PROJECTS

View online: https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/projects

A **project** is a complete game, app or animation made in Construct. Projects contain every element of your work, ranging from sprites to sound files. An overview of the project is shown in the Project Bar where elements can be added, renamed, removed and arranged in to folders for organisation. See Project structure for a summary of the elements making up a project. The rest of this manual section goes in to more detail about each part of a project.

Projects can be opened, closed and exported from the main menu. See also Saving and sharing projects, testing and publishing. It is recommended to follow some best practices while working on projects.

Project properties

The properties for a project can be edited in the Properties Bar after selecting the name of the project in the Project Bar, or using the *Project properties* shortcut in Layout Properties.

The *Name*, *Author* and *Description* properties are used for some of the export options, so be sure to fill them out accurately for any important projects.

About

Name

The name or title of the project.

Version

The version of the project, which conventionally is four numbers in descending importance (e.g. 1.0.0.0), where the first number is the major version and the last number is the revision number. This is also used by several exporters to assign the version to your published app. **Note:** different platforms have their own way of handling the version. To ensure the version works consistently across platforms, try to follow these rules with the project version:

- Use 3 or 4 version components (using too few could become limiting)
- Don't exceed the range 0-99 for any particular component. E.g. instead of incrementing 1.0.0.99 to 1.0.0.100, increment the next component, using 1.0.1.0.
- Enable *Auto-increment version* (see below), as some platforms do not allow you to publish an update unless the version is higher.

Auto-increment version

When enabled, every time you export the project, Construct will automatically increment the fourth number of the *Version* property. For example if the version is 1.4.0.0, when you export it will increment to 1.4.0.1, 1.4.0.2 etc. Note that this only supports numerical version components separated by a dot - it will not work with other version formats like "1.4a", "rev2" etc.

Description

A sentence or two giving a short summary of the project. Several exporters use this as the description for your published app.

ID

An ID uniquely identifying your application. This should be in reverse domain format, such as **com.mycompany.myproject**. Some exporters use this as the ID for your exported app, so try to ensure it will be unique.

Author

The name of the individual or organisation developing the project.

Email

A support or contact email address for the project. Some exporters use this to fill out the *Email* field of the published app.

Website

A link to the author's website or other related web address. Your site should be hosted securely (with https://). Some exporters use this to fill out the *Website* field of the published app.

Colors

Background color

If the viewport does not cover the whole screen, e.g. when using letterbox mode, this is the color of the bars that appear at the sides.

Splash color

When run as a web app, this is the background color of the splash screen which appears when the web app is first launched.

Use theme color

Check to enable the *Theme color* property, allowing to override the default browser color scheme.

Theme color

On some platforms, the theme color is used to tint the browser or OS color scheme, such as the address bar, app caption, or status bar. If *Use theme color* is disabled the system defaults will be used, otherwise the theme color will be applied instead.

Startup

First layout

Select which layout is the first to appear when the project is exported. When previewing in the editor usually a specific layout is previewed, selecting *Preview project* will also preview from this layout.

Use loader layout

Use *First layout* as a special layout which shows while the rest of the layout is loading. The *loadingprogress* system expression returns the current progress from 0 to 1 (e.g. 0.5 for half completed). For more information, see the tutorial how to make a custom loading screen.

Loader style Paid plans only

Change the default loader which is shown while the project is loading, or while the loader layout is itself still loading. See the tutorial how to make a custom loading screen for more information. The Free edition can only use the *Construct 3 splash* style. When using the *Progress bar & logo* style, the icon with the *Loading logo* purpose is used as the logo. See Icons & splash for more information.

Preload sounds

Whether to download and decode sounds before the project starts. If enabled, then sounds will be downloaded while the loading bar is showing. If disabled then sounds will be downloaded on-demand as the project runs, which can add a delay on the first time they are played, but it also means there is less to download before the project can start. Note this option does not preload music, which will still be streamed as the project runs.

Display

Viewport size

The size, in pixels, of the view area in a layout. A dashed line indicating the window size appears in the Layout View. The viewport aspect ratio is also displayed underneath to help you easily identify which aspect ratio your project is using.

Viewport fit

How to fit the viewport to the display on devices with non-rectangular screens (such as the iPhone X). The viewport is rectangular, and the default *Auto* will add borders around the screen to ensure the full viewport is visible. Using *Cover* will display the viewport covering the entire physical screen, but this can result in parts of the viewport being hidden on non-rectangular screens, such as if there are notches or rounded corners.

Fullscreen mode

This determines how to fill the available window or screen space with the viewport. By default it uses *Letterbox scale*, which stretches the viewport to fill all available space, using black bars down the sides to preserve the aspect ratio. There are several variations; for more information see the tutorial on supporting multiple screen sizes.

Fullscreen quality

This only applies when the viewport is being stretched (i.e. *Fullscreen mode* is not *Off*). *High quality* mode renders at the full resolution of the displayed size. *Low quality* mode first renders at the project viewport size, and then simply stretches the result to fill the screen. Low quality mode often improves performance on low-end systems and is often suitable for retro-style pixellated projects with *Point* sampling. However note that text, downscaled sprites and effects will appear with better detail in high quality mode.

Orientations

Whether to lock the orientation on mobile devices. *Any* allows the display to switch between portrait and landscape automatically; choosing either *portrait* or *landscape* will attempt to lock the orientation to prevent it changing. This is applied when publishing an app, but for web exports note that not all browsers or platforms support orientation locking or have limitations on when it can apply. In some browsers it must be in fullscreen mode (using the Browser object's *Request fullscreen* action) before orientation lock takes effect.

Sampling

Choose between *nearest* (pixellated), *bilinear* (smooth) and *trilinear* (smooth with better quality downscaling) sampling when resizing images. *Trilinear* is recommended for modern projects with hi-res graphics, and *nearest* is better suited to retro-style projects with blocky pixel art. *Bilinear* can be faster than *Trilinear* on low-end devices if the improved downscaling quality is not necessary.

Pixel rounding

By default objects can be drawn at sub-pixel positions, e.g. (100.3, 200.8). If *Sampling* is set to *Linear*, this can make fine pixel art appear blurry. If *Pixel rounding* is enabled, objects round their position to a whole number before drawing, e.g. (100, 201). This prevents any blurring, and can also prevent "seams" appearing on grids of objects. Note this does not affect their actual X and Y co-ordinates, which can still be between pixels - it only affects where they are drawn on the screen.

Z axis scale

Choose how the Z axis is measured, which affects 3D content like Z elevation and the 3D Shape object. The options are:

- **Normalized**: the default camera position is 100 units above the layout. However this means the Z axis has a different scale to the X and Y axes. This mode is suitable for 2D content which uses simple 3D features like Z elevation.
- **Regular** (default): the X, Y and Z axes all use the same scale. However this means the default camera position on the Z axis varies depending on the other project properties. This mode is more suitable for fully 3D content using the 3D Camera object.

The properties of the 3D Camera object reveal the Z axis scale and default camera Z position, which can be useful to refer to when altering this property.

Field of view

This property only appears when the *Z axis scale* is set to *Regular*. It adjusts the viewing angle of the 3D camera. Note this only affects perspective projections, as orthographic projections do not use a viewing angle. Also note adjusting the field of view will also change the default camera Z, as Construct adjusts it to ensure 2D content appears at 100% scale.

Advanced

Use worker

When enabled, the runtime is hosted in a Web Worker, off the main thread (where supported). This makes it less likely the browser will interrupt the project (also known as jank), generally improving performance. When disabled the runtime is hosted in the main thread with full access to the DOM (Document Object Model), but in some cases can be interrupted by the browser. *Auto* mode means Construct decides the mode automatically; currently this enables it unless you use the scripting feature, in which case it disables it on the assumption you will want to use DOM APIs. If your scripting code can run in a worker, you can still enable worker mode by changing the setting to *Yes*.

You can check if the runtime is actually hosted in a Web Worker by checking the browser console in preview mode. On startup it logs some technical details, which will include either "Hosted in DOM" or "Hosted in worker", the latter indicating worker mode is in use.

Enable WebGPU

Whether to enable the WebGPU renderer for this project. If disabled, WebGPU is not supported, or the project uses third-party effects that do not support WebGPU, then the

WebGL renderer will be used instead. In most cases the WebGPU renderer should have better performance than the WebGL renderer. The renderer in use can be identified by the Platform Info *Renderer* expression. The *Auto* setting means Construct will use the default, which is currently the WebGL renderer. Note the renderer used for the Construct editor (for Layout Views) is separately controlled in the Settings dialog.

Enable multitexturing

This option only appears when WebGPU may be used (the *Enable WebGPU* property is not *No*). Normally when the renderer changes texture it has to issue new rendering commands which has a CPU overhead. Construct's WebGPU renderer supports *multitexturing*, which can move most texture swapping to the GPU; however that usually adds some per-pixel performance overhead that can affect fill rate. In other words, multi-texturing off will use more CPU time and less GPU time, whereas multi-texturing on will use less CPU time and more GPU time. The optimal setting depends on your project and whether it is bottlenecked more on CPU or GPU performance. The default mode *Auto* enables multitexturing on desktop devices as they tend to have more powerful GPUs, but disables it on mobile devices as they tend to have less powerful GPUs where fill rate is more important.

Framerate mode

Adjust how the framerate is managed at runtime, providing a way to run at an uncapped framerate for performance testing. The default is to tick and draw a new frame every time the display hardware refreshes, which is the most efficient option and the only reasonable one to use when publishing a project. Two other options are provided mainly for performance testing purposes which allow the framerate to run as fast as possible. This makes it easier to test the performance impact of changes to your project.

Note that Construct provides both frames per second (FPS) and ticks per second (TPS) measurements, with FPS corresponding to rendered frames, and TPS corresponding to the engine processing logic. These measurements can differ depending on the framerate mode.

The options are:

- **V-synced** will run ticks to match the display refresh rate, and render a frame every tick as well. Note that if nothing changes then no frame may be rendered, in which case the frames per second measurement may be lower than the ticks per second measurement.

- **Unlimited (ticks only)** will run ticks as fast as possible, but still only draw a new frame every time the display refreshes. This means the engine will measure a very high ticks per second (TPS) rate, but it is still only visually producing frames at the normal rate (typically 60 FPS). This option is suitable for CPU performance testing.

- **Unlimited (full frames)** will run full frames as fast as possible, including issuing all the draw calls to draw a new frame. Therefore the frames per second and ticks per second will run at the same rate, although as with V-synced mode if nothing is changing then it may skip rendering frames. When running faster than the display refresh rate, many frames will not be seen, since they will be replaced by the next frame before the display hardware refreshes. However it ensures that draw calls are included in any performance measurement. This option is more suitable for testing rendering performance.

Do not publish a project using an unlimited framerate mode. It will drive the system hardware to the maximum, including draining the battery faster, spinning fans faster and louder, and raising the system temperature (possibly imposing thermal throttling). Many users notice these effects and it can result in negative reviews. These options are provided for performance testing during development; only V-synced mode should be used when publishing.

GPU preference

On devices with multiple GPUs, the type of GPU to prefer. The most common multi-GPU case is laptops that contain a weak low-power integrated GPU (designed to maximize battery life) and a powerful discrete GPU (designed to maximize performance). This setting controls the preferred GPU on such devices.

There is no guarantee this option will be used: it depends on the underlying platform having support for selecting a specific GPU, and even if it does, it may ignore the request in some circumstances (such as forcing the use of a low-power GPU if the system is running on battery power). In other words this option is considered as a hint rather than a requirement.

Downscaling quality

Adjusts the tradeoff between rendering quality and memory use when resizing images to smaller than their original size (downscaling). The options are:

- *Low quality*: mipmaps are disabled (reducing memory use), but downscaled sprites may appear blocky or pixellated. This mode is not recommended for most projects, since disabling mipmaps can reduce performance.

- *Medium quality*: mipmaps are enabled. Downscaling sprites generally looks better.

- *High quality*: mipmaps are enabled and the spritesheet after export pads out all images to power-of-two sizes. This can significantly increase memory use, but can resolve two minor rendering issues: light fringing that can sometimes occur along the borders of downscaled objects, or a quality change in the last frame of an animation. **Do not use**

this mode unless a rendering artefact is specifically observed and selecting this mode can be observed to resolve it: the increased memory usage can be very significant, and is not a cost that should be added for no reason. For more information see Memory usage.

Rendering mode

Whether to render the project in 2D or 3D mode. Normally Construct determines this automatically with the *Auto* setting. However if you only use 3D features dynamically, such as by altering 3D meshes at runtime, you may wish to opt in to 3D mode here. The options are as follows:

- **2D:** the project will render in 2D, without a depth buffer. Any 3D features, such as 3D shape objects, will render incorrectly. This mode may be slightly faster than 3D mode for 2D content, but normally you don't need to choose it, as *Auto* mode will use it for 2D projects anyway.
- **Auto:** uses 3D mode if your project uses any 3D features, otherwise uses 2D mode.
- **3D:** the project will render in 3D, with a depth buffer, which is necessary for correct rendering of 3D features.

Anisotropic filtering

The anisotripic filtering mode to use for all images in the project. This improves the appearance of surfaces at an oblique angle to the camera, such as the sides of 3D shape objects. It also improves the quality of 2D objects that are resized to extreme aspect ratios. Normally this can just be left at *Auto*. However in some cases this can affect performance, so is customizable. The options are as follows:

- **Off:** do not use anisotropic filtering. This can degrade the rendering quality of 3D and some 2D features, but may slightly improve performance.
- **Auto:** currently corresponds to 4x anisotropic filtering.
- **2x-16x:** enable a specific level of anisotropic filtering. Higher levels improve quality further but may have a slightly higher performance impact.

Near distance

Far distance

Set the distance of the near plane and far plane from the camera. Content closer to the camera than the near plane, or further from the camera than the far plane, will not be visible. This allows customizing the visible area when using a 3D Camera. It also controls the limits of how far the view can zoom in or zoom out from a 2D game, as in Construct that is implemented by moving a camera closer and further from the game. These limitations in zoom level will also be reflected in the Layout View's maximum and minimum zoom levels.

Max spritesheet size

The maximum spritesheet size in pixels Construct will use when grouping multiple images on to the same sheet. This adjusts the tradeoff between memory usage and performance: smaller sizes tend to reduce memory usage but can have reduced performance, whereas larger sizes tend to increase memory usage but improve performance. The special option *Disabled* will disable use of spritesheets completely, causing every single image used in the project to be exported as a separate image file. This can have a significant negative impact on the download size, loading time and runtime performance of the project, and in some cases large projects may crash due to running in to system limits on the number of images that can be loaded, so using some degree of spritesheeting is strongly recommended.

UID numbering

Sets how to allocate UIDs for newly created instances in the editor. The default mode *Increment* will use the lowest available UID, which tends to assign incrementing numbers like 1, 2, 3, 4 etc. However this can cause problems when collaborating on projects with source control, as it's possible two people could separately create new instances which get assigned the same UID. The *Random* mode is designed to avoid such problems: all newly created instances are assigned a random number with at least six digits, e.g. 129740, 652945, etc. This means there is a negligible chance that two people create new instances with the same UID.

Editor

Preview effects

Whether or not to display effects and blend modes in the Layout View. If enabled, WebGL must also be enabled for the effects to appear. If disabled, WebGL effects are not rendered in the editor, and all objects are drawn as if they have the *Normal* blend mode.

Pause on unfocus

If enabled, the preview will pause when the browser window loses focus, e.g. when switching back to work in Construct. This can be useful for certain workflows, or to prevent the project distracting you as you work. If disabled the preview will continue to run even without focus, but note switching to another browser tab or minimising the preview window will still pause (as it does with published projects).

Bundle addons Paid plans only

If enabled, all third-party addons that the project uses will be bundled with the project file when saved. This allows the project to be opened anywhere, such as on another system where the addons have not been pre-installed. This makes it more convenient to move projects using third-party addons between different devices. Note that addons can opt out of bundling; you will be notified when enabling this option if any addons cannot be bundled with the project. Bundled addons always use the version of the addon that was installed when they are saved. They can however be updated if the installed addon is a newer version via the **View used addons** dialog.