CONFIGURING EFFECTS

View online: https://www.construct.net/en/make-games/manuals/addon-sdk/guide/configuring-effects

The main configuration for an effect is set by additional effect-specific properties in addon.json. The additional properties used by effects are listed below.

category

The category the effect should appear in. This must be one of `"3d"`, `"blend"`, `"color"`, `"distortion"`, `"mask"`, `"normal-mapping"`, `"tiling"`, `"other"`.

supported-renderers

An array of strings indicating the supported renderers for this effect. By default (if omitted) it is `["webgl"]`. The string `"webgl2"` can be added to support a WebGL 2 variant of the effect - see the section on WebGL shaders for more details. The string `"webgpu"` can be added to support the WebGPU renderer with a shader written in WGSL - see the section on WebGPU shaders for more details.

blends-background

Boolean indicating whether the effect blends with the background. Objects and layers can use effects that blend with the background, but layouts cannot.

uses-depth

Boolean indicating whether the effect samples the depth buffer with the `samplerDepth` uniform. This is used for depth-based effects like fog.

cross-sampling

Boolean indicating whether a background-blending effect has inconsistent sampling of the background and foreground. A normal blending shader like Multiply will sample the background and foreground 1:1, so each foreground pixel samples only the background pixel it is rendered to. This is consistent sampling so `cross-sampling` should be `false`. However an effect that distorts the background, like Glass or a masking Warp effect, can sample different background pixels to the foreground pixel being rendered, so should set `cross-sampling` to `true`. This must be specified so the effect compositor can ensure the correct result is rendered when this happens.

preserves-opaqueness

Boolean indicating whether the effect preserves opaque pixels, i.e. every input pixel with an alpha of 1 is also output with an alpha of 1. This is true for most color-altering effects, but not for most distorting effects, since in some cases a previously opaque pixel will be distorted in to a transparent area of the texture. This information is not currently used, but is important for front-to-back rendering algorithms.

animated

Boolean indicating whether the effect is animated, i.e. changes over time using the `seconds` uniform. This is used to ensure Construct keeps redrawing the screen if an animated effect is visible.

must-predraw

Boolean indicating whether to force the pre-draw step. Sometimes Construct tries to optimise effect rendering by directly rendering an object with the shader applied. Setting this flag forces Construct to first render the object to an intermediate surface, which is necessary for some kinds of effect.

supports-3d-direct-rendering

Boolean indicating whether 3D objects can render directly with this effect. This defaults to `false`, causing all 3D objects with the effect to first perform a pre-draw step, and then processing the effect on a 2D surface. If set to `true`, then 3D objects with the effect are allowed to render directly to the display with the effect being processed for each triangle in the geometry. This is usually more efficient and can be more appropriate for processing 3D effects. Note however that the effect compositor may still decide to add a pre-draw step in some circumstances, so this setting is not a guarantee that it will always use direct rendering.

extend-box

Amount to extend the rendered box horizontally and vertically. Normally the effect is clipped to the object's bounding box, but some effects like Warp need to be able to render a short distance outside of that for the correct result. This property lets you extend the rendered box by a number of pixels. This property uses `"horizontal"` and `"vertical"` sub-properties, e.g. `"extend-box": { "horizontal": 30, "vertical": 30 }`

is-deprecated

Boolean to indicate a deprecated effect. This hides the effect from the *Add effect* dialog, but allows existing projects to continue using it. This allows an effect to be phased out without breaking projects.

parameters

An array of parameters that the effect uses. See the next section for more information.

Effect parameters

The `parameters` array in addon.json specifies a list of parameters that are passed to the shader as uniforms. These can be used to customise the appearance of the effect, and can also be changed at runtime. Each parameter is specified as an object with the following properties.

---

id

A string identifying this parameter.

---

c2id

Optional The corresponding ID used in a compatible legacy Construct 2 effect if this is not the same as the `id`. Note for color parameters, this can be a comma-separated list of the three parameter IDs previously used for the red, green and blue components, e.g. `"red,green,blue"`.

---

type

The type of the effect parameter. This can be one of `"float"`, `"percent"` or `"color"`. Floats pass a simple number. Percent displays a percentage in the 0-100 range but passes a float in the 0-1 range to the shader. Color shows a color picker and passes a vec3 with components in the 0-1 range to the shader.

---

initial-value

The initial value of the shader uniform, in the format the shader uses it (i.e. 0-1 range for percent parameters). For color parameters, use a 3-element array, e.g. [1, 0, 0] for red.

---

uniform

WebGL only The name of the corresponding uniform in the shader. The uniform must be declared in GLSL with this name. It can use whichever precision you want, but the uniform type must be `vec3` for color parameters, otherwise `float`.

This only applies to WebGL shaders written in GLSL. The WebGPU renderer ignores this setting.

---

interpolatable

Set to `true` so the property can be supported by timelines.

Writing shaders

Construct supports rendering with both WebGL and the newer WebGPU. However these technologies use different shader languages: WebGL uses GLSL, and WebGPU uses WGSL. To support both renderers your effect will need to provide both a GLSL and WGSL shader which both render equivalently.

For more details on writing shaders, see WebGL shaders for GLSL-specific information, and see WebGPU shaders for WGSL-specific information.

You should test your shader works with both renderers. Change the *Enable WebGPU* setting in the *Advanced* section of Project Properties to test both renderers. You can also change the *Enable WebGPU in editor* setting in Construct's Settings dialog to test both renderers with the editor's rendering in the Layout View.