of the resulting image.