DRAWING CANVAS

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

The **Drawing canvas** object provides a drawing surface that you can draw your own content on to using some drawing actions. This includes basic geometric shapes like rectangles, lines, polygons and ellipses, as well as the ability to paste other objects in to the canvas.

> *Drawing Canvas is a 2D feature. It does not work on 3D layers with a 3D camera view.*

Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IDrawingCanvasInstance script interface.

Colors

The colors for drawing commands or pixels, such as the color to fill a rectangle, are typically specified with the *rgba* system expression. This specifies the red, green, blue and alpha components of the color in the range 0-100. The *rgba255* expression is identical but uses a 0-255 scale that artists may be more familiar with. The *rgbEx* and *rgbEx255* expressions can also be used, but do not provide a parameter for alpha, so will act as if the alpha was fully opaque.

Handling resizing and resolution

The Drawing Canvas object has two ways of handling the resolution of the canvas, each with their own advantages and disadvantages.

Automatic resolution mode

By default, the canvas uses *automatic* resolution mode, which uses the same display resolution as the game. However the display resolution usually changes when the window is resized. For example in letterbox scale mode, when the window is resized larger the game displays the same content but at a larger size (using more pixels). A similar thing happens when you resize the object itself. The *Drawing canvas* object automatically changes the resolution of its canvas when this happens. This is destructive, though: the next draw to the canvas will clear it, because internally the drawing surface is destroyed and recreated at the new resolution. The advantage of this approach is it ensures drawn content will appear with high quality, as resizing or scaling will increase the resolution, rather than stretching a lower resolution image.

The reason it waits until the next draw to recreate the drawing surface is so that if you draw to it once, the same content is simply displayed stretched at the new resolution. This is often an acceptable result after drawing in *On start of layout*. Normally a better idea is to draw one-off

content in the trigger *On resolution changed* - this also triggers on startup, and again any time the resolution changes, ensuring your content can increase in detail if the window is resized larger.

Alternatively if you clear and re-render the canvas in *Every tick*, since it is always redrawing it will always draw at the current resolution. This approach is similar to how the runtime itself renders, although you may want to stop drawing if nothing is changing in the game, which is also what the runtime does.

Fixed resolution mode

Another option is to use a fixed resolution. This defaults to using the size of the object as the size of the drawing surface (i.e. if the object is 100x100 in the layout, then the drawing surface will be 100x100 pixels too). The drawing surface can also be set to a different size using the *Set resolution mode action*.

Fixed resolution mode will keep the drawing surface at the same resolution regardless of the object size, window size, or scale. In other words it will always use the same size surface and just stretch it to fit the display. The advantage of this is it will preserve the drawn content even as the display resolution changes. However a potential downside is as it does not increase quality as the resolution increases, in some cases it could appear blurry or pixellated.

Co-ordinate systems

The *Drawing canvas* object allows you to draw shapes and lines at specific positions. In automatic resolution mode it uses a co-ordinate system relative to the object in layout co-ordinates. For example if the object is sized 100x100 in the layout, then the point (50, 50) is always in the middle of the canvas, regardless of the resolution. This means you don't need to change the drawing co-ordinates depending on the canvas resolution: everything scales automatically, so if the user resizes the window larger, content automatically increases in detail.

In fixed resolution mode the co-ordinate system is instead in pixels on the drawing surface. For example if the drawing surface is 100x100, then the point (50, 50) is in the middle, regardless of the size or scale of the Drawing Canvas object.

> *Note that when using snapshots to read and write pixel data, snapshot pixels are given in actual pixel co-ordinates rather than object co-ordinates, even in automatic resolution mode. The PixelScale expression also gives the size of an actual pixel on the drawing surface in object co-ordinates, allowing you to convert between object and pixel co-ordinates.*

Pixel manipulation with snapshots

The way computer graphics works makes requesting to retrieve the color of even just a single pixel an extremely costly operation. Rendering is highly optimised to go one way: sending drawing commands from the CPU, rendering on the GPU, and then displaying the result on the user's screen. Getting the color of a pixel back to the CPU goes in the opposite direction, and if

the system has to wait for the result, it would completely stall the rendering process, almost certainly janking the framerate.

In order to avoid the performance pitfalls while allowing access to individual pixels, the *Drawing canvas* object uses a snapshot system, which works like this:

1. The *Save snapshot* action copies the entire canvas image on the GPU.
2. The copied image is then sent asynchronously (in parallel) back to the CPU. Rendering can continue while this happens.
3. When the copied image is available on the CPU, *On snapshot* triggers. Now the pixel data can be read using the snapshot expressions, e.g. *SnapshotRedAt(x, y)*.
4. If the pixel data needs to be altered, it can be modified using the *Set snapshot pixel* action. This only changes the copy available to the CPU and does not visibly affect the canvas. To make changes visible, use the *Load snapshot* action, which copies the snapshot back to the canvas.

The *Noise textures* example demonstrates how to do this, writing random pixel data generated by the *Advanced Random* object.

Drawing canvas properties

Resolution mode

How to handle the resolution of the actual drawing surface. For more information see the section above on *Handling resizing and resolution*.

Initially visible

Whether the object is initially visible at runtime.

Origin

Choose the location of the object origin relative to its box.

Antialiasing

Enable multisample antialiasing (MSAA) for better quality drawing. For example normally polygon and line edges will be hard ("jagged"). Enabling antialiasing improves the quality of the edges (making them "softer"), but makes rendering slower and uses more memory. Higher levels of antialiasing reduce performance further. This option requires WebGL 2+, and not all devices support all levels of antialiasing. For example if you select 8x MSAA but a particular device only supports 4x MSAA, it will fall back to 4x MSAA instead.

> *If antialiasing is enabled, the Paste object action cannot draw with effects enabled if the object uses any background-blending effects, e.g. Screen. This is because internally*

WebGL does not support directly reading from antialiased surfaces. If you need to paste objects with background-blending effects, turn off antialiasing.

Drawing canvas conditions

On resolution changed

Triggered when the display resolution of the canvas changes, e.g. due to resizing the window in *Letterbox scale* mode. See the section on *Handling resizing and resolution* above. If you draw to the canvas in this trigger, it will mean the canvas image always automatically scales to the display resolution. If you do not draw to the canvas in this trigger, it will simply display the old image stretched to the new size.

On saved image

Triggered after the *Save image* action when the saved image is ready. The image can be accessed using the *SavedImageURL* expression, e.g. to download it using the Browser object's *Invoke download* action.

On snapshot

Triggered after the *Save snapshot* action when the snapshot is ready. This allows reading and writing individual pixels in the snapshot image, e.g. using the *SnapshotRedAt* expression. See the section *Pixel manipulation with snapshots* above for more information.

Drawing canvas - canvas actions

This group of actions generally relates to the canvas image as a whole.

Set resolution mode

Change between automatic or fixed resolution modes for the drawing surface, or change the size of a fixed resolution canvas. Note that if the canvas resolution changes, including by changing the size of a fixed resolution canvas, the surface will be destroyed and recreated, causing the previously drawn content to disappear.

Clear snapshot

Release a snapshot saved with the *Save snapshot* action, saving the memory used to store it. This returns the state to as if no snapshot was taken at all. Any expressions attempting to retrieve snapshot pixels after this action will return 0.

Load snapshot

Copy the snapshot in memory back to the canvas, so any changes to its pixel data become visible in the object. The snapshot must match the resolution of the canvas - if the resolution

has changed, the snapshot cannot be loaded. Note this still leaves the snapshot in memory - use the *Clear snapshot* action to remove it and save memory if the snapshot is no longer needed.

Save image

Save the current canvas image in a compressed format (PNG or JPEG) suitable for the user to download. A subset of the canvas area can be saved (e.g. for saving a cropped image) by specifying the *X*, *Y*, *Width* and *Height* parameters, all given in device pixels. The expressions *SurfaceDeviceWidth* and *SurfaceDeviceHeight* give the size of the canvas in device pixels. The default (leaving all values as zero) will save the entire canvas area. This action triggers *On saved image* when ready, and sets the *SavedImageURL* expression to a URL that can be downloaded.

Save snapshot

Copy the current canvas image to make its pixel data available. Triggers *On snapshot* when ready. For more information see the section *Pixel manipulation with snapshots* above.

Set snapshot pixel

Set the color of a pixel in the saved snapshot. A snapshot must have previously been taken and *On snapshot* triggered. Unlike the other drawing commands, this takes a position in pixel co-ordinates. Use the *rgba* expression to set the pixel color to the given red, green, blue and alpha components. Note changes will not be visible until the *Load snapshot* action is used.

Drawing canvas - general actions

This group of actions provides general drawing actions such as filling rectangles and drawing lines. Drawing polygons involves more actions so is separated out in to its own group.

Clear canvas

Clear the entire canvas to a color. This overwrites all existing image content. For example clearing to transparent will make the entire canvas transparent, unlike drawing a transparent rectangle which will not make any visible difference.

Clear rectangle

As with *Clear canvas*, but only clears a rectangular area on the canvas.

Line

Dashed line

Draw a line between two points on the canvas, specifying the color and thickness of the line. The line cap specifies whether the end of the line ends exactly at the start and end points (*Butt*) or is squared off, extending slightly beyond the start and end points (*Square*). The

Dashed line variant also allows specifying a *Dash length*, which alternates between the line color and transparency for a dash effect.

Fill ellipse

Outline ellipse

Draw either a filled or outlined ellipse shape on the canvas. The shape is specified using the center point and the radius on the X and Y axes. If the radius on both axes is the same, the shape will be a circle. The outline variant also specifies the thickness of the outline. The edge can also be set to *Hard* for an aliased edge (suitable for pixellated games), or *Smooth* for a better quality soft-edged appearance.

Fill linear gradient

Fill a rectangular area on the canvas with a linear gradient from one color to another. The gradient direction can be either horizontal or vertical.

> *The gradient uses gamma-correct calculations, which may appear differently to other tools that use gamma-incorrect calculations (by linearly interpolating in sRGB space).*

Fill rectangle

Outline rectangle

Draw either a filled or outlined rectangle shape on the canvas. The outline variant also specifies the thickness of the outline.

Paste object

Draw all instances of the given object that are overlapping the canvas onto the canvas at their current positions. By default objects are drawn exactly as they appear, taking in to account any effects added to them; drawing without effects will draw as if all the object's effects were disabled.

> *The drawing actually happens at the end of the tick. The action is asynchronous, so it can be used with the System Wait for previous actions to complete action, which can be used 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.*

Set drawing blend

Set the background blending mode used for all drawing operations except *Paste object* (which takes the blend mode from the object being pasted). The options match the same blend modes as can be used by objects, but applies to the drawing to the canvas, rather than

the display of the Drawing Canvas object itself. Two useful options are setting the "copy" blend mode and drawing with transparency to clear an area, or using "destination out" to erase content.

Drawing canvas - polygon actions

This group of actions allows for drawing polygon shapes. Use *Add poly point* multiple times to add points. Then outlined or filled polygons can be drawn. Use *Reset poly* to clear added points and start again.

Add poly point

Add a new point to the current polygon. At least three points must be added before a polygon can be drawn.

Outline poly

Dashed outline poly

Draw dashed lines between the points of the current polygon, joining back to the start point. The parameters are similar to the *Line* and *Dashed line* actions.

Fill poly

Fill the current polygon area with a color. This supports both convex and concave polygons. 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, check the *Convex* 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 and will not draw correctly.*

Reset poly

Clear all polygon points that were added, so the next *Add poly point* action starts a new polygon.

Drawing canvas expressions

PixelScale

The size of a single canvas pixel in object co-ordinates. See the section *Co-ordinate systems* above for more information.

SavedImageURL

In *On saved image*, the URL of the saved image. This can be downloaded using the Browser object's *Invoke download* action.

SnapshotRedAt(x, y)

SnapshotGreenAt(x, y)

SnapshotBlueAt(x, y)

SnapshotAlphaAt(x, y)

Return the red, green, blue and alpha components of a pixel in a saved snapshot. The position is given in pixel co-ordinates, and the returned value is returned in the 0-100 range. For more information see the section *Pixel manipulation with snapshots* above.

SnapshotWidth

SnapshotHeight

Return the size of a saved snapshot in pixels. For more information see the section *Pixel manipulation with snapshots* above.

SurfaceDeviceWidth

SurfaceDeviceHeight

Returns the current size of the canvas surface in device pixels. These are useful to use when defining a subset area to use in the *Save image* action.