SYSTEM ACTIONS

View online: https://www.construct.net/en/make-games/manuals/construct-3/system-reference/system-actions

This section describes all the actions in the built-in System object in Construct. They are listed in the order they appear in the Add Action dialog.

Display

Set canvas size

Set the size of the canvas area in the page, if appearing inline to the page (i.e. a fullscreen mode is not used). If a fullscreen mode is in use, this effectively changes the size of the *Viewport size* project property, which adjusts the size of the viewport.

Set fullscreen scaling

Set the *Fullscreen quality* project property. This allows the quality setting to be adjusted at runtime.

Set pixel rounding

Set the *Pixel rounding* project property at runtime.

Snapshot canvas

Take a screenshot of the current display. 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 Platform Info expressions *CanvasDeviceWidth* and *CanvasDeviceHeight* 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 canvas snapshot* when the snapshot is ready, and the resulting image can be accessed with the *CanvasSnapshot* system expression. This can then be loaded in to a sprite or tiled background, sent to a server, or opened with the Browser object in a new tab.

General

Create object

Create a new instance of an object type on a layer at a given position. If a family is chosen, a random object type from the family is picked, and an instance created for that. Tick *Create hierarchy* when creating the root object in a hierarchy to automatically create the rest of the scene graph hierarchy with connections in place. Choose a valid *Template name* so the new instance is created based on the template rather than an arbitrary instance.

See Setting up a hierarchy in the Layout View manual entry for more information about hierarchies.

When Create hierarchy is ticked, the additional objects created are also picked. This means subsequent actions for those objects will only affect the newly created ones.

When using this action with a family a Template name can be used, but will only take effect if the object type
that is going to be created has that template name, otherwise it will be ignored.

See the Templates manual entry for more information on what templates are and how to start using them.

Create object (by name)

As with *Create object*, but allows using a string of the name of the object type to create. This allows using an expression to dynamically pick which kind of object to create. The *Pick last created* condition is sometimes useful to pick the resulting instance in a sub-event.

Reset persisted objects

Across the entire project, reset all objects using the Persist behavior to their initial state (as if the layout has not been visited yet). This is useful when restarting from the beginning again.

Set collision cell size

Construct optimizes collision checks by sorting all objects in to "cells". The default cell size is the viewport size. Changing the collision cell size adjusts the trade-off between collision performance, memory use, and overhead of moving objects. Usually the default works well for most projects, but projects where there are large numbers of objects testing collisions in a small area, such as "bullet hell" style games, may benefit from a smaller collision cell size. Use performance measurements to identify the optimal size.

Set group active

Set an event group active or inactive. None of the events in an inactive group run until it is activated again. The event group is identified by its name.

Sort Z order

Sort the Z order of instances of a given object according to an instance variable. This effectively removes all instances from their Z order, sorts them, then inserts the sorted sequence back in to the holes left behind. This means if the instances are spread across a range of layers, they will be sorted in the same Z order locations, maintaining the same order relative to other instances on those layers. Note this action is much faster than using an ordered *For each* with an action like *Send to front/back*, so this action should be the preferred way to sort the Z order of large numbers of instances.

Stop loop

Stop a *Repeat*, *For* or *For each* loop currently running. These loops are system conditions. The rest of the event's actions and subevents will still complete, but the loop will not run any further after that.

Global & local variables

Add to

Subtract from

Set value

Alter the value stored by a number or text type global or local variable.

Reset

Reset an individual global variable in the project to its initial value.

Reset global variables

Reset all the global variables in the project to their initial value. If *Reset static* is checked, then all static local variables in the project will also be reset to their initial value.

Set boolean

Toggle boolean

Set or toggle a boolean type global or local variable.

Layers

For more information about the effect actions, see Effects.

Add layer

Create a new layer and insert it to the layer tree at runtime (also known as a *dynamic layer*). *Layer name* is the name to use for the added layer, which must be different to all existing layers already added, including other dynamic layers. *Insert by* is the name or index of

another layer to insert the new layer relative to. *Where* specifies where to insert the new layer relative to the *Insert by* layer; *Above* and *Below* add it as a sibling layer adjacent to the *Insert by* layer, whereas *Add top/bottom sub-layer* inserts the new layer as a sub-layer of the *Insert by* layer. The *Insert by* layer may be left empty when *Where* is *Add top/bottom sub-layer*, in which case the new layer is added as the top or bottom layer at the root level of the layer tree.

Move layer

Remove and re-insert a layer to a new location in the layer tree. This works similarly to *Add layer*, except the specified layer must already exist; otherwise the *Insert by* and *Where* parameters are used in the same way.

Remove layer

Remove a layer from the layer tree by its name or index. This also removes any sub-layers of the removed layer, and all objects on the layer or any of its sub-layers will be destroyed. A layout must have at least one layer, so the last top-level layer cannot be removed.

Remove all dynamic layers

Removes all layers added using the *Add layer* action, leaving only the layers added in the editor. All objects on the removed layers will be destroyed. This can be useful to reset the state of dynamic layers.

Set layer scroll

Restore layer scroll

Independently scroll a layer, regardless of where the layout is scrolled to. By default layers all follow the layout scroll position. Upon using the *Set layer scroll* action, a layer will stop following the layout scroll position, and remain scrolled at the position provided. The *Restore layer scroll* action reverts the layer to the default mode where it follows the layout scroll position.

Set layer angle

Rotate an entire layer by a number of degrees.

Set layer background color

Set the background color of a layer. Note if the layer is transparent, setting the background won't change the appearance unless you also make the layer opaque.

Set layer blend mode

Set the blend mode of a layer. See the *Effects* section for a description of the available blend modes.

Set layer effect enabled

Enable or disable one of the effects added to a layer on the current layout. This action cannot be used to alter layers from other layouts.

Set layer effect parameter

Change the value of one of the parameters for an effect added to a layer on the current layout. This action cannot be used to alter layers from other layouts. The parameter to change is specified by its zero-based index, i.e. 0 to change the first parameter, 1 to change the second parameter, and so on. To set the value of a color parameter, use the *rgb* system expression.

Set layer force own texture

Change a layer's *Force own texture* property at runtime. For more information see the property in the Layers manual entry.

Set layer HTML

Set whether a layer acts as a HTML layer. For more information see HTML layers.

Set layer interactive

Set whether the content on the layer responds to mouse and touch input.

Set layer opacity

Set the opacity (or semitransparency) of an entire layer.

Set layer parallax

Set the horizontal and vertical parallax rates of a layer.

Set layer rendering mode

Set the *Rendering mode* layer property to either 2D or 3D mode. This allows dynamically changing a layer between rendering modes at runtime.

Set layer scale

Set the scale (or zoom) of an entire layer, taking in to account its scale rate property.

Set layer scale rate

Set the *scale rate* property of a layer, which affects how quickly it scales (if at all).

Set layer transparent

Set the *transparent* property of a layer. If transparent, the background color is ignored.

Set layer visible

Show or hide an entire layer.

> *Note a sub-layer cannot be made visible if one of its parent layers is invisible. All its parent layers must be made visible too for it to appear.*

Set layer Z elevation

Set the Z elevation of an entire layer. By default the camera is at Z = 100, and looking down to Z = 0. The default Z elevation is 0. Increasing it will move the layer upwards (towards the camera) and decreasing it will move it downwards (away from the camera).

> *The layer order takes precedence over Z elevation. In other words, Z elevating a layer above a layer on top of it will not make it appear above that layer.*

Layout

Go to layout

Go to layout (by name)

Switch to another layout in the project. Note that global variables keep their current value - they are not reset. To reset them use the system action *Reset global variables*.

Go to next/previous layout

Switch to the next or previous layout in the project. The order as they appear in the Project Bar is used, where layouts at the top are first and layouts at the bottom are last. If on the first layout, trying to go to the previous layout does nothing, and if on the last layout, trying to go to the next layout does nothing. Note that global variables keep their current value - they are not reset. To reset them use the system action *Reset global variables*.

Recreate initial objects

Recreate the objects in a rectangular area of the layout as they appear in the Layout View. In other words, this restores a section of the initial level design. Note that this does not destroy

any existing objects, so if used repeatedly will create multiple objects sitting on top of each other. Only objects of the given type are created, or alternatively a family can be passed and all objects belonging to that family are recreated. The initial objects to create can also optionally be sourced from a different layout by specifying a layout name (leaving it empty will use the current layout). A specific layer can also be chosen by entering a layer name or number; the default of using -1 will use objects from all layers. The created instances can also optionally be placed on a specific layer with the *Destination layer* parameter (-1 means to try to match the source layer), and also offset from their default positions so they are created at a different location. As with the *Create object* action, all the created instances are also picked so subsequent actions can further alter them. Tick *Create hierarchy* when creating root objects in a hierarchy to automatically create the rest of the scene graph hierarchy with connections in place.

Restart layout

Restart the current layout. Note that unlike *Go to layout*, this action resets all event groups to their initial activation state. Global variables keep their current value - they are not reset. To reset them use the system action *Reset global variables*.

Set layout angle

Rotate an entire layout by a number of degrees. This rotates all the layers in the layout.

Set layout effect enabled

Enable or disable one of the effects added to the current layout. This action cannot be used to alter other layouts.

Set layout effect parameter

Change the value of one of the parameters for an effect added to the current layout. This action cannot be used to alter other layouts. The parameter to change is specified by its zero-based index, i.e. 0 to change the first parameter, 1 to change the second parameter, and so on. To set the value of a color parameter, use the *rgb* system expression.

Set layout projection

Change whether the layout uses a perspective or orthographic projection. See the *Projection* property in Layout Properties for more information.

Set layout scale

Set the scale (or zoom) of an entire layout. This scales all the layers in the layout, taking in to account their *scale rate* property.

Set layout vanishing point

Change the location of the vanishing point in the viewport for this layout, using a percentage in the range 0-100. This affects how perspective appears for 3D features such as Z elevation and the 3D shape object. For more information refer to the *Vanishing point* layout property.

Memory management

By default Construct loads and releases memory automatically, so normally you do not need to worry about how memory is managed in your project. However very large projects may need to control when objects are loaded in to and released from memory. These actions provide control over the loaded image memory. Note that spritesheeting combines different images on to the same spritesheets; the spritesheet will be loaded in to memory if any single image on it is loaded, and only released when no images on it are loaded.

Load layout images

Load layout images (by name)

Load all the images used by a layout in to memory. The layout can be chosen either from a list or using a string of its name.

Normally Construct will already have the current layout loaded in to memory. Since this action causes two layout's images to all be loaded in memory at once, it can cause a spike in memory use, risking an out-of-memory error. To mitigate this only use it on minimal layouts, such as a loading screen.

Load object images

Load object images (by name)

Load all the images used by an object in to memory. The object can be chosen either by an object picker or a string of its name. If it's already loaded, this has no effect.

Objects can only be loaded as a whole. Loading a Sprite will load all animation frames from all its animations. Avoid using objects with a great many animation frames since it forces Construct to load them all. Consider using multiple Sprites instead and using Families to keep the events simple.

Unload object images

Unload object images (by name)

Unload all the images used by an object from memory. The object can be chosen either by an object picker or a string of its name. If it is not yet loaded, this has no effect. If there are still instances of the object in use (i.e. the object's instance count is greater than 0), then this has no effect, since the images are still in use and cannot yet be released.

Note that destroying objects does not really release them until the end of the next top-level event. This means destroying all instances of an object and unloading its images simulatenously will not work. Use a Wait action to ensure an object's instances have been fully destroyed before unloading memory.

Unload unused images

Automatically unload the images for any objects that have their images loaded, but currently have had all their instances destroyed (so the instance count is 0).

If an object is only temporarily unused, e.g. an explosion that currently has no instances but will be created later, then this action will unload it and force Construct to load it on-the-fly when it is next created. This is inefficient and can jank the project. Use this action with care, when you know any currently unused objects definitely won't be created again.

Save & Load

For more information on using savegames, see the tutorial How to make savegames.

Load

Load the state of the project from a save slot. When completed, this triggers *On load complete*, or *On load failed* if the save slot has not been saved to before.

Load from JSON

Load the state of the project from a string of JSON data previously returned by the *SaveStateJSON* system expression.

Save

Save the state of the game to a save slot. When completed, this triggers *On save complete*.

Save to JSON

Save the state of the game to a string of JSON data. When completed, this triggers *On save complete*, and the resulting string of JSON data can be accessed with the *SaveStateJSON* system expression.

Scrolling

To scroll, the size of the layout must be bigger than the size of the viewport, or the layout's *Unbounded scrolling* property must be enabled. Otherwise there is nowhere to scroll to and scrolling will have no effect.

Scroll to object

Center the view on a given object. This scrolls all layers taking in to account their *parallax* property.

Scroll to position

Scroll to X

Scroll to Y

Set the X and Y positions to center the view on. This scrolls all layers taking in to account their *parallax* property.

Time

Set object time scale

Restore object time scale

Change the rate time passes for a specific object. This affects the object's behaviors and its own *dt* expression. Restoring the object's time scale returns it to the same time scale the rest of the project is using. See the tutorial on Delta-time and framerate independence for more information.

Set framerate mode

Change the project *Framerate mode* property at runtime. For more details, see the corresponding project property.

Set min/max delta-time

Set the limits on the delta-time (*dt*) measurement. If the real-world delta-time increases above the maximum delta-time, it stops increasing the measurement used by Construct, corresponding to a dropping framerate causing the project to run in slow-motion. Conversely if the real-world delta-time decreases below the minimum delta-time, it stops decreasing the measurement used by Construct, corresponding to an increasing framerate causing the project to run in fast-forward. The defaults are a minimum delta-time of 0 (meaning the project never goes in to fast-forward mode) and a maximum delta-time of 1 / 30 (corresponding to a framerate of 30 FPS), meaning the project begins to go in to slow-motion as the framerate drops below 30 FPS. Going in to fast-forward can be useful for a "catch-up time" mode, and going in to slow-motion with low framerates is useful to prevent objects stepping too far every frame which can result in skipped collisions and unstable gameplay.

> *Note the inverse relationship between the framerate and delta-time: an increasing framerate results in an decreasing delta-time, and a decreasing framerate results in increasing delta-time.*

Set time scale

Change the rate time passes at in the project. Useful for slow-motion or pausing effects. See the tutorial on Delta-time and framerate independence for more information.

Signal

Resume any events paused with a *Wait for signal* action with the given tag.

Wait

Wait a number of seconds before continuing on to the next action or sub-events. Other events continue to run in the meantime.

> *Note this does not stop loops. The rest of the events continue to run, including the remaining loop iterations.*

The *Use time scale* parameter determines whether the time scale affects the wait time. If unchecked, then the time waited is in real-world (wall clock) time ignoring the time scale. This can be useful when pausing by setting the time scale to 0: in this case waits using the time scale are paused indefinitely, but waits not using the time scale will still complete in real-world time.

Wait for previous actions to complete

Wait until any previous actions have completed before continuing on to the next action or sub-events. Other events continue to run in the meantime. This only applies to *asynchronous actions*, which show with a special icon. Normally these types of action take a while to complete and run a trigger like *On completed* when the action has finished. However using *Wait for previous actions to complete* is often more convenient, since it allows you to keep everything within the same event block.

> *If JavaScript code using the `await` keyword precedes this action, it will also wait for the JavaScript code block to finish. See the section Using 'Await' in Scripts in event sheets.*

Wait for signal

Wait indefinitely until the *Signal* action is used with the same tag. Other events continue to run in the meantime.