# WELCOME TO THE CONSTRUCT 3 MANUAL

**View online:** https://www.construct.net/en/make-games/manuals/construct-3

Welcome to the official Construct 3 manual! Construct 3 allows you to quickly and easily develop games directly in your browser. This manual provides a comprehensive reference of all of Construct 3's features.

The manual starts by covering the interface of Construct and how to get basic tasks done. Later on it provides a detailed reference of all plugins and behaviors, including the System object. Remember if you get stuck or run in to an issue, it's always worth checking the relevant manual entry in case it includes information that will help you solve your problem.

Construct also has links to the manual in various dialogs and menus to help you jump directly to the relevant manual page for that part of the software. Finally, dedicated users may want to read the manual to learn about all the possible features you can take advantage of. There may be many hidden gems you didn't know about, but are documented here in the manual.

## Errata

If you find a mistake or omission in the documentation, please file an issue on the Construct issue tracker with a link to the relevant manual entry and an explanation of what you think is wrong. We'll take a look and make any necessary updates.

## Creative Commons license

The Construct 3 manual is licensed under Creative Commons BY-SA 3.0. You are free to copy, re-publish, adapt, modify or otherwise use the material in the manual, providing you correctly attribute its source.

# GET CONSTRUCT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/getting-started/get-construct-3

Construct 3 runs right in your browser. **There's nothing to download or install!** Just visit editor.construct.net in your browser on any modern browser and Construct 3 will start. This makes it easy to switch between devices, use public computer terminals (even with strictly limited access), or painlessly deploy Construct 3 across a computer lab or office.

## Run Construct 3 as an app

You can add Construct 3 to your desktop or home screen (on mobile devices). This creates an icon on your device to launch Construct 3 like an app. This is a great way to reach Construct 3 more easily, and also saves space on your screen since it hides the browser address bar and tabs.

Sometimes an **Install as app** option will appear in the main menu when this option is available. Click this menu option to install Construct as an app on your device. Sometimes an *Install* icon will also appear in the address bar. Alternatively you can usually find an option to install Construct in the browser menu in Chrome and Edge, possibly under the *Save and share* or *Apps* submenu.

In Safari, you can find *Add to home screen* on iOS or *Add to dock* on macOS by pressing the *Share* button.

### Associate Construct with the .c3p file extension

Construct project files typically use the .c3p file extension. Once you have installed Construct as an app, you can then double-click a .c3p file to open it directly in Construct. The first time you do this, you may see a prompt asking you to confirm this is what you want to do. There should also be a checkbox to remember your decision so you won't be prompted again.

### Construct 3 in the Microsoft Store

If you use Windows 10+, you can also find Construct 3 in the Microsoft Store and install it from there if you prefer.

## Work offline

Construct works offline! You don't have to always have an active Internet connection. You only need to be online the first time you load Construct. After Construct first starts, after a while you should see a notification in the corner indicating Construct is ready to work offline. Make sure you wait until you see that notification. Then Construct will continue to work even if you go offline.

It's more convenient to use Construct offline if you use the *Add to desktop/shelf/homescreen* feature to create an icon on your device to run Construct, as described in the previous section.

> *We also recommend previewing a project while online to ensure the preview window is also fully saved for use offline. While the Construct editor does attempt to save the preview window for use offline when it starts up, some browser's rules about storage may block Construct from saving it this way, and only allow it when directly accessing the preview window.*

## Updates

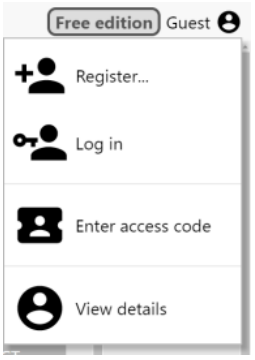
Construct automatically stays up-to-date. It will notify you when there's a new version available, and when it's updated. Our website also provides a list of all releases with detailed information about changes in each update, and also provides links to run older versions in case there's a problem with an update. You can check which version of Construct you're currently using by opening Construct's main menu and selecting About.

You can opt in to email notifications when updates become available in your subscription preferences. We also post news about updates on our Facebook page and Twitter account.

# USING AN ACCOUNT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/getting-started/using-an-account

When you first start Construct, you'll use it as a **Guest**. This means you are not logged in to an account. Construct shows your account status near the top-right corner. You can click this "badge" to show a menu with some account options.




*The Account menu*

## Account benefits

Until you purchase a subscription, Construct works in a limited *Free edition* mode, as indicated by the "Free edition" label on the account badge. The limitations of the Free edition are listed on the Construct 3 plans comparison page, where you can also purchase a subscription to remove the limits.

Guests have lower limits in the Free edition than registered users. Guests may only use up to 25 events in a project. Registering an account and logging in to Construct allows you to use up to 40 events, and then verifying your email address allows you to use **up to 50 events**.

If you purchase Construct, you must be logged in with the same account you purchased with to make use of the full features of Construct. The *Free edition* label next to your account will disappear to indicate you have an active subscription and no longer have the Free edition limits imposed.

At any time you can click the View details option in the Account menu to open a dialog displaying more information about your account.

## Working offline

If you purchase a subscription and work offline, you must start Construct while connected to the Internet at least once every 7 days to re-validate your subscription. However if you have not purchased a subscription, you can use Construct offline permanently.

## Access codes

If you have an **Education** subscription, you can create a time-limited access code in the subscriptions section of your profile. This allows students to use a licensed version of Construct for a period of time. The access code can be used for simultaneous users, up to the number of seats your subscription includes.

Once you have created an access code, share the code with your class. Students can then start Construct, choose the Enter access code option in the Account menu, type in the access code, and click OK. This will grant them access to the full version of Construct. This can also be done with a Guest account, so students do not need to register their own accounts. Once the access code expires, Construct will notify the user and revert back to the Free edition.

Access codes are designed for a classroom environment. Users with a Personal or Business subscription don't need to use access codes.

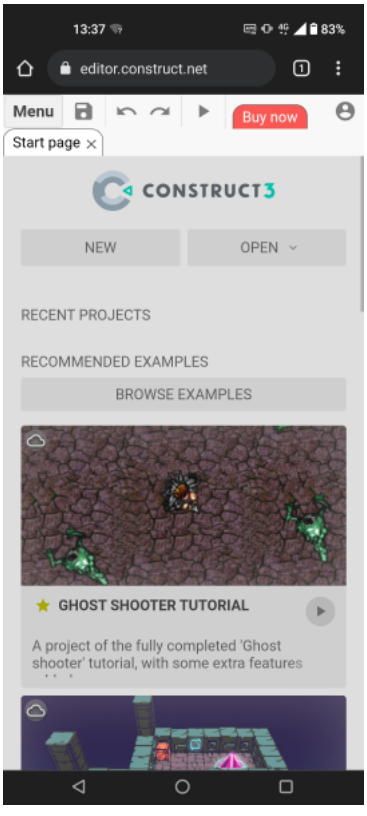
# CONSTRUCT ON MOBILE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/getting-started/on-mobile

Construct works on mobile devices like phones and tablets. (See System requirements for supported devices.) These devices typically use touch input, and often have small screens. While Construct can be used on a phone, it is much more comfortable to use a tablet device with a larger screen if you have one available.

> *Not all features of Construct are available on mobile. It's intended for light usage, such as reviewing and tweaking projects, rather than your main development work. For that we recommend using a device with a keyboard and mouse.*

Construct adapts its appearance to better suit these devices. On mobile it will look something like this.




*Construct mobile interface*

This manual is written primarily for desktop devices, since that is what most people will use for best productivity. However you can use touch equivalents to mouse clicks on mobile:

- Where the manual says to click or select an item, simply **tap** on it.
- Where the manual says to double-click, **double-tap** instead.

- Where the manual says to right-click, or open a *context menu*, **tap and hold** on the item. After a moment a menu will appear.

## Accessing bars

Since most mobile devices have small screens, Construct hides bars by default. To access these bars, **swipe in from the side** and the bar will slide in. Since there are a number of bars and only two sides, you can access the other bars by **repeatedly swiping in from the side again**. As you do this the previous bar will slide out and the next bar will slide in.

On the left side the sequence of bars is:

1. Properties Bar
2. Bookmarks Bar
3. Find Results bars
4. Z Order Bar

On the right side the sequence of bars is:

1. Project Bar
2. Layers Bar
3. Tilemap Bar

If you reach the end, the sequence will start again, cycling through the set of bars for that side of the screen. Bars can be closed by swiping them back the other way. The next time you swipe in from the side of the screen, you'll always get back the last bar you used that side. That helps you keep using the same bar for a while, and you can still keep swiping to switch between bars at any time.

### Animations editor

When using the Animations Editor on mobile, the same approach of swiping in from the sides can be used to access the Animation Editor's bars. This includes features like the color picker and animation properties.

## Changing UI mode

It's possible to connect up a mouse and keyboard to some mobile devices. If it has a large screen, this lets you effectively use a tablet like a small laptop.

If you do this, Construct may still use the mobile UI intended for touchscreens. You can make Construct switch in to desktop mode by opening Menu▶Settings and changing the **UI mode** to **Desktop**. This will always load Construct using the full desktop UI. If you disconnect your mouse and keyboard and go back to using the touchscreen, you may want to switch the UI mode back to **Automatic** or **Mobile** to restore the touchscreen UI.

# SYSTEM REQUIREMENTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/getting-started/system-requirements

These are the minimum system requirements for Construct to run.

## Internet connection

Construct can work offline. However you must be online to load Construct for the first time. When you load Construct for the first time, wait until you see a notification in the corner indicating Construct is ready to work offline. Then you can use Construct without an Internet connection.

Note that if you purchase a subscription and work offline, you must start Construct while connected to the Internet at least once every 7 days to re-validate your subscription. However if you have not purchased a subscription, you can use Construct offline permanently.

## Supported browsers

Construct should run in any modern browser. This includes:

- Google Chrome 87+
- Microsoft Edge 87+
- Other browsers that use the Chrome browser engine (Chromium), such as Opera and Yandex, providing they are updated to Chromium 87+
- Firefox 125+
- Safari 15.4+

While these are the minimum supported versions, we strongly recommend ensuring your browser is up-to-date with the latest version.

Construct does not support Internet Explorer, which was retired by Microsoft in June 2022. However in Windows 10 Microsoft replaced Internet Explorer with the Edge browser, which is supported from version 87+. (Note the modern Chromium-based Microsoft Edge is supported, but the legacy Edge is not supported.)

## Supported operating systems

Construct should run on any modern, supported system with an up-to-date browser. This includes:

- **Windows** 10, 11 or newer

- **Mac:** OS X / macOS 10.13 or newer
- **Linux:** 64-bit Ubuntu 18.04+, Debian 10+, openSUSE 15.2+, or Fedora Linux 32+
- **Chrome OS:** Any Chrome OS device updated to v87+
- **Android:** Any Android 8.0+ device with at least 1GB RAM
- **iOS:** Any iOS 15.4+ device

*Some browsers still support older operating systems such as Windows 7. However these are no longer officially supported and you may experience limited features if you continue to use them.*

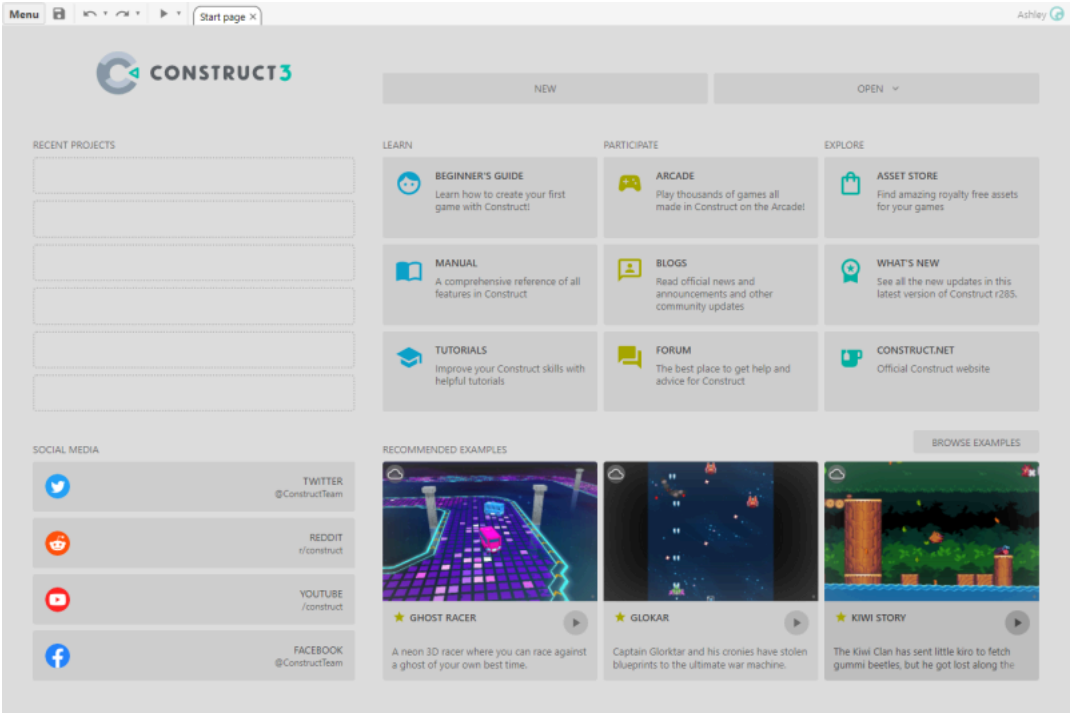
## WebGL support

Construct requires the browser to support WebGL, which is a modern high-performance graphics technology for browsers. Almost all modern devices support WebGL. However if you see a message about WebGL not being supported, try installing any available system updates, and check your graphics drivers are up-to-date.

# THE START PAGE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/start-page

When you first start Construct, it shows the Start Page. Note the appearance of the Start Page changes depending on the size of the window or screen. It will look something like this on a desktop display.




*The Start Page in Construct*

The Start Page gives you a useful starting point whenever you launch Construct. It provides shortcuts for tasks like creating a new project, opening an existing project including recent projects, and a set of useful links.

The Start Page initially fills the whole window. When you create or open a project, the rest of Construct's interface will appear.

## Starting a new project

Click **New** to create a new empty project. You'll be prompted for some basic details about the project to create. You don't need to enter anything though, just click **Create** and you'll get a new empty project with default settings.

## Opening existing projects

You can open projects from several sources: **Cloud** (projects saved to a cloud storage service like Google Drive), **local files and folders** (depending on browser support), or local browser

storage. You can also find previously saved projects in the **Recent projects** section. For more information about saving and loading projects, see Saving projects.

## Example projects

Construct comes packed with hundreds of example projects for you to learn from, or just try out for fun. Three randomly-picked recommended example projects appear along the bottom of the Start Page. Click the **Browse examples** button to open the Example Browser. You can learn more about it in the Example Browser manual entry.

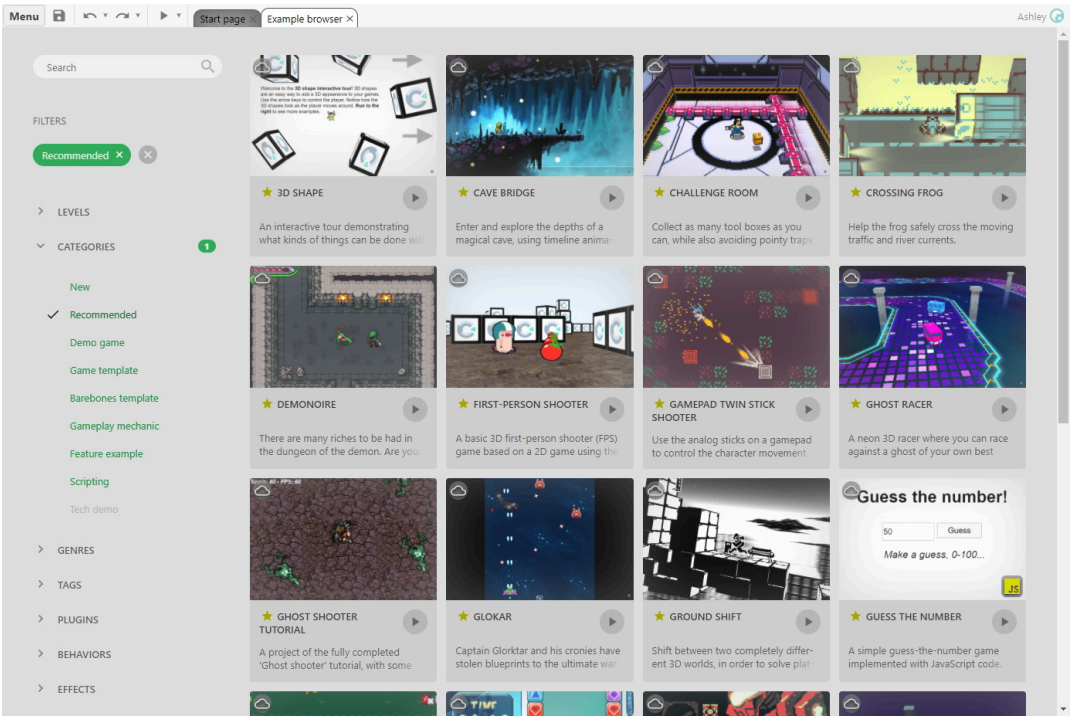
## Other resources

There are lots of links on the Start Page to other resources to help you get started and find out more about Construct. You can find links to community resources like the forums, social media accounts for Construct where you can follow news and updates, and other learning resources like tutorials.

# THE EXAMPLE BROWSER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/example-browser

Construct comes with hundreds of example projects to help you learn about it, as well as demonstrate the range of creative possibilities with Construct. Note the appearance of the Example Browser changes depending on the size of the window or screen. It will look something like this on a desktop display.




*The Example Browser in Construct*

There are several ways to open the Example Browser. The main ways are:

- Click **Browse examples** on the Start Page
- Click one of the three recommended examples on the Start Page to open it in the Example Browser
- Select Menu▶View▶Example browser

When no project is open, the Example Browser fills the whole window, like the Start Page. This makes it easier to browse the content. When you open a project, the rest of Construct's will appear.

## Browsing examples

Construct comes with hundreds of examples, so the Example Browser has lots of tags to help organise them. These are broadly organised like so:

- **Levels**: tags that indicate the approximate difficultly level to understand a project, covering *Beginner*, *Intermediate* and *Advanced*.
- **Categories**: tags describing broad categories of example projects. These include:
  - **New**: example projects added since the last stable release
  - **Recommended**: a hand-picked selection of the best or most interesting example projects
  - **Demo game**: complete games, covering title screens, multiple levels, and an ending
  - **Guided tour**: step-by-step interactive guides that show you how to get started with using various features of Construct. These are great for beginners or quick introductions to other features of Construct you might not have used before.
  - **Game template**: projects with a single level demonstrating a game concept
  - **Barebones template**: minimal projects with placeholder graphics demonstrating a game concept
  - **Gameplay mechanic**: projects demonstrating a specific mechanic of a game, such as a type of movement or special effect
  - **Feature example**: projects demonstrating some of Construct's features, showing how they work and what they can do
  - **Tech demo**: performance benchmarks and other demonstrations of the capabilities of Construct's engine
- **Coding**: tags that specify whether the project uses coding, and if so what language. Note that many coding examples come with both JavaScript and TypeScript variants. These tags include:
  - **Event sheets only**: projects that do not use any coding at all, and solely use event sheets for their logic.
  - **JavaScript**: projects that make use of JavaScript coding.
  - **TypeScript**: projects that make use of TypeScript coding.
- **Genres**: tags indicating the game genre of the example (if applicable)
- **Tags**: some other miscellaneous tags. These include:
  - **3D**: projects making use of Construct's various 3D features
  - **Mesh distortion**: projects making use of Construct's mesh distortion feature
  - **Mobile**: projects designed to work well on a mobile device with touch input
  - **Performance**: benchmarks or demonstrations of the performance of Construct's engine
  - **Hierarchies**: projects making use of Construct's hierarchies feature (i.e. connecting a child object to a parent)
  - **Timeline**: projects making use of Construct's Timelines animation feature
- **Plugins:** projects sorted by which plugins they use
- **Behaviors:** projects sorted by which behaviors they use

- **Effects:** projects sorted by which effects they use

**Click a tag** to toggle whether the list is filtering with that tag. A list of all filter tags appears in the *Filters* section. Only projects matching all tags will be listed. By default the *Recommended* tag is selected to show only the recommended example projects, but you can click the tag to remove it and filter the list another way.

You can also enter search terms in the search box. The list will further be filtered down to those matching both all tags and all the entered search terms.

## Previewing and opening projects

Click a project's card to expand it and see more details about it.

You can quickly preview a project by clicking its **Preview** button. To open the project so you can see how it works, click its card to expand it, then click the thumbnail image or the **Open** button. Once open you can then also try it out by pressing the **Preview** button in the main toolbar, or by pressing **F5**. Once you're done, close the project by selecting Menu▶Project▶Close project. When the project closes, you'll see the Example Browser again where you can choose another example project to preview or open.

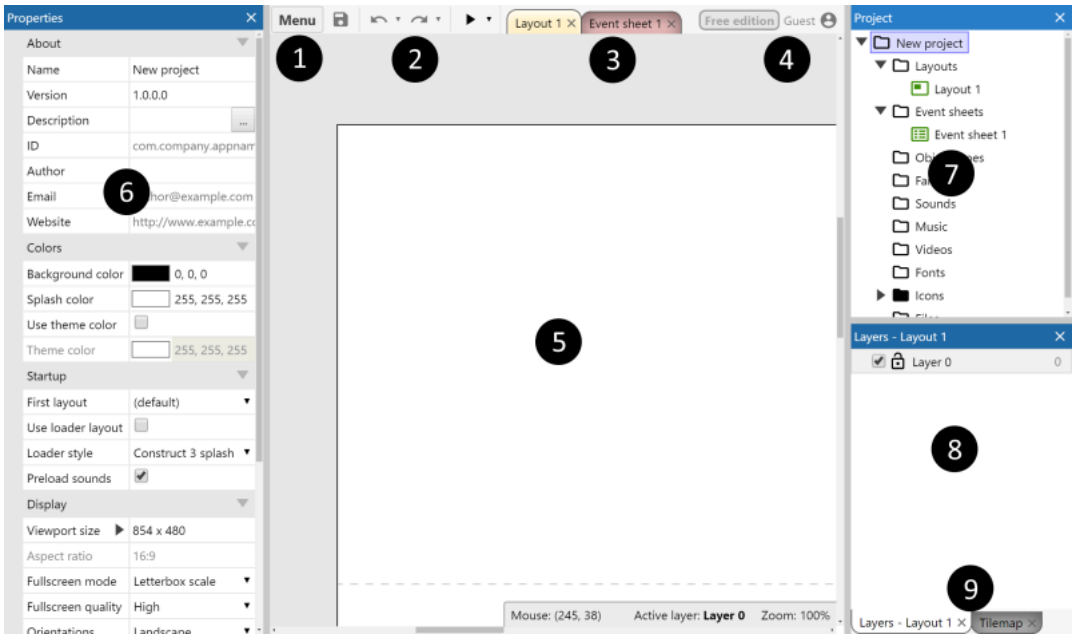
## Linking to examples

You can get a direct link to an example by **right-clicking a card** in the Example Project and selecting Copy direct link. This will copy a link with the example embedded, such as https://editor.construct.net/#open=kiwi-story. When visiting this link, Construct will load and automatically open that project. This is a useful way to share examples.

# CONSTRUCT'S INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/the-interface

The following image highlights the important parts of the Construct 3 user interface (UI) with numbers. Click the image to enlarge it. An overview of each part is provided below, and later the manual goes in to each section in more detail. Note that initially only the Start Page is visible. Much of the interface will not appear until you create or open a project. Also note the exact appearance of Construct can depend on which theme you have selected.




*The Construct user interface*

1. **Main menu button**: Click this to open the main menu. This provides options for basic tasks like opening and closing projects, exporting, changing settings and so on.
2. **Main toolbar**: This provides shortcuts to the most commonly-used features: save, undo, redo and preview. Note the arrows next to some buttons which provide a dropdown menu with more options.
3. **View tabs:** These tabs let you switch the main view between different layouts (where you place objects) and event sheets (where you define logic using the event system). You can customise the color of these by right-clicking them and using the **Colors** menu.
4. **Account badge**: This shows your account status. Click it to show the Account menu.
5. **Main view:** This is where the currently selected Layout View or Event Sheet View appears. The view tabs select which is visible. In this picture, it's showing an empty Layout View.
6. **Properties bar**: This lists all the properties for the selected item, allowing you to change settings for it.

7. **Project bar**: This lists everything in your project. It gives you an overview of what you've added, and lets you navigate around the project as well, such as by opening layouts or event sheets to view them.
8. **Layers bar**: When a Layout View is open, this shows the layers on the layout.
9. **Tabs:** By default the Layers bar and Tilemap bar are docked together. You can use these tabs to switch between the bars. You can drag and drop bars around to rearrange them. You can dock or tab together any combination of bars you want to customise the interface.

## Changing the theme

You can choose a different theme to alter the appearance of Construct's interface, such as by choosing a dark theme. The current theme can be changed from Settings. Third-party themes can also be installed as addons.

Note that the precise appearance of the interface depends on which theme you are using. The manual uses images based on the *Default (no theme)* style for a neutral appearance. If you change the theme, note the interface may look different to images in the manual.

# PROJECT STRUCTURE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/project-structure

Construct projects consist of the following elements. These can be accessed via the Project Bar, which contains a tree of all the elements in the project. Items in the Project Bar can also be organised in to subfolders Paid plans only which is very useful for medium to large projects. For more information, see the manual section on *Project primitives*.

## Layouts

Layouts are levels, menus, title screens and other pre-arranged *layouts* of objects. In other tools Layouts may be referred to as *scenes*, *rooms*, *frames* or *stages*. See the section on Layouts for more information.

Layouts also consist of multiple Layers, which can be used to arrange objects in to background and foreground layers. These are managed with the Layers Bar.

## Event Sheets

Event Sheets are a list of Events defining the game logic. In Construct, Events are the alternative to programming or scripting. Layouts have an associated Event Sheet for their logic. Event sheets can be re-used between different layouts with event sheet includes. Event sheets are edited in the Event Sheet View.

For more information on events, see the manual section on Events, especially the page on How events work.

## Object Types

An Object Type defines a 'class' of object. For example, *TrollEnemy* and *OgreEnemy* could be different object types. Multiple **instances** of an object type can be created. For example there could be three **instances** of the *TrollEnemy* **object type**, and five **instances** of the *OgreEnemy* **object type**.

It is important to be clear on the difference between **object types** and **instances**: they will be referred to as different things throughout the manual. For more information, see the manual entries for Plugins, Object Types and Instances.

Object Types can also be grouped in to Families Paid plans only.

## System object

The **System object** represents built-in functionality in Construct. It is the only object an empty project contains. It cannot be added again or removed from a project. There are no instances of

the System object: it is simply always present and provides access to the built-in aspects of Construct's engine. Its conditions, actions and expressions are documented in the System reference. Note the System object does not appear in the Project Bar, but it is still an important part of the project.

## Sounds and Music

These are audio files used for sound effects and music in the game. **Sounds** should be used for short-duration sound effects that are played when events like collisions and explosions happen. **Music** should be used for the longer musical tracks. It is important to organise audio files appropriately, because Sounds are loaded completely before playing, but Music is streamed. This means if a Music track is accidentally put in the Sounds folder, it would have to completely loaded before it started playing. However, audio in the Music folder can start playing immediately since it is streamed. For more information see Sounds & Music.

## Timelines and eases

Timelines are pre-designed sequences of changes over time. They often cover movement, such as changing an object's position, angle and size over time, but can affect many other kinds of properties too. To learn more about creating timelines, see the section on the Timeline Bar.

*Eases* are animation functions that determine how changes happen over time. There are a number of built-in eases in Construct, but custom ones can be designed in the editor too.

## Scripts

Construct also supports JavaScript and TypeScript coding in the editor. Event sheets can be combined with code, or code can be written in separate files. When using script files, they are added in the *Scripts* section of the Project Bar. For more information see the dedicated manual section on Scripting.

## Project Files

Additional files can be imported to or created in the project. These can then be loaded and used in your project. This also covers using a variety of media files in your project, such as videos and web fonts. For more information, see Project Files. Construct also provides some file editors Paid plans only for conveniently editing data files.

# SAVING PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/saving-projects

In Construct, there are several ways you can save your work. By default pressing **Save** on a new project will save with **Cloud Save**. You can select a different option, as well as change the save option at any time, in the Menu▶Project▶Save as menu.

## Cloud save

You can save your work to a cloud storage services, allowing you to access your work wherever you go. Since Construct runs in the browser and can be used on any device, this is a great way to ensure you can carry on from where you left off no matter which device you end up using. Many cloud storage services also provide built-in backups and file histories, helping ensure your work is safe even in the face of disaster.

Construct currently supports Google Drive, Microsoft OneDrive and Dropbox. The first time you select Menu▶Project▶Save as▶Cloud save, a dialog will appear asking you to choose one of the supported services. When you choose one, you'll be prompted to log in to your cloud storage account, so Construct has permission to save and open files from your account. Once you've entered your details they will be remembered, so you can keep using Cloud Save without having to keep entering your details.

> *Remember that your Cloud Save login is separate to your Construct login. The fact you are signed in to one does not automatically mean you are signed in to the other.*

When you press **Save** with a Cloud Save project, Construct will save your project and upload it to your cloud storage account. The upload will continue in the background showing the upload status in the corner of the window, allowing you to continue working on your project. Note you cannot save again until the upload completes.

Next time you use Construct, you can choose Menu▶Project▶Cloud open to find your project again. It'll also appear in the **Recent projects** section of the Start Page.

## Download a copy

Select Menu▶Project▶Save as▶Download a copy to download your project as a local file. Construct will ask if you want to change the downloaded filename; you can leave it empty to use the default. Construct projects use the **.c3p** file extension. Normally the file will go to your Downloads folder, but you may also be prompted to save to a different location depending on the browser. Alternatively you can usually drag-and-drop the resulting file directly out of the browser, such as from Google Chrome's downloads footer section.

Note that despite the name, this does not actually download a file from the Internet. All the project data is stored locally. The term *Download* refers to invoking the browser's download UI to save your project to a local file.

## Using local files

Currently some browsers such as Chrome support saving files directly to your system. This means you can use the Menu▶Project▶Save as▶Save as single file... option to save your project as a local file anywhere on your system. To open a local file, choose the Menu▶Project▶Open local file option. Then when you make changes and click the Save button, it will write back over the file you originally opened. You may see a permission prompt from the browser asking if you want to allow access to the file; be sure to allow permission to ensure your save works correctly.

*When saving to local files, be sure to set up backups to help avoid the risk of losing your work in the event of disaster. You can enable automatic backups in Construct's settings. See best practices for more information.*

## Using project folders

Where browsers support local files, they also allow the option to use project folders. These work similarly to saving local files, but instead of choosing a file, you select a folder to save to. Construct then saves the entire project as separate files within this folder. Be sure to choose an empty folder to avoid ending up with a confusing mix of files.

This option is good for very large projects, since saves are faster, as it only has to update the changed files in the folder, rather than generate an entire new .c3p file. It is also a good option to use with source control tools like GitHub, since you can track changes to individual text-based files - for a guide on that see the tutorial How to collaborate on Construct projects with GitHub.

This option can also be useful if you work with lots of JavaScript files in a Construct project and want to use an external editor with them. When saving as a folder project, new options will appear in the menu when right-clicking the script folder in the Project Bar. These options allow you to reload all script files from the project folder again either as a one off (also by pressing F9), or automatically every time the project is previewed. Note this reloading cannot be undone, so make sure you always make edits in the same place, as alterations within Construct will be overwritten when reloading. A similar approach is also used for using TypeScript in Construct.

*Construct's .c3p files are actually just a zipped folder project, with the extension .zip replaced with .c3p. You can convert a .c3p file to a folder project by renaming .c3p to .zip and extracting it. Similarly you can convert a folder project to a .c3p by zipping it, and renaming .zip to .c3p.*

*When saving to local folders, be sure to set up backups to help avoid the risk of losing your work in the event of disaster. You can enable automatic backups in Construct's settings. See best practices for more information.*

## Save to local browser

If saving local files is not supported, Construct provides an option to save projects to the local browser's storage instead. This storage is unique to both the specific device and browser. So for example if you save a project to browser storage on a specific laptop with Chrome, you can only find it again by using the same browser (Chrome) on the same device (that specific laptop).

Construct will ask for permission to use persistent storage the first time you use this option, to ensure the browser won't automatically delete your data. Note browsers sometimes also have storage limits. You can also check the status of the persistent storage permission, as well as how much space the browser is allowed to use and how much it is using, in the **About** dialog.

*If you use this option, be very careful about clearing your browser data. If you choose the wrong option while clearing browser data, you could still erase all your projects saved to browser storage. For this reason, using a different save option where possible is recommended.*

# SHARING PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/sharing-projects

The easiest and quickest way to share your project so someone else can run it is using **Remote Preview** Paid plans only. For more information see Testing projects.

If you want people to play your finished project, you should **Export** it. This produces a playable game ready for publishing. See the section on publishing projects.

## Share a project file

You can also save as a single file, or use the *Download a copy* option, to get a single .c3p file representing your entire project. Then you can share that file using other tools and services, such as by attaching it to an email, or uploading it to a storage service.

## Shared folders in cloud save

If you want to share your Construct 3 project itself, shared folders that you have access to also appear in Cloud Save when using Google Drive or Microsoft OneDrive. This can be useful for conveniently sharing your project files. For example in a classroom, all the students could save their work to a shared folder that the teacher has access to. Then the teacher has an easy way to access all the student's work.

All cloud save services also provide their own sharing options via their websites, e.g. for sharing a file on your account with someone else.

## Sharing with subscription administrator

For education accounts, there is also a *Share with admin* option that appears in the Project menu. This only appears for two kinds of accounts:

- When using access codes
- When using an account created for a seat as part of an education subscription

In both cases, choosing *Share with admin* will upload the project to a server operated by Scirra. It will then be made available to the administrator who created the access code or seat account. The administrator can find it by visiting the Your subscriptions section of their account, and clicking the *N shared projects* link next to the relevant education subscription (e.g. *10 shared projects* if ten projects had been uploaded this way).

This approach is pseudonymous - only the access code or account username, plus the student-entered filename, are provided. Further the project files are automatically deleted after a couple of days. Therefore no information is permanently stored when using this option.

# COLLABORATING ON PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/collaborating-projects

Currently Construct does not support any real-time online collaboration, due to the extreme technical complexity of the feature. However you can use existing source control tools like **Git** and **SVN** to manage collaborative changes by a team working on the same project.

Source control tools were designed for programmers, but they work well with Construct as well. Make sure you use a folder-based project (see Saving projects) in a browser that supports it, e.g. Chrome. This saves your project as a series of individual files within a folder. Then when you make changes, these will appear in source control tools as a series of smaller changes to individual files. Construct's main project data files are in the text-based JSON format, which works well for identifying and merging changes. Each team member can then submit their changes, and these will all be merged in to a single folder-based project. If team members submit contradictory changes, these tools also provide options to resolve the conflict.

> *When using source control, be sure to configure the tool to ignore* `.uistate.json` *files. These are only used to restore the user interface (UI) state for individual team members, and aren't meant to be shared. These files are also optional and can be deleted at any time (although Construct's user interface will revert to defaults in places). Leaving these files out of source control will make sure each team member's user interface remains as they prefer, and avoids having to submit unnecessary changes.*

It's also advisable to change the *UID numbering* project property to *Random* when collaborating on projects. This changes UIDs assigned in the editor from incrementing (e.g. 1, 2, 3, 4...) to random (e.g. 582953, 295630...). This helps avoid potential problems that may arise when two people working on the same project create two different instances which each get assigned the same UID.

Web services like GitHub also exist to simplify setting up and running source control, and these too can be used with Construct folder projects. For a guide on how to set up working on a Construct project on GitHub, see the tutorial How to collaborate on Construct projects with GitHub.

# TESTING PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/testing-projects

To test your project during development, you can **preview** it by clicking the "play" icon in the main toolbar, by selecting Menu▶Project▶Preview, by right-clicking a layout in the Project Bar and selecting *Preview*, or by pressing **F5**. This will start your game from the current layout.

By default, starting a preview opens a popup window. You may see a message that the popup was blocked. Clicking **Try again** normally works, but to permanently prevent the message appearing you may need to change your browser's settings. Usually an icon or message will appear somewhere in the browser interface indicating a popup was blocked; clicking this usually provides a way to always allow popups for the current website.

In Menu▶Settings, you can choose different preview modes. The three options are:

- **Popup window:** as described above, opens a popup window to run the project in.
- **Browser tab:** opens a new browser tab to run the project in.
- **Dialog:** opens a dialog inside the Construct interface to run the project in. This does not use a new browser window so is not subject to popup blockers, and does not include other browser interface items like the address bar. However it cannot appear larger than, or outside of, the Construct window.

If you select *Preview* again with a preview already running, the existing preview window or dialog will restart and begin previewing the latest version of your project.

### Multiple preview windows

Sometimes it's useful to have multiple simultaneous preview windows open, such as for testing multiplayer games. To open an additional preview window you can use any of the following methods:

- Click the dropdown arrow by the preview button, and choose *Add another layout/project preview*
- Hold `Alt` while clicking the preview button
- Use the `Alt` + `4` or `Alt` + `5` keyboard shortcuts to start an additional preview of the project or layout, respectively

## Other preview types

In the main toolbar, there is a dropdown arrow next to the Preview button that shows a menu with more preview options. These can also be found in the Menu▶Project submenu, or by right-clicking the project name in the Project Bar.

### Debug layout

This runs the current layout in a special **debug** mode. The debugger is a special development tool which helps you inspect the state of the project (such as the value of expressions and variables). It also provides diagnostic tools such as advancing the game frame-by-frame, changing values, destroying objects, setting breakpoints in events, and more. This can bring invaluable insight to how your project is working, particularly if you run in to a problem. For more information see the manual section on the Debugger.

### Preview project

This starts a preview from the first layout in the project. This is either the first layout that appears in the Project Bar, or whichever layout is set in the *First layout* project property.

## Remote preview

Paid plans only This allows you to preview your project on a different device. It is also useful for testing different browsers on the same device. Starting a Remote Preview does not actually directly run your game. It will open a window that provides a special URL you can use to load the game, or a QR code to scan. All you need to do is open the URL on another browser or device, or share the URL with someone else, or scan the QR code, and the project will start to load and run in the browser. The project is loaded directly from your device using a peer-to-peer connection - it is not uploaded anywhere else, but is still accessible from anywhere on the Internet. The game is no longer available from the provided URL **as soon as you close the Remote Preview window**, or press the **Stop** button.

You can dock the Remote Preview window somewhere else in the user interface, much like you can with bars like the Project Bar and Properties Bar. This can be a useful way to keep it out of the way when hosting a long-running Remote Preview session while continuing to edit the project. When the Remote Preview window is resized to a narrow width it will use a simplified display with less information to better fit in a bar docked to the side of the main Construct window.

Once a different browser starts running the project by loading the Remote Preview link, it will appear in the Remote Preview window as a connected client. You can have multiple copies of the project running simultaneously.




You can view some basic system details and real-time performance information for connected clients, including their browser and operating system, which layout they are on, the framerate and approximate CPU usage (and approximate GPU usage if available), and their graphics hardware. You can also click **Request video** to see a video stream of what that client can see. There is also a dropdown to choose a different first layout in case you want to test a specific layout with Remote Preview.

Like with a normal preview, you can **update a remote preview** by pressing the *Preview* button again. This updates the version of the project available at the same URL. Existing clients will be notified of an update and will see the update if they manually reload. Alternatively clicking the **Reload all** button will force all clients to reload. If you want to switch to a different kind of preview, such as the usual local preview or to use the debugger, then stop or close Remote Preview before selecting a new preview option.

Clients who are viewing your project via Remote Preview will see notifications in the following situations:

- When the host **updates the project**, clients will see a notification indicating an update is available. They must reload their browser to load the new version.
- When the host **closes the Remote Preview window** or presses the **Stop** button, the remote preview ends. Clients will see a notification that the host disconnected. Clients can continue to run the project (they are not cut off), but if they reload the project will no longer be available.
- When the host **starts or stops video** the client will be notified.

Remote Preview allows you to instantly share your project to anyone in the world with an Internet connection. This is particularly useful if you have remote testers or reviewers. On the other hand you can use the remote preview URL on the same device for cross-browser testing, such as using Remote Preview to test your project in Firefox while Construct runs in Chrome. In

this case data is not sent over the Internet and is only transferred across the local system. Similarly if you Remote Preview to a device on the same local area network (LAN), such as a mobile device, most browsers will try to establish a local connection for data transfer ensuring the project can load at a much faster LAN speed rather than transferring via the Internet.

# PUBLISHING PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/overview/publishing-projects

To publish a project, use the Menu▶Project▶Export option. Construct allows you to export to a range of platforms, such as the web (HTML5), Android and iOS (via Cordova), and desktop apps (via Windows/macOS/Linux wrappers). Each platform has an accompanying tutorial to guide you through how the exporter works and covering any exporter-specific settings. Note in some cases fees may be involved, such as for developer subscriptions to allow uploading to app stores, or for hosting web content if you don't already have a server.

Every kind of export has the following common settings:

- **Deduplicate images** will search the entire project for identical images and remove the duplicates. This helps save memory and reduce the download size by removing redundant images.
- **Lossless format** lets you choose what format to export images in your project set to use lossless quality (i.e. perfect).
- **Lossy format** lets you choose what format to export images in your project set to use lossy quality (i.e. allowing some reduction in quality in order to allow a greater reduction in the file size).
- **Optimize images** will optimize all the lossless images in the project with enhanced compression. This spends longer trying to compress images to an even smaller size. This can make exporting take longer but sometimes significantly reduces the download size of the exported project. This process is lossless, so is guaranteed to preserve the quality of all your artwork.
- **Minify script** will obfuscate and compress the main JavaScript file for your exported project. This also helps reduce the download size, improves load time, and makes it significantly more difficult to reverse-engineer the project. Normally *Advanced* mode is safe to use, but if you use the scripting feature you may want to switch to *Simple* mode or adjust how you write your code - see Exporting with Advanced minification in the scripting section for more details.

> *Minifying script uses a cloud service, so you must be connected to the Internet for it to work.*

> *Debug advanced mode is only intended for diagnosing possible problems with Advanced mode - don't use it when publishing your finished project.*

Each option helps optimise the exported project, but can make the export take longer. It is recommended to enable deduplication, image optimization and script minification when exporting the final finished project for publishing. However if you are simply doing a trial export, you may wish to disable them to speed up the process.

Web-based export options also have a checkbox to enable **Offline support**. This is enabled by default and allows all exported projects to continue to work offline after the first time they are loaded (see Offline games in Construct for more details). However this can sometimes interfere with testing if you are regularly updating a web-hosted project, so it can be disabled, but it is recommended to leave it enabled when publishing for release.

Most export options will prompt you for additional settings specific to that exporter. For example the Android exporter has a setting for the minimum supported Android version.

When the export finishes, you will be provided with a link to download a zip file containing the final exported project. You can also view some statistics about the export, such as how much download size was saved by image optimization (if it was enabled). The Export Manager can also be used to find the last few exported projects and download them again.

## More information about publishing

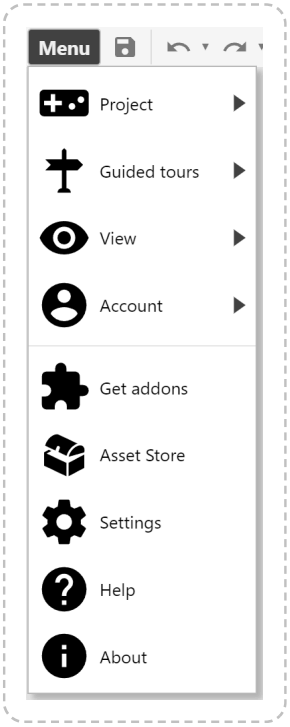
The publishing process is different depending on the chosen export option. The Tutorials section has guides to help you export to each platform. Here are some to get you started:

- Publishing to the web
- Publishing to the Scirra Arcade
- How to publish mobile apps
- Exporting to Windows
- Exporting to macOS

# THE MAIN MENU

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/main-menu

Click the **Menu** button next to the main toolbar to access the main menu. Note the exact options which appear in the menu depend on whether you have a project open, and whether you are logged in.




The menu structure, as a guest with a project open (which shows most options), is as follows:

- **Project**
  - **Save:** save the current project to its last saved location. If it has not been saved before, this will use the *Default save location* in Settings.
  - **Save As** (see Saving projects for more information)
    - **Cloud save:** save to a cloud storage service.
    - **Save as single file** (where supported by the browser): save to a local .c3p file representing the entire project.
    - **Save as project folder** (where supported by the browser): save as multiple files in a local folder, which is more suitable for large projects.
    - **Save to local browser** (where file/folder options are not supported): save to the browser's storage on the device.
    - **Download a copy:** save a copy of the current project by downloading a file.
  - **Preview:** run a preview of the current layout.

  - **Remote Preview:** start a Remote Preview of the current project. See Testing projects.
  - **Debug:** run a preview of the current layout with the debugger.
  - **Export:** export the current project for publishing. See Publishing projects.
  - **Close project:** close the current project, prompting to save if there are any changes.
  - **New:** create a new empty project.
  - **Open:** open a previously saved project from the cloud, local browser, or a local file. See Saving & sharing projects.
- **Guided tours:** these are a series of interactive, step-by-step tutorials shown within the editor itself. They're a great way to learn about the basics of Construct.
- **View**
  - **Bars:** in this submenu, you can hide and show any of the bars visible in the interface. If a bar is missing, use this menu to bring it back.
  - **Start page:** show or hide the Start Page.
  - **Addon manager:** open the Addon Manager to view and manage addons. See Installing third-party addons.
  - **Export manager:** open the Export Manager dialog, which lists the last few exports and allows you to download them again.
- **Account** (see Using an account)
  - **Register...:** register a new account to use with Construct.
  - **Log in:** log in to an existing account.
  - **Enter access code:** enter an access code to allow temporary use of the full version of Construct. This is typically used by educational institutions only.
  - **View details:** open a dialog displaying more information about the current account.
- **Get addons:** visit the Addon exchange on the Construct website where you can find new plugins, behaviors, effects and themes for Construct.
- **Asset Store:** visit the Asset Store on the Construct website to find art assets, sounds/music, templates, other software and much more.
- **Settings:** customise Construct's settings to work the way you prefer. See Settings for more information.
- **Help:** opens this manual.
- **About:** open a dialog displaying information about this version of Construct, as well as credits, storage information and diagnostic details.

# THE MAIN TOOLBAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/main-toolbar

The main toolbar provides quick access to a few of the most commonly-used options in Construct. It appears next to the main menu button.




*The main toolbar*

The main toolbar has the following buttons:

- **Save:** save the current project to its last saved location. If the project has not been saved yet, this defaults to Cloud Save. For more information see Saving & sharing projects.
- **Undo** and **Redo:** undo the last performed action in the editor. After pressing undo, you can then redo the action again. Click the dropdown arrow next to the button to see a list of the undo or redo actions. Selecting an item from the list will undo or redo all the actions up to the chosen item.
- **Preview:** run a preview of the current layout. By default this opens a popup window; you may be prompted to allow popups. Click the dropdown arrow next to the button to see a list of other kinds of preview. For more information see Testing projects. The dropdown arrow menu also allows changing the function of the button to one of the other options, such as *Debug layout*.

# BARS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars

In Construct, many features appear in **Bars**. These are the panes that appear at the sides of the window by default.

Note that bars work differently on mobile devices. See Construct on mobile for more information.

Bars can be rearranged in the interface to suit your tastes. Drag-and-drop a bar by its caption to move it. They can be left "floating" (appearing on their own anywhere in the Construct window), or "docked" to a side of the window, or with another bar. When you are dragging a bar, indicators will appear on screen as you move it around, showing where you can dock it. Move the mouse over one of the indicators and release the mouse button to dock the bar at that location. You can dock bars over other bars, which creates a split view, or directly on top to create a tabbed view of multiple bars.

You can reset the layout of the bars by clicking the **Reset bars & dialogs** button in Settings and then reloading Construct.

## Opening separate windows

You can open bars in to a separate window and use them outside of the main window. This is especially useful on multi-monitor displays, allowing you to move editors to other monitors.

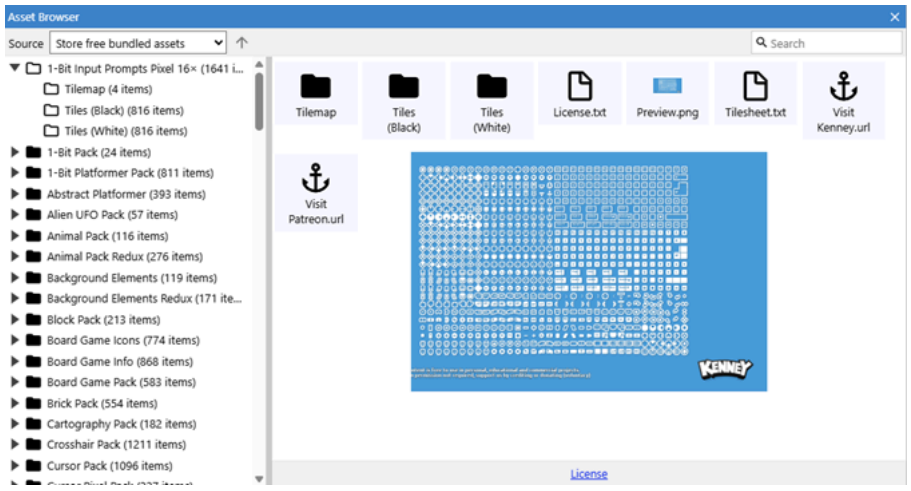
To open a popup window, undock a bar so that it is floating. Then **right-click** in the bar's caption at the top, and select Open to popup window.

When you close the popup window, the bar will automatically re-appear as floating in the main window.

# ASSET BROWSER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/asset-browser-bar

The **Asset Browser** is used to browse for assets and quickly import them into a project.




*The asset browser*

There are four main sources for assets that can be picked from the dropdown menu:

1. **Store free bundled assets:** these asset packs are bundled with C3 and available to all accounts, free and subscribed.
2. **Store premium bundled assets:** these asset packs are bundled with C3 and are available to accounts with an active subscription only.
3. **Store purchased assets:** these asset packs are the ones that have been purchased in Scirra's Asset Store.
4. **Local assets:** the asset browser can be used select one or more folders in the local file system. This feature's availability is subject to browser support.

## Importing assets

Assets can be imported into a project in three main ways:

### Double clicking

The quickest way to import a single asset. Depending on the type of the asset, something different will happen when importing. Below is a table showing how the different types will be handled.

| Type | Description |
| --- | --- |
| Images | Are imported as a new Sprite with a single animation and a single frame. |

| Type | Description |
|---|---|
| Animations | Formats such as **.gif**, are imported as a new **Sprite** with an animation with as many frames as the asset has. |
| SVG | Are imported as a project file in the project's **Files** folder and a new **SVG Picture** using the corresponding project file. An instance is created in the middle of the currently active layout view. |
| Audio | A dialog appears before importing audio assets to choose their destination and are then imported into either the **Music** or **Sounds** project folder. |
| Videos | Are imported into the **Videos** project folder. |
| Fonts | Are imported into the **Fonts** project folder. |
| Scripts | Are imported into the **Scripts** project folder. |
| Other | Other supported types are imported into the **Files** project folder. |

*Double clicking on a folder will just open it, use the other methods to import all the contents of a folder at the same time.*

## Drag & Drop

Assets can be dragged and dropped onto the layout view or the project bar to import them into the project. Below is a table showing how the different types will be handled.

| Type | Description |
|---|---|
| Images | A dialog appears with options about how the assets should be treated:<br>• **As object types:** will create a new object type for each asset, each one using an asset as it's image.<br>• **As single object type:** will create a single object type with an animation with as many frames as assets where selected.<br>• **As icons:** all assets are imported into the **Icons & screenshots** project folder. |
| Animations | This applies for animated formats, such as *.gif* or *.apng*. A dialog appears with options about how the assets should be treated:<br>• **As object types:** will create a new object type for each selected asset, each one with an animation with as many frames as each asset has.<br>• **As single object type:** will create a single new object type with a unique animation for each imported asset. |
| SVG | Are imported as a project file in the project's **Files** folder and a new **SVG Picture** using the corresponding project file. |
| Audio | A dialog appears before importing audio assets to choose their destination:<br>• **As music:** all the assets are imported into the **Music** project folder.<br>• **As sound:** all the assets are imported into the **Sounds** project folder. |
| Videos | Are imported into the **Videos** project folder. |
| Fonts | Are imported into the **Fonts** project folder. |
| Scripts | Are imported into the **Scripts** project folder. |
| Other | Other supported types are imported into the **Files** project folder. |

*Dropping a single asset has the same effect as double clicking it.*

*Dropping on the layout view will create needed instances in the dropped position. Dropping on the project bar will only create object types.*

*Dropping multiple assets of different types will show any relevant intermediate dialogs to decide what to do with each asset type.*

*Dragging and dropping a folder will import all the assets, showing any relevant intermediate*

*dialogs to decide what to do with each type of asset.*

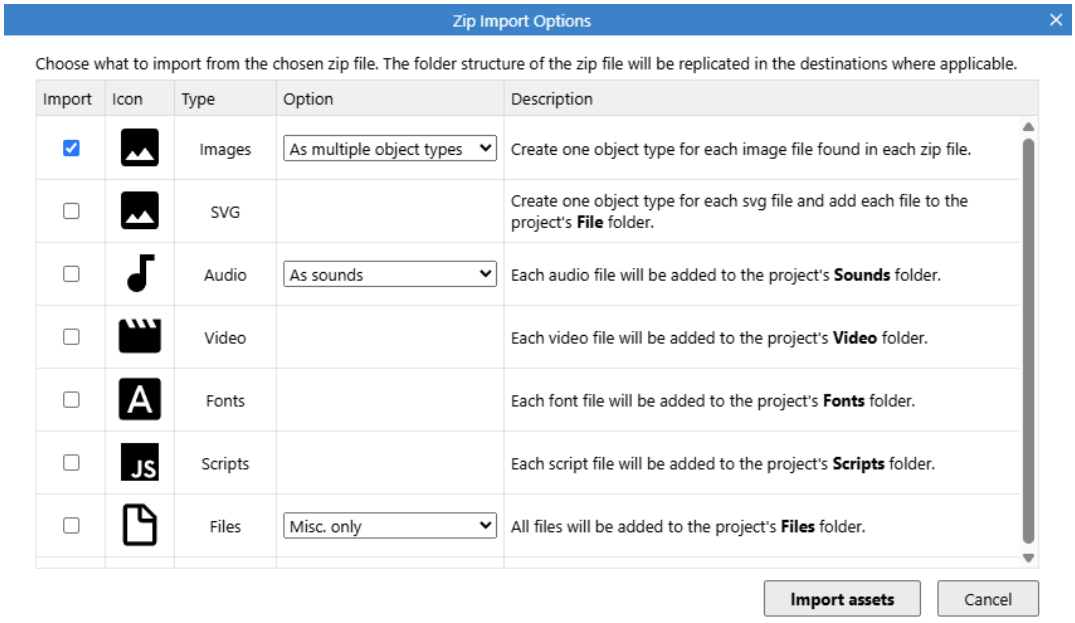
### Context menu

Assets can also be imported by clicking on the import option at the top of the context menu for a given asset. In the case of selecting a single asset or many assets of the same type, there will be a single import option. If the context menu is brought up when multiple assets of different types are selected, there will be options to import the assets of each type individually.

*The context menu for folders will show the import options for all the asset types in the folder.*

## Importing zip files

Zip files have a slightly different workflow to regular assets. After using any of the methods described previously to import, a dialog appears to choose which files of the zip file should be imported and what to do with them in the cases there are multiple options for them.




*Zip import dialog showing all possible options*

The image shows what the dialog looks like when the zip file has files of all the supported formats. In cases in which a format is not present in the zip file, then the corresponding option will not be shown in the dialog.

To import all the files of a given type tick the corresponding checkbox, if the asset type has additional options about how the assets should be treated, select it from the corresponding dropdown. Below is a table with all the possible options.

| Type | Description |
| --- | --- |
| Images | • **As multiple object types:** will create a new object type for each asset, each one using an asset as it's image. If the asset is an animated format, such as *.gif* or *.apng*, the asset is used to fill up an animation, if the chosen object type supports it.<br>• **As single object type:** will create a single object type with an animation with as many frames as assets where selected. Any animated formats are interpreted as an individual animation in the new object type.<br>• **As icons:** all assets are imported into the **Icons & screenshots** folder. Animated formats are not imported as icons. |
| SVG | Are imported as a project file in the project's **Files** folder and a new **SVG Picture** using the corresponding project file. |
| Audio | • **As sound:** all the assets are imported into the **Sounds** project folder.<br>• **As music:** all the assets are imported into the **Music** project folder. |
| Video | All assets are imported into the **Videos** project folder. |
| Fonts | All assets are imported into the **Fonts** project folder. |
| Scripts | All assets are imported into the **Scripts** project folder. |
| Files | • **Misc. only:** only assets which are not any of the other types are imported into the **Files** project folder.<br>• **All:** all the assets in the zip file are imported into the **Files** project folder. |

# Other Context Menu Options

Other options in the various context menus are the following:

| Option | Description |
| --- | --- |
| Download | Download the selected asset to the local file system. In case of selecting multiple assets, everything is compressed into a zip file. Using this option on a folder will download all the contents of the folder, compressing it into a zip file. |
| Preview | Bring up a dialog to preview the contents of the selected asset |
| License terms | Open a new page with the license associated with the asset. |
| Visit asset website | Open a new page with the asset's website. |
| Explore | This option appears when bringing up a context menu of a zip asset. Allows to explore the contents of the zip as if it was a folder. After using this option double clicking on the zip will open it as a folder instead of importing the contents. |
| Find folder in tree | Scroll the tree pane to show the folder a given asset belongs to. |
| Expand/Collapse All | Expand or collapse all the selected folders in the tree pane. |

# Settings dialog

Clicking on the three dot button next to the close button brings up the settings dialog for the Asset Browser, in it you can find the following options:

---

### Tag new frames with asset name when importing as many object types

Toggle this setting and the animation frame for each of the created object types will be tagged with the name of the corresponding asset.

---

### Tag new frames with asset name when importing as single object type

Toggle this setting and each animation frame of the created object type will be tagged with the corresponding asset's name. This option will not apply when importing an animated format, such as **.gif** or **.apng**, because the extracted images don't have a name. In the case of importing a zip file which has a **c3-import-setting.json** file used by the Animations Editor for bulk importing, any tag names in that file will be preferred over asset names.

# Browsing assets in the local file system

As mentioned earlier you can browse folders in the local file system, to do so first choose the **Local assets** option from the dropdown in the main toolbar and then click on the **+** button to choose a folder.

Different folders can be added and will be stored for future use.

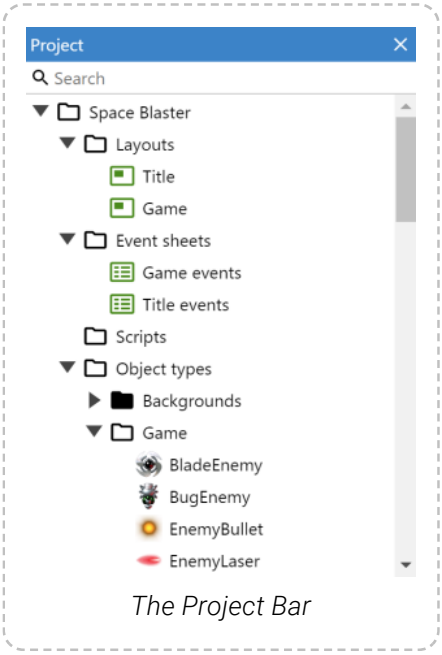
*This feature is only available as long as the browser supports it, if it's not supported it won't appear in the drop down.*

*You can also start browsing a folder in the local file system by dragging and dropping it on top of the asset browser.*

# THE PROJECT BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/project-bar

The **Project Bar** shows an overview of everything in your project. See Project Structure for a summary of the different elements that make up a project, or the manual section on Project Primitives for more detail.




*The Project Bar*

If something in the project has changed, it is displayed in *italics*. When you save the project, everything reverts to normal text, indicating nothing has changed since the last save.

To view the project properties, select the project item. This is the item at the top of the Project Bar with the name of the project. When you select it, the Properties Bar displays properties affecting the whole project. You can also **right-click the project item** to show a list of options, particularly the Tools submenu.

## Organising projects

The Project Bar can be used to arrange your project in to subfolders Paid plans only. Subfolders can be added by **right-clicking a folder** and selecting Add subfolder. Then, you can **drag and drop** folders and items to organise them in to folders.

You can hold **Control** or **Shift** to select multiple items and drag them in to a folder at the same time. However you can only organise items in to folders of the same type, e.g. you can't drag an event sheet in to a layout folder.

## Managing items in the project

**Right-click** any item in the Project Bar to show a list of options. Most items can be renamed and deleted. Right-clicking a folder also has the option to add a new item to that folder, such as a new layout or event sheet. Objects are more commonly added in the Layout View, but you can still add them from the Project Bar too.

## Deleting objects

Deleting from the Layout View will not remove an object from the project completely. The only way to fully remove an entire object type from the project is to delete it from the Project Bar.

## Searching the Project Bar

To search the Project Bar, simply **type in the search field** at the top of the bar. The contents of the Project Bar will filter down to matching items as you type, helping you quickly find things in your project.

Once you're done simply press `Escape` to clear the search and revert to the normal view of the Project Bar.

## Importing audio

**Right-click** either the **Sound** or **Music** folders and select Import sounds or Import music to open the Import Audio dialog. This allows you to pick audio files from your computer and import them to the project. Construct will convert them to the necessary format to support all browsers. To play back audio in your game, make sure you add the Audio object to the project. For more information see Sounds & Music.

## Importing files

You can import additional external files to the project, including web fonts. These can be categorised in to **Videos**, **Fonts**, **Icons** or the general-purpose **Files** folder. For more information, see Project Files and Icons & splash.

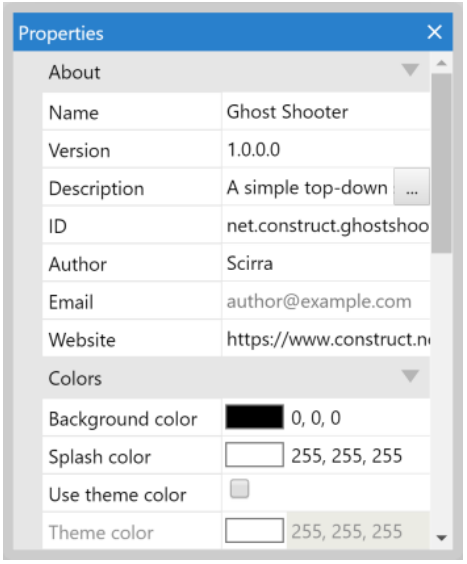
## Previewing files

From the Project Bar, you can preview several kinds of files added to the project. Audio and video files can be played back. Web fonts can be previewed with a dialog showing some text using the font. SVG files can also be previewed. Other kinds of file can be viewed and edited using the file editors Paid plans only.

# THE PROPERTIES BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/properties-bar

The **Properties Bar** is an essential of the interface. It displays a list of all the settings you can change on whatever is selected. The picture below shows the Properties Bar displaying a project's properties.




*The Properties Bar*

There are too many properties in Construct to list here. Instead, properties for different parts of the project are documented in the relevant manual section. For example, layout properties are described in the manual entry Layouts.

Properties are organised in to **categories** which can be expanded and collapsed. There are many kinds of properties, including number fields, text fields, dropdown lists and clickable links. The property name appears in the left column, and the editable value appears in the right column.

Whenever something in the project is clicked or selected, its properties display in the Properties Bar. For example, selecting objects in the Layout View or clicking items in the Project Bar shows the relevant properties in the Properties Bar.

## Things with properties

Some of the most important parts of the project with properties that you'll regularly modify are:

- Projects
- Layouts
- Layers

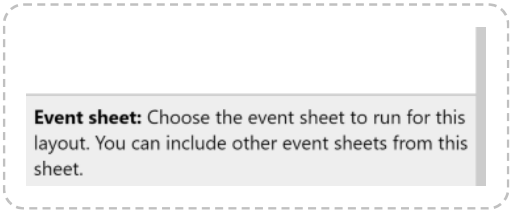
- Object instances

Many plugins, behaviors and effects have their own properties as well. See the Reference section of the manual for information on those.

There is also a **Help** link displayed at the end of every property list. Click that to open the relevant manual section for those properties.

## Property descriptions

All properties also have a *description* which provide additional information about what the property is used for. This is displayed in a panel at the bottom of the Properties Bar. It is worth keeping an eye on this since it can contain useful hints and tips. An example is shown below.




## Typing calculations

In number values, you can type calculations like `1920 / 2` and press enter to set the value to the result of the calculation (960). The syntax used is the same as expressions used in events. You can also use some basic system expressions like `sqrt(64)`.

## Dragging values

Number values can also be **smoothly dragged** with instant feedback in the Layout View. This is useful to try out a range of values and easily see which is best. To do this, **click and drag vertically** inside the number value cell. You can also hold **Control** or **Shift** while dragging to increase or decrease the rate of change. If you have trouble getting this to work, try first clicking inside the cell (which should select the text in the cell), and then click on the selected text and drag vertically.

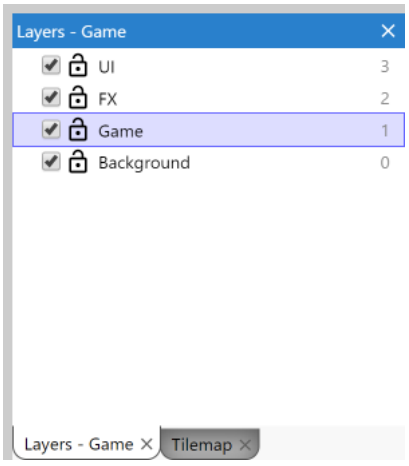
# THE LAYERS BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/layers-bar

The **Layers Bar** is used to add, edit and remove layers in a layout. A layer is like a sheet of glass objects are painted on to. This allows easy arrangement of which objects display in front of other objects, for example showing foreground objects in front of the background sprites. It also allows for interesting depth effects like Z elevation or parallax, and layers can be individually scaled and rotated as well.

Layers can be **dragged and dropped** in the Layers Bar to change their order. Layers at the **bottom** of the list are displayed **at the back** (e.g. background objects), and layers at the **top** of the list are displayed **at the front** (e.g. HUD objects).

Selecting a layer displays its properties in the Properties Bar and also sets it as the **active layer** which new objects are inserted in to.




*The Layers Bar*

## The Layers list

Each layer in the list has the following:

- A **checkbox** to toggle whether the layer is **visible in the editor** (this does not affect the game when previewing or exporting)
- A **padlock icon.** Clicking this toggles the layer's **locked** status. If a layer is locked, objects on that layer cannot be selected in the editor. This is useful to prevent accidental selections on rarely-used layers like backgrounds.
- An optional **globe icon** to denote global layers.
- An optional **tag icon** to denote HTML layers.

- A **number** to the right. This is a zero based index of the layer (the first layer is number 0, not 1). If you need to enter a layer number in the event system, this is the corresponding number. You can also enter layer names in the event system, which is often more convenient since unlike the numbers, the names don't change if you reorder layers.

You can **right click** a layer to see a menu of additional options, such as to add a new layer, rename or delete layers, or shortcuts to show, hide, lock or unlock all layers (or all but the selected one). Layers which are showing content from a global layer will show a context menu option to go to the layout where the original global layer is in.

## Sub-layers

Layers can also be added as sub-layers of another layer. They can be added as sub-layers or moved to be sub-layers via drag-and-drop. Sub-layers appear indented in the Layers Bar to show they come under another layer, and the layer they belong to can be expanded or collapsed to show or hide all its sub-layers.

Sub-layers can be used solely to help organize long layer lists, acting like layer folders. However applying an effect to a layer will also affect all its sub-layers. This allows for more efficiently processing effects (instead of having to add the same effect repeatedly to several layers), as well as more advanced effect composition across different groups of layers.

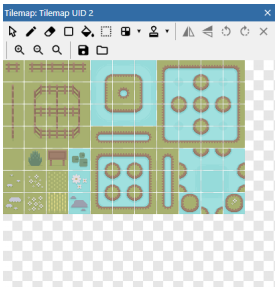
## Object Z ordering

To modify the Z order (front-to-back order) of individual objects on a layer, use the Z Order Bar
Paid plans only.

# THE TILEMAP BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/tilemap-bar

The **Tilemap Bar** allows editing tilemaps in the **Tilemap object** from the Layout View. It provides a toolbar with various tools and options, and a view of the current tileset image.



*The Tilemap Bar*

## Basic usage

To add a tilemap and start editing it, follow these steps:

1. Add a Tilemap object to the layout and make sure it is selected
2. Choose the Pencil or Rectangle tool from the Tilemap bar's toolbar
3. Select a tile in the tileset showing in the Tilemap bar
4. Click inside the Tilemap object to start drawing the selected tile

You can **hold shift** and **right-click** a tile in the Layout View to pick that tile to draw with. You can also hold shift and drag the right mouse button over a range of tiles to select that range of tiles as a patch you can stamp out.

To stop editing the tilemap's tiles and return to normal layout editing, click the mouse cursor on the Tilemap bar's toolbar to restore normal layout view selection. This also allows you to move and resize the entire tilemap object.

If you have multiple tilemap objects, only the selected tilemap is edited. It is often useful to layer tilemap objects directly on top of each other, in which case the tilemap to edit can be most easily selected using the Z Order Bar Paid plans only or hiding/locking layers with the Layers Bar.

If you are dealing with small tiles, you can also zoom the tileset image using the toolbar buttons. You can also access some of these options via a menu when right-clicking inside the Tilemap Bar.

There are a range of keyboard shortcuts that can be used when editing tilemaps. For more information, see the manual entry on Keyboard shortcuts.

If the bar is not visible anywhere in the UI, it can be opened by doing either of these:

1. In the main menu button make sure the View►Bars►Tilemap Bar option is ticked
2. Right-click on a tilemap instance in the Layout View and choose the option Tilemap►Open Tilemap bar...

## Toolbar tools

The Tilemap Bar's toolbar has the following options:

- **Normal layout view selection:** stop editing tiles and select the Tilemap object like any other object.
- **Pencil tile tool:** draw tiles with the mouse. You can also select an area of tiles by dragging across several tiles in the displayed tileset, and then use this tool to stamp that region of tiles in to the tilemap. You can also hold shift and right-click to drag an area over the Tilemap object to select a region of tiles to copy, or use the selection tool to do the same.
- **Erase tile tool:** erase tiles from the tilemap so they appear as transparent space. Larger areas can be erased by selecting a larger area of tiles in the tileset. A shortcut for erasing single tiles is to right-click while another tool is selected.
- **Rectangle tile tool:** draw a rectangular area of tiles by clicking and dragging in the Tilemap object. You can also select a 3x3 area of tiles in the displayed tileset, and the tool will **automatically nine-patch** the tiles. This also works for drawing single rows or columns with smaller selections such as 1x3 or 3x1, where the first and last tile are the first and last in the selection, and the rest are the middle tile repeated.
- **Fill tool:** much like using a fill tool in an image editor, this allows filling a continuous area with a new kind of tile. If multiple tiles are selected in the tileset, they are repeated over the fill area.
- **Select tool:** click and drag to select a range of tiles to use in the Tilemap object. Then switch to another tool to use that selection. For example switching to the **Pencil** tool allows you to stamp out copies of the selected range. A shortcut for this is to hold shift and right click and drag an area while the Pencil tool is selected. The individual tiles in the selection will be highlighted in the **Tilemap bar**
- **Auto tile tool:** this tool uses predefined brushes to automatically place the correct tile as you draw. Just pick the brush from the dropdown menu next to the tool's button and start using it. The brushes that are created by default are configured to work properly with the default tileset image. To create or modify brushes for use with different tilesets use the Tilemap Brush Editor.
- **Patch tool:** this tool uses brushes to draw a predefined set of tiles. Just pick the brush from the dropdown menu next to the tool's button and start using it. The brushes that are created by default work correctly with the default tileset image. To create or modify brushes for use with different tilesets use the Tilemap Brush Editor.
- **Mirror:** when using the Pencil tool, tiles will be placed flipped horizontally. This can also apply to an entire patch of tiles.

- **Flip:** when using the Pencil tool, tiles will be placed flipped vertically. This can also apply to an entire patch of tiles.
- **Rotate anti-clockwise:** when using the Pencil tool, tiles will be rotated 90° anti-clockwise. This can also apply to an entire patch of tiles. Click repeatedly to keep rotating tiles another 90°.
- **Rotate clockwise:** when using the Pencil tool, tiles will be rotated 90° clockwise. This can also apply to an entire patch of tiles. Click repeatedly to keep rotating tiles another 90°.
- **Reset transformation:** restores tiles to no mirror, no flip and no rotation.
- **Zoom in**, **Zoom out**, **Reset zoom**: adjust the zoom of the source tileset image displayed in the Tilemap Bar. This is useful if you are dealing with particularly small tiles.
- **Save to TMX:** export a zip with the current tileset image and the current tiles as a .tmx file (as used by the Tiled editor). Note that Construct does not support all of Tiled's features, so importing then exporting a TMX may lose some data, such as terrain definitions. Also since in Construct a Tilemap object represents a single layer of tiles, the exported TMX file will also only ever have one layer.
- **Load TMX**: import a .tmx tilemap as used by Tiled. All the tiles in the object are replaced with tile data from the TMX file. In Construct a Tilemap object represents a single layer of tiles, so if the TMX file has multiple layers you will be asked which layer to import. To import all layers, create a different tilemap object for each layer and import them separately. The tileset image can also be replaced by choosing a new image file. Note you can also **drag-and-drop** individual .tmx files, image files, and .zip files of both, in to the Tilemap Bar. This opens the load TMX dialog with all relevant fields already filled in, so you only need to press *OK*.

## Editing tile collision polygons

Each tile can have an individual collision polygon which is used when testing for collisions with the tilemap object. To edit a tile's collision polygon, **double-click the tile** in the Tilemap Bar. The Animations Editor will open to edit that tile. You can use the **collision polygon tool** to edit the tile's collision polygon. While the tool is active, you can also right-click and choose **Toggle collision polygon** to disable collisions for that tile entirely, such as if it is for decorative purposes only.

You can also use the image editing features of the Animations Editor to alter the image of the tile.

When hovering the mouse over a tile in the Tilemap Bar, its collision polygon is shown as an outline, if it has one. This helps you to quickly review the collision polygon set for each tile.

### Bulk editing

There are four context menu options to toggle the state of multiple collision polygons at the same time, they are the following:

- **Enable selected tile collisions:** enable the collision polygon of all the tiles highlighted in the **Tilemap bar**.
- **Disable selected tile collisions:** disable the collision polygon of all the tiles highlighted in the **Tilemap bar**.
- **Enable all tile collisions:** enable all the collision polygons of the tilemap.
- **Disable all tile collisions:** disable all the collision polygons of the tilemap.

*Using the **Select tool** will highlight the individual tiles in the **Tilemap Bar**, so it is easy to toggle the collision polygon state of a group of related tiles after making a selection in the tilemap instance.*

## The Tilemap object

For more information on how to use tilemaps, see the manual entry on the Tilemap object.

# THE TILEMAP BRUSH EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/tilemap-bar/tilemap-brush-editor

## The tilemap brush editor

This editor is used to create, edit and delete auto-tiling and patch brushes for use with the Tilemap Bar.

There are three types of brushes supported, the 16 tile auto tiling brush, the 47 tile auto tiling brush and the patch brush. Brushes are created by default with each instance of the Tilemap plugin. The default brushes are setup to work correctly with the default tilemap image.

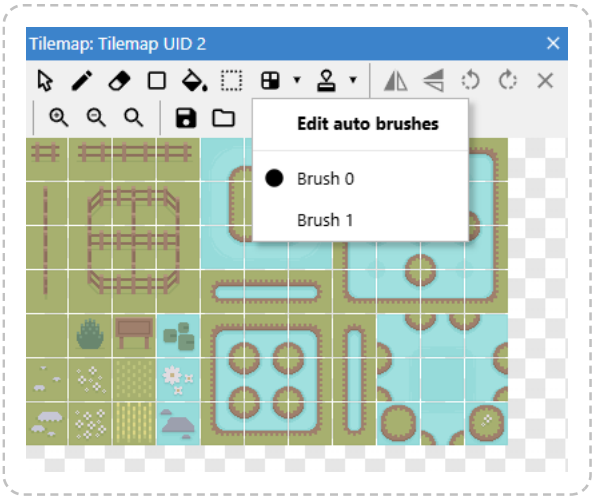
When the editor is opened from the auto tiling tool it will show the first auto tiling brush in the corresponding object type. If it is opened from the patch brush tool, it will show the first patch brush in the corresponding object type. This is just for convenience as both types of brushes can be edited regardless from which part of the UI the editor was opened from.

### Using the editor with auto tiling brushes

Here is a short summary on how to create, edit and use an auto tiling brush.

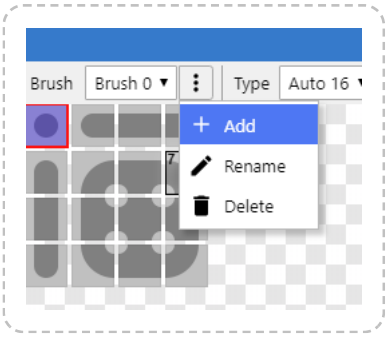
#### Step 1: Open the editor

Click the **Edit brushes** option from the dropdown menu of the auto tiling tool button in the Tilemap Bar toolbar.

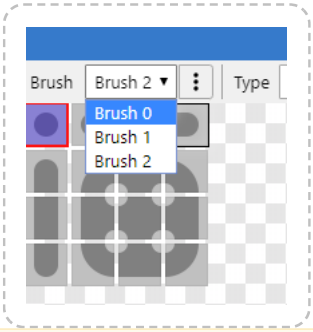



#### Step 2: Create (or edit) a brush

Click the **Add** option from the editor's toolbar.




If you don't need to create a new brush, one of the existing ones can be picked from the first dropdown in the editor's toolbar.




*Make sure the **Type** dropdown is either **Auto 16** or **Auto 47** if you want to create or edit an auto tiling brush!*

### Step 3: Make changes to an auto tiling brush

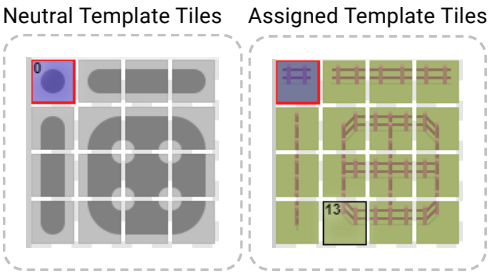
In order to properly set up an auto tiling brush, you will need to choose which tiles of the tileset will go in each position of the template. To do that, click on the tile you want to change in the template **(left pane)**. If this is a new brush it's best to start with the top left tile. After clicking the template tile, it will become highlighted. Then choose the tile from the tileset **(right pane)** that will go in that position.

After doing that, the selected tile will be assigned to that position and the next tile in the template will be highlighted. From here, all that needs to be done is to repeat the process until all the tiles in the template have a value assigned to them.

After you are done you can use the **"Toggle tiles"** button in the toolbar to quickly review what tiles have been assigned to each position of the template.

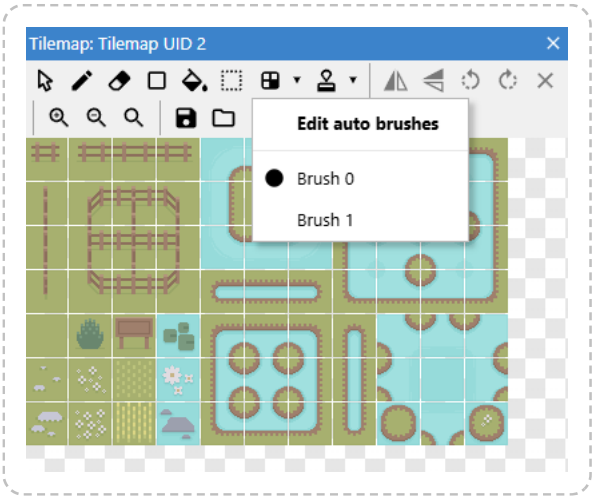
Toogle tiles button






**Step 4: Using an auto tiling brush**

Click the brush you want to use from the dropdown menu of the auto tiling tool button in the Tilemap Bar toolbar.




Make sure the auto tiling tool is activated (by clicking the main part of the button) and start drawing on the tilemap instance as you would with any other tilemap tool.

## Using the editor with patch brushes

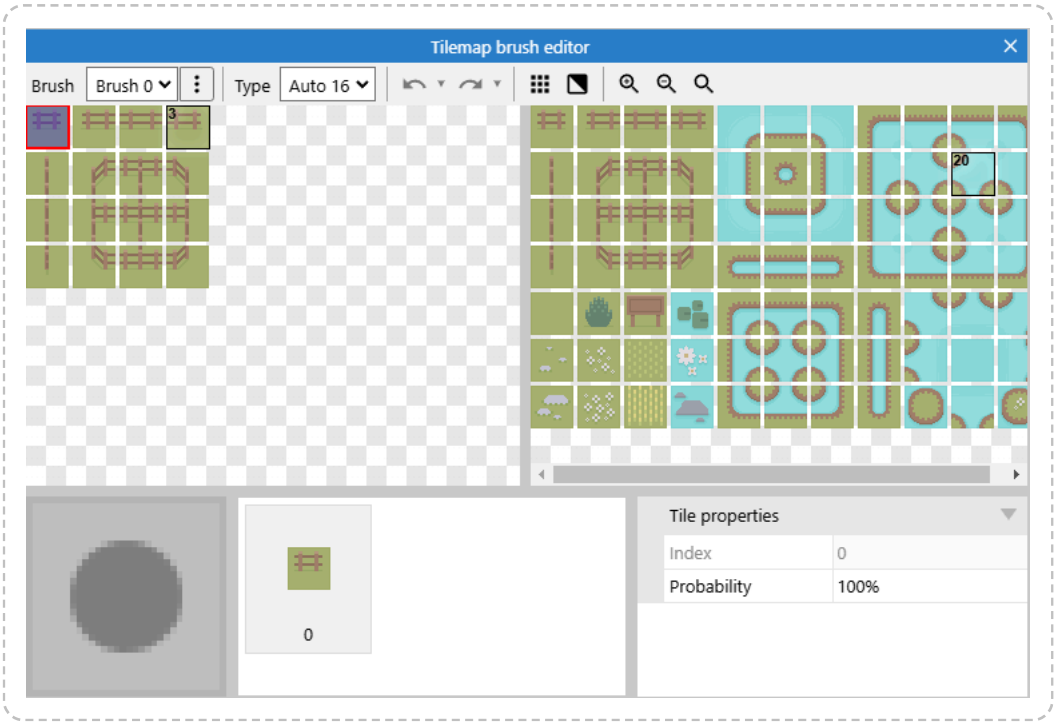
Creating and editing patch brushes is similar to auto tiling brushes, but since they perform different tasks there are a few differences in the UI.

When the type of a brush is changed to **"Patch"** the top toolbar will include **"Width"** and **"Height"** fields to set the size of the patch. The **"Toggle tiles"** button is not present when editing a patch brush.

Asides from that, the process of selecting tiles for each position of the patch is the same as with auto tiling brushes.

## Interface

This dialog is separated into 5 different panes which are described below.




### Panes

- **Template:** located in the top left, this pane shows the template of the auto tiling brush being edited. Its purpose is to select the tile of the template to be modified. Clicking on a template tile will update the **Current tile**, **Assigned tiles** and **Tile properties** panes. The template shown will change depending on the type of the current brush.
- **Tileset:** located in the top right, this pane shows the tiles being used by the tilemap object type. Its purpose is to select tiles to assign to the currently selected template tile. Just selecting a tile from here will assign it to the current template tile and move to the next. If you bring up the context menu of a tile (e.g. by right-clicking), it is possible to assign multiple tiles to the same template tile. The use of this feature is discussed later.
- **Current tile:** located in the bottom left, this pane shows the current template tile. It is only a visual aid and serves no other purpose. This pane is not shown when editing a patch brush.
- **Assigned tiles:** located in between the **Current tile** and **Tile properties** panes, this pane shows all the tiles assigned to a template tile. Selecting a tile from the list will update the **Tile properties** panes with the appropriate values. From this pane it is possible to remove an assigned tile using a context menu option.
- **Tile properties:** located in the bottom right, this pane shows the properties of the currently selected tile in the **Assigned tiles** pane. Tile properties are the following:
  - **Probability:** in the case there is more than one tile assigned to a template position, the auto tiling and patch tools will pick one of the available tiles at random. This value dictates the probability of a tile being picked over the others.

### Toolbar

From left to right these are all the tools in the main toolbar.




- **Brush**: dropdown to select a brush to edit from the existing ones.
- **Options**: the button next to the brush dropdown, has three basic options to manage brushes: **Add**, **Rename** and **Delete**.
- **Type**: change the type of the current brush. Currently two types are supported: **Auto 16**, **Auto 47** and **Patch**. Changing this setting will update the contents of the **Template** pane.
- **Width**: The width in tiles of the patch brush being edited. This button is only present when editing patch brushes.
- **Height**: The height in tiles of the patch brush being edited. This button is only present when editing patch brushes.
- **Undo**: reverts the last change.
- **Redo**: applies the last change.
- **Tiles toggle**: this is a visual aid to quickly see the first tile assigned to each template tile. If a template tile has nothing assigned the toggle will make no changes. This button is only present when editing auto tiling brushes.
- **Background toggle**: change the background color of the **Template** and **Tileset** panes between a light and dark color.
- **Zoom in**: zoom in the **Template** and **Tileset** panes.
- **Zoom out**: zoom out the **Template** and **Tileset** panes.
- **Zoom reset**: reset the zoom of the **Template** and **Tileset** panes.

### Context Menus

- **Template pane**
  - **Zoom in**
  - **Zoom out**
  - **Zoom reset**
  - **Help**
- **Tileset pane**
  - **Add tile(s)**: Add additional tiles to be used in a template position in addition to the first one. Multiple tiles might be added and will be visible in the **Assigned tiles** pane
  - **Set brush with selection**: Set all the tiles in the template at once with the current selection. This option clears any previous existing tiles in the template.
  - **Zoom in**
  - **Zoom out**

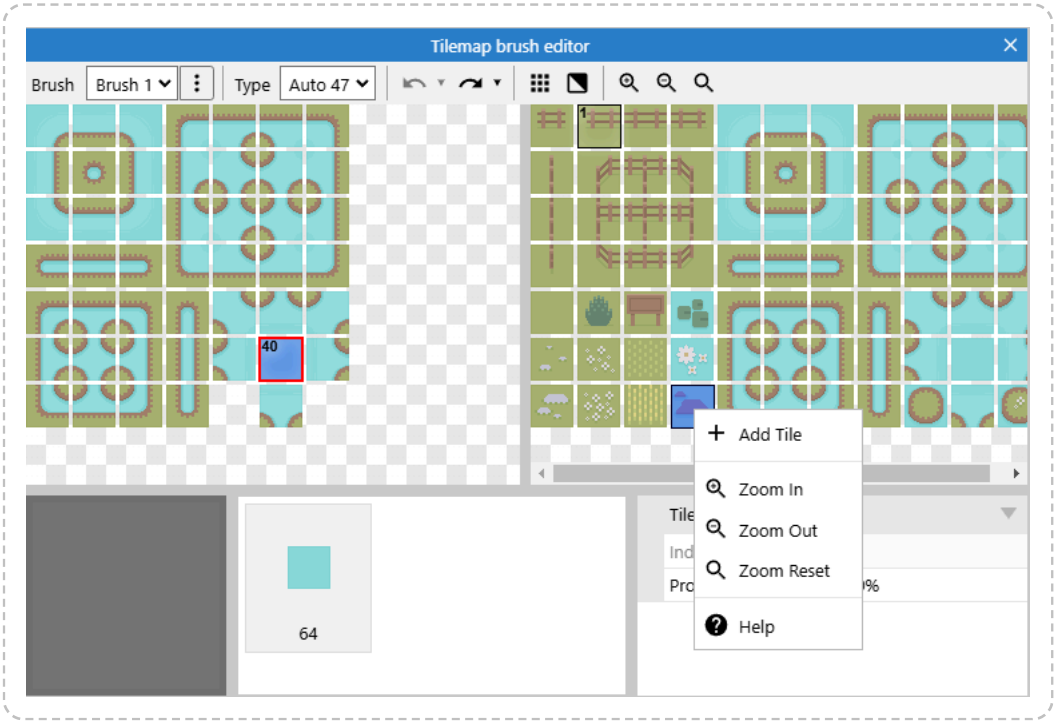
- **Zoom reset**
- **Help**

- **Assigned tiles pane**
  - **Remove Tile**: Remove the selected tile from the list of tiles available to the corresponding template tile.
  - **Help**

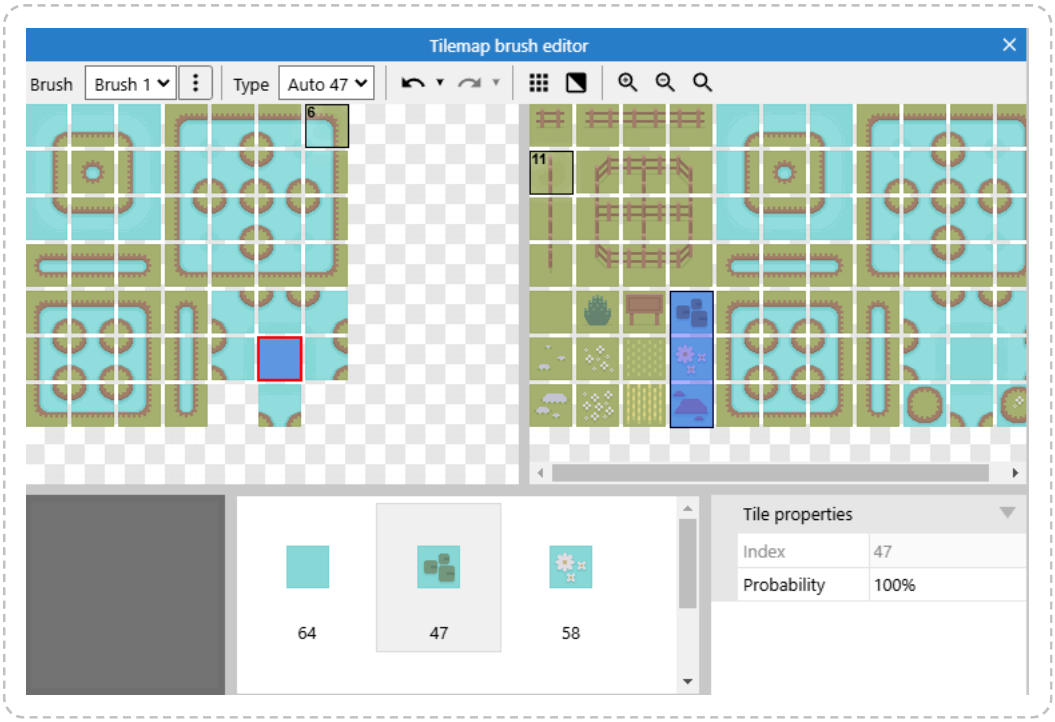
## Using multiple tiles on a template position

As mentioned earlier, it is possible to assign multiple tiles to a template position. The result of this is that for the positions where multiple tiles have been chosen, the auto tiling and patch tools will pick one of them at random when needed. This is useful to create more varied tilemap layouts without having to manually edit tiles one at a time.

The following image shows how would you add an additional tile to the selected template position.




Next you can see how the editor would look like after adding multiple tiles to the same template position.



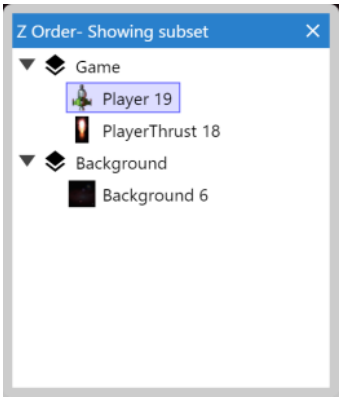

Using the **Probability** property of each tile, found in the **Tile properties** pane, it is possible to adjust which of the tiles is more or less likely to be chosen by the auto tiling and patch tools.

Keep in mind that all tiles with the same probability value will have the same chance of being chosen. For example, two tiles with 100% probability results in each tile having 50% chance of being chosen when using the auto tiling or patch tools.

# THE Z ORDER BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/z-order-bar

Paid plans only The **Z Order Bar** allows precise control over which objects appear in front of others. Although Construct is a 2D engine, the term *Z order* is used to refer to the display order of each individual object.




*The Z Order Bar*

To open the Z Order Bar, either right click in the Layout View and select Z Order▶Open Z Order bar..., or tick the check at Menu▶View▶Bars▶Z Order.

## The Z Order list

Instances are listed in the Z Order bar in front-to-back order, i.e. instances at the **top** of the list appear at the **front**, and instances at the **bottom** of the list appear at the **back**. Instances are grouped by the layer they belong to.

If no objects are selected, the Z Order Bar displays all instances in the layout. If some instances are selected in the Layout View, the list is filtered down to only those instances *and* any other instances overlapping them. This makes it convenient to see the relative Z order of a small area without having to take in to account the rest of the layout.

## Identifying instances

With lots of the same instances in the list, it can sometimes be difficult to tell precisely where a particular instance occurs in the list. To help identify each instance, its UID (unique identifier) appears after its name, e.g. *Player 41* (meaning a Player instance with UID 41).

Instances in the list which were selected in the Layout View are also selected in the Z Order bar. Selecting instances from the Z Order Bar itself will also select objects in the Layout View and show their properties, but will not affect the filtering of the list.

Sprite objects which have a different initial image set by changing the *Initial frame* or *Initial animation* properties also display an icon for that initial image in the Z Order list.

Instances can be **double-clicked** to make them flash briefly in the Layout View. Alternatively an instance can be right-clicked and then Flash this instance selected. This helps visually identify the instance in the layout.

The context menu also has the option Scroll selection into view, which will make the Layout View change it's scrolling to show all the selected instances.

## Editing the Z Order

To move an object in the Z Order - adjusting which other objects it appears in front or behind - **drag and drop it in the list**. You can also move objects to other layers this way. You can also select multiple objects by holding **Control** or **Shift** and drag them all as a block to another layer or location in the Z Order. When doing this, the relative order of the selection is also preserved.

If you want to add, remove or reorder layers themselves, use the Layers Bar instead.

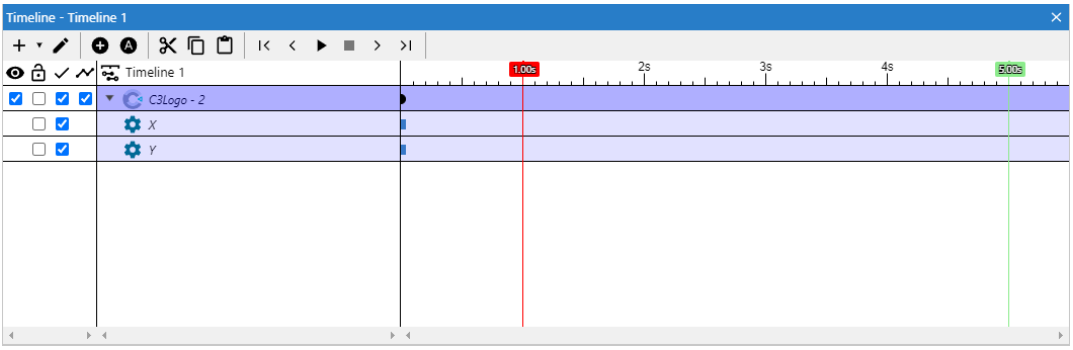
## View options

You can right-click the Z Order Bar and select Show active layer only. This further filters down the list to only display objects on the current active layer (the selected layer in the Layers Bar), which can be useful when working with a single layer.

# THE TIMELINE BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/timeline-bar

The **Timeline Bar** shows the currently active timeline. Through this control you can add and remove instances and other types of tracks to a timeline, and edit its properties.




## Using the Timeline Bar

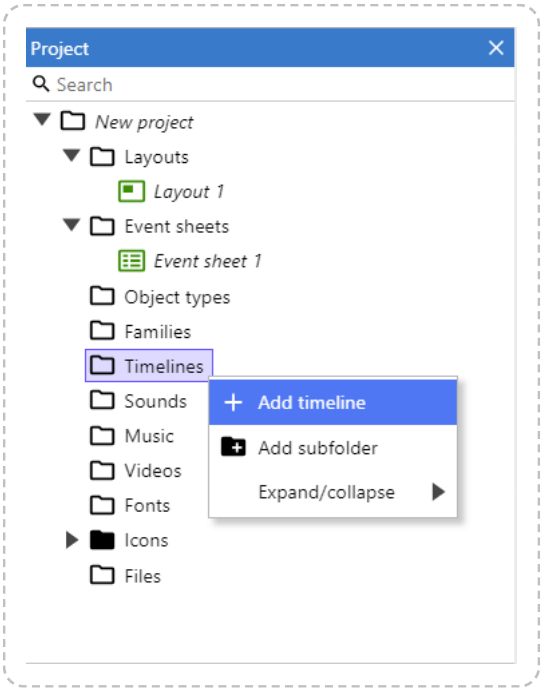
Here's a summary of how to use the Timeline Bar. A project can have multiple timelines, each timeline can have multiple instances, and each instance can have multiple property tracks. Tracks then use keyframes to mark points in a timeline. Timelines can also have nested timelines to produce more complex structures. Other types of tracks that do not directly reference instances are supported, those are not covered in this quick start guide.

### Step 1: create a timeline

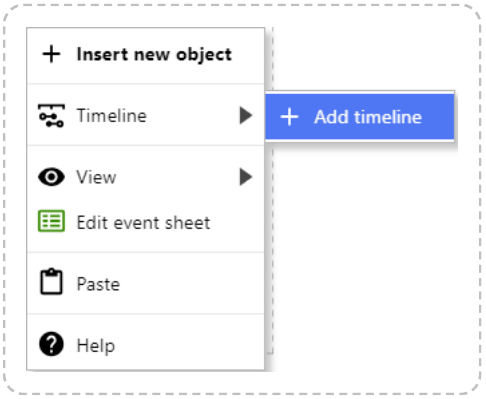
With the Timeline Bar open, you can create a timeline with any of the following methods:

- Right-click **Timelines** folder in the Project Bar and select *Add timeline*
- Right-click in the Layout View and select Timeline▶Add timeline

Project Bar context menu




Layout View context menu




## Step 2: add an instance

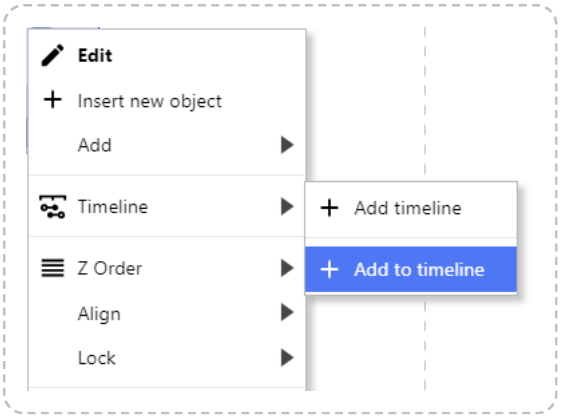
Once you have a timeline, add an instance to it with any of these methods:

- Use the add + button in the toolbar to bring up a dialog from which to choose instances to add
- Drag & drop instances from the Layout View into the bar
- Right click selected instances in the Layout View and select Timeline▶Add to timeline
- Use the Track▶Add instances option in the + split button

Add instances split button



Layout View context menu




## Step 3: add keyframes

Adding an instance adds property tracks for the position (X and Y co-ordinates). The next step is to add some keyframes, which you can do by following these steps:

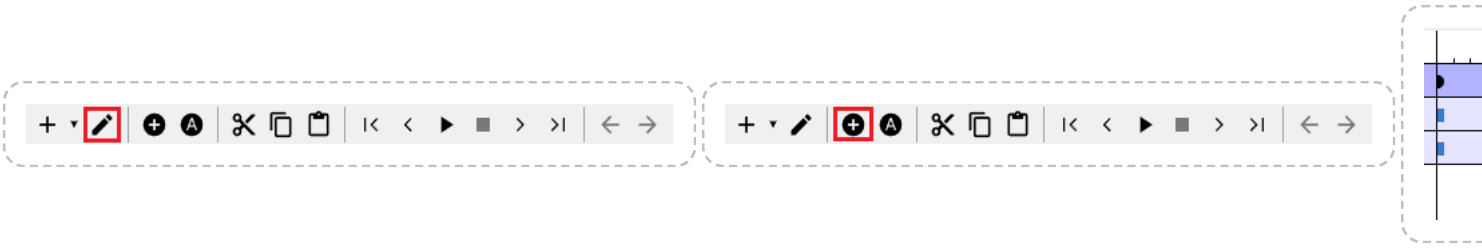
1. Turn on **Edit Mode** by pressing the **pencil** button in the toolbar

2. Move the current time marker to the position in the timeline where you want to create keyframes. This can be done by either clicking on the time ruler or by dragging the red line marker.
3. Make changes to the instances you want to animate
4. Use the **Set keyframes** toolbar button, the `S` keyboard shortcut or right click in the Layout View and select Timeline▶Set keyframes

Edit mode button

Set keyframes button

Current



Following those steps you should be able to setup the most basic timeline.

## Previewing a timeline

The timeline can be previewed in the editor by pressing the **Play** button in the toolbar or by scrubbing the current time marker (press and hold `Ctrl` or `Cmd` while scrubbing to move the marker without previewing the timeline).

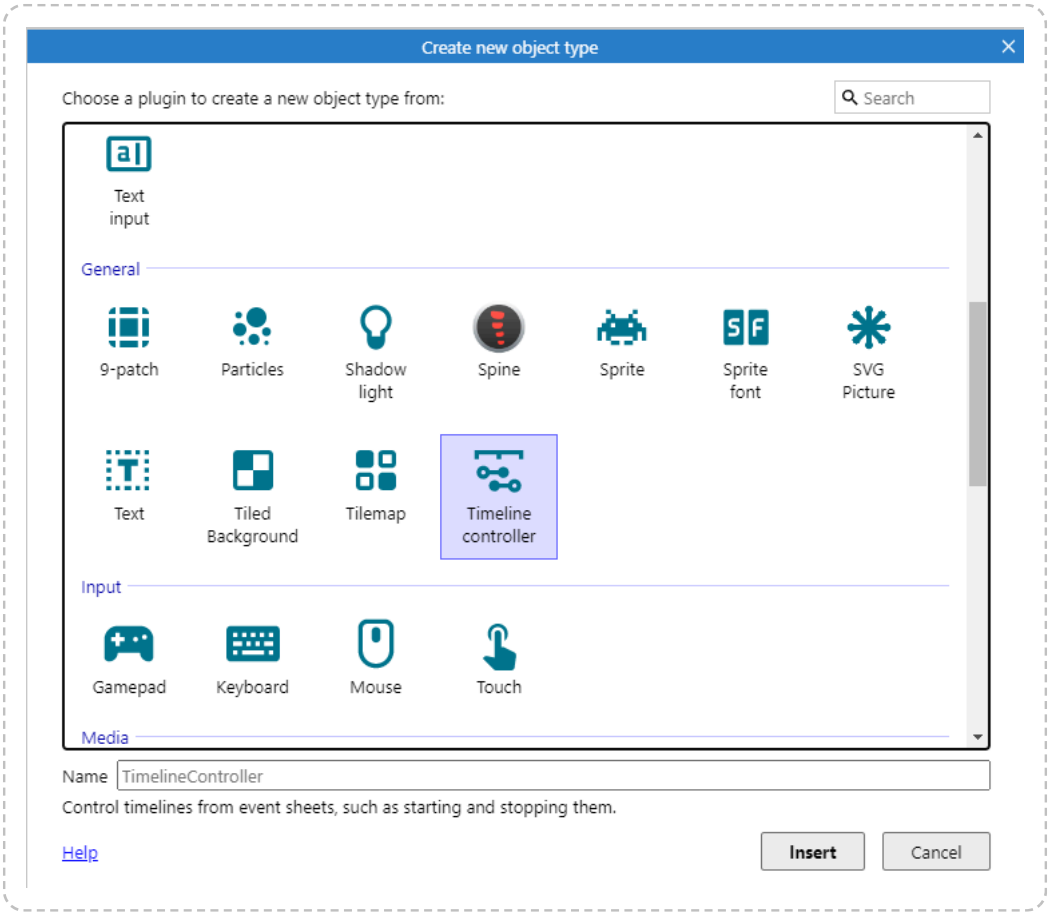
Playback Controls



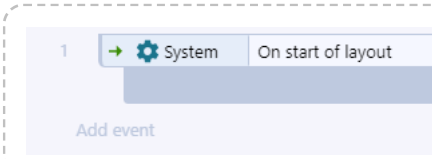
## Controlling timelines in events

You can control the timeline in your events using the Timeline Controller plugin.

Timeline Controller Plugin



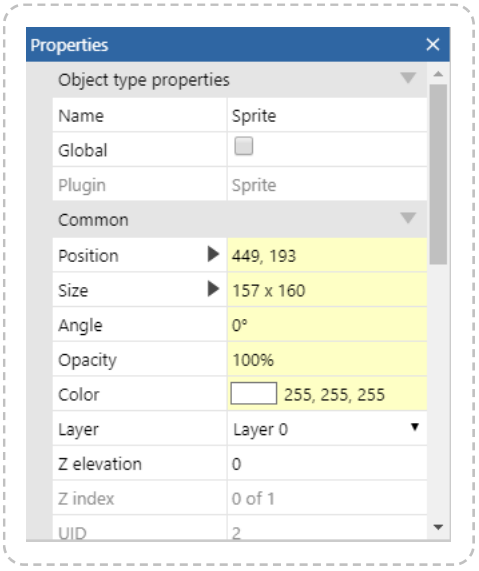

Timeline Events



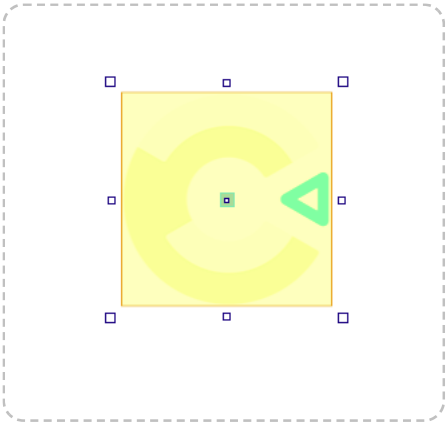

# Edit mode

When edit mode is turned on with the edit toolbar button, the selection boxes in the Layout View change color and the properties which can be animated with the timeline are highlighted in the Properties Bar. The highlighting only happens for instances which are part of the current timeline. Properties which are not highlighted in edit mode cannot be animated with a timeline. After finishing editing a timeline remember to turn the mode off as changes made in this mode are only relevant to the active timeline, rather than the whole project.

Highlighted Properties




Highlighted Selection



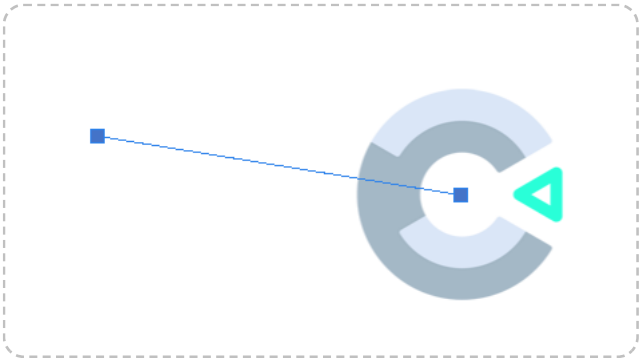
## Updating keyframes

To update keyframes, place the current time marker on top of the keyframes you want to update, make the necessary changes, and set the keyframe again, either by the toolbar button, keyboard shortcut or by the Layout View menu option.

Right-click master keyframes to update all the corresponding property keyframes with the current instance values.

Right-click property keyframes to individually update them with the current instance values.

Additionally, the X and Y properties of any instance can also be updated directly from the Layout View by dragging the handles for the keyframe.



## Drag & drop

There are various places in which it is possible to use a drag and drop workflow when using the Timeline Bar.

- To add new instances directly to the timeline from the current active layout. Drag the instances from the Layout View and drop them on a Timeline.
- To sort the different elements of the timeline that are represented by a row, such as a track.
- To add a nested timeline in the current timeline by dragging it from the Project bar.
- To update the starting offset of a nested timeline.
- To update the position of master keyframes and property keyframes. Notes on dragging keyframes:
  - Dragging a master keyframe updates it's position and the position of all related property keyframes.
  - Dragging a property keyframe by itself creates a new one at the new position, along with a corresponding master keyframe. Since property keyframes do not exist by themselves there is no notion of just moving them by themselves. The only way to move a property keyframe is to move the corresponding master.
  - Holding `Shift` while dragging a master keyframe will duplicate them and all related property keyframes in the new position.

## Auto keyframing



By toggling auto keyframes, keyframes will be added to the timeline at the position of the current time marker, as soon as changes are made in either the Properties bar or the Layout view.

## Note on animating Sprite plugin instances

It is possible to animate the current frame of a Sprite plugin instance by animating it's **initial frame** property.

When animating in this way make sure that the Sprite's normal animation is stopped as it would interfere with the changes made by a timeline. Likewise if you are just using a Sprite's regular animations, playing a timeline that changes the initial frame will cause un-expected results.

> *The two methods of animation can not coexist, you have to choose one over the other.*

It is also possible to change the current animation a Sprite plugin instance will show at runtime by animating the **initial animation** property.

## More information

For more details on a timeline's capabilities, properties and related objects see the Project Primitives section on timelines.

See the Timeline Controller plugin manual entry for more information on the actions, conditions and expressions available for controlling timelines.

The Timeline Bar has various keyboard shortcuts which are listed in the manual entry Keyboard shortcuts.

# INTERFACE

View online: https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/timeline-bar/interface

## Toolbar



The buttons are described from left to right.

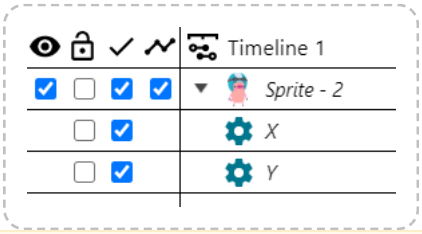
- **Add instances:** Brings up a dialog to add instances which are not already part of the current timeline.
  - Timeline▶add timelines option inside the **Add instances** split button. Brings up a dialog to add a nested timeline.
  - Timeline▶add subfolder option inside the **Add instances** split button. Will add a timeline folder to the root of the current timeline.
  - Track▶add instances option inside the **Add instances** split button. Does the same as the main button.
  - Track▶add value option inside the **Add instances** split button. Adds a value track to the root of the timeline.
  - Track▶add audio option inside the **Add instances** split button. Adds an audio track to the root of the timeline.
  - Track▶add subfolder option inside the **Add instances** split button. Will add a track folder to the root of the current timeline.
- **Edit mode:** Turns on timeline editing mode.
- **Set keyframes:** Set keyframes at the current time marker. Keyframes will be set in all tracks which have an instance with any meaningful change. Only works while in editing mode.
- **Auto keyframes:** Enable to set keyframes automatically in the current time marker as changes are made. Keyframes are only set in the corresponding tracks for the changes made. Only works while in editing mode.
- **Cut:** Cut the current selection of keyframes.
- **Copy:** Copy the current selection of keyframes.
- **Paste:** Paste the current selection of keyframes relative to the current time marker. If no tracks are selected each keyframe is pasted in it's respective track. If there is a selection of tracks, an attempt is made to copy as many keyframes as possible into the correct tracks. If a keyframe does not fit in any track, it is ignored.
- **Move to first keyframe:** Moves the current time marker to the first master keyframe

- **Move to previous keyframe:** Moves the current time marker to the nearest master keyframe moving backwards
- **Preview:** Start a preview of the current timeline.
- **Stop:** Stop the preview of the current timeline.
- **Move to next keyframe:** Moves the current time marker to the nearest master keyframe moving forwards.
- **Move to last keyframe:** Moves the current time marker to the last master keyframe
- **Previous timeline:** Go back to the previously focused timeline in a nested structure. Hidden if there are no nested timelines.
- **Next timeline:** Go to the next timeline in a nested structure. Hidden if there are no nested timelines.

## Toggles

These are the four options at the left most side of the bar. These affect not only the timeline element in the same row, but also any elements under the same hierarchy. If a row does not have a checkbox, the corresponding element does not support the property.

- **Visibility:** The **checkboxes** under the **eye** icon. These will toggle the visibility of the corresponding timeline track instance. The setting only takes effect while edit mode is turned on.
- **Lock:** The **checkboxes** under the **padlock** icon. These will toggle the lock state of a timeline element. Locked elements cannot be edited. Locked elements can't be edited and have a grey highlight when selected in the layout while timeline editing mode is turned on.
- **Enable:** The **checkboxes** under the **checkmark** icon. These will toggle the enable state of a timeline element. This setting affects the timeline at runtime. Disabled timeline elements are not taken into consideration when the timeline is interpolating values.
- **Show UI Elements:** The **checkboxes** under the **joined dots** icon. Using these toggles you can show and hide the layout UI elements associated with the instance. This includes paths lines, keyframe handles and cubic bezier handles. This can be useful when working with timelines which have many different elements.

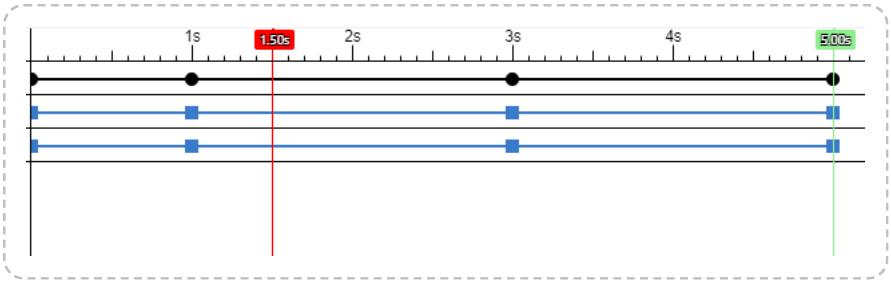



*The icons themselves can be clicked to modify the whole column.*

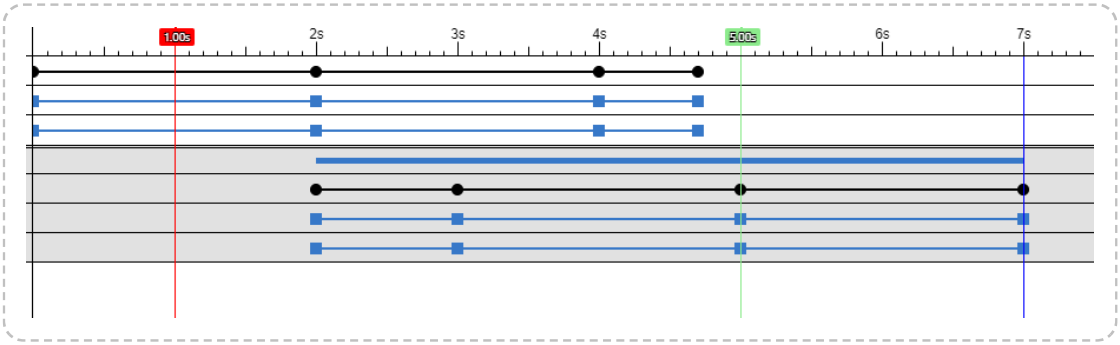
## Time markers

These are the three vertical lines in the right most section of the bar.

- **Current time:** The **red line**. Change its position to the place in the timeline you wish to add new keyframes. The marker can be dragged directly or by clicking on the time ruler and dragging. Dragging the marker will produce a preview of the timeline, provided that edit mode is turned on. Holding `Ctrl/Cmd` while dragging the marker to prevent the timeline from being previewed. Can also be changed from the Properties Bar when it is showing timeline properties.
- **Total time:** The **green line**. Indicates the total time the timeline takes to play to the end. Can be dragged directly like the current time marker and changed from the Properties Bar when it is showing timeline properties. This marker represents the total time of the top most timeline only, if there are nested timelines, those are not represented by this line.




- **Compound time:** The **blue line**. Indicates the total time a timeline has from start to finish including all of it's nested content. This marker can not be dragged and is just shown as reference. It won't be visible if there is no nested content or if the total time of the main timeline is enough to hold all of it's children.



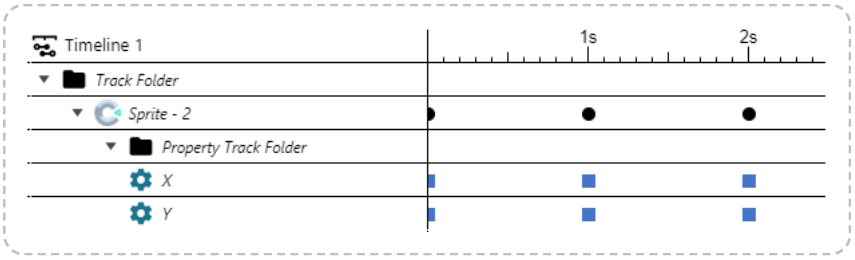

## Timeline elements

These are the parts that make up a timeline. When selected, these elements will show their properties in the Properties Bar.

- **Track:** Represented as a row with an icon of the corresponding instance. Tracks can be moved to and from track folders or the root of the timeline by dragging and dropping.
- **Master keyframe:** Represented by black dots in the same row as the track they are contained by. A master keyframe's main role is to serve as a control to modify corresponding property keyframes in bulk.
- **Property track:** Represented as a row with an icon of the corresponding property being affected by the track. They can be moved to and from property track folders or the root of

the parent track by dragging and dropping. They can not be moved outside of their parent track.

- **Property keyframe:** Represented by light blue square in the same row as the property track they are contained by.
- **Track folder:** Represented as a row with an icon of a folder. These are used to organise elements in a timeline, should it be needed. A track folder can have nested tracks as well as other track folders. They can be moved to and from track folders or the root of the timeline by dragging and dropping.
- **Property track folder:** Represented as a row with an icon of a folder. These are used to organise elements in a track, should it be needed. A property track folder can have nested property tracks as well as other property track folders, they can be moved to and from property track folders or the root of the corresponding track by dragging and dropping. Property track folders can not be moved outside their corresponding track.
- **Nested timeline:** Represented as a row with the icon of a timeline, these act like a folder for all the content inside and can be expanded and collapsed as such. Nested timelines can be moved to and from timeline folders or the root of the parent timeline by dragging and dropping. There are several details specific to nested timelines which are discussed later.
- **Timeline folder:** Represented as a row with an icon of a folder. These are used to organize nested timelines in a parent timeline. Timeline folders can only exist as children of the main timeline or nested inside other timeline folders.
- **Timeline offset handle:** Represented by a rectangle in the same row as the corresponding nested timeline. It's width represents the total playback time of the corresponding timeline and is positioned to be able to see, at a glance, when will the nested timeline will start and finish in relation to the parent. It can be dragged to adjust the starting time of the nested timeline in relation to the parent timeline.
- **Value Track:** Represented as a row with a name. Value tracks can be moved to and from track folders or the root of the timeline by dragging and dropping. They can only have one property track and are not associated with any instance. Must be used in tandem with the Timeline Controller plugin in order to query their value at runtime.
- **Audio Track:** Represented as a row with a name. Audio tracks can be moved to and from track folders or the root of the timeline by dragging and dropping. They can only have two property tracks, one representing the audio file and another optional one to change the volume as the timeline progresses. They are not associated with any instance.



# CONTEXT MENUS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/timeline-bar/context-menus

## Elements context menus

All the timeline elements have context menu options that will come up by right-clicking on them.

Right-clicking on any part of the Timeline Bar which does not reference any particular element, such as the section showing the name of the current timeline or the time ruler, will bring up a menu with options that affect the timeline itself, rather than any of its elements.

Here is a list with all the available options for each element. The more obvious ones such as **Delete** are not be described.

- **Timeline:**
  - **Timeline►Add timelines** Bring up a dialog to add nested timelines to the root of the current timeline.
  - **Timeline►Add subfolder** Add a timeline folder to the root of the current timeline.
  - **Track►Add instances** Bring up a dialog to add instances that don't already belong to the current timeline.
  - **Track►Add value** Add a value track to the root of the current timeline.
  - **Track►Add audio** Add an audio track to the root of the current timeline.
  - **Track►Add subfolder** Add a track folder at the root of the current timeline.
  - **Delete:** Only shown for nested timelines. Deletes the timeline from the parent, but not from the project.
  - **Focus:** Only shown for nested timelines. Gives focus to the timeline in the nested structure.
- **Track:**
  - **Add properties:** Bring up a dialog to manually add empty property tracks.
  - **Add subfolder:** Add a property track folder to the root of the track.
  - **Swap instance:** Brings up a dialog from which to choose instances that can be used to replace the existing one.
  - **Delete**
- **Master keyframe:**
  - **Update:** Update all the corresponding property keyframes.
  - **Disable:** Disable all the corresponding property keyframes.

    - **Enable:** Enable all the corresponding property keyframes.
    - **Delete**
    - **Add missing property keyframes:** Add any missing property keyframes under the specified master keyframe.
        - **With interpolated values:** The new property keyframes are given the values that the timeline would generate at that point.
        - **With current values:** The new property keyframes are given the values currently held by the instance.
- **Track folder:**
    - **Add instances from selection:** Add all the instances currently selected in the Layout View directly as children of the track folder.
    - **Add instances from dialog:** Bring up a dialog to add instances which are not already part of the timeline, directly as children of the track folder.
    - **Add subfolder**
    - **Rename**
    - **Delete**
- **Property track:**
    - **Convert to scale:** Convert width and height property tracks to corresponding scale X and scale Y property tracks.
    - **Convert all to scale:** Convert all width and height property tracks in a timeline to the corresponding scale X and scale Y property tracks.
    - **Convert to size:** Convert scale X and scale Y property tracks to corresponding width and height property tracks.
    - **Convert all to size:** Convert all scale X and scale Y property tracks in a timeline to the corresponding width and height property tracks.
    - **Delete**
- **Property keyframe:**
    - **Update:** Update the value of the property keyframe with whatever value the corresponding instance has at the moment.
    - **Disable:** Disable the property keyframe. A disabled property keyframe is not taken into account when playing the timeline.
    - **Enable:** Enable the property keyframe.
    - **Delete**
- **Property track folder:**
    - **Add properties:** Bring up a dialog to manually add empty property tracks for the corresponding track. The new property tracks are directly added as children of the property track folder.

    - **Add subfolder**
    - **Rename**
    - **Delete**
- **Timeline folder:**
    - **Add timelines:** Bring up a dialog to add a nested timelines to the timeline folder.
    - **Add subfolder**
    - **Rename**
    - **Delete**
- **Common options:**
    - **Cut:** Cut the current keyframe selection
    - **Copy:** Copy the current keyframe selection
    - **Paste:** Paste keyframes using the current time marker as reference. If no tracks are selected at the moment of pasting, the keyframes will be added in their respective tracks. If there are tracks selected at the moment of pasting, an attempt is made to paste the keyframes into the tracks they would fit best. If there are keyframes in the selection which can't be fit anywhere, they are ignored.

There is also the **Set keyframes** option that will show up in any menu provided there is at least a track, track folder, property track or property track folder selected. This will add keyframes at the position of the current time marker with the values the corresponding instances currently have. If there are already keyframes at the current time marker position, they will be updated with the most recent instance values.

## Timeline bar context menu

Aside from the above options, there are some common options which show up in all menus and are specific to the Timeline Bar itself.

- **Timeline:**
    - **Add timelines:** Bring up a dialog to add nested timelines to the root of the current timeline.
    - **Add subfolder:** Add a timeline folder to the root of the current timeline.
- **Track:**
    - **Add instances:** Bring up a dialog to add instances that don't already belong to the current timeline.
    - **Add value:** Add a value track to the root of the current timeline.
    - **Add audio:** Add an audio track to the root of the current timeline.
    - **Add subfolder:** Add a track folder at the root of the current timeline.
- **View:**
    - **Default:** Show the default view of the bar

  - **Animation modes:** Show the animation mode used by each element of the timeline
  - **Result modes:** Show the result mode used by each element of the timeline
  - **Eases:** Show the ease function that are currently in use in between each pair of keyframes
  - **Path modes:** Show which path mode is in use in between each pair of keyframes
- **Scale:** Change the zoom level of the bar. This is an editor only setting and will not affect the playback of a timeline.

The Properties Bar also shows most of the above options as properties. This allows you to see at a glance which settings are used all across the timeline, and offers a convenient way to change them as well.

# LAYOUT VIEW EDITING

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/timeline-bar/layout-view-editing

When edit mode is on, the Layout View will show handles indicating the path an instance will take as the timeline is played. The handles are only shown for **X** and **Y** properties, as those are the only that can really show any form of useful visual feedback. The handles can be used to update the path of the corresponding instance.

Additionally, when setting the **Path mode** property to **Bezier Curve**, more handles will appear to edit the path between each pair of keyframes as curves.

There are three types of handles that can be distinguished by color and size

- **Large Blue:** These represent the keyframe **X** and **Y** properties. Can be moved to update the corresponding keyframes. If there is no property keyframe for either the X or Y properties, the handle will only be able to move in one axis.
- **Small Green:** These show up when the **Path mode** property between a pair of property keyframes is set to **Cubic Bezier** and represent the first anchor point of a curve. If there is no property keyframe for either the **X** or **Y** properties, the handle will only be able to move in one axis.
- **Small Red:** These show up when the **Path mode** property between a pair of property keyframes is set to **Cubic Bezier** and represent the second anchor point of a curve. If there is no property keyframe for either the **X** or **Y** properties the handle, will only be able to move in one axis.

Aside from the handles, there are a few different lines that show up when the Layout View tool is turned on:

- **Blue:** These line represents the path instances will take as the timeline is played.
- **Grey:** These lines are just visual connection between keyframes to help identify their order in the timeline.
- **Red:** These appear to show which pair of keyframes an instance is currently between, and also to indicate that new keyframes can be added at that position.
- **Green:** These appear when the current time marker is on top of a master keyframe in the Timeline Bar. If the **Set Keyframes** option is used now, all the keyframes at that position will be updated.

## Show UI Elements Toggle

When creating a timeline with many different instances it can become confusing to edit their paths if all the UI elements mentioned above are visible for all the instances at the same time.

With this toggle you can hide the UI elements for the instances you are not interested in and only show the ones you want.

Further more, if you make a master keyframe selection in the Timeline bar, the layout will only show the corresponding UI elements for those specific master keyframes.

# NESTED TIMELINES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/timeline-bar/nested-timelines

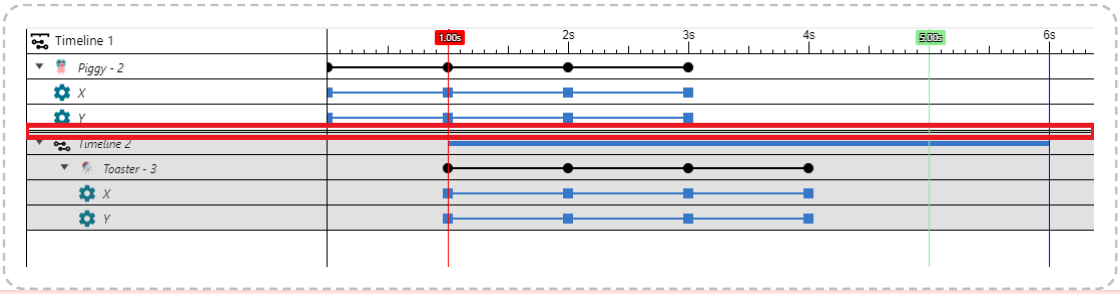
It is possible to nest timelines inside another parent timeline, this allows for coordination between timelines without the need of events. When a timeline is shown as nested inside a parent it works largely the same, some exclusive bits of UI are introduced as well as some differences when compared to the main timeline.

## Adding nested timelines

First you will need to create more than one timeline in the project, after doing that you can do any of the following:

- Drag a timeline from the Project Bar into the Timeline Bar to nest it in the currently active timeline.
- Use the sub option Timeline▶Add timelines of the + split button in the toolbar to bring up a dialog from which to choose timelines to add.
- Right-click on empty space in the Timeline Bar to bring up a context menu for the current timeline and use the option Timeline▶Add timelines.

The image below shows how a nested timeline shows up in the Timeline Bar, after the main timeline elements and separated by two horizontal lines (highlighted in red). You can also see the nested timeline rows have a grey background to indicate they are not the main focus of the bar. A few other UI elements unique to nested timelines appear as well, continue reading for more details.




*It is not possible to nest timelines that would produce recursive structures. Attempts to do so will just be ignored.*

## Editing nested timelines

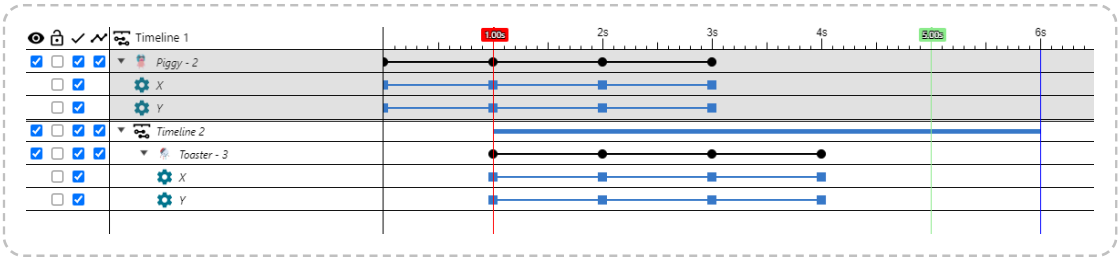
### Focus

While a timeline is shown as nested content most editing can be done on it as in the main timeline. This includes, editing through the Properties Bar, moving keyframes, deleting keyframes and changing the starting offset. Adding instances and adding keyframes is not directly available to nested timelines though. In order to do those actions the nested timeline must have focus.

Giving focus to a nested timeline is easy, you can do one of the following:

- Double-click the corresponding nested timeline row.
- Right-click the corresponding nested timeline row and use the focus option.

In the image below you can see the nested timeline having focus. Now all of it's content have a white background, while the rows that don't belong to it have a grey background. The timeline with focus is the only that will accept changes relating to adding instances, keyframes or updating keyframes.




### Toolbar arrow buttons

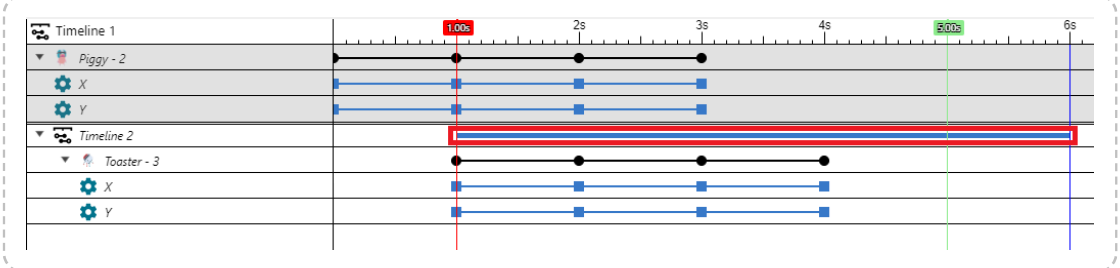
The left pointing arrow is used to go to the last timeline which had focus, the right pointing arrow is used to go to the next timeline which had focus. These buttons are only shown if there are nested timelines in the current main timeline, otherwise they are hidden.



### Starting Offset

The starting offset handle is the line that can be seen in the same row as the top most element of a nested timeline. It is positioned to show where the nested timeline will start playing in relation to the parent, and it is sized according to it's total time.

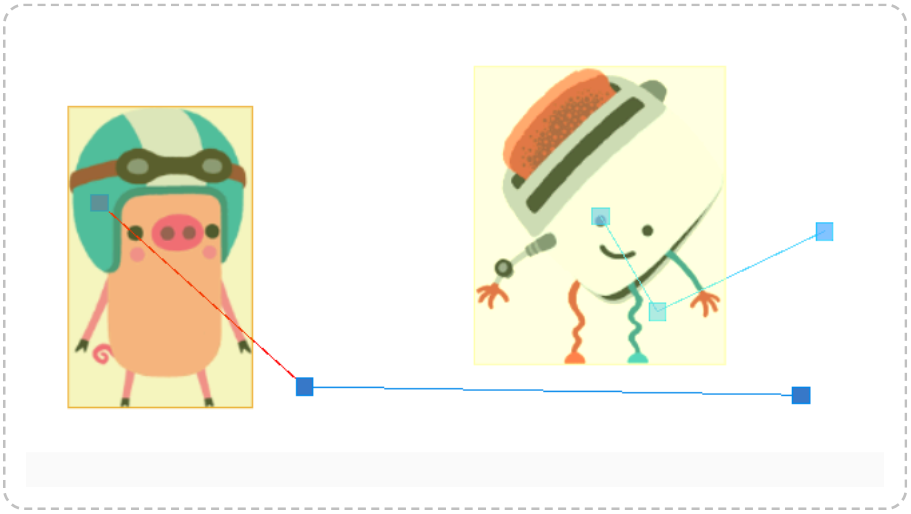
Similar to keyframes, dragging this handle allows you to change the starting offset of the nested timeline.




## Layout view feedback

When a timeline with nested content is previewed in the layout, everything works as expected, but the instances which don't have focus have slightly different feedback.

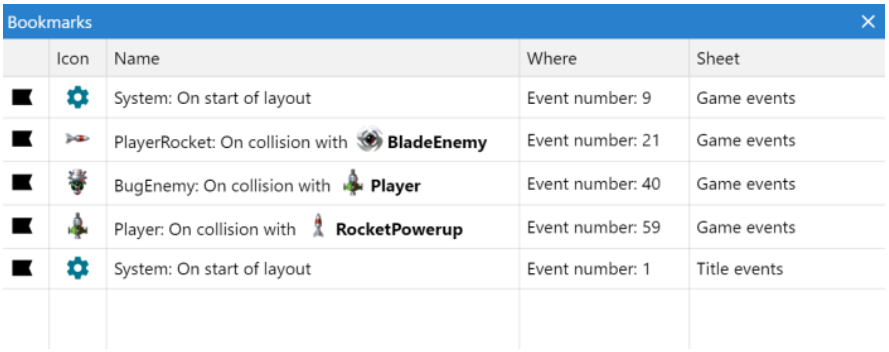
The instances and timeline UI elements which don't have focus, are shown with slightly transparent colors.



# THE BOOKMARKS BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/bookmarks-bar

Paid plans only The **Bookmarks Bar** shows an overview of all the bookmarks in the project. Bookmarks are a way of marking and quickly moving between events in the Event Sheet View.




*The Bookmarks Bar*

In the Event Sheet View, bookmarks can be toggled by pressing `F2` or right-clicking and selecting Toggle bookmark. Bookmarks can also be navigated between with Ctrl+F2 (for next bookmark) and Shift+F2 (for previous bookmark). Alternatively you can **double-click** the bookmark listed in the Bookmarks Bar. You can also **right-click** a bookmark listed in the Bookmarks bar to navigate to it, remove it, or remove all bookmarks in the project. The **Delete** key can also be used to remove bookmarks.

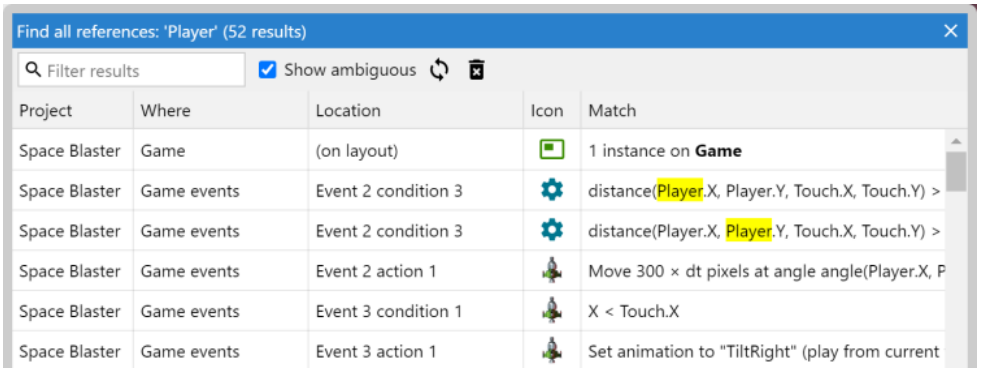
As with all bars, the Bookmarks Bar can be docked anywhere in the user interface or left floating as an individual window.

Bookmarks are listed in the order they occur in the project: first by event sheets in the order they appear in the Project Bar; then by their sequence within the event sheet. Each bookmark listed shows a description by it, if possible. For example a bookmarked group will show the title of the group, a bookmarked variable will show the name of the variable, and a bookmarked event will show some text from the first condition of the event. If the bookmark position has an event number, it is also shown, and the event sheet the bookmark belongs to is also listed.

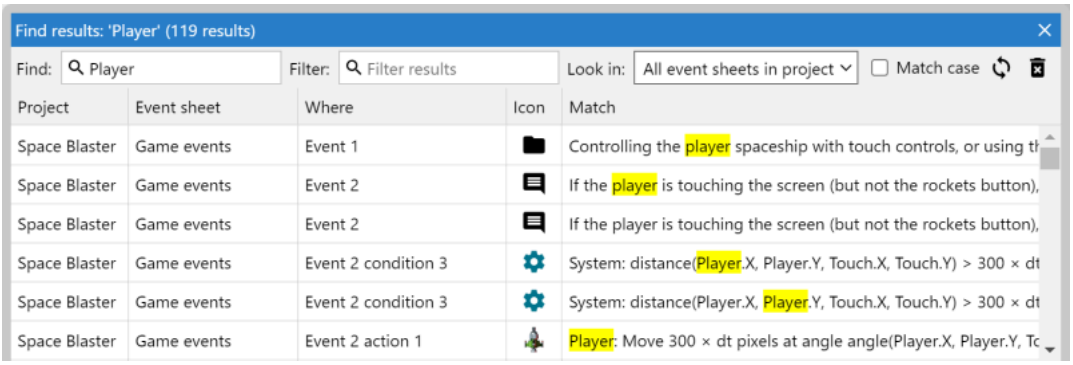
# THE FIND RESULTS BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/find-results-bar

Paid plans only The **Find Results Bar** is displayed either when you search for text in an event sheet (using **Ctrl + F** or Event sheet▶Find...), or when you use the **Find all references** feature (e.g. via the Project Bar to search for an object type's references). The results are listed with highlighting and information about their location, and they can be used to navigate to the result in the project.




*Find results for 'Find all references'*




*Find results for a search term*

There are various different kinds of find results. These can be text matches for text searches, event matches for *Find all references*, instances on a layout, family members, and more. Normally navigating to a result locates and selects the relevant event in the Event Sheet View or line of text in a project file. However other types display in different ways; for example navigating to a reference which indicates a number of instances on a layout will instead open the Layout View, select those instances, and adjust the scroll and zoom so all the instances are visible on-screen.

To navigate to a result, **double-click** on it, or right-click and select Go to. This allows you to review the result in its original context.

Results can also be removed via the right-click menu or the `Delete` key, or the results cleared entirely. You can also choose Redo search to update the results with the latest state of the project.

There are various other search options in the toolbar at the top of the bar. This includes extra options like a secondary search term to filter the results list by, and search location and case sensitivity options when searching by text.

Finding in projects, especially *Find all references*, is an excellent way to review large projects. It can also provide helpful reminders about what you've used and where. It's also a good way to check if something is unused and so can be safely deleted.

### Ambiguous results

In some cases, *Find all references* is unable to determine if a reference is a match or not. For example if you use the system *Create object by name* action to create an object with a name based on `TextInput.Text` the editor doesn't know what the value at runtime will actually be, because it can't predict what the user will type in to the text input. It's possible it could refer to the object being searched for, but it can't be certain.

To handle these cases, they are listed anyway so you can check them yourself. Ambiguous results appear with a question mark icon to indicate they are not certain results. You can also choose to hide all such results by unchecking *Show ambiguous* in the toolbar.

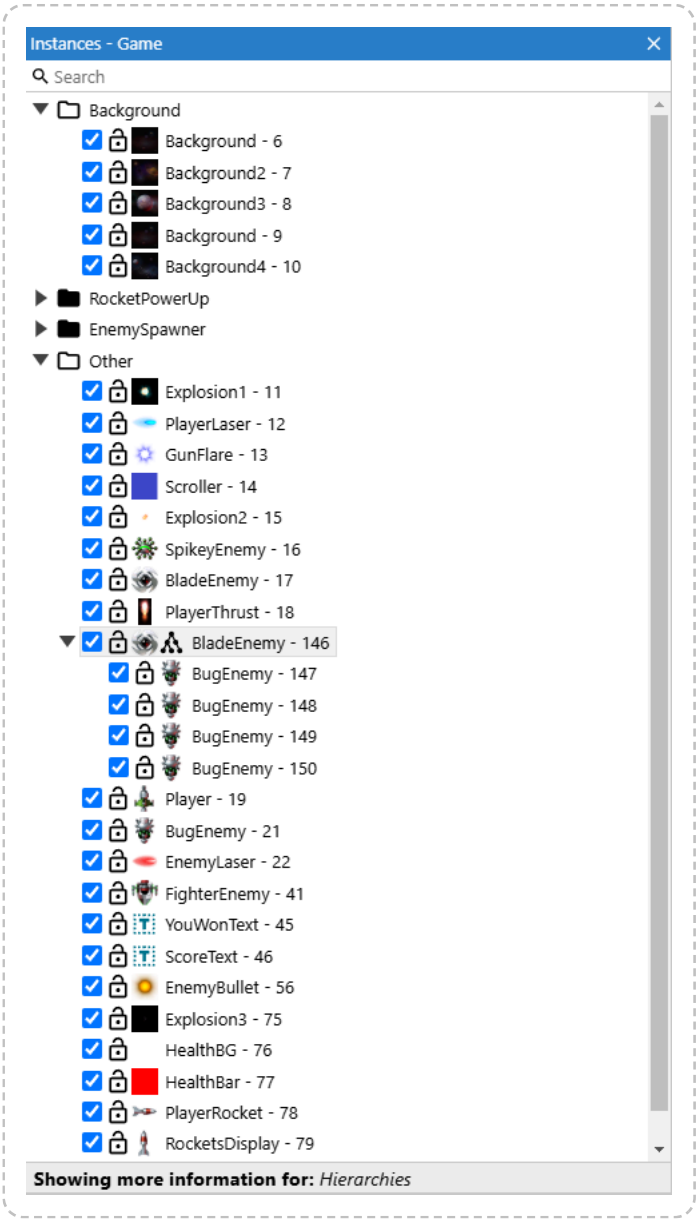
One reason to avoid heavy usage of dynamic features like *Create object by name* is to avoid clogging up the *Find all references* results with ambiguous results that can't be proven to refer to one object or another. The standard *Create object* action, on the other hand, uses an object picker instead of a string expression. This means the editor does know in advance what kind of object will be created, and thus can list it accurately when using *Find all references*.

# INSTANCES BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/bars/instance-bar

The **Instances Bar** shows all the instances in the current layout at a glance, it can be very useful in large or busy layouts to quickly find specific instances.

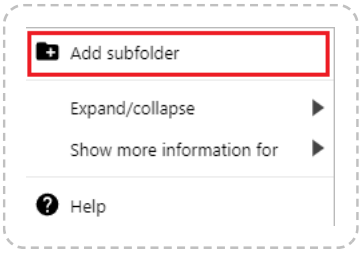
You can also use folders to group related instances together. Hierarchies are shown as a tree structure to quickly tell which instances are where in complex structures.




To open the Instances Bar tick the check at Menu▶View▶Bars▶Instances Bar.

## Folders

To add a folder right-click anywhere in the bar and choose the appropriate option from the context menu.




Once a folder is created you can drag & drop instances in and out of them by dragging the items in the bar.

It is also possible to drag instances into a folder directly from the Layout View.

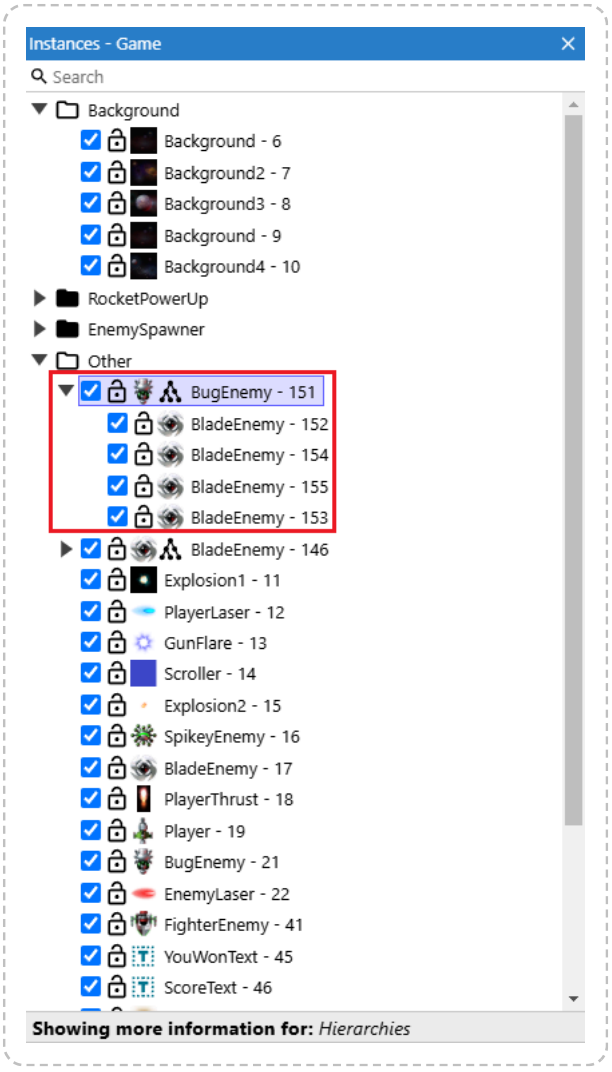
The order in which instances appear in the bar can also be altered by dragging and dropping.

> *The order of the instances and the folder structure shown by the Instances Bar are only relevant for visual organization in the editor, it doesn't matter at runtime.*

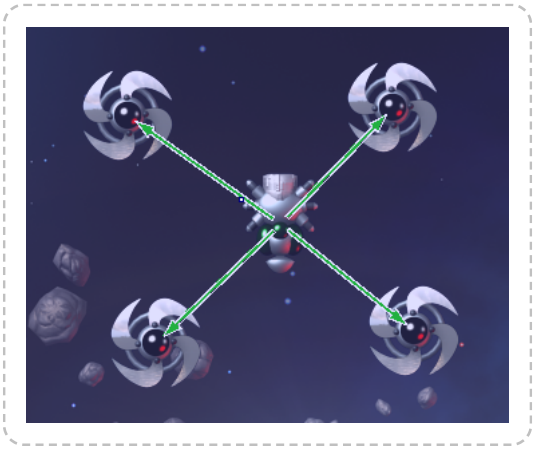
## Hierarchies

Hierarchies can be set directly in the bar by dragging and dropping items on top of each other. Dragging an instance from the Layout View on top of an item in the bar will also work for this purpose.

Hierarchy in the Instances bar



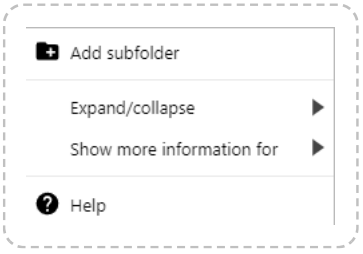

Hierarchy in the Layout View



# Context menu options

## Common

These options are shown with any context menu in the bar.



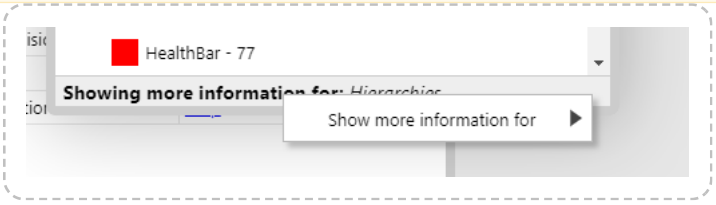

- **Add folder:** adds a new folder.
- **Expand / Collapse:** expand or collapse all items with nested content.
- **Show more information:** show additional information about each instance.
  - **Hierarchies:** show an icon next to instances which have children. In the case an instance has a parent in a different layout (which can happen when working with global layers), that layout will also be shown next to the item.

- **Meshes:** show an icon next to instances which have a mesh and also show the horizontal and vertical size of the mesh.
- **Layers & Layouts:** show an icon indicating if the layer is global and the name of the layer the instance belongs to. If the layer is not part of the current layout, also shows the layout the layer is coming from.

  *Clicking the layer name will make it the currently active layer.*

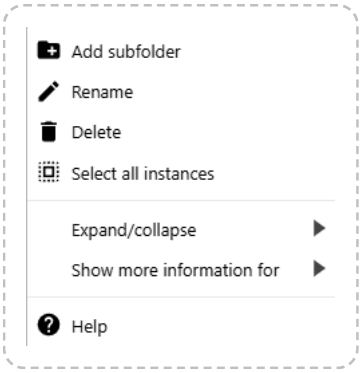
- **Plugin:** show the icon and the name of the plugin.
- **Templates:** template and replica instances show a unique icon as well as the corresponding template name.
- **Timelines:** show an icon next to instances belonging to a timeline as well as the names of the timelines the instance belongs too.

*Clicking the lower part of the bar also shows this menu.*




## Folder

These options are shown when showing a context menu for a folder.

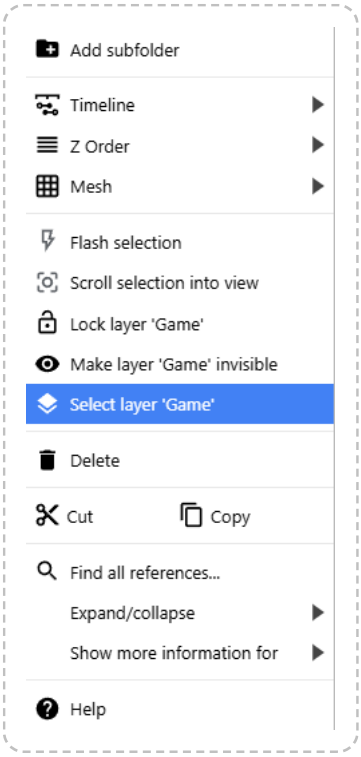



- **Rename:** rename the selected folder.
- **Delete:** delete the selected folders.
- **Select all instances:** select all the instances in the folder.

## Instances

These options are shown when showing a context menu for an instance or group of instances. Some are equivalent to the context menu options with the same names found on context menus

of the Layout View.




- **Timeline:** shown for all instances as long as there is a timeline in the project. Will have different options depending on how the instance relates to the current timeline. This is the same as the equivalent Layout View context menu option.
- **Tilemap:** only shown for Tilemap plugin instances. This is the same as the equivalent Layout View context menu option.
- **Z Order:** shown for all instances. This is the same as the equivalent Layout View context menu option.
- **Hierarchy:** shown for instances that are already part of a hierarchy or when selecting multiple instances. This is the same as the equivalent Layout View context menu option.
- **Template:** shown only for instances which are either a template or a replica. This is the same as the equivalent Layout View context menu option.
- **Mesh:** shown for all instances that support meshes. This is the same as the equivalent Layout View context menu option.
- **Flash selection:** will make the current selection blink in the Layout View.
- **Scroll selection into view:** scroll the Layout View to show the selected instances.

  > *Double clicking the item in the bar will also trigger this action.*

  > *The Layout View context menu has a related option **Find in Instances bar**.*

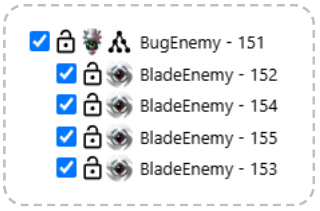
- **Lock / Unlock layer:** toggle the lock state of the layer the selected instance belongs to.

- **Toggle layer visibility:** toggle the visible state of the layer the selected instance belongs to.
- **Select layer:** select the layer which the selected instance belongs to. This is the same as selecting a layer through the Layers bar.
- **Delete:** delete the the selection of instances.
- **Cut, Copy and Paste:** these options do the same as the ones in the Layout view context menu.
- **Find all references:** searches the whole project for the selected instance.

## Icons

Depending on what **Show more information for** option is selected, additional icons might be shown next to an instance to better identify it.

The **layer visibility checkbox** and **layer lock icon** are shown at all times to change the corresponding visibility and locked states of each instance individually at any time.




## Locking and Visibility

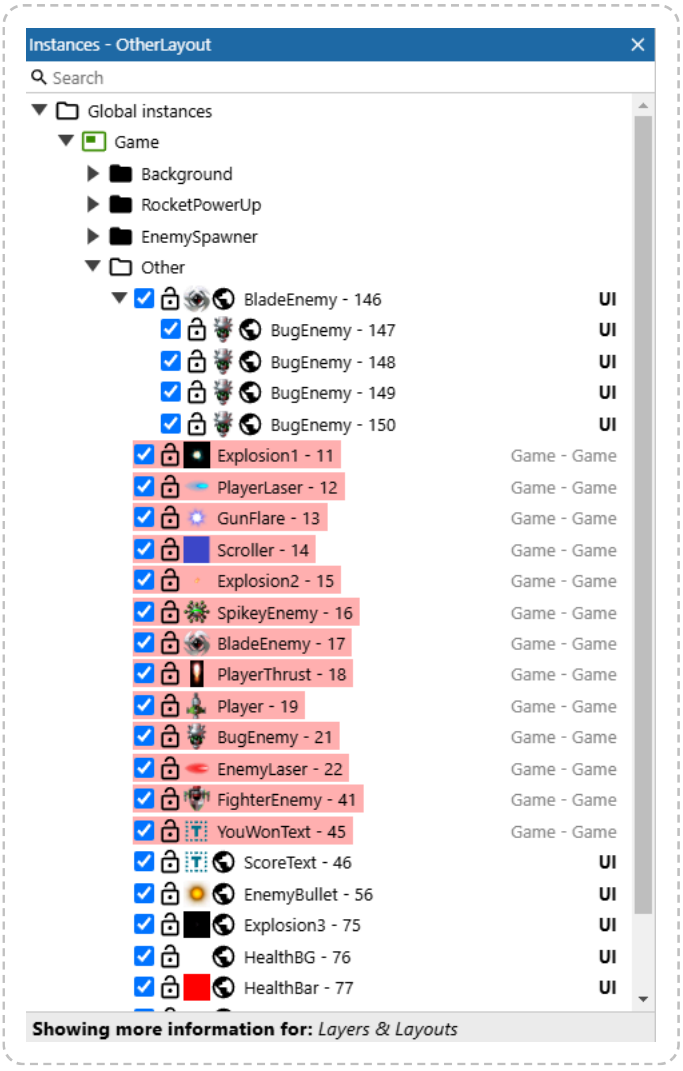
Locking instances through the Instances bar is similar to locking them using the Layout View context menu option **Lock selection**. This means this locked state is independent from the locked state of the corresponding layer.

Toggling visibility of an instance works in a similar way to the locked state in that it is independent to the visibility state of the corresponding layer.

## Global Layers

When a layout is showing instances that are coming from a global layer a few unique items will appear in the bar to make it more clear where those instances are coming from.

In the image below you can see the special **Global instances** folder and a **Layout** item showing which instances are coming from a global layer.

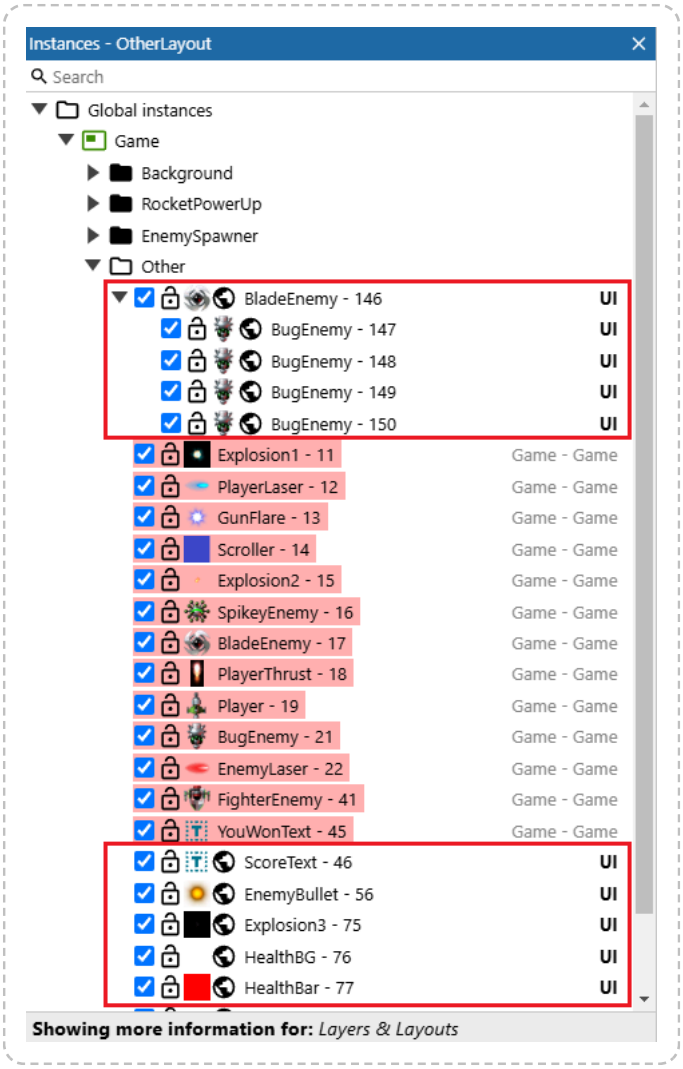



In that example the current layout is **OtherLayout** and the bar is showing that some instances shown are coming from a global layer in the **Game** layout. At the bottom you can see the **Layers & Layouts** option is toggled so all instances are showing which layer they belong too.

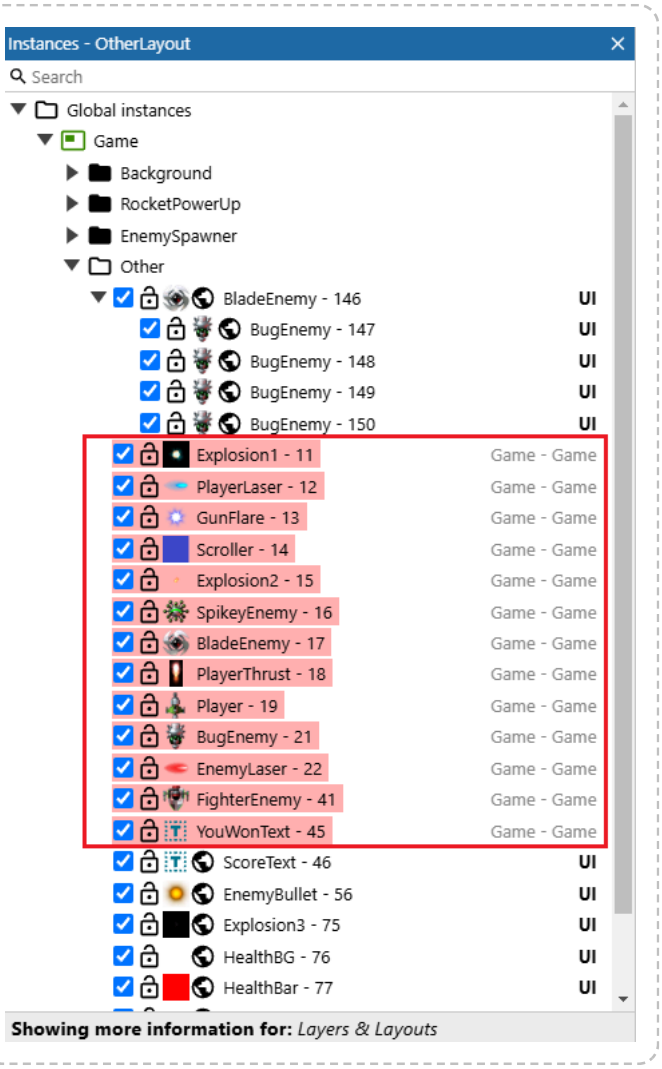
The globe icon next to some instances indicates they are part of a global layer. Instances which don't have a globe are not part of any global layer in the current layout.

Instances highlighted in red are shown in the bar because they belong to the **Game** layout, but are not in any global layer in the current layout.

Instances from a global layer




Instances not in any global layer




## Searching

Use the search field at the top of the bar to filter instances. Items are filtered using the following criteria and order:

1. Any match with the item name as shown in the bar.
2. Any match with the Object type name.
3. Any match with the plugin name.
4. Any match with any of the tags an instance might have.
5. Any match with the template name an instance might have.
6. Using the special **"mesh" (or it's translated form)** string will match all instances which have a mesh.
7. Using the special **"hierarchy" (or it's translated form)** string will match all instances which are part of a hierarchy.

## Drag & Drop

As mentioned earlier it is possible to drag & drop instances within the bar to arrange them in which ever way is more useful.

It is also possible to drag instances from the bar into the Layout View, which will create clones of the dragged instances in the dropped position.

Dragging instances from the Layout View into the Instances Bar has the same effect as dragging the corresponding items within the Instances bar.

# THE LAYOUT VIEW

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/layout-view

The **Layout View** is a visual designer for your objects. It allows you to set up a pre-arranged *layout* of objects, such as a game level, menu or title screen. In other tools, *layouts* may be referred to as *scenes*, *rooms*, *frames* or *stages*. See also the manual section on layouts.




*The Layout View*

The dashed rectangle in the top left of the layout area indicates the **viewport size** in the layout. By default the viewport appears in the top left of the layout, so to align something relative to the viewport, it should be placed inside this rectangle.

In the corner of the view appears a small **status bar** with information about the current **mouse position** in the layout, the current **zoom level**, and the current **active layer**. The active layer is important since it is **the layer new object instances are added to.** The active layer can be changed by selecting a different layer in the Layers Bar.

## Adding, modifying and deleting objects

**Double-click a space** in the layout or right-click and select Insert new object to add a new object type. This will bring up the Create New Object Type dialog.

To create new instances of an existing object type, another object can be **control + dragged**, copy and pasted, or dragged and dropped from the Project Bar. (Make sure you're clear on the difference between Object Types and Instances as described in Project Structure.) When dragging and dropping from the Project Bar, you can also use templates to define the default properties of the instance that is created.

A shortcut for importing image files as Sprite objects is to **drag and drop image files** in to the Layout View. This automatically creates a new Sprite object type with the dragged image. If multiple image files are dragged, the Sprite is assigned an animation with the dragged images as animation frames. Where supported, animated image file formats like GIF and APNG can also be dragged and dropped in and will be used as a Sprite animation. (Animated image file formats can also be imported to the Animations Editor where they will also be split out in to separate frames.)

> *Chrome and Edge support importing animated image file formats this way, but other browsers may not support it, in which case they will only be able to use the first frame.*

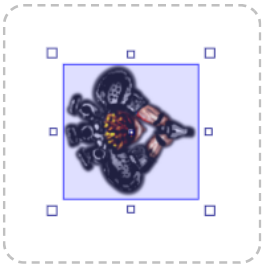
SVG files can also be drag-and-dropped in and a SVG Picture object will be created for it.

Instances can be moved by dragging and dropping them with the mouse. **Hold `Shift`** to axis-lock the drag to diagonals. Alternatively they can be nudged 1 pixel at a time with the arrow keys (hold shift to nudge 10 pixels), or co-ordinates can be typed in directly to the Properties Bar.

The **Delete** key or right-click Delete option deletes instances. Deleting all instances of an object does **not** remove the object type from the project. To entirely remove an object from the project it should be deleted via the Project Bar.

Click objects to select them. Objects cannot be selected if their layer is locked. **Hold `Control`** while clicking to select multiple objects, or **click and drag** a selection rectangle to select all objects in an area. The Properties Bar displays properties for *all* currently selected objects, so changing a property sets it for every selected object.

When a single object is selected it appears with **resize handles** around it.



**Click and drag** the resize handles to stretch the object. **Hold `Shift`** to proportionally resize the object. Hold control to resize relative to the object origin, which appears as a small dot on the selected object.

Rotatable objects like Sprite can be rotated by **moving the mouse just outside the resize handles**, away from the object. When you do this the mouse cursor will **change to a rotation arrow**. When you see this, click and drag to rotate the object.

Sometimes the resize handles, or rotate cursor, can get in the way of other objects. If this happens, **hold `Alt`** to temporarily hide the resize handles and disable rotation. This allows you to select another object instead of modify the selected object.

## Using tilemaps

If you are designing a tile-based game, you can insert the Tilemap object and edit tiles in the Layout View. To find out more, see the manual entry on the Tilemap Bar.

## Scrolling and zooming

There are a few ways to scroll in the Layout View:

- **The vertical and horizontal scrollbars** at the edges of the view
- **Scroll the mouse wheel** to scroll vertically. You can also hold `Shift` to scroll horizontally.
- **Hold the middle mouse button** and drag the mouse
- **Hold** `Space` and move the mouse (useful for laptops with track pads)

On desktop systems, middle-mouse dragging is probably the most convenient way to move around the layout.

Zooming is useful to focus on a small area or see an overview of the entire layout. There are several ways to zoom:

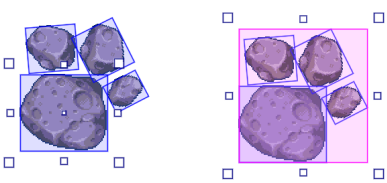
- The Zoom options in the **View** menu when right-clicking in the Layout View
- **Hold** `Control` and **scroll the mouse wheel**. Hold both `Control` + `Shift` to double or halve the zoom (e.g. 100%, 200%, 400%...)
- **Ctrl and + or -** on the keyboard. Hold `Shift` to double or halve the zoom.

Press `Control` + `0` to return to 100% zoom.

## Selection wrapping

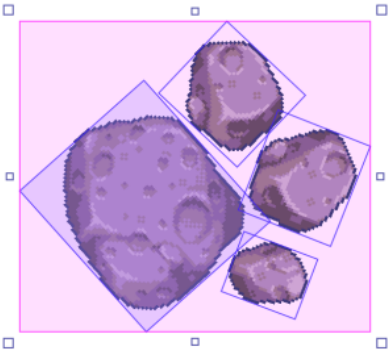
If you select two or more objects, you can wrap the selection by pressing `Enter` or right-clicking and selecting Wrap selection. This allows you to rotate and stretch the selection as a whole.

Wrapped selections appear with a different color selection box, as shown below:



*Wrapping a selection in the Layout View*

Wrapped selections can be resized and rotated as if they are one large object. For example the selection can be enlarged and rotated, and all objects maintain their position relative to each other.



*Wrapped selections resize and rotate as one*

While a selection is wrapped, click any of the objects in the wrapped selection to make that object the rotation origin.

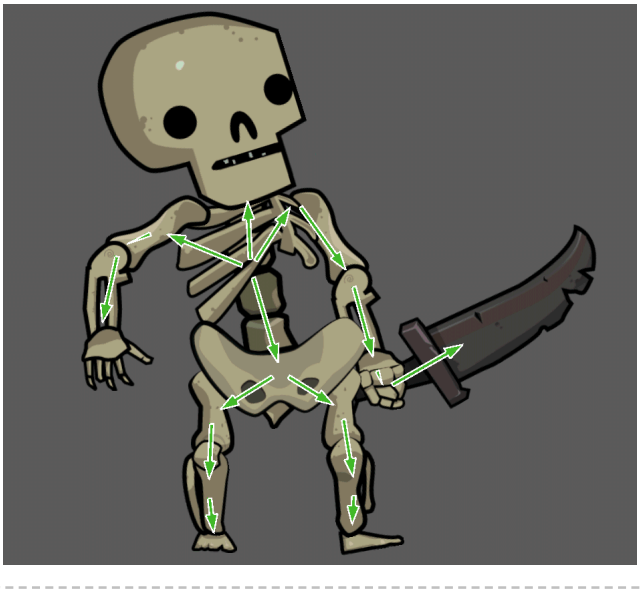
### Containers

Objects that are grouped in to Containers highlight yellow in the Layout View. Containers can also be set to automatically wrap their selection. If you still need to select an individual object in an automatically wrapped selection, **hold `Alt`** and click one of the objects.

## Setting up a hierarchy

You can connect objects together in a hierarchy - also known as a *scene graph* - in the Layout View. This works similarly to using the Add child hierarchy action, but set up in the editor.

To set up a hierarchy, select multiple objects, and then right-click the object you wish to be the parent (i.e. above the others in the hierarchy) and select Hierarchy▶Add selection to this instance. Arrows will appear pointing from the parent to the children to indicate the hierarchy.



When the children are selected, a new hierarchy section appears in the Properties Bar allowing you to choose which properties the child transforms with, such as the position, angle, and whether the child destroys with the parent. At runtime the child will follow any changes to the

parent (for the enabled properties) - often giving a visual appearance that the objects are connected, or form a single larger object.

Once a hierarchy is set up, other options also appear to detach objects: *Remove from parent* to remove a child from its parent, and *Remove all children* to detach the entire hierarchy below that object.

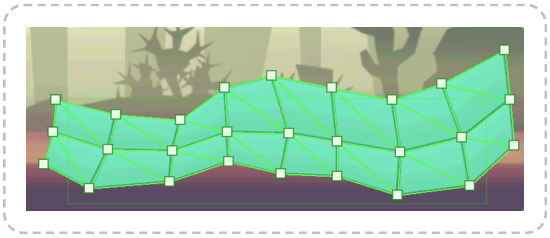
The root instance also has a *Select mode* property that, similar to containers, allows automatically selecting the entire hierarchy, also optionally with selection wrapping. If this is enabled, you can always hold **Alt** and click an instance to select it individually regardless of the select mode. The layout's editor properties also include a *Show hierarchy* option that lets you toggle whether or not arrows indicating the hierarchy, pointing from parents to children, are shown on top of objects.

The layout view will preview hierarchies both when previewing timelines and when previewing behaviors that support previewing.

## Editing meshes

You can create meshes for certain kinds of objects in the Layout View, as the editor counterpart to the mesh distortion feature. This lets you do things like create fluid level designs as shown in the Mesh platforms example. Meshes also affect collisions, so behaviors like Platform interact with them as they appear.

To create a mesh, right-click an instance and choose Mesh▶Create mesh.... You must specify a mesh size of at least 2x2. Once created, the mesh appears highlighted in green, with new green handles that you can click and drag to adjust the mesh. The mesh starts in a simple grid that does not alter the appearance of the object - once you move a mesh point it will start to change from its default appearance.



You can also **hold Shift** while dragging a mesh point to move it without distorting the image. This can create a kind of mask or cut-out appearance.

When you click a mesh point, it will also appear selected and display properties for that mesh point in the Properties Bar. This allows precise control over the exact details of the mesh point, as well as providing informational values such as the mesh column and row. The **Z elevation** of the mesh point can also be modified in the Properties Bar, allowing for 3D mesh distortion. To learn more, see the tutorial Using 3D features in Construct.

Once an object has a mesh you can access some new options in the *Mesh* sub-menu:

- **Set mesh size**: change the number of columns and rows in the mesh. Note this will also reset the mesh back to its default grid.
- **Reset mesh**: resets the mesh back to its default grid, which does not alter the appearance of the object.
- **Stop editing mesh**: removes the green handles so the mesh can no longer be edited, restoring the default selection for standard move and resize interactions with the object. Once selected you can use the **Edit mesh** option to go back to editing the mesh.
- **Remove mesh**: removes the mesh entirely, reverting the object to not using a mesh.

The layout's editor properties also include a *Show meshes* option that lets you toggle whether or not the green mesh outline is shown on top of objects with meshes.

## Other

To go to the associated event sheet, press `Ctrl` + `E` or right-click and select Edit event sheet.

The Z order of objects within a layer can be adjusted by right-clicking and selecting Z Order▶Send to top of layer or Z Order▶Send to bottom of layer. You can also open the Z Order Bar Paid plans only for advanced control.

Objects can be snapped to a grid for tile placement, and the collision polygons of the displayed objects can also be outlined. These features can be enabled in the layout's properties.

The right-click menu in the layout view also provides some alignment tools under the Align sub-menu. These allow you to quickly space objects equally or align objects along their edges. When aligning, the objects are aligned to the particular object you right-clicked.

The Animations editor can be brought up by **double-clicking** objects with images or animations like Tiled Background and Sprite. You can also double-click Text objects to edit their initial text in a dialog.

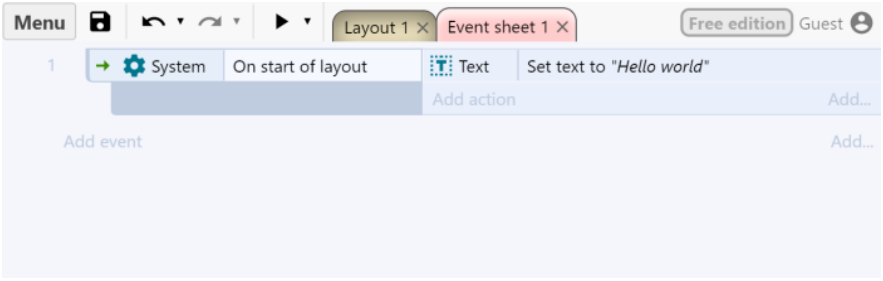
Effects will be displayed in the layout view if enabled in project properties.

# THE EVENT SHEET VIEW

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/event-sheet-view

The **Event Sheet View** is where events can be added, viewed and edited in an event sheet using the event system - Construct's alternative to traditional programming.

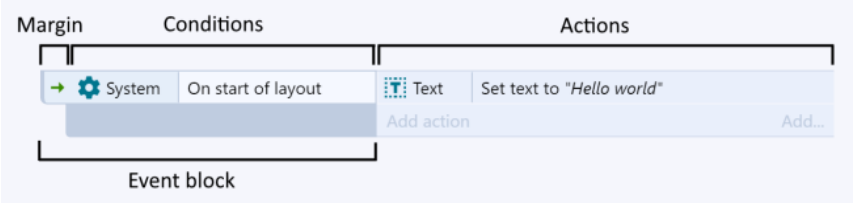
The event system has a lot of features, so the event system has its own section of the manual. This section will simply cover the basics of using the Event Sheet View.




*The Event Sheet View*

## Diagram of an event

The following image illustrates the key parts of an event.




Events are made up of three major sections:

1. The **event block**, which contains the conditions. Notice the margin to the left of the condition which allows you to select the entire event.
2. The **conditions**, which are each listed inside the event block.
3. The **actions**, which are listed to the right of the event block.

Conditions and actions can be selected by clicking on them. The entire event can be selected (which also selects all its conditions and actions) by clicking the **event margin**, or the bottom part of the event block. The event margin can also be right-clicked to access a menu allowing things like adding conditions or sub-events.

As with the Layout View, multiple selections can be made by holding `Control` and clicking different items. However, you can only have either events, conditions or actions selected at a

time (e.g. you can't have both a condition and action selected at once). You can also hold `Shift` and click an event, condition or action to select all the items in a line between the selection and clicked item.

## Creating events

There are a number of ways to add a new event:

- **Double-click** a space in the event sheet, or **right-click** in a space to see a menu of things to add
- Click the **Add event** link which comes after the last event in a sheet or group, or click the **Add...** link on the right
- **Right-click an event's margin** and choose an item from the Add menu

When you add a new event, the dialog that appears is for adding the first **condition** (see the Add Condition dialog). To add more conditions to an event, right-click the margin or an existing condition and select **Add another condition**, or use the **Add...** link on the right of the *Add action* link.

Actions can be added by clicking the **Add action** link (if it has not been hidden in the ribbon), or right-clicking the margin or an existing action and selecting **Add another action**. See also the Add Action dialog.

## Modifying events

**Double-click** or select and press `Enter` on condition or action to edit it.

Events, conditions and actions can be **dragged and dropped** around the event sheet. Holding `Control` and dragging will duplicate the dragged event, condition or action. Event items can also be **cut, copied and pasted**.

You may find it convenient to organise events in to groups, which can also be activated and deactivated as a whole.

Press `R` or right-click and use the Replace object option to quickly swap objects referenced in the selection. Note that objects with references to instance variables or behaviors in the selection can only be swapped with other objects with the same instance variables and behaviors which have the same names and types.

## Scrolling and scale

There are several ways to scroll in the Event Sheet View:

- **The vertical scrollbar** to the right of the view
- Scrolling the **mouse wheel**
- **Middle-clicking** to pan the view

- Pressing `Space`, **up/down** arrows or **page up/down**

There are some options to adjust the text size in this view as well:

- Hold `Control` and **scroll the mouse wheel**
- Press `Control` + **+** or **-**
- Right-click and use the Event sheet▶Font size menu
- Use the browser's zoom feature, but note this scales the whole of Construct, not just the text scale in the Event Sheet View.

# Finding in events

Paid plans only You can search for some text in an event sheet by pressing `Ctrl` + `F` or right-clicking and selecting Event sheet▶Find.... This opens a dialog that allows you to enter text to search for, with options to look in the current sheet or the entire project, and whether to make it a case-sensitive search. (Case sensitive searches count uppercase and lowercase characters as different, e.g. "SPRITE" and "sprite" are different with a case-sensitive search.) When you click *Find*, the results are displayed in the Find Results Bar.

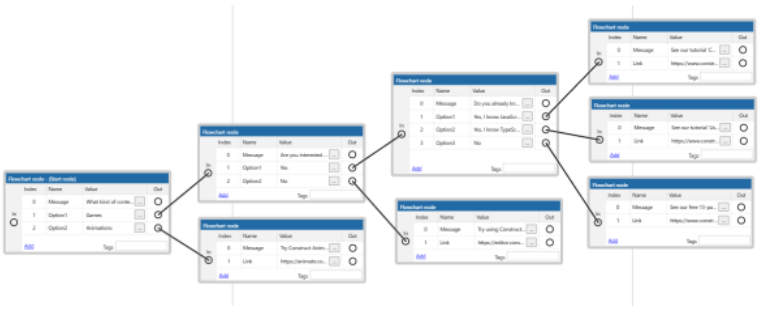
## Find all references

Paid plans only Text search is not always appropriate for finding in events. For example if you want to find all events referring to an object named **Sprite**, searching for the text *Sprite* will also return results for other names like **Sprite2**, since they also include the search term. To solve this, you can use the **Find all references** feature. This is available in many places in Construct for various kinds of things like behaviors and instance variables as well. For objects, you can right-click an object in the Project Bar and select *Find all references*. This will open the Find Results Bar with a comprehensive and precise list of all references to that object, excluding any other references that happen to include the object name. This is a great way to easily review your project with confidence the results are what you want.

# THE FLOWCHART VIEW

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/flowchart-view

The **Flowchart View** allows visually editing a flowchart by setting up nodes, making connections between the nodes to form a tree structure and adding information that will be associated with each node. See also the manual section for flowcharts.

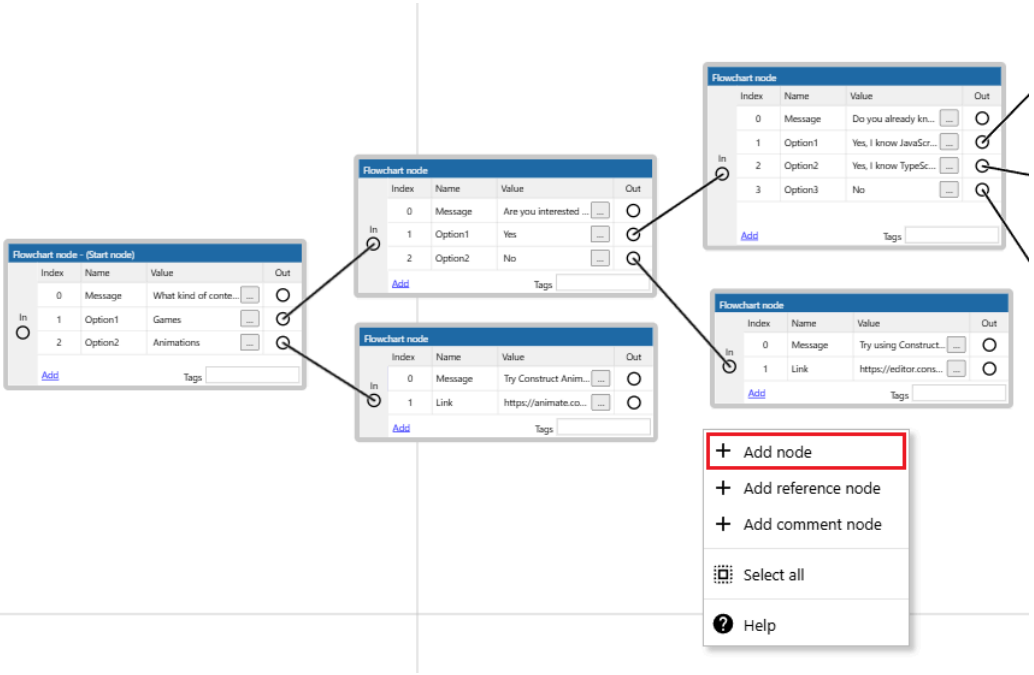


*The Flowchart View*

Asides from showing the nodes and the connections between them, there is also a **status bar** at the bottom which displays the current **zoom level**, the **caption** of the last selected node and the **index**, and the **name** and **value** of the last selected output.

## Adding and deleting nodes

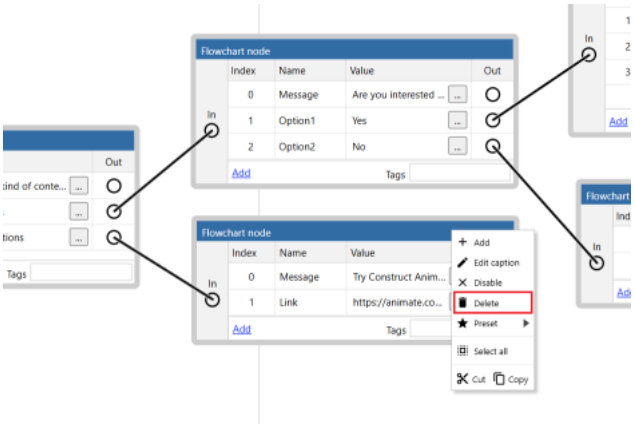
To add new nodes, right-click in any empty space of the flowchart view and tap the Add node option.




*Context menu option to add a node*

To delete a node you can do one of two things:

- Right-click on the caption of a node or any part which is not an output and choose the Delete option.
- Press `Backspace` or `Delete` after a node has been selected.




*Context menu option to delete a node*

# Editing nodes

Once a node has been created, there are a number of things that can be edited about them, including:

---

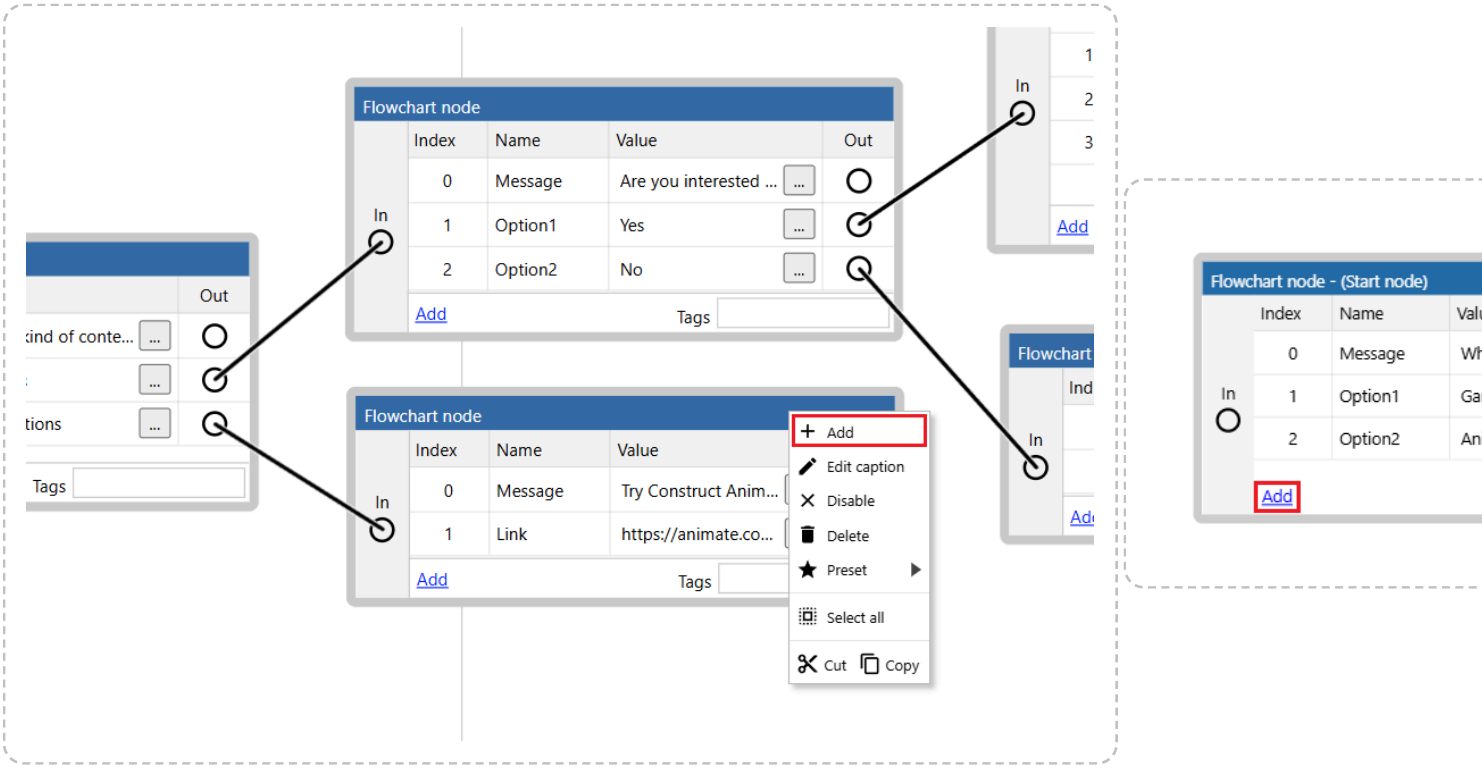
### Adding outputs

This can be done by:

- Right-clicking on the node and choosing the Add option from the context menu.
- Clicking on the Add link at the bottom left of each node.
- Clicking the Add link in the Properties Bar when it is showing properties for a node.

Context menu option

Node link

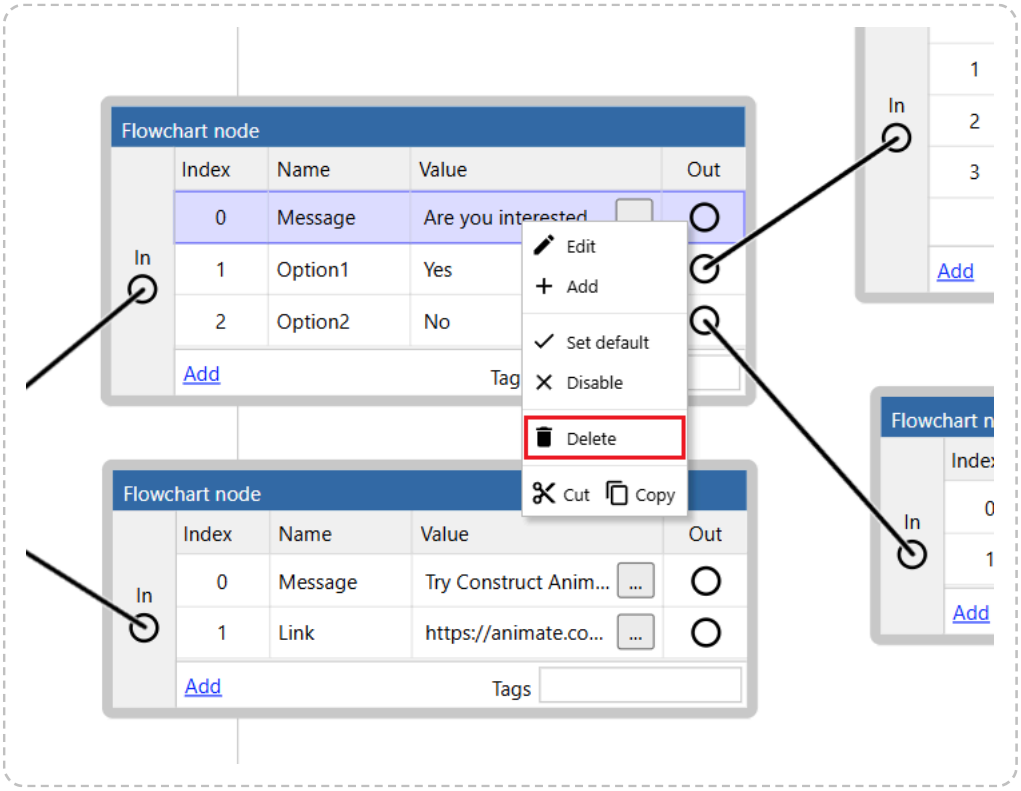



## Deleting outputs

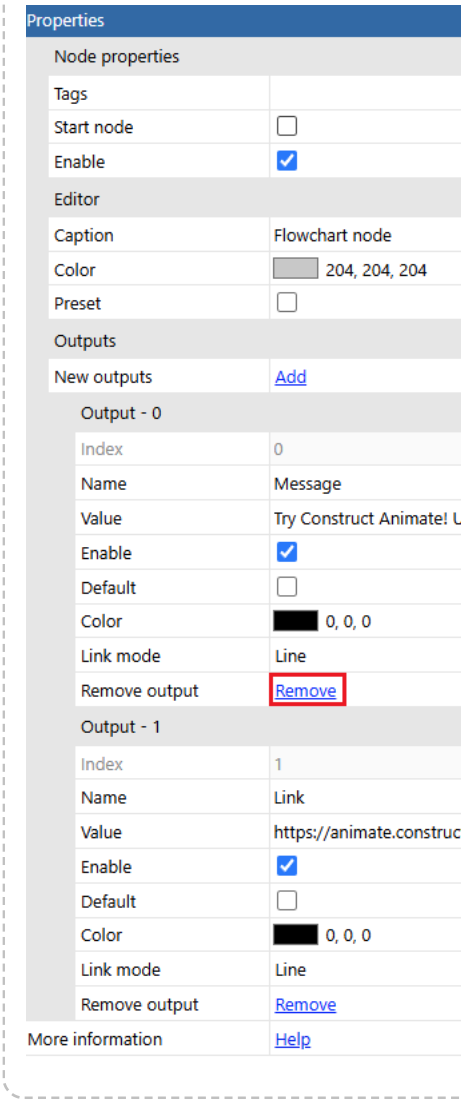
This can be done by:

- Right-clicking on the output and choosing the Delete option from the context menu.
- Clicking the corresponding Remove link in the Properties bar when it is showing properties for a node.

Context menu option




Properties bar



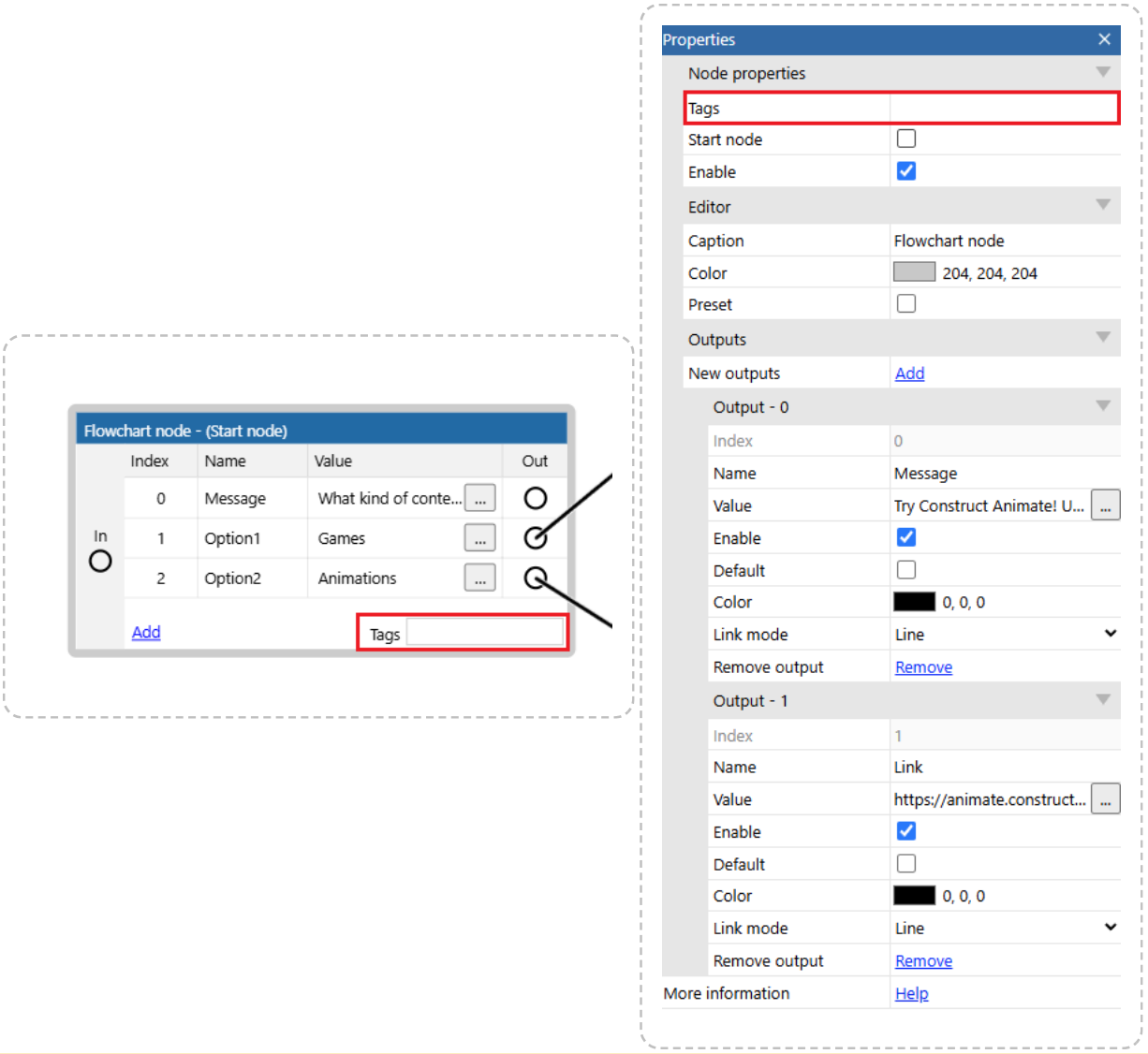

## Editing the tags

This can be done by:

- Typing in the labelled text box in the node itself.
- Editing the corresponding property in the Properties bar.

Tag text box

Properties bar



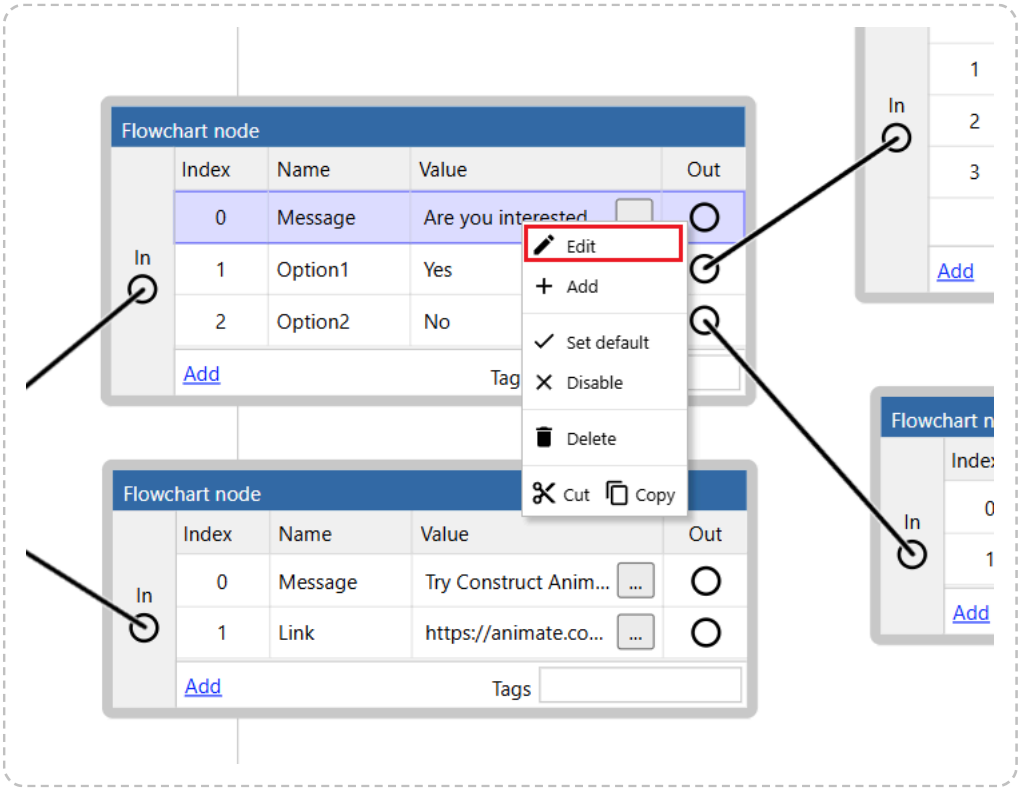

*Use multiple tags by separating them with spaces.*

## Editing an output's name

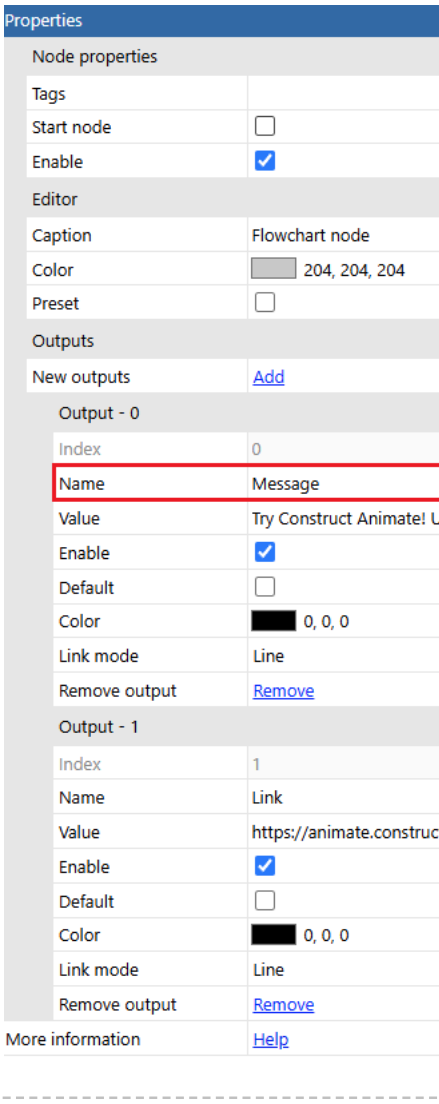
This can be done by:

- Double clicking on an output's name.
- Right-clicking on an output's name and choosing the Edit option.
- Editing the corresponding property in the Properties bar.

Context menu option




Properties bar

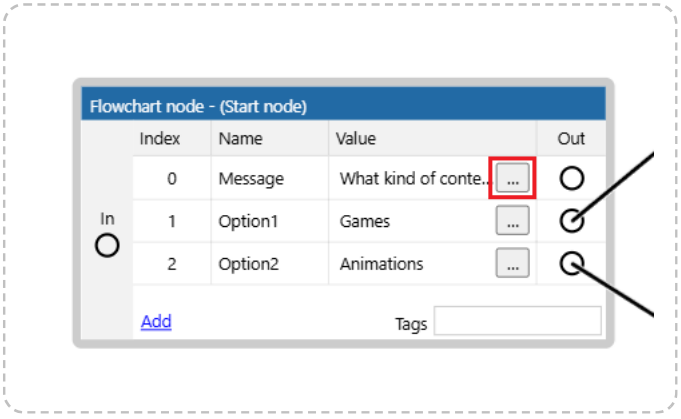



## Editing an output's value

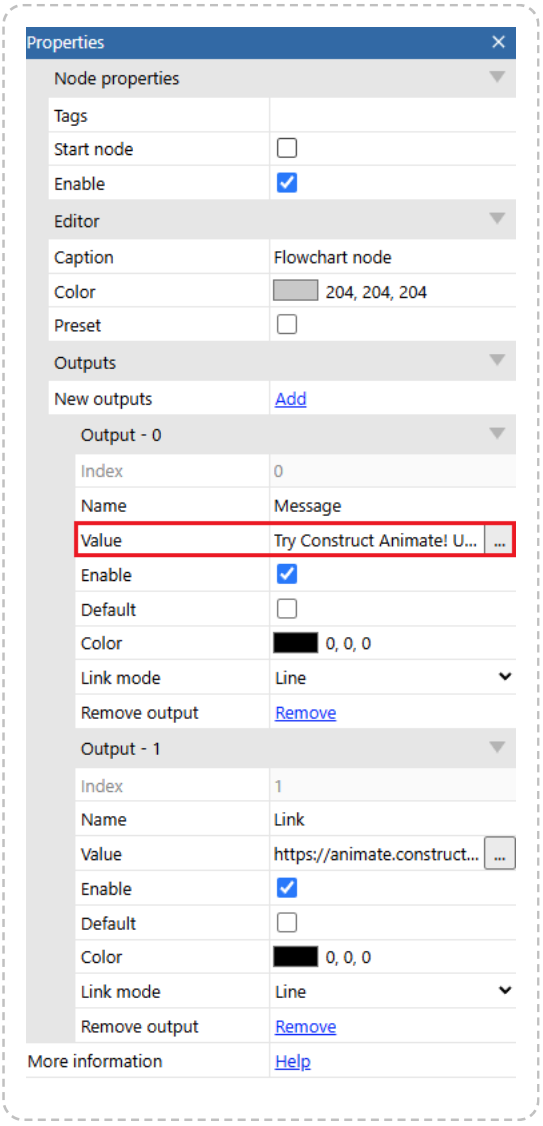
This can be done by:

- Double clicking on an output's value.
- Right-clicking on an output's value and choosing the Edit option.
- Clicking on the button next to the value.
- Editing the corresponding property in the Properties bar.

Button next to value




Properties bar




The **Caption** of a node can be edited from the Properties bar, by right clicking it and choosing the Edit caption option or by double clicking it on the corresponding node.

A few properties of nodes can only be edited from the Properties Bar, such as **Start node** and **Color**.

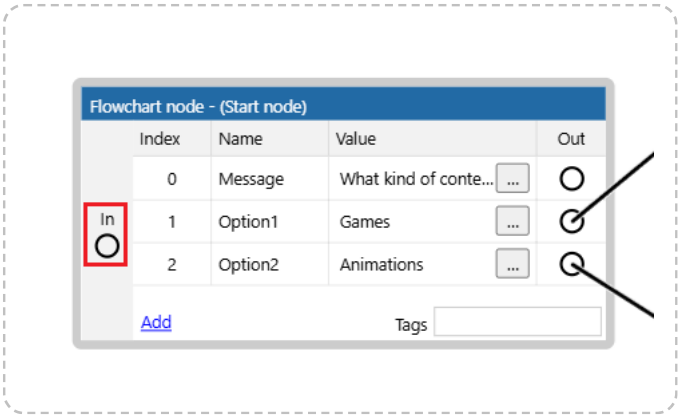
## Sorting outputs in a node

After an output is added to a node it is possible to change its position by clicking and dragging it to be on top or below another output in the same node.

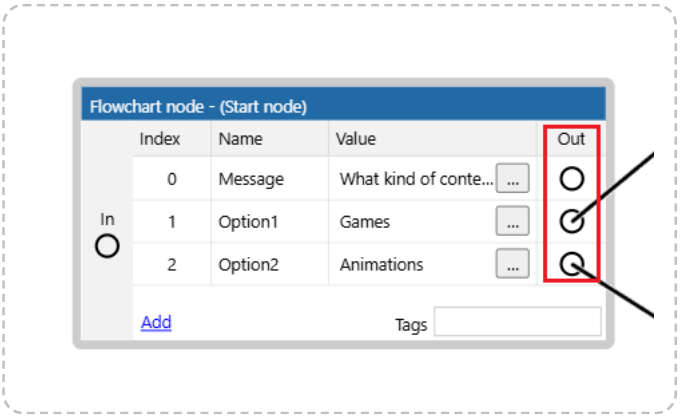
## Connecting nodes

To connect two nodes click an **output** icon and then drag to the **input** of another node. This can be done the other way around too, clicking and dragging from an **input** icon into an **output** icon.

Input Icon




Output Icon



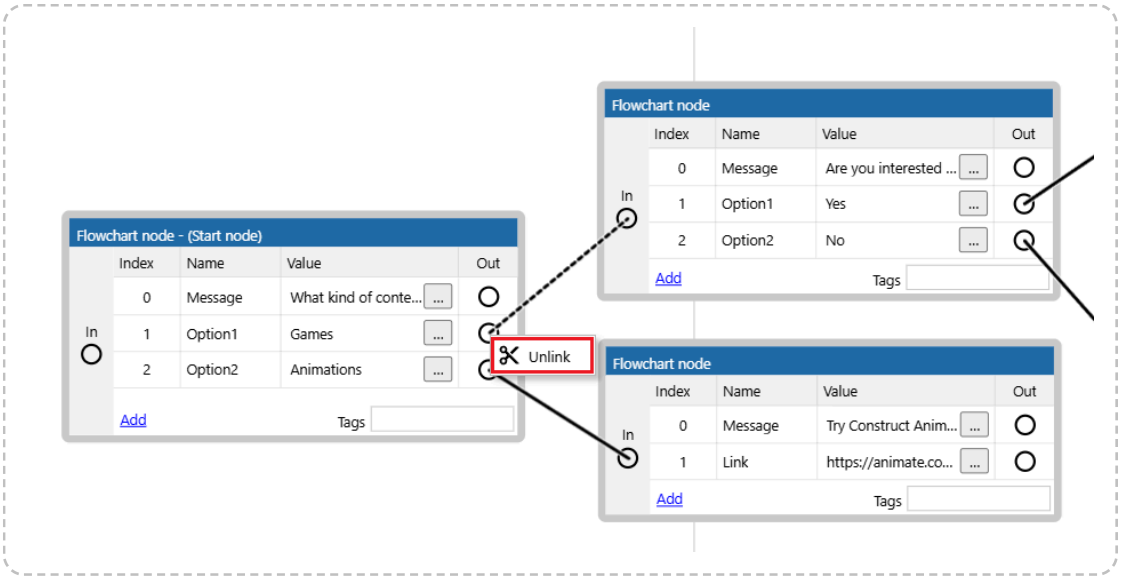

*Clicking and dragging from an **output** and letting go over empty space in the flowchart will show a context menu to add nodes. If any option is picked a new and already connected node will be created.*

## Unlinking nodes

To remove the connection between two nodes, right click on the corresponding **output** or **input** and choose the Unlink option from the context menu.

Links can also be selected by clicking on them (the cursor changes when clicking will select it), when a link is selected it will show as a *"marching ants"* animation and at that point it can be deleted by using the `Backspace` or `Delete` keys in the keyboard.

Unlink from Output Icon

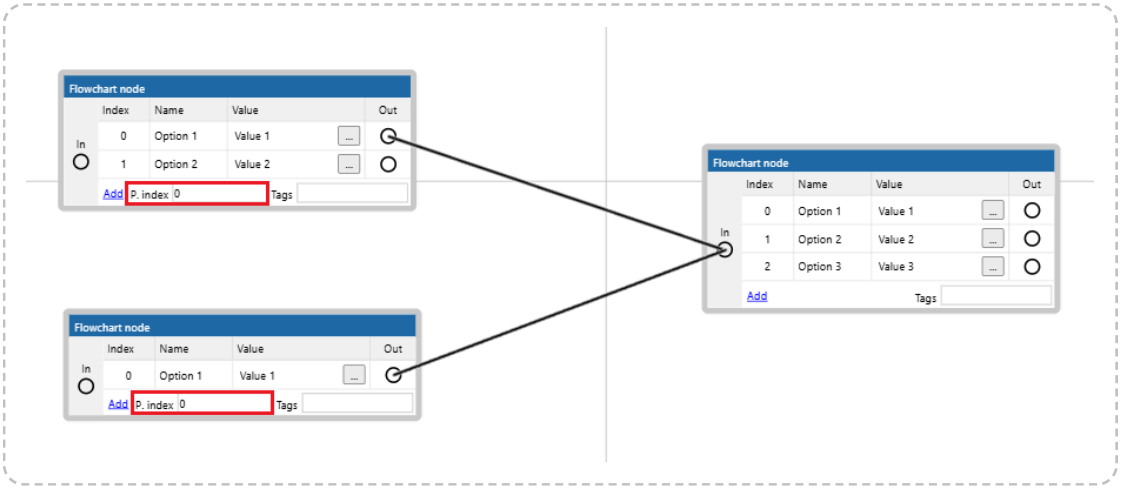



## Connecting many outputs to the same input

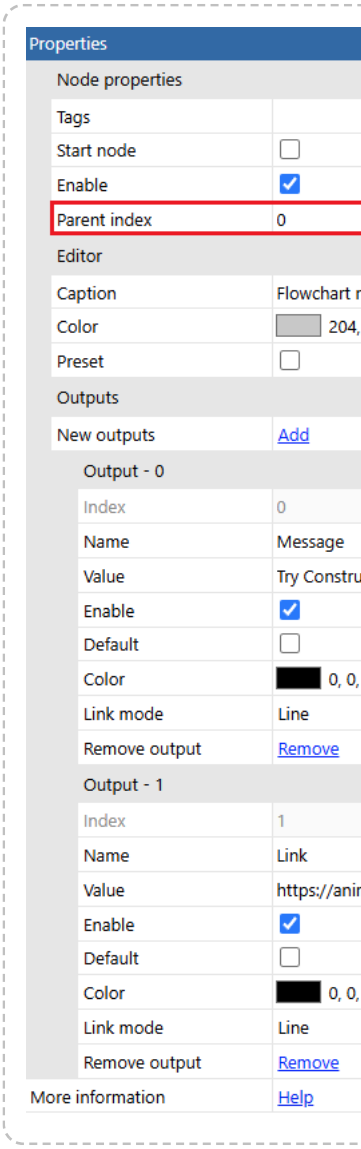
In the case of connecting multiple outputs to the same input, each of the parent nodes will start showing a **Parent Index** property, both in the node and the Properties bar.

The **Parent Index** can be used with some actions and expressions that need to choose which parent to use in the case a node has more than one.

Parent Index in node




Properties Bar

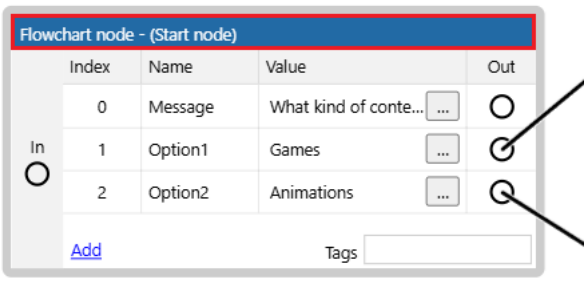



# Moving and resizing nodes

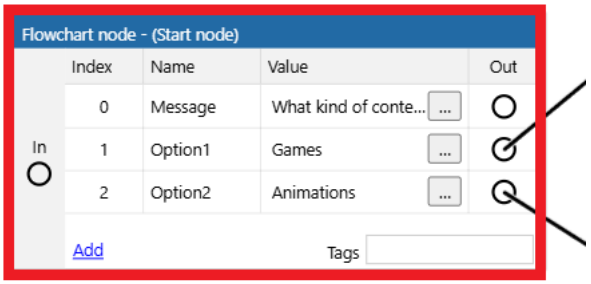
Nodes can be moved by clicking and dragging from the caption and can be resized by clicking and dragging from the borders.

The columns in each node can also be resized by clicking and dragging between them.

Move drag




Resize drag




Column resi



*The position, size and column size of the nodes in a flowchart is only for display purposes and has no effect at runtime.*

## Multi-selection

By holding `Shift` while selecting nodes and outputs it is possible to select multiple elements at the same time.

Doing this it is possible to delete multiple nodes and outputs at the same time as well as moving multiple nodes at the same time.

## Rectangle selection tool

Clicking and dragging in empty space will trigger the appearance of a selection rectangle. Any nodes overlapping the rectangle when the pointer is released will be selected. If `Shift` is held down the nodes will be added to any existing selection.

## Cut, copy and paste

Cut, copy and paste of nodes is supported through context menu options and the common keyboard shortcuts. Cutting or copying when there is an active multi-selection will perform the appropriate action on the whole selection.

Once there is flowchart content on the clipboard it is also possible to paste it in a different flowchart to the original.

## Scrolling and Zooming

There are a few ways to scroll in the Flowchart View:

- Use the vertical and horizontal scrollbars at the edges of the view.
- Scroll the mouse wheel to scroll vertically. You can also hold `Shift` to scroll horizontally.
- Hold the middle mouse button and drag the mouse.
- Hold `Space` and move the mouse.

On desktop systems, middle-mouse dragging is probably the most convenient way to move around the flowchart.

To zoom hold `Control` (`Meta` on Mac) and scroll the mouse wheel.

Make sure to check the Keyboard shortcuts section of the manual for a complete list.

## Managing complex flowcharts

When flowcharts start becoming large it can be difficult to keep track of what is going on just by looking at it.

These are a few features that can help with that:

**Node colors**

All nodes can be given a unique color which is applied to it's border. Helps highlight important nodes.

**Output link colors**

Each output can be given a unique color that is used to draw it's corresponding link. This can help to differentiate between links when it starts getting crowded. This property can only be changed from the properties of an output shown in the Properties Bar.

**Output link mode**

Each output can use two different methods to draw it's corresponding link. It can be either a straight line (default method), or it can use path finding so the link avoids overlapping other nodes. This can help in cases were a straight link is overlapping other nodes making the flowchart look messy. This property can only be changed from the properties of an output shown in the Properties Bar

**Reference nodes**

This types of nodes are used as a placeholder for a different flowchart. Can be useful to split up large flowcharts in more manageable chunks.

**Comment nodes**

This types of nodes are used to place comments directly in a flowchart.

## Preset nodes

Nodes can be saved to be used later as a base to create new nodes.

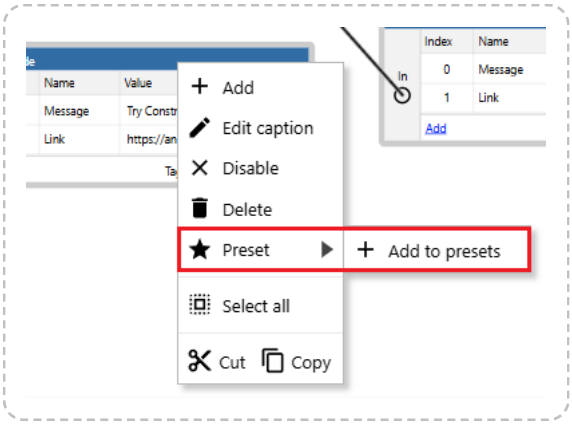
This can be useful in more complex use cases as it is likely that in any given project all of the existing flowchart nodes will follow a similar structure so they can be easily processed by an Event Sheet.

Manually adding all the outputs and editing the names of the outputs so they match the format being used every single time a node is created can quickly become tedious and also error prone.

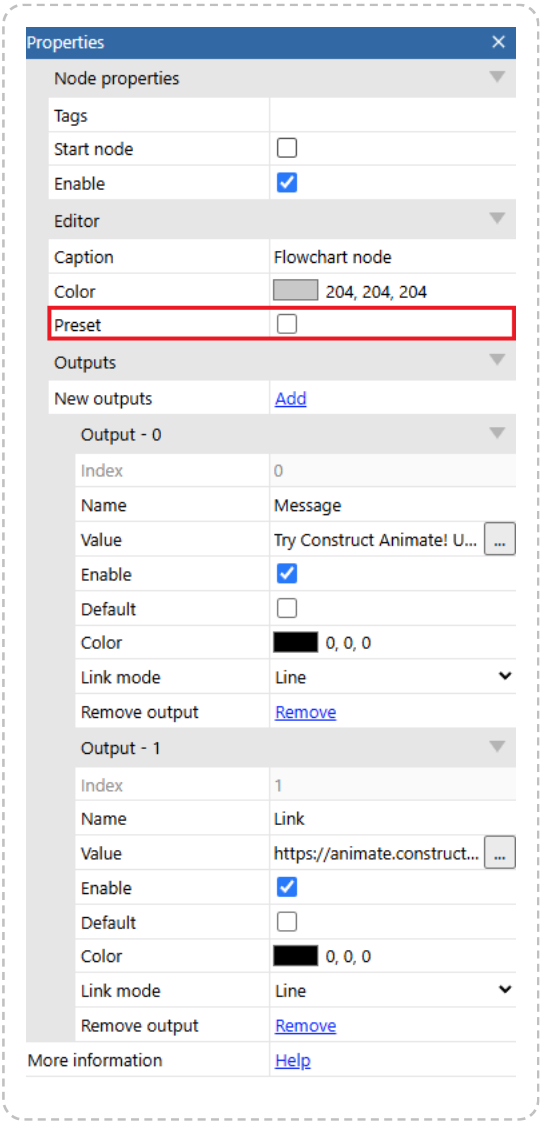
Identifying the format of the nodes that are going to be used more often and saving them as presets can save time, avoid errors and just make the experience of editing a flowchart a little bit more enjoyable.

Saving a node as a preset is easy, you can either change the corresponding property in the Properties bar or use the Preset▶Add to presets context menu option of a node.

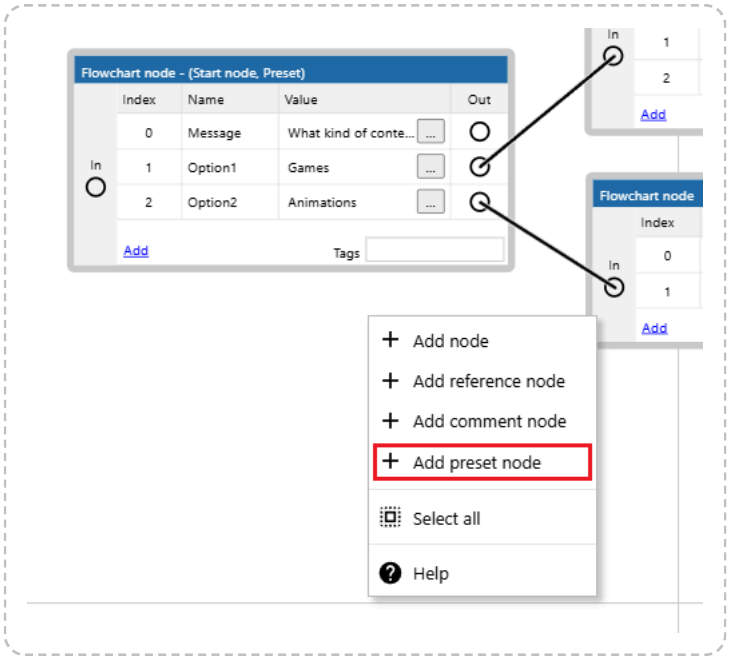
Preset Context Menu



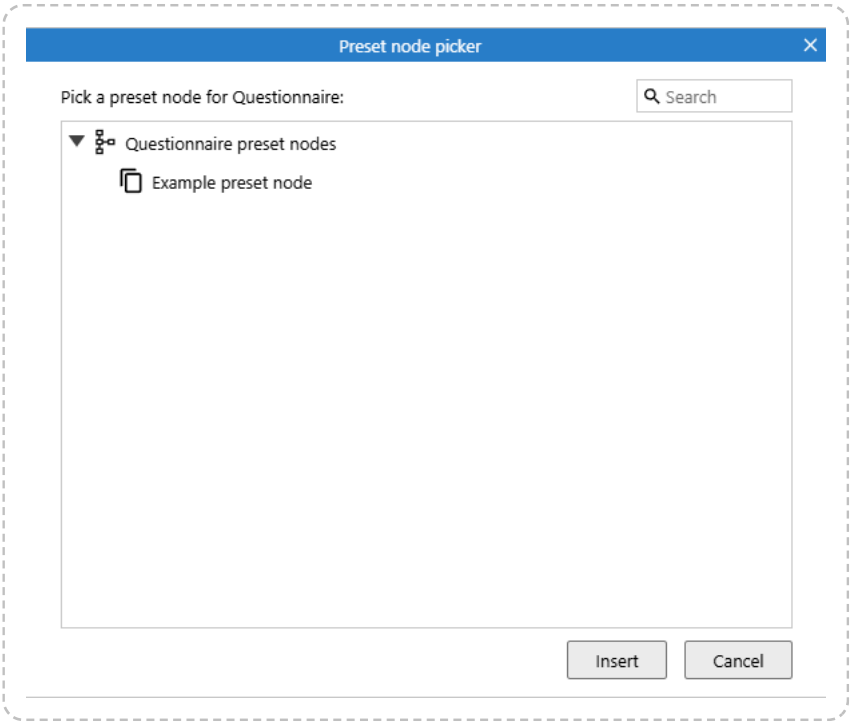

Properties Bar




Once there are some nodes set as presets, a new option to create nodes will appear among the context menu options to create nodes.




Clicking on it will show a dialog with all the preset nodes in the current project.




The **Preset node picker** dialog shows all the preset nodes as a tree with the flowchart they belong to as their top most parent. The tree like other similar controls in Construct allows to, create folders, delete, rename and rearrange items as needed.

The name used for the items in the tree is the caption of the corresponding node, so renaming an item in the picker dialog will change the caption of the corresponding node.

> *Once a new node is created from a preset, the base node and the new node are independent. Modifying the base won't have any effect on the new node!*

*All preset nodes are available to be used in any flowchart of the same project.*

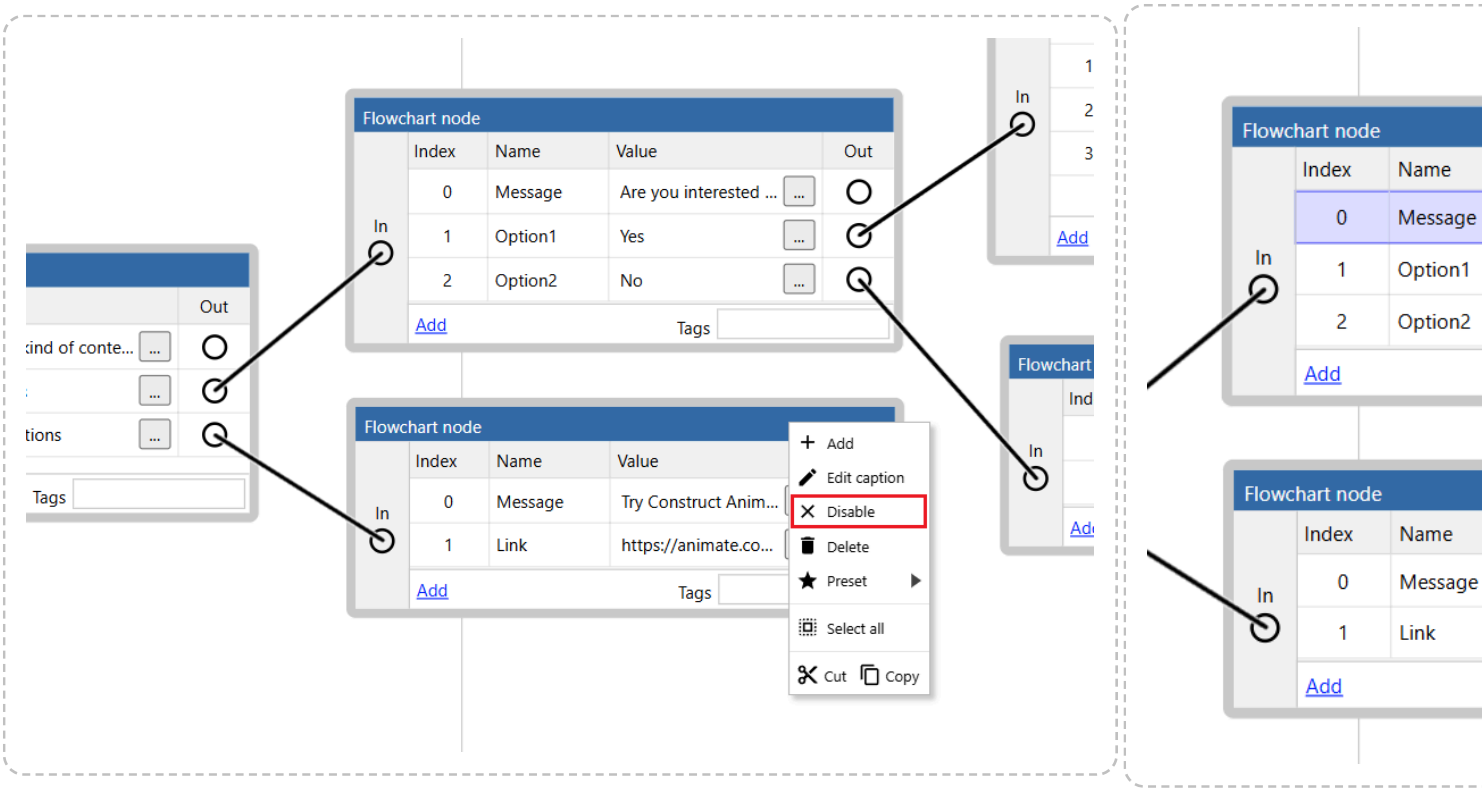
## Disabling nodes and outputs

When editing a flowchart it can be useful to disable nodes or outputs to try out runtime changes without outright deleting elements of the flowchart. This can be thought as commenting lines of code in traditional programming, a method to make changes that are not yet final.

Nodes and outputs can be disabled using the corresponding context menu options or by means of the Properties bar.

Node Context Menu

Output Context Menu

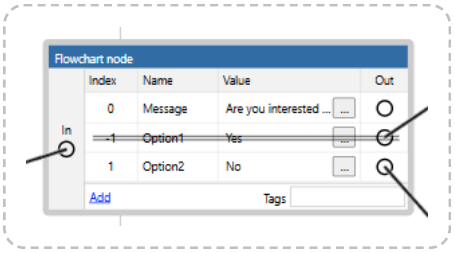



Both disabled nodes and outputs give feedback of the state in the main view to make them easily identifiable.

Disabled Node




Disabled Output



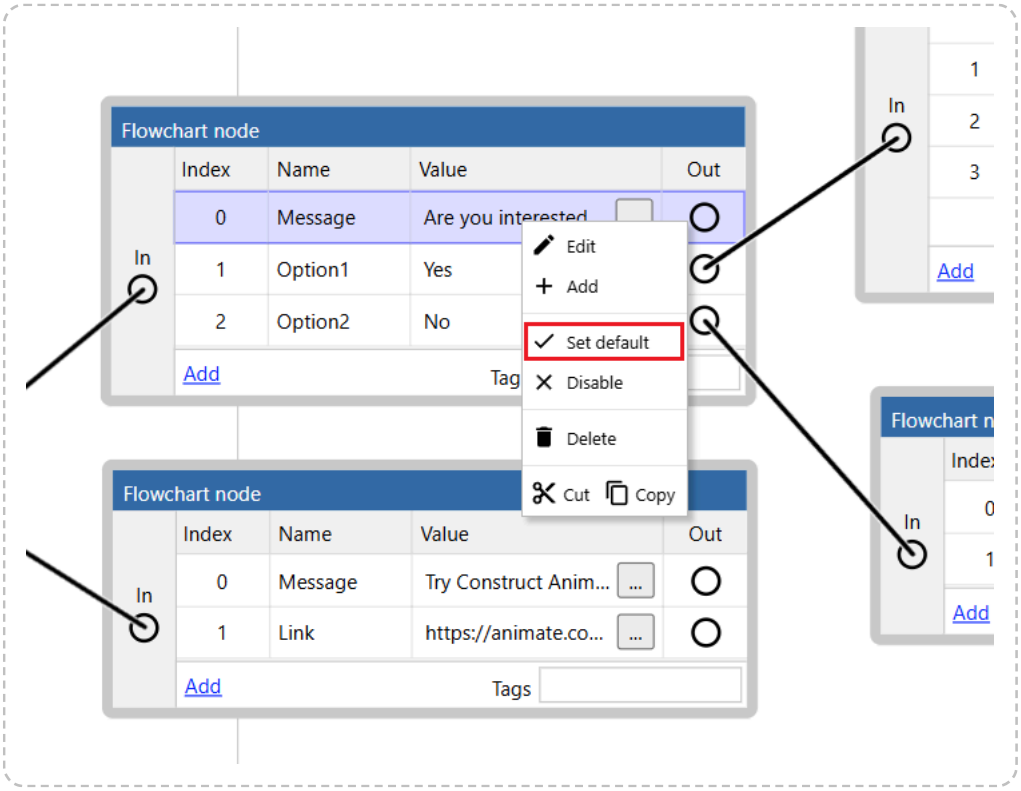

At runtime, a disabled node is treated as if it's not there, so if the state of the flowchart tries to advance into a disabled node, it won't happen, effectively cutting off any nodes past the disabled one.

Similarly a disabled output is treated as if it doesn't exist at runtime, it can't be followed, it's information can't be queried and it's not considered in the for each loop that iterates through outputs.

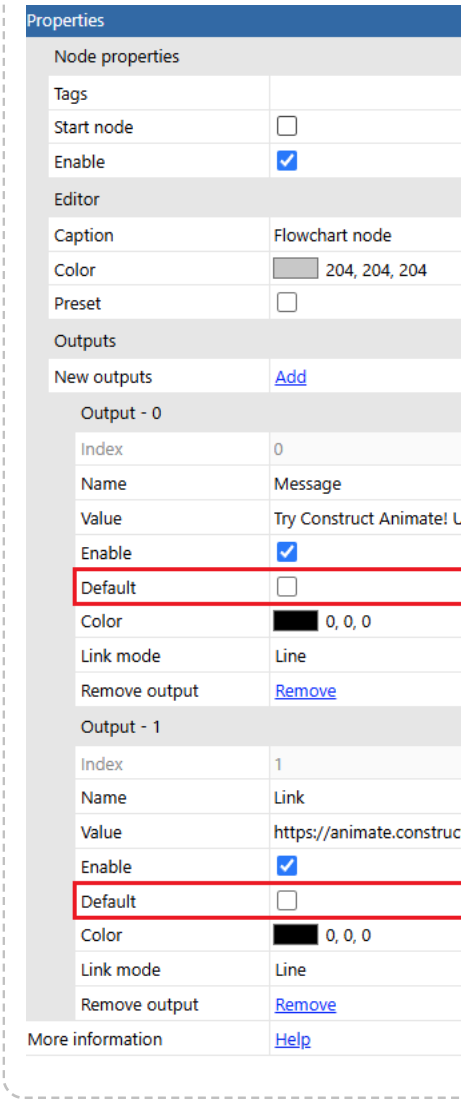
## Default outputs

Outputs can be set as the default one in their node by a context menu option or through the Properties bar.

Context Menu

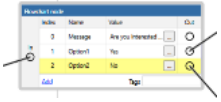



Properties Bar



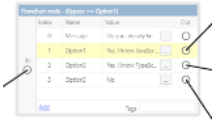

The two use cases of this setting are the following:

1. A default output can be chosen at runtime using the Flowchart Controller ACEs that specifically target them. This allows to more easily manage a default option in each node without the need of enforcing a convention when designing the nodes.



*Default output*

2. In the case of a node being disabled, the runtime follows default outputs automatically if the state of the flowchart falls in a disabled node. If the following node is also disabled the runtime will attempt to continue following default outputs until it lands on a node which is not disabled.



*Disabled and Bypassed node*

*If a node doesn't have an explicit default output set, the Flowchart Controller ACEs that target default outputs don't do anything.*

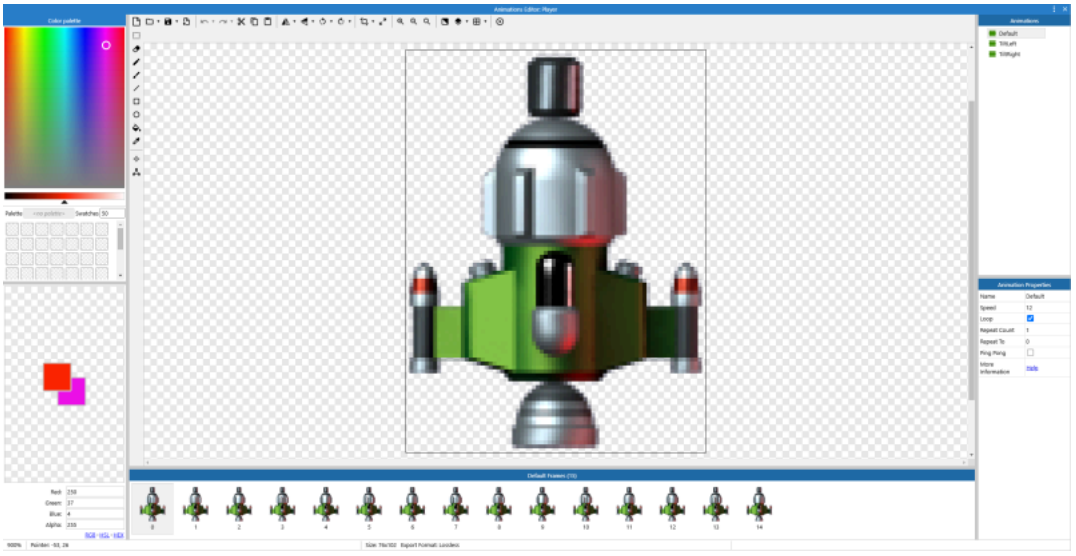
*The editor doesn't allow to form infinite loops of disabled nodes with default outputs because it would crash the runtime. The editor will give feedback if such a structure is attempted to be setup and revert the changes.*

# THE ANIMATIONS EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/animations-editor

Construct has a fully-featured built-in image and animations editor, used to create animations for the Sprite object. When opening for an object without animations, such as Tiled Background, the animation editing features are hidden and it acts as a normal image editor. For brevity it is consistently referred to as the Animations Editor, even in cases where it is only editing a single image.

To open this editor, **double-click** an object with an image or animations in the Layout View or Project Bar.



*The Animations Editor*

Note each pane in the Animation Editor can be resized by dragging the borders, similar to how you can with the main Construct interface. This lets you customise the layout of the Animation Editor.

## Color palette

The color palette appears on the left and allows a color to be picked for the drawing tools. You can choose both a primary and secondary color with left and right click. The pane also has a number of cells that can be used to remember a set of colors. Right-click a cell to save or use the primary or secondary color. By default left-clicking the cell will set the primary color.

### Saving a palette

Right clicking anywhere in the area where swatches of the color palette are will bring up a context menu with the option Save palette, click it to save the palette and be able to retrieve it

later. Saved palettes are associated with the object type or project file that is currently being edited, but can be used on other object types or project files.

### Picking a palette

If the project has any saved palettes, one can be picked by either:

1. Clicking on the Pick palette option in the context menu.
2. Clicking the button on top of the swatches with the label **Palette**.

Any of those methods will bring up a dialog from which any of the palettes stored in the project can be picked.

### Downloading palettes as a file

One of the secondary options of the Save tool, found in the top toolbar, is to save a single palette as a JSON file, or multiple palettes as a zip file. You can choose to save the currently selected palette, all the palettes available to the current object type or all the palettes available to the current project.

### Loading palettes from a file

Similar to the Save tool, the Load tool has a secondary option to load palettes. It can load individual palettes or multiple ones bundled in a zip file. All the palettes are loaded into the current object type.

> *The tool only understands palette files in the format that is saved by the Save tool. If you wish to load a color palette generated by an external program, you will need to generate a file which the Animations editor can understand. See the color palette file format below for a description of it.*

### Pasting colors

You can paste text specifying a color in to any of the color inputs to set the overall color. The text can be in any of the following formats:

- `r, g, b` or `r, g, b, a`
- `rgb(r, g, b)` or `rgba(r, g, b, a)`
- `hsl(h, s, l)` or `hsla(h, s, l, a)`
- Hex as either `#ffffff` or `ffffff`

Color components can be in the range **0% - 100%** or **0 - 255**. Alpha components should be between **0 and 1**.

# Top toolbar: image tools

The top toolbar in the image pane provides tools that affect the entire image, such as mirroring and flipping. The tools available are as follows.

- **Clear:** resets the image to all transparent.
- **Open:** import an image from a local file. Note you can also choose **SVG** files, but these will be rastered in to a bitmap at a given size. The dropdown next to the main button also has an option to load color palettes from JSON or zip files.
- **Save:** export a copy of the current image. In the browser this downloads the current image as a PNG file. You can use the dropdown next to the button to save the current animation or all animations, bundled in a zip file. It is also possible to download the images with the associated image point and collision polygon data. One of the dropdown options allows you to save the current color palette to a JSON file, all the color palettes of the object type to a zip file or all the color palettes of the project to a zip file.
- **Set export format:** opens the *Image Format dialog*, allowing you choose whether the image is saved as lossless (i.e. perfect quality) or lossy (i.e. allowing some quality reduction in order to further reduce the file size) when the project is exported. The specific formats that these mean are chosen when exporting - for example lossless images can be exported as either PNG or WebP. This can also be applied to the current animation, or all animations. Note Construct stores all images in the project in a lossless PNG format; images are only converted on export.
- **Undo** and **Redo**: step through the change history.
- **Cut**, **Copy**, **Paste**: perform clipboard operations with the image.
- **Mirror** and **Flip**: invert the image on one of its axes. Use the dropdown next to the button to affect the entire animation.
- **Rotate anti-clockwise** and **Rotate clockwise**: rotate the image in 90°. Use the dropdown next to the button to affect the entire animation.
- **Crop:** resize the image smaller to remove transparent space around the edges of the image. This is a good idea to save memory. Note this leaves a 1px transparent border to improve the image quality at the edges. Use the dropdown next to the button to affect the entire animation or all the animations in the object type.
- **Resize:** resize the current image. A dialog will open with options for the resize, including a checkbox to apply the resize to the current animation or to all the animations in the object type.
- **Zoom in**, **Zoom out**, **Reset zoom**: adjust the zoom level in the image editor. Alternatively use **Control + mouse wheel**.
- **Toggle background brightness:** switch between a light and dark background for the image editor. Changing to a dark background can be useful when editing very light images.
- **Toggle onion skin:** Paid plans only display adjacent frames translucently over the current image when editing an animation. This can help when drawing animations. The options are

to display the previous frame, next and previous frames, or next and previous two frames over the current image.

- **Grid:** toggle the display of a grid over the current image. Use the dropdown next to the button to adjust the grid settings, such as the grid size, color and whether to snap to the grid.
- **Preview:** preview the currently selected animation.

## Side toolbar: drawing tools

The side toolbar provides some tools for drawing in the image, as well as some extra Construct-specific tools for setting image points and adjusting the collision polygon. Some tools have extra settings, such as the size for the brush tool, which appear underneath the top toolbar. The following tools are available.

- **Rectangle select:** select, move, delete, cut, copy and paste rectangle sections of the image.
- **Pencil:** draw with a solid square, useful at 1px size for pixel art.
- **Brush:** draw with a soft round brush.
- **Line:** draw straight lines. Hold shift to lock the angle to 5° increments.
- **Rectangle:** draw a rectangle in the image. The secondary color is used as a border. Hold shift to draw a square.
- **Ellipse:** draw an ellipse in the image. The secondary color is used as a border. Hold shift to draw a circle.
- **Fill:** fill a continuous area of the image with a color.
- **Eye dropper:** select a color from the image. Alternatively hold Control and click with another tool selected.
- **Image points:** display and edit the origin and image points in the image. This switches the color palette pane to a list of image points, allowing you to add and remove image points. The origin determines the rotation point of the image, and is where the X and Y co-ordinates of the object are aligned to. Image points can be used in the event system to refer to alternative positions. For example you may place an image point at the end of a gun, so spawned bullets can appear at the end of the gun barrel instead of at the origin.
- **Collision polygon:** adjust the area that counts as colliding for this image. By default Construct guesses a collision shape, but it is not always accurate. Click and drag the points of the collision polygon to alter its shape. Right-click on a point or in a space to display a menu of additional options for the collision polygon, such as adding and deleting points. Some objects, like Tiled Background, do not use collision polygons.

## Frames pane

The bottom pane displays a list of all frames in the current animation. Frames can be added and deleted here. Select a frame to switch to editing its image. Frames can also be **dragged and dropped** to adjust their sequence.

Selecting a frame shows a few properties in the properties pane:

- **Index:** the 0 based index of the currently selected frame, this property is for feedback only and can not be edited through the properties pane.
- **Duration:** is a multiplier for the amount of time to spend on the frame. For example, a frame duration of 2 will spend twice as long on that animation frame, 0.5 half as long, etc. relative to the current animation speed.
- **Tag:** a string to identify this animation frame at runtime.

**Right-click a frame** to duplicate or delete it.

**Right-click in an empty space** in the pane to see additional options for managing the animation, which include:

- **Add frame:** add a new empty frame at the end of the animation
- **Duplicate last:** add a new frame which is a copy of the last frame in the animation
- **Reverse frames:** invert the sequence of frames in the animation
- **Import frames**
  - **From files:** add multiple animation frames by selecting a set of local image files to import
  - **From sheet:** add multiple animation frames by selecting a local image file with multiple images placed on it (often called a *sprite strip*), and cutting it up in to individual images. A dialog shows up when selecting this option which allows to choose how the source image should be sliced and which images to extract from it.

You can also adjust the size of the frame icons appearing in the pane by adjusting the **Thumbnail size**.

## Animations and properties panes

The right side of the Animations Editor shows the Animations pane, where animations can be added, edited and deleted. **Right click** a space to add a new animation or add a subfolder to organise animations. Right click an animation to see options like **Preview**, which shows how the animation will look in the game. You can also **Find all references** Paid plans only for an animation to locate all its references in events.

**Right-click in an empty space** in the animations pane to see a few context menu options, of note are:

- **Import animations**
  - **From files:** allows you to select multiple files to be added to a new animation. Also supports choosing zip files containing multiple images. In that case, a new animation is created for each zip file and each image found in the zip is added as a new frame of the corresponding animation. If the zip file contains a **c3-import-settings.json** file, it will

affect the animation creation accordingly. See the **Bulk Importing** section for more information on that.

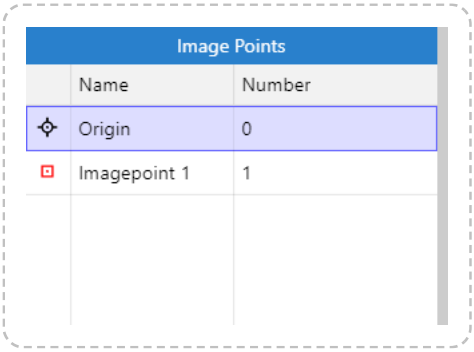
- **From sheet:** shows a dialog that allows to extract multiple animations from a local image file.

Selecting an animation also switches which frames are showing in the frames pane, and displays settings for the animation in the Properties pane. The following properties are available for animations.

- **Name:** the name of the animation. This can also be directly edited in the Animations pane.
- **Speed:** the rate at which to play the animation, in animation frames per second. For example if set to 5, each frame will last 1/5th of a second. Note this cannot be faster than the game framerate, which is typically 60. Set to **0** if you do not want the animation to play, which is useful if you want to control which frame is showing by events. You can also use negative speeds, which causes the animation to play backwards. Note in this case repeating animations should set the *Repeat to* frame at the *end* of the animation, otherwise by default it repeats to frame 0 (the start of the animation), causing the animation to stop after playing in reverse.
- **Loop:** enable to infinitely repeat the animation.
- **Repeat count:** if the animation is not looping, the number of times to repeat the animation.
- **Repeat to:** the zero-based index of the animation frame to go back to when the animation loops or repeats.
- **Ping pong:** play the animation alternately forwards and backwards when looping or repeating.

## Image points pane

When you select the **Image points** tool in the image editor, the left pane switches to a list of image points for the current animation frame.




The **Origin** is a special kind of image point defining the center of the object, or its point of rotation. It has a different icon to denote it. The term *image point* usually means "image points including the origin". Image points have a **zero based index**, and the first image point (number 0) is always the origin. The origin cannot be renamed.

You can also add additional image points. These are useful to create spawn points for other objects. Since you can create objects at image points in events, it is often useful to place an image point in places like the end of the player's gun in the image. Image points can also be given a name, and referred to in events by this name.

### Editing image points

**Select an image point in the list** and a corresponding point will appear on the image. Left click to place the point under the mouse. The arrow keys can also nudge it 1 pixel in each direction.

An image point can be quickly placed using the **num pad**, e.g. **7** for the top-left corner or **5** for centered. Alternatively the image point can be **right clicked** in the Image Points pane and an option chosen from the **quick assign** menu.

**Right clicking** an image point in the Image Points pane also provides options to:

- **Apply to whole animation:** sets the image point in the same relative place in all frames in the current animation.
- **Apply to all animations:** sets the image point in the same relative place in all frames in all the animations of the object type.
- **Apply to all animations in subfolder:** sets the image point in the same relative place in all frames in all the animations in the same subfolder as the current animation.

If an image point does not exist in all frames, these options also creates it. **Holding shift** while placing the image point is a shortcut for **Apply to whole animation**.

## Drag-and-drop

There are various ways you can import images by drag-and-drop in to the Animations Editor window.

### Into main image pane

An image file can be dropped in to the main image editing pane to replace the content of the current frame with the dropped image file. This works the same way as using the **Open** button on the top toolbar.

### Frames pane

Dropping a single image file in to the Frames pane will prompt you asking how the image should be treated. The image can be treated as a plain image file, in which case the image is added as a new frame in the current animation. This works the same way as the context menu option Import▶From files. The image can also be treated as a sprite sheet, which works the same way as the context menu option Import▶From sheet.

Dropping multiple image files in to the Frames pane will add a new frame for each dropped image file. This works the same way as the context menu option Import▶From files.

### Animations pane

You can drag-and-drop either a single image file or multiple image files in to the Animations pane, and it is handled in the same way as with the Frames pane, except that the frames are added to a new animation.

## Bulk importing

The Animations Editor offers a few different methods to import images in bulk.

### Importing folders

- Dragging and dropping a folder into the **Animations panel**, will create a new sub folder with a new animation inside of it. The animation will have frames for all images found at the root level of the folder. Both the new sub folder and the animation will have the name of the dropped folder.
- Dragging and dropping a folder into the **Frames panel** or the **Main drawing area** will add all the individual images in the folder as frames of the current animation.

### Importing zip files

- Dragging and dropping a zip file into the **Animations panel**, will create a new animation with all images found at the root level of the zip file. The animation will have the same name as the zip file.
- Dragging and dropping a zip file into the **Frames panel** or the **Main drawing area** will add all the individual images in the zip file as frames of the current animation.

### Importing from the toolbar options

The toolbar **Load** button has two options. **Load frames** and **Load animations**.

- **Load frames** allows you to pick images to be loaded as frames of the current animation. It is also possible to pick zip files from the file picker. All images in a zip file will be added as frames of the current animation.
- **Load animations** allows you to pick images to be loaded into a new animation. Is is also possible to pick zip files from the file picker. A new animation will be created with all of the individual images picked, while each picked zip file will correspond to a new animation being created.

> ***Note:*** *loading from the toolbar does not support picking folders.*

### Animated image file formats

In supported browsers, animated image file formats like GIF and APNG can be imported as a sequence of frames. This covers any method of importing an image file, including via drag-and-

drop or by the toolbar *Load frames* option.

> *This is supported in Chrome and Edge. Other browsers may not support this feature, in which case only the first frame of the animation will be imported.*

### Nested content

Folders and zip files which in turn also have folders and zip files inside of them are supported, depending on how you try to import these, this is how C3 will behave.

- Dropping in the **Animations Panel:** C3 will process everything and create animations and sub folders as needed.
- Dropping in the **Frames Panel or Drawing Area:** all of the nested content will be turned into a flat list and added as frames to the current animation.
- Toolbar **Load Frames** option: all the nested content inside a zip file will be turned into a flat list and added to the current animation.
- Toolbar **Load Animations** option: if a zip file with nested content is loaded, the images at the root level of the zip will be used to create an animation, if other zip files are found, new animations will be created for them. If a folder is found in a zip file, a sub folder with an animation inside of it will be created, following the same patters as importing a folder.

The key points to remember are:

- Importing folders will create a sub folder with an animation inside of it, both named after the original folder. The animation will have frames corresponding to the images found at the root level of the original folder.
- Importing zip files will create an animation named after the zip file and will have frames corresponding to all the images found at the root level of the zip file.
- If nested content is found, the same pattern applies. Each folder will correspond to a new sub folder with a new animation inside of it. Each zip file will correspond to a new animation.

> ***Note:*** *Construct does not allow duplicate animation names. If content with duplicate names is imported, Construct will always assign unique names to each animation created.*

> ***Note:*** *By default Construct will assign unique names to imported sub folders aswell, but this is not strictly necessary and can be overridden using the "use-raw-folder-names" configuration option described below.*

## Configuration file options

Normally when importing a single file, C3 asks if the file should be treated as a sprite sheet or as a single file. In the case of importing multiple files this isn't really an option. Because of that, when importing folders or zips, even if an animation ends up having only one frame, C3 never asks how it it should be treated.

To get around this problem, a special configuration file can be added to a folder or zip file to tell C3 how it should handle the files found on them.

It's a simple JSON file, must be named **c3-import-settings.json** and should look like this:

```
{
	"import-mode": "spritesheet",

	"sort": "alphabetical",

	"order": "ascending",

	"replace-existing-animation": false,

	"replace-existing-folder": false,

	"use-raw-folder-names": false,

	"spritesheet": {
		"horizontal-cells": 4,
		"vertical-cells": 4,
		"direction": "horizontal"
	},

	"svg": {
		"width": 100,
		"height": 100
	},

	"animation": {
		"name": "optional-animation-name",
		"speed": 5,
		"loop": false,
		"repeat-count": 1,
		"repeat-to": 0,
		"ping-pong": false,
		"frame-durations": [1, 2, 3, 4],
		"frame-collision-polys": [
			{"points":[0,0,1,0,1,1,0,1]},
			{"points":[0,0,1,0,1,1,0,1]},
			{"points":[0,0,1,0,1,1,0,1]},
```

```
                        {"points":[0,0,1,0,1,1,0,1]}
                ],
                "frame-image-points":[
                        [{"originX":0.5,"originY":0.5}, {"name":"Image Point 1","x":0.5,"y":0.5}
                        [{"originX":0.5,"originY":0.5}],
                        [{"originX":0.5,"originY":0.5}],
                        [{"originX":0.5,"originY":0.5}]
                ],
                "frame-tags":["tag-1", "tag-2", "tag-3", "tag-4"]
        }
}
```

## Configuration file options

- **import-mode** can be either "spritesheet" or "files".

  > ***Note:*** *The* ***import-mode*** *in the JSON file reflects the Animations Editor's interface, so it is only relevant when an animation with only 1 frame is going to be created.*

- **sort** can be either "alphabetical", "numerical" or "no-sort". Defaults to "alphabetical" if not provided.
  - **alphabetical** sort will interpret the file names as characters and will sort alphabetically.
  - **numerical** sort should be used when the names of the files are numbers, so the importer interprets the names as such and sorts as expected.
  - **no-sort** won't do any sorting and leave the incoming files in whatever order they were initially read in. This might not be the expected order.

  > *If a sort mode is not provided and all the files have numerical names* ***Ej. 1.png, 2.png, etc...****, then sorting will default to "numerical".*

- **order** can be either "ascending" or "descending". Defaults to "ascending" if not provided. Allows you to choose the sorting order of the imported files.
- **replace-existing-animation** is optional, can be either true or false and will default to false if not provided. If set to true, if an animation with the same name is found when importing, the imported one will replace the existing one, instead of creating a new one.
- **replace-existing-folder** is optional, can be either true or false and will default to false if not provided. If set to true, if there is an existing folder with the same name as the imported one, all the contents of the existing folder will be replaced with the content of the imported folder.
- **use-raw-folder-names** is optional, by default the importer will create unique folder names. Setting it to true will allow the importer to create folders with duplicate names. When using the Animations editor export with data options, the generated file has this option set to true.

- **spritesheet** mimics the settings you can pick when importing a sprite sheet through the Animations Editor UI.
- **svg** defines what size to use when encountering SVG files.
- **animation** these settings can be used to override the default properties of an animation. If an animation name is provided, then it will take precedence over the name of the folder or zip file.
- The **frame-durations** property of **animation** is an array that allows you to specify the frame duration of each frame in the animation. Add one number for each frame in the animation. If a frame does not have a matching value in the array, the default value of "1" will be used. If the array is not specified at all, all imported frames will have a duration of "1".
- **frame-collision-polys** is an array that allows you to specify the collision polygon for each frame in the animation. Add one object for each frame in the animation. If a frame does not have a matching object in the array, the default collision polygon will be used. If the array is not specified at all, all imported frames will have the default collision polygon. Each object in the array must follow the syntax in the example. This option is not meant to be used manually, instead it's values will be generated when using the options in the Animations editor to export animations with data.

  > ***Note: frame-collision-polys*** *can be used manually but is meant to be generated automatically by Construct when using the exporting options of the Animations editor.*

- **frame-image-points** is an array of arrays that allows you to specify the image points for each frame in the animation. The first element in each array always refers to the origin image point and must be specified using the syntax in the example, subsequent elements refer to image points and must also follow the syntax in the example. Add one array of image points for each frame in the animation. If a frame does not have a matching array of image points, the default image point will be. If the array is not specified at all, all imported frames will have the default image point. This option is not meant to be used manually, instead it's values will be generated when using the options in the Animations editor to export animations with data.

  > ***Note: frame-image-points*** *can be used manually but is meant to be generated automatically by Construct when using the exporting options of the Animations editor.*

- **frame-tags** is an array that allows you to specify the tag for each frame in the animation. If a frame does not have a matching tag in the array, the animation frame will default to having an empty tag. If the array is not specified at all, all imported frames will have the default empty tag. This option is not meant to be used manually, instead it's values will be generated when using the options in the Animations editor to export animations with data.

  > ***Note: frame-tags*** *can be used manually but is meant to be generated automatically by*

*Construct when using the exporting options of the Animations editor.*

## Sorting options examples

These are a few examples illustrating how the sorting options will behave when doing a bulk import.

### Example 1

The following list of file names:

```
"pig.png", "octocus.png", "rock.png", "robot.png", "toaster.png", "monster.png"
```

when sorted in ***"alphabetical"*** and ***"ascending"*** order will end up like this

```
"monster.png", "octocus.png", "pig.png", "robot.png", "rock.png", "toaster.png"
```

### Example 2

The following list of file names:

```
"1.png", "2.png", "3.png", "4.png", "5.png", "6.png", "7.png", "8.png", "9.png", "10.png"
```

when sorted in ***"alphabetical"*** and ***"ascending"*** order will end up like this

```
"1.png", "10.png", "2.png", "3.png", "4.png", "5.png", "6.png", "7.png", "8.png", "9.png"
```

*Notice that in **Example 2**, **"alphabetical"** sorting does not work as expected with file names which are purely numbers.*

### Example 3

The following list of file names:

```
"10.png", "9.png", "8.png", "7.png", "6.png", "5.png", "4.png", "3.png", "2.png", "1.png"
```

when sorted in ***"numerical"*** and ***"ascending"*** order will end up like this

```
"1.png", "2.png", "3.png", "4.png", "5.png", "6.png", "7.png", "8.png", "9.png", "10.png"
```

### Example 4

The following list of file names:

```
"a-10.png", "a-9.png", "a-8.png", "a-7.png", "a-6.png", "a-5.png", "a-4.png", "a-3.png", "a-2.pn
```

when sorted in ***"numerical"*** and ***"ascending"*** order will end up like this

```
"a-10.png", "a-9.png", "a-8.png", "a-7.png", "a-6.png", "a-5.png", "a-4.png", "a-3.png", "a-2.pn
```

*Notice that in **Example 4**, **"numerical"** sorting does not work as expected because the file names are not purely numerical. In this case they can not be interpreted as numbers, even though they might look like they can, so no sorting is performed.*

In order to avoid unexpected behaviour, the best is to choose a naming convention for the files and stick to that, either numbers or characters.

It is worth mentioning that because of the way **"alphabetical"** sorting works it is possible to generate names that include both characters and numbers that will be sorted as expected.

**Example 5**

The following list of file names:

```
"anim-010.png", "anim-009.png", "anim-008.png", "anim-007.png", "anim-006.png", "anim-005.png",
```

when sorted in ***"alphabetical"*** and ***"ascending"*** order will end up like this

```
"anim-001.png", "anim-002.png", "anim-003.png", "anim-004.png", "anim-005.png", "anim-006.png",
```

*Notice that in **Example 5**, **"alphabetical"** sorting with numbers in the file names works as expected because the numbers are **zero padded**.*

## Location of the configuration file

The configuration file can be placed at the root of a folder or zip structure and will affect all content found.

Another file with the same name can be placed further down the hierarchy and will take precedence over the ones found before. That way you can configure different sets of files to be interpreted differently when imported.

## Exporting animations with data

The **Save** option allows you to export animations with their corresponding image point and collision polygon data. This can be done for an individual animation as well as for all the animations in the object type.

The generated file can then be used to bulk import all those animations into a different project. To do so, you can use any of the importing options described earlier.

# Color palette file format

Bellow is an example of the file generated by C3 when it saves a color palette to a JSON file.

```
{
        "name": "MyPalette",
        "slots": "3",
        "palette": [
                "rgba(255, 0, 0, 1)",
                "rgba(0, 255, 0, 1)",
                "rgba(0, 0, 255, 1)"
        ],
}
```

## Color palette file properties

- **name** is optional, if not provided the loaded palette will be given a unique name. If it is provided it will be used, unless it already exists in the project in which case it is used as a base for a unique name.
- **slots** is optional and represents the amount of swatches a palette has. If it is not provided the size of **palette** is used.
- **palette** is a JSON array of colors and must exist. The example uses the **CSS rgba** format for the colors, but any valid **CSS color format** can be used.

*When downloading multiple palettes, C3 generates a zip file which bundles these type of files.*

*If you wish to import a color palette from an external program you will need to generate this type of file.*

# Miscellaneous settings

Clicking on the **three dot** button in the top right of the editor (next to the **close button**), brings up a dialog where you can set the following options:

## Import Settings

---

**Guess collision polygon after importing images**

Toggle this setting to choose if the collision polygon for new imported images should be guessed automatically or not. Affects all methods of importing.

---

**Use file name when importing animations**

Toggle this setting to choose if the file name of the imported file should be used to name a new animation or not. Affects all methods of importing animations.

## Palette Settings

---

**Swatch size**

Choose the size the swatches of the color palette should have.

## Download Settings

---

**Image format**

Choose the file format, quality and compression that the Animations editor should use when saving files to the local file system.

# THE DEBUGGER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/debugger

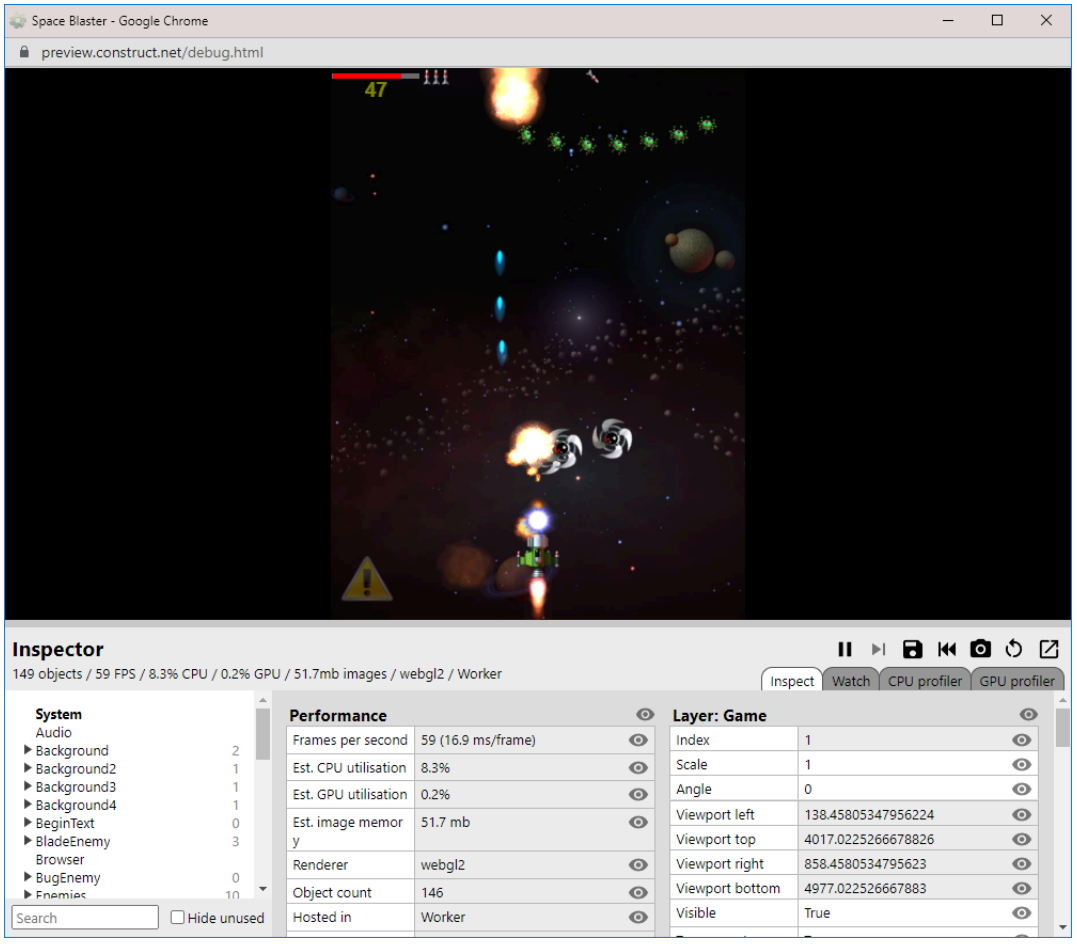
Bugs refer to software defects - things not working as you expected in your project. *Debugging* refers to the process of fixing these issues. Construct's debugger is a tool to help you find and fix bugs in your project.

The debugger has three tabs: Inspect, Watch Paid plans only, CPU Profiler Paid plans only and GPU Profiler Paid plans only. For more information, see the manual entries for each.

## Running the debugger

The debugger appears when you choose the *Debug* preview mode. This can be reached via the main toolbar, the main menu, the Project Bar or using the keyboard shortcut `Ctrl` + `F5`.

The debugger works much like an ordinary preview, except that an extra panel appears alongside the project showing lots of information and some diagnostic tools.




*The debugger showing information about the running project*

### Customising the debugger panel

The debug panel can be resized by dragging the resize border along the top. This allows you to pull it out to see more information, or collapse it down to just its tools and a summary of the performance information.

The debugger can also be popped out in to its own window. This is especially useful on multi-monitor setups. The project will show using the full browser window, and a separate browser window displays the debug panel. Click the **pop-out button** in the top-right of the debugger panel to do this. Clicking it again, or closing the popup window, will restore the debugger panel to the main browser window again.

> *You may see a permission prompt when clicking the pop-out button. Granting permission will allow Construct to remember the position of the pop-out window across displays if you have a multi-monitor setup.*

## Main debugger commands

Alongside the pop-out button are some other useful tools. They are as follows:

- **Pause:** pause the project so it is no longer progressing. This is useful to spend a while inspecting some information at a particular moment. When paused it turns in to a **Resume** button; click it again to resume running.
- **Step** can only be used when paused. It advances the project by a single frame. *Delta-time* (dt) is set as if the project were running at 60 FPS. This can be useful to inspect a moment frame-by-frame and watch how an event like a collision is handled.
- **Save** and **Load** make a temporary save, allowing you to quickly save the state of the project and then restore back to that state at any time later on. This can be useful for repeatedly running the same part of a project over and over again. The state is stored to the current browser's local storage. The save will not be available in a different browser, but will be available in the same browser even after closing and reopening it, rebooting, etc.
- **Take screenshot** will download a screenshot of the main project view, providing a useful tool to capture images of your project.
- **Restart** will refresh the project, loading it from scratch again.

## Performance summary

Some details about the performance of the project appear in the debugger's main title bar, and in the *Inspect* tab area for the System object, which is displayed initially. For more advice on performance, see Performance Tips. Note that since the debugger displays and manages a lot of information, it can have a significant performance overhead itself; when measuring performance, it's best to switch to one of the Profiler tabs Paid plans only, or use the normal preview mode and display performance measurements with objects. The performance information the debugger displays includes the following:

- **The object count** (e.g. *500 objects*): how many objects are currently created. Using too many objects can degrade performance. This value corresponds to the *objectcount* system expression.
- **The framerate** (e.g. *60 FPS*): how many frames per second the project is rendering. The most common display refresh rate is 60 Hz, so typically an efficiently designed project will render at 60 FPS. Note however if nothing is changing on-screen, then nothing is rendered, and so the framerate may fall to 0 or display a lower result; this does not indicate poor performance, only that fewer frames are necessary to render. The *Ticks per second* measurement in the System performance section of the inspector shows how frequently the engine is stepping, which may be different to the frames rendered per second. This value corresponds to the *fps* system expression.
- **The estimated CPU time** (e.g. *20% CPU*): an estimate of how much CPU time is being spent in the logic of the project. This is not always accurate, especially since it only takes in to account time spent on the main JavaScript thread, and should only be considered a ballpark figure. The profiler Paid plans only can break this down in to how much time is being spent in each area of the project, and is described in more detail later on in this guide. This value corresponds to the *CPUUtilisation* system expression.
- **The estimated GPU time** (e.g. *20% GPU*): an estimate of how much GPU time is being spent in the rendering of the project. This is also an estimate based on hardware timers in the GPU. This value corresponds to the *GPUUtilisation* system expression.
- **The estimated image memory** use (e.g. *32.5mb images*): an estimate of how much memory is being used by the currently loaded images in the project. Images typically use up the most memory in a project, but note this value excludes everything else, such as memory required to run the logic of the project or to play music and sound effects. See the guide on Memory usage for more information. This value corresponds to the *ImageMemoryUsage* system expression.

Some additional performance details appear in the **Performance** section of the System object's inspector view, which is displayed by default:

- **Collision checks/sec** (e.g. *1144 (~22 / tick)*): how many times in the last second the engine had to test for a collision between two objects. Collision checks are invoked by the *On collision* or *Is overlapping sprite* conditions, and many behaviors perform additional collision checks automatically. In brackets, the average checks per tick is also shown. For example if there were 600 collision checks in the last second and the framerate is 60 FPS, the estimated checks per tick will be 10. This tells you on average there were about ten collision checks per frame, although the actual value will often vary frame-by-frame.
- **Poly checks/sec** (e.g. *60 (~1 / tick)*): most collision checks are very fast, and the engine can tell trivially that two objects are not overlapping (by verifying that their bounding boxes do not overlap). However if two object's bounding boxes are overlapping, the engine must do a more expensive check where the collision polygons of each object are tested against each other. This value tells how many checks of this kind were made in the last second, as well as with the average per tick as with the *Collision checks/sec* value. Usually the *Poly checks/sec* value is considerably smaller, but if it is high, it indicates a possible performance problem.

# Breakpoints

When running the debugger, it's possible to set *breakpoints* to pause execution of an event sheet on a specific event, condition or action. For more information, see the manual entry on breakpoints.

# THE DEBUGGER 'INSPECT' TAB

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/debugger/inspect-tab

The **Inspect** tab is used to view and edit values in the project, such as an object's position.

The Inspect view is divided in to two sections. On the left appears a list of all the object types in the project, including the System object. On the right appears a list of tables of values relating to the selected object, similar to the Properties Bar.

## The object list

**Click the name of an object type** in the object list to expand it. The number of instances of that object type appears in brackets after the object name. If the object type only has one instance, or is a global object like the System object or Audio object, it will immediately start inspecting the object. Otherwise it expands a dropdown with a list of all the object instances sorted by their *index ID* (IID). Clicking a particular instance will then inspect just that instance.

There are two ways to filter down the object list:

1. Type in to the search box underneath the object list to instantly filter down the list to only show objects matching the entered search term.

   > *You can also enter a number in the search box to inspect the instance with the corresponding Unique ID (UID).*

2. Tick *Hide unused* underneath the object list to remove any objects from the list with zero instances. This will still list project-wide plugins like the Keyboard object.

The System object is always shown at the top of the object list, regardless of any filtering options.

## The values view

As with the Properties Bar, the values view shows a categorised list of tables displaying all the information about the currently inspected object. The displayed values depend on what is being inspected; for example the Sprite object displays information about its current animation frame, the Audio object displays information about currently playing sound and music, and the System object displays information about the engine, layout, and layers. Most often these values correspond to object properties from the Properties bar, as well as the object's expressions.

Instance variables and behavior values are also shown if the selected object has any.

It is often useful to view these values while running the project. However, they can also be edited. As with the Properties Bar, simply click on a value and type in a new value to change it. Edited

values appear in bold. This can be an excellent way to experiment with how your project works. Note that not all values are editable - those with a light grey background are read-only.

The **eye icon** to the right of values can be used to add the value to the Watch tab Paid plans only. This is described in more detail later on.

The **Tools** section gives you the ability to destroy the inspected object. By default the inspected object is also outlined with a dotted rectangle in the project to help identify it. If this is distracting, uncheck the *Highlight* checkbox.

Objects in a container also provide a list of links to inspect the other instances in the container with the currently inspected object. Objects in a hierarchy also provide a list of links to inspect the parent or child instances in the hierarchy relative to the currently inspected object.

# THE DEBUGGER 'WATCH' TAB

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/debugger/watch-tab

Paid plans only The Inspect tab only allows you to view one object at a time, and often also includes a great deal of information, much of which you may not be interested in. Clicking the **eye icon** (circled in red below) beside values in the Inspect tab adds that single value to the Watch tab. This allows you to combine different values from different objects in to a single space, as well as reduce the displayed values to just the ones you're interested in.




In the Watch tab, values can be edited as with the Inspect tab.

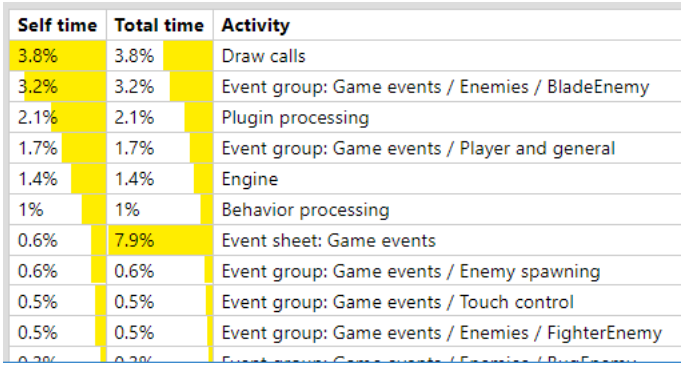
The Watch tab shows crosses instead of eyes to the right of the values. Click the cross to remove the value from the watch. Note if you are watching a value from an object and the object is destroyed, the watch value will automatically be removed.

Category headers also have their own eye or cross icons. Clicking them will add or remove the entire section to or from the watch.

# THE DEBUGGER 'CPU PROFILER' TAB

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/debugger/profile-tab

Paid plans only The **CPU profiler** tab provides a more detailed breakdown of the estimated CPU usage. The project must be running continuously for the profiler to be able to collect and display information. It then displays a breakdown of the estimated CPU time spent in each part of the project logic. It updates once a second and the values shown are for the previous second only.



| Self time | Total time | Activity |
|---|---|---|
| 3.8% | 3.8% | Draw calls |
| 3.2% | 3.2% | Event group: Game events / Enemies / BladeEnemy |
| 2.1% | 2.1% | Plugin processing |
| 1.7% | 1.7% | Event group: Game events / Player and general |
| 1.4% | 1.4% | Engine |
| 1% | 1% | Behavior processing |
| 0.6% | 7.9% | Event sheet: Game events |
| 0.6% | 0.6% | Event group: Game events / Enemy spawning |
| 0.5% | 0.5% | Event group: Game events / Touch control |
| 0.5% | 0.5% | Event group: Game events / Enemies / FighterEnemy |

*The CPU profiler tab displaying some results*

It must be noted that the overall CPU usage is an estimate to begin with, and all other values are therefore estimates as well. The details shown in the profiler only relate to the main JavaScript thread, and the CPU could be busy with other tasks, such as processing audio or running pathfinding calculations. Additionally the time for the GPU to render the project is not taken in to account at all by the profiler (and is instead covered in the GPU profiler tab).

> *CPU measurements can be unreliable, especially when the system is largely idle. Most modern devices deliberately slow down the CPU if not fully loaded in order to save power. This means work takes longer to get done, and these measurements will misleadingly return a higher measurement, since it's based on timing how long the work takes. It will generally only be reliable in the device's maximum performance mode, i.e. under full load.*

Despite the above caveats, the profiler can be used to identify "hot spots" which would be good candidates to attempt to optimise first if there is a performance problem. For more performance advice, see Performance Tips. Note that optimisation is often not necessary and is a waste of time if the project is already running fast enough. For a deeper discussion of the subject, see the blog post Optimisation: don't waste your time.

## Profiler breakdown

The profiler shows a table identifying how much CPU time has been spent in each part of the engine, down to individual event groups. It shows both the **Self time**, which is the time spent in

just that item, as well as the **Total time**, which is the self time plus the time for any sub-items. The total time is mainly applicable for events, since it shows how much time was spent in that item *and* all its sub-items. For example an event group's self time is the time spent processing the group excluding any sub-groups, and its total time is the time spent processing the group including any sub-groups. By default the table is sorted showing the highest self time at the top, which is normally the best way to identify what needs to be optimised. However you can click the table headers to sort by total time instead.

The top-level items are:

- **Events:** a breakdown of how much time was spent running event logic in the event sheets used by the layout. This is first broken down in to each event sheet (in case includes were used), and then further down in to groups and nested groups of events. This can help identify the most CPU-intensive events which you may want to optimise. **Note:** this category includes time spent running scripts in events.
- **Triggers:** some triggers, like *On mouse clicked*, run outside the normal event processing that happens every tick. These are not covered in the *Events* section, so are included under this item instead. **Note:** this category includes time spent running scripts in triggered events.
- **Scripts:** how much time was spent running scripts in your project. This only covers script files - scripts in events are measured under the *Events* and *Triggers* categories. **Note:** only the time spent in synchronous event callbacks from the engine can be counted. Construct cannot definitively attribute other time spent running your scripts to this category, such as asynchronous code or callbacks outside of the Construct engine. For example code that runs synchronously in the `"tick"` event is counted here. However code after an `await`, or in a `setTimeout` callback, is not attributed to this category, because Construct is unable to attribute the time spent running script in those cases. It will either be counted under the *Engine / Other* category, or may not be counted by the CPU profiler at all. If you use this type of code heavily, rely on the profiler in browser's developer tools instead.
- **Plugin processing:** how much time was spent updating plugins in the engine. Many plugins require a small amount of work to update them every tick, such as for Sprite to advance animations. If there are a large number of instances, this amount of work can become significant. The CPU profiler can also break down this time per plugin to help identify which objects might be contributing most to the plugin processing time.
- **Behavior processing:** how much time was spent updating behaviors in the engine. Many behaviors require some work every tick to process movement, collisions and so on. If there are a large number of instances, this amount of work can become significant. The CPU profiler can also break down this time per behavior to help identify which objects might be contributing most to the behavior processing time. In particular the Physics behavior is often CPU-intensive as it must run a physics simulation.
- **Tweens / Timelines:** how much time was spent updating currently running tweens and timelines in the project. This is not normally significant unless you have very large numbers of instances running tweens or timelines.

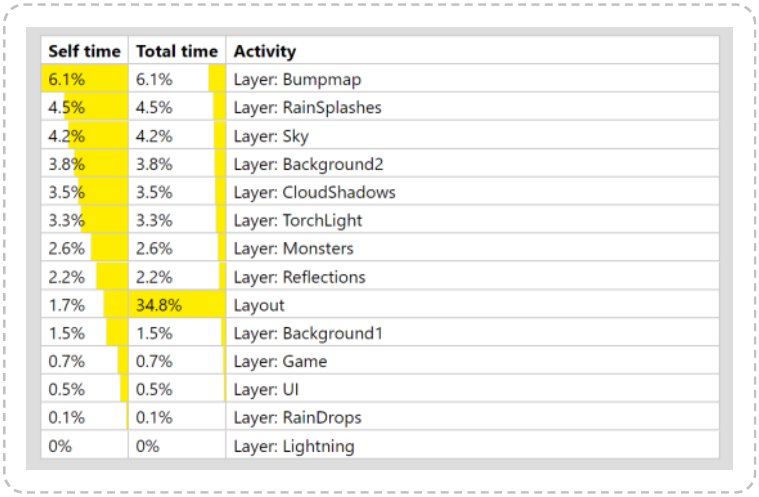
> *This measurement also includes event sheet triggers for tweens and timelines, such as the Tween On finished trigger. This means this measurement may have some overlap with the Triggers measurement.*

- **Draw calls:** how long it took the CPU to issue rendering calls, *not including* the time for the GPU to complete them. In some cases, rendering calls can be quite CPU intensive, especially when very large numbers of objects are on-screen. Some browsers also forward all draw calls to another thread to be processed in parallel, in which case the *Draw calls* measurement will likely be an underestimate.
- **Engine / Other:** the remaining time spent in Construct's runtime engine, which is the overall estimated CPU with the events, scripts, plugin/behavior processing, and draw calls times subtracted away. This covers the general runtime overhead. It sometimes also includes time spent running scripts in your project - see the *Scripts* category description for more details.

# THE DEBUGGER 'GPU PROFILER' TAB

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/debugger/gpu-profile-tab

Paid plans only The **GPU profiler** tab provides a more detailed breakdown of the estimated GPU usage. This covers work done to render the project's graphics, which is typically done on separate hardware (the Graphics Processing Unit, or GPU). The project must be running continuously for the profiler to be able to collect and display information. It then displays a breakdown of the estimated GPU time spent on each layer. It updates once a second and the values shown are for the previous second only.



| Self time | Total time | Activity |
|---|---|---|
| 6.1% | 6.1% | Layer: Bumpmap |
| 4.5% | 4.5% | Layer: RainSplashes |
| 4.2% | 4.2% | Layer: Sky |
| 3.8% | 3.8% | Layer: Background2 |
| 3.5% | 3.5% | Layer: CloudShadows |
| 3.3% | 3.3% | Layer: TorchLight |
| 2.6% | 2.6% | Layer: Monsters |
| 2.2% | 2.2% | Layer: Reflections |
| 1.7% | 34.8% | Layout |
| 1.5% | 1.5% | Layer: Background1 |
| 0.7% | 0.7% | Layer: Game |
| 0.5% | 0.5% | Layer: UI |
| 0.1% | 0.1% | Layer: RainDrops |
| 0% | 0% | Layer: Lightning |

It must be noted that the overall GPU usage is an estimate to begin with, and all other values are therefore estimates as well. However it is usually sufficient to identify which layers are responsible if the GPU usage is high.

> *GPU measurements can be unreliable, especially when the system is largely idle. Modern devices can deliberately slow down the GPU if not fully loaded in order to save power. This means work takes longer to get done, and these measurements will misleadingly return a higher measurement, since it's based on timing how long the work takes. It will generally only be reliable in the device's maximum performance mode, i.e. under full load.*

The measurements are based on the time it takes for the GPU hardware to do the rendering work. It should be noted that these are hardware measurements, and do not involve software. Therefore high GPU measurements are not usually the consequence of any particular software or technology, and will be similar across different tools that send the same work to the GPU. A high GPU measurement indicates that the capabilities of the hardware have been reached; the solution is to adjust the design of the project to require less rendering work, such as fewer objects, fewer layers, reduced use of effects, and so on.

## Profiler breakdown

The GPU profiler works similarly to the CPU profiler, but it displays the approximate GPU time required to render each layer. It shows both the **Self time**, which is the time spent to render that layer alone, as well as the **Total time**, which is the self time plus the time spent rendering any sub-layers. There is also a separate item for the layout itself, whose total time includes all layers, and includes the time to process any layout effects, as well as any layout-level compositing that is required (such as stretching the final image larger in low-quality fullscreen mode).

*Layers that use their own texture will always require more GPU time to render, since they require an additional step of copying the entire layer texture to the display afterwards. When using a large viewport in high-quality fullscreen mode, this can consume a lot of GPU bandwidth (also known as fillrate).*

# FILE EDITORS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/file-editors

Paid plans only Some kinds of project files added to the Project Bar can be edited directly in Construct using one of the following file editors. See the section on each editor for more information.

- Array files (stored in JSON format) for the Array object can be viewed and edited with the Array editor.
- Dictionary files (stored in JSON format) for the Dictionary object can be viewed and edited with the Dictionary editor.
- Any other text-based file can be viewed and edited with the Text editor.

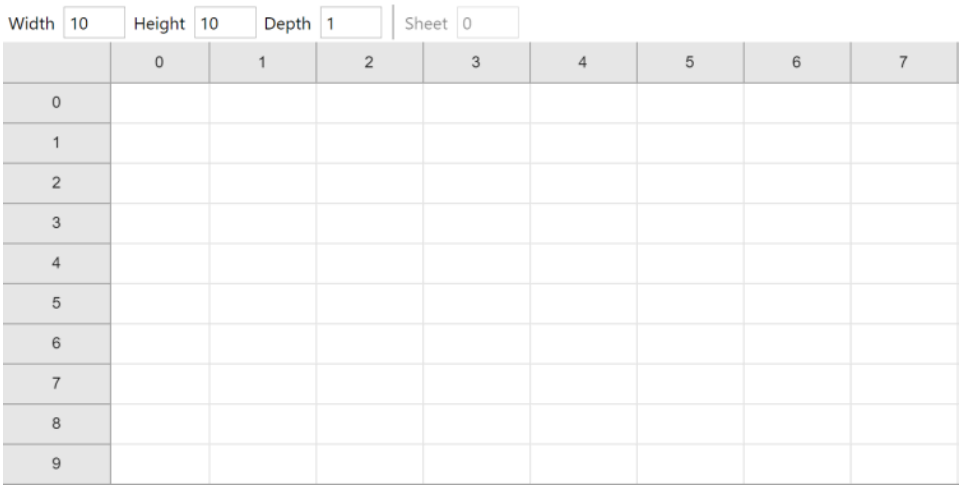
## Opening the file editors

To open a file editor, start by adding a new file in the *Files* folder of the Project Bar. For more information, see project files.

# ARRAY EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/file-editors/array-editor

Paid plans only The **Array editor** allows editing an array data file for the Array object. The data you enter can be loaded at runtime by loading the project file in to the Array object. It provides a visual way to set the initial data for an Array. The Array Editor appears when editing or adding an array data file (in JSON format) in the Project Bar.




*The Array Editor*

## Opening the Array Editor

To open the Array Editor in a new project, start by adding a new *Array* file in the *Files* folder of the Project Bar. For more information, see project files.

## Editing arrays

Initially the array is sized to 1 x 1, which means there is just one cell available. Use the *Width* and *Height* settings to change how many cells are available. This determines how many rows and columns appear in the editor, allowing you to enter more data.

Enter values in cells simply by typing in them. You can also navigate between cells using `Tab` or `Ctrl` + **Arrow keys** on the keyboard. Note that the type of cells are determined automatically: if you enter a number (e.g. *4.2*), the value is set to a number type, otherwise it is saved as a text string.

**Right-click** a cell to open a context menu with options to clear, insert or delete rows and columns.

## Editing 3D arrays

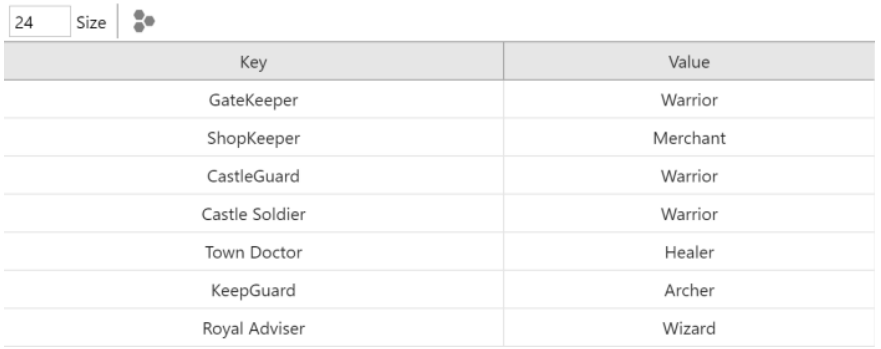
The Array Editor only displays a 2D grid of numbers, like a spreadsheet. However you can create a 3D array by setting the *Depth* greater than 1. For example if the width, height and depth are all 10, then there are 1000 elements in a 10 x 10 x 10 array.

To allow editing 3D arrays conveniently, you simply set which Z index you are editing, and edit values in that 2D plane of the array. Use the *Sheet* setting to move between each 2D "sheet" of the 3D array and edit them separately. This allows you to set all the values in a 3D array, while only editing a 2D section at a time.

# DICTIONARY EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/file-editors/dictionary-editor

Paid plans only The **Dictionary editor** allows editing a dictionary data file for the Dictionary object. The data you enter can be loaded at runtime by loading the project file in to the Dictionary object. It provides a visual way to set the initial data for a Dictionary. The Dictionary Editor appears when editing or adding an dictionary data file (in JSON format) in the Project Bar.




*The Dictionary Editor*

## Opening the Dictionary Editor

To open the Dictionary Editor in a new project, start by adding a new *Dictionary* file in the *Files* folder of the Project Bar. For more information, see project files.

## Editing dictionaries

Initially the dictionary has a single item, which means there is just one row available. Use the *Size* setting to change how many items are available. This determines how many rows appear in the editor, allowing you to enter more data. Each row represents a key in the dictionary (in the left column) and its associated value (in the right column).

Enter values in cells simply by typing in them. You can also navigate between cells using `Tab` or `Ctrl` + **Arrow keys** on the keyboard. Note that keys are always stored as strings, and the type of values are determined automatically: if you enter a number (e.g. *4.2*), the value is set to a number type, otherwise it is saved as a text string.

**Right-click** a cell to open a context menu with options to clear, insert or delete rows.

# De-duplicating rows

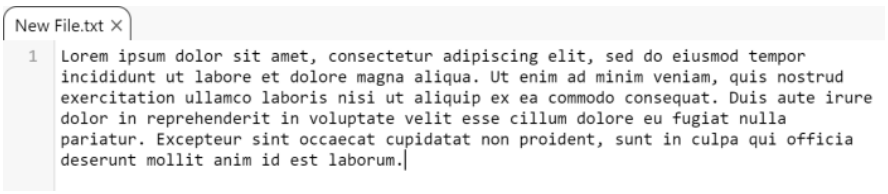
Dictionaries associate a value with a key. Due to the nature of dictionaries, there cannot be two identical keys, since there can only be one value per key. If you accidentally enter two identical

keys, they will both highlight in red. You can click the *Automatically deduplicate keys* button to add numerical suffixes to any duplicated keys to automatically fix this problem.

# TEXT EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/file-editors/text-editor

Paid plans only The **Text editor** allows editing text-based files in the project. These can be in a range of formats, such as plain text, comma-separated values (CSV), or bundled JSON, XML, HTML, CSS or JavaScript files. The data you enter can be loaded at runtime by loading the project file. The Text Editor appears when editing or adding any text-based file in the Project Bar.




*The Text Editor*

## Opening the Text Editor

To open the Text Editor in a new project, start by adding a text-based file in the *Files* folder of the Project Bar. For more information, see project files.

## Using the Text Editor

Using the Text Editor to edit long pieces of text/data in a project file is often a lot more convenient than trying to paste large amounts of text data in to events or in to Text objects.

The Text Editor provides a number of features including:

- Line numbering
- Syntax highlighting for a range of formats
- Code folding (collapsing code sections)
- Find/replace/replace all, including with regular expressions
- New file templates for formats like HTML

# ADD CONDITION/ACTION DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/add-condition-action

The **Add Condition dialog** and **Add Action dialog** are very similar so they are both documented here. These dialogs allow you to add or edit a condition or action in an event sheet.

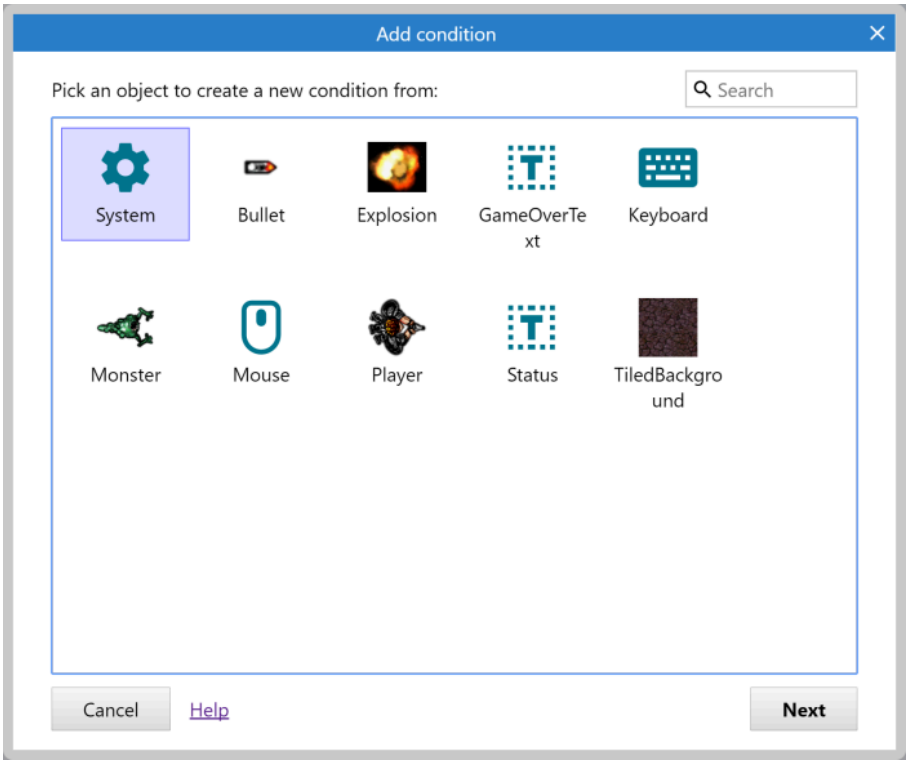
Adding a condition or action takes three steps:

1. **Choose the object type** that has the condition or action
2. **Choose the condition or action** in that object type
3. **Enter parameters**, if any, such as the X and Y co-ordinate for *Set Position*.

The **Next** and **Back** buttons can be used to move forwards and backwards through these steps.

## Step 1: Choose the object

In this step a list of all the object types in the project is shown. The **System object** (which represents built-in features) always appears first, and the rest of the objects are listed in alphabetical order. If object types are arranged in to subfolders Paid plans only in the Project Bar, then folders also appear in this dialog.

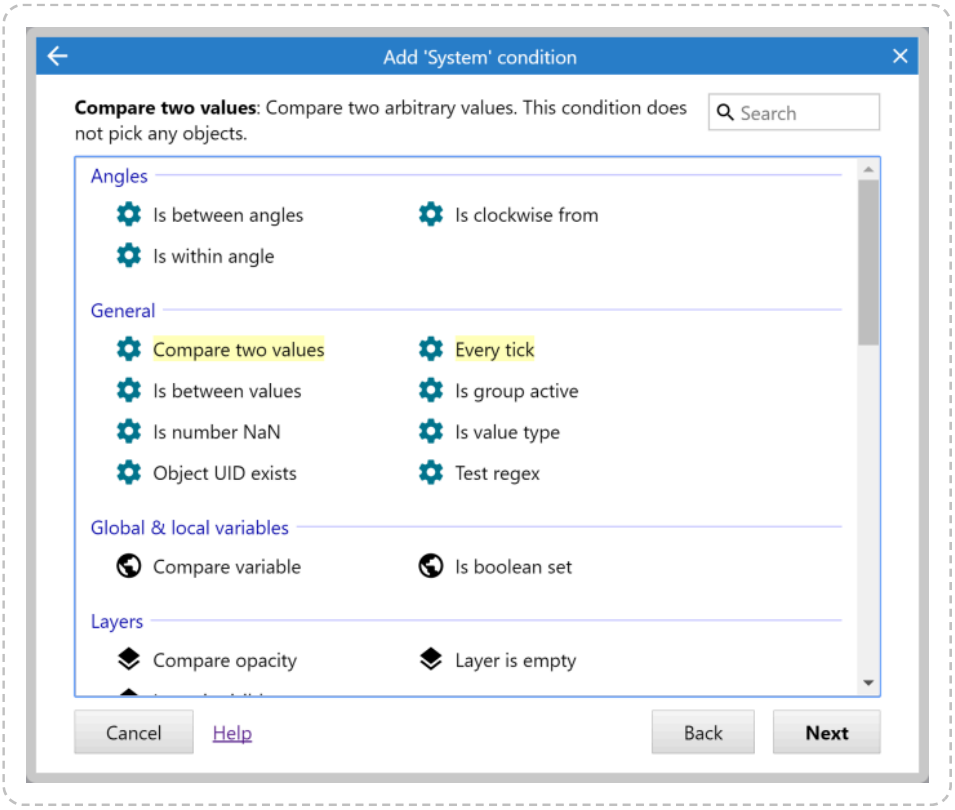



**Double-click** an object to choose it, or a folder to open it. Typing in the **search box** in the top-right can quickly filter the list if there are many objects. Searching lists results from all folders.

The conditions and actions each object contains is related to the kind of object it is (e.g. Sprite, Keyboard or Audio). For example, to locate the *Play sound* action in the Add Action dialog, first double-click the Audio object. Remember the System object contains some conditions which can be used for any object, such as *Pick random* and *For Each*.

## Step 2: Choose the condition or action

In this step a list of all the **conditions** or **actions** available for the chosen object is displayed. They are arranged in to related categories. Below a list of the System object's conditions is shown.




Some conditions and actions are very commonly used, such as *Start of layout* or *On collision*. To help you identify these quickly, commonly used conditions and actions are highlighted with a yellow background.

**Double-click** a condition or action to choose it. As before, typing in the **search box** can help quickly locate a condition or action in the list.

For more information on each condition or action, see the reference section of the manual. If the chosen object has any behaviors, they may add extra conditions and actions in to the dialog as well.

Some conditions and actions require **parameters**. For example, the *Set position* action for a Sprite requires the X and Y co-ordinates to be entered. This is done in the Parameters dialog in the next step. However, some conditions and actions do not use any parameters, such as the *Destroy* action. In this case once the condition or action is chosen the process is complete.

Functions and custom actions both provide ways to add your own items to the list of available actions.

## Step 3: Enter parameters

If the chosen condition or action requires parameters, the **Parameters dialog** appears for the parameters to be entered in to. For more information, see the manual entry for the Parameters dialog.

# THE ADDON MANAGER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/addon-manager

The **Addon Manager** shows a detailed list of all addons (plugins, behaviors, effects and themes) available in Construct 3. It can be opened via Menu▶View▶Addon Manager.

By default built-in addons are included in the list. Third-party addons appear at the top so you can easily see what you have installed. You can also uncheck *Show built-in* in the toolbar to reduce the list to only your installed addons. The search field also allows filtering the list to help you find an addon.

You can find a list of third-party addons on the Addon section of the website, including some additional official addons such as Steamworks for Steam integration. See Installing third-party addons for more information about how to install and uninstall additional addons.

The Addon Manager can also identify if updates for any installed addons are available. This is enabled by default according to the *Check for updates for installed addons* setting in the Settings dialog. If any updates are detected, it will show a notification when the editor starts up. Addons with updates available can be found by checking *Show updates only*. Addons can be updated individually by right-clicking on them, or you can install all available updates with the *Update all* button.

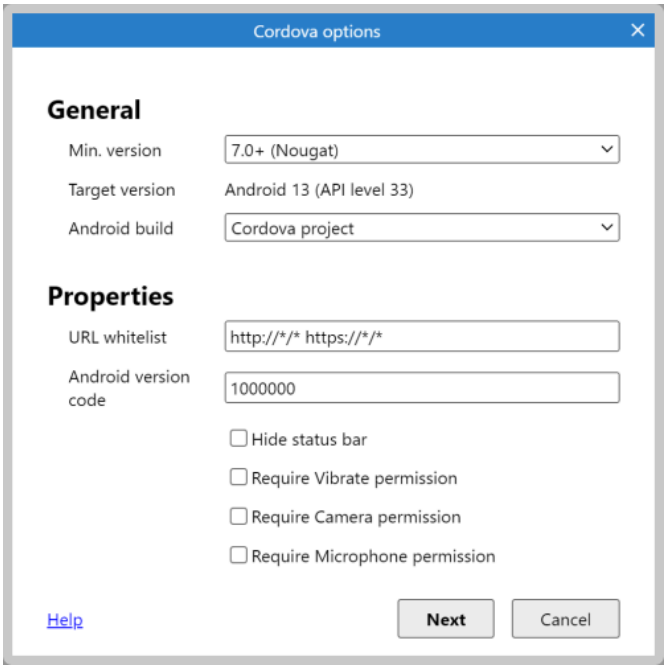
Third-party addons can be bundled with projects Paid plans only, so you don't need to install all the addons it uses when moving to a different system. For more information, see the *Bundle addons* project property.

# CORDOVA OPTIONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/cordova-options

Paid plans only The **Cordova options** dialog is used to change settings for an **Android** or **iOS** export, both of which are built with Cordova.

For more information, also refer to the tutorial How to publish mobile apps.




*The Cordova options dialog*

The dialog shown is for an Android export, but the appearance is similar for an iOS export. The following options are available.

### Min. version

Choose the minimum Android or iOS version that the app will support. A lower minimum version will make your app available on a greater number of devices on the market. On the other hand a higher minimum version ensures your app runs on more modern systems with better features and performance, and often correlates with higher-spec hardware.

### Target version

(Android exports only) This field displays the *Target API level* that Construct has configured for your app, displayed as both the Android API level and the corresponding Android version number. This is different to the minimum version and is set by Construct so is not changeable. It is usually updated annually when the Google Play publishing requirements

specify a new required target API level. If you need an updated target API level, make sure you are using the latest version of Construct, or check the latest beta release.

---

**Android build**

(Android exports only) Choose what to export. For more details see the tutorial How to publish mobile apps.

- **Cordova project:** export a Cordova project which can be built locally with the Cordova CLI, or used with a different build service.

- **Android Studio project:** use Cordova to generate an Android Studio project, which can then be opened in Android Studio for customisation, testing and completing an Android build.

- **Debug APK:** build a test Android application (APK) via the Scirra Mobile App Build Service. Debug APKs are intended for testing only. Normally to install a debug APK on an Android device requires adjusting the system settings to enable a special developer mode. Debug APKs cannot be published to the Google Play Store.

- **Unsigned release APK:** build a release Android application (APK) via the Scirra Mobile App Build Service. Release APKs are intended for publishing to the Google Play store and must first be signed before they can be published. Additionally you cannot normally install a release APK to a device unless it comes from the Google Play store. If you simply wish to test your app without publishing it, use a Debug APK instead.

- **Unsigned Android App Bundle:** build a release Android application, in the new Android App Bundle (ABB) format, via the Scirra Mobile Build Service. ABBs are intended for publishing to the Google Play store and must first be signed before they can be published. An ABB file cannot be directly installed onto a device. You should not use these for local testing

- **Signed Debug APK:** build and sign a test Android application (APK) via the Scirra Mobile App Build Service. While signing a debug APK is not required to install it onto a device, some services require that you sign the APK with your publishing certificate before you can test them ( such as Google Play Games ). Under most situations you will be fine to use an Unsigned Debug APK.

- **Signed Release APK:** build and sign a release Android application (APK) via the Scirra Mobile Build Service. This is intended for publishing to the Google Play store, it is already signed so can be uploaded directly to the store.

- **Signed Android App Bundle:** build and signed a release Android application, in the new Android App Bundle (ABB) format, via the Scirra Mobile Build Service. This is intended for publishing to the Google Play store, it is already signed so can be uploaded directly to the store. An ABB file cannot be directly installed onto a device.

---

**URL whitelist**

A space separated list of URLs that the app can ask the system to open. The default option `http://*/* https://*/*` allows the application to open any http or https URL.

---

**Android Version Code**

(Android exports only) Specify a Android version code for this export. The initial value is automatically calculated from the project's version number.

---

**iOS build**

(iOS exports only) Choose what to export. For more details see the tutorial How to publish mobile apps.

- **Cordova project:** export a Cordova project which can be built locally with the Cordova CLI, or used with a different build service.

- **Xcode project:** use Cordova to generate an Xcode project, which can then be opened in Xcode on a Mac for customisation, testing and completing an iOS build.

---

**Hide status bar**

When the app is running, hide the system status bar if possible. This allows the app to use up more space on the screen.

---

**Require Vibrate permission**

Enable if your app makes use of vibrating the device to ensure the app has permission to do this.

---

**Require Camera permission**

Enable if your app makes use of camera input via the User Media object to ensure the app has permission to access this.

---

**Require Microphone permission**

Enable if your app makes use of microphone input via the User Media object to ensure the app has permission to access this.

---

**Choose Keystore**

**Create Keystore**

(Signed Android builds only) Choose a keystore from the filesystem, or create a new one, for use in signing the build.

---

**Key Name**

(Signed Android builds only) The name of the key within the keystore you wish to use for signing the build. This is referred to as the "alias" by the create keystore dialog.

---

**Key Store password**

(Signed Android builds only) The password for the keystore you have selected.

---

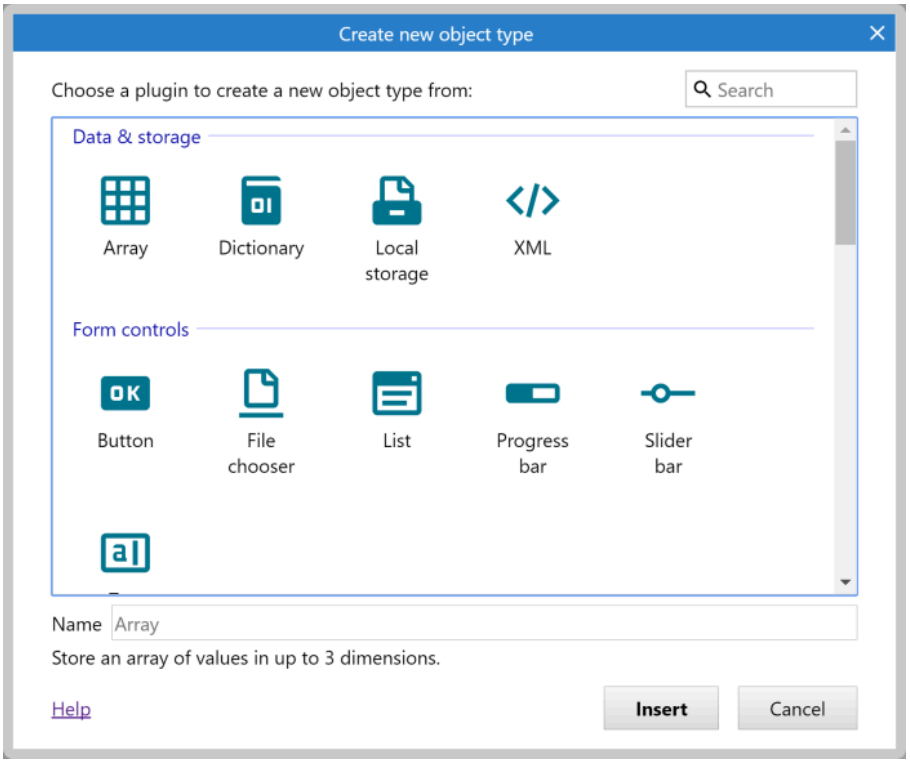
**Key password**

(Signed Android builds only) The password for the key you have chosen in the keystore.

# CREATE NEW OBJECT TYPE DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/create-new-object

The **Create New Object Type dialog** typically appears after **double-clicking** a space in a Layout View. It allows you to choose the kind of object type to add to the project. Each kind is known as a plugin. Selecting a plugin then creates a new object type based on that plugin. See Project Structure for a description of the difference between object types and instances. The plugin reference section of the manual includes documentation for each plugin.




Plugins are categorised in to related groups. Within each group they are arranged alphabetically. Selecting an item will display a brief description summarising what the plugin does at the bottom of the dialog. Typing in the **search box** in the top right can quickly filter down the list to help you find what you are looking for.

If you regularly use the same few plugins, you can **right-click** them and choose **Pin to top**, and it will be kept at the top of the dialog for quicker access. If you change your mind, you can right-click a pinned plugin and choose **Unpin** to return it to its usual location. This also works for adding behaviors and effects.

The **Name** field indicates what the name of the object type in the project will be after you add it. By default this will be a numbered version of the plugin name e.g. *Sprite2*, *Sprite3* etc. It is a good

idea to give objects descriptive names so you don't get confused when working on your project. You can enter a descriptive name in this field which the object will use when added, but if you don't you can still easily rename objects via the Properties Bar or Project Bar.

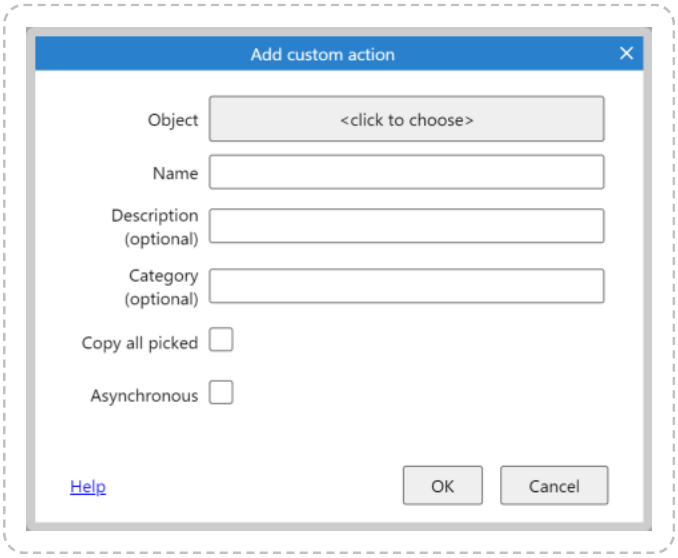
**Double-click** an item to add it to the project. If the object can be placed in a layout and a Layout View is open, **the cursor turns to a crosshair** for you to place the first instance of that object. For objects with images or animations (like Sprite), after you have placed this instance the Animations editor will appear to design the image or animations for the object.

Other kinds of object (like the Keyboard and Audio object) do not need placing in a layout. After adding one of these objects, the dialog closes but there is no need to place it anywhere. Instead, a notification appears indicating that it is available to the entire project. This type of object can only be added once, and will show as disabled in the list if the dialog is brought up again.

# ADD/EDIT CUSTOM ACTION DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/custom-action

The add/edit function dialog appears when adding or editing a custom action.




This dialog has the following fields.

### Object

Click the button to choose the object type or family the custom action will belong to.

### Name

The name of the custom action. This will appear in the Add Action dialog and in the event sheet to identify this custom action. Object types are allowed to add a custom action with the same name as a family custom action, in which case the object type's custom action works as an override.

### Description Optional

An optional description of the custom action, for your organisational purposes. This is displayed in the Add action dialog and can be a helpful reminder of what the custom action does.

### Category Optional

An optional category for the custom action, for your organisational purposes. Custom actions with the same category are grouped together in the *Add action dialog*, providing a way to arrange related custom actions together. This field autocompletes with existing category names used, making it easier to use the same category names.

### Copy all picked

By default custom actions run with only the same instances of the given object picked as the action that called it. Check *Copy all picked* to instead run the custom action with the same instances of all objects picked as the action that called it, much like the *Copy picked* setting for functions.

### Asynchronous

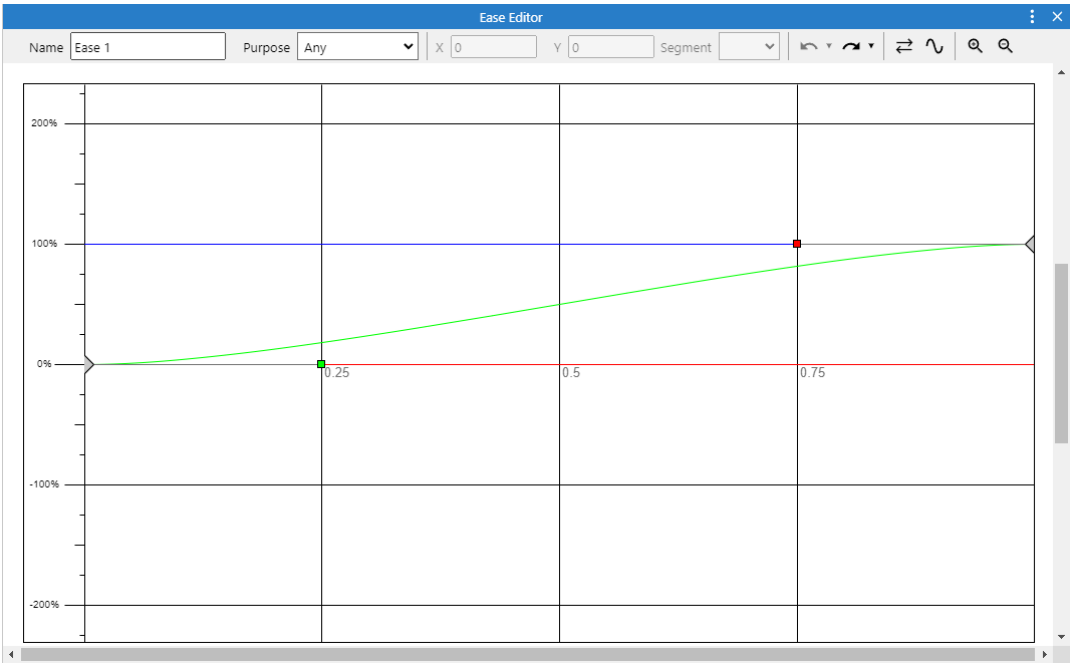
Tick to mark the custom action as *asynchronous* (or *async* for short). This means calls to this custom action can be used with the System action *Wait for previous actions to complete* if the custom action does any of its own waiting. Note this has a small performance overhead, so for best performance leave this disabled if you don't need it.

# THE EASE EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/ease-editor

This editor enables the possibility of creating custom ease functions to be used with timelines and tweens. This is a little bit of extra work when compared to the built in ease functions, but it makes it possible to customize your timeline animations and tweens even further.

The most basic use is that of a cubic curve with a beginning, end and anchor points. This allows variations on all the basic ease functions (e.g. Sinusoidal, Back etc). It is also possible to add additional points in the middle of the curve to create variations of the more complex curves (e.g. Bounce, Elastic). Of course it is also possible to create completely original curves by using multiple points.

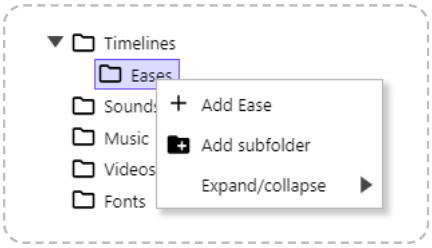



## Using the ease editor

Here is a short summary on how to create eases, use the ease editor and finally apply a custom ease.

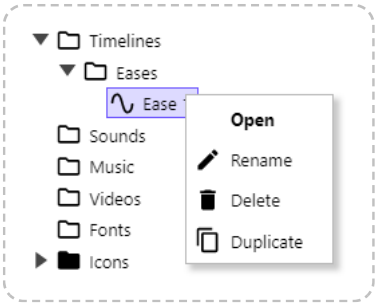
### Step 1: Create a custom ease

Right-click **Eases** folder in the Project Bar and select *Add ease*.




### Step 2: Open the editor

Right-click on the newly created ease element under the **Eases** folder, then select the *Open* option or Double-click the ease element in the Project Bar.




### Step 3: Make changes

With the ease editor open, you can make changes to the curve by dragging the colored handles. The default ease has four points, the start and end points (yellow squares) and the two anchor points (green and red squares). In any ease it is not possible to change the position of the start and end points. This means that with the default curve you will only be able to move the anchor points.

If more ease keyframes are added in the middle of the curve, it is possible to move those.

There are some limitations to the position the anchor points can take in relation to the main points, but those will be covered later.

The default ease will be something similar to the built in **In Out Sinusoidal** ease function.

### Step 4: Use the custom ease

Once a custom ease is complete, it can be used in any of the places the built in eases are available. The names of the custom eases will appear after the names of the built in ease functions in all the relevant places. These include the Properties Bar, Timeline Bar and Tween behavior.

## Interface

### Handles

The different handles are the main method to edit the ease function.

- **Main Handle:** The main points, or keyframes of the curve are represented by yellow squares. This points are positions the curve must include. More points can be added to the curve to edit how it looks like. Depending on the *"Purpose"* of the curve the starting point and ending point might not be editable.
- **Anchor Handle:** The anchor points of the curve, represented by green and red squares. Each pair of main points has two anchor points in between them. The first anchor point of a sub section of a curve will always be green, while the second anchor point will always be red. Anchor points can never go outside the range defined by the two main points that contain them. Dragging the main handles will force anchor handles out of their previous position to keep the previously mentioned condition.

## Colored Lines

These are the main visual aid to understand how the curve will behave when it is used.

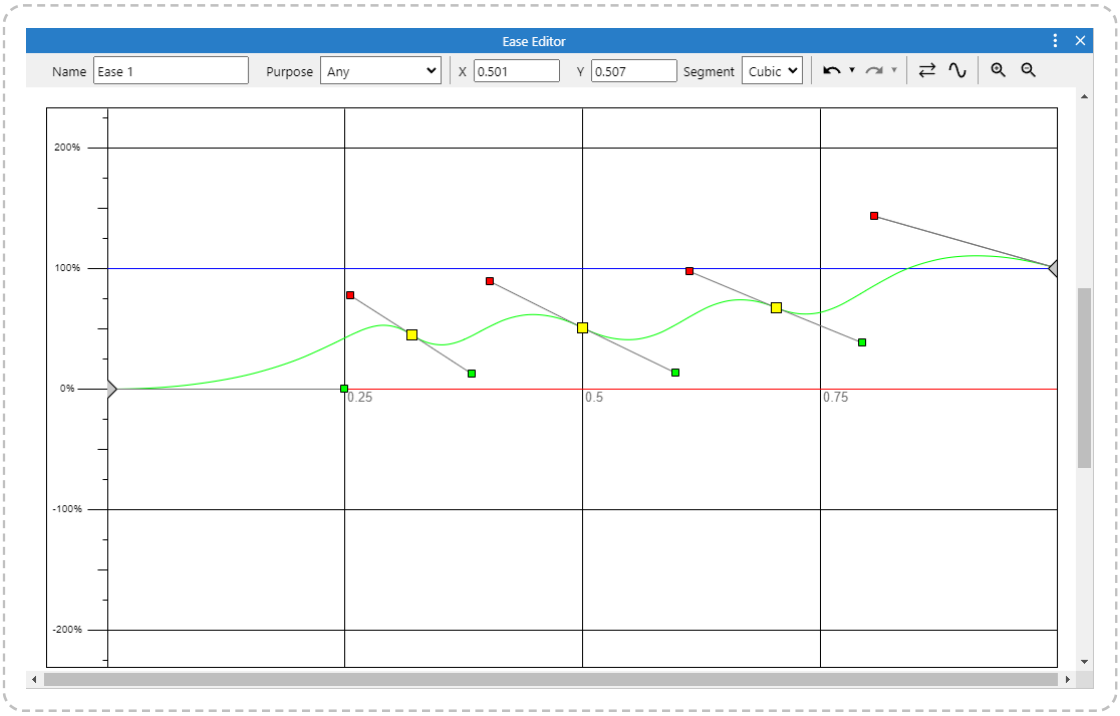
- **Green line:** The green line represents the ease function itself. The closer it is to the red line, the closer the timeline or tween using the ease will be to the initial value. Going bellow the red line means the animation using the ease will be going past the starting value, an example of this is the **In Back** built in ease. The closer it is to the blue line, the closer the timeline or tween using the ease will be to the ending value. Going above the blue line means the animation using the ease will be going past the ending value, an example of this is the **Out Back** built in ease.
- **Red line:** Represents the starting value of the ease.
- **Blue line:** Represents the ending value of the ease.

## Horizontal and vertical axis

Each axis of the graph shown by the editor has a meaning that will help you to better understand what the ease function will be doing.

- **Horizontal axis:** This axis represents time. The further along the axis, the closer the animation using the ease is to finishing. The marker numbers are there to help identify what the ease will be doing at 25%, 50% and 75% completion.
- **Vertical axis:** This represents the value the ease will produce. 0% is the starting value of the timeline or tween using the ease, while 100% represents the ending value.

The image bellow shows a custom ease, with multiple keyframes as well as their corresponding anchor points.




## Context Menu

There are two options available in the context menu.

- **Add ease keyframe:** Add a new keyframe to the curve.
- **Delete ease keyframe:** Remove an existing keyframe from the curve. The first and last keyframes can not be deleted.
- **Invert ease:** Invert all the keyframes in the ease to produce the opposite ease.
- **Toggle linear and cubic:** Toggle between a cubic ease and a linear ease.
- **Make segment linear:** Toggle the *"Segment"* property of a keyframe to make the corresponding segment a linear transition.
- **Make segment cubic:** Toggle the *"Segment"* property of a keyframe to make the corresponding segment a cubic transition.

## Toolbar




The toolbar at the top shows a few useful fields and buttons:

- **Name:** Change the name of the ease.
- **Purpose:** This property can take the values of *"Any"* or *"Expressions only"* and affects how the ease can be edited.
- **Position:** The inputs show the position of the last handle that was clicked on. Useful to make more precise adjustments that what is possible by dragging with a pointer device.

- **Segment:** This property is keyframe specific and can take the values of *"Linear"* or *"Cubic"*. It affect how the ease behaves from the point where a keyframe is until the next one.
- **Undo:** Works like elsewhere in the application.
- **Invert ease:** Invert all the keyframes in the ease to produce the opposite ease.
- **Toggle linear and cubic:** Toggle between a cubic ease and a linear ease.
- **Zoom:** Works like elsewhere in the application

## Editing built-in eases

When the ease editor is opened from the common **Ease editor** property of a Timeline element, the editor can be opened showing a built-in ease.

In this case a new custom ease is created to look just like the corresponding built-in ease, it is also given a unique name and can be edited normally.

After closing the editor, if any meaningful changes where made, the new custom ease is added to the project and assigned to the corresponding timeline element. If no meaningful change was made after the editor is closed, everything is discarded.

## The special "Default" ease value

In the case of opening the editor from a timeline element which is using the special *"Default"* value the **Ease** property can take, C3 will look up the corresponding timeline's inheritance structure for a concrete ease value to use, be it custom or a built-in one.

After closing the editor, any changes are applied to the timeline element which had the real ease, rather than the element which was using the special *"Default"* value.

## The "Purpose" property

As mentioned earlier, this property can take the following values:

---

**Any**

Can be used anywhere where custom eases are supported. Their first and last keyframes can not be edited at all, that means the ease is bounded to **start at a value of 0** and **end at a value of 1**.

---

**Expressions only**

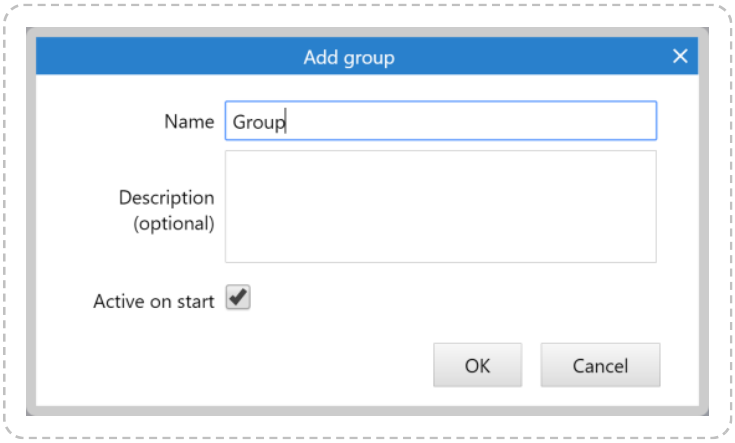
Can only be used as argument in expressions that support providing a custom ease, such as the **Ease** expression of the Timeline Controller plugin. This purpose allows to edit the first and last keyframes of the ease, allowing it to **start at values different from 0** and **end in values different from 1**.

If a custom ease is already used in a place which expects it to have the **"Any"** purpose, such as Tweens, then the property will not be able to be changed. The same goes if it is already used in a place which expects the **"Expressions only"** prupose.

# ADD/EDIT EVENT GROUP DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/event-group

The **Add/Edit Event Group dialog** contains settings for a group of events.




This dialog has the following fields.

### Name

A name identifying this event group, displayed as its title in the event sheet. When enabling or disabling event groups, the name identifies the group.

### Description Optional

An optional description summarising what the events in the group do, for your organisational purposes. This is displayed beneath the group title in the event sheet.

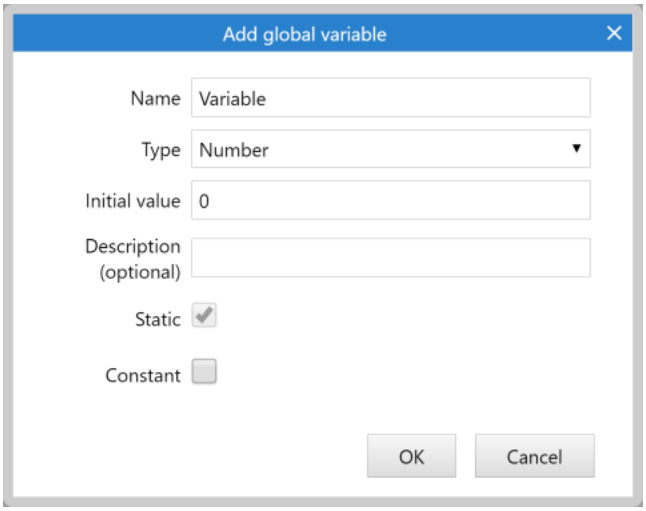
### Active on start

Determine whether or not the event group is enabled when the project begins. If this is unchecked, the event group is disabled, and none of the events inside it will run until it is enabled by the *Set group active* system action. Groups which are inactive on start are displayed with faded out text in their header.

# ADD/EDIT EVENT VARIABLE DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/event-variable

The **Add/Edit Event Variable dialog** allows you to change the details of a global or local variable in an event sheet.




The dialog has the following fields.

### Name

The name identifies the event variable. This is typed in to expressions to retrieve the value of the event variable. Some names cannot be used, like the names of system expressions, since they would conflict when entering an expression.

### Type

The type specifies what kind of value the variable holds. This can be *Number*, *String* (text), or *Boolean* (an on/off value). The type of an event variable does not change - you can't store text in a Number variable and vice versa.

> *Once you create events that use this variable, its type cannot be changed, since that could make the events invalid.*

### Initial value

The initial number for a *Number* variable, the initial text for a *String* variable, or a checkbox for a *Boolean* variable to specify if it is initially true (checked) or false (unchecked). Note that unlike expressions, the initial text entered here does **not** need double quotes around it. In

other words, *Hello* is a valid entry, and if you enter *"Hello"* (as you would in an expression) the initial string will include the double quotes.

**Description Optional**

An optional comment you can use to briefly describe what the variable is used for. It is displayed next to the name in the editor to help remind you what to use the variable for.

**Static**

Only applies to local variables. By default, local variables reset their value to the initial value every tick. However if *Static* is checked, the local variable's value will persist permanently, like a global variable. Static local variables differ from global variables in that they can still only be used within their scope. Global variables always hold their values permanently so the *Static* option does not apply to them. For more information about local variables, see Event Variables.

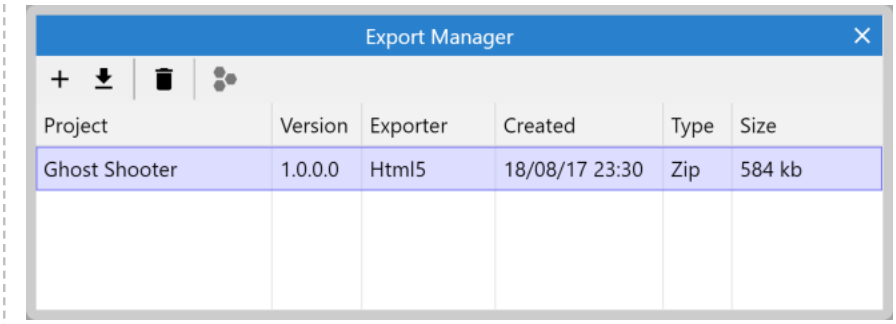
**Constant**

Make a variable read-only. You will be able to compare and retrieve the variable, but not change its value using any actions. This is useful for referring to a number like the maximum number of lives, without having to repeat the number in your events. If you want to change the value, there is only one place you need to change, which is a lot easier than having to hunt down the multiple places you entered a particular number in your events. According to programming convention, the names of constants are displayed in upper case, e.g. *MAX_LIVES*.

# THE EXPORT MANAGER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/export-manager

The **Export Manager** displays a list of the last few projects you've exported from Construct. This helps you find an exported project again if you forget to save the resulting file, or otherwise lose it. It can be opened via Menu▶View▶Export Manager. Note only the last few exports are kept and old ones are automatically deleted, so don't rely on it to keep everything - be sure to save exported files at the time you export them.




To download a previous export again, select it in the list and click the **Download** button in the toolbar. If you need to free up storage space you can also manually delete saved exports with the **Delete** button in the toolbar.

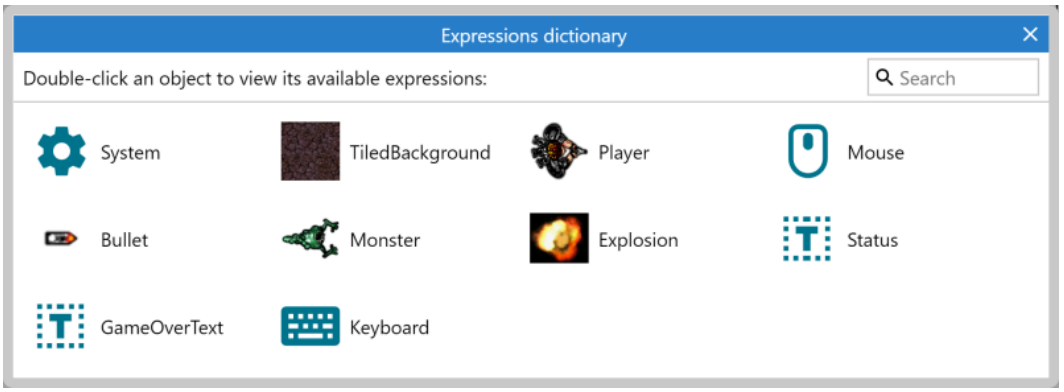
## Building Cordova exports

Paid plans only If you export for Android or iOS and select the **Cordova project** option, the result is a zipped Cordova project. You can then choose to build it by selecting it in the Export Manager and clicking the **Build** button on the toolbar. This will then take you through the steps to build a mobile app with Construct 3's build service. Note it's quicker to select one of the other build options at the time you export for Android or iOS, but this can be useful if you forget, or want to run more than one kind of build.

# THE EXPRESSIONS DICTIONARY

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/expressions-dictionary

The **Expressions Dictionary** lists all the expressions available in your project. This saves you having to remember them or look them up elsewhere. It is shown floating next to the Parameters dialog. It can be hidden or shown by clicking the **Find expressions** link on the Parameters dialog or by pressing **F4**. By default it is semitransparent so it does not distract you while entering parameters. However on small screens if there is no room to display it without overlapping the Parameters dialog, then it is hidden by default.




The **search box** in the top right can be used to quickly filter the list.

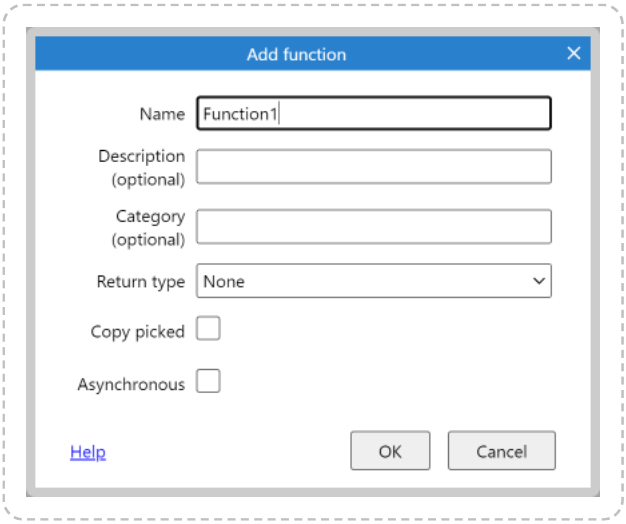
**Double-click** an object to list all its expressions. Descriptions are also shown next to each expression to help indicate what value will be retrieved. **Double-click** one of the listed expressions to insert it to the current expression in the Parameters dialog. Click the **Back** button in the caption to return to the object list.

Behavior expressions are also shown for objects with behaviors. The System page also lists all system expressions as well as all the global and local variables in scope (see Event Variables).

# ADD/EDIT FUNCTION DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/function

The add/edit function dialog appears when adding or editing a function.




This dialog has the following fields.

### Name

The name of the function. Note that if the function is used as an expression (its *Return type* is not *None*), the name must be a valid expression, so cannot contain special characters or whitespace. Functions used as actions (with a *Return type* of *None*) can use any name.

### Description Optional

An optional description of the function, for your organisational purposes. This is displayed in the Add action dialog or Expressions dictionary depending on the return type of the function, and can be a helpful reminder of what the function does.

### Category Optional

An optional category for the function, for your organisational purposes. Functions with the same category are grouped together in the *Add action dialog* and *Expressions dictionary*, providing a way to arrange related functions together. This field autocompletes with existing category names used in the project, making it easier to use the same category names.

### Return type

The return type of the function. This also determines whether the function is used as an action or an expression. Functions with a return type of *None* are used as actions; otherwise they are used as expressions. A return type of *Any* means the function can return either a

number or a string. Functions with a return type must have a name which is a valid expression, so cannot contain special characters or whitespace.

> *Once a function is used in your project, the return type cannot be changed.*

---

### Copy picked

Normally calling a function will run its actions with all objects reset to picked. For example calling a function that modifies a Sprite will modify all instances of that Sprite regardless of whether any conditions picked specific instances before calling the function. Enabling *Copy picked* means the function will run with the same picked instances as the event that calls it, so actions still run on the same instances picked by any previous conditions.

---

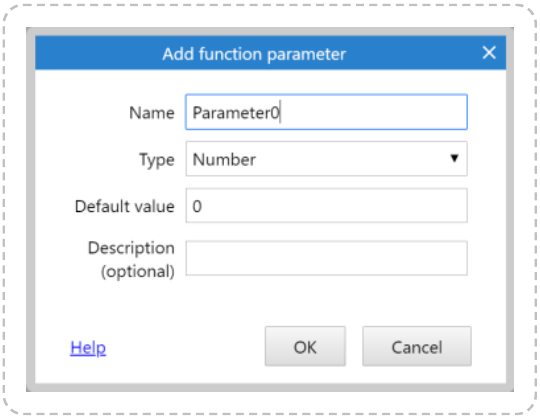
### Asynchronous

Tick to mark the function as *asynchronous* (or *async* for short). This means calls to this function can be used with the System action *Wait for previous actions to complete* if the function does any of its own waiting. Note this has a small performance overhead, so for best performance leave this disabled if you don't need it.

# ADD/EDIT FUNCTION PARAMETER DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/function-parameter

The **Add/Edit Function Parameter dialog** allows you to change the details of a function parameter. Since function parameters work similarly to local variables, the dialog is also similar to the Add/edit event variable dialog.




The dialog has the following fields.

### Name

The name identifies the function parameter. This is typed in to expressions to retrieve the value of the parameter. Some names cannot be used, like the names of system expressions, since they would conflict when entering an expression. The name is also displayed when calling the function.

### Type

The type specifies what kind of value the parameter holds. This can be *Number*, *String* (text), or *Boolean* (an on/off value, displayed as a checkbox). The type of a parameter does not change - you can't pass text for a number variable and vice versa.

> *Once you create events that call the function, its parameter types cannot be changed, since that could make the events invalid.*

### Initial value

The initial number for a *Number* parameter, the initial text for a *Text* parameter, or a checkbox for a *Boolean* variable to specify if it is initially true (checked) or false (unchecked). Note that

unlike expressions, the initial text entered here does **not** need double quotes around it. In other words, *Hello* is a valid entry, and if you enter *"Hello"* (as you would in an expression) the initial string will include the double quotes. The default parameter value is pre-filled when calling the function, or used if the function ends up being called with missing parameters.

---

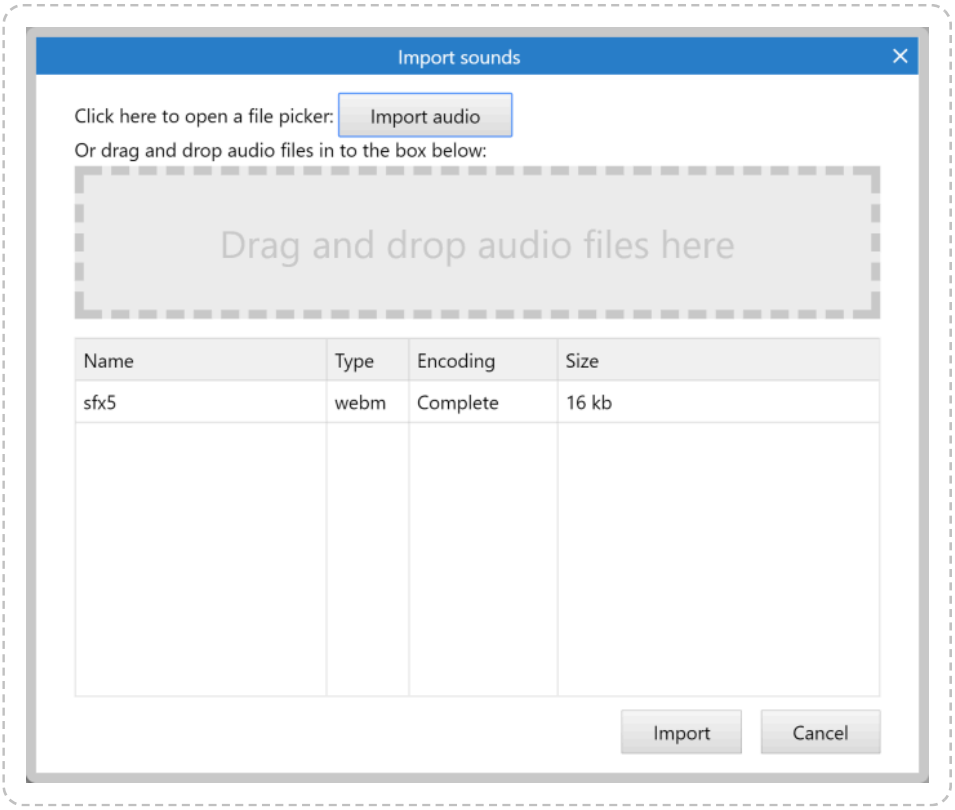
**Description Optional**

An optional comment you can use to briefly describe what the parameter is used for. It is displayed in the editor when calling the function to help remind you what to use the variable for.

# THE IMPORT AUDIO DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/import-audio

The **Import Audio dialog** allows audio files to be added to the project from disk. It is accessed by selecting Import Sounds or Import Music from the right-click menu of the *Sounds* or *Music* folders in the Project Bar. Be sure to add the Audio object to your project to play back sounds and music.




There are two ways to import audio files:

1. Click the **Import audio** button, which opens a file picker to choose some local files to import
2. **Drag-and-drop** some local files in to the area that says *Drag and drop audio files here*

As soon as you choose some files or drop them in, Construct will encode them to **WebM Opus** for best compatibility across platforms. It is recommended to import **16-bit PCM WAV** or **FLAC** files to ensure Construct can encode them without any quality loss. If you import files which are already WebM Opus, they will simply be copied to the project.

Construct will use all available CPU cores to encode imported audio, which is useful if you need to import a lot of audio files. The progress of encoding is displayed in the table of files. Once all encoding is done, you can click **Import** to add them to the project.

## Categorise audio files correctly

It is important to organise audio files appropriately. Audio files in the **Sounds** project folder are loaded completely before playing, but files in the **Music** folder are streamed. This means if a music track is accidentally put in the *Sounds* folder, it would have to load completely before it can start playing. However, audio in the *Music* folder can start playing much more quickly since it is streamed, and also uses less memory for long tracks.

## Platform compatibility

There is not one audio format that can play across all browsers and platforms with built-in support. To avoid having to use multiple audio formats, Construct uses its own WebM Opus decoder for Safari & iOS, which are the last platforms left that do not have built-in support for WebM Opus.

In most browsers Construct can transcode AAC or MP3 audio for you. However if the browser does not support decoding these formats, Construct may be unable to transcode these formats and may show a warning. It is strongly recommended to use WebM Opus for all audio in Construct projects, as it is the only format that is guaranteed to play on all platforms. You will need to use other software to encode your audio to WebM Opus, and then import the .webm files to your project. Construct will simply copy these files in to your project if they are already WebM Opus.

## Import formats

The following formats can be imported to Construct. PCM WAV or FLAC files are recommended.

- **PCM WAV** (.wav): converted to WebM Opus
- **FLAC** (.flac): converted to WebM Opus
- **WebM** (.webm): copied to project if Opus, else converted to Opus
- **MPEG-4 AAC** (.m4a): converted to WebM Opus in most browsers (note this is lossy)
- **MP3** (.mp3): converted to WebM Opus in most browsers (note this is lossy)
- **Ogg Vorbis** (.ogg): converted to WebM Opus (note this is lossy)
- **Ogg Opus** (.opus): converted to WebM Opus (note this is lossy)

If you need to import a different format, you'll need to find third-party software to convert it. If you can, convert it directly to WebM Opus, as that is the preferred format for Construct.

## Playback preference

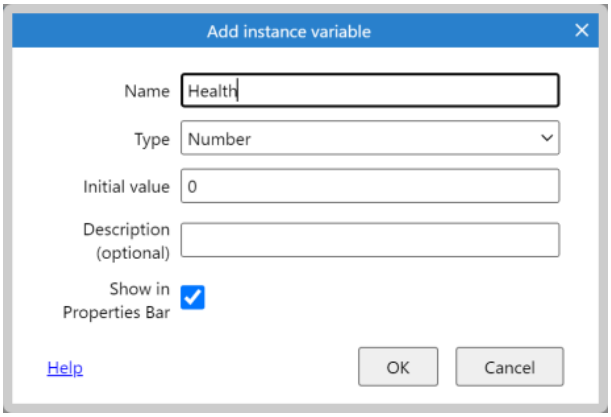
Projects only need to use WebM Opus files to support all platforms. However in some cases, especially with older projects, audio files may be available in multiple formats in the project. In that case Construct uses the following order of preference to pick which to play at runtime. The first format in the list that has built-in support is used. If none of the available formats has built-in support, and there is a WebM Opus file available, then Construct falls back to the software decoder last of all.

1. WebM Opus
2. Ogg Vorbis (used by Construct 2)
3. MPEG-4 AAC
4. MP3
5. Software decode WebM Opus

# ADD/EDIT INSTANCE VARIABLE DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/instance-variable

The **Add/Edit Instance Variable dialog** allows you to set the name, type and initial value for an instance variable in an object type or family.




The dialog has the following fields.

### Name

The name identifies the instance variable. This is typed in to expressions after the object name (e.g. `Sprite.MyVariable`) to retrieve the value of the instance variable. Some names cannot be used if they conflict with the names of the object's other expressions or behaviors.

### Type

The type specifies what kind of value the instance variable holds. This can be *Number*, *Text* (also known as a *string*), or *Boolean* (an on/off value). The type of an instance variable does not change - you can't store text in a Number variable and vice versa. Also note that once you create events that use this variable, its type cannot be changed, since that could make the events invalid.

### Initial value

The initial number for a *Number* variable, the initial text for a *Text* variable, or a checkbox for a *Boolean* variable to specify if it is initially true (checked) or false (unchecked). Note that unlike expressions, the initial text entered here does **not** need double quotes around it. In other words, *Hello* is a valid entry, and if you enter *"Hello"* (as you would in an expression) the initial string will include the double quotes.

### Description Optional

An optional comment you can use to briefly describe what the variable is used for. It is displayed in the Properties Bar description panel when the instance variable's property is selected.

### Show in Properties Bar

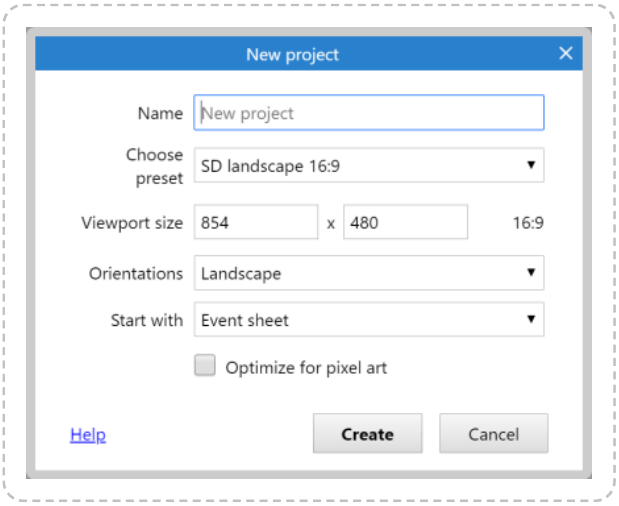
By default all instance variables are shown when listing properties in the Properties Bar. This can be unchecked to hide it from the listed properties. However it will still be available for use in event sheets. Hiding instance variables can be useful if you have a large number of them and some are only used in event sheets.

# THE NEW PROJECT DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/new-project

The **New project** dialog appears when you create a new project. It allows you to specify some basic details about the project, such as its name. New projects can be created from the Start Page or the main menu.

All values are optional so you can simply click **OK** right away to get a basic new project with default settings. Note the `Alt` + `N` keyboard shortcut creates a new project skipping this dialog, as if you clicked OK after it opened.




The dialog has the following fields.

### Name

Choose the name of the project. Construct uses this to identify your project.

### Choose preset

Select a preset from this list to fill out the rest of the fields quickly. For example choosing *1080p landscape* will automatically fill out a 1080p viewport size and set the landscape orientation.

### Viewport size

Set the size, in pixels, of the view area in the game. This corresponds to the *Viewport size* project property. The viewport size also defines the aspect ratio of the project, which is displayed to the right.

### 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, where supported. This corresponds to the *Orientations* project property.

---

**Start with**

Choose the type of project to start with. *Event sheet* starts with an empty event sheet for using Construct's block-based approach. *Script* instead starts with a template JavaScript file for coding your game instead. See the Scripting overview for more information about the scripting feature in Construct. You can easily switch between the two after creating a project by adding a new event sheet or a new script file.

---

**Optimize for pixel art**

Check to apply settings that are more suitable for retro-style graphics. The following settings are applied:

- **Pixel rounding** is enabled
- **Letterbox integer scale** fullscreen mode
- **Low** fullscreen quality
- **Nearest** sampling

These settings can be changed back any time after creating the project. For more information on each, see project properties.

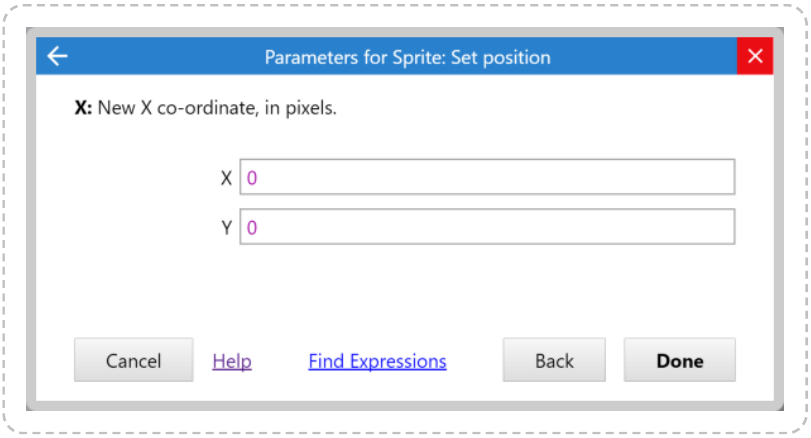
# THE PARAMETERS DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/parameters

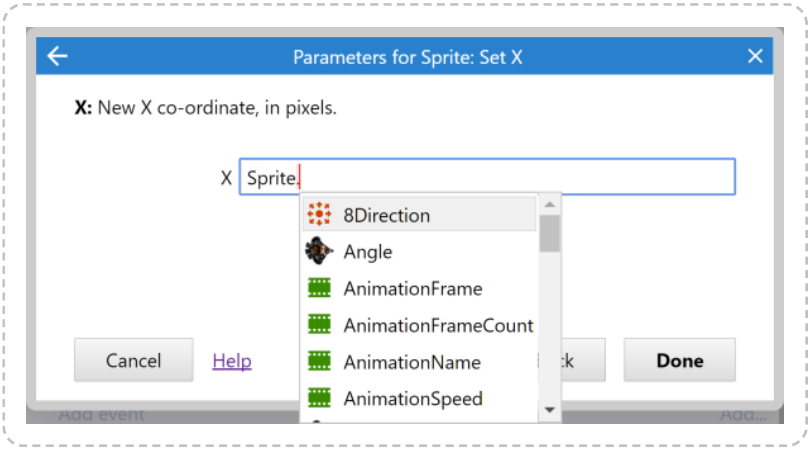
The **Parameters dialog** appears after the Add condition/action dialog, or when editing a condition or action. It allows you to enter *parameters* such as the X and Y co-ordinate for the Sprite object's *Set Position* action (shown below). It does not appear for conditions or actions which do not use any parameters, such as the *Destroy* action. Expressions can be entered for many parameters, which can be anything from a simple value to a sophisticated mathematical calculation.

The parameters that are displayed depend on the condition or action that was selected. To find out more about what to enter for each field, check the related documentation in the Plugin reference, Behavior reference or System reference.

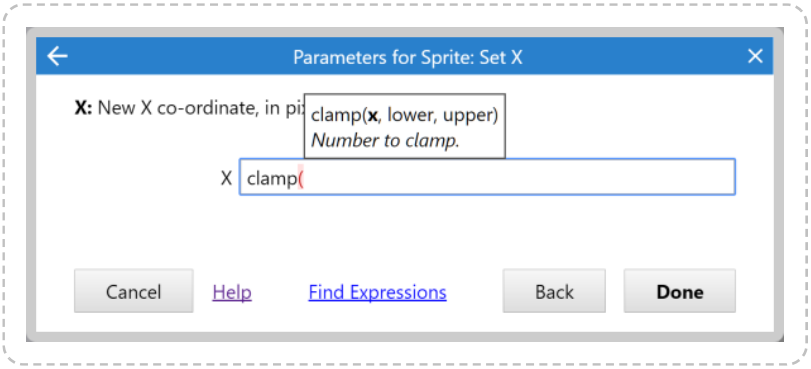
The Expressions dictionary appears next to the Parameters dialog providing a list of all the expressions you can enter. By default the Expressions panel is faded out unless you hover the mouse over it. It can also be hidden or shown by clicking the *Find expressions* link or by pressing **F4**. Note on small screens if there is no room to display the Expressions dictionary without overlapping the Parameters dialog, then it is hidden by default.




Use the Expressions dictionary to locate system expressions, object expressions, behavior expressions, global and local variables and instance variables. Alternatively you can use **autocomplete**: just start typing, and a list of all the options appears. Additionally if you type as far as the dot in an object expression (e.g. `Sprite.`) a list of all the object's expressions appears, shown below. This also appears after the dot when entering a behavior expression (e.g. `Sprite.8Direction.`). Use the **Up** and **Down** arrows to pick an item in the suggestions list, and press **Enter** to insert it. This can help you enter expressions much more quickly.




Some expressions also show tips to help you remember how to use the expression, shown below. This appears when you type the open bracket `(` for the expression. This is most useful for System expressions which tend to have multiple parameters.




Press `Tab` to move to the next parameter or `Shift` + `Tab` to move to the previous. `Enter` is also a shortcut to press **Done**. (Note you may have to press Enter twice if autocomplete is showing: once to choose the autocomplete entry, and again to close the dialog.)

If you make a mistake, the part of the expression which is wrong will be highlighted with a red background. If you press Enter or Done when this is showing, a tip will appear with more information about the problem.

# THE SETTINGS DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/settings

The **Settings dialog** allows you to change various settings for the Construct 3 editor. It can be opened via the main menu.

## User interface

### Language

Select the language for the editor interface. Construct 3 must be reloaded after changing this.

### UI mode

Whether to use a desktop interface, a mobile interface, or automatically decide which to use (the default). This can be useful to force tablets with a mouse and keyboard attached in to desktop mode. Construct 3 must be reloaded after changing this.

### Use simplified user interface

Hide advanced options from the user interface. This is intended to provide a simpler interface for purposes such as classroom teaching with young students. For more information about using this mode, see the tutorial Using the simplified user interface.

### Enable Example Browser

When unchecked, both the Example Browser and all methods of opening it will be hidden. To bring back the Example Browser, check this setting again and then reopen the Example Browser (e.g. via Menu▶View▶Example browser).

### Theme

Change the style of the editor interface to a different theme. A couple of alternative themes are built-in. It's also possible to install third-party themes. Construct 3 must be reloaded after changing this.

### Enable UI animations

Enable animations in the user interface (UI), such as when opening menus, dialogs and so on. If disabled then these actions will happen instantaneously without any kind of transition.

### Enable UI effects

Enable effects in the user interface (UI), such as shadows behind dialogs and menus. If disabled these effects will be hidden, which can help improve performance on low-end devices.

**Enable notifications**

Allow Construct to show information by making a small notification box appear in the corner of the window. This is recommended since sometimes the information is significant, but if they are distracting you can turn them off.

**Show 'take a break' reminder every 2 hours**

Enable a message displayed every 2 hours while the editor is open reminding you to take a break. This is intended to encourage your digital well-being, as taking regular breaks is important for healthy usage of computing devices. It is enabled by default, but can be turned off, for example if you already have other software that shows such reminders.

**Zoom with mouse wheel only**

Normally to zoom the Layout View or Animations Editor, the Control or Command key must be held down while scrolling the mouse wheel. Enabling this setting means no keyboard key needs to be held down and scrolling the mouse wheel alone will zoom the view.

**Use default icon color**

Allow the current theme to specify the icon color. If disabled, the *custom icon color* setting enables.

**Custom icon color**

If *Use default icon color* is disabled, this allows you to choose a custom color for icons in the editor.

**Enable multi-monitor features**

Click to enable features designed to help when using multiple displays, including remembering the position of windows across monitors. This may not be shown if the browser does not support multi-monitor features. When the button is clicked, you may see a permission prompt - be sure to allow permission for the multi-monitor features to take effect.

**Reset user interface**

Click to reset the size and position of all bars and dialogs to their defaults. Construct must be reloaded after doing this.

## Text / code editor

These settings are for Construct's built-in text and code editor, such as when editing a JavaScript file in the project.

**Font size (px)**

Set the font size used in text editors to a size in *px*.

**Automatic indentation**

Check to enable automatic indentation when editing code in the text editors.

**Word wrap**

Enable wrapping text at the edge of the text editor. If unchecked a horizontal scrollbar will appear when text goes past the right edge of the text editor.

**Enable code folding**

Enables expand/collapse buttons in the margin of the code editor which allow for collapsing or "folding" regions of code.

**Use sticky scrolling**

Enables a sticky header at the top of the code editor that shows the current class or function declaration even after you've scrolled past it. This can be helpful in long functions or classes to help you quickly see the context you are working in.

**Show minimap**

Enables the additional scaled-down view of the file by the scroll bar, called the "minimap".

**Minimap scale**

When the minimap is enabled, the size to display the minimap.

## Save & backup

**Default save location**

Choose the default save location when pressing the 'Save' button on a new project. The default depends on which save options the browser supports; where possible it defaults to saving a local file, otherwise it defaults to cloud save.

**Periodically back up active project**

If enabled, Construct will periodically auto-save your project. This can help avoid lost work in the event of a crash or hardware failure.

**Backup location**

Choose which save location to use for backups. By default this saves backups in the same place your project is saved (e.g. on Google Drive if your project is saved on Google Drive). However you can also set it to save to a specific cloud storage service, or a local folder on the device (where supported).

**Local backup folder**

Choose a local folder to save backups to. This option is only used if supported by the browser/platform, and the backup location is set to *Local folder* or *Same location* (in which case it is used for projects saved to the local system). Click the *Choose* button to pick the folder. A folder must be picked for backups to a local folder to work. Construct may have to prompt you the first time it writes to this folder in a session for permission to write to it.

**Backup interval (minutes)**

The duration in minutes after which Construct will automatically make a backup, if enabled.

**Clear recent projects**

Click to clear the entries listed in *Recent projects* in the Start Page and main menu.

**Download local browser saves**

This option is only shown when local browser saves are enabled (which is only in browsers which do not support saving directly to files). Click to download a zip file with all projects saved to the local browser. This is useful for archiving, diagnostics, and recovery if local storage is somehow corrupted.

# Default properties

**Default project author**

**Default project email**

**Default project website**

Set the default project properties used when creating new projects. These settings will be pre-filled in to the author, email and website project properties.

**Default animation speed**

Set the default animation speed for new animations in the Animations Editor. By default this is 5 to play animations at 5 frames per second. It can be useful to change this to 0 if you don't want animations to play by default, allowing manual control of the animation frame.

# Updates

### Notify me about updates for

Opt-in to see notifications about new beta releases. Beta releases are more frequent and include new features sooner, but may have more bugs. By default you will only be notified about new stable releases which are generally more reliable. Note you can try beta releases at any time by visiting the Releases page - this setting only controls which automatic update notifications you see.

> *Note if you save a project in a newer release of Construct 3, such as a beta release ahead of a stable release, that project cannot be opened in the older release. You may wish to back up your projects before using a beta release.*

### Check for updates for installed addons

Enable automatically checking if any updates are available for installed addons via the Addons section of the website. If enabled updates are checked for when the editor starts up, and will show a notification if updates are found; updates can then be installed via the Addon Manager.

## Preview

### Preview with

Choose one of the following ways for previewing projects:

- **Popup window:** open a popup window to display the project in. The popup uses a reduced browser interface, such as hiding tabs, to conserve space. This allows you to view the project in a separate window. However sometimes popup blockers can prevent the window from appearing.
- **Browser tab:** open a full new browser tab to display the project in. The new tab uses the full browser interface. Normally when previewing the browser will add a new tab in the same browser window, and switch from Construct 3 to the project being previewed.
- **Dialog:** open a dialog inside the Construct 3 window to display the project in. This does not involve opening a new browser window at all, so is never blocked by popup blockers. However the dialog cannot be moved outside of the Construct 3 window.

Construct 3 must be reloaded after changing this.

## Start page

### Show the Start Page on startup

Whether to show the Start Page when Construct starts up.

---

**Hide the Start Page when opening a project**

If enabled, then the Start Page will automatically be closed when you open a project or example.

---

**Occasionally show message banners from the Construct team on the Start Page**

When there is a major new update or an active promotion, Construct may show an official message banner on the Start Page. Note this message comes directly from the Construct team - it is not a third-party advert. Uncheck this option if you would prefer not to see these messages.

# Event sheets

---

**Hide 'Add action' links**

Hide the row beneath actions in the Event Sheet View that contains the *Add action* link. This can save vertical space in the Event Sheet View making it easier to read events, but makes it slightly less convenient to edit events. Actions can still be added even when the *Add action* links are hidden using context menus or keyboard shortcuts. Also events with no actions still show an *Add action* link, since it does not take up any extra space.

---

**Translate expressions**

This only applies if you have changed Construct's language to something other than English. For compatibility reasons, expressions in the event sheet must still be written in English, even when the interface is showing a different language. By default Construct translates expressions so you can read and autocomplete them in the same language; however they must still be written in English. Uncheck this setting to keep displaying expressions in English even when the interface is in a different language, which means they appear the same way they are written. This may be particularly useful for bilingual users who also know English.

# Advanced

---

**Use in-app clipboard**

By default when selecting a 'Copy' command, Construct will try to write to the system clipboard. However due to restrictions in when browsers allow this to happen, sometimes the copy is blocked and Construct must prompt you to allow the operation. To avoid this, you can enable the **in-app clipboard**, which does not write to the system clipboard. This avoids these prompts ever appearing, but means you can only copy and paste within the same Construct window.

---

**Cache cloud metadata**

If enabled, this saves information about files and folders in your cloud storage account locally for the duration of the session. This makes it faster to use Cloud Save, since all file listings are immediately available. However it will not update to reflect changes made elsewhere unless you manually refresh the list. If this option is disabled it will always update the file list from the cloud storage service and therefore always be up-to-date, but if your connection is slow (or the cloud storage service is slow) this can make it slow to use Cloud Save.

---

**Limit editor/preview to WebGL 1**

Enable this setting to force the editor and preview to use WebGL 1 even when the device supports WebGL 2. Exported projects will still continue to use WebGL 2 where supported when this setting is enabled. It is intended for developers testing both WebGL 1 and WebGL 2 shader variants in effect addons and should not need to be used for anything else.

---

**Show in-progress languages**

Show languages in the *Language* setting that are not yet complete. This option is intended for translators to help them review their work.

> *Translations are fully reviewed before release, but in-progress translations have not had any kind of review yet. There is no guarantee that the content of in-progress languages is suitable or appropriate.*

---

**Enable experimental features**

Opt-in to testing pre-release features that may not be ready for full release yet. See the blog for news about such features and how to use them. Note there may not always be experimental features to be enabled, but when any are available enabling this setting will make them available in the editor.

---

**Enable WebGPU in editor**

Whether to enable the WebGPU renderer in the Construct editor (which is used to draw Layout Views). If disabled or WebGPU is not supported, a WebGL renderer will be used instead. The renderer in use can be viewed in the Platform Information dialog (via the About dialog). The renderer in use also affects which shader variants will be used for effects; in some cases using third-party effects may require using a specific renderer otherwise they will not be able to render in the editor. The *Auto* setting means Construct will use the default, which is currently the WebGL renderer.

---

**GPU preference**

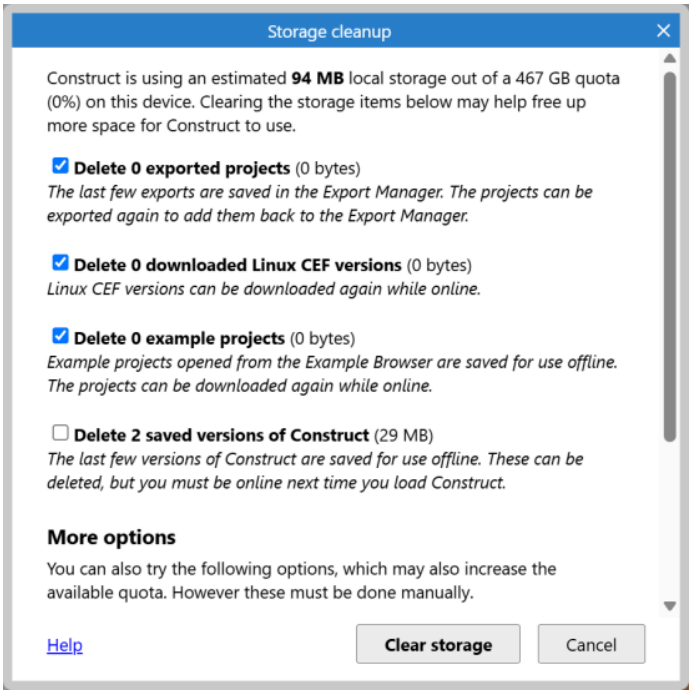
Some systems have multiple GPUs. For example many laptops have a weak, low-power integrated GPU for use on battery, and a more powerful discrete GPU for gaming. This

setting allows you to specify which GPU to prefer. Construct 3 must be reloaded after changing this. You can check which GPU is in use by opening the **About** dialog, clicking *Platform information*, and inspecting the *Renderer* under *WebGL information*. Note not all browsers support this setting, in which case it will have no effect.

# STORAGE CLEANUP DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/storage-cleanup

The **Storage Cleanup dialog** is a tool that helps you to free up storage space on the local device. If you run out of available storage space, it can cause errors to appear and may stop some features of Construct working correctly. This tool provides a way to help solve the problem if that happens. In some cases Construct will recommend you use this dialog to free up more space if it detects a storage problem that may have been caused by lack of free space.




The Storage Cleanup dialog can be opened by choosing Menu▶View▶Storage cleanup. In some cases it may take a while to fully scan storage before listing deletion options.

> *The estimated storage usage and quota are estimates provided by the browser. They may not be exactly accurate, and also may not match up with the amount of space Construct thinks it can save by deleting items. Further it may not immediately update after clearing storage. It is best to only treat it as a guide to whether storage is mostly full or whether there is a reasonable amount of storage space available.*

In general, providing you are online, everything can be safely cleared. Exported projects that are cleared can always be re-exported from the original project, and downloaded Linux CEF versions, example projects, and saved versions of Construct can all be re-downloaded while online. However if you are offline, you may wish to be more careful about the items that you clear, since they may then become unavailable until you next go online. In particular deleting all saved

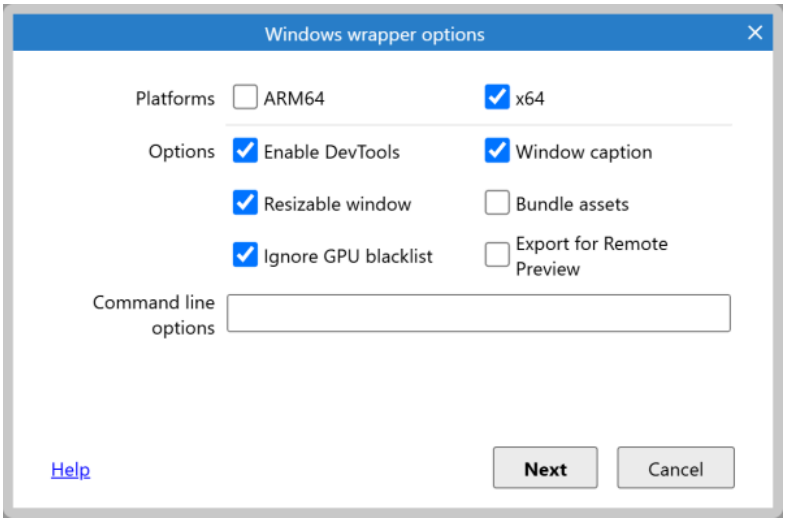
versions of Construct will delete the copy used to work offline, and therefore stop you being able to work offline until you next go online, where Construct will save itself for use offline again. For this reason that option is unchecked by default since you should check you are online before using it. After using it, it's a good idea to reload Construct, which will make it save itself for use offline again.

The items listed in the dialog include explanations of what they are and the consequences of deleting them. There is also additional advice at the bottom of the dialog with additional things you can do to free up space which you will have to do separately since Construct cannot do them for you. Some browsers set a storage quota based on a proportion of the available disk space, so in general anything else that frees up storage space on the device should increase the quota available to Construct.

# WINDOWS WRAPPER OPTIONS DIALOG

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/dialogs/windows-wrapper-options

The **Windows wrapper options** dialog allows you to customize a Windows wrapper (WebView2) export Paid plans only.




*The Windows wrapper options dialog.*

The following options are available.

### Platforms

Select which platforms are exported. You can choose between Windows 64-bit (for Intel-compatible systems, technically referred to as x64), and Windows ARM64 (for Windows on 64-bit ARM-based chips).

### Enable DevTools

Whether the app should allow opening the Microsoft Edge DevTools by pressing F12 or using the *Inspect* context menu option. This is useful for development purposes.

### Window caption

Whether the application window includes the system-provided caption and border.

> *Unchecking this also disables resizing the window.*

### Resizable window

Whether the application window can be resized by the user. If disabled the window can still be made fullscreen using events.

### Bundle assets

By default all your exported project files end up in a subfolder named *www* inside the app. It is possible inquisitive users could find some of your project's assets in here, including artwork, music and sound effects. If you enable *Bundle assets*, all these files are combined in to a single file named *assets.dat*. This file is not encrypted, but it does prevent casual browsing of the exported files. On modern systems, enabling this option should not significantly impact performance.

### Ignore GPU blacklist

Some systems with poor quality graphics drivers can end up crashing or causing severe display glitches when running games. Browsers provide blacklists to recognise faulty drivers or hardware and fall back to software rendering. This guarantees the game will work, but can result in poor performance on such systems. Disabling this option always uses GPU rendering which can be much faster, but can then run in to issues on systems with poor quality drivers.

### Export for Remote Preview

If enabled this exports the project with all its resources and extensions, but the app itself will show preview.construct.net. This allows it to be used to load a Remote Preview from Construct, allowing faster testing with WebView2-specific features enabled.

### Command line options

This option allows advanced users to customise the Chromium command-line arguments used by WebView2.

# KEYBOARD SHORTCUTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/interface/keyboard-shortcuts

The following keyboard shortcuts are available in Construct 3.

Note that on macOS the **Command key** ⌘ is used instead of **Control** for most keyboard shortcuts. However since it is the only exception, for brevity the keyboard shortcuts below refer to `Ctrl`.

## General

`Ctrl` + `X` Cut

`Ctrl` + `C` Copy

`Ctrl` + `V` Paste

`Ctrl` + **Drag with mouse left button** Duplicate

`Ctrl` + `Z` Undo

`Ctrl` + `Y` Redo

`Ctrl` + `A` Select all

`Ctrl` + `D` Unselect all

`Ctrl` + `F` Find by text

`Ctrl` + `Shift` + `F` Focus the Project Bar search field

`Alt` + `N` New project

`Ctrl` + `O` Open project from local file

`Ctrl` + `S` Save project

`Alt` + `W` Close current editor tab

`F4` Preview project from first layout

`F5` Preview current layout

`Shift` + `F4` Debug from first layout

`Shift` + `F5` Debug current layout

`Alt` + **Preview** button to start an additional preview

`Alt` + `4` Start additional preview from first layout

`Alt` + `5` Start additional preview from current layout

`F6` Export project

`F9` Reload all script files from disk (only available when saved as project folder)

`Delete` Delete selected item(s)

`F2` Rename selected item

`Enter` Edit selected item

`Ctrl` + **click** Add or remove clicked item from selection

`Shift` + **click** Select everything in between the last selected item and the clicked item

`Escape` Cancel drag or placement or event search

`Ctrl` + `Shift` + ↑ Move to top

`Ctrl` + `Shift` + ↓ Move to bottom

`Alt` + `Shift` + ← / `B` Back

`Alt` + `Shift` + → / `N` Next

`Ctrl` + `Shift` + ← Go to next tab to right

`Ctrl` + `Shift` + → Go to next tab to left

`Shift` + `S` Go to associated view (switches between Layout View and Event Sheet View)

## Layout view

See also: Layout View

`Ctrl` + **Mouse Wheel Up** or `Ctrl` + `+` Zoom in

`Ctrl` + **Mouse Wheel Down** or `Ctrl` + `-` Zoom out

Hold `Shift` to increase the zoom rate.

`Ctrl` + `0` Return to 100% zoom

**Middle mouse button drag** or **Hold space and move mouse** Pan the view

`Ctrl` + `E` or `Shift` + `S` Go to associated event sheet

**Arrow keys** Nudge selected objects 1 pixel. **Hold** `Shift` to nudge 10 pixels.

> *When grid snapping is enabled, nudging moves a whole grid cell at a time. Hold* ***Alt*** *to disable this and nudge 1 pixel again.*

**Hold** `Shift` while resizing objects for proportional resize

**Hold** `Shift` while rotating objects to lock to 5 degree increments

**Hold** `Shift` while dragging objects for axis-lock (move along diagonals only)

**Hold** `Tab` and **click a selected object** to select the next object underneath in the Z order

**Hold** `Alt` while moving selection to disable resize handles, rotation and grid snapping while held.

**Hold** `Alt` when selecting an instance to bypass any container selection.

**Hold** `Control` while resizing selection to resize relative to the object origin

**Hold** `Tab` while right clicking to show the context menu on the current selection rather than the top instance

`Ctrl` + `Shift` + ↑ Send to front of layer

`Ctrl` + `Shift` + ↓ Send to back of layer

`C` Center horizontally in viewport

`T` Align to top of viewport

`Enter` Wrap selection (to rotate or stretch the selection as a whole)

`W` Select container and wrap. For example, select one object in a container of eight objects, press W, then all eight objects are selected and wrapped.

To paste objects in-place (so they paste at their original positions, instead of relative to the mouse), **hold** `Shift` **while placing a paste**. The full process is: `Ctrl` + `C` to copy an object; press `Ctrl` + `V` to turn the mouse to a crosshair, hold `Shift` and click, all objects paste at their original positions (instead of by the mouse) and the mouse returns to a normal cursor.

`Ctrl` + `R` Start all Live Previews Paid plans only

`Ctrl` + `Shift` + `R` Stop all Live Previews Paid plans only

When editing tilemaps using the Tilemap Bar:

`1` - `6` Switch current tool

`X` Flip horizontal

`Y` Flip vertical

`Z` Rotate 90 degrees clockwise

`A` Rotate 90 degrees anti-clockwise

`R` Reset transformations

`Shift` + **right click** Select a patch of tiles from the tilemap

`Ctrl or Alt Gr` + `[` Use the **replace whole hierarchy** option on the current selection of templates and/or replicas

`Ctrl or Alt Gr` + `]` Use the **modify existing hierarchy** option on the current selection of templates and/or replicas

When editing timelines using the Timeline Bar:

Hold `Ctrl` when dragging a cubic bezier anchor point to also modify the adjacent anchor point if there is any.

## Event Sheet View

See also: Event Sheet View

`Ctrl` + `+` Increase text size

`Ctrl` + `-` Decrease text size

`Ctrl` + `L` or `Shift` + `S` Go to associated layout (if any - event sheets only used via includes have no associated layout)

`Ctrl` + `Home` Go to top of sheet

`Ctrl` + `End` Go to bottom of sheet

`F2` Toggle bookmark at selected event

`Ctrl` + `F2` Go to next bookmark in project

`Shift` + `F2` Go to previous bookmark in project

`F3` Toggle breakpoint

↑ and ↓ Move the selection up and down the event sheet

← and → Move the selection sideways between Events, Conditions and Actions

`-` Insert event above

`+` Insert event below

`A` Add action

`Shift` + `A` Add custom action

`B` Add blank subevent

`C` Add condition

`D` Toggle selected items disabled

`E` Add event below

`Shift` + `E` Add event above

`F` Add function

`G` Add group

`I` Invert selected conditions

`J` Add script (script block if event selected, or script action if action selected)

`Shift` + `J` Add script action (regardless of selection)

`N` Add include

`P` Add parameter to function

`R` Replace object

`Q` Add comment (block comment if event selected, or action comment if action selected)

`Shift` + `Q` Add action comment (regardless of selection)

`S` Add subevent

`V` Add variable

`X` Add 'Else' event following selected event

`Y` Toggle 'Or' block

In the Parameters Dialog, press `F4` to toggle the Expressions Dictionary.

## Animations Editor

See also: Animations Editor

`B` Brush tool

`E` Eraser tool

`F` Fill tool

`I` Color picker tool

`L` Line tool

`N` Pencil tool

`R` Rectangle tool

`S` Rectangle select tool

`T` Ellipse tool

`Shift` + `I` Image points tool

`Shift` + `P` Collision polygon tool

`C` Clear image

`Ctrl` + `E` Export image

`Ctrl` + `M` Mirror image

`Ctrl` + `F` Flip image

`Ctrl` + `R` Rotate image clockwise

`Ctrl` + `L` Rotate image anti-clockwise

`Alt` + `C` Crop image

`Alt` + `R` Resize image

`Ctrl` + `B` Toggle background color

`Ctrl` + `G` Toggle grid

`Shift` + `O` Toggle onion skin Paid plans only

`Ctrl` + `1` Zoom to fit

Quick assign origin and image points:

`Num pad 1` or `End` Bottom left

`Num pad 2` Bottom

`Num pad 3` or `Page down` Bottom right

`Num pad 4` Left

`Num pad 5` Center

`Num pad 6` Right

`Num pad 7` or `Home` Top left

`Num pad 8` Top

`Num pad 9` or `Page up` Top right

`Arrow keys` Nudge 1 pixel

Hold `Shift` to apply the origin/image point to the entire animation. `Shift` + **Click** also applies that positioning to the entire animation.

`Shift` + **Crop button** or `Alt` + `Shift` + `C` Crop entire animation

`Shift` + **Mirror button** or `Ctrl` + `Shift` + `M` Mirror entire animation

`Shift` + **Flip button** or `Ctrl` + `Shift` + `F` Flip entire animation

`Shift` + **Rotate Clockwise button** or `Ctrl` + `Shift` + `R` Rotate entire animation clockwise

`Shift` + **Rotate Anticlockwise button** or `Ctrl` + `Shift` + `L` Rotate entire animation anticlockwise

Animations only:

`Ctrl` + `Up` Previous animation

`Ctrl` + `Down` Next animation

`Ctrl` + `Left` Previous animation frame

`Ctrl` + `Right` Next animation frame

`Ctrl` + `P` Start / restart preview animation

`Ctrl` + `Shift` + `P` Close animation preview

## Timeline Bar

See also: Timeline Bar

`E` Toggle timeline edit mode

`S` Set or update master keyframes and property keyframes at the current time marker position

`Ctrl` + `D` Disable the current selection of timeline elements

`Ctrl` + `E` Enable the current selection of timeline elements

`M` Add missing property keyframes at the current time marker position using the current instance values (a master keyframe must exist at the position for this shortcut to work)

`Ctrl` + `M` Add missing property keyframes at the current time marker position using values which seamlessly fit in the timeline (a master keyframe must exist at the position for this shortcut to

work)

`Space` Preview/pause the current timeline

`Ctrl` + `Space` Stop the current timeline if it is being previewed

`Shift` + `,` Move the play head to the first master keyframe

`,` Move the play head to the previous master keyframe

`.` Move the play head to the next master keyframe

`Shift` + `.` Move the play head to the last master keyframe

**Hold** `Ctrl` while dragging the current time marker to move the marker without previewing the timeline

**Hold** `Shift` while dragging keyframes to duplicate the dragged selection into the new position

`Ctrl` + `X` Cut the current keyframe selection

`Ctrl` + `C` Copy the current keyframe selection

`Ctrl` + `P` Paste keyframes using the current time marker as reference.

If no tracks are selected at the moment of pasting, the keyframes will be added in their respective tracks.

If there are tracks selected at the moment of pasting, an attempt is made to paste the keyframes into the tracks they would fit best. If there are keyframes in the selection which can't be fit anywhere, they are ignored.

`Esc` Clear highlighting on all timeline elements

`Ctrl` + **Mouse Wheel** to scale the timeline UI

`Alt` + `T` Add position property tracks in the current timeline

`Alt` + `S` Add size property tracks in the current timeline

`Alt` + `Z` Add a Z elevation property track in the current timeline

`Alt` + `A` Add an angle property track in the current timeline

`Alt` + `O` Add an opacity property track in the current timeline

`Alt` + `C` Add a colour property track in the current timeline

## Tilemap bar

See also: Tilemap Bar

`Esc` or `1` Restore normal layout editing

`2` - `6` Switch tilemap tool

`X` Mirror tile or patch

`Y` Flip tile or patch

`Z` Rotate tile or patch clockwise

`A` Rotate tile or patch anti-clockwise

`R` Reset all transforms

**Hold** `Shift` with any tile drawing tool to temporarily switch to the Select tool. Releasing Shift returns to the previous tool.

**Hold** `Ctrl` with pencil tool to erase tiles

**Hold** `Ctrl` with eraser tool to erase single tiles

**Hold** `Ctrl` with rectangle tool to erase tiles in the drawn rectangle

## Ease editor

See also: Ease Editor

Hold `Ctrl` when dragging an anchor point to also modify the adjacent anchor point if there is any.

## Flowchart View

See also: Flowchart View

`Ctrl` + **A** Select all nodes.

`Ctrl` + **Mouse Wheel Up** Zoom in.

`Ctrl` + **Mouse Wheel Down** Zoom out.

**Middle mouse button drag** or **Hold space and move mouse** Pan the view

**Hold** `Shift` while selecting nodes and outputs to add them to the same selection to be able to drag or delete them at the same time.

`Backspace` or `Delete` to delete the current selection of nodes, outputs or links.

# 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** (default): 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**: 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.

# LAYOUTS

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

A **layout** is a pre-arranged set of objects. It can represent a game level, menu or title screen, or a scene in an animation. In other tools layouts may be referred to as *scenes*, *rooms*, *frames* or *stages*. They can be added, renamed and deleted from the Project Bar. Layouts are edited with the Layout View. Every layout has an associated event sheet which defines how the layout works.

Layouts contain a stack of layers. A layout must have at least one layer. Objects that appear on the screen do not belong directly to a layout - they belong to one of the layers in the layout.

Layouts do not have a background color. To set a background color, make the bottom layer opaque and set its background color. This can be done in the Layers Bar.

Layouts can also have effects applied, which affects all content appearing in the display.

## Adding and removing layouts

To add a layout, right-click a layout folder (such as the root level *Layouts*) in the Project Bar and select **Add layout**.

To rename or remove a layout, right-click the layout itself in the Project Bar and select **Rename** or **Delete**.

## Layout properties

The properties for a layout can be edited in the Properties Bar after clicking a space in the layout or selecting the name of the layout in the Project Bar.

**Name**

The name of this layout.

**Event sheet**

The associated event sheet that defines how this layout works. Event sheets can be shared between layouts using includes if you have a lot of similar events between layouts.

**Size**

The size, in pixels, of the layout area. If *Unbounded scrolling* is enabled, this still affects how much of the layout area is shown in the Layout View.

### Unbounded scrolling

By default the game window cannot scroll past the edges of the layout. Enable this to allow unlimited scrolling in any direction, even past the edges of the layout.

### Projection

Set the projection used for rendering 3D features. The default *Perspective* projection means things get smaller as they get further away. The *Orthographic* projection instead keeps everything the same size, regardless of its distance from the camera, creating a flat appearance. For an interactive example of both projections, see the Orthographic projection example.

### Vanishing point

When using 3D features such as Z elevation and the 3D shape object with perspective, this specifies where the vanishing point is relative to the viewport area. (This setting does not apply with an orthographic projection, as there is no perspective.) The default is 50%, 50% meaning the middle of the viewport. Consequently as things move in to the distance, they will also move towards the middle of the screen. Altering this will adjust the perspective such that objects moving in to the distance move to a vanishing point elsewhere on the screen. For example setting the vanishing point to 0%, 0% moves the vanishing point to the top-left corner of the screen. This can be used to adapt the 3D perspective to the style of your project. To learn more see the tutorial Using 3D in Construct.

### Effects

Add and edit effects that apply to the whole layout.

## Editor properties

These properties only affect how the layout works in the editor, and don't change how it works at runtime.

### Margins

The size in pixels of extra padding space around the actual layout area that you can scroll around in. Some padding is often useful for conveniently editing the edges of the layout area.

### Show grid

Whether to display a grid in the Layout View.

### Snap to grid

Whether to snap all object placements and sizes to the grid in the Layout View.

**Grid size**

The size of the grid in pixels. This is only used if *Show grid* or *Snap to grid* is enabled.

**Grid offset**

By default the grid is aligned with the top and left edge of the layout area. Adding an offset shifts the grid horizontally or vertically so it is offset from the edges of the layout.

**Show collision polygons**

Display outlines of object's collision polygons in the Layout View. This can help arrange objects with regards to how they collide, rather than just how they appear.

**Show meshes**

Display outlines of object's meshes in the Layout View, if a mesh has been created. See *Editing meshes* in the Layout View manual entry for more details.

**Show translucent inactive layers**

Enable to display all layers other than the active layer at a reduced opacity. This can help identify the content on the active layer.

**Show hierarchy**

Display arrows over scene graph hierarchies pointing from parents to children. See *Setting up a hierarchy* in the Layout View manual entry for more details.

**Project properties**

A shortcut to view the project's properties.

# LAYERS

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

A **layer** is like a transparent sheet of glass that objects are painted on to. Layers can be used to show different groups of objects in front or behind each other, like the foreground and background. Layers belong to a layout and can be added, edited and removed in the Layers Bar. Layers can be scrolled at different rates for parallax effects, and also individually scaled and rotated, which makes them a powerful way to make interesting visual effects.

Layers are also important to add non-scrolling content (e.g. HUDs or UIs) to scrolling projects. A layer with its parallax set to **0, 0** will not scroll at all, so any objects placed on this layer will always stay in the same place on-screen. Note that in this case, objects should be placed within the dashed rectangle that appears in the top-left of the Layout View.

A common arrangement for layers might be:

- **HUD** (top layer - health bar, UI info etc.)
- **Foreground** (objects appearing on top, e.g. explosions and effects)
- **Middleground** (main game objects such as the player and enemies)
- **Background** (bottom layer - the background)

Note that the Free edition is limited to using two layers only.

Layers can also have effects applied, which affects all content appearing on the layer.

## Sub-layers

Layers can also be added as sub-layers of another layer. Sub-layers appear indented in the Layers Bar to show they come under another layer.

A layer with both objects and sub-layers will show its objects on top of its sub-layers. In other words, sub-layers come beneath a layer's own objects in the Z order. This also means that sub-layers act a lot like a simple flat list of layers, and so can be used solely for organizing long layer lists, much like layer folders.

However adding an effect to a layer with sub-layers allows for more efficient and more advanced effects. An effect on a layer with sub-layers will alter the appearance of both the layer and all its sub-layers. This is more efficient than adding the same effect to multiple layers, as it ensures the effect is only processed a single time, while affecting the content of multiple layers.

Layer effects involving sub-layers also allow for composition of more advanced effects. For example a group of layers can be combined to make a single lighting layer, which then affects

the appearance of another group of layers beneath it. See the Shadows: blending multiple lights example for a demonstration of this technique.

## Global layers

Sometimes many layouts in a project have the same content on a particular layer, such as for interface or HUD overlaid on to the project. Changing this content then becomes a chore since changes must be repeated on every layout. **Global layers** are aimed at solving this problem.

If a layer's *Global* property is enabled, then **every layer in the project with the same name** is overridden by that layer. The initial objects, as well as its properties, are used instead of the other layer's own content and properties. Then changes can be made once to the original global layer, and the changes will be applied project-wide.

The layer with the *Global* property enabled is the "master" layer. On other layers in the project with the same name, the *Global* property will be read-only and display *Overridden* to indicate it is being substituted by a different layer. The same layer's content will appear in the editor, and all edits will affect the master layer, no matter which layout it is being edited from.

Whether a layer is the original global layer or is overridden will be shown next to a layer's name in between parenthesis in all relevant places, these includes the *Layers* dropdown in the Properties bar when an instance is selected and next to each item of the Layers Bar.

## Layer properties

The properties for a layer can be edited in the Properties Bar after clicking the layer in the Layers Bar. Note this also changes the **active layer**.

---

**Name**

The name of the layer, which can be used to refer to the layer in the event system.

---

**Initially visible**

Whether or not the layer is initially visible **when previewing**. This is different to the *Visible in editor* property which only affects the Layout View.

---

**Initially interactive**

Whether or not the layer is initially interactive when previewing. If disabled, then the content of the layer will not respond to mouse or touch input.

---

**HTML elements layer**

Allow HTML elements to appear above this layer. This allows content on other layers above this layer to render on top of HTML elements on this layer. Layers which enable this are shown with a special icon in the Layers Bar. For more details see HTML layers.

### Use render cells

Optimise the rendering of this layer for extremely large layouts with a large number of static objects spread out across this layer. This is not normally necessary except for certain types of large projects. If this is used incorrectly, it can actually make rendering less efficient, so make sure you can measure a performance improvement before using it. For more information, see the blog post How render cells work.

### Scale rate

Change the rate at which the layer zooms if scaling is applied to the layer or layout, a bit like parallax but for zoom. A scale rate of 0 means the layer will always stay at 100% scale regardless of the scaling applied. A scale rate of 100 means it will scale normally.

### Parallax

Change the rate at which the layer scrolls in the horizontal and vertical directions. A parallax rate of *100% x 100%* means ordinary scrolling, *0% x 0%* means it will never scroll (useful for UIs), *50% x 50%* means scrolling half as fast, etc. Also useful for multi-layer parallaxing backgrounds.

### Z elevation

The Z elevation of the 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). You can learn more about Z elevation in the tutorial Using 3D features in Construct.

### Transparent

Make the layer have a transparent background. If enabled, the background color is not used.

### Background color

The background color for the layer, if it is opaque (i.e. *Transparent* is disabled).

### Opacity

Set the opacity (or semitransparency) of the layer, from 0% (invisible) to 100% (opaque).

### Force own texture

Force the layer to always render to an intermediate texture rather than directly to the screen. This is useful for some kinds of effects. However it slows down rendering, so it should be disabled unless specifically needed.

### Uses own texture

A read-only property indicating if the layer renders to an intermediate texture. This has a performance overhead. The *Force own texture* setting enables this, but some other properties also cause the layer to use its own texture, including changing the layer opacity from 100%, changing the blend mode, or adding effects.

### Rendering mode

When using 3D rendering mode, this setting can change a layer back in to rendering in 2D mode. This allows projects with 3D content to still use 2D layers as backdrops or overlays which are not affected by depth. For example a HUD layer ought to display on top of all 3D content, regardless of depth, so would typically use a 2D rendering mode for the layer. Otherwise in 3D mode, 3D features may still overlap the layer content if they rise higher than the layer contents. For an example, see Combining 2D & 3D layers, and you can learn more about 3D features and 2D layers in the tutorial Using 3D features in Construct.

> *This property only appears for projects using 3D rendering mode. See the Rendering mode project property.*

### Draw order

This setting only appears for layers using a 3D rendering mode. The default draw order is *Z order*, meaning objects are drawn in a back-to-front order according to the Z order of instances on the layer. 3D layers can also be set to *Camera distance* draw order, which instead ignores the Z order and draws instances on the layer according to how far away from the camera they are, from furthest away to nearest. This has no effect on opaque objects, but is important for rendering transparency in 3D. For more information see the tutorial Using 3D in Construct.

### Blend mode

Change the way the layer is blended with the background when it is rendered to the display. See the *Blend modes* example that comes with Construct 3 for a visual demonstration of each.

### Effects

Add and edit effects that apply to the whole layer.

### Visible in editor

Whether or not the layer is showing **in the Layout View**. Note this is different to the *Initially visible* property which only affects previewing. This setting can also be accessed via the Layers Bar.

### Locked

Whether or not the layer is locked in the Layout View. Objects on locked layers cannot be selected. This setting can also be accessed via the Layers Bar.

---

**Parallax in editor**

If enabled, the *Parallax* property will also be applied in the Layout View, allowing you to preview what the effect will look like.

---

**Global**

See the section above on *Global layers*. If enabled it will override every other layer in the project with the same name with its own contents and properties. Overridden layers display this property read-only as *Overridden*. If disabled its contents and properties are unique to itself.

# OBJECTS

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

In Construct **objects** perform most of the useful work in a project. Most of the things you see in a Construct project are represented by objects, and there are also hidden objects for other purposes (e.g. audio playback).

When inserting a new object, typically you first choose the plugin in the dialog (e.g. *Sprite*). This then creates an object type (e.g. *TrollEnemy*). When the mouse turns to a crosshair this allows you to place the first *instance*, and you can duplicate the instance to create more of them.

Understanding the differences between them is essential to use Construct effectively, especially *object types* and *instances*. The rest of this manual section goes in to each aspect of objects in more detail.

# PLUGINS

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

Plugins define a kind of object. For example, a Sprite is a kind of object, and the Audio object is a different kind. These are defined by the *Sprite plugin* and *Audio plugin* respectively. See the plugin reference for more information on individual plugins.

JavaScript developers can make new plugins (and behaviors) using the Addon SDK. See also Third-party addons for information about installing addons.

Most plugins define their own properties in the Properties Bar. To see a full list, locate the plugin in the reference section.

There are three main kinds of plugins:

1. **Visual plugins** (e.g. Sprite) appear in the layout and draw something to the screen.
2. **Hidden plugins** (e.g. Array) are placed in a particular layout, but do not draw anything to the screen.
3. **Project-wide plugins** (e.g. Mouse, Audio) are added to the entire project, and can only be added once. There cannot be more than one object type or instance of a project-wide plugin. They simply enable a new capability (such as being able to take mouse input) to events.

Construct is designed modularly. That means not much functionality is built in: you must insert a plugin before you can use the related features. For example, you cannot play back any audio before adding the Audio plugin to a project. This might seem unnecessary, but there are many project-wide plugins and it is unlikely every project will need to use all of them. For example, if the Audio plugin was automatically included with every project, even projects which do not need Audio support would end up burdened with its features and code. So remember if you do not add a plugin, it is not at all included in your project, and this helps your projects remain lean and efficient when exported.

# OBJECT TYPES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/objects/object-types

**Object types** are a central part of game design in Construct. Object types define a 'class' of an object. For example, *TrollEnemy* and *OgreEnemy* could be different object types of the Sprite plugin. They have different animations and events can be applied separately to make them look and act differently, despite the fact they are both Sprite objects.

There can be multiple instances of an object type in a project. For example, you may wish for there to be four *TrollEnemy* objects awaiting the player in a game. These four **instances** share the same animations, images, behaviors, instance variables and events. (In the case of instance variables, each instance stores its own unique value, e.g. for health, and behaviors work independently for each instance too.)

**Object types** do not themselves have a position, angle or size. These are properties of the **instances** of the object type. The Project Bar displays the object types in the project, but not the instances. You can also add, rename and delete object types from the Project Bar.

Events are made to apply to an object type. The event then filters the instances that meet the condition. For example, the event "Bullet collides with Alien" is an event that applies to all instances of the *Bullet* and *Alien* object types. However, when the event runs, the actions only apply to the specific *instances* involved in the collision. For more information see How events work.

Object types can also be grouped together in to Families Paid plans only. This can help avoiding repeating the same events for different object types.

# OBJECT INSTANCES

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

Object **instances** are the actual objects you see in a game: an *instance* of an object type. For example, if there are four *TrollEnemy*s in a layout, those are four *instances* of the TrollEnemy *object type*.

It is **instances** which have a position, angle and size in the layout. Object types do not have these properties - they simply define a 'class' of object.

Instances can be created at runtime in events (typically by the System *Create object* action and the Sprite *Spawn an object* action). They can also be pre-arranged in layouts with the Layout View to design levels, menus and title screens. Instances can also be individually animated in timelines.

Selecting an instance in the Layout View shows its properties in the Properties Bar. These are a mix of properties in common with all (or most) objects, and plugin-specific properties. The common properties are described below, and plugin-specific properties are described for each plugin in the reference section.

## Common instance properties

The following properties are common to most objects, depending on their capabilities.

**Name**

The name of the associated object type.

**Global**

By default, all instances are destroyed when the layout ends (e.g. when going to the next layout). If enabled, none of the instances of this object type will be destroyed when switching layouts.

**Plugin Read-only**

A reminder of the plugin this object is based on.

**Position**

The X and Y co-ordinates in the layout in pixels. This is measured to the object's origin. This can also be altered by moving the instance in the Layout View.

**Size**

The width and height of the instance in pixels. This can also be altered by dragging the resize handles in the Layout View.

**Angle**

The angle in degrees the instance is oriented at. This can also be altered by rotating the object in the Layout View by clicking and dragging just outside the resize handles.

**Opacity**

The instance opacity (or semitransparency), from 0% (transparent) to 100% (opaque).

**Color**

A color tint to apply to the instance. This works by normalizing each color component in the 0-1 range, and multiplying it with the object's color. This means a white color (with 1 for each color component) displays the original color of the object. Choosing another color will tint the object, e.g. choosing red will preserve only the red color component of the object's image.

**Layer**

The layer the instance is placed on. In the case the selected instance is from from a global layer in a different layout to the one currently active, the dropdown will show first the layers of the layout the instance is really coming from, followed by the layers of the layout which is currently active.

**Z elevation**

The instance's elevation on the Z axis. By default the camera is at Z = 100, and looking down to Z = 0. The default Z elevation is 0. Increasing it will move it upwards (towards the camera) and decreasing it will move it downwards (away from the camera).

*Z elevation only affects the appearance of the object. It does not affect collisions - everything else continues to work in 2D as if its Z elevation was still 0.*

*Z elevation takes precedence over Z order. In other words, using Send to top of layer will not make an object appear on top of an object that has a higher Z elevation.*

**Z index Read-only**

Indicates the zero-based Z index of the instance on its layer relative to all the other instances on the layer. A value of 0 means it is the bottom instance, and increasing values mean it is closer to the top of the layer. The Z index can be modified using the Z Order Bar Paid plans only.

**UID Read-only**

Every instance in the project has a unique number assigned, called its unique ID or UID. This value is displayed in the editor so you can view the UID for specific instances. You can use conditions like *Pick by unique ID* in events to pick specific instances by their UID.

**Tags**

A space separated list of string tags to identify an instance. The first tag of an instance is used in some places in the editor to better distinguish specific instances. There are associated common ACEs and a scripting interface to work with tags.

**Edit variables**

Open the **Object Instance Variables dialog**.

**Edit behaviors**

Open the **Object Behaviors dialog**.

**Edit effects**

Open the **Effects dialog**.

**Container**

Group a set of object types together so they create, destroy and pick in events together. See the dedicated section on Containers for more information.

**Template**

A set of properties for managing templates, which allow conveniently updating properties of instances across the entire project. See the dedicated section on Templates for more information.

**Visible in editor**

An editor-only property, determines if the instance is visible or not. Can also be set through the Instances Bar.

**Locked**

An editor-only property, determines if the instance can be interacted with. Can also be controlled through the Instances Bar.

## Index IDs (IIDs)

As well as unique IDs (UIDs, described above), all instances are also assigned an Index ID (IID). This is the zero-based index of the instance within its own object type. The first instance created for each object type is assigned an IID of 0, and subsequent instances are assigned incrementing numbers. Unlike UIDs, IIDs can change: if an instance is destroyed, all the object type's instance's IIDs are reassigned so they are continuous (i.e. 0, 1, 2, 3... N with no gaps). Therefore an IID does not persistently refer to one instance - use UIDs for that purpose. However IIDs can be useful for advanced users taking advantage of object expression indexing, the *Pick Nth instance* system condition, or the *IID* expression.

# INSTANCE VARIABLES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/objects/instance-variables

**Instance Variables** are added to object types but store numbers, text or booleans (on/off flags) individually for each instance. This makes them ideal for things like health counters in a game, since each instance tracks its own value. Instance variables are added to object types with the **Object Instance Variables dialog**, and the initial values for each instance can be set from the Properties Bar.

Click here to open an example of instance variables.

Instance variables can also be used to help control instances independently of each other. For example, a Boolean instance variable could be used to determine if an enemy is hunting down the player (*true*) or running away (*false*). If instances all have different values, the condition *Is boolean instance variable set* can be used to apply actions to enemies hunting down the player. Inverting the condition (picking instances with the value being *false*) can then be used to apply actions to enemies running away. The end result is a number of instances of the same object type acting independently: some chasing and others running away. This is a simple example - much more complex methods can be made using multiple instance variables. In other words, an instance's *state* can be controlled using instance variables.

Instance variables can also be added to Families Paid plans only. All the object types in the family then *inherit* the instance variable.

## Autocompleting string instance variables

When using string instance variables, Construct will offer to autocomplete the instance variable with other strings it is referenced with in both event sheets and properties. The autocomplete options will appear in both the Parameters Dialog (after typing the first `"` character) and the Properties Bar.

This is useful for string instance variables that represent a fixed set of states, such as "idle", "searching" and "attacking". If your event sheets or properties reference a set of strings like this, then they will be offered for autocomplete in properties and parameters, helping show the list of available strings and avoiding typos from re-entering the values.

# BEHAVIORS

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

Behaviors add extra capabilities to object types. They can be added with the **Object Behaviors dialog**. See the behavior reference section for more information on individual behaviors.

An example of a behavior is the **8 direction** behavior. This instantly allows an object to be moved around with the arrow keys. This helps speed up development and increase productivity. Behaviors are not intended to do everything in your game for you: the events are where the majority of your game logic will be defined. Behaviors are essentially time-savers and shortcuts. Most behaviors can be replicated with events, but it simply is far more time consuming to do so. Behaviors are very customisable, but if a behavior isn't doing quite what you want it to, you can usually resort to reproducing it in a customised way with events.

All instances of an object type use its behaviors. You cannot add a behavior to only some of the instances - they all use the behavior - although you may be able to enable or disable the behavior for individual instances.

Some behaviors add their own properties to the Properties Bar. See the behavior reference for each behavior's properties

Some behaviors also extend the objects they are added to with their own conditions, actions and expressions. These are shown alongside the object's own conditions, actions and expressions in the **Add Condition or Action dialog** and **Expressions dictionary**.

Behaviors can also be added to Families Paid plans only. All the object types in the family then *inherit* the behavior.

# EFFECTS

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

**Effects** change the visual appearance of objects. They can be added with the **Effects dialog**. Effects can also be added to layers and layouts, although effects which blend with the background cannot be used on layouts. Effects are also sometimes referred to as *shaders* or *shader effects*, since this refers to the underlying technology. Below is an example of the Water effect on an image.



*An example of the Water effect*

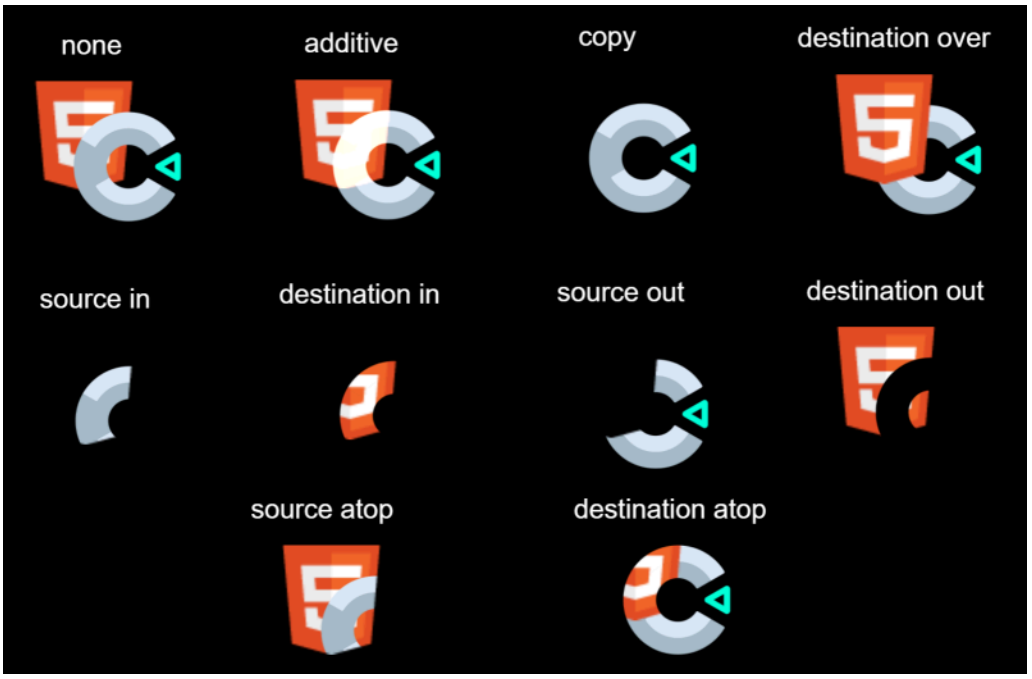
Construct provides a library of over 80 effects. Adding effects also displays them in the Layout View if *Preview effects* is enabled in project properties. A number of examples of effects are also provided in Construct's examples which you can find by searching for *Effects* in the Start Page.

Multiple effects can be applied to a single object, layer or layout. In this case the effects are chained. The result of the first effect is processed by the second effect, then the result of that is processed by the third effect, and so on.

Note that the Free edition is limited to using two effects in a project only.

## Blend mode

The *Blend mode* provides a simple set of pre-defined ways to blend the object with the background. Click here to open an example of blend modes in Construct. The image below also demonstrates the available blend modes.




*The blend modes available in Construct*

If multiple effects are used, the blend mode is applied only to the last effect. For example with three effects, the effect chain is processed normally, and the blend mode is only used to blend the result of the third effect with the background.

## Changing effects at runtime

Objects supporting effects provide common actions to enable or disable effects, or set an effect parameter. This allows you to switch effects or adjust effect parameters at runtime, allowing for greater possibilities and creative uses. To enable or disable layout or layer effects, or change their parameters, use the relevant system actions.

## Performance

Using too many effects can cause poor performance, especially on mobile devices. Try to only use effects when it is important to the appearance of the game.

Creating many instances of an object using effects can be very inefficient, since the effect must be processed repeatedly for small areas. If many instances need to use an effect, sometimes it is more efficient to place all the instances on their own layer, and apply the effect to that layer instead. This can improve performance whilst producing the same visual appearance.

Never use effects to process a static effect on an object. For example, do not use the *Grayscale* effect to make an object always appear grayscale. Instead apply the grayscale effect in an image editor and import a grayscale image to the object, without using any effects. This has the same visual result, and avoids performance-degrading effect processing. Effects like *Grayscale* should only be used for transitions or making objects only occasionally appear grayscale.

For more information, see the manual section on performance tips.

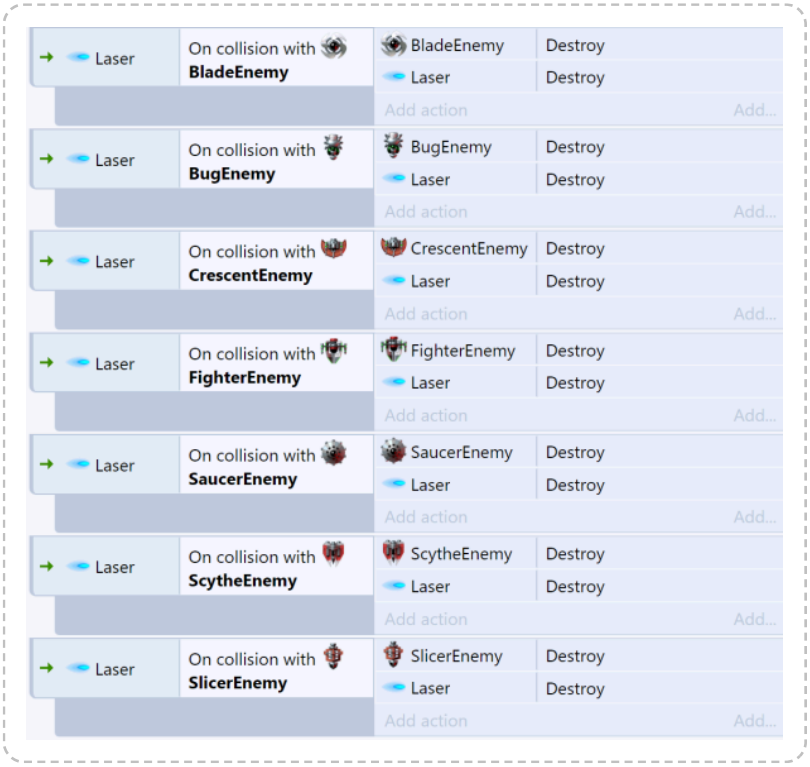
# FAMILIES

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

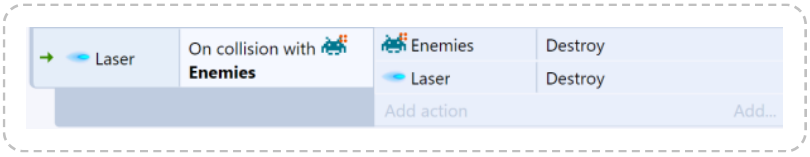
Paid plans only In Construct, **Families** are groups of object types. All the object types in a family must be from the same plugin, e.g. all Sprite objects (and not a mix of Sprite and Tiled Background objects, for example).

Families can help you avoid repeating events. For example, instead of having the same events for the *Enemy1*, *Enemy2* and *Enemy3* objects, you can add them all to an *Enemies* family and make the events once for the family. Then, the events automatically apply to all the object types in the family.

The Families example in Construct demonstrates the advantage of this. There are seven kinds of enemy, and they all need to be destroyed when the laser hits them. Without families, seven separate events are necessary, as shown below:



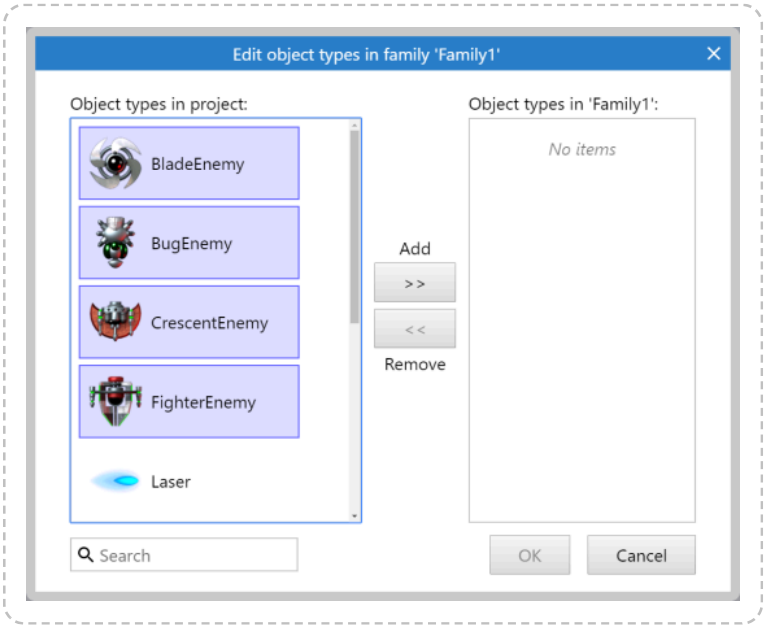

Using families all seven events can be replaced with a single event:




This makes it far easier to create and maintain projects with lots of objects that need to work in similar ways.

# How to create a family

**Right-click** the *Families* folder in the Project Bar and select **Add family**. The *Edit Family* dialog appears.




Objects on the left are the objects in the project that can be added to the family. Objects on the right are the objects already in the family. **Double-click** an object to transfer it to the other side. You can select multiple objects by holding `Control` and clicking several objects, then clicking one of the buttons in the middle to transfer them.

When done, click **OK** and the family will appear in the Project Bar. It can be expanded to show all the objects in the family as well. The family, and the objects in the family, can be edited by **right-clicking** them and choosing options from the menu, like *Remove from family* or *Edit family*.

Objects can be added to multiple families. All events for the object's families will apply to the object.

# Family instance variables

Instance variables can also be added to a whole family by **right-clicking** the family name in the Project Bar and selecting Family instance variables.

If you add an instance variable to a family, *all* the object types in the family inherit the instance variable. For example, adding the instance variable *health* to the family *Enemies* in the above example will mean *BladeEnemy*, *BugEnemy*, *CrescentEnemy*, *FighterEnemy*, *SaucerEnemy*, *ScytheEnemy* and *SlicerEnemy* all gain a *health* instance variable. It will also appear in the editor alongside each object's own instance variables. However in the Event Sheet View the family will only show its own instance variables (those added directly to the family). This means any instance variables you want to be available to the family's events must be added to the family, and not to the objects in the family.

If an object type belongs to multiple families, it inherits every family's instance variables.

## Family behaviors

Behaviors can also be added to a whole family by **right-clicking** the family name in the Project Bar and selecting Family behaviors.

As with family instance variables, if you add a behavior to a family, *all* the object types in the family inherit the behavior. The behavior will appear in the events for all the objects in the family *and* the family itself. For example adding the *Bullet* behavior to a family called *Bullets* means the bullet's *Set speed* action is available to every object in the family, as well as the family itself.

If an object type belongs to multiple families, it inherits every family's behaviors.

## Family effects

Effects can also be added to a whole family by **right-clicking** the family name in the Project Bar and selecting Family effects.

As with family instance variables and behaviors, if you add an effect to a family, *all* the object types in the family inherit the effect. This can be useful for quickly applying effects to a number of different object types.

If an object type belongs to multiple families, it inherits every family's effects.

## Picking families in events

Families pick instances in the event sheet independently of the object types in the family. For example, consider *Family1* consisting of *SpriteA* and *SpriteB*. Conditions for *Family1* will never affect which *SpriteA* and *SpriteB* instances are picked. It will only affect which instances are affected when running an action for *Family1*. Likewise, conditions picking *SpriteA* and *SpriteB* instances will never affect which instances are picked in *Family1*. In other words, in the event sheet families are treated like an entirely separate object type, which just happens to have instances from other object types. This can be taken advantage of if you need a single event to pick two separate lists of instances from the same object type.

## Upgrading ordinary objects to families

If you make a lot of events forgetting to use a family and want to replace them, it's possible to use the *Replace Object* feature to save you re-doing every event. The process is described in the tutorial How to upgrade an object to a family.

## Summary

Families are a very powerful feature which are essential to help keep large projects simple. Instance variables and behaviors added to families are inherited by every object in the family, which allows for sophisticated logic to be easily applied to many object types at once.

# CONTAINERS

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

**Containers** are an advanced feature to help build *composite objects* - that is, elements of your game made from multiple objects. For example, a tank in a strategy game might be composed of two parts: a sprite for the tank base, and a different sprite for the tank turret. This allows them to rotate independently. Adding them both to a container then allows events to treat both objects as if they were one, because they are always picked together.

> ***Hierarchies** are another feature that help with building composite objects (see Setting up a hierarchy in the Layout View). Containers generally apply to picking groups of objects in event sheets, whereas hierarchies generally apply to making objects move and rotate together. Both features can be used together as well.*

It is essential to be familiar with how events work in order to understand how containers work.

## Creating a container

To add an object to a container, select one of the objects you want in the container and click the **Create** link in its properties (which appears under the *Container* category next to the label *No container*). A dialog opens allowing you to choose the object to add to the container.

Further objects can be added to a container by clicking the *Add object* link in the *Container* category again. Objects can be removed by clicking the *Remove* link.

## What containers do

Placing objects in a container has the following effects:

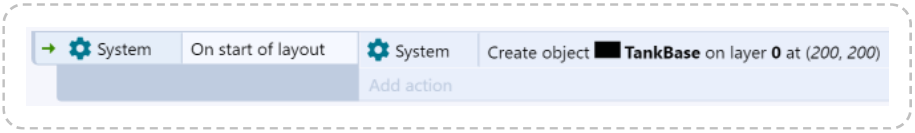
1. If one object in a container is created, every other object in its container is also automatically created.
2. If one object in a container is destroyed, every other associated object in its container is also destroyed.
3. If a condition picks one object in a container, every other associated object in its container is also picked.

The first two points basically guarantee that there is the same number of instances for all the objects in a container. In other words, containers are created and destroyed as a whole. Using the tank base and turret container example, it is impossible to create a tank base without also automatically getting a new turret for it as well.

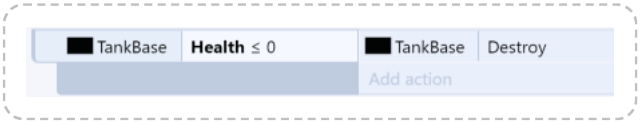
The third point is the main purpose of containers. Containers are also *picked* in events as a whole. This makes events treat containers as if they were one object. For example, if a condition picks a tank base instance, it also automatically picks the base's associated turret.

## Examples

In the following events, assume both *TankBase* and *TankTurret* are in a container.




In the above event, a *TankTurret* instance is also created since it is in a container with *TankBase*. It might also be useful to add an action to set the position of the *TankTurret* to make it appear on top of the base.




In this event, the associated *TankTurret* instance is also destroyed since it is in a container with *TankBase*.




In this event, when a bullet hits the tank base, only its associated turret flashes. If the objects were not in a container, *all* the turrets in the game would flash, as per the rules of how events work (since no turret was referenced in the conditions, the action applies to all of them). However, since the objects are in a container, when the *TankBase* that was hit by a bullet is picked, its associated turret is also picked. This makes the event work as intended, and the event treats both objects as if they were one. This is the crux of containers, and for some uses like strategy games, there will be a large number of events taking advantage of this type of picking to ensure objects work as units and don't accidentally affect other instances.

## Placing container objects in a layout

In the Layout View, it's possible to create an instance in a container by itself. This appears to break rules 1 and 2 under *What containers do*, since objects in a container must always create and destroy together. However, the editor does not enforce this. Instead, any missing objects are created automatically when the layout starts. It is a good idea to make sure you create enough objects anyway so you can edit the object's position, instance variables, and other properties from the Layout View.

## Using containers in the Layout View

In the Layout View, when you highlight an instance in a container, the other instances in the same container highlight in yellow to help you identify which instances are grouped together.

You can also change the *Select mode* property to one of the following:

- **Normal:** instances in the container select individually like normal.
- **All:** instances in the container all select together whenever you select one of them. This is like always multi-selecting every instance in the container whenever you click one of them.
- **Wrap:** also selects all instances in the container, and then also wraps the selection so they stretch and rotate as one. This means you can treat containers as if they are one object. For more information see *Selection wrapping* in the Layout View.

You can circumvent the selection mode by holding `Alt` and selecting an instance. This will let you select just that instance even when *Select mode* is *All* or *Wrap*. This helps you change the container as well, such as to add or remove another object type to the container.

## Data storage objects in a container

It's possible to add data storage objects like Array and Dictionary to a container with another object. Despite the fact these objects are invisible, a separate instance of the object is still created for each container. This allows you to have a dedicated Array or Dictionary for each instance of an object. This can be very useful as an advanced substitute for instance variables, such as if a very large number of variables is necessary, or if variables need to be dynamically added and removed.

# TEMPLATES

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

Templates help managing instances in larger projects. The main uses for them are:

1. Conveniently updating properties for multiple instances across the project in the editor. This can prove to be time consuming and prone to error if done individually. Using templates you can declare a single instance to be the source **template** and other instances of the same object type to be **replicas** of it. After doing so, changing the template will also change all replicas, even across layouts.
2. A template can be used as a preset when creating an instance at runtime using the **Create Object** action. Doing this is can help avoid needing lots of initialization actions and makes it easier to make changes to existing presets or add new ones later on.
3. Using templates it is possible to define different hierarchies in the editor, and then choose which one to create at runtime.

## Terms

1. **Template:** an instance that has been set to be the source of property values for other instances to use. These include common properties, plugin properties, instance variables, behavior properties and effect parameters. A template can also be used to decide what values a new instance created at runtime should take when using the **Create Object** action. Modifying a template in the Layout view or through the Properties bar will immediately be reflected in all instances which are replicas of it.
2. **Replica:** an instance that has been set to use an instance already defined as a template. A replica takes its values from the source template unless they are explicitly modified, at which point the replica's own values are used instead.

## Template properties

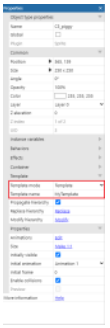
The Properties Bar shows relevant properties in the *Template* section. The properties that can be shown there are:

- **Template mode:** an instance can be turned into a template by selecting the *Template* value from the drop-down list. If there are any instances in the object type which have already been set to be a template, then the *Replica* value will also be available in the drop down menu.
- **Template name:** the name to identify a template. This is only shown when **Template mode** is set to *Template*.
- **Template source:** the name of the template a replica is using. This is only shown when **Template mode** is set to *Replica*, and only lists templates of the same object type.

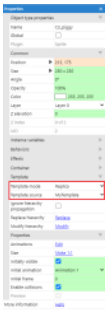
The three main cases of using templates in the Properties Bar are shown in the images below. Click the thumbnails to expand them.



*Properties for an instance not using templates.*



*Properties for an instance set to be a template.*

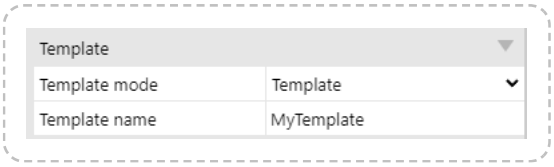


*Properties for an instance set to be a replica. Note the highlighting on the properties to identify which are from the template, and which have been overridden for this instance.*

## Setting up templates and replicas

To start using this feature the first thing you will need to do is set an instance to be a template. To do that, follow these steps:

1. Select the instance you want to be a template in the Layout View.
2. Pick the *Template* option from the **Template mode** dropdown in the Properties bar.
3. The **Template name** input will appear.
4. Give the template a name using the **Template name** text input.



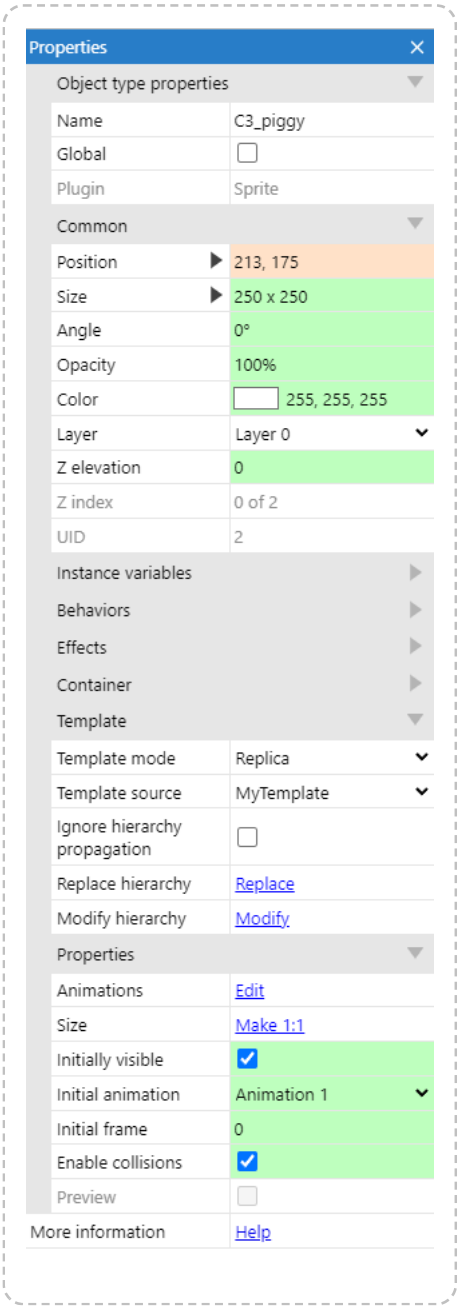

That's all you need to do to set an instance to be a template.

Now let's see how to set other instances to use the template in the editor:

1. Select the instance you want to be a replica of the template in the Layout View. The instance must be of the same object type as the template.
2. Pick the *Replica* option from the **Template mode** dropdown.
3. The **Template source** dropdown will appear.
4. Select the template the replica instance should use from the **Template source** dropdown.




After doing that you will notice that some of the properties in the Properties bar are highlighted. This indicates that the instance is taking that value from the source template.

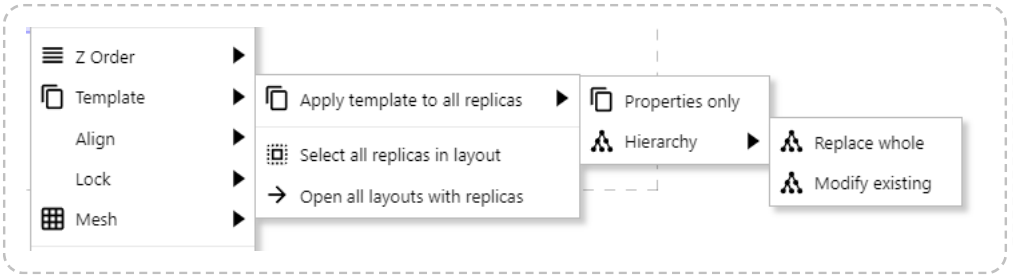



From now on, when the template is modified, all replicas using it will reflect the changes. If a replica is modified individually, the affected property will stop taking the value from the template and instead will take the value from the replica itself. To indicate this, the property will be highlighted in a different color.

# Miscellaneous options

## Layout view context menu options

Template context menu



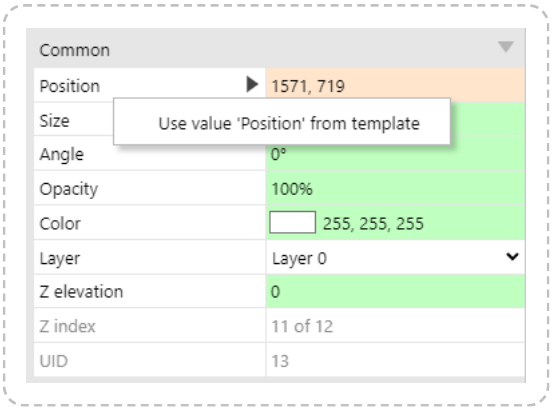

Replica context menu




- **Template instance context menu options:**
  - Apply template to all replicas▶Properties only forces all replicas using this template to use the values of the template.
  - Apply template to all replicas▶Hierarchy▶Replace whole replaces the whole hierarchy of each replica with that of the template. This creates and deletes instances as necessary.
  - Apply template to all replicas▶Hierarchy▶Modify existing updates the existing hierarchy of each replica to match as best as possible the hierarchy of the template. This does not create or delete any instances.
  - Select all replicas in layout selects all replicas of this template in the layout.
  - Open all layouts with replicas opens all layouts with replicas of the selected template.
- **Replica instance context menu options:**
  - Apply template▶Properties only forces the selected replica to use the values from the template.
  - Apply template▶Hierarchy▶Replace whole replaces the whole hierarchy of the replica with that of the template. This creates and deletes instances as necessary.
  - Apply template▶Hierarchy▶Modify existing updates the existing hierarchy of the replica to match as best as possible the hierarchy of the template. This does not create or delete any instances.
  - Select all replicas in layout selects all replicas of this template in the layout.
  - Open template layout opens the layout containing the template the selected replica is using.

## Properties bar context menu options

When a replica's property is modified and so starts using its own value, it's possible to individually set it back to use the template value by right clicking on the property name.




### Other Properties bar options

#### Propagate hierarchy

This property is only shown for template instances, if toggled all hierarchy related changes will be propagated automatically to the replicas. Turned on by default.

#### Ignore hierarchy propagation

This property is only shown for replica instances, if toggled any changes automatically propagated from a template are ignored. Turned off by default.

#### Replace hierarchy

When used from a template, will cause all the replicas to match the template's hierarchy. When used on a replica, that replica matches it's templates hierarchy. Creates and destroys instances as needed. Using this option ignores **Propagate hierarchy** and **Ignore hierarchy propagation** settings.

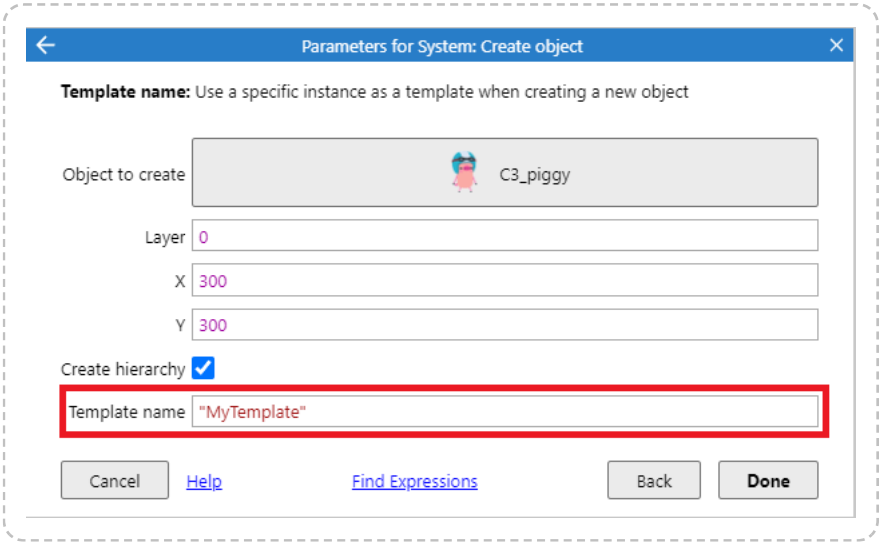
#### Modify hierarchy

When used from a template, will cause all the replicas to match the template's hierarchy as best as possible. When used on a replica, that replica matches it's templates hierarchy as best as possible. No instances are created or deleted. Using this option ignores **Propagate hierarchy** and **Ignore hierarchy propagation** settings.

## Runtime actions

The **Create Object** system action has an additional **Template name** parameter. When specified the new instance being created will be a clone of the template instance, rather than an arbitrary one.

As well as using the properties of the template instance, the new instance will also use the hierarchy from the template if one exists. This means that it is possible to create different hierarchies in the editor that use the same type of instance as a root, while also being able to choose which one to create at runtime.




If the provided template name is not available, the *Create Object* action will behave as if the parameter wasn't provided (using an arbitrary instance instead).

## Dragging and dropping from the Project Bar

Instances can be created by dragging and dropping an object type in the Project Bar to the Layout View. Normally this creates a default instance based on another instance in the project. However it may be desirable to control which specific kind of instance is created when you do this. Templates can also be used to determine which instance is created in this case.

When you create a template, selecting the object type in the Project Bar will now show two additional properties in the Properties Bar:

- **New instance template:** the template to use when creating a new instance from the Project Bar. You can choose any template in the project for the object type. The created instance will copy all its properties from the template.
- **New instance replica:** check this to set an instance created from the Project Bar to be a replica of the template. This means it will automatically update when the template is modified. If unchecked, the new instance copies the template's properties, but it is not made a replica, so it will not automatically update when the template is modified.

Also note that creating instances from the Project Bar will respect the *Propagate hierarchy* option of the template. If enabled and you create the root instance of a hierarchy from the Project Bar, then it will automatically also create the rest of the hierarchy as well.

## Other considerations

- By default, replicas do not use the position properties of the template. In most cases it is more useful for these two properties to remain individual for each replica.
- At runtime, replicas are no longer connected to their templates as they are in the editor. This means that changing a template at runtime will not affect the replicas. This applies for properties and hierarchy changes.

# EVENTS

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

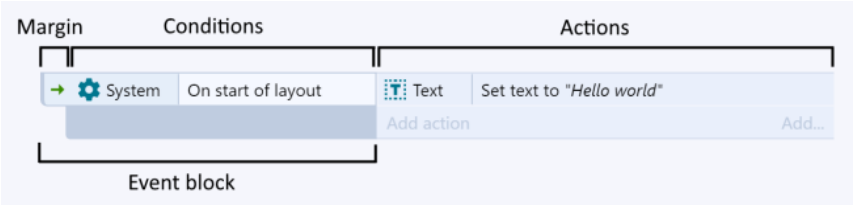
Events are one of Construct's main features: instead of complicated scripting or programming languages with fiddly syntax and difficult errors, you can define how your project works using a simpler block system. The blocks are collectively referred to as events, although there are several kinds of block making up an event sheet.

Construct's event system is likely to be unfamiliar to new users. Read How Events Work for a summary of how to use them. Events are not cookie-cutter blocks that limit what you can do. Instead they are designed to provide the basic tools you need to create sophisticated content.

The basic concept of events is that conditions **filter** the instances meeting the condition, then the actions run for *those instances only*. This allows you to control instances independently, especially when used with instance variables. A common misconception is that all instances of an object type must act the same. That is not the case: the fundamental way events work is to filter out individual instances, and run actions on just those that met the conditions.

Events are edited in the Event Sheet View using the Add Condition/Action dialog, the Parameters dialog and Expressions dictionary.

A diagram of a simple event is shown below. (This does not include every feature of events - the rest are explained in this section.)




**Events** typically consist of **conditions** that must be met, **actions** that run if so, and optionally further sub-events that test more conditions, run more actions, etc. A simple way to think about events is "If the conditions are true, then run the actions". However remember that a key feature is that it also filters the instances matching the condition. For example if the condition *Bullet collides with alien* is met, the action *Destroy alien* will run, and the *Destroy* action affects only the instance involved in the condition.

Once you are familiar with events, you will likely find it useful to also use Functions to help manage events as your project gets larger.

## Scripting

You can use Construct's event system exclusively and still make complex projects with sophisticated logic. However if you are interested in learning a programming language, you can

also use JavaScript coding in Construct. You can also mix and match code and event blocks, such as using a line of JavasScript code in the place of an action. To find out more see the manual section on Scripting.

If you already have programming experience, you may be interested in extending Construct using the Addon SDK.

# HOW EVENTS WORK

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/how-events-work

If you're new to Construct's events, this section will outline how they work. This is essential reading for beginners! You will be able to make much better and more reliable games with a thorough understanding of how events work.

To learn how to add and edit events, see Event Sheet View.

Events are designed to be easily readable and to intuitively "just work". However, they have specific, well-defined ways of working which is described here.

Events work by **filtering** specific instances that meet some conditions. The actions then run for *those instances only*. For example, consider the following event:

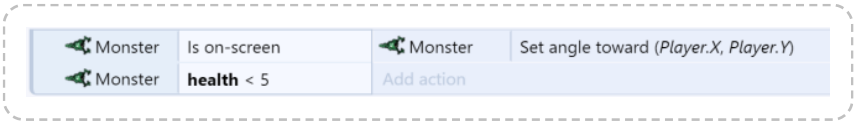



In this example, when a *Bullet* collides with a *Monster* the event condition is met. The specific instances of *Bullet* and *Monster* that collided in the game are "picked" by the event. Actions only run on the "picked" instances. If there are other instances of *Bullet* and *Monster* in the layout, they won't be affected by the *Subtract 1 from health* and *Destroy* actions. It would be very difficult to make good games if every bullet hurt every monster!

Another way to think about an event is "**If** all conditions are met **then** run actions on the instances meeting the conditions".

## Multiple conditions

Adding more conditions to an event progressively filters the instances to run actions on. For example:



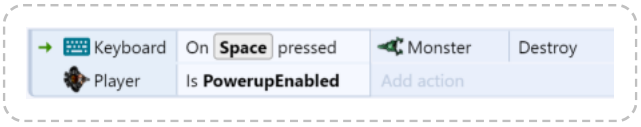

This event runs like this:

1. First all *Monsters* that are on-screen are picked.
2. Then, of those on-screen, it is reduced to those with less than 5 health.
3. The action makes all monsters that are **both** on-screen **and** have less than 5 health look directly at the player. Monsters that are off-screen or have 5 or more health are not affected.

Therefore, using multiple conditions you can run actions on just the instances meeting several criteria. Users from programming languages or other tools might recognise this as a logical "AND". All conditions of an event must be met for the actions to run. If no monsters are on-screen or none of those on-screen have less than 5 health, the actions do not run at all.

## Unreferenced objects

Have a look at the following event:




If the user presses `Spacebar` and the Player's *PowerupEnabled* flag is set, the action does *Monster: Destroy*. Note that there aren't any conditions that filter or pick Monsters in this event. In this case, *all* Monster instances are destroyed. In other words, if an event doesn't reference an object in its conditions, actions apply to all the instances of that object.

Think of conditions as starting with all instances being picked, and progressively filtering them from there. If there were no conditions, there are still all instances picked, so the action affects all of them.

## Picking resets between events

After an event ends, the next event begins from scratch. Its conditions will start picking from all instances again.

On the other hand, sub-events (which appear indented) **carry on from where its parent event left off**. A sub-event will further filter the instances left over by the event that came before it. If an event has two sub-events, they both pick from the same set of instances the parent left behind - the second sub-event is not affected by the first. In other words, events at the same indentation level always pick from the same set of instances, and events at a lower indentation level are always working with the instances handed down from above.

## The System object

In Construct the System object represents built-in functionality. It has no instances. This means most system conditions do not pick any instances: they are either true or false. If they are false the event stops running, otherwise the event continues without the picked instances having been changed. There are exceptions, though: if a system condition uses an object, such as *Pick random instance*, that will affect the picked objects.

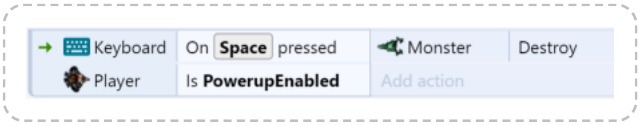
System actions do not run on any picked objects: they simply run if all of the event's conditions were met.

## Events run top to bottom

The order of events is important. Every event is checked once per tick (about 60 times a second on most computers), and they are run from top to bottom in the event sheet. The screen is drawn once every event has been run, then the process starts again. This means if one event does something and the next event undoes it, you'll never see that anything happened.

The same applies within events: conditions are checked from top to bottom, and the actions run from top to bottom.

However, **triggers** are an exception. See the green arrow to the left of *Keyboard: On Space pressed* from the previous example:



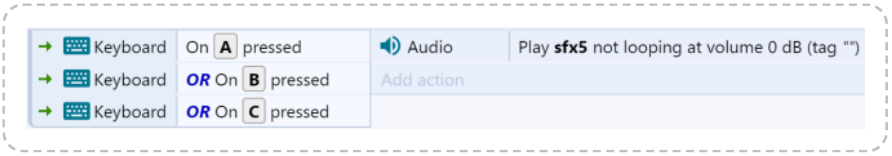

This indicates the event is **triggered**. Rather than running once per tick, this event simply runs (or "fires") upon something actually happening. In this case, the event runs when the user hits the Spacebar key on the keyboard. It is never checked any other time. Since triggers run upon an event happening, they aren't checked in top-to-bottom order like other events. This means the ordering of triggers relative to other events is not important (except relative to other triggers of the same type, since triggers still fire top-to-bottom).

There can only be one trigger in an event, because two triggers cannot fire simultaneously. However, multiple triggers can be placed in 'Or' blocks (see the next section).

## 'Or' blocks

As mentioned before, all conditions have to be met for an event to run. This is called a 'Logical AND', because "condition 1 AND condition 2 AND condition 3..." must be true. However, you can change an event to run when any condition is true. This is called a 'Logical OR', because the event will run if "condition 1 OR condition 2 OR condition 3..." are true.

Normally blocks work as 'AND' blocks. To make an 'OR' block, right-click the block and select Make OR block. It will then display with - or - between each condition, as shown below.



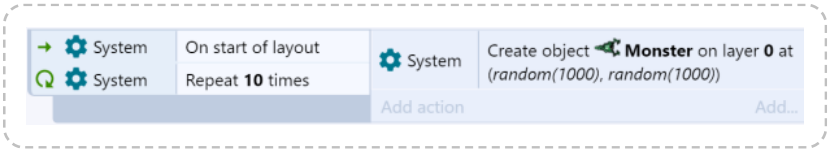

Note that because OR blocks run if any condition is true, it's possible the event will still run if some conditions were false and did not pick any instances. In this case the actions will still run, but possibly with zero instances picked for any objects where no instances met the condition. If any actions are run for objects with no instances picked, nothing happens.

Also note normally you can only put one trigger in an event, but you can put multiple triggers in an 'Or' block, and the event will run if any of triggers run.

You can combine the block types by using sub-events. This allows you to build up more advanced logic, such as an 'Or' block followed by an 'And' block.

## Looping conditions

Some events **loop**, which simply means they repeat more than once. Note the green circular arrow in the below example to indicate this.




This means when the layout starts, the *Create object* action repeats 10 times. The end result is 10 monsters are created at random positions in the layout on startup.

There can also be more conditions following the *Repeat* condition. These are tested on each of the repeats as well, and must be true for the actions to run. There can even be more than one loop in an event, but this is rare.

## Families and containers

Note families Paid plans only pick their instances entirely separately from any of the object types in the family. For more information, see the section *Picking families in events* in the manual entry on Families.

Containers are an advanced feature that can also make groups of instances always be picked together. For more information see the manual entry on Containers.

## Summary

Using this event system it's possible to make sophisticated logic for games quickly and easily. It is a very powerful alternative to scripting or programming languages but much easier for non-technical people to use.

Although this section has described the essential parts of the event system, it still has not covered everything. The rest of this manual section covers more features you can use in events. The reference sections also cover all the conditions, actions and expressions in Construct.

# EVENT SHEETS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/event-sheets

---

**Event Sheets** are lists of events. They are edited in the Event Sheet View. All the event sheets in a project are listed in the Project Bar.

## Adding and removing event sheets

To add a new event sheet, right click an event sheet folder in the Project Bar (such as the root level *Event sheets* folder) and select Add event sheet.

Event sheets can be renamed or deleted by right-clicking the event sheet itself in the Project Bar and selecting Rename or Delete.

When adding a layout, Construct will prompt to ask if you'd also like to create an event sheet for that layout.

## Sharing events between layouts

Layouts have an associated event sheet to define how the layout works. However it is often useful to use one event sheet for multiple layouts to prevent having to duplicate all your events. There are two ways to share events between layouts:

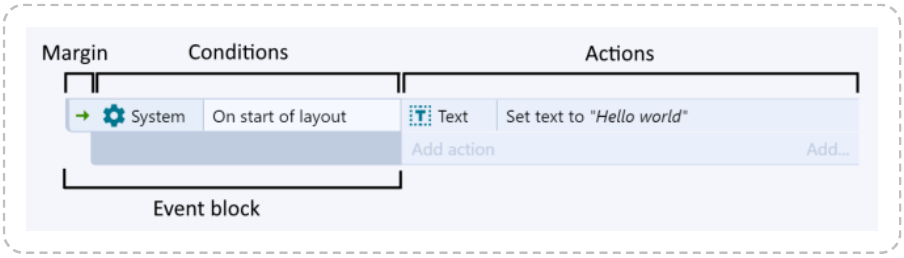
1. Set several layout's **Event sheet** property to the same sheet.
2. Make a separate event sheet with all the common events on it, then include that event sheet on other event sheets.

The second option is usually preferable since you are not forced to use exactly the same events for different layouts - you can add a few extra events to customise how it works depending on the layout.

# CONDITIONS

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

In events, **conditions** filter instances that meet some criteria. They appear to the left of the event. All conditions in an event must be met by at least one instance for the actions to run. The actions then only apply to the instances that met the conditions.




**System conditions** do not pick any instances: they are simply either true or false, unless they specifically reference an object, such as with *Pick random instance*.

There are three kinds of conditions: normal conditions, **triggered** conditions, and **looping** conditions. You can also create OR blocks which run if *any* condition is true, rather than all the conditions. See How events work for more information.

## Adding and editing conditions

When you add a new event, you are taken through the process of adding the first condition for the event. This is described in more detail in the Event Sheet View manual entry.

You can add multiple conditions to an event block. To add another condition, right-click either an existing condition or the event margin and select Add another condition. All conditions must be met for the event to run, unless you set the event to be an OR block, in which case *any* condition can be true for the event to run. To set an OR block, right-click the event margin and select Make OR block.

To edit a condition, **double-click** it. You can also **right-click** it and choose Replace or Delete.

## Inverting conditions

Conditions can be **inverted**, which flips the thing they test. For example, the condition *Monster is overlapping Player* is true whenever a monster is touching the player. However, if inverted, it appears with a red invert icon and means *Monster is **not** overlapping Player*.



Not all conditions can be inverted (e.g. triggers can't be, because the event doesn't make sense inverted in that case).

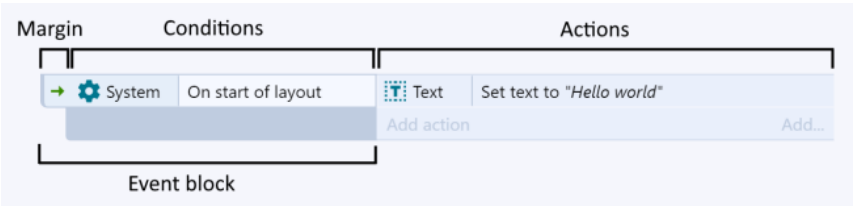
## Breakpoints

Paid plans only It is possible to place a breakpoint on a condition, to pause execution when it is reached in the debugger. For more information, see breakpoints.

# ACTIONS

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

In events, **actions** do something, such as create an object or go to another layout. They appear to the right of the event.




Actions only affect the instances that met the event's conditions. See How Events Work for more information.

To add an action to an event, click the **Add action** link that appears beside the event underneath any existing actions. For more information on adding and editing events, see Event Sheet View.

## Breakpoints

Paid plans only It's possible to place a breakpoint on an action, to pause execution when it is reached in the debugger. For more information, see breakpoints.

# EXPRESSIONS

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

In events, **expressions** are used to calculate sums or retrieve information from objects, such as a Sprite's X co-ordinate. Expressions are entered in to the Parameters dialog when adding or editing a condition or action which has parameters. The Expressions dictionary is also shown at the same time and provides a dictionary of all the system and object expressions available in a project.

Some examples of expressions, which can range from a simple number to a complex calculation, are given below:

- `0`
- `random(360)`
- `Sprite.X`
- `(Player1.X + Player2.X) / 2`
- `Sprite.8Direction.Speed`
- `Sprite.X + cos(Sprite.Angle) * Sprite.Speed * dt`

## Numbers

Numbers are simply entered as digits with an optional fractional part separated by a dot, e.g. `5` or `-1.2`. Fractional numbers may begin with a dot, e.g. `.5`.

## Text (strings)

Text is also known as *strings* in software development, and Construct also sometimes uses this naming convention. Text in expressions should be surrounded by double-quotes, e.g. `"Hello"`

The double-quotes are not included as part of the text, so setting a text object to show the expression `"Hello"` will make it show **Hello**, without any double-quotes. To include a double-quote in a string, use two double-quotes next to each other (""), e.g. `"He said ""hi"" to me"` will return **He said "hi" to me**.

> *Using quotes for strings only applies to expressions. Don't use them in other places like in property values in the Properties Bar.*

You can use `&` to build strings out of mixed text and numbers, e.g. `"Your score is: " & score`

To add a line break to a string use the system expression **newline**, e.g. `"Hello" & newline & "world"`

The **StringSub** system expression is also useful for building strings. You can use it like this: `StringSub("Hi {0}, your score is {1}!", "Sam", 100)` which will produce the string **Hi Sam, your score is 100!**. The first string can use *placeholders* like `{0}`, `{1}`, `{2}` etc. and these are replaced by the following parameters. You can have as many placeholders as you like and they can appear anywhere in the first string, so long as the appropriate number of parameters follow on.

## Operators

You can use the following operators in expressions:

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulo (remainder after division) |
| ^ | Raise to power, e.g. 5 ^ 2 = 25 |
| & | Build strings, e.g. `"Your score is: " & score` |
| =, <>, <, <=, >, >= | Comparison operators, e.g. `score < 10`. Return 1 if comparison is true or 0 if false. |
| ?: | Conditional operator, in the form `condition ? result_if_true : result_if_false`. Allows testing conditions in expressions. The condition counts as true if it is non-zero, and false if it is zero. E.g. `score < 0 ? "Game over!" : "Keep going!"` |
| &, \| | When used on numbers, `&` is logical AND and `\|` is logical OR. (Note if either side is a string, `&` instead does string concatenation.) These are useful combined with the comparison operators, e.g. `score < 0 \| health < 0`, which returns 1 if either condition is true, else 0 for false. |

*Note a common mistake is to write comparison expressions like `value = 1 | 2` with the intent to match `value` to either 1 or 2. However this doesn't work as it is actually evaluated as `(value = 1) | 2`, which always evaluates as true. Similarly `value = (1 | 2)` won't work as `1 | 2` evaluates to true, so it only tests if `value` is true. The correct way to test this is using `value = 1 | value = 2`.*

## Object expressions

Objects have their own expressions to retrieve information about the object. These are written in the form `Sprite.X` (the object name, a dot, then the expression name). The Expressions dictionary lists all the available expressions in the project, and they are further documented in the reference section of the manual.

The expression `Self` can be used as a short-cut to refer to the current object. For example, in an action for the *Player* object, `Self.X` refers to `Player.X`.

You can add a zero-based object index to get expressions from different object instances. For example `Sprite(0).X` gets the first Sprite instance's X position, and `Sprite(1).X` gets the second instance's X position. For more information see index IDs (IIDs) in instances. You can

also pass another expression for the index. Negative numbers start from the opposite end, so `Sprite(-1).X` gets the last Sprite's X position.

## Behavior expressions

If an object has a behavior with its own expressions, they are written in the form `Object.Behavior.Expression`, e.g. `Sprite.8Direction.Speed`.

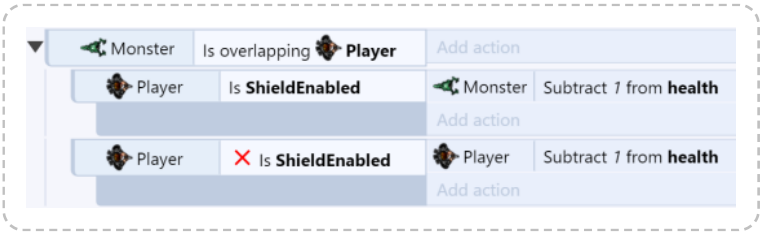
## System expressions

The built-in system expressions are listed in the reference. These include some basic mathematical functions like `sqrt` (square root).

# SUB-EVENTS

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

**Sub-events** appear indented beneath other events. They **carry on picking instances** from where the "parent" event left off. They run after the parent event's actions have finished. An example is below.




This event runs like so:

1. Test if any *Monster* instances are overlapping *Player*. If so, the instances involved are remembered.
2. The top event's actions would run next, but it doesn't have any.
3. The second event (note it is indented) then tests if the *Player*'s **ShieldEnabled** instance variable is set. If so, it subtracts 1 from the *health* of the Monster overlapping the player.
4. The third event (also indented) tests if the *Player*'s **ShieldEnabled** instance variable is not set (see inverting conditions). If so, it subtracts 1 from the health of the *Player*.

In other words, monsters hurt the player when they touch, unless the player's **ShieldEnabled** instance variable is set, in which case the monsters are hurt instead.

This works because the objects picked by the top event are remembered and also used for sub-events. If the second and third events were not sub-events (not appearing indented) the second event would subtract 1 from the health of *all* Monsters, because it was not referenced in the event. (See *Unreferenced objects* in How events work for more on this.)

Sub-events can have other sub-events too, which makes sub-events very powerful and flexible for setting up advanced game logic.

Note sub-events run after the actions only if the actions run - in the above example, if no monsters are overlapping the player, neither the actions of that event nor its sub-events run at all.

## Triggers in sub-events

If a trigger is in a sub-event, **all** of its parent event's conditions must be true at the time the trigger fires, otherwise the event will not run.

There can also only be one trigger in a single branch of sub-events. In other words, one of the events above a trigger cannot also contain a trigger.

## Adding and removing sub-events

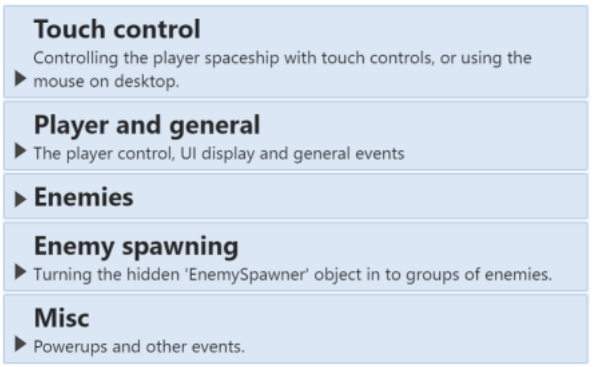
To add a sub-event, **right-click** the event margin and choose Add▶Add sub-event. Alternatively choose the Add sub-event option from the **Add...** link on the right, or press the S keyboard shortcut.

Adding and editing conditions to sub-events works identically to ordinary events. You can also create more deeply nested sub-events by adding a sub-event to a sub-event.

# EVENT GROUPS

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

**Groups** of events are mainly for organising events. They can be collapsed and expanded using the arrow button in the title. They are edited with the Event Group dialog. To add an event group, right click an event or empty space in the event sheet and select Add group or press the `G` keyboard shortcut.




*Some event groups from the Space Blaster game example in Construct 3*

Events can be **dragged and dropped** in to and out of groups (be sure to drag from the event margin, and not from a condition or action). Entire groups can also be enabled or disabled with the *Set Group Active* system action, which is useful for situations like pausing the game. Disabling unnecessary groups can also help improve performance.

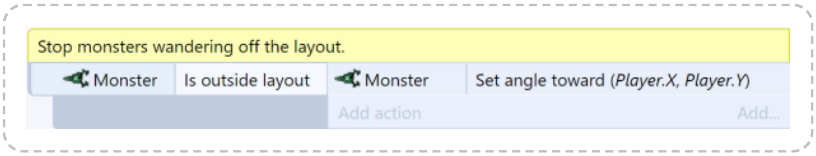
## Customising colors

You can customise the event group header's colors by **right-clicking** and selecting Colors▶Change text color or Colors▶Change background color. To reset back to the default colors (which are based on the current theme), select Colors▶Restore default colors.

# COMMENTS

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

Event **comments** are simply notes to help you remember how events work and what they do. They are very important to help you remember how things work in large projects! By default comments have a yellow background and appear above the event they are describing.




To add a comment, **right-click** on an event or an empty space in the event sheet and select Add comment or press the `Q` keyboard shortcut. Comments can be edited by **double clicking** on them. You can make a comment with a line break by holding shift and pressing Enter.

Comments can also be added in between actions. The `Q` keyboard shortcut will add an action comment if an action is selected. Alternatively you can right-click an existing action and select *Add comment*, or use the *Add...* menu next to the *Add action* link.

If you use Construct a lot, you will find comments essential to help yourself organise and understand large projects. Coming back to a project after a few months with no comments at all can be very difficult, so don't underestimate the importance of comments.

Comments do not affect how anything works at all. They are solely for your information. Nothing typed in to comments is exported to the game whatsoever.

## Customising colors

You can customise the comments's colors by **right-clicking** and selecting Colors▶Change text color or Colors▶Change background color. To reset back to the default colors (which are based on the current theme), select Colors▶Restore default colors.

## Using formatting

Comments can include some simple formatting tags known as "BBCode". This involves using square bracket tags such as [b] and [/b] around text to make bold, e.g. `[b]bold text[/b]`. The tags you can use in comments are listed below.

| Tag | Description |
|---|---|
| [b]...[/b] | **Bold text** |
| [i]...[/i] | *Italic text* |
| [s]...[/s] | ~~Strikethrough text~~ |
| [u]...[/u] | Underline text |
| [sub]...[/sub] | Subscript text |
| [sup]...[/sup] | Superscript text |
| [small]...[/small] | Smaller text |

| Tag | Description |
| --- | --- |
| [mark]...[/mark] | Mark text with highlight |
| [code]...[/code] | Format as code snippet |
| [h1]...[/h1] | Header 1 |
| [h2]...[/h2] | Header 2 |
| [h3]...[/h3] | Header 3 |
| [h4]...[/h4] | Header 4 |
| [item] | Item bullet point: • |

# EVENT SHEET INCLUDES

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

Event sheet **includes** allow you to include an event sheet on another event sheet. They are a sort of automatic copy-paste of one sheet's contents to another. This is useful for sharing events common to multiple layouts. For example, a game could use different event sheets for different parts of the logic. You could have event sheets called *Player input*, *Enemy control*, *Effects*, and so on. Then each level of the game can include these common event sheets in the layout's own event sheet. This allows you to share the same events between multiple layouts, without having to copy and paste all your events over and over again.




Includes can be added from right-clicking an empty space in the event sheet and selecting Include event sheet or press the N keyboard shortcut.

Construct automatically prevents cyclic includes. If two event sheets both include each other, Construct will use both event sheets but not include either more than once.

# EVENT VARIABLES

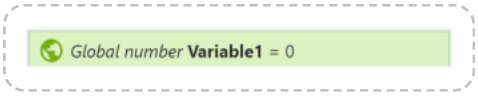
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/variables

**Event variables** are number or text values which are either **global** to the whole project or **local** to a range of events. They are modified using the Event Variable dialog. To add an event variable, **right-click** on an event, another variable, or an empty space in the event sheet, and select Add global variable or Add local variable, or press the **V** keyboard shortcut. Variables at the root level of the event sheet (not indented beneath anything else) become global variables, whereas variables in groups or sub-events become local variables.

Event variables are modified with the system actions in the *Global & local variables* category. They can be retrieved by simply using their name in expressions.

## Global variables

Global variables show a globe icon. They are always at the top level of an event sheet - they are not sub-events or inside any groups.

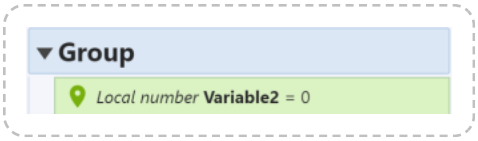



Global variables store their values between layouts. Events in any layout can access any global variable, even if it was created in a different event sheet that is not included.

Global variables can be moved to another event sheet by cutting and pasting them. After being cut, references to the global variable will disappear because it has been removed; this is normal and nothing to worry about. When you paste the global variable, the references that disappeared will reappear again. Alternatively you can **right-click** the global variable and select Move to event sheet....

## Local variables

Local variables are variables placed nested under other events, or inside a group. They also show with a different icon to global variables.




The main difference between global and local variables is **local variables can only be accessed in their scope**. A local variable's scope is its level of sub-events. All other events at the same level of indentation, or lower levels, can access the local variable. Events *above* it (less indented) *cannot* access the local variable.

For example, if an event variable is in a group of events, it becomes a local variable. Then, it will only appear as an option for a variable in events inside that group. In other groups or in other event sheets it does not appear at all and cannot be accessed. This makes the variable *local* to the *scope* in which it is placed.

Local variables convenient for temporarily holding variables over a short range of events, such as to calculate an average value (where a temporary *sum* variable may be necessary). It also helps keep the project simple, since it prevents the need to create more global variables, which appear everywhere in the project even if they are not needed everywhere.

The scope of local variables is designed to mimic how the scope of variables works in real programming languages.

### Function parameters

Function parameters are a special kind of local variable, scoped to a function event. For more information see the section on Functions.

## Static and constant variables

By default, local variables reset to their initial value whenever entering their scope (usually every tick), like local variables in programming languages. If the variable is marked *static* in the Event Variable dialog it will persist its value permanently, like a global variable.

Both global and local variables can be marked *constant*. This makes them read-only: they can be retrieved and compared, but not changed.

## Finding references to variables

Paid plans only You can quickly see a list of all references to a global or local variable by **right-clicking** it and selecting Find all references.... This will open the Find Results bar with a list of all places in the project the variable is used. This is also helpful for identifying if there are no references so the variable can be safely deleted.

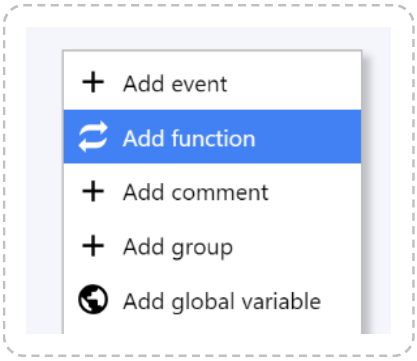
# FUNCTIONS

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

**Functions** are special kinds of event blocks that can be called from actions. They are designed to be analogous to functions in real programming languages. Using functions can help you organize event sheets and avoid having to duplicate groups of actions or events.

## Adding a function

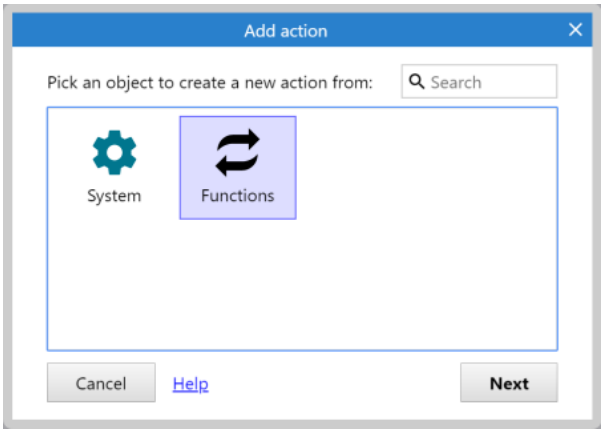
In the event sheet, functions are represented as a different type of event block. To create one, use the *Add function* menu option instead of *Add event*.



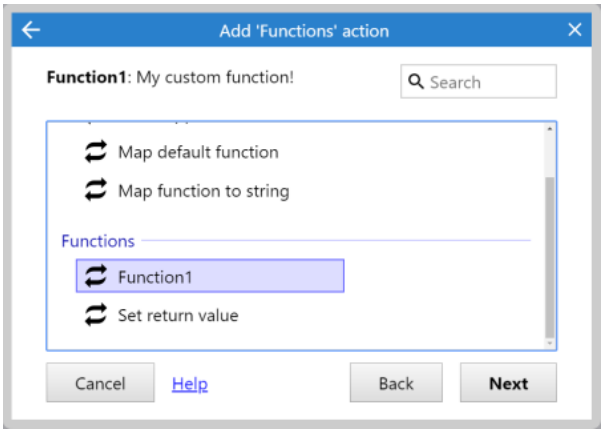

When you select this the Add function dialog will appear for you to fill in details about the function. Once created, the function appears in the event sheet similar to a normal event, but with a special function icon and *On function* text at the top.



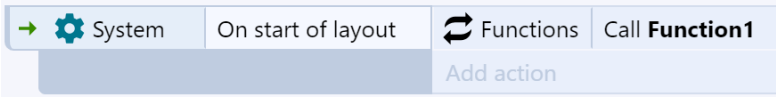

You can add conditions, actions and sub-events to functions, just like you can with normal events. However functions do not run unless you call them in an action. Once you've added a function to your project, a new special *Functions* object appears in the Add action dialog, next to the System object.




When you choose this object, it displays the functions in your project as if they are actions. (There are also some other built-in actions that relate to functions.)




Choosing the function adds an action that calls (runs) the function.

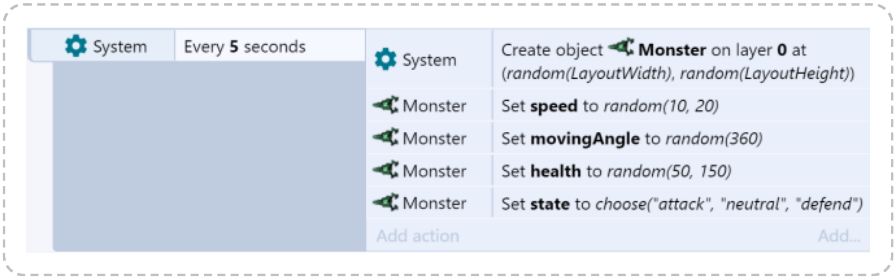



Functions are similar to custom actions, but not associated with a specific object type or family. This action will run the corresponding *On function* event, including testing its conditions, running actions, and running any sub-events, and then return to the original action and continue from where it left off.

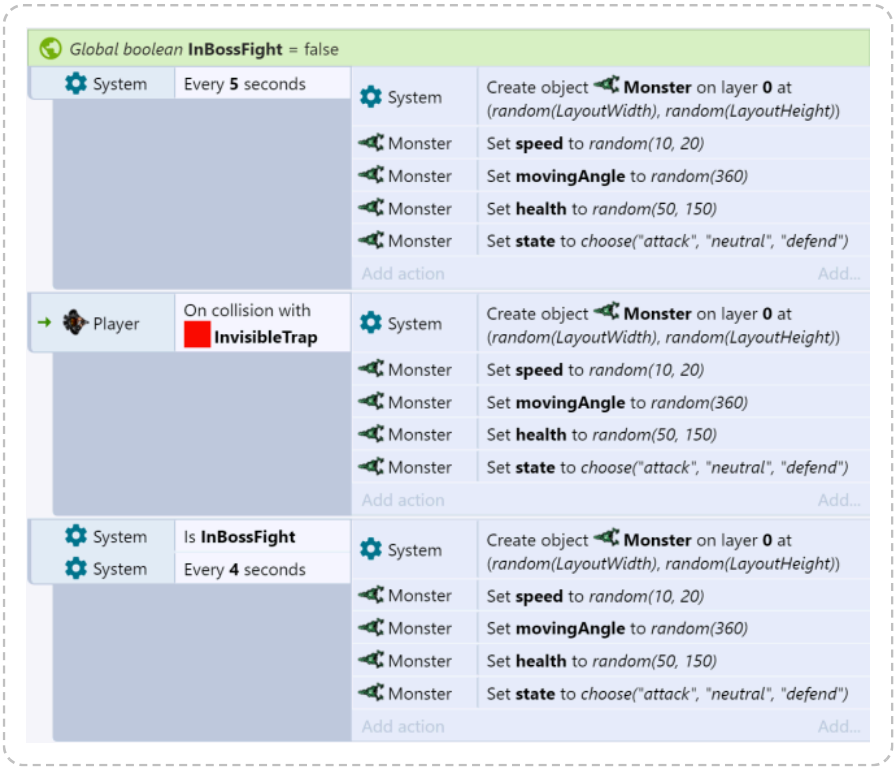
> *Functions are global. This means you can call a function from anywhere in your event sheets, even if the function is in a different event sheet that is not included in the event sheet you call it from.*

## Using functions

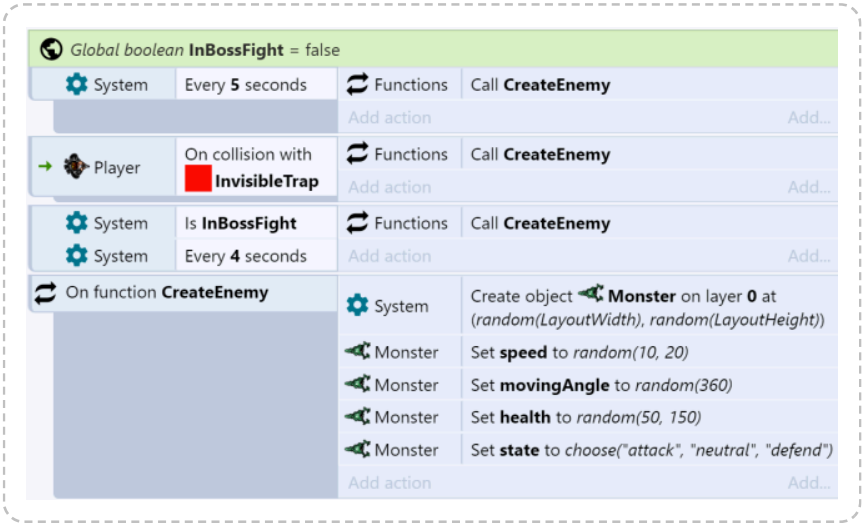
A good example of using functions is to eliminate repeated sets of actions or events. For example suppose you create an enemy with random properties every 5 seconds using this event:




Suppose there are two other events where you want to create an enemy the exact same way: one when a player walks in to a trap, and another one every 4 seconds when in a boss fight. Without functions, you may have to copy-and-paste the same actions multiple times, like this:




Notice this is becoming inconvenient. There may be times you need to repeat the actions in even more places. If you want to make a change, you then have to find every place you repeated the actions, and repeat the change. We can remove the repetition using functions. By creating a *CreateEnemy* function which has the repeated actions, we can replace all the repeated actions with function calls, like this:



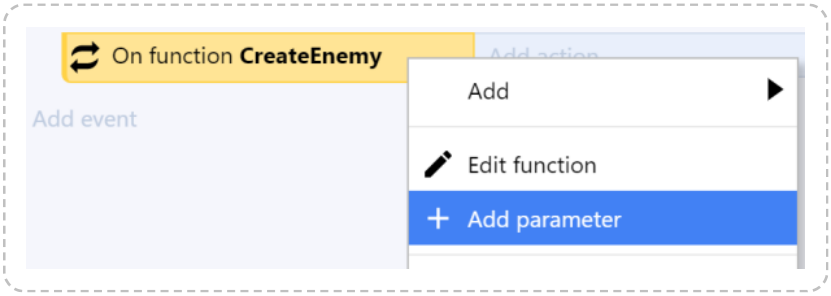

This works identically to the previous events, but is much shorter and more convenient. We can call the *CreateEnemy* function anywhere in our events we want to create an enemy, and it uses the same set of actions in the corresponding *On function* event.

It is often useful to split many parts of your events in to functions like this, so they can be conveniently re-used across event sheets.

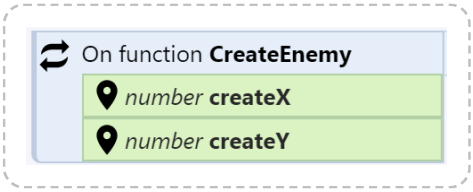
## Parameters

When calling a function, you can also pass parameters. These are numbers or strings that are made available to the function. For example, the *CreateEnemy* function from the previous example could be modified to take two parameters: the X and the Y co-ordinates at which to create the enemy. This helps functions to be made more general purpose by using extra information from the action calling the function.

To add a parameter to a function, use the *Add parameter* menu option when right-clicking the function. (Note you need to right-click on the header or margin, since if you right-click a condition, it will show a menu for the condition instead.)

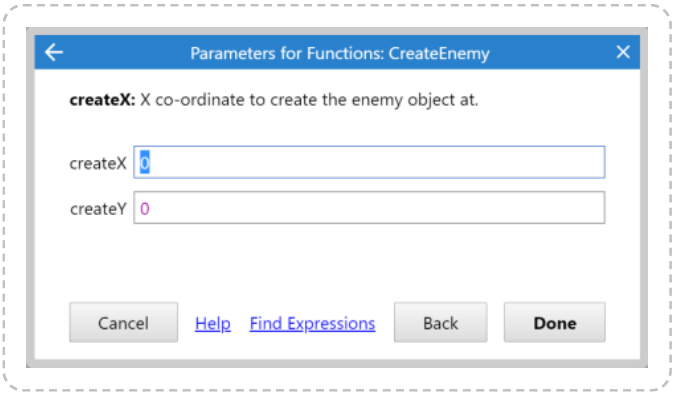



When you select this the Add function parameter dialog appears for you to fill in details about the parameter, including its name, description and type. Parameters appear similar to local variables, but inside the function block.



Parameters work very similarly to local variables - you can use them in expressions, compare them, and set them just like any other kind of local variable. Similar to local variables they are limited in scope to just the function event and its sub-events.

Now when you call the function, you can also provide the parameters. Notice the name and description you set for the parameters are used. These appear like parameters for any other action, but they will set the values of the parameters when calling the function.




## Returning values from functions

Functions can also return a result. For example, a factorial function could calculate the mathematical result and return it.

By default, functions have a return type of *None*, meaning they don't return any value. This also means they are used as actions. However if you set a return type of *Number*, *String* or *Any*, the function returns a value. This also means it is used as an expression instead, so it won't appear as an action.

A function can set its return value using the *Set return value* action in the built-in Functions object. It can then be called using it as an expression, such as:

```
Functions.MyFunction
```

Parameters can also be added in parentheses, e.g.:

```
Functions.MyFunction(1, 2, 3)
```

The expression returns the value set by the *Set return value* action in the function call.

Functions which return a value will also appear in the Expressions dictionary, also show up in autocomplete, and also show call tips when entering parameters, just like other expressions. In summary, while functions with no return type are essentially custom actions, functions with a return type are essentially custom expressions.

## Picking

Normally, calling a function will run the function with picking reset. That means if an event picks some instances with conditions, then calls a function, the function runs with all instances picked again, ignoring the fact that conditions previously picked some instances.

Enabling *Copy picked* on the function changes this so the function keeps the same picked instances when it is called. This can be convenient for making a function that affects a single instance, for example - its actions will run on the instance picked by the caller, rather than having to pick the instance another way (e.g. by its UID).

Note that if the function changes which instances are picked with its own conditions, that does not affect the place that called the function. When returning after the function has finished, any changes the function made to picking are discarded. In other words, calling a function does not affect the calling event's picking, even if *Copy picked* is enabled.

## Asynchronous functions

A function can be set to *Asynchronous* (or *async* for short) in the Add/Edit Function dialog. This allows it to be used with the System *Wait for previous actions to complete* action. This means if the function does any waiting itself, such as with an action like *Wait 3 seconds*, the caller can also wait for the function call to complete with *Wait for previous actions to complete*.

> *Make sure the asynchronous function ends with a Wait for previous actions to complete action if it uses async actions. This ensures all async actions have been completed before the function itself is marked as having completed. Alternatively you may want to leave some async actions at the end that are not waited for, allowing the async function to finish without waiting for the last actions.*

Note this imposes a small performance overhead, so for best performance leave it disabled if you don't need it.

## Function maps

Sometimes it's useful to be able to call a function depending on a string determined at runtime. The function maps feature allows for this. Try out the Function Maps example to see how it works.

## Nesting and recursion

Like in programming languages, functions support calling functions from other functions, and functions calling themselves (recursion). Functions calling other functions or recursing create a new call stack entry with their own unique variables. In other words, like in programming languages, local variables and parameters are unique at each level of function call. This does not apply to static local variables or global variables.

## Renaming the Functions object

When using functions, you'll notice actions and expressions refer to a built-in "Functions" object. This can be renamed in case you want to change it, but it does not appear in the Project Bar like other objects, so it has to be renamed a different way. There are two ways to rename it:

1. When adding an action, at the first step when you choose an object, right-click on *Functions* and click *Rename*.
2. Right-click on the name of the project in the Project Bar, and then choose Tools▶Rename Functions object. Note this will only appear if there are functions in your project.

# JavaScript integration

## Scripting

When using scripts in Construct, use runtime.callFunction() to call an event function from script.

## External calls

In other cases, it is strongly recommended to use the Addon SDK to integrate JavaScript code with Construct. However it is possible to trigger a function from JavaScript using the following function:

```
c3_callFunction("name", ["param1", "param2"]);
```

*Do not call this with Construct's scripting feature. It will not work correctly. In that context you must use* `runtime.callFunction()` *instead. This method only applies to external JavaScript.*

The function with the given "name" is called by this method. Parameters are optional and can be omitted, but must be provided as an array in the second argument, and parameters may only be string, number or boolean values. The method also returns the return value set in Construct (if any), and also can only return a string or number.

*If the project is running in a Web Worker with the Use worker setting, this method is still available on the DOM. However it instead returns a Promise resolving to the return value, and asynchronously calls the function by posting a message to the Web Worker. In an async function,* `await c3_callFunction(...)` *will always work.*

# CUSTOM ACTIONS

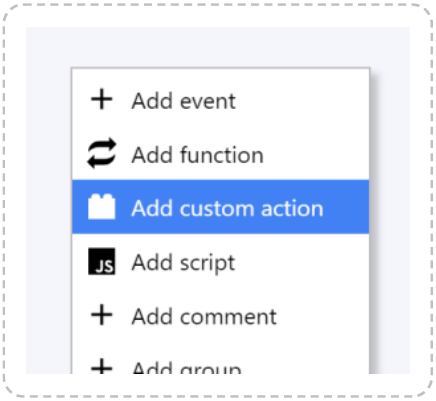
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/custom-actions

**Custom actions** are special kinds of event blocks that can be called from an action in an associated object type or family. They work similarly to functions, so it is useful to understand how functions work first before reading about custom actions.

Using custom actions can help you organize event sheets and avoid having to duplicate groups of actions or events. Custom actions also have more advanced uses when added to families, allowing for members of the family to override or extend a family custom action.

## Adding custom actions

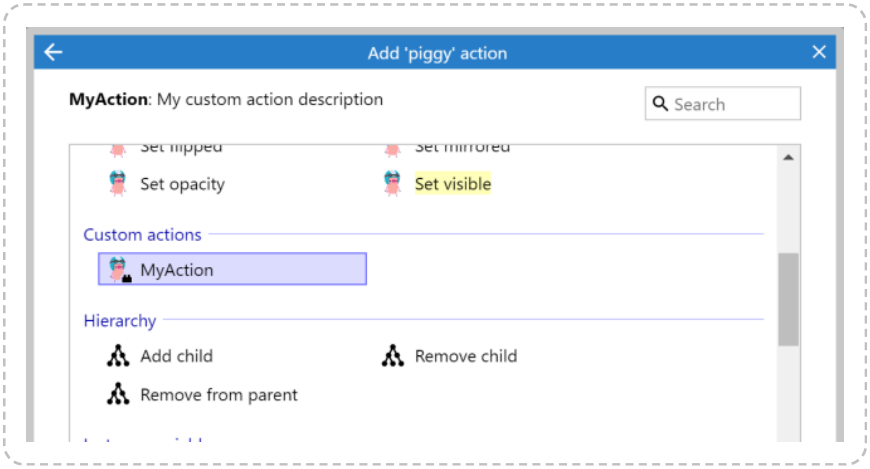
In the event sheet, custom actions are created by adding a special kind of event block. To create one, use the *Add custom action* menu option instead of *Add event*.



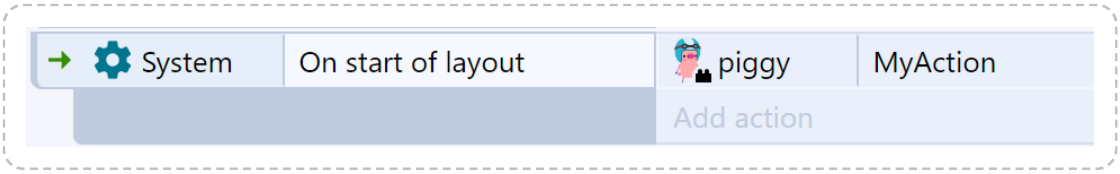

When you select this the Add custom action dialog will appear for you to fill in details about the custom action. Once created, the custom action appears in the event sheet similar to a normal event, but with a special icon and descriptive text at the top. This is referred to as the *custom action block*.




You can add conditions, actions and sub-events to custom action blocks, just like you can with normal events. However custom actions do not run unless you run them as an action in its associated object type or family. Once you've added a custom action block to your project, it will appear in the Add action dialog alongside all the other object type or family's usual actions.




Choosing the custom action from the *Add action* dialog adds an action that calls (runs) the associated custom action block.




Running the custom action action will run the corresponding custom action block, including testing its conditions, running actions, and running any sub-events, and then return to the original action and continue from where it left off.

> *Custom actions are global. This means you can use custom actions anywhere in your event sheets, even if the corresponding custom action block is in a different event sheet that is not included in the event sheet you call it from.*

## Parameters

Much like functions, custom actions can also use parameters. Since these work the same as with functions, refer to the section on *Parameters* in the Functions manual entry for more details.

## Picking

When running a custom action, the custom action block is run with the same instances picked as the calling event block. For example this means running a custom action in a *On object clicked* trigger will run the custom action block with just the clicked instance picked. This means custom actions automatically alter just the picked instances, much like normal actions. However when the custom action block finishes running, any changes to the picked instances it made are discarded, so it does not affect the running of the original event that called it.

The *Copy all picked* setting of the custom action block can alter how this works. Normally only instances of the custom action block's object are automatically picked. However if *Copy all picked* is checked, the custom action block will inherit *all* picked instances from the calling event

block - including other object types and families. This makes it work similarly to a function with *Copy picked* enabled.

## Asynchronous custom actions

Much like functions, custom actions can also be made asynchronous, so they can be used with the system *Wait for previous actions to complete* action. Since this feature works the same as with functions, refer to the section on *Asynchronous functions* in the Functions manual entry for more details.

## Family custom actions

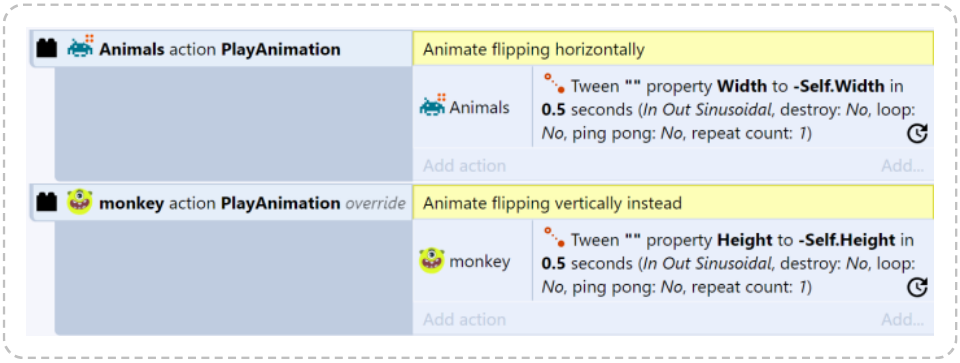
A custom action can be created for a family. This allows some more advanced uses of custom actions.

Much like with inheriting family instance variables, behaviors and effects, family custom actions can also be used as actions for every object type in the family. This allows every member of the family to share the same custom action.

### Overrides

When a family has a custom action, it's still possible to create a custom action with the same name for a specific object type in that family. In that case, the object type's custom action *overrides* the family custom action.

For example suppose there are three sprite object types named *Piggy*, *Octopus* and *Monkey* in a family named *Animals*, and there is a custom action named *PlayAnimation* for the family *Animals*. A custom action named *PlayAnimation* can also be added for *Monkey*. Then when running the family custom action *PlayAnimation*, the family custom action block will run for *Piggy* and *Octopus* instances, but the *Monkey* custom action block will run instead for *Monkey* instances.

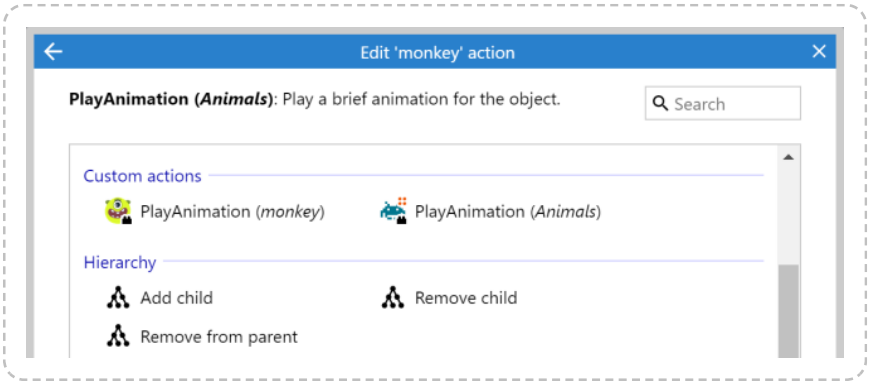



This allows specific object types in the family to override what a custom action will do for instances of that object type.

### Choosing overrides

Consider the previous example with the *Monkey* object type having an override for the family custom action *PlayAnimation*. In this case, the action list for *Monkey* includes both actions,

listing *PlayAnimation (Monkey)* and *PlayAnimation (Animals)*.




This allows choosing which custom action is run: the *Monkey* variant will run the override custom action for *Monkey*, and the *Animals* variant will run the original family custom action ignoring the override. This allows you to explicitly choose whether to run the override or the original family custom action for the *Monkey* object type.

## Extending a family custom action

By default, an override custom action will entirely replace the family custom action. However it's also possible to make it *extend* what the family custom does. This can be done by using the ability to choose overrides to add an action to the override custom action block that calls the original family custom action block. (In programming languages, this is sometimes referred to as a "super" call.)

Continuing the previous example, the *Monkey* custom action block can add a *Monkey* action to run *PlayAnimation (Animals)*, which is the original family custom action.




Therefore when running the *Monkey* custom action override, it will first do the original family custom action, and then do its own actions after that. This allows extending what the family custom action does to do additional things for specific members of the family, rather than completely replacing the custom action.

> *Be sure to get the right action in this case. If you accidentally call the Monkey custom action again from the custom action block, it will create an infinite loop of calling the custom action repeatedly. To run the family custom action, be sure to add an action for the same object type as the custom action block, and then choose the family variant of the custom action.*

# BREAKPOINTS

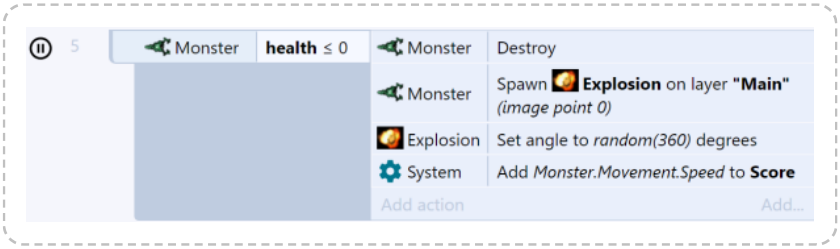
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/breakpoints

Paid plans only Breakpoints are an advanced feature that allow you to pause execution of the event sheet on a specific event, condition or action when running in the debugger. This can be a significant aid to debugging, since the full capabilities of the debugger can be used while stepping through events, conditions and actions one-by-one.

## Setting a breakpoint

Breakpoints can be toggled on and off for the selected event block, condition or action by **right-clicking** them and selecting Toggle breakpoint or pressing the `F3` keyboard shortcut. Breakpoints can also be toggled while debugging.

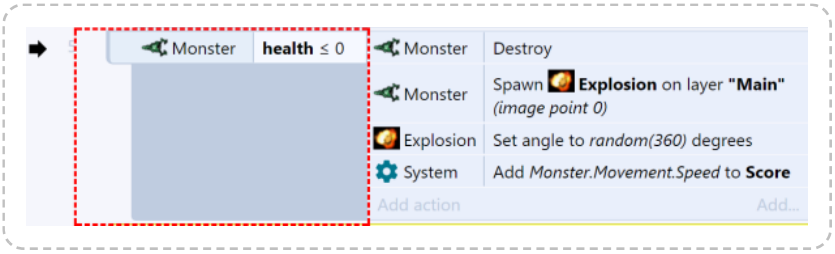
When a breakpoint is set on an event, condition or action, a breakpoint icon appears beside it.




## Pausing on a breakpoint

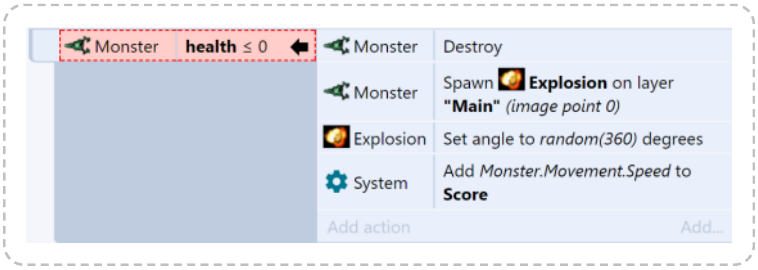
When running the debugger, the project will automatically pause *just before* it runs an event, condition or action with a breakpoint.

For events, this means it pauses just before it tests the first condition. This means a top-level event with a breakpoint will pause every tick, since the event engine reaches it every tick to test its conditions. It is usually more useful to place event breakpoints on sub-events, since they will only pause when the parent events have been run. When paused on a breakpoint, the event has a dashed outline and the icon changes to an arrow.

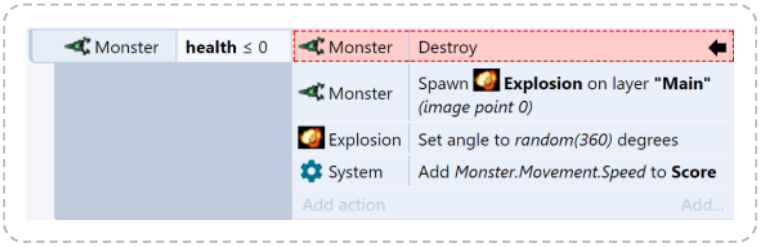



For conditions, this means it pauses just before it tests the condition. Since the condition has not yet been tested, the debugger does not know whether the condition will return true or false; you must resume execution to be able to tell. Note that Construct bails out of events early if any

condition is false. This means a breakpoint on the second condition won't pause the project if the first condition is false, since Construct will immediately skip to checking the next event. When paused on a condition, it is also indicated with a dashed outline, an arrow and also a changed background color.




For actions, this means it pauses just before the action runs. Placing a breakpoint on the first action in an event is often more useful than placing a breakpoint on the event itself, since it will only pause when all the conditions have been met and before any actions have run, as opposed to just before it starts checking any conditions. When paused on an action, it is indicated similarly to a condition.




## Continuing execution

When paused on a breakpoint, the debugger can be used as normal to inspect or edit values. However the *Pause* and *Step* buttons change in to **Continue** and **Next**.

Pressing **Continue** resumes until the next breakpoint is hit. Remember you can still toggle other breakpoints while paused on a breakpoint.

Pressing **Next** will step to the next event, condition or action in sequence in the event sheet. Alternatively, press `F10` with the browser window focused. This is useful for stepping through an event action-by-action and watching what happens in the debugger's inspector.

## Limitations

Unfortunately due to the architecture of the Construct engine, breakpoints cannot be placed inside some (but not all) triggered events. When not supported, this also means breakpoints cannot be anywhere inside sub-events to the triggered event.

## Using breakpoints

Breakpoints can be incredibly useful to learn more about how your own events work and to help track down problems. They may take some getting used to in order to get the most out of them, but once familiar they can be indispensable.

It is especially useful to have a multi-monitor setup when using breakpoints, or with the debugger in general. This allows you to have Construct showing which event it has reached on one screen, and the project running on another screen.

# TIMELINES

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

There are several different elements that make up a timeline. When any of them are selected in the Timeline Bar, their properties will be shown in the Properties Bar. Some of the properties only affect the timeline element that owns them, but there are others that also affect other elements below them in the hierarchy.

The hierarchy is as follows:

- **Timelines** contain Track Folders, Tracks, Value Tracks, Timelines and Timeline Folders
- **Timeline Folders** contain Timelines and Timeline Folders
- **Track Folders** contain Track Folders, Tracks and Value Tracks
- **Tracks** contain Property Track Folders, Property Tracks and Master Keyframes
- **Value Tracks** contain a single Property Track and Master Keyframes
- **Property Track Folders** contain Property Track Folders and Property Tracks
- **Property Tracks** contain Property Keyframes
- **Master Keyframes** contain Property Keyframes (which share a parent Track and position in the timeline)
- **Property Keyframes** are always the last elements in the hierarchy and don't contain anything

> *In the specific case of the **Ease** and **Path mode** properties, master keyframes takes precedence over the corresponding property track.*

It is useful to remember that structure when making changes to the following properties of a timeline element:

- **Animation mode**
- **Result mode**
- **Ease**
- **Path mode**
- **Enabled**
- **Visible**
- **Locked**
- **Show UI Elements**

### Explicit property changes

Making a change to **Enabled**, **Visible**, **Locked** or **Show UI Elements** properties will trigger changes down the element's hierarchy, explicitly modifying all children.

### Inherited property changes

**Animation mode**, **Result mode**, **Ease** and **Path mode** properties follow an inheritance pattern. This means that the special **"Default"** value means to use the value defined by the element immediately above it in the hierarchy.

In the case of **Animation mode** and **Result mode** the value **"Default"** can be used at the timeline level but since it is the top most element in the hierarchy it has a different meaning. If the special **"Default"** value is used at the timeline level and all previous elements in the hierarchy were using the **"Default"** value as well, it means to use the inherent value associated with the type of each property track. The inherent values for each type are described in the tables below.

### Animation mode inherent values

| Property type | value |
|---|---|
| Numeric | Continuous |
| Color | Continuous |
| Text | Keyframe |
| Boolean | Keyframe |

### Result mode inherent values

| Property type | Value |
|---|---|
| Numeric | Relative |
| Color | Absolute |
| Text | Absolute |
| Boolean | Absolute |

## Common Timeline Element Properties

- **Animation mode:** used by property tracks and refers to the method used to interpolate between property keyframes. It can have the following values.
  - **Default:** use the value defined by the element immediately above it in the hierarchy.
  - **Continuous:** a smooth transition between values using an easing function. Only numeric and color properties can use this mode.
  - **Keyframe:** This mode will not do a smooth transition - instead it will change the properties of the instances as the play head of the timeline reaches each property keyframe.
  - **Step:** a smooth transition, but it only shows values that fall in the step defined by the **Step** property of the timeline.
- **Result mode:** how the values in each property keyframe in a property track are interpreted when playing. Numeric values default to **Relative** mode, while text, boolean and color values default to using **Absolute** mode.

  - **Default:** use the value defined by the element immediately above it in the hierarchy.
  - **Relative:** the timeline assigns values relative to the initial values each instance had, before the timeline started playing.
  - **Absolute:** the timeline assigns absolute values, and will not be affected by the initial state before the timeline started playing. In this mode the timeline overrides all other behaviors that might be affecting an instance.
- **Ease:** the function used to transition between each pair of property keyframes in a property track. There are several built in functions to choose from. Custom ease curves can also be designed in the Ease editor. The special **"Default"** value uses the value defined by the element immediately above it in the hierarchy.
  - **Ease editor:** A link to open the Ease editor with the ease selected in the **Ease** property. Doing this it is possible to edit not only a custom ease, but also a built in ease.
- **Path mode:** only relevant for the X and Y properties. Sets how to transition between property keyframe pairs.
  - **Default:** use the value defined by the element immediately above it in the hierarchy.
  - **Line:** interpolate between the starting and ending position of each property keyframe pair to form a straight line.
  - **Cubic Bezier:** This mode will enable a few additional controls in the Layout View to allow for transitions following a curved path.
- **Visible:** used by tracks to toggle the visibility of the corresponding instance. This setting only takes effect while **Edit Mode** is turned on. It is only relevant for the editor and will not affect the timeline at runtime.
- **Enabled:** used by property keyframes. A disabled property keyframe is not taken into consideration when playing a timeline.
- **Locked:** A locked timeline element and its children cannot be modified through the Timeline Bar or Properties Bar. It is only relevant for the editor and will not affect the timeline at runtime.
- **Show UI Elements:** changing this property will turn off the UI elements shown in the layout associated with the affected instances. It is only relevant for the editor and will not affect the timeline at runtime.

# TIMELINE

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

A timeline can gradually change values of an instance over time to produce an animation. It allows for the co-ordination of complex transitions of many different instances at the same time.

Although the most basic usage is to change the **X** and **Y** properties of an instance to produce movement, a timeline can be configured to affect almost every property that can be part of an instance. This includes all the common instance properties, instance variables, effect parameters, behavior properties and plugin properties. Not all properties are supported by timelines as some of them don't make sense, such as any property that is only used on start up.

Timelines can modify boolean and text properties, but since these can not be interpolated like numbers or colors to achieve a smooth transition, they just change as the timeline's play head reaches their property keyframes.

## Adding Timelines

With the Timeline Bar open, create a timeline using any of the following methods:

- Right-click the **Timelines** folder in the Project Bar and select *Add timeline*
- Right-click a space in the Layout View and select Timeline▶Add timeline (when the Timeline Bar is open)

## Nesting Timelines

With the Timeline Bar open and more than one timeline added to the project, use one of the following methods:

- Use the Timeline▶Add timelines option from the Timeline Bar **+** split button.
- Right-click any empty space in the Timeline Bar and use the context menu option Timeline▶Add timelines.
- Drag the timeline you want to nest from the Project Bar into the Timeline Bar.

## Timeline Properties

**Name**

The name of the timeline. It is a unique identifier and is used to refer to the timeline from an event sheet.

**Animation mode**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

---

**Time**

The current time of the main time marker. This only affects the editor.

---

**Total time**

The total time for the timeline to be completed.

---

**Raw Step**

The increments the current time marker can take in the editor.
When using the Step animation mode, this value will be used at runtime to produce the correct increments while playing the timeline. When **Steps per second** property is changed, this property is updated automatically.

---

**Steps per second**

This property is connected to the **Raw step** property, it is used to generate the correct Raw step value in a more intuitive way. When Raw step is changed, this property is updated automatically.

---

**Show UI elements**

See the section on common timeline element properties in Timelines. Changing this property will apply the change to every sub-element.

---

**Use Step**

Use this property to avoid using the step value in the editor. If disabled, scrubbing to preview the timeline in the editor will be completely smooth.

---

**Resize mode**

Choose between **Width & Height** and **Scale X & Scale Y**. This tells the editor which pair of property tracks need to be created when creating keyframes after making size changes in the Layout View.

---

**Loop**

When the timeline finishes, continuously repeat it from the start again. When the timeline is nested this value is ignored and instead the value of the top most parent timeline is used.

**Ping pong**

When the timeline finishes, reverse the direction of playback, so it plays alternately forwards and in reverse. When the timeline is nested this value is ignored and instead the value of the top most parent timeline is used.

**Repeat count**

The number of times to repeat the timeline animation when not looping indefinitely. When the timeline is nested this value is ignored and instead the value of the top most parent timeline is used.

**Start on layout**

This is a dropdown that gives the option to choose at the start of which layout the timeline should start playing automatically. This property serves as a shortcut for the simple use case of just starting playback of a timeline at the beginning of a layout, without any further manipulation.

**Transform**

Whether a timeline's position keyframes should be transformed by changes produced by ancestors in it's own hierarchy, or not. Enabled by default.

**Use system timescale**

Whether a timeline will be affected by the system timescale or not. When disabled a timeline will continue producing changes regardless of the current system timescale.

# TIMELINE TRACK

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

In the Timeline Bar, a track is represented as a row with an icon of the corresponding instance.

Tracks can be moved to and from track folders or the root of the timeline by dragging and dropping. A timeline can have many different tracks in its hierarchy, one for each instance added to it.

## Adding Tracks

To add a track to a timeline follow any of these methods:

- Use the **+** button in the Timeline Bar toolbar to bring up a dialog from which to choose instances to add to the timeline.
- Drag & drop instances from the Layout View into the bar.
- Right-click some instances in the Layout View and select Timeline▶Add to timeline.
- Right-click some Timeline Bar empty space and select Track▶Add instances.

## Track Properties

**Name**

The name of the track. This can not be changed and is automatically generated from the object name and the instance UID.

**Animation mode**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

**Visible**

**Enabled**

**Locked**

**Show UI Elements**

See the section on common timeline element properties in Timelines. Changing these at the track level will apply the change to every sub-element.

---

**Track ID**

An optional identifier to be used in tandem with the **Set Instance** action. See the Timeline plugin section of the manual for more details.

# VALUE TRACK

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/track/value-track

A value track is a special kind of track which is not tied to any instance.

Because they are not tied to any instance the value they interpolate doesn't immediately affect anything, instead it needs to be queried at runtime using the Timeline plugin before it can be used.

They are a little bit harder to use, but offer great flexibility in what they can be used for.

Value tracks can only have a single properly track and a timeline can have as many as needed.

## Adding Value Tracks

To add a value track to a timeline follow any of these methods:

- Use the split button of the + button in the Timeline Bar toolbar and choose the option Track▶Add value.
- Right-click some Timeline Bar empty space and select Track▶Add value.

## Value Track Properties

**Name**

Must be unique in the timeline. Can be used to query the value of the track at runtime.

**Animation mode**

**Ease**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

**Enabled**

**Locked**

See the section on common timeline element properties in Timelines. Changing these at the track level will apply the change to every sub-element.

**Track ID**

An optional identifier to query the value of the track at runtime.

# AUDIO TRACK

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/track/audio-track

An audio track is a special kind of track which can be used to trigger audio playback from a timeline.

Audio tracks can have two types of property tracks, the main audio source one and another one to control it's volume. A timeline can have as many as needed.

While the audio playback is connected to the corresponding timeline, it is not exclusively controlled by it. If a value is provided in the **Tag** property of the audio source property track, then the Audio plugin can be used to further control playback.

## Adding Audio Tracks

To add an audio track to a timeline follow any of these methods:

- Use the split button of the **+** button in the Timeline Bar toolbar and choose the option Track▶Audio audio.
- Right-click some Timeline Bar empty space and select Track▶Add audio.
- Drag & Drop from either from the **Sound** or **Music** folders in the Project bar into the Timeline bar.

The audio track itself doesn't have much functionality in on itself, instead it is the audio source property track, which holds the more specific properties. Look into that for more details.

## Audio Track Properties

**Name**

Must be unique in the timeline.

**Animation mode**

**Result mode**

**Ease**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

*These properties only affect the corresponding volume property track of an audio track.*

**Enabled**

**Locked**

See the section on common timeline element properties in Timelines. Changing these at the track level will apply the change to every sub-element.

**Track ID**

Audio tracks don't use this property.

# TIMELINE PROPERTY TRACK

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/property-track

In the Timeline Bar, property tracks are represented as a row with an icon of the corresponding property being affected by the track they belong to.

A property track represents the property of an instance that can be interpolated and is nested inside a parent track. Property tracks can refer to common instance properties, instance variables, effect parameters, behavior properties and plugin properties.

A track can have many property tracks in its hierarchy, one for each property of the instance the timeline will be affecting. A property track can exist at the root of a track or nested inside property track folders. They can be moved to and from property track folders or the root of the parent track by dragging and dropping. They can not be moved outside of their parent track.

## Adding Property Tracks

Property tracks are added automatically when:

- A new track is added to a timeline.
- A keyframe for a new property is added.

To add an empty property track, right-click a track or property track folder and select **Add properties**.

## Property Track Properties

**Name**

The name of the property track. This can not be changed it takes the same name as the property being modified.

**Animation mode**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

**Visible**

**Enabled**

**Enabled**

**Locked**

See the section on common timeline element properties in Timelines. Changing these at the property track level will apply the change to every sub-element.

# AUDIO SOURCE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/property-track/audio-source

The audio source property track is unique to the audio track and has a few unique properties to it that are not present in any other type of property track.

## Audio Source Property Track Properties

**Name**

The name of the property track. This can not be changed it takes the same name as the property being modified.

**Start offset**

The starting time in the timeline

**Audio duration**

The total duration of the audio source, this property can not be modified

**Tag**

An optional tag to be able to control the corresponding audio object by using the Audio plugin

**Enabled**

**Locked**

See the section on common timeline element properties in Timelines. Changing these at the property track level will apply the change to every sub-element.

# TIMELINE MASTER KEYFRAME

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/master-keyframe

In the Timeline Bar, a master keyframe is represented by marks in the same row as the track they belong to.

A track can have many different master keyframes, one for each position in the timeline with values that must be reached when animating. By themselves master keyframes only mark a time - the values used for interpolation are kept by the property keyframes.

The main role of master keyframes is to serve as a control to modify all related property keyframes at the same time. Any changes made to a master keyframe will be applied to all corresponding property keyframes, including **Enabled**, **Time**, **Ease** and **Path mode** changes as well as deleting.

## Adding Master Keyframes

To add master keyframes, follow these steps:

- Turn on **Edit mode** by pressing the **Edit** button in the Timeline Bar toolbar.
- Move the current time marker to the position in the timeline where you want to create keyframes. This can be done by either clicking on the time ruler or by dragging the red line.
- Make changes to the instances you want to animate.
- Use the S keyboard shortcut or right-click in the Layout View and select Timeline▶Set keyframes.

### Master Keyframe Properties

**Index**

The index of the master keyframe in its track. It can not be changed. This value is updated if the keyframe's position in the timeline changes.

**Time**

The position of the master keyframe in the timeline. This can be updated from the Properties Bar or by dragging the keyframe in the Timeline Bar. Either change will update all property keyframes under the master.

**Ease**

**Path mode**

### Enabled

See the section on common timeline element properties in Timelines. **Ease** and **Path mode** follow an inheritance pattern while changing **Enabled** at the master keyframe level will apply the change to every associated property keyframe.

> *The **Ease** and **Path mode** values of master keyframes take precedence over the corresponding property track. This is an exception on the inheritance pattern these properties follow.*

---

### Tags

A space separated list of identifiers that can be used with a set of Timeline plugin conditions, to identify when a master keyframe has been reached while a timeline is playing.

# TIMELINE PROPERTY KEYFRAME

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/property-keyframe

## Property Keyframe

In the Timeline Bar, property keyframes are represented by a mark in the same row as the property track they belong to.

Property keyframes hold the values the timeline uses when it is playing. They are at the end of the timeline hierarchy so any changes made at this level will only affect the property keyframe itself.

A property track can have many different property keyframes, one for each position in the timeline with values that must be reached when animating. Unlike master keyframes, property keyframes cannot be moved and must have a parent master keyframe. If a property keyframe's time is changed by either dragging it or changing the value in the Properties Bar, a new master keyframe will be created at the new position along with new property keyframes to go with it.

## Adding Property Keyframes

Property keyframes can be created following the same method to create master keyframes, since property keyframes will be created for each property track at a given position in the timeline.

To create property keyframes for specific property tracks, follow these steps:

- Turn on **Edit mode** by pressing the **Edit** button in the Timeline Bar toolbar.
- Move the current time marker to the position in the timeline where you want to create property keyframes. This can be done by either clicking on the time ruler or by dragging the red line.
- Make changes to the instances you want to animate.
- Right-click on the property track you wish to add a property keyframe to and select **Set keyframes**.

If multiple property tracks are selected when using the **Set keyframes** option, property keyframes will be created for all of them.

Either method will always create a master keyframe along with the property keyframes.

### Property Keyframe Properties

**Index**

The index of the property keyframe in its property track. It can not be changed. This value is updated if the keyframe's position in the timeline changes.

---

**Name**

The name of the property track this property keyframe belongs to. This cannot be changed since it takes the same name as the property being modified.

---

**Value**

The value the property keyframe holds. This can be either numeric, text, boolean or color. Depending on the result mode in use this will be Relative or Absolute.

---

**Time**

The position of the property keyframe in the timeline. Since property keyframes cannot really be moved, changing this value will create a new master keyframe at the new position with new property keyframes to go with it. This can be updated from the Properties Bar or by dragging the keyframe in the Timeline Bar.

---

**Enabled**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines.

## Unique properties to angle values

---

**Direction**

Allows you to choose which direction the angle will take to arrive at it's final value. By default Construct will attempt to take the shortest path but it can be forced to rotate clockwise or anti-clockwise by changing this property.

---

**Revolutions**

The amount of additional 360 degree turns the angle should take before arriving at it's final value.

# TIMELINE TRACK FOLDER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/track-folder

In the Timeline Bar, track folders are represented as a row with an icon of a folder.

These are used to organize elements in a timeline, and don't have any impact in the playback. A track folder can have nested tracks as well as other track folders. They can be moved to and from track folders or the root of the timeline by dragging and dropping.

It is worth noting that the only specific property of a track folder is the **Name**. Every other property in it doesn't have a direct impact on the folder itself and exists only as a convenience to make modifications in all of the items within it.

## Adding Track Folders

Track folders can be added to the root of a timeline by doing any of the following:

- Right-clicking any section of the bar which doesn't correspond to any other element of a timeline and selecting Track▶Add subfolder.
- Using the split menu of the Timeline Bar **+** toolbar button, and selecting Track▶Add subfolder.

A track folder can be created directly as a subfolder by right-clicking another track folder.

## Track Folder Properties

**Name**

The name of the track folder.

**Animation mode**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

**Visible**

**Enabled**

**Locked**

### Show UI Elements

See the section on common timeline element properties in Timelines. Changing these at the track folder level will apply the change to every sub-element.

# TIMELINE PROPERTY TRACK FOLDER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/property-track-folder

In the Timeline Bar, property track folders are represented as a row with an icon of a folder.

These are used to organize elements in a track, and don't have any impact in the playback of the timeline. A property track folder can have nested property tracks as well as other property track folders. They can be moved to and from property track folders or the root of the corresponding track by dragging and dropping. Property track folders can not be moved outside their corresponding track.

When adding effect parameters or behavior properties to a timeline, the created property tracks will be placed inside a special property track folder which is automatically given the name of the effect or behavior. The folder can not be renamed and only accepts property tracks which refer to the same effect or behavior.

It is worth noting that the only specific property of a property track folder is the **Name**. Every other property in it doesn't have a direct impact on the folder itself and exists only as a convenience to make modifications in all of the items within it.

## Adding Property Track Folders

Property track folders can be added to the root of a track by right-clicking a track and selecting **Add subfolder**. They can also be created as sub folders by right-clicking another property track folder.

## Property Track Folder Properties

**Name**

The name of the property track folder.

**Animation mode**

**Result mode**

**Ease**

**Path mode**

See the section on common timeline element properties in Timelines. These properties follow an inheritance pattern.

**Enabled**

**Locked**

See the section on common timeline element properties in Timelines. Changing these at the property track folder level will apply the change to every sub-element.

# TIMELINE FOLDER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/timelines/timeline-folder

In the Timeline Bar, timeline folders are represented as a row with an icon of a folder.

These are used to organise nested timelines in a parent timeline and don't have any impact in the playback. A timeline folder can have nested timelines as well as other timeline folders. They can be moved to and from timeline folders or the root of the timeline by dragging and dropping.

It is worth noting that the only specific property of a timeline folder is the **Name**. Every other property in it doesn't have a direct impact on the folder itself and exists only as a convenience to make modifications in all of the items within it.

## Adding Timeline Folders

Timeline folders can be added to the root of a timeline by doing any of the following:

- Right-clicking any section of the bar which doesn't correspond to any other element of a timeline and selecting Timeline▶Add subfolder.
- Using the split menu of the Timeline Bar **+** toolbar button, and selecting Timeline▶Add subfolder.

A timeline folder can be created as a sub folder by right-clicking another timeline folder.

## Timeline Folder Properties

**Name**

The name of the timeline folder.

**Visible**

**Enabled**

**Locked**

**Show UI Elements**

See the section on common timeline element properties in Timelines. Changing these at the timeline folder level will apply the change to every sub-element.

# FLOWCHARTS

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

Flowcharts allow you to arrange information in a tree like structure and establish logical connections between each node in the tree. Additionally each node can hold an arbitrary amount of information in the form of outputs.

In the editor flowcharts are edited using the Flowchart View.

At runtime the Flowchart Controller plugin is used to query the current node of a flowchart for information, to navigate to different nodes and to react to changes in the state of the flowchart and have the application respond accordingly (e.g. moving from one node to the next one).

*Flowcharts themselves don't have any means to perform logic. Since event sheets are already very good at managing logic, flowcharts are primarily meant as a tool to represent a tree structure in a way which makes it easier to understand the relationship between the nodes.*

# FLOWCHART

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

The flowchart itself does not have a lot of information - in fact it only has a name which is used at runtime by the Flowchart Controller to create instances of flowcharts defined in the editor. Usually it is the nodes inside a flowchart that are of the main interest.

## Adding flowcharts

Right-click the flowcharts folder in the Project Bar and select the **Add flowchart** option.

To edit a flowchart, double click on it in the Project Bar or right-click on it and select the **Open** option from the context menu. Doing any of those things will bring up the Flowchart View.

## Flowchart properties

**Name**

The name of the flowchart. Used at runtime to create an instance of a flowchart that the Flowchart Controller can manipulate.

# NODE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/flowcharts/flowchart-node

The node is the main component of a flowchart. It represents a state the flowchart can get into and can hold arbitrary data in the form of outputs, similar to having a small amount of Dictionary data in each node.

The information a node holds can be queried at runtime using the Flowchart Controller plugin.

## Creating nodes

To create a node in a flowchart, open a Flowchart View for it from the Project Bar and then right-click in any empty space of the Flowchart View and select the **Add** option from the context menu. Doing that will create a node at the position of the pointer.

For more information on further editing of nodes, see the Flowchart View section of the manual.

## Node properties

**Tags**

A string used to identify a node at runtime. Multiple different tags can be assigned by separating them with a space. The Flowchart Controller plugin has several actions, conditions and expressions that can be used with one or multiple tags to target specific nodes.

**Start node**

A boolean property specifying the first node a flowchart will be in once it is instantiated at runtime.

**Enable**

A boolean property indicating if the node is enabled or not. A disabled node is ignored at runtime and the flowchart state will not go past it. A disabled node can be bypassed at runtime if it has a default output set.

**Caption**

An editor-only property. This is an optional name that can be given to a node to help distinguish it from others.

**Color**

An editor-only property. This is an optional color that will be applied to the border of the node.

---

**Preset**

An editor-only property. Indicates if the node is a preset or not. Preset nodes can be used as a base to create new nodes.

---

**Outputs**

A list of the outputs each node has.

---

**Parent Index**

In the case a node is connected to multiple parent nodes, this property appears and displays an index that can be given to each parent. Used by some actions and expressions in the case a node has more than one parent.

# OUTPUT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/flowcharts/flowchart-node-output

Nodes can have an arbitrary amount of outputs, which have a name, an associated value, and can be linked to the input of another node.

## Editing outputs

For a more complete overview on the editing tools available, look at the Flowchart View section of the manual.

## Output Properties

**Index**

The index of an output is a read-only property and can be used by the Flowchart Controller to query information about an output or to move to the next connected node.

**Name**

A string of a name for the output. This can be used by the Flowchart Controller plugin to get the associated value or to move the state of the flowchart to the next connected node.

**Value**

A string which can hold an arbitrary value with no specific purpose, to be accessed via event sheets.

**Enable**

A boolean property indicating if the output is enabled or not. A disabled output is ignored at runtime, it can not be followed and it's information can not be queried. It is not considered when using the for loop condition to iterate the outputs of a node.

**Default**

A boolean property indicating if the output is the default one or not. Default outputs can be targeted by the Flowchart Controller ACEs. If a node doesn't have any explicit default output, the corresponding ACEs don't do anything. This setting can also be used together with disabled nodes to bypass them at runtime. When the runtime encounters a disabled node with a default output it will attempt to follow all bypasses until it can land in an enabled node.

**Color**

An editor only property to help better identify links between nodes when there are many of them.

**Link mode**

An editor only property to choose how to draw the link between nodes. Can either be a straight line or use path finding to avoid overlapping nodes. Using path finding can be useful when a link ends up going over other nodes.

# REFERENCE NODE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/flowcharts/reference-node

The reference node is a secondary type of node that can be added to a flowchart.

Unlike regular nodes, which represent a state and can have information, these are placeholders for other flowcharts. When the flow reaches one, a new instance of the corresponding flowchart is created and control is transferred to it.

This type of nodes are meant to help splitting up flowcharts in case they become too large or if it's necessary for unique portions of a larger flowchart to be available individually.

## Creating reference nodes

To create a reference node in a flowchart, open a Flowchart View for it from the Project Bar and then right-click in any empty space of the Flowchart View and select the **Add reference node** option from the context menu. Doing that will create a node at the position of the pointer.

For more information on further editing of nodes, see the Flowchart View section of the manual.

## Node properties

**Tags**

A string used to identify a node at runtime. Multiple different tags can be assigned by separating them with a space. The Flowchart Controller plugin has several actions, conditions and expressions that can be used with one or multiple tags to target specific nodes.

**Enable**

A boolean property indicating if the reference node is enabled or not. A disabled node is ignored at runtime and the flowchart state will not go past it.

**Reference flowchart**

The flowchart that should be created upon reaching this node.

**Reference start node**

The tags of the node that should visited in the referenced flowchart. The first node found that matches all passed tags will be used as the starting position of the flowchart. If left empty the default start node set in the editor is used.

**Runtime tag**

The tag given to the referenced flowchart at runtime to be able to identify it. This is analogous to the tag a flowchart can be given when it is instantiated through the Flowchart Controller's **Start flowchart** action. If left empty a unique tag is assigned.

**Caption**

An editor-only property. This is an optional name that can be given to a reference node to help distinguish it from others.

**Color**

An editor-only property. This is an optional color that will be applied to the border of the node.

**Parent Index**

In the case a node is connected to multiple parent nodes, this property appears and displays an index that can be given to each parent. Used by some actions and expressions in the case a node has more than one parent.

# COMMENT NODE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/flowcharts/comment-node

The comment node is a secondary type of node that can be added to a flowchart.

Unlike regular nodes, which represent a state and can have information, these are meant to be used exclusively as notes to describe what is going on in a certain part of a flowchart and have no impact at runtime.

## Creating comment nodes

To create a comment node in a flowchart, open a Flowchart View for it from the Project Bar and then right-click in any empty space of the Flowchart View and select the **Add comment node** option from the context menu. Doing that will create a node at the position of the pointer.

For more information on further editing of nodes, see the Flowchart View section of the manual.

## Main comment

The main body of the node is a text box that can be clicked on to write in it.

> *Comment nodes can overlap with other nodes and are always shown below nodes which have a logical purpose in the flowchart.*

## Node properties

**Caption**

An editor-only property. This is an optional name that can be given to a reference node to help distinguish it from others.

**Color**

An editor-only property. This is an optional color that will be applied to the border of the node.

**Font**

An editor-only property. The font to use for the main comment.

**Size**

An editor-only property. The size of the font used in the main comment.

---

**Bold**

An editor-only property. Formats the main comment text to be **bold**.

---

**Italic**

An editor-only property. Formats the main comment text to be *italic*.

---

**Font color**

An editor-only property. The color of the main comment text.

# SOUNDS AND MUSIC

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/sounds-and-music

Sounds and Music in Construct are audio files. Any audio files the project uses will be exported with the project. To ensure your audio can be played across a wide range of platforms, Construct will encode them to **WebM Opus** when importing, if possible. It is recommended to import **16-bit PCM WAV** files to ensure they can be encoded without any unnecessary quality loss.

Imported audio files can be previewed by **double-clicking** them in the Project Bar. This will open an audio player that you can use to listen to the audio file.

To import and play back audio in a project, follow these steps:

1. Import sounds by right-clicking the **Sounds** or **Music** folders in the Project Bar and selecting **Import sounds** or **Import music**.
2. Import audio via the **Import Audio dialog**.
3. Add the Audio object to the project and add events to play back audio.

For more information on audio support in Construct, see the Import Audio dialog.

Note that when publishing to the web, you must ensure your server has the correct MIME types set up to ensure audio files can load correctly. If the server sends the wrong MIME types, audio may not play in some browsers.

# PROJECT FILES

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

Any external files can also be imported to your project via the Project Bar. This is useful for including any other resources your project might need, such as videos, additional images, JSON/XML/CSV/text data, HTML, CSS, documents, or other general files you might need in your project. Often project files are requested in events with the **AJAX** object, allowing data files to be read by the game.

Project files (excluding sounds and music) can be categorised in to folders for **Videos**, **Fonts**, **Icons** (see Icons & splash) and **Files** (for anything that does not belong to one of the other folders). Files can be imported by right-clicking one of the folders in the Project Bar and selecting the **Import** option. Note importing files copies them to the project.

## Previewing files

Several kinds of files can be previewed simply by **double-clicking** their item in the Project Bar:

- Like sounds and music, previewing a video file will play it in a video player.
- Fonts preview in a dialog showing some sample text using the font.
- SVG images are previewed in a resizable dialog that displays the image.
- Other images than SVG can be opened in the Animations Editor for viewing and editing. This is particularly useful with icon files.
- Array, dictionary and text-based files can be viewed and edited in the Array editor, Dictionary editor and Text editor respectively Paid plans only.

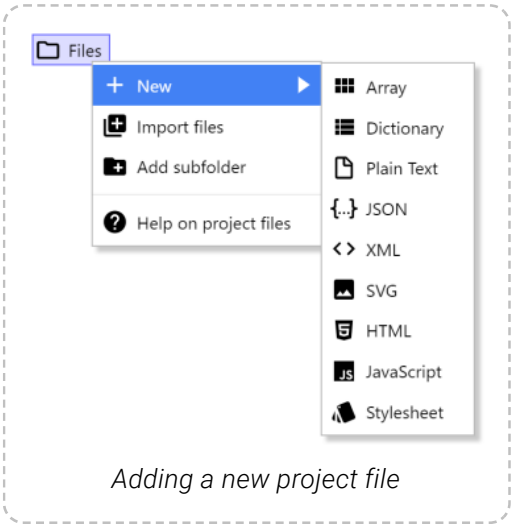
Other text-based files, like HTML and CSS files, can also be edited with a built-in text editor by double-clicking them Paid plans only.

## Using Web Fonts

You can use any web font (.woff or .woff2) files imported to the Fonts folder in Text objects. For more information see the section on *Using web fonts* in the Text object.

## Adding new files

Paid plans only When right-clicking the general-purpose **Files** folder, there is also a **New** submenu which provides a range of common kinds of file that you might want to add to the project.




*Adding a new project file*

Adding an **Array** adds a JSON file in a format that can be loaded by the Array object, and opens the Array editor. Adding a **Dictionary** adds a JSON file that can be loaded by the Dictionary object, and opens the Dictionary editor. Each allows the data in the files to be edited in a visual editor. The other files open a Text editor for editing the contents of the file. Once added, **double-click** the file in the Project Bar to re-open its editor.

## Adding HTML files

Paid plans only HTML files can be added to the project, and edited with a built-in text editor. This is useful for writing long HTML content to be displayed in the HTML Element object. The content can then be fetched using the AJAX object, and then set as the content of the HTML Element.

HTML project files can also have a *Purpose* set in the Properties Bar. This allows using them to insert content in to the exported HTML file. If the file is used this way, it will no longer be exported as an additional file. The possible purposes are:

- **(none)**: the HTML file will not be automatically used anywhere. It will be exported as an additional file.
- **End of <head>**: the contents of the HTML file will be inserted just before `</head>` in the exported HTML.
- **Start of <body>**: the contents of the HTML file will be inserted just after `<body>` in the exported HTML.
- **End of <body>**: the contents of the HTML file will be inserted just before `</body>` in the exported HTML.

## Adding stylesheets

Paid plans only Stylesheets, using the CSS file extension (short for *Cascading Style Sheet*), can be added to the project and edited with a built-in text editor. This is particularly useful for customising the appearance of the HTML Element object.

Custom CSS files are also useful for customising the appearance of other HTML-based objects like Button and Text Input. These objects have *ID* and *Class* properties which can be used to

identify them so custom styles can be applied from stylesheets.

> *Construct sets a CSS variable named* `--construct-scale` *on the root* `html` *element of the document, with a number representing the canvas scale as a multiplier. You can use this to scale CSS properties to match the displayed canvas size. For example* `height: calc(var(--construct-scale) * 2em);` *sets a height size of 2em at 100% scale, but also adjusts the height to follow Construct's fullscreen scaling.*

CSS files have a *Purpose* property in the Properties Bar. The default is *Stylesheet* which means Construct will use it as an additional stylesheet for your project. The purpose can also be set to *(none)* in which case its styles will not automatically be applied, and the stylesheet will merely be exported as an additional file on export. This may be useful if the stylesheet is needed for other purposes, such as for the content of an iframe.

## MIME types

When publishing to the web, you must ensure the server hosting the project has the correct MIME types set up for all the kinds of project file you use. Otherwise some project files may fail to load, or could be loaded with corrupted data.

# COMMON CONVENTIONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/common-conventions

For consistency, Construct usually uses some common conventions in how values are used. These are described below.

## Common units

In Construct, sometimes you need to enter values such as angles, speeds or sizes. For consistency these always use the same units in Construct, except where noted by descriptions or tips shown in the editor. The common units are described below.

- **Positions** are in pixels. The origin (0,0) is at the top-left of the layout, and the Y axis increments downwards (as is often the case with game engines).
- **Sizes** are in pixels
- **Angles** are in degrees. 0 degrees faces **right** and increments **clockwise**.
- **Times** are in seconds
- **Speeds** are in pixels per second
- **Accelerations** are in pixels per second per second

## Zero-based indexing

To be consistent with programming languages, all features of Construct using a number of an item in a list (indices) start from 0 instead of 1. This might seem odd at first and take some getting used to, but in many cases it is actually much more convenient than 1-based indexing.

Traditionally lists are numbered **1, 2, 3...** but in Construct (and all other programming languages) they are numbered **0, 1, 2...**.

## Ranges

Sometimes the documentation will refer to ranges of valid values. These are in square brackets for an inclusive range, such as [0, 1] meaning any value between 0 and 1, including both 0 and 1. For example valid values in this range are 0, 0.5, and 1. A round bracket indicates a non-inclusive boundary of the range, such as [0, 1) meaning any value between 0 and 1, including 0 but not including 1. For example valid values in this range are 0, 0.5 and 0.999, but not 1.

# BEST PRACTICES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/best-practices

## Back up regularly!

Neither the hardware nor software in your computer is perfect. Computers fail and software can crash. Back up your projects to protect yourself from losing work. It is essential to also maintain off-site backups. If all your backups are in the same computer or saved to disks all in the same building, catastrophic events like fire, flood, theft, software/hardware failure, or cyber attacks/related malware like ransomware, can all cause you to lose all your work and backups together.

Cloud Save is a good way to save your work where it is safe in case of disaster. However it is wise to keep secondary backups anyway, in case you lose access to your account, or the service has an outage or even shuts down. Keeping additional backups is particularly important when saving to local files or folders on your system. Construct can help you do this by automatically making backups. See the *Save & backup* section of the Settings dialog. Check *Periodically back up active project*, and choose the location and backup interval. For example you could set up an automatic save to the same location as the project every 10 minutes, or select a local backup folder (where supported by the browser) to save backups to.

> *Saving an extra copy to a local backup folder isn't a safe backup when using local file/folder saves: it's just another copy on the same system, and any disaster that renders the system inaccessible will cause you to lose both your work and its backups. It's best to use this option if you're already using Cloud Save (so there's a secondary copy on your system), or if the local folder is automatically copied to the cloud (such as a local Dropbox folder which will automatically upload all files in the folder to your cloud account).*

The advice to back up regularly is not specific to Construct. It is vital to adopt this practice for any work on a computer which is important to you. Do not wait until you've lost work before starting to do this. People lose work regularly from having poor backup practices. Don't be one of them!

## Test on multiple platforms, browsers and devices

It is essential to test your project works as intended across a range of different systems. While Construct exports are based on the HTML5 standard which in theory is implemented the same on all platforms, in practice there are variations between browsers and devices (e.g. in performance, features, text rendering, etc). You should install a range of browsers on every device you have available and test with them all to ensure your project will work well for everyone. Remote Preview Paid plans only can help with this, especially since you can get anyone in the world to help test with their devices. You may also need to make test exports to

check how your project works as published, since app containers like Cordova (for Android and iOS) can have differences too.

## Support touchscreen devices

Many users browse the web with touchscreen devices. Where applicable, you should design your project to also support touch input. Often you can simply use the Touch plugin instead of the Mouse plugin.

## Avoid superseded features

Some features exist mainly for backwards compatibility reasons and should be avoided in new projects in favor of newer features that essentially replace them. See Superseded features for a list of such features to avoid.

## Recommended file formats

You may wish to prepare artwork and audio in other software before importing to Construct. These are the formats we recommend.

### Images

Use **32-bit PNG** (Portable Network Graphics) while preparing images. Be sure to select 32-bit if you are given a choice; the 8-bit or lower versions may degrade quality. 32-bit PNGs are lossless and fully support alpha-channel transparency. Note some images such as Microsoft Paint do not support PNG transparency. Use may need to use a different editor instead, such as Paint.NET on Windows.

You can choose different export formats like JPEG or WebP inside Construct to reduce the size of your finished project. However when importing you should still stick to 32-bit PNGs if possible, and leave Construct to recompress them when exporting. Construct does a lot of optimisation on export for you. It is unlikely that any third party tools or services will be able to beat Construct's existing lossless optimisations, unless they degrade the image quality. Remember there is no point optimising images before importing them to Construct since it stores them in projects as 32-bit PNGs with default compression settings; they are only optimised on export.

### Audio

Use **16-bit PCM WAV** while preparing audio. These are typically .wav files, but note that not all .wav files are 16-bit PCM. Importing a 16-bit PCM .wav file to Construct will automatically encode it to WebM Opus. PCM WAV files are lossless, ensuring there is no quality degradation while you prepare your audio files. Allowing Construct to perform the encoding ensures the encoding is only done once (so there is no unnecessary degradation), and that the correct format is used for support across a wide range of platforms.

### Fonts

Use fonts in **WOFF** (Web Open Font Format) format wherever possible. Fonts in other formats such as TTF may or may not work depending on specific details of browser support and the features used in the font file. WOFF was designed specifically for web browsers and so is the best supported format for web-based technology such as Construct.

## Security

**Never enter usernames or passwords in to events.** These will be visible in plain text in exported Javascript, and malicious users will very quickly be able to take control of the account. If you need to connect to something like a database, write a server-side script that talks to the database, then connect to the URL of the server.

## Performance

Many people worry about performance but just end up wasting their time. There is a specific way that you ought to approach performance questions and issues - see the guide on Performance Tips for more about that.

## Memory use

Sometimes people accidentally create projects that waste large amounts of memory due to not understanding the way things like imags are handled in memory. See the guide on Memory usage for more about that.

## Download size

It's a good idea to at least check your project's download size, and try to reduce the size of any particularly large files. This can be important for platforms that impose a file size limit, or for people who have slow connections. See the guide on Download size for more about that.

# PERSISTING BROWSER STORAGE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/persisting-browser-storage

Construct asks the browser to store various kinds of information to ensure it persists across sessions. This includes:

- Your account login
- The list of recent projects
- The user interface layout
- Your chosen settings
- Installed third-party addons
- Saving Construct itself for use offline
- Saved large downloads so they don't have to be downloaded again every session
- Many more details

If your browser refuses to save this data to storage, or regularly clears it, then you may experience problems like your account login being forgotten, your recent projects being cleared, the user interface and your chosen settings resetting, installed addons becoming uninstalled, and so on.

To fix this, you usually need to adjust your browser settings, as it is the browser that is causing your data to be lost rather than Construct. This guide covers a range of possible causes and solutions for this kind of problem.

## Privacy, tracking or cookie settings

Some websites abuse storage for things like unauthorized tracking of users. Also, many countries around the world have increasingly strict regulation of data management. Therefore most modern browsers have comprehensive settings to control what websites are allowed to do to preserve privacy and mitigate unwanted tracking. However some settings could prevent Construct from being able to store data persistently.

Check your browser settings to see if you've changed something that is blocking Construct from storing data. These settings could relate to privacy, anti-tracking, storage, or "cookies" (a general term for any data stored by websites on your device). For example if you have a setting that clears all cookies every 7 days, then it's possible everything Construct has remembered will be reset every 7 days.

## Browser extensions

Some browser extensions provide additional privacy, tracking or data control. Similar to browser settings, be sure to check any browser extension settings relating to privacy, anti-tracking, cookies or storage. For example if you have a browser extension that only allows specific websites to permanently store data, then you would need to add Construct to the list to allow it to persist its storage. Alternatively you could uninstall the browser extension to see if that solves the problem.

## Manually clearing browser data

It's worth noting that manually clearing browser data may also clear Construct's saved data. As above, choosing to clear "cookies" or "site data" may cause everything Construct has saved to be deleted. You may need to carefully choose what categories of data to clear, or just avoid manually clearing browser data unless you're willing to also reset Construct.

## Private browsing modes

Most modern browsers provide a special private browsing mode for privacy reasons. These modes by design do not remember anything after you close the private browsing window. Therefore if you use a private browsing window, Construct will always act as if it is the first time you are running it, with no login, default settings, and so on.

## Request persistent storage

Something that may help Construct remember data permanently is to request permission for persistent storage. In Construct choose Menu▶About to open the About dialog, and find the *Local storage* section. This shows how much storage Construct is using and the available storage quota, as reported by the browser. Where supported by the browser, there may also be a *Request persistent storage* link, with text reporting the current persistence state (e.g. "Browser may not persist storage"). Click the *Request persistent storage* link to ask the browser to persist Construct's storage. You may see a permission prompt, so be sure to approve that. If it works, the status text should change to read "Browser will persist storage".

Note that this still does not provide a guarantee that data will be held permanently. It is still possible that things like manually clearing data, browser extensions, or running out of disk space, may cause the data to be lost anyway. However this generally improves the chance that Construct's data will be able to persist long term.

## System storage running out

If your system is running out of disk space, sometimes browsers respond by starting to erase website data. This can cause Construct's data to be lost as well.

As described above, requesting persistent storage may help avoid this - the browser should start by erasing other website's data first. However if the system runs out of storage space repeatedly, it may end up deleting Construct's saved data anyway.

Construct's Storage Cleanup dialog can help free up space used by Construct without having to clear everything like your settings and login. Sometimes things like saved exported projects, downloaded example projects, and some export options, can end up using a lot of data. So if you are running low on disk space, try using that dialog to reduce the storage space Construct is using. You can also try general-purpose methods to save on disk space, such as using any system storage cleanup tool, uninstalling unused apps, deleting unnecessary files, and so on.

## Try another browser

If all else fails, you can try using a different browser. In general most modern browsers with their default settings should be able to persistently remember data that Construct saves.

# PERFORMANCE TIPS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/performance-tips

This section covers some general performance tips. Many people unnecessarily waste time when thinking about performance. This guide is aimed at a practical approach so you only investigate performance where it matters. There's also some general advice on common problems and pitfalls to avoid.

## Modern computers are fast

Many people underestimate just how powerful modern computers are. Modern hardware, operating systems, browsers, and Construct itself, are all exceptionally fast and well-optimized. Many projects will have excellent performance at the end even if you never put any effort in to improving performance. That's a good thing - it means you can spend your precious time making your project better rather than trying to figure out how to improve performance. Sometimes people try to improve performance before there's even a problem, which usually means you are just wasting your time. That includes questions like "which is faster, A or B?" - usually the answer is "it doesn't matter at all and you're wasting your time".

## Regularly test on target devices

If you are previewing your project on a high-end gaming PC, it is likely to be able to handle high-intensity content that would slow to a crawl on weaker devices. That might be OK for a PC game intended for other gamers, but be too slow for content designed for typical consumer devices. For this reason it is still important to regularly test your project on the kinds of devices you expect a typical player will use, to make sure you can identify any problems before publishing. The sooner you can identify a performance problem, the more likely you are to know what change caused it. The worst case scenario is to finish an entire project without testing, then find it's too slow, in which case you may have no idea why (although measurements can help, as described later). As noted previously you may find everything is running smoothly all the way through development, which is a great outcome, but it's best to be sure!

In the past, mobile devices used to have significantly worse performance to desktops and would often need special consideration for performance. However many modern high-end mobile devices are now about as powerful as mid-range laptops, and are likely to handle most kinds of Construct content fine as well. If you want to target particularly low-power budget devices, you may need to take special care though. Either way, as noted, have a target device to hand and regularly test on it so you can be sure!

## Measure, measure, measure

There is only one way to get a good answer about a performance question: **measure it.** If you don't measure it, or you guess, or you ask on the forum (so someone else guesses), you may get

an inaccurate answer, and you may end up wasting your time trying to optimise something that makes no difference at all to performance. You must use measurements and be scientific about proving what makes a difference.

The ultimate performance measurement is the **frames per second** (FPS). If the FPS rate is good, there is no need to optimise anything! Don't waste your time. If it's not good enough and your project is running slowly, the key measurements to look at are the CPU and GPU utilisation. You can find all these measurements in Construct's debugger Paid plans only.

## CPU vs. GPU

There are two major tasks to running a Construct project: running the logic, including all your event sheets, JavaScript code, and behaviors - done by the CPU (Central Processing Unit) - and drawing the graphics, including visual effects, done by the GPU (Graphics Processing Unit). Usually these are two different pieces of hardware (or different components even if on the same chip).

If the CPU measurement is very high, you probably need to optimise the project logic. If the GPU measurement is very high, you probably need to optimise rendering. So these measurements will guide your overall approach.

In both cases, once again, the key is to **make measurements**. Be scientific: try making a change, and see how it affects the key performance measurements. If it doesn't make a difference, the change doesn't matter to performance. If you make the right change, you will see a measurable difference. This is also how you answer "which is faster, A or B?" type questions: try both and measure them. (Often you will find no measurable difference!)

The following sections provide some tips about how to track down the cause of a meaningful performance problem that you've identified, and some of the common culprits.

## CPU performance

The CPU is generally responsible for everything except drawing graphics. Event sheets, JavaScript code, plugins, behaviors, and everything else apart from graphics tend to happen on the CPU.

The best place to start looking to identify a CPU performance problem is the CPU profiler Paid plans only in Construct's debugger. This can break down CPU time spent in different areas, including individual event groups in event sheets. If you see an item measuring a particularly high usage, that is generally the item to think about optimising. For example you may see a single event group using a substantial percentage of CPU time by itself. If you look at that event group you may then find it does something like repeat an event thousands of times; adjusting that to do less work could then solve the problem.

Other times it may be less clear, such as time spent processing behaviors. Here are some general tips about the types of things that can cause high CPU usage. As before the way to check for these is to rely on measurements: try removing them, or significantly reducing their usage, and see what difference it makes to the measurements.

- **Physics behavior**: physics simulations are highly CPU intensive. Using a relatively small number of objects with the Physics behavior should be manageable. However if you use thousands of objects with the Physics behavior it could cause a considerable slowdown.
- **Creating too many objects**: while modern computers and software are very fast, everything has a cost, even if small. Creating thousands of objects can significantly increase the CPU usage, depending on what kind of logic your event sheets, code, and behaviors run.
- **Using too many particles**: while a single particle is even cheaper than an object, it is still not free. Like creating too many objects, having too many particles could also cause high CPU usage.
- **Running too many events:** if you have a very large project with thousands of events, those events will all need to be run, which may end up with a significant processing overhead. Usually most of the events don't need to be checked all the time. You can significantly reduce the number of events that need to be run by organising them in to event groups, and disabling the groups that are not currently needed. (Events in disabled groups are skipped entirely.) Sub-events can perform a similar role, as sub-events are only checked if the parent event is true.
- **Using too many loops:** using too many loops like *For*, *For Each* and *Repeat* can cause the project to slow down if used intensively. Nested loops are especially likely to cause this, as it quickly multiplies the number of iterations run. To test if this is the problem, try temporarily disabling the looping events.

## GPU performance

The GPU is generally responsible solely for drawing graphics - that is, rendering your artwork to the screen, as well as any effects.

The best place to start looking to identify a GPU performance problem is the GPU profiler Paid plans only in Construct's debugger. This can break down GPU time spent on each layer in the project. If one layer is showing a high measurement, it is likely the graphics content on that layer that is responsible for the high GPU usage. For example you may see a layer using hundreds of objects all with effects showing a high measurement; removing the effects or reducing the object count could then solve the problem.

### Fill rate

A key point to understand about GPU performance is the **fill rate**. Drawing pixels on the screen (also known as "filling in" pixels) requires writing them to memory. Drawing more images, and larger images, requires writing more pixels to memory. The data rate of writing all this pixel data to memory is called the *fill rate*. Once the fill rate exceeds the GPU memory bandwidth (the rate at which data can be written), the project will start to slow down as the GPU cannot keep up. Note this is a hardware limitation, not a limitation in Construct or any other software on your device.

To fully understand fill rate, it's important to also know that Construct renders projects **back-to-front**. This means it starts by drawing the background (the objects lowest down in Z order) and

progressively drawing everything else on top until it reaches the front. Since the objects at the top of Z order are drawn last, they appear on top. However when objects overlap, this involves writing to the same pixels repeatedly. This is called *overdraw* and it still uses up fill rate, i.e. the object underneath still consumes memory bandwidth, even though it is later covered up by something on top. Therefore the worst-case scenario for fill rate is a stack of large overlapping images.

There are a couple more points to consider about fill rate:

1. Transparent areas of images still use up fill rate. In other words, transparent pixels are still rendered, they simply have no visual effect.
2. Layers which use their own texture (indicated by *Uses own texture* in the Properties Bar) must be copied to the screen after the layer finishes rendering. This means every layer that uses its own texture is equivalent to rendering a screen-sized sprite that covers up everything. Having too many layers which use their own texture can quickly use up your available fill rate.

In short, the more pixels that are drawn on the screen, the more work there is for the GPU to do. This includes transparent pixels, overlapping images, and "own texture" layers.

**GPU performance tips**

Here are some general tips about the types of things that can cause slow rendering performance. As before the way to check for these is to rely on measurements: try removing them, or significantly reducing their usage, and see what difference it makes to the measurements.

- **Avoid objects with large areas of transparency.** Crop all images you use to remove wasteful transparent space. (This also saves memory!) Split up large objects with large transparent areas in to a series of smaller objects. For example, adding a window border using a screen-sized transparent sprite with borders drawn at the edges will perform poorly as it still has to fill a large transparent area in the middle. Splitting it in to four separate objects for each edge is much more efficient since a smaller area is rendered.
- **Avoid large areas of overlap between objects.** The overlapped area will have the pixels rendered to repeatedly, which wastes fill rate.
- **Avoid too many layers which use their own texture.** Enabling *Force own texture*, changing the opacity or blend mode, or adding an effect, all cause the layer to render to its own texture, which uses a lot of fill rate. While this is necessary in some cases to get the visual effect you want, avoid doing it too many times with layers in the same layout.
- **Avoid using too many effects.** While effects can be visually impressive, adding lots of them to layers or objects can significantly increase the amount of rendering work for the GPU. Certain types of effects are also more performance intensive than others. If you have lots of objects all with the same effect, experiment with having the objects on a layer with a layer effect, or having the effect on the individual objects; sometimes one approach can be more efficient than the other.

- **Unnecessary use of effects.** Never use effects to process a static effect on an object. For example, do not use the Grayscale effect to make an object always appear grayscale. This will degrade performance when you could simply import a grayscale image to the object and not use any effects at all.
- **No hardware acceleration.** In some cases, the GPU may not be used at all, and then the CPU is forced to perform all rendering work, which is usually much slower. This is usually caused by a system-level problem, such as out-of-date hardware or software, or broken graphics drivers. It can also be affected by browser settings (e.g. turning off "Use hardware acceleration"). You can check the hardware-acceleration status in Chrome by visiting **chrome://gpu** in the address bar. You should see WebGL listed as *Hardware accelerated* in green. Construct supports both WebGL and WebGL 2; it is sufficient if either is hardware accelerated, since Construct will pick that one.

## Interpreting performance measurements

An important caveat to note about Construct's performance measurements is that the CPU and GPU measurements are **based on timers**. This makes them subject to variance due to hardware power management, which both CPUs and GPUs use.

To understand how power management affects timer measurements, consider a modern processor that can run in a low-power mode that is half as fast. In full power mode a task might take 5ms to complete. However if not under significant load, it will switch in to the low-power mode not only to save power (especially important for battery-powered devices), but also to avoid overheating the chip. In this mode the task will then take 10ms to complete. As Construct's measurements are based on timers, this means it will look like it is taking up twice as much processor time. However it is not a true reflection of the usage of the processor's full capacity.

Modern processor power management schemes are much more complicated than this, often involving many power modes that have different trade-offs between power usage and performance, and they constantly switch between them depending on the system load and temperature measurements. However the point remains that timer-based measurements can vary in unexpected ways due to power management. This can result in some unusual measurements, especially in low-power modes (when the system is mostly idle) - you might see things like the usage measurements suddenly dropping as the amount of work increases, which is actually due to the processor stepping up in to a higher power mode. The CPU and GPU usage measurements can be helpful for solving performance problems, but remember they are not perfectly accurate. In particular they are not reliable for "micro-benchmarking", such as testing which of two small tests are faster, as the measurements will mostly reflect the hardware power mode. It's another reason to not try to optimise performance until you have identified a real problem!

It's also worth noting Construct's measurements are only for your project, and the CPU measurement is only for the main thread (i.e. a single core). They are not system-wide measurements, and can be affected by other activity on the system. The CPU measurement does not measure other work done on background threads, which can also be significant in some cases, but usually does not directly contribute to the framerate.

Remember the framerate (FPS) is the ultimate performance measurement: as long as the framerate is good, then the overall performance is OK. The other measurements are there to help you diagnose what the problem might be when the framerate is dropping.

## Displaying performance at runtime

While Construct's debugger displays key performance measurements, sometimes it's useful to show these in the project itself, such as for testing with Remote Preview Paid plans only or in exported projects.

The following three system expressions provide basic performance measurements.

- **fps** - returns the current frames per second rate. The maximum framerate depends on the display refresh rate, which is commonly 60.
- **CPUutilisation** - returns the current estimated CPU usage, ranging from 0 to 1. The expression `round(cpuutilisation * 100)` will return a percentage. Note this is subject to the caveats under *Interpreting performance measurements*.
- **GPUutilisation** - returns the current estimated GPU usage, ranging from 0 to 1. The expression `round(gpuutilisation * 100)` will return a percentage. Note this measurement may not be available on some devices (in which case it will say *NaN*, which stands for Not A Number). This is also subject to the caveats under *Interpreting performance measurements*.

You can create an in-project display of these values with a Text object to keep an eye on performance while testing your project, using an action to update it *Every tick*:

Set text to `fps & " FPS, " & round(cpuutilisation * 100) & "% CPU, " & round(gpuutilisation * 100) & "% GPU"`

This will display a string like `60 FPS, 30% CPU, 40% GPU` indicating the framerate and approximate CPU and GPU usage.

## Common misconceptions

The following things are often accused of affecting performance but usually have little or no effect.

- *Off-screen objects* are **not** still rendered. Construct does not issue draw calls for objects that do not appear in the window, and the GPU is also smart enough to know not to render any content that appears outside the window - even when a single image is only partially on-screen.
- *Image formats* (e.g. JPEG, PNG or WebP) affect the download size but have no effect on runtime performance or memory use. All images are decompressed to 32-bit bitmap on startup.
- *Audio formats* also only affect the download size but have no effect on runtime performance.

- *Number of layers* usually has no effect. Layers which use their own texture have a performance overhead, as described above. However a layer with the default settings does not use its own texture, and has no performance overhead by itself. You can use as many of these layers as you like.
- *Number of layouts* also is unlikely to have any effect on performance. The *layout size* also does not have any direct effect; larger layouts do not use more memory or require more processing, unless you use more objects.
- *Angle or opacity* of objects and *floating-point positions* (e.g. positioning a sprite at X = 10.5) generally has no effect, since modern graphics chips are very good at handling this, even on weak devices. Using lots of very large sprites can still sometimes cause a slowdown - see the section on fill rate.

## Summary

In short, here are the key points when considering performance:

- Regularly test on target devices.
- Don't attempt to optimise the project if it's fast enough. You'll just be wasting your time.
- If it's not fast enough, rely on measurements to guide your optimisation.
- Frames per second (FPS) is the ultimate measurement. CPU and GPU usage are timer-based approximations and are mainly to help you make performance measurements to guide optimisation.

# MEMORY USAGE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/memory-usage

Managing the amount of memory used is important to ensure a wide range of devices can run your project. If a project uses more memory than is available on a particular device, it may crash due to running out of memory. Reducing memory use can also in some cases improve performance.

## Images

Usually object images (including sprite animations) are the most memory-consuming part of a project. For this reason Construct estimates the peak memory use from images and displays it in the *Project Statistics* dialog. (Right click the project name in the Project Bar and select Tools▶View project statistics to view the dialog.) You should check this from time-to-time while developing your project, but it should only be regarded as an estimate. The debugger can also show the image memory usage during runtime, but note that it can vary depending on what is happening in your game. Remember this measurement is only for images, so your project will need at least that much memory to run.

Some tips to ensure the image memory usage remains reasonable are:

1. **Crop all animation frames.** Transparent pixels still use up memory. If you have a 1000x1000 image that is all transparency and just a 200x200 size piece of artwork in the middle, it will use as much memory as a 1000x1000 image - about 4 MB, whereas if cropped a 200x200 image would use less than 0.2 MB - a saving of over 95%
2. **Avoid unnecessarily high-resolution images.** If you import an image that is 2000x2000, but only display it sized 400x400 inside the game, then it uses as much memory as the source image (about 16 MB), whereas if the image was more appropriately sized it would only use about 0.6 MB of memory - also a saving of over 95%.
3. **Avoid excessively long animations.** All images in an animation need to be loaded in to memory. If an animation with 100 frames can be reasonably achieved with just 50 frames, it will use half as much memory.
4. **Stay under power-of-two sizes.** Construct packs Sprite images onto spritesheets, which for technical reasons are always a power-of-two size (e.g. 512, 1024, 2048, 4096...). This means that images just over a power-of-two size, like 530px, pack inefficiently: they don't fit neatly across a spritesheet and end up wasting space. However images just under a power-of-two size, like 500px, do pack efficiently as they can fit more images on a spritesheet. Note that Construct also adds some padding so also avoid exactly power-of-two sizes like 512px - you should try to keep images at least 4px under this size to account for padding. To illustrate this, consider that on a spritesheet sized 2048x2048, an image that is 530x530 wide can only be fit in 9 times (3 across and 3 down), but an image that is 500px wide can be fit in 16 times (4 across and 4 down) - in both cases the spritesheet is the same size and so uses the same amount of memory, but the

latter case is a much more efficient use of that memory; also consider that if both cases use 16 images, the former case will then need to spill another 7 images on to another spritesheet, which will use more memory and also pack inefficiently. (Note an exception is Tiled Backgrounds are not used on spritesheets, and so can be a power-of-two size.)

5. **Use composition for level design.** Some people are tempted to use lots of unique large image tiles to design levels as if they were one huge image. This will use a lot of memory and is not how most modern games are designed. Instead use a Tiled Background to make a repeating base layer (possibly using tile randomization to make the repetition less obvious), and then design lots of individual pieces of a level that can be independently placed and re-used at different sizes and angles across the level. For example rather than designing a huge single image with a whole forest drawn on to it, use a repeating background, design a small selection of unique trees, and then place these at different sizes, scales and angles across the level to build a forest. Effects like color tint and opacity can also be used to add more variety and break up any repetition. Then the entire level's image memory usage is only the amount for the background and each unique tree, even though a much larger level was designed.

The estimated peak memory use is based on only the single layout with the largest memory requirement, because only images for one layout are loaded at a time. In Construct this is called *layout-by-layout loading*.

## Layout-by-layout loading

Construct only loads the images for the current layout. This avoids loading the entire project in memory which would be slow and consume a great deal of memory. When starting a layout, all images for the objects placed in the Layout View are pre-loaded. This includes all frames in all animations of any Sprite objects. (In other words, Sprites are either fully loaded in to memory, or not at all - they are never part-loaded.) When the layout ends, all images that are loaded but not used on the next layout are released from memory.

If an object is not placed in the Layout View, but an event sheet creates it at runtime, its images are not pre-loaded. Construct is forced to load the object images at the moment of creation, which can cause a momentary pause, or in extreme cases a constant stuttering (also known as "jank"). Construct's image memory usage estimate in the editor will not include such images as it doesn't know if they will be used, and so the memory usage of these kinds of dynamic loaded images will only be reflected in the debugger's runtime image memory measurement. A better approach is to place any objects that will be used by the layout in the Layout View. If they are not immediately needed then they can be destroyed in a *Start of layout* event. They will then not exist when the layout starts, but Construct will still have pre-loaded their images, ensuring that they can later be created at runtime without any jank.

## Calculating image memory use

First of all it is important to note the image format has no effect on memory use. You can save on your project's download size by setting some images to a different format like PNG, JPEG or WebP. However this does nothing to memory use: compressed images cannot be directly rendered, so upon loading all images are decompressed in to a 32-bit ARGB bitmap format. This

means each pixel takes four bytes for the alpha, red, green and blue channels. (Note images are also stored compressed inside your project file - so the size of your project file has nothing to do with its memory use. You can easily have a small project file with a huge memory requirement.)

Consequently, the approximate memory use of an image in bytes is its number of pixels multiplied by 4. For example a 100x100 image would use 100 x 100 x 4 = 40000 bytes, or about 39 KB. A HD-sized image at 1920x1080 would take 1920 x 1080 x 4 = 8294400 bytes, or about 7.9 MB. Note that transparent pixels still count!

It is also important to note that the memory use is based on the source image - that is, as it appears in the image editor. If the object is stretched in the Layout View or at runtime, it does not use any more or less memory. It is just rendering the source image in memory (which is always at its original size) at a different size on to the screen.

The image memory usage is based on all the images (including all animation frames) of all objects on the current layout. Therefore using fewer, smaller images will always result in less memory usage.

### Don't mis-use 'Downscaling quality'

Construct uses several optimisations, including spritesheeting, to save memory. Setting the *Downscaling* project property to *High quality* forces spritesheeting to pad out all sprites to power-of-two sizes, negating the memory saving of spritesheets. This can significantly add to the memory requirement of your project. *High quality* mode exists only to address two relatively minor rendering issues (edge fringing on sprites, or altered quality on the last animation frame). It should not be used unless one of these rendering issues has been specifically observed and the issue is resolved by choosing this option. Otherwise your project will be paying a very high price in memory usage for no reason.

## Audio

Usually images take up the most of a project's memory use. However it's worth noting how audio is loaded in to memory.

It's important to categorise audio between the sound and music folders because they are loaded differently and therefore have different memory usage.

### Sounds

Audio in the *Sounds* folder is fully decompressed in to memory. This allows sound effects to be played instantly without any latency from having to first load or decompress the audio, ensuring sound effects are heard at the appropriate time. Like with images, the compressed size helps reduce the download but does not reduce memory use: the sound will be decompressed in to PCM wave buffers.

By default the project property *Preload sounds* is enabled, meaning all sounds are downloaded and decompressed while the loading bar is showing. As a result *all* sounds in the project will be decompressed in to memory on startup. With a common playback configuration of 16-bit

samples at 44.1 KHz, one second of single-channel audio will use 88000 bytes (about 86 KB), and double that for stereo (about 172 KB). For one minute of audio, that is about 5 MB for single channel and 10 MB for stereo. This is the primary reason music tracks should not be categorised as "sound": if you have 15 minutes of stereo music, that will consume 150 MB of memory. On some platforms, decoding a full music track can also be quite slow, adding a lot to the startup time.

If *Preload sounds* is disabled, sounds are not loaded during startup and the project can start quicker. However the first time each sound is played may be delayed since it must first download and decode it in full. Using the *Preload* action in the Audio object can help mitigate this. Note Construct does not release sounds once loaded to ensure they can be played quickly again. To release the memory, you must use one of the *Unload* actions. This makes you responsible for managing audio memory in your project.

Audio in the *Sounds* folder should be short, latency-sensitive sound effects. Consider cutting down any very long duration sound effects - or, if playback latency is not important, consider categorising it as "Music" instead.

### Music

Contrary to sound effects, music is streamed. Generally this means the audio engine will have a small playback buffer of a fixed length, and while the audio is playing it is loaded, decoded and played in small chunks that connect together seamlessly. This means the memory use is low regardless of the length of the track, and it can even start playing the audio before it has finished downloading. This is why music is not pre-loaded while the loading bar is showing - there's no need to wait for it to finish downloading before starting the project. However playback cannot always start immediately, since it may need to wait for the download to finish buffering, or for the first chunk to load and decode. In terms of memory use, the main consideration is simply to make sure long audio tracks is categorised as music and not sound.

## Other resources

Many other types of resources will use a small amount of memory. However these are not usually significant relative to the memory usage of images and audio, so usually you can disregard them. Remember though that nothing in computing comes for free - for example every object that you create will use a small amount of memory, so you could also end up using lots of memory if you create tens of thousands of objects. This could even happen by accident, such as if you have an event sheet that creates objects but you forgot to add any logic to destroy them later, so it just endlessly creates more and more objects.

Anything that is pushed to an extreme will likely end up using a lot of memory, so try to make sure all aspects of your project remain in reasonable proportions, and it is unlikely you will have any problems with anything else.

# OPTIMIZING DOWNLOAD SIZE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/download-size

Construct's engine has a very low overhead - an empty and fully-optimized exported project is around 300 KB compressed. Most of the download size of your project will come from the content of your project. While many people have fast Internet connections, download size can still be an important consideration: some people in certain places or circumstances still have slower connections; some platforms have file size limits; and the faster people can start running your project, the more happy customers you'll have. This guide includes some advice about reducing your project's download size.

## Check your download size

A quick way to check your project's download size is to export it, extract the zip to an empty folder, and then check the size of the files in the folder (for example in Windows, select all, right-click and choose *Properties*). Then you can also browse through the files looking for any particularly large files or folders to help identify what you need to focus on to best reduce the download size.

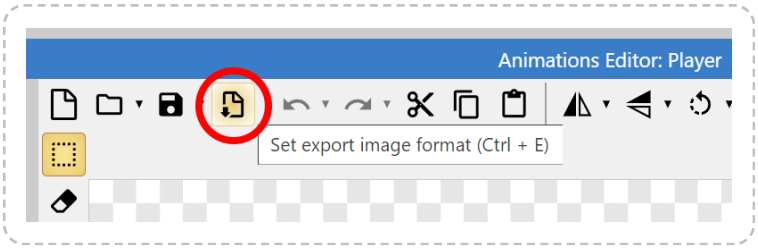
## Images

Often much of a project's download size come from its images. A key point to be aware of is the difference between **lossless** and **lossy** image formats. *Lossless* means the image quality is preserved exactly, with pixel-perfect reproduction of the original image. This has the best quality but such images can only be compressed so far, particularly if they have complex content. *Lossy* images do not preserve the image quality exactly, but consequently can reduce the file size much further than lossless images. There is also a quality option for lossy images, allowing you to choose the trade-off between file size and quality.

### Use lossy formats

The easiest way to reduce the download size of your project's images is to switch them to use a lossy format. By default Construct uses lossless compression to avoid affecting your artwork. Choosing a lossy format allows for further compression, although it does affect the display quality. Usually large, photographic-style images, or background images where quality differences may not be noticed, are best suited to lossy formats.

To set an image to use a lossy format, use the **Set export image format** button in the image editor toolbar.




You can apply the changes to individual images, the entire animation, the entire object, or even the entire project. Switching an entire project to use lossy images is a surprisingly effective approach, particularly with the AVIF image format. See the blog post Reducing download sizes with lossy spritesheets for more analysis. A good technique may be to set an entire project to use lossy images with a relatively low quality, and then after exporting look at your project and identify if you can see any places the quality isn't sufficient. Then for those specific objects you can either increase the quality, or switch it back to lossless compression. This should leave your project using the most efficient compression possible for images, while achieving a sufficient visual quality.

### Use modern formats

When you export your project, the smallest download size can be achieved by choosing WebP and AVIF for the export image formats. PNG and JPEG are both quite dated and produce larger file sizes for the same quality. AVIF is particularly effective for lossy images.

### Other export settings

When you export your project, be sure to also enable *Deduplicate images* (which removes redundant identical images), and *Optimize images* (which will use more processing time to compress images, so export is slower but the resulting files are smaller). Both will help you have the smallest download size possible for your images.

### Use fewer and lower resolution images

The smallest image file is one that is not exported at all - if you can, remove any redundant images, and reduce the number of frames in animations. Smaller images also compress better, so try resizing images smaller in the image editor. In particular, avoid having highly detailed images at a higher resolution than necessary.

## Audio

Sound effects are usually relatively brief in duration and so don't produce large files. However music tracks and longer sound effects may end up using a lot of data.

The best way to reduce the download size of audio is to choose the lowest acceptable bitrate when importing audio to Construct. The import dialog has a dropdown to select the bitrate in the corner; lower bitrates produce smaller files, but will have poorer quality. Note that Construct encodes audio to WebM Opus, but if you import a WebM Opus file directly, it uses the file as-is. So the best way to control the bitrate is to import uncompressed audio (such as PCM WAV).

You can also save your project as a folder, and use external audio tools to try compressing audio different ways - but note it must be in WebM Opus format. Another trick is reducing stereo or multi-channel audio to mono (single-channel). Lastly, using fewer audio files, and shorter sound effects and music tracks, will mean less data too.

## Video

Video files can be particularly large to download. Construct doesn't include its own video encoder like it does with audio. Try to use shorter video clips at lower resolution and with a lower framerate. You can also use external video tools to try to better compress video files, with a trade-off between bitrate and quality, much like with audio. Also prefer to use modern codecs where possible - for example AV1 is now pretty widely supported, and will be able to achieve a smaller download size at the same quality as an older codec like VP8.

## Other project files

You can import any files you like to Construct in the Project Bar. It's worth checking in case you imported any particularly large files of other kinds! These won't necessarily need to be downloaded up-front if your project only loads them later on, but it's worth remembering them too.

## Other tips

Here is some other advice relating to reducing the download size or wait time for your project.

### Minify script on export

When exporting your project, choosing the *Advanced* script minify option will produce the smallest possible JavaScript files. For large projects this will make a smaller difference than optimizing images, audio and video, but it will help achieve the smallest download size possible.

If you write JavaScript code in your project, note you may need to make changes to use *Advanced* minify mode - see the manual section on advanced minification. Alternatively *Simple* mode should always be safe, but won't reduce the JavaScript file size as much.

### Use a loader layout on the web

For web exports, consider using a loader layout. This is a special layout that can start running while the rest of the project is still downloading. Construct only needs to download the content shown on the loader layout before it can start it. This means the user can see something interactive much more quickly, even if the overall download size is the same. See the tutorial How to use loader layouts to make custom loading screens.

Note if you aren't publishing to the web and are using an "offline" export option like a desktop export, then there's no need to use a loader layout - the entire project is already downloaded by the time it starts running.

### Avoid bundling browser engines

Where possible, avoid using export options that bundle the browser engine with your project. With Construct's modern export options, this generally only means Windows WebView2 in "fixed" distribution mode (but the default uses the system webview so doesn't bundle a browser engine), or the Linux CEF export option. Bundling an entire browser engine with your project can add 100-200 MB to the download size, even compressed. On Linux the browser engine has to be bundled as there is no consistent system webview support, but consider a web export as an alternative.

# ICONS & SPLASH SCREENS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/icons-splash

The **Icons & screenshots** folder in the Project Bar holds a set of image files that are used for icons, splash screens or screenshots for your project. The specific images used depend on the platform you export to, but where possible Construct will automatically use the images from the folder where appropriate.

## Image purposes

When you select an image in the *Icons & screenshots* folder in the Project Bar, the Properties Bar updates to show properties for that image. The most important property is the *Purpose* property. This tells Construct what you want to use the image for. The different purposes and what they are used for are described below.

### None set

If the purpose is *(none set)*, Construct won't automatically use this icon for anything. It will be treated as a simple image file that is bundled with your project. This may be useful if you want to manually configure an icon after exporting. It's also the default for newly imported image files, so you may want to change it after importing if you intend for the icon to be used for something else.

### App icon

This will use the image as the standard app icon. This is used for apps on mobile devices, desktop exports, the page icon on the web, and also the app icon for installable web apps.

The icon size is automatically derived from the image size. For most platforms, icons should be square, and are typically in a range of power-of-two sizes, such as 32x32, 128x128, 256x256, etc. Most platforms support a range of icon sizes, so Construct will list all icons with this purpose in the exported project, and the target platform will select an appropriate icon size from the available set.

### App icon maskable

Currently this purpose is specific to installable web apps (Progressive Web Apps, or PWAs). All web exports from Construct are PWAs, and so in supported browsers can be installed, such as adding to the device home screen. The *App icon* purpose can be used for this case, but optionally you can also provide a *maskable* icon, where the outer 10% edges of the image may be cropped. This is used to give the system more flexibility in how to crop or shape the icon. As with app icons, you can also provide multiple maskable icons in different sizes.

You can test maskable icons with maskable.app, and read more about them at the web.dev page on maskable icons.

### Loading logo

The loading logo is the image shown on the loading screen while loading the project. Typically this is most relevant to web exports, since the project may take a while to download, during which time the loading screen is showing. Other export types, like apps, bundle all the resources locally so typically load quickly enough that the loading screen isn't seen or is only seen briefly.

For the loading logo to appear, the project *Loader style* property must be set to *Progress bar & logo*. The progress bar will appear beneath the loading logo with the same width as the loading logo image. You can also only specify a single loading logo image.

### Adaptive foreground/background

Currently these purposes are specific to Android exports, for specifying adaptive icons. Similar to maskable icons, these are special icons for Android that comprise of two layers, and may also be cropped by the system to adjust their appearance. For more information about how Android uses these icons, refer to the Android developer documentation on adaptive icons.

To specify an adaptive icon, you need to add two icons with identical image sizes, e.g. both 256x256 pixels. Set one to use the *Adaptive background* purpose, and the other the *Adaptive foreground* purpose. Construct will export the icon pair as a single adaptive icon when exporting for Android.

You can also specify multiple adaptive icons by adding multiple pairs of icons this way, so long as each pair has a different image size. Construct will export all icon pairs as adaptive icons for Android, and the system will select an adaptive icon with an appropriate size from the available set.

### iOS splash screen

This purpose is specific to iOS exports. The image with this purpose is used as a universal splash screen image. Note this image may be significantly cropped depending on the size of the viewport - the image is displayed centered so any important details should be in the middle, and the rest of the image decorative.

When creating the splash screen, Construct will first create an image sized 2732x2732 filled with the project *Splash color*. The splash image will then be pasted centered on this image. Note if no splash image is specified, the image with the *Loading logo* purpose will be used instead.

### Android splash icon

This purpose is specific to Android exports. The image with this purpose is used on the splash screen when the app is first loaded. Note this image may be significantly cropped by Android. For guidance on the required image size and how Android crops it, refer to the official Android design guidelines.

### Screenshot

This purpose indicates that the given image is a screenshot of the project, suitable for using as a preview of what the project looks like. Currently this is only used for web exports in order to allow adding screenshots to a Progressive Web App (PWA) install prompt, which is also known as the Richer install UI. When choosing the *Screenshot* purpose for an image, an additional two properties appear:

- **Label:** a brief description of the image, which may be used as a caption for the screenshot.
- **Form factor:** indicates the type of the screenshot. Typically *Narrow* is used for mobile screenshots and *Wide* is used for desktop screenshots. It is recommended to add at least one *Narrow* and one *Wide* screenshot to ensure the richer install UI appears.

## Manually customizing icons and splashes

Some platforms, particularly on mobile, require a range of icons and splash screens in a wide variety of sizes and formats. These formats can also change over time as system updates are released with new requirements.

While Construct provides options to specify the basic icons and splashes, for full control you may wish to refer to the platform specific developer tools. For example when using Xcode, or exporting to Android Studio, you can use the developer IDEs to specify the full range of possible icons and splash screens. For more information about using these tools, refer to the official documentation at Apple's Xcode documentation and Android Studio's documentation. Note the *(none set)* icon purpose may be useful for bundling image files that you manually configure after export in these tools.

# HTML LAYERS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/html-layers

Construct supports integrating HTML content with your project. This includes the HTML Element object, as well as other plugins in the *HTML elements* category, such as Button and Text Input. Layering HTML objects works differently to other kinds of objects. In particular, to have other Construct objects appear on top of HTML objects, it is necessary to use **HTML layers**. This guide explains how HTML layers work.

Using HTML layers is useful for better integration of HTML content in your project. For example if you want to use HTML for part of your project's user interface, but then have some decoration like a Particle effect appear on top of the user interface, then it is necessary to use HTML layers to get the Particle effect to appear on top of the HTML content.

## Canvas vs. HTML objects

Most Construct objects, such as Sprite, Tiled Background and Particles, render in to a `<canvas>` element. These kinds of objects are collectively referred to as *canvas objects*. The canvas element is a single HTML element that effectively acts as a large image that changes every frame. Much like the usual `<img>` (image) element, it can only be placed entirely in front or behind other HTML elements - there is no way for it to appear both partly in front and partly behind another HTML element.

HTML objects like Button and Text Input are themselves represented by other HTML elements (`<button>` and `<input>` respectively). These kinds of objects are collectively referred to as *HTML objects*. They are not drawn in to the canvas. Therefore they can only appear in front of, or behind, a canvas element.

By default Construct creates a single canvas element, and all other HTML elements are placed on top of the canvas. For example if a project uses a Sprite and a Button object, Construct will use the following stack of HTML elements (in top-to-bottom order):

- `<button>` for the Button object (on top)
- `<canvas>` for canvas objects like Sprite (underneath)

With this arrangement, it is not possible for the Sprite object to appear on top of the Button object, even if the Sprite is on a layer above the Button object, or if it is on top in Z order on the same layer. In other words, the HTML layering takes precedence over Construct's Z order.

## HTML layers

It is possible to have canvas objects render on top of HTML objects by making use of Construct's *HTML layers* feature. Checking the *HTML elements layer* property of a layer will turn

that layer in to a HTML layer, and indicate this status in the Layers Bar with a special tag icon. Then content on other layers above it will appear on top of HTML objects on that layer.

This works by creating an additional `<canvas>` element per HTML layer. For example consider the following arrangement of layers:

- **Layer 1** with a Sprite object
- **Layer 0** with a Button object

Normally, the Button will appear on top of the Sprite object even though it is layered beneath it, due to the reason described above. However if *Layer 0* is made a HTML layer, it is now possible for the Sprite to appear on top of the Button. This is because Construct now creates the following HTML elements (in top-to-bottom order):

- `<canvas>` for Layer 1, with a Sprite object
- `<button>` for the Button object
- `<canvas>` for Layer 0, with any other canvas objects

Note that HTML objects still appear on top of any other canvas objects on the same layer. However canvas objects on all layers above a HTML layer can then appear on top of the HTML objects.

If multiple HTML objects are on the same HTML layer, then they are ordered in between the two canvas elements, respecting their own relative Z order. In actual fact Construct uses a `<div>` element wrapper to contain all HTML content in between canvases. This also ensures HTML content is clipped to the canvas area, so they cut off when moving to the edge of the screen like canvas objects.

## The implicit top HTML layer

The top layer is always implicitly a HTML layer. In other words, if you have no HTML layers, then HTML objects appear on top of all canvas objects, as all HTML elements are placed on top of the `<canvas>` element by default, as illustrated in the first example in this guide.

## Multiple layers

HTML objects on non-HTML layers will visually appear on the next HTML layer above them in Z order. To illustrate this, consider this arrangement of layers.

- Layer 7 (normal layer)
- Layer 6 (normal layer)
- Layer 5 (HTML layer)
- Layer 4 (normal layer)
- Layer 3 (normal layer)

- Layer 2 (HTML layer)
- Layer 1 (normal layer)
- Layer 0 (normal layer)

This arrangement uses two HTML layers on Layer 2 and Layer 5. In this case, all HTML objects on Layer 0, Layer 1 and Layer 2 will appear on Layer 2 (above all other canvas objects on layer 2); all HTML objects on Layer 3, Layer 4 and Layer 5 will appear on Layer 5; and all HTML objects on Layer 6 and Layer 7 will appear on top of those, as there is an implicit HTML layer at the top. This is because Construct creates the following HTML elements (in top-to-bottom order):

- HTML elements for layers 6-7
- `<canvas>` for canvas objects on layers 6-7
- HTML elements for layers 3-5
- `<canvas>` for canvas objects on layers 3-5
- HTML elements for layers 0-2
- `<canvas>` for canvas objects on layers 0-2

In this case, layers 0-2, layers 3-5, and layers 6-7 can each be thought of as separate HTML layers. So in this case there are three HTML layers, as there are three options for where HTML objects can appear relative to canvas objects, whereas there are eight canvas layers as there are eight options for where canvas objects can appear (which are distributed across three canvas elements).

Note that HTML objects always appear on top of all canvas objects on the same HTML layer. A HTML layer can be thought of as a canvas element with all canvas objects paired with a layer of HTML objects above it. Within a HTML layer, HTML objects will respect their relative Z order. For example if a Button is layered on top of a Text Input in the same HTML layer, then Construct will ensure the Button object's HTML element is layered on top of the Text Input's HTML element.

Only top-level layers can be made HTML layers. With this method of stacking canvas elements and other HTML elements, it is not possible to support making sub-layers HTML layers.

## Effects

When using effects, note they can only process content on the same canvas. This usually affects background-blending effects. Taking the previous example of multiple layers, if there was a background-blending effect on layer 5, then it will only be able to blend with content on layers 3-5. Layers below that are on a different HTML layer and so render to a different canvas, and thus the background blending effect is not able to process them. This is a side-effect of the way the rendering composition works: as each canvas is rendered separately, it is not possible for effects to work across them.

## Performance overhead

As illustrated, for each layer which you make a HTML layer, an additional canvas element is created at the size of the viewport. This comes with a performance overhead. It will impact two aspects of performance:

1. Each canvas must be copied to the display every frame. This has the effect of drawing a viewport-sized surface, much like a *Force own texture* layer. This uses up the GPU fill rate (see Performance Tips for more details about fill rate). In some cases the compositing process may require more than one copy, making it potentially two or three times as costly as a *Force own texture* layer.
2. Each canvas must be allocated in memory, which will use at least as much memory as a viewport-sized image. For example a 1920x1080 size canvas will require at least 8 MB of memory (see Memory usage for more details). In some cases the compositing process may require more than one surface, making it potentially use two or three times as much memory.

For this reason, avoid using too many HTML layers. Try to use the minimum necessary number of HTML layers to achieve the layering you need for your project.

## Positioning

HTML objects use a size and position based on the Construct layer they are on. In other words they match the size and position that they would have if they were a canvas object. Therefore if HTML objects are on layers with different scroll positions, scales, or parallax, they will match the position of the layer they are on, even though they are displayed on the next HTML layer above them.

Browsers do not always update canvas elements and other HTML elements at the same time. You may find that if you move both canvas objects and HTML objects simultaneously at high speed then a visible difference appears between them as one lags behind the other by a frame or two, due to the browser drawing them at different times.

## Dynamic layers

It's possible to dynamically add or change HTML layers using the *Set layer HTML* action or `isHTMLElementsLayer` script property. This will cause Construct to add or remove canvas elements, and rearrange the layering of HTML elements, to reflect the changes. However this is complicated to synchronize, especially in worker mode where changes cannot be made synchronously. This may result in a brief flicker as HTML changes are made. This may be able to be avoided by ensuring sufficient HTML layers are created in advance of filling them with content, or using techniques such as fading in content so it initially starts invisible and so a flicker is not noticable.

# INSTALLING THIRD-PARTY ADDONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/installing-third-party-addons

Third party developers can extend Construct 3 with new plugins, behaviors, effects and themes (collectively referred to as "addons") using the Addon SDK. Addons are typically distributed as a **.c3addon** file.

***Only install addons from trustworthy developers who actively support their addons.*** *Malicious addons have the potential to compromise the security of your project, your Construct account, or have hidden unwanted features like surprise adverts or tracking users. Badly written addons can also cause bugs or glitches in your game,* ***including corrupting your project****. While addons can be useful, remain vigilant about them, especially in regards to whether the developer seems trustworthy and if they are still available and actively supporting their addon to provide support and fix any problems that arise.*

If you have problems with third-party addons, you must report the issues to the developer who provided them. Scirra cannot offer any support for third party addons whatsoever.

## Finding addons

You can visit the Addons section of the website to find plugins, behaviors, effects and themes created by third-party developers. When you download an addon, you'll receive a .c3addon file, which you can then install via the Addon Manager.

## Installing an addon

To install an addon from a .c3addon file, first select Menu▶View▶Addon manager to open the Addon Manager. In this dialog, click **Install new addon...** and choose the .c3addon file. Construct 3 will prompt to confirm installation of the addon. If you confirm the install, you must restart Construct 3 before the addon is available. In the browser you can just press the Reload button.

Modern addons (using SDK v2) can also be installed by directly dragging and dropping the .c3addon file in to Construct. Note however this is unavailable for legacy addons (using SDK v1).

## Updating addons

By default Construct will check if any updates are available for installed addons via the Addons website. If updates are available, it will show a notification on startup. Addons can then be updated via the Addon Manager.

*Addons installed from other sources rather than the Construct Addons website will not show update prompts.*

## Uninstalling an addon

Third-party addons are listed at the top of the Addon Manager. Locate the addon, right-click on it (or tap-and-hold), and select the **Uninstall** option. You can uncheck *Show built-in addons* or use the search field to help you locate an addon.

## Bundling addons to projects

Paid plans only Enable the *Bundle addons* project property to include any third-party addons the project uses in the saved project file. This allows you to open the project on another device without having to have the same third-party addons pre-installed, since the addons will be loaded directly from the project file.

Sometimes you'll need to update a bundled addon to the latest version, if you update the installed addon in Construct. To update a bundled addon, open the *View used addons* dialog by right-clicking the project name in the Project Bar and selecting Tools▶View used addons. In this dialog you can right-click a bundled addon (highlighted in bold) and select Update to editor version.

# MIME TYPES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/mime-types

When publishing to the web, it is important the server sends certain types of file with the correct MIME type. For example, most servers are correctly set up to send a **.html** file with the MIME type **text/html**. However some server's defaults don't include every MIME type your project might need to use. Also some servers may be configured to send the wrong MIME type, which may still cause some features to work incorrectly. The list below should be used as a reference of the correct MIME types to have set.

Configuring your server's MIME types depends on your host. If you don't know how to do this, contact your host for support, or ask them to set up the list below for you.

Having the wrong MIME type set can result in problems like the browser refusing to load the file; the file returning "404 Not Found"; or causing the server to send a corrupt file (e.g. incorrectly sending an audio file as a text file). This can result in issues like audio playback not working, AJAX requests failing, or the project failing to start up.

## MIME types for Construct projects

A server hosting an exported Construct project should have these file extensions associated with these MIME types.

| File ext. | MIME type | Notes |
|---|---|---|
| .html | text/html | **Required** |
| .js | application/javascript | **Required** |
| .json | application/json | **Required** |
| .css | text/css | **Required** |
| .wasm | application/wasm | **Required** |
| .png | image/png | **Required** |
| .jpg, .jpeg | image/jpeg | **Required** |
| .webp | image/webp | **Required** |
| .avif | image/avif | Optional, if AVIF images used |
| .webm | video/webm | **Required** (also covers WebM audio) |
| .m4a | audio/mp4 | Optional, if AAC audio used |
| .mp3 | audio/mpeg | Optional, if MP3 audio used |
| .ogg | audio/ogg | Optional, if Ogg Vorbis audio used (common in Construct 2 projects) |
| .mp4 | video/mp4 | Optional, if MP4 video used |
| .woff | application/font-woff | Optional, if web fonts used |
| .woff2 | font/woff2 | Optional, if web fonts used |
| .txt | text/plain | Optional, for data files |
| .csv | text/csv | Optional, for data files |
| .xml | text/xml | Optional, for data files |
| .svg | image/svg+xml | Optional |
| .scml | text/xml | Optional, for Spriter animations |
| .scon | application/json | Optional, for Spriter animations |
| .c3p | application/zip | Optional, for Construct project files |

## Offline caching

If you find there is a problem with your MIME types after exporting and then fix them, the problem may not appear to be immediately corrected if the browser has cached the previous server responses for offline support.

The easiest way to verify the problem is fixed after correcting a server's MIME types is to change the URL of the project, e.g. renaming the folder it was in on the server. This prevents the previous offline cache being used to load the game and it will start fresh with the new MIME type configuration taking effect.

## More information

For more reading, see the Mozilla Developer Network (MDN) article on Configuring server MIME types.

# CONSTRUCT RELEASE CHANNELS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/release-channels

You can find every release of Construct, past and present, on the releases page. Construct releases are split in to three types of releases, called *release channels*: **beta**, **stable** and **LTS** (Long Term Support). Broadly speaking, beta releases are the most frequent releases but most likely to have unexpected issues, and LTS releases the most infrequent releases and least likely to have unexpected issues. This guide briefly describes the differences between them and why you might want to choose each.

## Beta releases

Released about once a week, beta releases are mainly for testing purposes. Beta releases haven't had as much testing as other releases, so unexpected problems are more likely to come up. Be prepared for occasional disruption. You should probably not publish a finished project from a beta release. However these releases get the latest bug fixes, new features and other improvements the soonest. You may want to try a beta release to verify that a bug fix works correctly for your project. People who want the latest features can use beta releases to try out the latest improvements. Enthusiasts, or anyone else who wants to shape the development of Construct, may also be interested in using beta releases and providing feedback to help us make Construct in to what you want and need it to be - beta releases are where the main active development of Construct takes place.

Occasionally very serious bugs may warrant a patch release of a beta release (i.e. a .2 release the same week), but in general you should assume any issues will only be fixed in the following week's beta release at the earliest.

You can load the latest beta release by visiting: editor.construct.net/beta

## Stable releases

Released every 2-3 months, stable releases are the culmination of previous work on beta releases. Once a suitable batch of work has been completed and any significant known issues are identified and resolved, a new stable release is published including all the changes from the past beta releases since the previous stable release. By this point all the changes are much more likely to be reliable and bug-free. However sometimes mistakes happen and new issues appear. Any major issues will likely be fixed in follow-up patch releases (i.e. a .2 or .3 release). Other issues may be dealt with in subsequent beta releases and so only reach the next stable release.

Most people use stable releases - when you visit editor.construct.net in the address bar, it will load the most recent stable release.

The time in between stable releases is called a release cycle. Usually the few beta releases prior to a stable release stop introducing new features and instead focus solely on bug fixes and

reliability to ensure the next stable release goes smoothly.

## LTS releases

LTS stands for Long Term Support. These are based off a mid-year stable release (around June or July), and are supported with essential bug fixes and maintenance updates only for a period of 18 months (so until the end of the following year). The aim is to have as few changes as possible to ensure a consistent platform for purposes like testing and launching a major new project, or in education to minimize any potentially disruptive changes during the academic year. LTS releases start out as a stable release, but before the next stable release comes out, the stable release is "promoted" to an LTS release.

Most people should use stable releases over LTS releases. You should only choose an LTS release where the importance of predictability outweighs the importance of bug fixes and the latest new features. Software development is complex, and sometimes new bugs can accidentally make their way to new stable releases. It's even possible a project depends on a bug to work correctly, and fixing the bug causes the project to stop working; in a large project, fixing such issues can sometimes be hard work. Sometimes changes which affect compatibility are intentionally made for the long-term benefit of Construct. Suppose you are preparing to launch a game that has been in development for a few years - in this case the last thing you want is some new bug or compatibility difference to break your game just before launch. This is a good situation to choose an LTS release: as you approach launch and start to focus on testing your game, you may want to switch to an LTS release to minimize any unexpected changes to Construct.

The downside of using an LTS release is that you won't get access to the latest new features and updates, nor will you necessarily get the latest bug fixes either. If you are wondering if any particular fix or change will be applied to an LTS release, the default answer is "no". Every change brings a risk of breakage - as noted even bug fixes can cause breakages - and in order to provide a consistent platform, LTS releases will change as little as possible. In general only fixes which are absolutely critical will be applied to LTS releases. Generally this involves a new requirement makes it impossible to publish projects any more unless something is changed. For example if mobile publishing requirements are updated and a change must be made to continue publishing mobile apps, or if a browser bug or compatibility change meant existing projects were broken, that would warrant an update to an LTS release. In virtually all other cases, the change will only be made for the latest beta release and be rolled out to everyone at the following stable release. Even where LTS updates are made, they may only be some time after the change is made to other release channels, in order to verify that the change works as expected.

If you use LTS releases, there is a chance you run in to some bug that is only fixed in the latest stable release and does not meet the high bar to be applied to the LTS channel. In this case you will have to update to the latest stable release. Therefore whether to use an LTS release, and when to start using it, depends on your judgement of the trade-off between regular maintenance and a predictable platform to publish with.

If you use an LTS release for more than a year, you may wish to consider switching to the LTS release that comes out the following year. This choice depends on how far through your project

you are: if publishing is still some way off, it is probably a good idea to update and deal with any compatibility differences; if your project has already been published and you are just maintaining it and don't think you'll need many more updates, then you may want to just stick with the same LTS release.

You can load the latest LTS release by visiting: editor.construct.net/lts

## Release numbering

Construct's release numbering scheme merely increments the release number for every beta release, and is prefixed with an "r" for "release". For example a beta r400 will be followed by another beta r401 about a week later. Every 2-3 months, a new stable release is made continuing the incrementing release numbers. Occasionally "patch" releases are made to apply changes to an existing release. These increment a number after a dot - e.g. r407.2 is a patch release for the original r407 release.

Once a year the latest stable release is "promoted" to a new LTS release. As LTS releases are supported for longer, they will tend to get more patch releases, such as r397.2, r397.3, and so on.

## The life of a change

Suppose a change is made to Construct, such as a bug fix. Normally such a change will work its way through the release channels as follows.

1. If the change is for a critical issue, it may be released as a patch release for beta, stable or LTS releases. (Note the LTS release may be updated some time after other channels to verify the change works as expected.)
2. Otherwise the change will first appear in the following weeks' beta release.
3. Within a couple of months, the next stable release will be published including the change.
4. At the next annual update, the current stable release will be promoted to LTS, including the change.

The release channel you choose depends on the trade-off you want between the latest bug fixes and new features, and the chance that unexpected problems could cause disruption to your project.

# SUPERSEDED FEATURES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/superseded-features

Construct has been in development for many years. Over time new features are occasionally introduced that supersede older features and essentially make them redundant. Sometimes in this case the older features are phased out and ultimately removed from Construct. For this case, see Deprecated features. However in some cases these features are left in Construct indefinitely for backwards compatibility reasons. For example the newer feature may not have exactly the same capabilities as the older feature and so cannot fully replace it in every possible usage scenario. Alternatively the older feature may have been around for such a long time that removing it would require updating vast amounts of documentation, including printed books, third-party tutorials, lesson plans used in educational institutions, and so on, which is on the whole infeasible and so it's better to leave the older feature in.

Where older superseded features are left in, it's perfectly acceptable for existing projects, guides and so on to keep using them. However new projects should avoid using them in favor of the newer feature that supersedes them. Further, usually the older feature will not be supported to the same extent as the newer feature, such as fixing possible issues or adding new requested features, as we would instead recommend using the feature that supersedes it. In order to help Construct users know which features to prefer and avoid, this guide lists the features we consider superseded.

## Pin behavior

The Pin behavior was widely used for many years, but it is almost completely superseded by the Hierarchies feature. The 'Add child' hierarchy action is similar to pinning the child on to the parent, but works more reliably when chains of objects are connected. Hierarchies can also be set up in the Layout View.

## Fade behavior

The Fade behavior was widely used for many years, but is superseded by the Tween behavior. The Fade behavior essentially runs a pre-defined series of opacity tweens, which is now better done in a more general way with the Tween behavior.

## Deprecated features

See also the section on Deprecated features, which covers features that will be completely removed in future.

# DEPRECATED FEATURES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/deprecated-features

Construct has been in development for many years. Much like with superseded features, over time new features are occasionally introduced that replace older features. Usually the older features are supported as long as possible to improve backwards compatibility. However sometimes the maintenance burden and additional complications caused by having to continue to have the older features means that they must ultimately be retired.

When you open a project, Construct will warn you of any deprecated or retired features the project uses. The *Deprecated features* dialog may appear when opening a project to list all such features. It is strongly recommended to act as soon as possible to resolve all the items listed, thereby removing the use of all the deprecated features from your project. This will ensure your project continues to be supported and work correctly in future. If you don't take action, you may find your project stops working, or possibly cannot be opened, in a future release of Construct.

You can view the deprecated feature list for a project at any time, including if you previously checked *Don't show again for this project* in the dialog, by right-clicking the project name in the Project Bar and selecting Tools▶View deprecated features. If the option does not appear, it means the project does not use any deprecated features.

This guide covers every type of deprecation or retirement warning and describes steps to take to stop using the affected feature.

## Legacy SDK v1 plugins/behaviors

The industry-standard approach for addon systems is to use encapsulation to limit what addons can do to known safe features only, which ensures they continue to work in the long-term. Historically, partly due to past limitations of the JavaScript programming language, Construct did not use encapsulation in its addon system. This meant addons could use unsafe features that they were not meant to. The proliferation of addons using unsafe features ended up breaking a lot of user's projects, and continued support for those addons meant facing the prospect of continuing to break user's projects on a regular basis. This old system is referred to as SDK (Software Development Kit) v1.

To fix this Construct moved its addon system to an industry-standard approach with encapsulation. The new system is referred to as SDK v2. This should guarantee that addons are permanently supported and safe to use in the long-term with virtually no risk of breaking your project. However plugins and behaviors must be updated by the addon developer to support the newer SDK v2.

As of Construct r450+, released in 2025, support for SDK v1 addons was removed. Construct will notify you that you cannot open a project if it uses SDK v1 addons. If you see such a message,

consider the following options to resolve it:

- Addons installed from the official Addons website should auto-update to the latest version. If you see a prompt about addon updates being available on startup, click the notification and update addons with the Addon Manager. If the addon developer has published an update using SDK v2, this will then update the addon to that version.
- If the addon was not installed from the official addons website, check with the addon developer to see if an update is available, and manually download and install the updated addon.
- If there is no update for the addon using SDK v2, contact the addon developer and ask them to provide an update.
- Consider replacing the addons with other features. For example if your project uses a third-party addon for tweening, it may be possible to use the built-in Tween behavior instead. Alternatively a different third-party addon may be available that uses SDK v2. It may even be the case that you can just remove the addon if it turns out it is not really necessary.
- Otherwise SDK v1 addons are still supported in the r449.x LTS releases, which will be supported up until the end of 2026.

The SDK v2 was first made available to addon developers in May 2024. LTS support up to the end of 2026 should provide plenty of time for addon developers to update their addons, or for existing projects to either be updated or completed.

## Export file structure 'Flat' mode

Historically Construct exported projects with all project files in the root folder and with lowercased filenames, referred to as *flat* mode. In 2022 Construct was updated to preserve the folder structure and filename case of project files, referred to as *folders* mode. This is a much better mode as it preserves your folder organization and is important for features like JavaScript Modules. However changing the mode could break some projects, as it changes where some project files are found, such as when requesting project files by URL.

For example consider a project using a file named "Hello.txt" in a subfolder named "myfolder". The file path is in fact "myfolder/Hello.txt". However in *flat* mode, where project files are referred to by a string, this file could be loaded from the path "hello.txt", as it exports all files in the root folder with lowercased names. After changing to *folders* mode, the file must be referred to with the string "myfolder/Hello.txt", reflecting its true path; the old string will no longer refer to the file and may return an error such as 404 Not Found. This also applies to audio files referred to by a string, although the file extension is omitted for those.

In r450+, released in 2025, the legacy *flat* mode was removed, and all projects automatically use *folders* mode. Older projects may show a warning that the project used to use *flat* mode, which may mean you need to update any changed file references as described above.

## Cordova iOS/Android scheme

In the past mobile exports internally ran on the *file:* scheme, much like opening a local HTML file on your computer in your browser. This mode is inefficient, has limited capabilities, and is extremely difficult to support due to severe technical restrictions on the limited features available with the *file:* scheme.

In 2020 Construct added support for the *app:* scheme for iOS, and in 2021 the *https:* scheme for Android. These modes work almost identically to the way a real HTTP server does, and are much more efficient, more capable, and easier to support. In r450+, released in 2025, support for the legacy *file:* scheme was removed, and all projects automatically use the *app:* or *https:* schemes. Older projects may show a warning that the project used to use the legacy *file:* scheme.

***Warning:*** *publishing a mobile app with a changed scheme will have the effect of clearing storage, as it changes the URL used internally to load the project, and storage is remembered based on the URL. Therefore you should not publish an update to an existing app that changes the scheme if that app uses storage. If you still need to maintain an already-published mobile project using the legacy file: scheme, you should continue using the r449.x LTS releases to update your app, which are the last releases that still support the file: scheme.*

## NW.js exporter

For many years Construct used a framework called NW.js for its desktop exports, along with an NW.js plugin for further integration. Modern versions of Construct switched over to separate Windows, macOS and Linux exporters using different technologies, and as of r450+ released in 2025, support for NW.js has been retired. The Browser plugin provides most of the windowing features of the old NW.js plugin, and the File System plugin provides access to local files and folders. The legacy Greenworks plugin for Steam integration with NW.js has also been replaced by the Steamworks plugin which works with the modern export options.

## Deprecated plugins/behaviors

Construct will list any plugins or behaviors used in your project that are marked deprecated. This means the entire plugin or behavior is no longer supported and should not be used any more. Usually this is because it has been replaced by a newer addon or feature, or the addon was for a third-party service that was shut down. If your project uses deprecated addons, you should delete them from your project, and where applicable replace them with a newer addon or different feature.

To remove a deprecated plugin, all object types based on the plugin must be deleted from the Project Bar. To remove a deprecated behavior, all objects using the behavior must have the behavior removed.

To help you identify where a deprecated addon is used, follow these steps.

1. Right-click the name of the project in the Project Bar.

2. Choose Tools ▶ View used addons.
3. Find the deprecated addon in the list.
4. In the *References* column, the first few object names using the plugin or behavior are listed. To see a comprehensive list, right-click on the addon and choose Find all references....

Deprecated plugins and behaviors are hidden from the list when adding a new object or behavior. Therefore any plugins or behaviors you can still add from Construct are not deprecated. Any of these can be used as replacements for deprecated addons.

### Functions

The legacy Function plugin will also be listed as a deprecated plugin. In 2019, Construct 3 introduced a new built-in functions feature which replaces the old Function plugin. Projects using legacy Functions should switch to using the new built-in system.

To help with converting to built-in functions, you can right-click an *On function* condition in the old Function plugin, and select *Replace with built-in function*. Note due to differences between the features, it may not always be possible to automatically replace the function, and you will need to do it manually instead.

For more information on built-in functions, see the manual section on Functions.

## Classic scripts

Construct's JavaScript coding feature relies on JavaScript Modules, using `import` and `export` statements. However it originally used "classic" mode scripts, which don't support `import` or `export` statements. In 2020 Construct added support for modules, and in 2021 it removed support for the legacy "classic" mode scripts. To help you identify compatibility problems with old projects, Construct continues to warn you if you open a project that was still using "classic" mode scripts.

To update the project to use modules, refer to the tutorial Upgrading projects from classic scripts to modules. The next time you save the project the deprecation warning will no longer appear.

## Other deprecated features

Sometimes individual conditions, actions or expressions are marked deprecated, or effects are deprecated. Construct does not currently notify you about these, because they are generally minor and easy to continue to support in the long term. However you may notice these as some conditions, actions, expressions or effects that you can access from the user interface differ from the ones referenced in an old project. If this is annoying for any reason, you can delete the old usage and add it back with the new usage to update it, but usually this is not necessary to keep the project working.

# IMPORTING CONSTRUCT 2 PROJECTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/importing-c2-projects

Construct 3 is largely backwards-compatible with Construct 2. You can usually import Construct 2 projects in to Construct 3 and it will open them so you can continue working on them in Construct 3. One easy way to do this is to drag-and-drop a Construct 2 .capx file in to the Construct 3 window. However sometimes you will need to make changes to an imported Construct 2 project to continue developing it in Construct 3.

*Note that Construct 3 saves projects in a different format to Construct 2. While C3 can open C2 projects, C2 can't open C3 projects. Make sure you are ready to move entirely to Construct 3 before making significant changes to an imported Construct 2 project.*

This guide covers some points to be aware of when importing Construct 2 projects to Construct 3.

*Construct 2 was retired in July 2021 and is no longer officially supported. For more information see the blog post Sunsetting Construct 2. However you can still import your C2 projects in to C3.*

## Replaced features

Construct 3 originally inherited all of Construct 2's features. Over the years some key features have been replaced and the old feature retired. These include:

- The entire runtime was rewritten and upgraded to an enhanced, modernized and faster engine, referred to as the **C3 runtime**. This is highly compatible with the old runtime but there are some compatibility differences.
- The Function plugin has been replaced with a **built-in functions** feature
- Construct 3 now exports projects with a proper **"folder" file structure**, preserving subfolders and filename case, instead of the legacy "flat" mode which exported everything in the root folder with a lowercased name.

You may need to update an imported C2 project to take in to account these changes. For more information, see the deprecated features section. The rest of this guide also includes details about other compatibility differences.

# Compatibility differences

This section covers some changes made either with Construct 3 or its new runtime that may affect how imported Construct 2 projects work.

## Setting effect color parameters

In Construct 2, some effects like *Tint* and *Set color* specified a color parameter as three separate parameters for the red, green and blue components. In Construct 3, these parameters have been replaced by a single color parameter. Construct 3 will show a color picker instead of three number fields for red, green and blue.

Construct 3 should correctly import the right color value. However if you use *Set effect parameter* in your events to change the effect's color, this will need updating. Previously you may have used actions like:

- *Set effect "Tint" parameter 0 to redAmount*
- *Set effect "Tint" parameter 1 to greenAmount*
- *Set effect "Tint" parameter 2 to blueAmount*

In Construct 3 there is now only one parameter instead of three. So this action needs to be replaced by:

- *Set effect "Tint" parameter 0 to rgbEx(redAmount, greenAmount, blueAmount)*

## Physics behavior stepping mode

In Construct 2, the Physics behavior defaulted to *fixed* stepping mode. This meant simulations were deterministic, but also meant it assumed a fixed display rate of 60 Hz. This was a reasonable assumption in the early 2010s, but now devices with varying refresh rates are much more common, such as phones with 90 Hz displays, tablets with 120 Hz displays, and gaming monitors with 144Hz+ displays. On these devices the fixed framerate mode of the Physics behavior appears to run in fast-forward mode.

To avoid this, the Physics behavior in Construct 3 instead defaults to *framerate independent* mode. This ensures it works at the same speed on all displays, but means the simulation is no longer deterministic. If you want to restore a deterministic simulation you can use the *Set stepping mode* action to switch back to *Fixed* mode - but then note you will have the aforementioned problem of the gameplay running too fast on high refresh rate displays.

## Dynamically creating objects

In the C2 runtime, creating an object which is not placed on the layout (and so is not loaded in to memory) immediately loads images for the object, which can "jank" the game (cause a short pause while loading is done). In the C3 runtime, in this case the game continues to run and the images are loaded in parallel. This provides better loading performance and avoids janking the game. However the object can exist for a short period of time while the images are still being

loaded. During this time the object is not drawn, as if it were set to invisible. In some cases this can cause a noticable flicker, especially if the object is intended to cover up something else. The workaround is to place the object in the layout, and destroy it in *On start of layout* if it is not needed. This ensures Construct loads its images when loading the layout, avoiding any delay when creating it.

### System object changes

The following changes were made to the System object, and system expressions, in the C3 runtime:

- The *WindowWidth* and *WindowHeight* expressions have been removed. These were confusingly named, but if you need the same values, use the Platform Info object's *CanvasDeviceWidth* and *CanvasDeviceHeight* expressions. Usually it's better to use the new *ViewportWidth* and *ViewportHeight* system expressions instead, which return a size in layout pixels rather than device (physical display) pixels.
- The *Is on mobile* and *Is on platform* conditions have been moved to the Platform Info object.
- The *Renderer* and *RendererDetail* expressions have been moved to the Platform Info object.
- The *rgb* expression has been removed. Instead use the new *rgbEx* or *rgba* expressions. Note *rgb* used values in the 0-255 range, but *rgbEx* and *rgba* use values in the 0-100 range, and also support a wider range with better precision.

### User Media object changes

In the C3 runtime, the User Media object's speech synthesis, speech recognition, and canvas recording features have been moved to separate plugins (*Speech Synthesis*, *Speech Recognition* and *Video recorder*). Existing projects using these User Media features can be used in C3, but those features will no longer work until you replace them with the new plugins.

Also note that the *Snapshot* action no longer provides the snapshot immediately: you must use the *On snapshot ready* trigger to know when *SnapshotURL* is available.

### Browser object changes

The following changes have been made for the Browser object in the C3 runtime:

- The battery features have been removed due to lack of browser support.
- The network information and various display related expressions have been moved to the Platform Info object.
- *On suspended* and *On resumed* have been moved to the System object. The *Page is visible* condition has been replaced with the *Is suspended* system condition (which makes the inverse check).

### Third-party addons

If your project uses third-party plugins, behaviors or effects, these need to be available in Construct 3 as well before you can import a Construct 2 project using them. Note that Construct 2 addon files (.c2addon) cannot be used in Construct 3. The addon needs to be updated to work with Construct 3, and distributed as a Construct 3 addon file (.c3addon).

Note that third-party addon developers are independent of Scirra and we cannot provide support for their addons. If you need help with third-party addons you'll need to contact the addon developer.

# BEHAVIOR REFERENCE

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

This section provides a reference of all the official behaviors that come with Construct. Each has an overview of its use, a list of its properties, and a detailed list of the actions, conditions and expressions specific to that behavior. Behaviors add their actions, conditions and expressions to the object they are added to, appearing alongside the object's own features in the Add condition/action dialog and Expressions dictionary.

Behaviors can be added and removed from objects via the Properties Bar.

## Custom controls

By default, user-controlled movement behaviors use a standard set of controls (usually based around the arrow keys). To change these, disable the *Default controls* property, and set up some events using the *Simulate control* action. The following events show how to use the **W**, **A** and **D** keys to control the Platform movement.




*Note that the input events must be continually true, i.e. using Key is down and **not** On key pressed.*

The *Simulate control* action does not have to be used with keyboard input. For example, it can be used with touch input to create on-screen touch controls, or with Gamepad input. Alternatively, you can set up events to automatically control objects with the *Simulate control* action. This allows you to set up computer-controlled enemies or non-player characters (NPCs), sometimes loosely referred to as "AI".

# 8 DIRECTION BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/8-direction

The **8 Direction behavior** allows an object to be moved up, down, left, right and on diagonals, controlled by the arrow keys by default. It is often useful for controlling the player in a top-down view game. It can also be set to 4 directions or simple up/down or left/right movement which is useful for paddles or sliders. Click here to open an example of the 8-direction behavior.

The 8 Direction behavior is blocked by any objects with the Solid behavior.

To set up custom or automatic controls, see the behavior reference summary.

### Scripting

When using JavaScript or TypeScript coding, the features of this behavior can be accessed via the I8DirectionBehaviorInstance script interface.

## 8 Direction properties

**Max speed**

The maximum speed the object can travel at in any direction, in pixels per second.

**Acceleration**

The rate the movement accelerates at, in pixels per second per second. When reversing against the current movement, acceleration and deceleration both contribute to slowing down.

**Deceleration**

The rate the movement decelerates to rest when not being moved, in pixels per second per second. When reversing against the current movement, acceleration and deceleration both contribute to slowing down.

**Directions**

Set how many directions the movement can move in. By default it is *8 Directions*, allowing movement on diagonals. *4 directions* prevents movement on diagonals, and *Up & down* or *Left & right* only allows movement along a single axis.

**Set angle**

Whether or not the movement should also affect the objects angle. *360 degree (smooth)* will always set the object's angle to the current angle of motion. *45-degree intervals* will set the object's angle to 8 possible directions. *90-degree intervals* will set the object's angle to 4 possible directions. *No* means the behavior will not set the object's angle at all, which is useful if you want to control this yourself with events (e.g. to make the object point towards the mouse cursor).

**Allow sliding**

If disabled, the object will simply stop when it collides with a solid. If enabled, the object will be able to continue moving along angled solids when it collides with them, essentially 'slipping' or 'sliding' along them.

**Default controls**

If enabled, movement is controlled by the arrow keys on the keyboard. Disable to set up custom controls using the *Simulate control* action. For more information see the behavior reference summary.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## 8 Direction conditions

**Allows sliding**

Test if the behavior currently allows sliding along solids.

**Compare speed**

Compare the object's current speed in pixels per second.

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

**Is moving**

True if the object has a non-zero speed (is not stopped). Invert to test if the object is stopped.

## 8 Direction actions

**Reverse**

Invert the direction of motion. Useful as a simple way to bounce the object off an obstacle.

---

**Set acceleration**

**Set deceleration**

**Set allow sliding**

**Set max speed**

**Set default controls**

These set the corresponding properties, described under *8 Direction properties*.

---

**Set enabled**

Enable or disable the movement. If disabled, the movement no longer has any effect on the object.

---

**Set ignoring input**

Set whether input is being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

---

**Set speed**

Set the current speed the object is moving at, in pixels per second.

---

**Set vector X**

**Set vector Y**

Set the X and Y components of the movement, in pixels per second.

---

**Simulate control**

Simulate one of the movement controls being held down. Useful when *Default controls* is disabled. See the behavior reference summary for more information.

---

**Stop**

A shortcut for setting the speed to zero.

## 8 Direction expressions

---

**Acceleration**

**Deceleration**

**MaxSpeed**

Return the corresponding properties, described under *8 Direction properties*.

---

**MovingAngle**

Get the current angle of motion (which can be different to the object's angle), in degrees.

---

**Speed**

Get the current object's movement speed, in pixels per second.

---

**VectorX**

**VectorY**

Get the object's current speed on each axis, in pixels per second. For example, if the object is moving to the left at 100 pixels per second, *VectorX* is -100 and *VectorY* is 0.

# ANCHOR BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/anchor

The **Anchor behavior** is useful for automatically positioning objects relative to the viewport. This is useful for supporting multiple screen sizes.

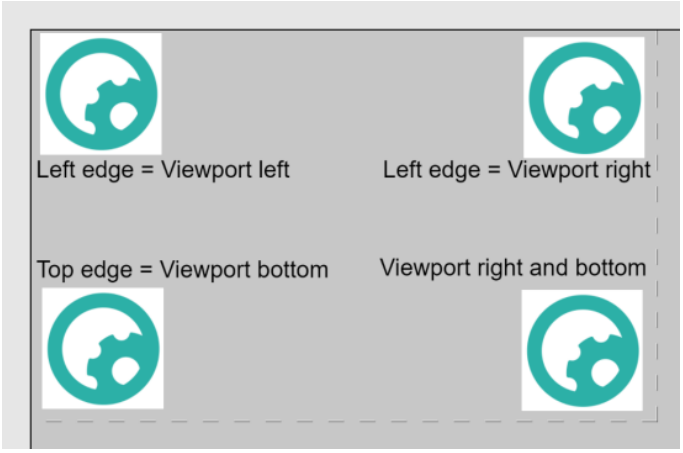
Objects using the Anchor behavior should also be placed on a layer with its parallax set to 0% x 0%. Otherwise as the game scrolls the objects may "lag" behind the screen.

### Scripting

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

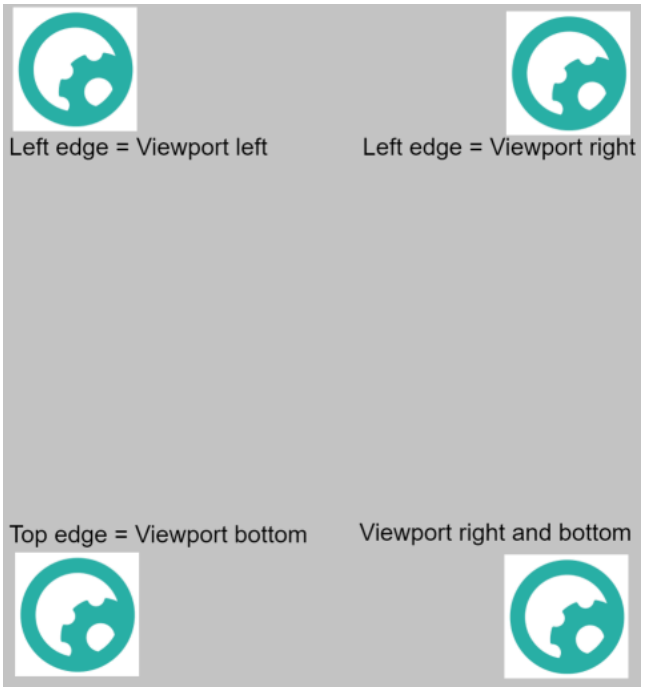
## Positioning objects relative to the viewport

In the top-left of the Layout View, a dotted outline represents the viewport. Anchored objects should be positioned inside the dotted viewport area, as shown below.




*Positioning anchored objects in the layout*

Now if the window is resized during preview, the objects maintain their relative positions, as shown below. Note this demo uses *Scale outer* fullscreen mode to allow the aspect ratio to change.




*Anchored objects in preview*

This is useful for interface elements like notifications and heads-up displays (HUDs).

## Relative positioning

The *Left edge* and *Top edge* position the object relative to the viewport edges, without changing the object size. For example, if the *Left edge* is set to *Viewport right*, the object will always stay the same distance from the right edge of the viewport.

The *Right edge* and *Bottom edge* resize the object relative to the viewport edges. For example, if you want a Tiled Background to stretch wider as the window widens, set *Right edge* to *Viewport right*.

## Anchor conditions

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

## Anchor actions

**Set enabled**

Set whether the behavior is enabled or disabled. If disabled, the behavior will not alter the size or position of the object.

## Anchor expressions

The Anchor behavior has no expressions.

# BOUND TO LAYOUT BEHAVIOR

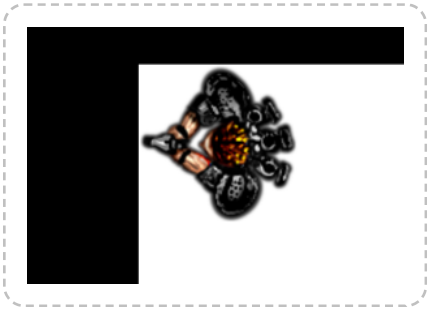
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/bound-to-layout

The **Bound to Layout behavior** simply prevents an object leaving the edge of the layout. It is most useful on objects which move around but should not leave the layout area.

It has no conditions, actions or expressions, just the following property:

**Bound by**

Set to *Edge* to prevent any part of the object leaving the layout. The object will stop at the position shown below:



Set to *Origin* to only prevent the object's origin leaving the layout. The result depends on where the origin is placed on the object, but with a centred origin the object will stop in a position similar to the image shown below:



# BULLET BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/bullet

The **Bullet behavior** simply moves an object forwards at an angle. However, it provides extra options like gravity and bouncing that allow it to also be used like a bouncing ball. Like the name suggests it is ideal for projectiles like bullets, but it is also useful for automatically controlling other types of objects like enemies which move forwards continuously.

For two examples of the Bullet behavior see the Bouncing balls and Bouncing bullets examples in the Example Browser.

### Scripting

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

## Stepping bullets

Normally the bullet behavior moves a fixed distance every tick. However if it is moving extremely fast, the distance it moves in one frame can be enough to jump over obstacles in their entirety. To avoid this problem you can enable the *Step* property. This will move the object in small increments, triggering *On step* each time. In *On step* you can add an *Is overlapping* condition to check if the bullet is overlapping an obstacle with improved accuracy.

In an *On step* event the *Stop stepping* action will prevent the bullet advancing any further. This leaves the bullet in the first position it collided with an object, which is a good place to create an effect like an explosion. You can also *Destroy* the object which will also automatically stop stepping.

For an example of how stepping works, see the Bullet stepping example in the Example Browser.

While the same technique can also be used to make instant hit bullets, this is usually better done using the raycasting feature of the Line of Sight behavior, as demonstrated in the Instant hit laser example.

## Bullet properties

**Speed**

The bullet's initial speed, in pixels per second.

**Acceleration**

The rate of acceleration for the bullet, in pixels per second per second. Zero will keep a constant speed, positive values accelerate, and negative values decelerate until a stop (the object will not go in to reverse).

**Gravity**

The force of gravity, which causes acceleration downwards, in pixels per second per second. Zero disables gravity which is useful for top-down games. Positive values cause a parabolic path as the bullet is bullet down by gravity.

**Bounce off solids**

If enabled, the bullet will bounce off any objects with the Solid behavior. It will also calculate the angle of reflection to bounce off realistically depending on the object's shape and angle.

**Set angle**

If disabled, the behavior will never change the object's angle. If enabled, the behavior always sets the object angle to the angle of motion, and if the object angle is changed, the angle of motion will be updated correspondingly.

**Step**

Enable stepping mode, which moves the object in small increments triggering *On step* to improve the accuracy of collisions. See *Stepping bullets* above.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Bullet conditions

**Compare speed**

Compare the current speed of the bullet, in pixels per second.

**Compare distance travelled**

Compare the total distance the bullet has moved since creation, in pixels. This does not take in to account altering the object position with other actions like *Set position*.

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

### On step

When *Step* is enabled, this triggers as the object moves in small increments. Add an *Is overlapping* condition to test for collisions with improved accuracy. See *Stepping mode* above for more information.

## Bullet actions

### Bounce off object

Make the bullet bounce off an object it has just touched. Since this requires the bullet have only just touched the object, it is generally only useful in an *On collision* event. It will also calculate the angle of reflection to bounce off realistically depending on the object's shape and angle. If the bullet is not currently overlapping the given object, or is stuck deep inside it, this action will have no effect.

### Set acceleration

Set the bullet acceleration in pixels per second per second.

### Set bounce off solids

Enable or disable the *Bounce off solids* property.

### Set angle of motion

Set the angle the bullet is currently moving at, in degrees. *Note:* when the speed is 0, the angle of motion is always 0 and cannot be changed, since there is no motion. Therefore setting the angle of motion then the speed does not work if the object is stopped. Instead, set the speed first and then the angle of motion.

### Set distance travelled

Simply changes the counter returned by the *DistanceTravelled* expression. The counter still increments according to the movement of the object.

### Set enabled

Enable or disable the movement. If disabled, the behavior will stop moving the bullet, but will remember the current speed, acceleration etc. if enabled again.

### Set gravity

Set the acceleration caused by gravity, in pixels per second per second.

### Set speed

Set the bullet's current movement speed, in pixels per second.

---

**Stop stepping**

When *Step* is enabled, prevent the object advancing any further. This is usually done when a collision is detected with *Is overlapping*. Note that destroying the object also automatically stops stepping. For more information, see *Stepping bullets* above.

## Bullet expressions

---

**Acceleration**

Get the bullet's current acceleration in pixels per second per second.

---

**AngleOfMotion**

Get the angle the bullet is currently moving at (which can be different to the object's angle), in degrees. Note when the object is stopped (with a speed of 0), the angle of motion is always 0.

---

**DistanceTravelled**

Return the total distance the bullet has moved since creation, in pixels. This does not take in to account altering the object position with other actions like *Set position*.

---

**Gravity**

Return the currently set gravity property, in pixels per second per second.

---

**Speed**

Get the bullet's current movement speed in pixels per second.

# CAR BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/car

The **Car behavior** allows an object to accelerated forwards and backwards and have steering. It also has a simple "drift" feature where the object can "skid" around corners (by pointing in a different direction to that it is moving in). For an example of the Car behavior see the *Driving* example in the Start Page.

By default the object is controlled by the arrow keys on the keyboard (Up to accelerate, down to brake, left and right to steer). To set up custom or automatic controls, see the behavior reference summary.

The Car behavior will bounce off any objects with the Solid behavior. The effect on the movement depends on the angle of impact - glancing collisions nudge the car off its current path, whereas head-on collisions stop it more or less dead. The amount of speed lost depends on the *Friction* property.

### Scripting

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

## Car properties

**Max speed**

The maximum speed, in pixels per second, the car can accelerate to.

**Acceleration**

The rate the car accelerates at, in pixels per second per second.

**Deceleration**

The rate the car brakes at, in pixels per second per second.

**Steer speed**

The rate the car rotates at when steering, in degrees per second.

**Drift recover**

The rate the car recovers from drifts, in degrees per second. In other words, this is the rate the angle of motion catches up with the object angle. The angle of motion can never be more

than 90 degrees off the object angle. If the drift recover is greater or equal to the *Steer speed*, no drifting ever occurs. The lower the drift recover, the more the car will drift on corners.

**Friction**

The amount of speed lost when colliding with a solid, from 0 (stop dead) to 1 (speed not affected at all). For example, to slow the speed down by half when colliding with a solid, set *Friction* to 0.5.

**Turn while stopped**

Set whether or not the object can rotate while not moving. When enabled, the rotate speed is always the same regardless of the speed. When disabled, the rotate speed adjusts with the movement speed resulting in a fixed turning circle, which also means the object cannot rotate while stopped.

**Set angle**

If enabled, the behavior will set the object's angle, otherwise the behavior never changes the object's angle.

**Default controls**

If enabled, the car movement is controlled by the arrow keys on the keyboard. Disable to set custom controls. For more information see the behavior reference summary.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Car conditions

**Compare speed**

Compare the current speed of the car, in pixels per second.

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

**Is moving**

True if the current speed is non-zero. Invert to test if the car is stopped.

## Car actions

**Set acceleration**

**Set deceleration**

**Set default controls**

**Set drift recover**

**Set friction**

**Set max speed**

**Set steer speed**

**Set turn while stopped**

Set the corresponding properties. See *Car properties* for more information.

---

**Set enabled**

Enable or disable the movement. If disabled, the movement no longer has any effect on the object.

---

**Set ignoring input**

Set whether input is being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

---

**Set speed**

Set the current speed the object is moving at, in pixels per second.

---

**Simulate control**

Simulate one of the movement controls being held down. Useful when disabling *Default controls*. See the behavior reference summary for more information.

---

**Stop**

A shortcut for setting the speed to zero.

## Car expressions

---

**Acceleration**

**Deceleration**

**DriftRecover**

**Friction**

**MaxSpeed**

**SteerSpeed**

Return the corresponding properties. See *Car properties* for more information.

---

**MovingAngle**

Get the current angle of motion (which can be different to the object's angle), in degrees.

---

**Speed**

Get the current object's movement speed, in pixels per second.

---

**VectorX**

**VectorY**

Get the object's current speed on each axis, in pixels per second. For example, if the object is moving to the left at 100 pixels per second, *VectorX* is -100 and *VectorY* is 0.

# CUSTOM MOVEMENT BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/custom-movement

The **Custom Movement behavior** does not directly implement any movement for an object. Instead, it provides features that make it easier to implement your own "custom" (event-based) movement.

The way different movements are made is out of the scope of this manual section. Instead, it will outline the basics of the Custom Movement behavior and what its features do. For an example of Asteroids style movement using the Custom Movement behavior, open the *Custom movement ('Asteroids' style)* example.

For many games, the built-in behaviors like Platform and 8 Direction are perfectly sufficient. Recreating existing behaviors with the Custom Movement should be avoided, since movements are difficult and time consuming to implement correctly. The built-in behaviours have been thoroughly tested, probably have more features than you imagine (like slope detection in Platform), and are much quicker and easier to use than making your own movement.

## Overview of custom movements

Most movements in Construct work by manipulating two values: the speed on the X axis (*dx*) and the speed on the Y axis (*dy*). These are also known as *VectorX* and *VectorY* in some other behaviors. For example, if an object is moving left at 100 pixels per second, *dx* is -100 and *dy* is 0. The object can then be accelerated to the right by adding to *dx*. This is also how most of the other movement behaviors that come with Construct work internally (like Platform and 8 Direction).

The Custom Movement behavior stores the *dx* and *dy* values for you, and provides features that help easily implement the math and algorithms necessary to make a movement.

Every tick the Custom Movement adjusts the object's position according to the *dx* and *dy* values. This is called a **step**. The Custom Movement can also use multiple steps per tick, which can help detect collisions more accurately if the object is moving very quickly. Each step will trigger *On step*, *On horizontal step* or *On vertical step* depending on the *Stepping mode* property.

## Custom Movement properties

**Stepping mode**

How to step the movement each tick. The number of steps taken (if not *None*) depends on the *Pixels per step* property. The different modes are:

- **None** simply steps the object once per tick according to its velocity.

- **Linear** will step the object in a straight line towards its destination position, triggering *On step*.

- **Horizontal then vertical** will step the object to its destination first on the X axis (triggering *On horizontal step*), then on the Y axis (triggering *On vertical step*).

- **Vertical then horizontal** will step the object to its destination first on the Y axis (triggering *On vertical step*), then on the X axis (triggering *On horizontal step*).

---

**Pixels per step**

If *Stepping mode* is not *None*, this is the distance in pixels of each step towards the destination position each tick. The default is 5, which means if the object is moving 20 pixels in a tick, it will move in four five-pixel steps.

---

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Custom Movement conditions

---

**Compare speed**

Compare the current speed of the movement, in pixels per second. *Horizontal* and *Vertical* compares to the *dx* and *dy* speeds respectively, and *Overall* compares to the magnitude of the vector (*dx*, *dy*) (the overall movement speed).

---

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

---

**Is moving**

True if either *dx* or *dy* are not zero. Invert to test if stopped.

---

**On horizontal step**

**On vertical step**

Triggered for each step along an axis when *Stepping mode* is either *Horizontal then vertical* or *Vertical then horizontal*. This can be used to accurately detect collisions with *Is overlapping*.

---

**On step**

Triggered for each step when *Stepping mode* is *Linear*. This can be used to accurately detect collisions with *Is overlapping*.

## Custom Movement actions

**Set enabled**

Enable or disable the behavior. If disabled, the behavior will not modify the object's position.

**Rotate clockwise**

**Rotate counter-clockwise**

**Set angle of motion**

Adjust the angle of motion. This will calculate new values for *dx* and *dy* reflecting a new angle of motion with the same overall speed. **Note:** if the overall speed is 0, then setting the angle of motion has no effect, because there is no motion. A common mistake is to set the angle of motion then the speed, and find that the angle is not used. Instead simply set the speed first then the angle of motion and it will work as expected.

**Accelerate**

Accelerate either the overall movement, or movement on a specific axis.

**Accelerate toward angle**

**Accelerate toward position**

Accelerate the movement towards an angle or position.

**Push out solid**

Only valid when the behavior is currently overlapping an object with the solid behavior. Automatically move the object until it is no longer overlapping the solid. This has no effect if the object is not currently overlapping a solid. The following techniques can be used:

- **Opposite angle** reverses (or 'backtracks') the object from its current angle of motion until it is no longer overlapping.
- **Nearest** moves the object in an eight-direction spiral out one pixel at a time until it is no longer overlapping. The aim is for the object to end up in the nearest free space, but since only eight directions are used it will be an approximation.
- **Up**, **down**, **left** and **right** moves the object along a specific axis until it is no longer overlapping.

### Push out solid at angle

Only valid when the behavior is currently overlapping an object with the solid behavior. Move the object from its current position at a given angle until it is no longer overlapping the solid. This has no effect if the object is not currently overlapping a solid.

### Reverse

Inverts the movement by flipping the signs of *dx* and *dy*.

### Set speed

Set the current speed in pixels per second either for the horizontal or vertical axes, or the overall movement speed. Setting horizontal or vertical speeds assigns *dx* and *dy* directly. Setting the overall speed calculates new values for *dx* and *dy* such that they reflect the new overall speed while keeping the same angle of motion.

### Stop

A shortcut for setting both *dx* and *dy* to 0, stopping the movement.

### Stop stepping

Only valid in *On step*, *On horizontal step* and *On vertical step*. Stop the current stepping for this tick. The object can either go back to its old position (where it was at the start of the tick) or stay at its current position (possibly half way between its start and end positions). Note that in *Horizontal then vertical* or *Vertical then horizontal* modes, only the current axis is stopped. The next axis will still continue stepping, unless you also use *Stop stepping* for that axis as well.

## Custom Movement expressions

### dx

### dy

Return the movement's *dx* and *dy* values, which are the speed in pixels per second on each axis.

### MovingAngle

Return the current angle of motion, in degrees, calculated as the angle of the vector (*dx*, *dy*).

### Speed

Return the current overall speed in pixels per second, calculated as the magnitude of the vector (*dx*, *dy*).

# DESTROY OUTSIDE LAYOUT BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/destroy-outside-layout

The **Destroy Outside Layout behavior** simply automatically destroys an object if it leaves the layout area. It only destroys the object if it is entirely outside the layout (i.e. no part of its bounding box is inside the layout). The Destroy Outside Layout behavior has no properties, conditions, actions or expressions.

This behavior is often useful to prevent bullets and other projectiles flying off the layout forever. For example, without this behavior, if the user shoots some enemies and misses, the bullets could carry on flying off outside the layout forever. As the game goes on, more and more bullets build up over time, all moving further away forever. This gradually uses up more and more memory and CPU power causing the game to slow down. This can easily be avoided by adding this behavior to the bullets, so they are automatically removed when they leave the layout area.

# DRAG & DROP BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/drag-drop

The **Drag & Drop behavior** enables objects to be dragged and dropped either by mouse or touch. On multi-touch devices, multiple objects can be dragged and dropped at once.

An object starts dragging when a mouse click or touch falls inside the object's collision polygon. It is released when the mouse button is released or the touch ends.

### Scripting

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

## Drag & Drop properties

**Axes**

Optionally axis-limit the movement, for example only allowing the object to be dragged horizontally or vertically. The default is *Both*, allowing the object to be dragged anywhere.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Drag & Drop conditions

**Is dragging**

True if the object is currently being dragged by mouse or touch.

**Is enabled**

True if the behavior is currently enabled; false if disabled by the *Set enabled* action.

**On drag start**

Triggered when the object is clicked or touched inside its collision polygon.

**On drop**

Triggered when the object is being dragged, and the mouse button is released or the touch ends.

# Drag & Drop actions

**Set enabled**

Enable or disable the Drag & Drop behavior. If disabled, clicking or touching the object has no effect and the object cannot be dragged. It also becomes 'transparent' to clicks and touches, meaning other objects underneath it can still be dragged and dropped. If the object is being dragged when it is disabled, the drag is cancelled, but *On drop* will not trigger.

**Set axes**

Set the *Axes* property, allowing changing which axes movement is limited to.

**Drop**

Stop dragging the object if it is currently being dragged. This also triggers *On drop*.

# Drag & Drop expressions

The Drag & Drop behavior has no expressions.

# FADE BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/fade

The **Fade behavior** fades objects in and out by changing the object's opacity over time. By default, it makes an object fade out over 1 second then destroys it.

> *The Fade behavior is made redundant by the Tween behavior. It's recommended to use opacity tweens instead of the Fade behavior. See also Superseded features.*

Fades run in the following order. If any of the times are 0, the step is skipped.

1. The object fades in from invisible to its set opacity, over the *Fade in time*.
2. The object remains at its current opacity for the *Wait time*.
3. The object fades out to invisible, over the *Fade out time*.
4. If the *Destroy* property is enabled, the object is then destroyed.

For example, with each time set to 1 second, the object will fade in from invisible for 1 second, wait for 1 second, then fade out to invisible for 1 second.

It is recommended to leave *Destroy* enabled. If disabled, the object still exists after fading out, but is invisible. If many objects are using the Fade behavior, this can build up many invisible objects over time, which gradually use more memory and CPU causing the game to slow down.

### Scripting

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

## Fade properties

**Fade in time**

Time, in seconds, to fade in from invisible. If 0, the fade in is skipped.

**Wait time**

Time, in seconds, to wait between fade in and fade out. If 0, the step is skipped.

**Fade out time**

Time, in seconds, to fade out to invisible. If 0, the fade out is skipped.

**Destroy**

If enabled, the object is automatically destroyed after the fade out finishes. If disabled, the object is never destroyed by the behavior. Be sure to destroy objects yourself as necessary, as a build-up of invisible faded-out objects can cause the game to slow down.

**Enabled**

If enabled, the object will begin fading as soon as it is created. Otherwise the fade will not run until you use the *Start* action.

**Preview Paid plans only**

Enable to run a preview of the fade effect directly in the Layout View.

## Fade conditions

**On fade-in finished**

**On wait finished**

**On fade-out finished**

Triggered when each stage of the fade finishes.

## Fade actions

**Restart fade**

Run the entire fade from the beginning again.

**Set fade-in time**

**Set fade-out time**

**Set wait time**

Set the corresponding properties described under *Fade properties*.

**Start fade**

If the *Enabled* property is disabled, this will begin the fade.

## Fade expressions

**FadeInTime**

**FadeOutTime**

**WaitTime**

Return the corresponding properties described under *Fade properties*.

# FLASH BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/flash

The **Flash behavior** makes an object blink by toggling its visibility on and off for a period of time.

Simply adding the Flash behavior to an object does not do anything. You must use the *Flash* action to make the object flash.

### Scripting

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

## Flash conditions

**Is flashing**

True if the object is currently within a flash duration.

**On flash ended**

Triggered when the end of the flash duration is reached, and the object has returned to visible.

## Flash actions

**Flash**

Make the object flash by toggling its visibility on and off. The *On* time is the duration in seconds the object remains visible. The *Off* time is the duration in seconds the object remains invisible. The object will alternate between these two states for the given duration in seconds. The object is always set back to visible after the flash duration finishes.

**Stop flashing**

If the object is currently flashing, this stops the flashing and sets the object back to visible. If the object is not currently flashing this action has no effect.

# FOLLOW BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/follow

The **Follow behavior** allows an object to follow another object on a time or distance delay. More generally, it records a short history of an object's changes, allowing it to also perform tasks like record and replay, or rewinding time.

Only adding the Follow behavior to an object does not do anything. You must use an action like *Follow object* before it starts changing the object the behavior has been added to.

You can find a number of examples using the Follow behavior in Construct's Example Browser, such as the Follow behavior example and Follow record/replay.

### Scripting

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

## Follow properties

**Mode**

The following mode to use. *Time* mode follows on a time delay. *Distance* mode follows with a distance delay. In *Distance* mode the X and Y properties are always followed, as it is necessary for determining the distance.

**Delay**

The delay on which to follow the tracked object. In *Time* mode this is the delay in seconds to follow. In *Distance* mode this is the distance in pixels at which to follow. The delay cannot exceed the max delay.

**Max delay**

Determines the amount of data remembered in the same units as *Delay*. For example in time mode, if the delay is 1 second but the max delay is 3 seconds, then the behavior is remembering 3 seconds of history but following at a 1 second delay. The delay can be increased up to but not past the max delay. Usually the max delay can be the same as the delay, as additional data does not need to be remembered, but having a higher max delay can be useful if the delay may be increased. Note that the higher the max delay the more data is remembered and so the more memory will be used, so it's best to use the shortest max delay possible.

**History rate**

**History rate**

The rate in entries per second at which data about the object being followed is saved. For example if the history rate is 10, then the state of the followed object is saved every 100ms. When following, values in between entries are interpolated. Higher rates use more memory and have a higher performance overhead, but have a smoother movement; for efficiency it is best to choose the lowest rate which produces acceptably smooth movement. The default of 30 usually produces good results.

---

**Follow X**

**Follow Y**

**Follow Z elevation**

**Follow width**

**Follow height**

**Follow angle**

**Follow opacity**

**Follow visiblity**

**Follow destroyed**

Choose the built-in properties to follow. For example if following X and Y is enabled, but not width and height, then if the followed object both moves and changes size, the following object will only move but not change size. Following more properties uses more memory. Following the destroyed state means that if the followed object is destroyed, then the following object will also be destroyed at the same point.

---

**Enabled**

Whether the behavior is initially enabled.

## Follow conditions

---

**Has follow data**

True if the behavior has enough data to be able to start following on a delay. For example if the behavior starts following an object on a 5 second time delay, then for the first 5 seconds there is no follow data and so the object will not be updated, and *Has follow data* will be false. Once 5 seconds has elapsed it then starts updating and *Has follow data* will be true. Note that if following starts with *From current position* enabled, then that counts as having follow data immediately.

---

**Is following object**

True if any object has been set to be followed. If false then the behavior is not recording any information.

**Is paused**

True after using the *Set paused* action to pause following.

**Is following custom property**

Test if a particular custom property, specified by a string, is currently enabled for following.

**Compare delay**

**Compare max delay**

**Compare history rate**

**Compare mode**

**Is enabled**

**Is following property**

Test the current values of the behavior properties. See *Follow properties* for more details.

## Follow actions

**Follow object**

Begin following the specified object. This starts recording the changes over time of the specified object, and after the delay period has passed, it will then start following the changes for the enabled properties. Until the delay has passed, *Has follow data* will be false as there is not yet any data to follow. Alternatively the *From current position* setting can be enabled, which allows immediate following. When enabled this creates an initial history entry based on the following object's current state in the past at the delay time. Therefore *Has follow data* is immediately true and the object is able to immediately start updating. This has the effect of interpolating from the following object's starting position to the followed object's starting position over the delay time.

**Follow self**

Begin following the object the behavior belongs to. This records the changes over time of the current object. In this mode, the behavior does not update the object; it merely records the history. Note that as with following a different object, data is only retained up to the max delay. The history can then be saved to JSON and later replayed, or it can stop following and then set the delay to move to a previous position.

**Stop following**

Stops recording the history of a followed object. Any recorded history of the object that was followed is still preserved, and it will still continue following changes up until the time that this action was used.

### Clear history

Erases any recorded history about the object being followed. This will cause *Has follow data* to become false and stop updating the object until enough data has been collected again. This is useful for resetting the behavior.

### Set paused

Set whether following is paused. While paused, no further history is recorded, but it also stops advancing the follow time. Upon resuming, the behavior will restart recording the history of the followed object. If the followed object has substantially changed while following was paused, then it will skip to the new position as it follows the history, as no changes in between will have been saved.

### Rewind history

Rewinds the follow time, deletes history entries past that time, and then continues recording history. Note that it is not possible to rewind further than the max delay, as data beyond that time is erased. This action allows for implementing a 'rewind time' feature where an object can go backwards in time and then continue from a different location, while preserving the history prior to the time it continued from.

### Load history JSON

Load the recorded history of the object being followed from a string of data in JSON format previously saved by the *HistoryAsJSON* expression. This also sets the delay to the time of the oldest history entry loaded, so it then immediately follows the amount of data originally saved. This allows for creating a record/replay feature.

### Start following custom property

### Stop following custom property

Start or stop following a custom property. This allows the Follow behavior to track a custom value other than one of the built-in properties such as the X and Y position. Multiple custom properties can be followed, each identified by a case-insensitive string. The interpolation mode of the custom property determines how values in between history entries are determined. *Step* does not interpolate and just uses the previous history entry. *Linear* uses linear interpolation, suitable for linear values like position and size. *Angular* uses angular interpolation, suitable for rotational values like angles. Note that if a custom property value is a string, then only *Step* mode is supported. The value to be recorded must be set with *Set custom property value*, and then the value to be followed can be retrieved with the *DelayedCustomPropertyValue* expression.

### Set custom property value

Set the current value of a custom property that is being followed. The custom property is identified by a case-insensitive string. The value can be either a string or a number, but if it is a string then it will only use *Step* interpolation mode. This action should be used every tick while following a custom property, so that the latest value is available when the behavior decides to add a history entry.

---

**Set delay**

**Set enabled**

**Set following property**

**Set history rate**

**Set max delay**

**Set mode**

Set the corresponding behavior properties. See *Follow properties* for more details.

---

**Set property interpolation**

Change the interpolation mode of one of the built-in properties. Generally this is used to change one of the built-in properties from smooth interpolation to step interpolation. For example mirroring an object with the Platform behavior should always update the width instantly, and not interpolate any in-between values.

## Follow expressions

---

**FollowUID**

The UID of the current object being followed, or -1 if no object to follow has been set.

---

**HistoryAsJSON(MaxDelay)**

Save the current recorded history of the followed object to a string in JSON format. This can then be loaded again later using the *Load history JSON* action. The *MaxDelay* parameter can be used to save only a portion of the most recent history, rather than all the recorded history, suitable for a record/replay feature when the recording duration is less than the behavior's max delay. If *MaxDelay* is 0, then it saves all the history.

---

**DelayedCustomPropertyValue(CustomProperty)**

Retrieve the current value to be followed for a custom property, specified by a case-insensitive string.

---

**Delay**

**MaxDelay**

**HistoryRate**

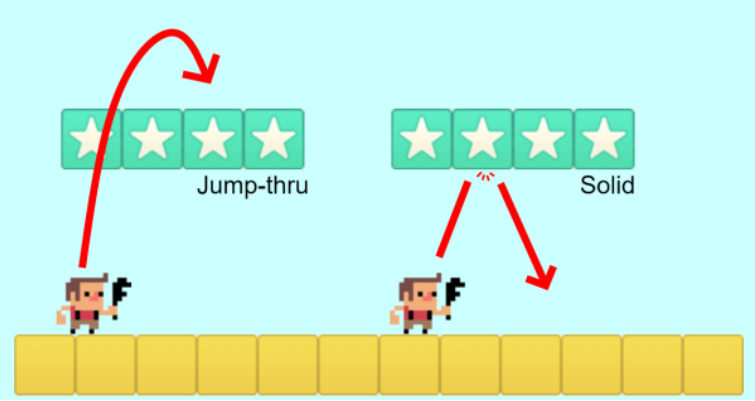
Return the corresponding behavior properties. For more information, see *Follow properties*.

# JUMP-THRU BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/jump-thru

The **Jump-thru behavior** allows the Platform behavior to stand on the object, and jump on to it from underneath. This differs from the Solid behavior, which the Platform behavior can stand on, but not jump on to from underneath. The image below illustrates the difference.

Note the Jump-thru behavior **does not support slopes**. Any slopes in your game should use the Solid behavior instead.




*Jump-thru vs. Solid behaviors*

### Scripting

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

## Jump-thru properties

**Enabled**

Set whether the behavior is initially enabled or disabled. If disabled, the object no longer acts as if it is a Jump-thru, and the Platform behavior will always fall through it.

## Jump-thru conditions

**Is enabled**

True if the behavior is currently enabled. This can be changed by the *Enabled* property or the *Set enabled* action.

## Jump-thru actions

**Set enabled**

Enable or disable the Jump-thru behavior for this object.

# LINE-OF-SIGHT BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/line-of-sight

The **Line-of-sight (LOS) behavior** allows the ability to check if two objects can "see" each other. More precisely, it will check if there are any obstacles blocking a line between the two objects. Click here to open an example of Line-of-sight.

Line-of-sight can also perform *Raycasting*. Normal line-of-sight checks there are no obstacles in a straight line between two objects. With raycasting, if there is an obstacle in the way, you can find the exact position of the obstacle in the way, as well as the surface normal and angle of reflection. The Instant hit laser example provides a demonstration of how to use raycasting.

### Scripting

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

## Line-of-sight properties

**Obstacles**

Whether to use *Solids* as blocking line-of-sight, or *Custom*, where the objects blocking line-of-sight must be added using the *Add obstacle* action.

**Range**

The maximum distance in pixels that line-of-sight can reach. If an object is further away than this distance, the object will never have line-of-sight to it, even if the intervening space is clear.

**Cone of view**

The angle of the cone of view in which the object can have line-of-sight to other objects, relative to the current angle of the object. For example if this is 180, then the object can have line-of-sight to any objects anywhere in front of it, but never behind it. If 360, the object can have line-of-sight to objects at any angle.

**Use collision cells**

Whether to use the collision cells optimisation when testing line of sight. Usually this is faster, but in some cases over extremely long distances it can be slower.

## Line-of-sight conditions

### Has LOS to object

Check if the object currently has line-of-sight to another object. For the condition to be true, the object must be within range, within the cone of view, and with no obstacles in the way of a straight line between the two objects. This condition also picks the instances of the chosen object that are in the line of sight. By default this checks if there is line-of-sight to the object origin, but you can optionally specify an image point instead.

### Has LOS to position

Check if the object currently has line-of-sight to a position in the layout. For the condition to be true, the object must be within range, within the cone of view, and with no obstacles in the way of a straight line between the two objects.

### Has LOS between positions

Check if there is line-of-sight between any two positions in the layout, instead of using the object's own position.

### Ray intersected

Use after a *Cast ray* action to determine if the ray intersected with any obstacle.

## Line-of-sight actions

### Add obstacle

If the *Obstacles* property is *Custom*, adds an object type to count as an obstruction to line-of-sight.

### Clear obstacles

If the *Obstacles* property is *Custom*, clears any object types added as obstacles with the *Add obstacle* action.

### Set cone of view

### Set range

Sets the corresponding behavior properties. For more information, see *Line-of-sight properties*.

### Cast ray

Check for obstacle intersection between any two positions in the layout. This sets the *Ray intersected* condition true or false depending on if the ray intersected an obstacle; typically this can be checked in a sub-event after the action. If the ray did intersect an obstacle, the hit, normal and reflection expressions are set according to the obstacle that was hit.

*This action ignores the range and cone of view properties, to allow raycasting anywhere in the layout.*

## Line-of-sight expressions

**ConeOfView**

**Range**

Retrieve the corresponding behavior properties. For more information, see *Line-of-sight properties*.

**HitX**

**HitY**

If *Ray intersected* is true, the position of the first obstacle the ray intersected, in layout co-ordinates.

**HitDistance**

If *Ray intersected* is true, the distance between the ray start point and the hit position.

**HitUID**

If *Ray intersected* is true, the UID of the instance that was the first obstacle the ray intersected.

**NormalAngle**

If *Ray intersected* is true, the angle of the surface normal at the point of intersection, in degrees.

**NormalX(length)**

**NormalY(length)**

If *Ray intersected* is true, return a position at a given distance along the surface normal vector.

**ReflectionAngle**

If *Ray intersected* is true, the angle of the reflection at the point of intersection, in degrees.

**ReflectionX(length)**

**ReflectionY(length)**

**Reflection (length)**

If *Ray intersected* is true, return a position at a given distance along the reflection vector.

# MOVE TO

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/move

The **Move To behavior** moves an object to a position at a maximum speed with acceleration and deceleration. It also supports rotating the object so it turns towards its target, and can add multiple waypoints to move to in sequence.

Click here to open an example of the Move To behavior.

> *'Move to' moves objects at a fixed speed, which means the time it takes to arrive at the target can vary. If you need to move objects in a fixed time period, use the Tween behavior instead.*

### Scripting

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

## Move To properties

**Max speed**

The maximum speed the object will move at, in pixels per second.

**Acceleration**

The rate at which the object will accelerate to the maximum speed, in pixels per second per second. Use 0 to disable acceleration, meaning the object will immediately start moving at the maximum speed.

**Deceleration**

The rate at which the object will decelerate towards its target, in pixels per second per second. Use 0 to disable deceleration, meaning the object will come to an abrupt stop without slowing down. The object will only slow down towards its last position; if there are further waypoints to move to, it will continue accelerating up to the maximum speed. Note that the deceleration also imposes a stopping distance during which the object must be slowing down, which in some cases may prevent the object reaching its maximum speed.

**Rotate speed**

The rate the object will turn towards its target position, in degrees per second. Use 0 to disable turning, meaning the object will always move directly towards the target. Note a slow rotate speed results in a large turning circle, which can have awkward results if the object has to navigate through several close waypoints. For this reason it's recommended to use a high rotate speed.

---

**Set angle**

When enabled, the object angle will be set to the angle it is moving at. If disabled the movement will not affect the object angle.

---

**Stop on solids**

If enabled, movement will stop if the object collides with something with the Solid behavior, and trigger *On hit solid*. If disabled solids are ignored.

---

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Move To conditions

---

**Compare speed**

Compare the object's current speed in pixels per second.

---

**Is enabled**

True if the behavior is currently enabled.

---

**Is moving**

True if the object has a target position it is moving towards.

---

**On arrived**

Triggered whenever the target position is reached. This is triggered for every waypoint if there are multiple waypoints to move to.

> *In this trigger, Is moving is true if there are further waypoints to move to, and false if it arrived at the last waypoint.*

---

**On hit solid**

Triggered if *Stop on solids* is enabled and the object collided with a solid.

## Move To actions

**Move to position**

**Move to object**

Start moving towards a position, given either by layout co-ordinates or an object (or optionally an image point on the object). If *Mode* is *Direct*, any existing waypoints are removed, so the object will immediately move towards the given position. If *Mode* is *Add waypoint*, it will instead add a new waypoint to move to after all existing waypoints have been reached.

**Move along Pathfinding path**

This is an alternative to the Pathfinding behavior's *Move along path* action. It only works with a Pathfinding behavior on the same object, and like the Pathfinding action can only be used after *On path found* triggers. The 'Move To' behavior uses a different algorithm for moving along waypoints, and this action lets you use its approach instead of the built-in Pathfinding movement.

**Move along timeline**

**Move along timeline (by name)**

Add all the points from the X and Y property tracks in a timeline track. The timing is ignored; the positions are treated only as a sequence of waypoints to move along. This is useful for using a timeline to visually design a path to follow in the Layout View. See the Move along path and Move along curved path examples for a demonstration.

**Set angle of motion**

Set the angle the object is currently moving at, in degrees.

**Stop**

Stop any current movement. This also removes all waypoints.

**Set speed**

Set the current movement speed in pixels per second. Note this cannot exceed the maximum speed, nor can it exceed the current speed while within the stopping distance, since increasing the speed while decelerating would cause the object to miss its target.

**Set acceleration**

**Set deceleration**

**Set enabled**

**Set max speed**

**Set rotate speed**

**Set stop on solids**

Set the corresponding properties. For more information see *Move to properties* above.

## Move To expressions

---

**MovingAngle**

The current angle in degrees the object is moving at.

---

**Speed**

The current speed the object is moving at, in pixels per second.

---

**TargetX**

**TargetY**

The current position in layout co-ordinates that the object is moving towards. When multiple waypoints are used, this is the current waypoint.

---

**WaypointCount**

**WaypointXAt(index)**

**WaypointYAt(index)**

Use these expressions to access the full list of waypoints added, given in layout co-ordinates.

---

**Acceleration**

**Deceleration**

**MaxSpeed**

**RotateSpeed**

Return the corresponding properties. For more information see *Move to properties* above.

# NO SAVE BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/no-save

The **No Save behavior** simply causes the object to be omitted from save states when using the *Save* and *Load* system actions.

Normally all objects are saved and loaded with these actions. Adding the No Save behavior will skip saving any data for the object when saving, and will not affect the object when loading. After a load, all the same objects that were there before the load are still present, and with the same properties.

It is a good idea to add the No Save behavior to objects which don't need to be saved, like scenery and background objects. It can also be used on automatically updated objects, like interface elements and text objects which update their text every tick. This will help make the saves smaller in size, and also complete saving and loading quicker.

For more information see the tutorial How to make savegames.

# ORBIT BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/orbit

The **Orbit behavior** moves an object in a circle or ellipse around a point. The object's initial position is used as the point to orbit around.

### Scripting

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

## Orbit properties

**Speed**

The speed to orbit at, in degrees per second. Positive is clockwise and negative is anticlockwise.

**Acceleration**

The rate of change to the orbit speed, in degrees per second per second. Positive will accelerate in a clockwise direction and negative will accelerate in an anticlockwise direction.

**Primary radius**

The distance of the orbit from its center point, in pixels. For a circular orbit, ensure the primary and secondary radii are the same. For elliptical orbits, the primary radius is the one in the direction of the offset angle.

**Secondary radius**

The perpendicular distance of the orbit from its center point, in pixels. For a circular orbit, ensure the primary and secondary radii are the same. For elliptical orbits, the secondary radius is the one perpendicular to the offset angle.

**Offset angle**

For elliptical orbits, the rotation of the ellipse in degrees. For circular orbits, this does not affect the orbit path (since rotating a circle has no effect), but it changes the initial angle the orbit starts from.

**Match rotation**

If enabled, sets the object's angle to match the direction of travel in the orbit. If disabled the behavior only changes the object's position without affecting the angle.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

**Preview Paid plans only**

Enable to run a preview of the behavior directly in the Layout View.

## Orbit conditions

**Is enabled**

Test if the behavior is currently enabled.

## Orbit actions

**Pin**

**Unpin**

Set another object as the location to orbit around, following the object if it moves. The *Unpin* action will stop following the object.

**Set acceleration**

**Set enabled**

**Set match rotation**

**Set offset angle**

**Set radius**

**Set speed**

Set the corresponding behavior properties. See *Orbit properties* above.

**Set rotation**

Set the current orbit position by its angle from the center point in degrees.

**Set target**

Set the center point of the orbit in layout co-ordinates.

**Reset total rotation**

Sets the counters for the TotalRotation and AbsoluteTotalRotation to 0

## Orbit expressions

**Acceleration**

**OffsetAngle**

**PrimaryRadius**

**SecondaryRadius**

**Speed**

Return the corresponding behavior properties. See *Orbit properties* above.

**DistanceToTarget**

Return the distance from the object to the center point of the orbit, in pixels.

**Rotation**

Return the current position of the orbit as its rotation relative to the center point in degrees.

**TargetX**

**TargetY**

Return the current center point of the orbit in layout co-ordinates.

**TotalRotation**

Return the total rotation of the instance in degrees. This value does not wrap at 360 degrees. If the instance is rotating counter-clockwise then the value will decrease over time. This counter can be cleared using the *Reset total rotation* action.

**TotalAbsoluteRotation**

Return the total rotation of the instance in degrees, ignoring rotation direction. This expression is very similar to the *TotalRotation* expression, but rotation deltas are converted to absolute values. This means the counter will always increase even if the instance is rotating counter-clockwise. This counter can be cleared using the *Reset total rotation* action.

# PATHFINDING BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/pathfinding

The **Pathfinding behavior** uses the A* pathfinding algorithm to efficiently find a short path around obstacles. It can either report the path as a list of nodes through expressions, or automatically move the object along the determined path.

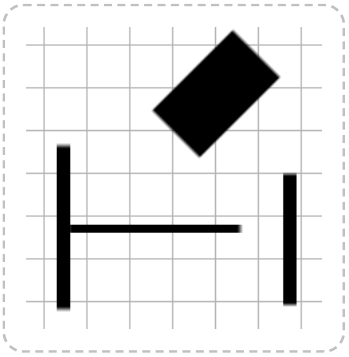
Click here to open an example of the Pathfinding behavior to see how it can be used. Search for *Pathfinding* in the Start Page to find an additional example.

### Scripting

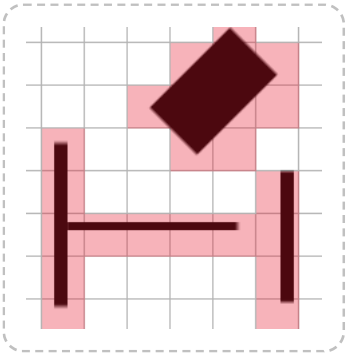
When using JavaScript or TypeScript coding, the features of this behavior can be accessed via the IPathfindingBehaviorInstance script interface.

## The pathfinding grid

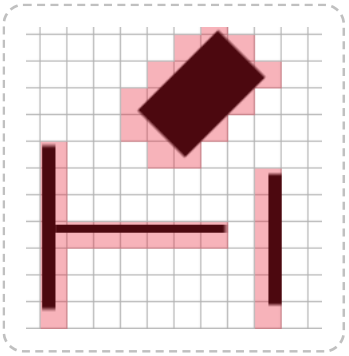
The pathfinding behavior works based on dividing the layout in to a grid. Since pixel-perfect pathfinding can be extremely slow to process, dividing the layout in to cells makes the pathfinding enormously more efficient. The cell size can be set in the behavior property, and the larger it is the more efficient pathfinding is. However setting a large cell size can cause problems: a cell can only be entirely obstacle or entirely free, and using large cells can close up small gaps. For example take the following arrangement of obstacles using a cell size of 32:



It appears that objects should be able to freely move around in between these objects. However if the cells that the pathfinding behavior marks as obstacle are highlighted in red, we see this:

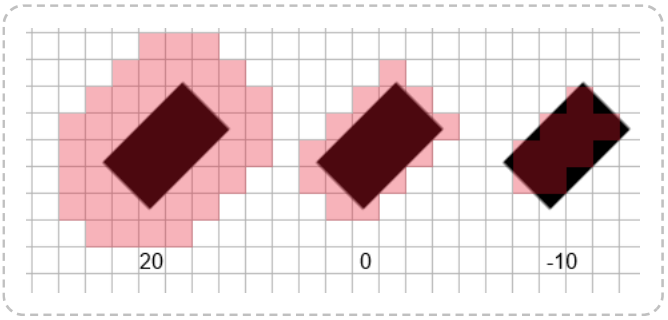


Some of the gaps have been closed off due to the cell size being relatively large compared to the size of the gap. This will make the pathfinding behavior route paths entirely around the obstacles, and never through them. We can help fix this by reducing the cell size to 20:



Now we can see that the Pathfinding behavior will be able to find routes between these obstacles. However, the smaller cell size will make the pathfinding more CPU intensive. Generally, try to use the largest cell size that does not cause problems navigating around obstacles.

The **Cell border** property can adjust how cells are marked as obstacle. If the border is larger than 0, then cells close to obstacles but not actually touching may also be marked as obstacles, effectively giving an extra "obstacle border". If the border is negative, cells only just touching an obstacle may *not* be marked as an obstacle, effectively shrinking the obstacle area inwards. The image below demonstrates the effect of different cell border values when using a cell size of 20.




For best efficiency, **use the same cell size and border** for all objects using the Pathfinding behavior in a layout. If different objects use different values, then the Pathfinding behavior must generate multiple obstacle grids in memory, and pathfind along them separately. You should also **avoid pathfinding every tick**, since this will cause extremely high CPU usage and also increase the amount of time it takes for other objects to determine their paths.

The grid of obstacles is only determined once on startup. If objects are moved in the layout, the pathfinding grid is **not** updated, and objects will continue to pathfind as if the objects were in their old positions. To update the entire obstacle grid use the *Regenerate obstacle map* action, but note this is a very CPU-intense operation and should only be done on one-off occasions. It is much more efficient to update only small parts of it (ideally only the area that has changed), which can be done with the *Regenerate region* and *Regenerate region around object* actions.

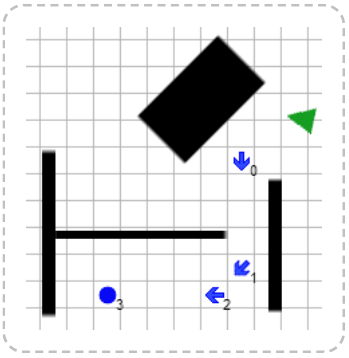
The obstacle map also applies to all instances using the Pathfinding behavior - different instances do not store separate obstacle maps, and so different per-instance obstacle settings will not take effect.

Note all cells outside the layout area are always obstacles. Areas outside the layout area cannot be included in the pathfinding grid, since doing so would require an infinite amount of memory.

## Finding paths

Calculating a path can take a long time, especially if the cell size is small. To prevent this reducing the game's framerate, the paths are calculated in the background (using a Web Worker). This means after using the *Find path* action, the resulting path is **not** immediately available. You must wait for the *On path found* trigger to run. Only then can you move the object along the path, or access the list of nodes from the behavior's expressions. The game may continue to run for a fraction of a second in between *Find path* and *On path found*.

The result path is a sequence of nodes along the grid. The image below demonstrates a four-node path (nodes 0 to 3).



The nodes can be retrieved (only after *On path found*) using the *NodeCount* and *NodeXAt*/*NodeYAt* expressions. Alternatively, the *Move along path* action can be used to automatically move the object along the nodes, using the speed, acceleration and rotation rate set in the behavior's properties.

Note it may be impossible to find a path, such as trying to navigate to a destination inside a ring of obstacles. In this case, *On failed to find path* will be triggered instead of *On path found*.

If you ask the pathfinding behavior to pathfind to a destination inside an obstacle, it will simply find the nearest clear cell and pathfind to there instead.

## Pathfinding properties

**Cell size**

The cell size, in pixels, of the grid of obstacles. See above for more details about how this is used.

**Cell border**

The amount, in pixels, to expand the cell size by when testing for obstacles. See above for more details about how this is used.

### Obstacles

If *Solids*, the behavior will automatically mark cells touching objects with the Solid behavior as being obstacles. If *Custom*, you must define which objects are obstacles by using the *Add obstacle* action on startup.

> *Note this applies to all instances, since the obstacle map is shared. This setting cannot be used to affect individual instances differently.*

### Max speed

If the *Move along path* action is used, the maximum speed in pixels per second the object can move at.

### Acceleration

If the *Move along path* action is used, the acceleration rate in pixels per second per second.

### Deceleration

If the *Move along path* action is used, the deceleration rate in pixels per second per second, used when approaching the final node.

### Rotate speed

If the *Move along path* action is used, the rate at which the object can rotate in degrees per second. Note this can affect the speed of the object: if the rotation speed is low, the object will have to slow down on tight corners.

### Rotate object

Whether to automatically set the angle of the object with the behavior to the angle of motion.

### Diagonals

Whether paths moving along diagonals are allowed. If disabled, the result nodes along paths will only ever change at 90-degree angles (up, right, down and left). If enabled nodes can move along diagonals as well.

### Direct movement

Specifies where path nodes may be removed if the box of cells enclosing nodes is completely clear. The options are:

- **None:** no path nodes are ever removed. This means even paths across entirely clear areas will still place nodes for movement along the pathfinding grid.

- **To destination:** if the area enclosing both the start and end positions of a path are completely clear, then this removes all path nodes except the last, allowing direct movement from the start to the destination. However if the area is not completely clear, it will still add path nodes to move along the pathfinding grid.

- **Anywhere along path:** checks for any groups of nodes along the whole path where the surrounding area is completely clear, and removes nodes so the path goes directly across the clear area. This can provide a smoother path with fewer nodes and fewer turns.

See the Pathfinding direct movement example for a visualization of how the setting changes paths.

---

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Pathfinding conditions

---

**Compare speed**

If moving along a path, compare the current speed of the object in pixels per second.

---

**Diagonals are enabled**

True if the *Diagonals* property allows moving diagonally along cells. This can also be changed with the *Set diagonals enabled* action.

---

**Is calculating path**

True if the object is currently calculating a path in the background. This is true between the *Find path* action and the *On path found* or *On failed to find path* triggers.

---

**Is cell obstacle**

Test if a cell in the obstacle grid is marked as an obstacle. This is useful for debugging or displaying the obstacle grid. Note the position is taken in cell co-ordinates rather than layout co-ordinates.

---

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

---

**Is moving along path**

True after using the *Move along path* action until *On arrived* triggers.

---

**On arrived**

Triggered after *Move along path* when the object finally arrives at its destination.

---

**On failed to find path**

Triggered after the *Find path* action if no path can be found to the destination, such as if it is surrounded by a ring of obstacles.

---

**On path found**

Triggered after the *Find path* action once a path has successfully been found to the destination. The nodes are now available via the *NodeCount*, *NodeXAt* and *NodeYAt* expressions, and the *Move along path* action can also be used.

## Pathfinding actions

---

**Add obstacle**

If the *Obstacles* property is *Custom*, add an object type to mark as an obstacle in the pathfinding grid. If this is done during the game (after *Start of layout*), you must also use *Regenerate obstacle map* for it to take effect.

> *Note this applies to all instances of both object types involved. This action does not affect individual instances.*

---

**Add path cost**

Add an object to increase the path cost in the pathfinding grid. This can be used to simulate rough terrain - the behavior will try to find paths around them if possible, unless the route is a major shortcut. See the *Pathfinding path cost* in the Start Page for a demo. If this is done during the game (after *Start of layout*), you must also use *Regenerate obstacle map* for it to take effect.

---

**Clear cost**

Remove all path cost objects added with *Add path cost*. You must also use *Regenerate obstacle map* for this to take effect.

---

**Clear obstacles**

Remove all obstacle objects added with *Add obstacle*. You must also use *Regenerate obstacle map* for this to take effect.

**Find path**

Start calculating a path to a destination in the layout. This is processed in the background and the results are not immediately ready after this action; you must wait until the *On path found* or *On failed to find path* triggers run before the result is known or the path can be moved along. If this action is used while *Is calculating path* is true, the old path is still calculated and the result triggered, but it then immediately begins calculating the new path and will also trigger for that result.

**Regenerate obstacle map**

Determine whether each cell in the obstacles grid is an obstacle again. This is a very CPU intensive action and should not be used regularly. If only part of the obstacle map has changed, prefer to use one of the *Regenerate region* actions. Any changes made by using *Add obstacle*, *Clear obstacles*, *Add path cost* and *Clear cost* will take effect the next tick after this action. Note this means if you attempt to find a path immediately after this action, the obstacle map won't have been updated yet; add a *Wait* action with a short delay to make sure the updated map is used in that case.

**Regenerate region**

**Regenerate region around object**

As with *Regenerate obstacle map*, but only the specified area is updated. This is usually considerably faster than regenerating the entire map. However as with regenerating the entire obstacle map, changes only take effect next tick. *Regenerate region* takes a rectangle in layout co-ordinates to regenerate. *Regenerate region around object* similarly regenerates the rectangle in the layout given by an object's bounding box. Note if multiple instances have met the event's conditions, this will regenerate multiple rectangles (one for every picked object).

**Set enabled**

Set whether the behavior is enabled or disabled. If disabled, it will not calculate any paths or move the object.

**Set move cost**

Set the base path cost for moving a single cell. This affects the relative cost of additional costs added by other features such as the *Add path cost* action and path groups. The default is 10. The move cost is rounded to an integer, and it is multiplied by the square root of 2 for the diagonal move cost if diagonals are enabled.

**Start path group**

**End path group**

Start and end a *path group*, which can be used to spread out paths found while inside the group. When a path is found inside the group, it adds a cost to cells along the discovered path in order to discourage subsequent paths in the same group from following the same path. Once the path group is ended, all added costs from the group are removed and normal pathfinding is restored. The parameters that affect path groups are:

- **Base cost:** The cost to add for each cell along a found path. Higher costs will spread out paths more. Paths are found across multiple independent workers which don't include the costs added by paths found in other workers; in order to compensate for this, the base cost is multiplied by the number of workers.

- **Cell spread:** How many cells around the found path to add the cost to. A higher cell spread will cause paths to be spread out more. Prefer to use an odd number for a symmetrical spread around the path. For example a cell spread of 1 only adds costs to the cells directly along the path, and a cell spread of 3 will add costs to every 3x3 box of cells along the path.

- **Max workers:** The maximum number of Web Workers allowed to be used for finding paths inside this path group. As each worker is independent and doesn't see the costs added from paths found in other workers, using multiple workers can still result in some paths being found along the same route. Limiting to 1 worker avoids this happening, but also can significantly reduce performance, as it will not use all available CPU cores for pathfinding. Using a higher number improves performance, but can still result in some repeat paths. This number can only reduce the number of workers used (i.e. if it is higher than the number of CPU cores it will have no effect).

See the Pathfinding groups example for a visualization of the effect of spreading out paths with this feature.

---

### Move along path

Automatically start moving the object along the found path. This can only be used after *On path found* - the path is not immediately known after the *Find path* action.

> *You can also use the Move To behavior's Move along Pathfinding path action as an alternative, since the Move To behavior uses a different movement algorithm.*

---

### Set speed

Set the current speed of the object if it is currently moving along its path, in pixels per second. This cannot be negative or greater than the maximum speed of the behavior.

---

### Stop

If the object is moving along its path, causes it to stop.

---

**Set acceleration**

**Set deceleration**

**Set diagonals enabled**

**Set max speed**

**Set rotate speed**

**Set direct movement**

Set the corresponding behavior properties. See the property definitions above for more information.

## Pathfinding expressions

---

**CurrentNode**

When moving along a path, the zero-based index of the node the object is currently moving towards. This may skip ahead just before the object actually reaches the next node, in order to help it round corners.

---

**MovingAngle**

The current angle of motion when moving along a path, in degrees.

---

**NodeCount**

The number of nodes in the path that was found. This is only updated after *On path found*.

---

**NodeXAt**

**NodeYAt**

Return the position of a node in the path that was found, in layout co-ordinates, using the zero-based index of the node. This is only available after *On path found*.

---

**Speed**

The current speed in pixels per second when moving along a path.

---

**Acceleration**

**CellSize**

**Deceleration**

**MaxSpeed**

**RotateSpeed**

Return the current values of the behavior properties. For more information, see the property definitions above.

# PERSIST BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/persist

The **Persist behavior** makes the object remember its state when going to a different layout, then coming back. If a layout has objects with the Persist behavior, it is also referred to as a *persistent layout*. For an interactive example of its use, click here to open the *Persistent Layouts* example.

Normally if you leave a layout then come back, all non-global objects reset to their initial design in the Layout View. However for many games this is undesirable; powerups come back, enemies respawn and return to full health, and so on. Adding the Persist behavior to certain objects means that they are restored to the same state you left them in when returning to a layout. Any objects that were previously destroyed remain destroyed; any new objects that were created will come back; and all properties such as instance variables are remembered. This is important for allowing the user to return to previous areas without having to redo the whole section.

# PHYSICS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/physics

The **Physics behavior** simulates realistic object physics. It is powered by Box2D. Construct comes with several examples demonstrating what the Physics behavior can do; filter by the *Physics* behavior in the Example Browser to find them.

The Physics behavior is relatively complex. The following tutorials are recommended to gain a basic understanding of how to use the Physics behavior and some important points to know before beginning to use it:

- Physics in Construct: The basics
- Physics in Construct: Forces, impulses, torque and joints

This manual section will not repeat the information in these tutorials. Instead it will describe each feature of the Physics behavior. The tutorials describe how Physics engines work, what the different types of joints are, the difference between impulses and forces, and so on in case you're not already familiar with them.

### Scripting

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

## Using Physics in Construct

The Physics behavior simulates physics separately to the Construct layout. Construct will try to keep the Physics and Construct "worlds" synchronised if one changes but not the other, but this can be unpredictable. For example, setting an object's position or angle will cause Construct to teleport the corresponding object in the physics simulation to the object's new position, which does not always properly take in to account collisions. The same is true of using other Construct behaviors at the same time as Physics.

Therefore it is highly recommended to control Physics objects entirely via the Physics behavior (by setting forces, impulses, torques etc.), rather than trying to manipulate objects by *Set position*, *Set angle* etc.

Another consequence is Physics won't respond to objects with the *Solid* or *Jumpthru* behaviors. These behaviors are totally redundant when using Physics and have no effect. Instead, use the *Immovable* property.

## Physics properties

**Immovable**

If enabled, simulate the object having infinite mass. Its density is ignored and it will never move.

---

**Collision mask**

How to handle physics collisions for this object. The options are:

- **Use collision polygon** uses the object's collision polygon from the Animations editor for physics collisions. If it doesn't have a collision polygon it will use the object's bounding box.

- **Bounding box** ignores the object's collision polygon if any, and for the purposes of Physics collisions considers the object to be a rectangle.

- **Circle** ignores the object's collision polygon if any, and for the purposes of Physics collisions considers the object to be a circle. This allows objects to smoothly roll along (like for example barrels). This is especially useful since object's collision polygons is made out of straight lines, so a smooth circle cannot be created that way.

---

**Prevent rotation**

Prevents the object from rotating when colliding with other Physics objects. Note this does not completely prevent the object from rotating - it only stops rotation caused by Physics collisions.

---

**Density**

The density of the physics object. Only used if *Immovable* is disabled. The object mass is calculated as its density multiplied by the area of its collision mask. The exact density values used are not important and have no specific units - only the relative density is significant (i.e. an object with density 6 will be twice as dense as an object with density 3).

---

**Friction**

The friction coefficient of the physics object from 0 (no friction) to 1 (maximum friction). This adjusts how easily objects move against each other while touching.

---

**Elasticity**

The elasticity (also known as *restitution* or *bounciness*) of the physics object, from 0 (inelastic, like a rock) to 1 (maximum elasticity, like a rubber ball). This affects how high objects bounce when hitting the floor.

---

**Linear damping**

The rate the object slows down over time while moving, from 0 (no slowdown at all) to 1 (maximum slowdown).

**Angular damping**

The rate the object slows down over time while spinning, from 0 (no slowdown at all) to 1 (maximum slowdown).

**Bullet**

Enable enhanced collision detection for fast-moving objects. This can affect performance, so do not enable it unless the object moves so fast the physics engine's standard collision detection is unreliable.

**Enabled**

Whether the physics simulation is initially enabled or disabled. If disabled, no physics is processed for the object, and other physics objects can pass through the object as if it were empty space.

## Physics conditions

**Compare angular velocity**

Compare the current angular velocity of the physics body, in degrees per second. A positive value indicates clockwise rotation and a negative value indicates anticlockwise rotation.

**Compare mass**

Compare the mass of the physics body. This is determined by multiplying the *Density* by the area of the object's collision polygon.

**Compare velocity**

Compare the current velocity (speed) of the physics body, in pixels per second. The velocity can be compared on an individual axis, such as just the X axis to compare the horizontal motion, or the overall velocity can be used.

**Is enabled**

True if the physics behavior is currently enabled. When disabled the physics body is completely removed from the physics simulation, so other physics objects will pass through the object.

**Is immovable**

True if the physics behavior is currently set to immovable (having an infinite mass and so never moving).

---

**Is sleeping**

True if the object has been at rest and not moved or been disturbed for a while, so that the physics engine can stop processing it. Note objects can still be moving imperceptibly which can prevent them from being asleep even when they appear to be stopped.

# Physics actions

## Forces

---

**Apply force**

**Apply force at angle**

**Apply force towards position**

Apply a force on the object, either at an angle, towards a position, or with custom X and Y axis forces. Applying a force causes the object to accelerate in the direction of the force. Forces can be applied at an image point, which normally also causes the object to rotate. Using `0` for the image point uses the object's center of mass, which does not cause rotation. Use `-1` to use the object's origin, which may be different to the center of mass and cause rotation.

---

**Apply impulse**

**Apply impulse at angle**

**Apply impulse towards position**

Apply an impulse on the object, either at an angle, towards a position, or with custom X and Y axis impulses. Applying an impulse simulates the object being struck, e.g. hit by a bat. Impulses can be applied at an image point, which normally also causes the object to rotate. Using `0` for the image point uses the object's center of mass, which does not cause rotation. Use `-1` to use the object's origin, which may be different to the center of mass and cause rotation.

---

**Set velocity**

Set the object's current velocity directly, providing a speed in pixels per second for the X and Y axes.

---

**Teleport**

Set the object position preserving the Physics velocity. Normally changing the position of the object will reposition it, but alter the velocity to try to ensure the physics simulation remains realistic even though some external change was made to the object position. Using the

*Teleport* action will reposition the object but not alter its Physics velocity in any way, which is sometimes desirable for purposes such as if a Physics object goes through a portal and is meant to appear somewhere else but with the same velocity.

## Global settings

These actions affect Physics behaviors in general, not just the one it was set for.

---

**Enable/disable collisions**

By default, all Physics objects collide with each other. You can disable collisions between the object and another Physics object so they pass through each other. This affects all instances of both object types. Note: enabling collisions again when objects are overlapping can cause instability in the simulation.

---

**Set stepping iterations**

Set the number of velocity iterations and position iterations used in the physics engine. The default is 8 and 3 respectively. Lower values run faster but are less accurate, and higher values can reduce performance but provide a more realistic simulation.

---

**Set stepping mode**

Choose whether the Physics time step uses *dt* (delta time, for *Framerate independent*) or a *fixed* value. By default it uses delta time to ensure physics progresses at a regular speed on displays with different refresh rates. However in some circumstances this can produce non-deterministic results, and note the Physics behavior clamps the maximum time step to 1/30 (about 33ms, equivalent to 30 FPS) to prevent the instability that can result from large time steps. Set to *Fixed* to always use a time step of 1/60 (about 16ms) regardless of the framerate, which makes the simulation deterministic, but will cause it to run at different speeds on displays with different refresh rates.

---

**Set world gravity**

Set the force of gravity affecting all Physics objects. By default this is a force of 10 downwards.

## Joints

---

**Create distance joint**

Fix two physics objects at a given distance apart, as if connected by a pole. An image point can be specified to connect to a specific part of the object. Note that an image point of 0 specifies the center of gravity of the object - if you intend to connect to the object origin, use -1.

---

**Create revolute joint**

**Create limited revolute joint**

Hinge two physics objects together, so they can rotate freely as if connected by a pin. Limited revolute joints only allow rotation through a certain range of angles, like the clapper of a bell. An image point can also be specified to connect to a specific part of the object. Note that an image point of 0 specifies the center of gravity of the object - if you intend to connect to the object origin, use -1.

---

**Create prismatic joint**

Restrict the movement of two physics objects along a specific axis, given by its angle. *Enable limit* specifies whether there is a lower and upper movement limit; if enabled these are given by the lower and upper translation, otherwise unlimited movement is allowed along the axis. A motor can also be enabled to provide a continuous force along the axis.

---

**Remove all joints**

Remove all joints from the object. Any objects this object was attached to via joints is also affected. Note some joints automatically disable collisions between the objects, so you may want to manually disable collisions again after removing joints otherwise overlapping objects will "teleport" apart (as the physics engine will try to prevent them overlapping).

## Object settings

These set the corresponding properties. For more information, see *Physics properties*. The one exception is the **Set awake** action.

---

**Set awake**

Physics simulations are relatively CPU intensive, requiring a lot of calculations. To save processor time, the Physics engine will make objects that have come to a complete stop go in to "sleep" mode so they no longer require processing. However sometimes changes like repositioning an adjacent object will leave the object in "sleep" mode so it will not respond properly. In this situation the *Set awake* action can be used to force a sleeping object to resume simulation. (It can also be used to force an object to go in to "sleep" mode, but note this is normally done automatically when possible.)

## Torque

---

**Apply torque**

**Apply torque towards angle**

**Apply torque towards position**

Apply a torque (rotational acceleration) to the object, either directly, or towards an angle or position.

**Set angular velocity**

Set the angular velocity (rate of rotation) directly, in degrees per second.

## Physics expressions

**AngularVelocity**

The current angular velocity (rate of rotation) of the physics object, in degrees per second.

**CenterOfMassX**

**CenterOfMassY**

The position of the center of mass of the physics object, as calculated by the physics engine. This depends on the *collision mask* property, and is not necessarily in the middle of the object.

**ContactCount**

Return the number of locations the physics engine has identified this object as touching other physics objects.

**ContactXAt(index)**

**ContactYAt(index)**

Return the position of a contact with another physics object, in layout co-ordinates, given by the zero-based index of the contact.

**Mass**

The mass of the physics object, as calculated by the physics engine. This is the area of the object's collision mask multiplied by its density.

**VelocityX**

**VelocityY**

The current speed of the physics object, in pixels per second.

**AngularDamping**

**Density**

**Elasticity**

**Friction**

**LinearDamping**

These return the corresponding properties. For more information, see *Physics properties*.

# PIN BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/pin

The **Pin behavior** positions an object at a relative distance and angle to another object, giving the impression it has been "pinned" to the object. It can also pin other properties like the size. For examples, search for *Pin* in the Start Page.

Simply adding the Pin behavior to an object does not do anything. You must use the *Pin* action to pin the object to another object.

> *The hierarchies feature is a modern built-in replacement for the Pin behavior. Consider using the Add child action to connect objects together instead. This has better support for connecting chains of objects together, provides conditions that can pick instances across the hierarchy, and also allows hierarchies to be set up in the Layout View. See also Superseded features.*

## Pin properties

**Destroy with pinned object**

Enable to automatically destroy this object if the object it is currently pinned to is destroyed.

## Pin conditions

**Is pinned**

True if the object is currently pinned to another object.

**Will destroy with pinned object**

True if the *Destroy with pinned object* option is enabled.

## Pin actions

**Pin at distance**

Pin the object to another object, restricting the distance between the objects. The two possible modes are:

- **Rope style (maximum distance)**: the object is kept at a maximum distance from the other object, but is allowed to move closer.

- **Bar style (fixed distance)**: the object is kept at a fixed distance from the other object, and will be moved away if the object object gets closer.

Note the distance between the objects at the time this action runs is used as the distance limit between the objects. The distance limit can be changed with the *Set pin distance* action.

---

**Pin to object**

**Pin to image point**

Pin one or more properties of the object to another object. The relative difference between the objects at the moment the *Pin* action is used is remembered. A series of checkboxes allows selection of which properties are to be kept pinned. For example ticking only the *X* and *Y* options will keep the object at the same relative position, but not change its angle. The *Width* and *Height* options have two possible modes if enabled: *Absolute*, which will apply the same size change value (e.g. if the pinned object gets 10 pixels wider, this object will also get 10 pixels wider); and *Scale*, which will apply the same relative size change (e.g. if the pinned object gets 50% wider, this object will also get 50% wider, relative to its starting size).
*Pin to image point* works the same as *Pin to object*, except instead of enabling the *X* and *Y* properties, it specifies an image point on the pinned object (by its name or number). The object will be positioned exactly at that image point while pinned, rather than keeping a relative difference.

---

**Set destroy with pinned object**

Set the current state of the *Destroy with pinned object* property.

---

**Set pin distance**

When using *Pin at distance*, set the distance limit in pixels that is used.

---

**Unpin**

Unpin the object, so it is no longer being positioned by the Pin behavior.

## Pin expressions

---

**PinnedUID**

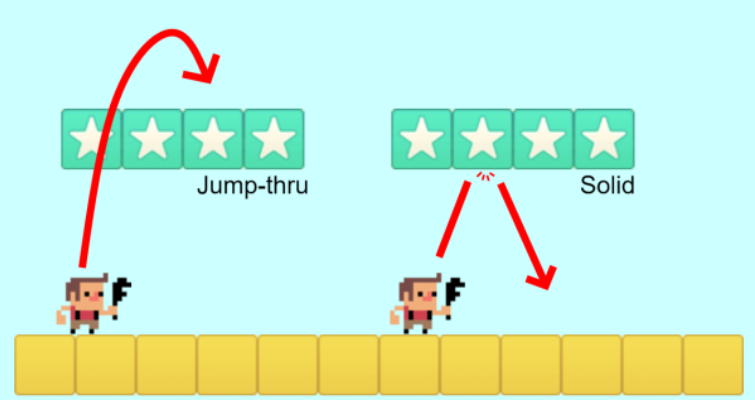
Get the UID of the object currently pinned to, or -1 if not pinned. For more information on UIDs, see instances.

# PLATFORM BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/platform

The **Platform behavior** implements a side-view "jump and run" style movement. It supports slopes, moving platforms, "jump-thru" platforms, and arbitrary angles of gravity. There are several examples of the Platform behavior that can be found in the Start Page.

The Platform behavior will land on any objects with the Solid or Jump-thru behaviors. Jump-thru is different in that the Platform movement can jump on to a Jump-thru from underneath, whereas jumping in to a solid from underneath causes the player to bounce off. The image below illustrates the difference.




*Jump-thru vs. Solid behaviors*

By default the Platform movement is controlled by the left and right arrow keys and up arrow to jump. To set up custom or automatic controls, see the behavior reference summary.

### Scripting

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

## Implementing reliable platform movements

For the most reliable platform movement, it is recommended to use a **invisible rectangle sprite with no animations** with the Platform movement. Then, the animated player object can be positioned on top of that. Otherwise, the changing size and collision polygon of the object as its animation plays can interfere with the Platform movement's floor, wall and slope detection, causing a shaking or glitchy movement. For more information and a complete guide, see the tutorial How to make a Platform game.

## Mirroring the player

It is not necessary to duplicate all your artwork for the player for facing both left and right. Instead, simply draw all the player's artwork for facing to the right, and use the Sprite object's *Set mirrored* action to mirror the player's image. Set the player mirrored when pressing the movement controls. An example is shown below.




A common question is how to make the player shoot left and right, since even when mirrored the player will still shoot objects to the right. To solve this, use the *Is mirrored* condition, and if it is true, shoot to the left instead.

# Platform properties

**Max speed**

The maximum floor speed in pixels per second.

**Acceleration**

The horizontal movement acceleration in pixels per second per second.

**Deceleration**

The horizontal movement deceleration in pixels per second per second. When moving in the opposite direction to the direction of motion, acceleration and deceleration combine.

**Jump strength**

The initial vertical speed of a jump in pixels per second when the jump key is pressed.

**Gravity**

The acceleration caused by gravity, in pixels per second per second.

**Max fall speed**

The maximum speed in pixels per second the object can accelerate to when in free-fall.

**Double jump**

If enabled, the player may make one extra mid-air jump before landing on the ground.

**Jump sustain**

Maximum time in milliseconds that the jump strength is sustained at while the jump control is being held before the effect of gravity takes over. This allows for variable height jumps depending on whether the jump control is tapped or held. For example if set to 200, then the jump velocity is sustained for up to the first 200ms of holding the jump control.

**Default controls**

If enabled, movement is controlled by the left and right arrow keys and the up arrow key to jump. Disable to set up custom controls using the *Simulate control* action. For more information see the behavior reference summary.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the Set enabled action.

## Platform conditions

**Compare speed**

Compare the current speed of the object in pixels per second.

**Is by wall**

Test if a solid blocking horizontal movement is immediately to the object's left or right. Jump-thrus do not count as walls.

**Is double-jump enabled**

True if double jumps are currently enabled. This is set by the *Double jump* property or *Set double-jump* action.

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

**Is falling**

True if the object is in free-fall.

**Is jumping**

True if the object is moving upwards.

**Is moving**

True if the object's speed is non-zero.

**Is on floor**

True if the object is currently standing on a solid or jump-thru.

**On fall**

**On jump**

**On landed**

**On moved**

**On stopped**

These are *animation triggers*, which trigger when the platform movement is moving in to each state. If your object has animations for any of these states, you should set the appropriate animation in each trigger. This helps save you implementing the logic to detect state transitions yourself.

## Platform actions

**Fall through**

If the player is currently standing on a jump-thru platform, this action will make them fall through it. This is useful for adding an additional control, e.g. down arrow, to jump down from jump-thru platforms.

**Reset double jump**

Change whether a double-jump is allowed during the current jump. If disabled a double-jump will no longer be allowed in the current jump, even if it is the first jump; if enabled a double-jump will be allowed again, even if a double-jump was already made in the current jump.

**Set acceleration**

**Set deceleration**

**Set default controls**

**Set double-jump**

**Set gravity**

**Set jump strength**

**Set jump sustain**

**Set max fall speed**

**Set max speed**

Set the corresponding properties. For more information, see *Platform properties*.

**Set angle of gravity**

Change the angle of gravity, in degrees. By default it is 90 (downwards on the screen). This can interact interestingly with layer rotation.

---

**Set ceiling collision**

Change how the behavior handles collisions with the ceiling, when the movement is jumping up and hits a solid. The default mode is *Stop* which means the vertical momentum is immediately set to zero so the movement immediately falls down again. Changing the mode to *Preserve momentum* will not change the vertical momentum when hitting a solid while jumping. This means the player can stick to the ceiling until gravity overcomes their upwards momentum; it can also allow the player to jump up past a corner of a solid even if they hit the solid vertically first.

---

**Set enabled**

Enable or disable the Platform movement. When disabled, the behavior has no effect on the object at all.

---

**Set ignoring input**

Set whether input is being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move, e.g. if in free-fall.

---

**Set vector X**

**Set vector Y**

Manually set the horizontal and vertical components of motion, in pixels per second. For example, setting the vector Y to -1000 would cause a jump with strength 1000, which could be useful for implementing springs.

---

**Simulate control**

Simulate one of the movement controls being held down. Useful when *Default controls* is disabled. See the behavior reference summary for more information.

## Platform expressions

---

**Acceleration**

**Deceleration**

**Gravity**

**JumpStrength**

**JumpSustain**

**MaxFallSpeed**

**MaxSpeed**

Return the corresponding properties. For more information, see *Platform properties*.

---

**GravityAngle**

Get the current angle of gravity, in degrees.

---

**MovingAngle**

Return the current angle of motion in degrees, which can be different to the object's angle.

---

**Speed**

Return the current overall speed in pixels per second.

---

**VectorX**

**VectorY**

Return the current X and Y components of motion, in pixels per second.

# ROTATE BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/rotate

The **Rotate behavior** makes an object spin.

### Scripting

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

## Rotate properties

**Speed**

The rate of rotation, in degrees per second. Use a positive value for clockwise rotation and negative for anticlockwise rotation.

**Acceleration**

The rate the rotation speed changes, in degrees per second per second. Use a positive value to accelerate clockwise and a negative to accelerate anticlockwise.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

**Preview Paid plans only**

Enable to run a preview of the behavior directly in the Layout View.

## Rotate conditions

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

## Rotate actions

**Set acceleration**

**Set speed**

Set the corresponding properties. See *Rotate properties*.

### Set enabled

Enable or disable the movement. If disabled, the behavior will stop rotating the object.

## Rotate expressions

### Acceleration

Return the corresponding *Acceleration* property.

### Speed

Return the current rotating speed, in degrees per second. A positive value indicates clockwise rotation and a negative value indicates anticlockwise rotation.

# SCROLL TO BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/scroll-to

The **Scroll To behavior** centers the view on the object with the behavior. It is a shortcut for the *Scroll to object* system action. However, it also provides a *Shake* action to shake the screen, and if multiple objects have the Scroll To behavior, it will center the view in between all of them.

If you need more advanced scrolling, e.g. limited to certain regions or following the player after a delay, scroll to an invisible object which you control through events.

To scroll, the size of the layout must be bigger than the project's viewport size, 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 properties

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Scroll To conditions

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

## Scroll To actions

**Set enabled**

Enable or disable the behavior. When disabled, the scrolling will not be affected.

**Shake**

Shake the screen for a duration of time, by randomly offsetting the scroll position every tick. The *Magnitude* is the maximum distance in pixels from the scrolled position the view will be offset. The *Duration* is how long the shake will last in seconds. In *Reducing magnitude* mode, the *Magnitude* will gradually reduce to zero by the end of the shake duration. In *Constant magnitude* mode, the *Magnitude* will stay the same throughout the full duration of the shake, ending abruptly.

## Scroll To expressions

The Scroll To behavior has no expressions.

# SHADOW CASTER BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/shadow-caster

The **Shadow caster behavior** marks an object as casting a shadow from a Shadow light object. For more information, see the documentation for *Shadow light*.

Shadows are cast from the object's collision polygon, if it has one, otherwise its bounding rectangle.

### Scripting

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

## Collision polygon shape

Objects with the *Shadow caster* behavior must use convex collision polygons. Shadows will not render correctly if they use concave polygons. If you need a concave shape, this can always be achieved by placing multiple shadow caster objects next to each other to compose a concave shape out of convex parts.

## Shadow caster properties

**Height**

The simulated height of the object, which adjusts the length of shadow it casts. If the *Shadow light* height is less than or equal to the object height, it casts an "infinite" shadow which goes all the way offscreen; if it is higher, it uses the relative heights to calculate how long a shadow to cast. For example two objects with different heights will cast different length shadows.

**Tag**

A tag for this shadow casting object. A *Shadow light* object also has a tag, and can be set to only cast shadows from shadow casters with the same or different tags to itself. This can be used to have different *Shadow lights* casting shadows off different sets of objects, such as to have shadows working at different levels of Z order.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, the object will not cast a shadow.

## Shadow caster conditions

**Compare height**

Compare the current height property of the behavior to a value.

**Is enabled**

True if the behavior is currently enabled so it can cast shadows.

## Shadow caster actions

**Set enabled**

Enable or disable the behavior. If disabled, the object will not cast a shadow.

**Set height**

Set the height property of the behavior. For more information see *Shadow caster properties*.

**Set tag**

Change the tag of the behavior. For more information see *Shadow caster properties*.

## Shadow caster expressions

**Height**

Return the current height property.

**Tag**

Return the currently set tag for the behavior.

# SINE BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/sine

The **Sine behavior** can adjust an object's properties (like its position, size or angle) back and forth according to an oscillating sine wave. This can be used to create interesting visual effects. Despite the name, alternative wave functions like 'Triangle' can also be selected to create different effects. A visualisation of the different wave types can be found on Wikipedia.

Click here to open an example of the Sine behavior, which demonstrates each type of movement the behavior can use.

### Scripting

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

## Sine properties

**Movement**

The Sine behavior has the following modes:

- **Horizontal** moves the object left and right on the X axis
- **Vertical** moves the object up and down on the Y axis
- **Forwards/backwards** moves the object in a straight line back and forth along the angle the object is facing at, like an angled *Horizontal* mode.
- **Width** stretches the object wider and narrower
- **Height** stretches the object taller and shorter
- **Size** makes the object grow and shrink
- **Angle** rotates the object clockwise and anticlockwise
- **Opacity** oscillates the object opacity. Note object opacities can never go less than 0 or greater than 100.

- **Z elevation** moves the object up and down on the Z axis.

- **Value only** does not modify the object. It simply stores the oscillating value which can be accessed by the *Value* expression. This can be useful to create custom effects or modify any other object or behavior property based on the sine behavior.

---

**Wave**

The wave function used to calculate the movement. For a visualisation see this Wikipedia diagram.

- **Sine:** the default smooth oscillating motion based on a sine wave.

- **Triangle:** a linear back-and-forth motion.

- **Sawtooth:** linear motion with a jump back to start.

- **Reverse sawtooth:** reverse linear motion with a jump back to start.

- **Square:** alternating between the two maximum values.

---

**Period**

The duration, in seconds, of one complete back-and-forth cycle.

---

**Period random**

A random number of seconds added to the period for each instance. This can help vary the appearance when a lot of instances are using the Sine behavior.

---

**Period offset**

The initial time in seconds through the cycle. For example, if the period is 2 seconds and the period offset is 1 second, the sine behavior starts half way through a cycle.

---

**Period offset random**

A random number of seconds added to the period offset for each instance. This can help vary the appearance when a lot of instances are using the Sine behavior.

---

**Magnitude**

The maximum change in the object's position, size or angle. This is in pixels for position or size modes, or degrees for the angle mode.

**Magnitude random**

A random value to add to the magnitude for each instance. This can help vary the appearance when a lot of instances are using the Sine behavior.

**Enabled**

If disabled, the behavior will have no effect until the *Set active* action is used.

**Preview Paid plans only**

Enable to run a preview of the behavior directly in the Layout View.

## Sine conditions

**Compare magnitude**

Compare the current magnitude of the movement.

**Compare movement**

Compare the current movement property of the behavior.

**Compare period**

Compare the current period of the movement, in seconds.

**Compare wave**

Compare the current wave property of the behavior.

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

## Sine actions

**Set cycle position**

Set the progress through one cycle of the chosen wave, from 0 (the beginning of the cycle) to 1 (the end of the cycle). For example setting the cycle position to 0.5 will put it half way through the repeating motion.

**Set enabled**

Enable or disable the behavior. When disabled, the behavior does not affect the object at all.

**Set magnitude**

Set the current magnitude of the cycle. This is in pixels when modifying the size or position, and degrees when modifying the angle.

**Set movement**

Change the movement type of the behavior, e.g. from *Horizontal* to *Size*.

**Set period**

Set the duration of a single complete back-and-forth cycle, in seconds.

**Set wave**

Change the wave property of the behavior, choosing a different wave function to be used to calculate the movement.

**Update initial state**

The Sine behavior records the object's initial state upon its creation, and always oscillates relative to that, even if it is deactivated and later activated after the object has been modified. If the object changes and you wish for the Sine behavior to oscillate relative to the new state instead of its state upon creation, use this action to reset the initial state to the object's current state.

## Sine expressions

**CyclePosition**

Return a value from 0 to 1 representing the progress through the current cycle. For example, exactly half way through a cycle this returns 0.5.

**Magnitude**

Return the current magnitude of the cycle. This is in pixels when modifying the size or position, and degrees when modifying the angle.

**Period**

Return the current period of a single complete back-and-forth cycle in seconds.

**Value**

Return the current oscillating value. This will alternate as a positive and negative value centered on zero. This is useful to create custom effects when in *Value only* mode.

# SOLID BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/solid

The **Solid behavior** makes other behaviors react to the object as if it were an impassable obstacle. Objects with this behavior are referred to as Solids. It affects the following behaviors:

- 8 Direction, which is blocked by Solids
- Bullet, which can optionally bounce off Solids
- Car, which bounces off Solids
- Line-of-sight, which by default has line-of-sight obstructed by Solids
- Move To, which can optionally stop on solids.
- Platform, which can land on Solids. Platform cannot jump on to solids from underneath - for this, use the Jump-thru behavior.
- Pathfinding, which by default uses solids as path obstacles.
- Tile movement, which is blocked by Solids

Note that the Physics behavior is **not** affected by Solid objects. Instead, use the Physics behavior with *Immovable* enabled.

The Solid behavior is a fundamental attribute in Construct, and several other Construct features also interact with Solid objects. For example, the Custom Movement behavior has actions to push the object out of solids.

### Scripting

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

## Avoid crushing/trapping objects with Solids

The behaviors which respond to the Solid behavior usually get stuck if Solid objects crush or otherwise trap the object deep inside the Solid object. In this case there is no solution for the movement. The only three options are 1) let the object get stuck, 2) allow the object to move inside solids, or 3) teleport the object to the nearest free space, which in some cases can be quite far away. Since options 2 and 3 can cause strange glitches if allowed, Construct will deliberately make the object unable to move, and this is the intended behavior. Therefore, it is up to you to design your games in such a way that the player cannot be crushed or trapped by moving Solid objects. You should be especially careful when moving Solids up against other Solids.

It is only by moving (or re-enabling) Solids, or using *Set position*, that objects can become trapped. If none of the Solids in your game move and you do not "teleport" the player around with *Set position*, it should be impossible for the player to ever get trapped in solids.

## Solid properties

### Enabled

Set whether the behavior is initially enabled or disabled. If disabled, the object no longer acts as if it is solid, and objects will be able to pass through it.

### Tags

An optional list of tags to apply to the Solid, separated by spaces. This is referenced by the *Set solid collision filter* action, allowing collisions to be enabled and disabled with different sets of solids.

## Solid conditions

### Is enabled

True if the behavior is currently enabled. This can be changed by the *Enabled* property or the *Set enabled* action.

## Solid actions

### Set enabled

Enable or disable Solid for this object. Be careful not to trap objects by enabling the solid when an object is overlapping it; see *Avoid crushing/trapping objects with Solids*.

### Set tags

Change the *Tags* property, affecting solid collision filtering.

# TILE MOVEMENT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/tile-movement

The **Tile movement behavior** allows an object to be moved up, down, left and right, moving a fixed distance at a time, controlled by the arrow keys by default. This is useful for tilemap-based games, where the level is designed around a grid: by using the same grid size with the Tile movement, the controlled object will always be aligned with a grid cell.

The Tile movement aligns the object's origin with the grid. To ensure the object appears in the correct alignment with a visible grid, ensure the object size is the size of the cell, and place the origin in the top-left corner. Often it's easiest to make this object invisible, and place the player sprite on top so that it can keep its own size and origin, as is done in this example of the Tile movement behavior.

The Tile movement behavior is blocked by any objects with the Solid behavior.

To set up custom or automatic controls, see the behavior reference summary.

### Scripting

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

## Tile movement properties

**Grid width**

**Grid height**

The size of the movement grid cells, in pixels. The object moves in increments of this size. Normally this matches the size of a corresponding tilemap.

**Grid offset X**

**Grid offset Y**

The offset of the movement grid, in pixels. This can be used to align with a tilemap that is not aligned with (0, 0) in the layout.

**Speed X**

**Speed Y**

The speed of the movement on each axis, in pixels per second.

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

**Default controls**

If enabled, movement is controlled by the arrow keys on the keyboard. Disable to set up custom controls using the *Simulate control* action. For more information see the behavior reference summary.

**Isometric**

Check to enable an isometric grid movement, along diagonals. In this mode the "up" direction is towards the top-right. The default is unchecked, using a cartesian grid with horizontal and vertical movement.

## Tile movement conditions

**Can move to**

Test if the object can move into a grid cell, given by its grid column and row. This is true if the cell is clear of any obstacles.

**Can move in direction**

Test if the object can move N cells in a given direction. Unlike the **Can move to** condition this checks for obstacles between the object and the destination.

**Is enabled**

Test if the behavior is currently enabled.

**Is moving**

Test if the behavior is currently moving in to a cell.

**Is moving in direction**

Test if the behavior is currently moving left, right, down or up. This is useful for setting the corresponding animation.

## Tile movement actions

**Set default controls**

Enable or disable the *Default controls* property (see above for more details).

**Set enabled**

Enable or disable the movement. If disabled, the movement no longer has any effect on the object.

**Set grid dimensions**

Specify new values for the *Grid width*, *Grid height*, *Grid offset X* and *Grid offset Y* properties.

**Set grid position**

Set the object to a cell in the grid, given by its grid column and row. The *Movement* option allows the object to either be set to that position instantly, or animate over for a smooth movement over time.

**Set ignoring input**

Set whether input is being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

**Set speed**

Specify new values for the *Speed X* and *Speed Y* properties.

**Simulate control**

Simulate one of the movement controls being held down. Useful when *Default controls* is disabled. See the behavior reference summary for more information.

## Tile movement expressions

**GridTargetX**

**GridTargetY**

Return the grid cell the object is currently moving towards, by its column and row.

**GridX**

**GridY**

Return the current grid cell the object is in by its column and row.

**SpeedX**

**SpeedY**

Return the current speed of the object on each axis, in pixels per second.

**TargetX**

**TargetY**

Return the layout co-ordinates of the grid cell the object is currently moving towards, by its column and row.

# TIMER BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/timer

The **Timer behavior** triggers its *On timer* condition regularly or once after a delay. This is like using the system *Every X seconds* condition, or the system *Wait* action, except that times are kept for each instance individually. The rate of *On timer* triggering is affected by the time scale. The timer behavior is a more convenient alternative to adding *dt* to an instance variable every tick.

### Scripting

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

## Tags

A single Timer behavior can keep track of multiple timers. To distinguish between them, a *tag* is used. A tag is just a string, which can be anything. For example, starting a timer with tag "attack" will trigger *On timer "attack"*, but not *On timer "defend"*.

When a timer is stopped, or after a one-off timer triggers, it no longer exists and the timer expressions cannot be used to retrieve any information about it.

## Timer conditions

**Is timer paused**

True if a timer with the given tag has been started and then subsequently paused with the *Pause/resume timer* action.

**Is timer running**

True if a timer with the given tag has been started with the *Start timer* action. Once *Stop timer* is used, the timer no longer counts as running. Paused timers also count as running - use the *Is timer paused* condition to identify these timers separately.

**On timer**

Triggers either regularly, or once off, after a timer that was started with the same tag has reached its duration.

> ***Note:*** *this trigger can fire with multiple instances picked, if their timers all reach their time in the same tick. This can sometimes work unexpectedly if the actions expect there*

*to be just one instance picked. The workaround is to add a For each condition after this trigger to ensure the actions run once per instance.*

## Timer actions

**Pause/resume timer**

Set a currently running timer (started with the *Start timer* action) either paused or resumed. When a timer is paused, it will stop triggering *On timer*. When resumed, it will continue triggering *On timer*, resuming from the time that it was paused at. In other words if a timer is set for 1 second, it is paused after 0.5 seconds, and then after some time it is resumed again, *On timer* will trigger 0.5 seconds after resuming.

**Pause/resume all timers**

This does the same thing as the *Pause/resume timer* action, but affecting all existing timers rather than only one with a given tag.

**Start timer**

Set a new timer, or if the timer exists, re-start it with new options. *Duration* is the time until *On timer* triggers. If *Type* is *Once*, then *On timer* will fire once and not again until *Start timer* is used again; if *Regular*, then *On timer* will keep firing every *Duration* seconds. The tag allows multiple timers to run at once. The corresponding *On timer* condition must use the same tag.

**Stop timer**

Stop a timer with a specific tag. *On timer* will no longer trigger with the given tag after this action.

**Stop all timers**

Stop all currently running timers regardless of their tags. *On timer* will no longer trigger for any timer after this action unless a new timer is started.

## Timer expressions

**CurrentTime(tag)**

The time in seconds since *On timer* last triggered, for a timer with a specific tag.

**Duration(tag)**

The duration in seconds for a timer with a specific tag.

**NormalizedProgress(tag)**

The current progress for a timer with a specific tag represented as a number between 0 and 1, regardless of the duration of the timer. For example a normalized progress of 0.5 means half way through the duration.

---

**TotalTime(tag)**

The time in seconds since a timer with a specific tag was started with the *Start timer* action. This is only useful with regular timers, since it will always equal the *CurrentTime* expression for one-off timers (after which they fire and the timer no longer exists, so these expressions return 0).

# TURRET BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/turret

The **Turret behavior** can automatically detect objects within a certain range and rotate towards them. It optionally includes features to determine when to fire, as well as **predictive aim**.

For examples of the Turret behavior, search for *Turret* in the Example Browser.

### Scripting

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

## How Turrets work

Before a turret will target anything, you must use the *Add object to target* action so the object knows which objects to look for. Calling this once on the start of layout is sufficient. You can also give the turret a Family Paid plans only to target, allowing it to easily target a collection of different objects.

Once one of these objects enters the Turret's range (the distance between the objects is lower than the *Range* property), and the turret does not already have a target, then the turret acquires that object as a target. *On target acquired* is triggered, and if *Rotate* is enabled the object will start rotating towards the target. Once the turret is pointing in the direction of the target, *On shoot* will trigger at the frequency determined by the *Rate of fire* property. If you want the turret to fire upon the target, spawn a projectile in the *On shoot* event.

If the target leaves the turret's range, the turret will lose the target and stop firing. If another target is already in range, it will immediately acquire that; otherwise it will simply wait until the next target enters range. Also note if *Target mode* is set to *Nearest*, the turret may switch to another target before the current target leaves range, if the new target comes closer than the existing target.

## Predictive aim

A useful feature of the Turret behavior is the ability to use **predictive aim**. For an interactive demonstration of this see the *Turret predictive aim* example in the Start dialog.

Normally turrets aim directly at a target. This often means moving targets are never hit, because by the time the projectile arrives, the object has moved somewhere else. Predictive aim solves this by aiming the turret at where the object will be by the time the projectile arrives, *if* it maintains the same velocity. In order for this to work, the Turret behavior must have the speed of the projectile set in its *Projectile speed* property, so it can determine how long it will take for the

projectile to arrive. The projectile must also use a fixed speed, and not have any acceleration or deceleration.

It is still possible for targets to dodge predictive aiming turrets, by changing direction or speed while the projectile travels towards it. However this is considerably more difficult compared to not using predictive aim, and the overall accuracy of the turret will be significantly improved.

## Turret properties

**Range**

The range, in pixels, that the turret can detect targets in. Any targets further away from the turret than this distance will be ignored.

**Rate of fire**

The rate at which to trigger *On shoot*, when the turret has both acquired a target and rotated to point in the direction of the target.

**Rotate**

Whether to automatically set the object's angle according to the angle of the turret.

**Rotate speed**

The speed at which the turret can rotate towards targets, in degrees per second.

**Target mode**

If *First in range*, the turret will always track the same target until it leaves range, even if other targets come in range. If *Nearest*, the turret may switch to a different target before its current target leaves range, if another target comes closer.

**Predictive aim**

Whether to enable predictive aim or not. If enabled, you must set the correct *Projectile speed* for the predictive aim to work correctly. For more information see the section on *Predictive aim* above.

**Projectile speed**

If *Predictive aim* is enabled, this must be set to the projectile speed in pixels per second for the predictive aim to work correctly. For more information, see the section on *Predictive aim* above.

**Use collision cells**

Whether to use the collision cells optimisation when looking for targets that are within range. Usually this is faster, but with extremely long ranges it can sometimes be slower.

---

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Turret conditions

---

**Has target**

True if the turret currently has a target acquired.

---

**Is enabled**

Test if the behavior is currently enabled. When disabled it will have no effect on the object.

---

**On shoot**

Triggered at the frequency given by the *Rate of fire* property, when the turret both has a target and has rotated to point towards it. If the turret is to fire upon the target, you should spawn a projectile from the turret in this trigger.

---

**On target acquired**

Triggered when the turret has no target, but acquires one as it enters range.

## Turret actions

---

**Acquire target**

Target a specific object if it is in range. If the object is out of range, the action is ignored. If in range, the turret will switch to targeting the given object, even if it already has a different target. Note if *Target mode* is *Nearest*, the turret may still immediately switch to a nearer target.

---

**Add object to target**

Use on startup to tell the turret which objects it should target. Use a Family Paid plans only to conveniently target a whole group of objects.

---

**Clear targets**

Remove all targets added using the *Add object to target* action. The turret will no longer target any objects at all.

**Unacquire target**

Tell the turret to forget its existing target, even if it is in range. This frees it up to target a different object, but it may choose to immediately target the same object again unless the *Acquire target* action is used immediately afterwards.

**Set enabled**

Enable or disable the behavior. If disabled, the behavior will not detect targets, rotate the object, or run any triggers.

**Set predictive aim**

**Set projectile speed**

**Set range**

**Set rate of fire**

**Set rotate**

**Set rotate speed**

**Set target mode**

Set the corresponding properties. For more information, see *Turret properties*.

## Turret expressions

**Range**

**RateOfFire**

**RotateSpeed**

Retrieve the corresponding properties. For more information, see *Turret properties*.

**TargetUID**

Get the UID of the currently targeted object, if any. For more information about UIDs, see instances.

# TWEEN

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/tween

The **Tween behavior** animates the values of properties over time. For example you can "tween" an object's position to (100, 100), which will move it from its current position to the target position over time. Different *ease functions* can be used to alter the rate at which the value changes over time, for example using a *Linear* function for a constant-rate motion, *In Out Sinusoidal* for a sine-curve based movement which speeds up then slows down, and many more. Custom ease curves can also be designed with the Ease editor.

Click here to open an example of the Tween behavior.

The term "tween" comes from the term "Inbetweening", referring to generating intermediate frames in between two states.

> *Tweening to a position moves objects over a fixed time. If you need to move objects to a position using a maximum speed with acceleration and deceleration, use the Move To behavior instead.*

### Scripting

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

## Tween types

Construct's Tween behavior has three types of tween:

1. One-property tween, affecting only a single value, e.g. the X co-ordinate or the angle
2. Two-property tween, affecting a pair of values, e.g. the position (X and Y co-ordinates), size (width and height) or the scale (a percentage of real size)
3. Value tween, which just tweens a number rather than affecting a property. This can be used to apply the tween to anything else in events, such as behavior properties or effect parameters. Value tweens must also specify the start value, since it cannot be taken from a property.

These types are used by the three tween actions: *Tween (one property)*, *Tween (two properties)* and *Tween (value)*.

## Tagging

The Tween behavior can manage multiple tweens simultaneously. To help identify them separately, each tween can be given a *tag*, which is just a string of any name you like to identify the tween. The tag is optional and can be left empty if you don't need to modify or identify the tween at any point later on.

Tweens can also be given multiple tags, separated by spaces. This can be useful to group tweens together under a common tag while also providing a unique tag to target them individually.

Actions, conditions and expressions which expect tags as an argument will match any tweens which include all of the provided tags.

### Tagging examples

Given these tweens with these tags:

| Name | Tags |
|---|---|
| **Tween1** | "common size width" |
| **Tween2** | "common size height" |
| **Tween3** | "common color" |

This will be the expected matches:

| Tag | Matches |
|---|---|
| "common" | **All Tweens** |
| "common size" | **Tween1** and **Tween2** |
| "common size width" | **Tween1** |
| "color" | **Tween3** |
| "width" | **Tween1** |
| "size" | **Tween1** and **Tween2** |
| "common size width height" | **No Tweens** |

Also note:

- Actions using tags affect all matching tweens.
- In the case of conditions, all the matching tweens are evaluated and if any is true, then the condition will be fulfilled.
- In the case of expressions, the first matching tween will be the one used to get the value from.
- Tags are case insensitive. For example "mytag", "MyTag" and "MYTAG" are all the same as far as tweens are concerned.

## Tween properties

**Enabled**

Whether the behavior is initially enabled or disabled. If disabled, it can be enabled at runtime using the *Set enabled* action.

## Tween conditions

**Is any playing**

True if any tween is currently playing.

**Is playing**

Test if a tween matching all the given tags is currently playing.

**Is any paused**

True if any tween is currently paused.

**Is paused**

Test if a tween matching all the given tags is currently paused.

**Is any ping-pong**

True if any tween is in the provided ping-pong *State*

**Is ping-pong**

True if a tween matching all the given tags is in the provided ping-pong *State*

**On any finished**

Triggered when any tween finishes. Use the *Tags* expression to get the tag string for the tween that finished.

**On finished**

Triggered when a tween matching all the given tags finishes.

**On any released**

Triggered when any tween is released. Use the *Tags* expression to get the tag string for the tween that was released.

**On released**

Triggered when a tween matching all the given tags is released.

**On any looped**

Triggered when any tween loops. Use the *Tags* expression to get the tag string for the tween that looped.

### On looped

Triggered when a tween matching all the given tags loops.

### On any ping-pong

Triggered when any tween ping-pongs. Use the *State* parameter to specify at which point of the ping-pong the trigger should take place. Use the *Tags* expression to get the tag string for the tween that ping-ponged.

### On ping-pong

Triggered when a tween matching all the given tags ping-pongs. Use the *State* parameter to specify at which point of the ping-pong the trigger should take place. Use the *Tags* expression to get the tag string for the tween that ping-ponged.

## Tween actions - one property

### Set end value

Change the end value for an existing one-property tween matching all the given tags.

### Tween (one property)

Start a tween for a single property. *Tags* are optional space-separated tags to identify the tween, and can be left blank if not used. *Property* chooses which property of the object to modify. The start value of the tween uses the current value of the property. *End value* specifies the value to tween to. *Time is the duration of the tween in seconds. Ease* specifies an ease function affecting the rate of change over time. *Destroy on complete* can be set to *Yes* to automatically destroy the instance when the tween finishes, useful for fade-out effects. Like timelines, a one property tween can be set to *Loop* and/or *Ping Pong* and given a *Repeat count*.

## Tween actions - two properties

### Set end values

Change the end value for an existing two-property tween matching all the given tags.

### Tween (two properties)

Start a tween for two properties. *Tags* are optional space-separated tags to identify the tween, and can be left blank if not used. *Property* chooses which property pair of the object to modify. The start value of the tween uses the current value of the properties. *End X* and *End Y* specify the end value for each of the two properties. *Time is the duration of the tween in seconds. Ease* specifies an ease function affecting the rate of change over time. *Destroy on complete* can be set to *Yes* to automatically destroy the instance when the tween finishes,

useful for fade-out effects. Like timelines, a two property tween can be set to *Loop* and/or *Ping Pong* and given a *Repeat count*.

## Tween actions - value

**Set end value**

Change the end value for an existing value tween matching all the given tags.

**Set start value**

Change the start value for an existing value tween matching all the given tags.

**Tween (value)**

Start a tween for a number, independent of any properties. *Tags* are optional space-separated tags to identify the tween, and can be left blank if not used. *Start value* and *End value* specify the start and end value to tween through. *Time is the duration of the tween in seconds. Ease* specifies an ease function affecting the rate of change over time. *Destroy on complete* can be set to *Yes* to automatically destroy the instance when the tween finishes, useful for fade-out effects. Use the *Value* expression to retrieve the current value of the tween over time, such as to apply it to a different object, behavior or effect. Like timelines, a value tween can be set to *Loop* and/or *Ping Pong* and given a *Repeat count*.

## Tween actions - playback

**Pause**

**Resume**

Pause and resume an existing tween matching all the given tags. Pausing a tween will stop it at its current progress, and resuming will continue from where it was paused.

**Pause all**

**Resume all**

Pause and resume all current tweens.

**Stop**

Stop a tween matching all the given tags. Stopping a tween permanently ends a tween - it cannot be resumed afterwards.

**Stop all**

Stop all tweens. This permanently ends all tweens so no more tweens can be referenced until a new tween is started.

## Tween actions - setters

**Set destroy on complete**

**Set all destroy on complete**

**Set ease**

**Set all eases**

Set the *Destroy on complete* and *Ease* parameters specified in the *Tween* action for existing tweens matching all the given tags. The *all* variants modify all tweens regardless of tags.

**Set playback rate**

**Set all playback rates**

Set the playback rate of existing tweens matching all the given tags, or all tweens regardless of tags. A playback rate of 1 is normal speed, 0.5 half as fast, 2 twice as fast, and so on.

**Set time**

**Set all times**

Set the playback time in seconds of existing tweens matching all the given tags, or all tweens regardless of tags. For example setting a time of 1 will skip the tween to playing as if it was 1 second after the tween was started.

**Set enabled**

Enable or disable the entire behavior. If disabled, the behavior will not affect any properties or advance any tweens.

## Tween expressions

Note expressions can only return a single value. When specifying tags, expressions return the value for the first tween with all the given tags.

**Progress(tag)**

Return the progress of a tween by its tags in the range 0-1.

**Tags**

This expression can be used in the various triggers to get the tag string of the associated tween.

**Time(tag)**

Return the playback time of a tween by its tags, in seconds since the tween was started.

**PlaybackRate(tag)**

Return the playback rate of a tween by its tags.

**Value(tag)**

Return the current value of a value tween by its tags.

# WRAP BEHAVIOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/behavior-reference/wrap

The **Wrap behavior** re-positions an object to the opposite side of the layout or viewport if it leaves the area. The object only wraps once it is fully outside the area, i.e. no part of its bounding box is in the layout or viewport area.

## Wrap properties

**Wrap to**

Whether to wrap the object when it leaves the layout area, or when it leaves the visible viewport.

## Wrap conditions

**On wrap**

Triggered when the object position is changed by wrapping. For example if the object goes past the right edge of the wrap area and so is repositioned to the left of the wrap area, then *On wrap* will trigger.

# PLUGIN REFERENCE

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

This section provides a reference of all the official plugins that come with Construct. Each has an overview of its use, a list of its properties, and a detailed list of the actions, conditions and expressions specific to that plugin.

Many plugins share common actions, conditions and expressions. These are described in Common features rather than repeating the information for each plugin.

# COMMON FEATURES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/common-features

Many plugins share common conditions, actions and expressions, such as for the object's size and position. Rather than repeating the descriptions of these for each plugin, they are listed in this section.

Not all objects use all the common features; some may use only a few of the ones listed here. This is provided as a reference for all the possible features that may be shown to you in Construct rather than describing any particular plugin's features.

- Common conditions
- Common actions
- Common expressions

## Other common features

Many objects share other common features. This manual section only covers the common conditions, actions and expressions. For more information on other common features see the manual section on instances.

# COMMON CONDITIONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/common-features/common-conditions

The following conditions are common to several plugins.

## Angle

Note angles in Construct start with 0 degrees facing right and increment clockwise.

### Is between angles

True if the object's current angle is between the two given angles in degrees. The first and second angles must be in clockwise order. That is, *Is between 0 and 45 degrees* is true if the object is in a 45 degree area, but *Is between 45 and 0 degrees* is true if the object is in the 315 degree area from 45 degrees through 0 degrees.

### Is clockwise from

True if the object's current angle is clockwise from the given angle in degrees. Invert to test if the object is anticlockwise from the given angle. For example, an object at 45 degrees is clockwise from 0 degrees, but an object at 0 degrees is anticlockwise from 45 degrees.

### Is within angle

True if the object's current angle is within a number of degrees of another angle. This is more reliable than testing if the object's angle exactly equals an angle, e.g. *Is within 0.5 degrees of 90 degrees* is probably better than *Angle equals 90 degrees*, since there are many cases an object can be very close to, but not exactly, 90 degrees.

## Appearance

### Compare opacity

Compare the object's current opacity, from 0 (transparent) to 100 (opaque).

### Is effect enabled

Test if one of the effects added to the object is currently enabled.

### Is visible

True if the object is currently visible. Invert to test if invisible. This only tests the visibility set by the *Set visible* action; it is not affected by the object being offscreen, having 0 opacity, or

being on an invisible layer.

## Collisions

**Is overlapping another object**

**Is overlapping at offset**

True if any instance is overlapping any instance of another object. Collision polygons are taken in to account (if any), as well as the object's size and rotation. The *offset* variant will test for an overlap at an offset from the first object. For example, testing for an overlap at an offset of (100, 0) will temporarily move the object to the right 100 pixels, test for the overlap, then move it back again.

**On collision with another object**

Triggered upon the first time any instance starts overlapping any instance of another object. The collision polygons are taken in to account if set, as well as the object's size and rotation.

## Hierarchy

These conditions are available for plugins that support the *scene graph* feature, allowing objects to be connected together so they move, rotate and scale as if they were one large object.

**Compare child count**

Compare the number of children that are currently attached to the object (using the *Add child* action).The *Which* option can be set to:

- *Own*: only the object's direct children are compared
- *All*: all children of the object are compared, including children-of-children, all the way to the bottom of the hierarchy

**Has children**

True if any children have been attached to the object (i.e. the child count is greater than 0).

**Has parent**

True if this object is currently attached to another object.

**On hierarchy ready**

Triggered for the root (top parent) instance in a hierarchy after *On created* has triggered for all instances in the hierarchy. For a specific instance in a hierarchy, when *On created* triggers, it is not certain that *On created* has yet triggered for other instances in the hierarchy, which

can complicate initializing hierarchies. This trigger gives you an opportunity to initialize a hierarchy once *On created* has been triggered for all instances in the hierarchy.

---

**Pick children**

Pick the children of a given object type or family attached to this object. The *Which* option can be set to:

- *Own*: only the object's direct children are picked
- *All*: all children of the object are picked, including children-of-children, all the way to the bottom of the hierarchy
- *Bottom*: only children at the bottom of the hierarchy from this object are picked, i.e. children with no further children of their own.

---

**Pick nth child**

Pick a specific child of this object at a given zero-based index. This only picks from the object's own children (children-of-children are excluded). When *Index type* is *all*, the index is in to the list of all children of any type. The object type or family of the child must still be specified; if it is the wrong type, it won't be picked even if there is a child at the given index. When *Index type* is *filtered*, the index is in to the list of children of the specified object type only. For example if a parent has three children of object types *Leg*, *Arm*, and *Leg*, and the condition is used to pick child *Leg* at index 1, index type *all* will not pick anything (as there is an *Arm* child there instead), whereas index type *filtered* will pick the second *Leg* instance (as that is index 1 in to all *Leg* children).

---

**Pick parent**

Pick the parent of a given object type or family this object is attached to. The *Which* option can be set to:

- *Own*: only the object's direct parent is picked
- *All*: all parents of the object are picked, all the way to the top of the hierarchy
- *Top*: only the parent at the top of the hierarchy from this object is picked, i.e. the parent not attached to any other parent.

# HTML element

These conditions are available for some plugins in the *Form controls* category, like Button and Text Input. These objects are HTML elements placed on top of the canvas.

---

**Is enabled**

True if the element is currently enabled, and can be interacted with.

---

**Is focused**

True if the element is currently focused, meaning it will receive keyboard input. Typically this also involves a visual indication of focus as well.

---

**Is visible**

True if the element is currently visible. Otherwise the element still exists and preserves its contents, but is hidden.

## Instance variables

---

**Compare instance variable**

Compare the current value of one of the object's number or text type instance variables.

---

**Is boolean instance variable set**

Test if one of the object's boolean instance variables is set to *true*. Invert the condition to test if false.

---

**Pick highest/lowest**

Pick the single instance with the highest or the lowest instance variable value of all the instances. Note this still only picks a single instance even if multiple instances have the same highest or lowest value; in this case an arbitrary instance is selected.

## Misc

---

**On created**

**On destroyed**

Triggered for each instance that is created or destroyed during the running of the game. *On created* is also trigged for each object already on a layout when the layout starts. For example, a one-shot particle effect could be spawned every time an object is created, and an explosion created every time the object is destroyed. These conditions are analogous to *constructors* and *destructors* in traditional programming languages (commands which run at the creation and destruction of an object). Be careful not to create an object of the same type in an *On created* event (e.g. *On Sprite2 created: create Sprite2*) since this will create an infinite loop and cause the game to hang.

**On signal**

Triggered when the *Signal* action is run with a matching tag (case insensitive).

## Pick

**Pick by unique ID**

Pick the instance matching a given unique ID (UID) number.

**Pick nearest/furthest**

Pick the instance either nearest or furthest from a given position in the layout.

## Size & Position

**Compare width**

**Compare height**

Compare the object's current size, in pixels.

**Compare X**

**Compare Y**

Compare the object's current position in the layout, in pixels. Note that objects can be positioned between pixels, e.g. at (5.5, 10.33333). Because of this it's usually a bad idea to rely on an object being at an exact position.

**Is on-screen**

True if any part of the object's bounding box is within the screen area. This is not affected by the object's visibility or opacity.

**Is outside layout**

True if the entire object's bounding box is outside the layout area.

## Tags

**Has Tags**

Pass in a string of space separated tags to check if an instance has all of them or not.

## Z Order

**Compare Z elevation**

Pick instances according to their elevation on the Z axis.

---

**Is on layer**

Pick all instances on a given layer, specified either by its name or zero-based index.

---

**Pick top/bottom**

Pick either the top-most or bottom-most instance, taking in to account layers and Z index. For example, the instance at the front of the top most layer is the top instance.

# COMMON ACTIONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/common-features/common-actions

The following actions are common to several plugins.

## Angle

Note angles in Construct start with 0 degrees facing right and increment clockwise.

**Rotate clockwise**

Rotate the object a number of degrees clockwise.

**Rotate counter-clockwise**

Rotate the object a number of degrees counter-clockwise.

**Rotate toward angle**

Rotate the object a number of degrees toward another angle. If the object is close to the target angle it will not overshoot (it will end up exactly at the target angle).

**Rotate toward position**

Rotate the object a number of degrees toward a position. If the object is close to facing the target angle it will not overshoot (it will end up exactly at the target angle).

**Set angle**

Set the object's angle in degrees, from 0 to 360. 0 degrees is facing right and angles increment clockwise.

**Set angle toward position**

Set the object's angle to face a position in the layout.

## Appearance

Not all objects support the actions relating to effects. For more information see Effects.

**Set blend mode**

Change the way the object blends with the background by altering the object's *Blend mode* property.

---

### Set color

Change the *Color* property of the object, which applies a tint. Use the `rgbEx(r, g, b)` expression to set the color. To restore the original color of the object set a white color (i.e. `rgbEx(100, 100, 100)`).

---

### Set effect enabled

Enable or disable one of the effects added to the object.

---

### Set effect parameter

Change the value of one of the parameters for an effect added to the object. 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.

---

### Set opacity

Set the object's opacity (or 'semitransparency'), from 0 (transparent) to 100 (opaque).

---

### Set visible

Set the object visible or invisible (hidden). The *Toggle* option sets the opposite state, i.e. visible if the object is invisible and vice versa.

## Hierarchy

These actions are available for plugins that support the *scene graph* feature, allowing objects to be connected together so they move, rotate and scale as if they were one large object.

---

### Add child

Add another object as a child of this object, which becomes its parent. The relative differences between the objects are remembered at the time the action is used. Then as the parent moves, scales and rotates, the child will move, scale and rotate with it. A series of checkboxes allows selection of which properties are to be updated. For example ticking only the *X* and *Y* options will keep the child at the same relative position, but not change its angle. The *Destroy with parent* option can also be enabled to automatically destroy the child if its parent is destroyed.

---

### Remove child

Remove a child from this object, disconnecting it from the parent and no longer updating it as the parent changes. The child still keeps its own children, if it has any.

---

**Remove from parent**

Remove this object from its parent if it has any, disconnecting it and no longer updating it as the parent changes. The object still keeps its own children, if it has any.

# HTML element

These actions are available for some plugins in the *Form controls* category, like Button and Text Input. These objects are HTML elements placed on top of the canvas.

---

**Set attribute**

**Remove attribute**

Add/set or remove an attribute on the HTML element for the form control. Attributes are part of the HTML standard and are documented on the Mozilla Developer Network.

> *Some attributes only need to exist to take effect and don't use a value. In this case, leave the value as an empty string.*

---

**Set CSS style**

Set a CSS (Cascading Style Sheets) style on the HTML element for the form control. CSS is part of the HTML standard and is documented on the Mozilla Developer Network.

> *Note that if the object has the Auto font size property enabled, any changes to the font-size property will be overridden.*

---

**Set enabled**

Enable or disable the control. When disabled, the control can no longer be interacted with. It also usually adjusts the appearance to indicate it is disabled, e.g. by greying out the control.

---

**Set focused**

**Set unfocused**

Focus or unfocus (also known as "blurring") the control. When focused, the control will receive keyboard input exclusively, and usually has an adjusted appearance to indicate active focus. When unfocused, the control will not respond unless directly clicked or touched, and the project will receive keyboard input instead.

### Set visible

Set whether the HTML element is visible or hidden. The visibility can also be toggled, to switch the visibility state to the opposite (e.g. hide if showing, or show if hidden).

## Instance variables

### Add to

### Subtract from

Modify a number instance variable.

### Set

Set a number or text instance variable.

### Set boolean

Set a boolean instance variable, which can hold either a true or false value.

### Toggle boolean

Toggle a boolean instance variable, which flips it from true to false or vice versa.

## Mesh distortion

These actions are available in objects that support the mesh distortion feature. This allows the object to be split in to a grid of points, and each point moved around individually to deform the appearance of the object. Moving mesh points also affects the object's collisions accordingly. See the mesh distortion examples in the Start Page for a demonstration.

### Set mesh size

Create a mesh using the given number of points horizontally (in columns) and vertically (in rows). At least 2 points must be used for both sizes for a mesh to be created. Use 0 for both to remove any existing mesh. Since the mesh is initialised with points in their default positions, no visual difference will be observed until a mesh point is altered.

### Set mesh point

Alter one of the points on the mesh given by its zero-based column and row number. In *Absolute* mode the position and texture locations are set to the given values; in *Relative* mode the given values are added to their current values. The position is given as normalized co-ordinates in the range [0, 1] across the object box, i.e. 0.5 being in the middle. This allows the mesh to scale proportionately with the object size. Mesh points can be positioned outside the object box. The texture location is also given in normalized co-ordinates in the range [0, 1], but cannot go outside that range. Changing the texture location alters where on

the source image corresponds to that mesh point, allowing for a different type of deformation. In absolute mode, texture positions of -1 leave the value unchanged. (In relative mode, use 0 to apply no change to the texture position, since -1 is a valid relative texture offset.) Optionally a Z elevation can also be specified for the mesh point, to move it in 3D. This works similarly to Z elevation for the entire object, but only applying to a single mesh point - see the *Set Z elevation* action for more details. The Z elevation is always interpreted as an absolute number regardless of the mode.

## Misc

### Destroy

Remove the object from the game.

### Set from JSON

Set the state of this object from a string of data in JSON format. This must come from a prior use of the *AsJSON* expression.

### Signal

Trigger *On signal* for the picked instances with the given tag. Any events paused with a *Wait for signal* action with the given tag will also be resumed once all the picked instances it is waiting on become signalled.

### Wait for signal

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

## Size & Position

### Move at angle

Move the object a number of pixels at a given angle in degrees.

### Move forward

Move the object a number of pixels forward at the object's current angle.

### Set width

### Set height

### Set size

Set the object's current size in pixels.

**Set X**

**Set Y**

**Set position**

Set the object's current position in the layout, in pixels. The origin (0,0) is the top-left of the layout and the Y axis increments downwards.

**Set position to another object**

Position the object at another object. It can also be positioned relative to an image point on the given object.

# Tags

**Change Tags**

Pass in an action, either *"add"* or *"remove"* and a string of space separated tags to add or remove from the instance.

# Z Order

**Move to bottom**

**Move to top**

Position the object either at the bottom or top of its current layer.

**Move to layer**

Move the object to the top of a given layer, either by its name or zero-based index. If the object is already on the given layer this action has no effect.

**Move to object**

Move the object next to another object in the Z order. You can choose to place the object to be placed in front or behind another object. If the target object is on a different layer, the object will also be moved to the target object's layer and then Z ordered next to it.

**Set Z elevation**

Set the object's elevation on the Z axis. By default the camera is at Z = 100, and looking down to Z = 0. The default Z elevation is 0. Increasing it will move it upwards (towards the camera) and decreasing it will move it downwards (away from the camera). You can learn more about Z elevation in the tutorial Using 3D features in Construct.

# COMMON EXPRESSIONS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/common-features/common-expressions

The following expressions are common to several plugins.

## Angle

### Angle

The object's current angle, in degrees. 0 degrees is facing right and angles increment clockwise.

## Appearance

### ColorValue

A number representing the color property of the instance. This can be used with the *Set color* action to set an object's color to match another object.

### Opacity

The object's current opacity (semitransparency), from 0 (transparent) to 100 (opaque).

## Hierarchy

These expressions are available for plugins that support the *scene graph* feature, allowing objects to be connected together so they move, rotate and scale as if they were one large object.

### ChildCount

Return the number of children currently attached to this object (with the *Add child* action) in the hierarchy. This is 0 if the object has no children.

### AllChildCount

Return the number of children currently attached to this object (with the *Add child* action) in the hierarchy, including all descendants. This is 0 if the object has no children

### ParentUID

The unique ID (UID) of this object's parent in the hierarchy. If this object has no parent, it returns -1.

# Misc

### AsJSON

Save the object state to a string of data in JSON format, and return it. This can be downloaded or otherwise stored, and later the state of the object restored using the *Set from JSON* action.

### Count

The number of instances of the object type.

### PickedCount

The number of instances meeting the event's conditions. For example, if the event has the condition "Mouse is over Sprite", *Sprite.PickedCount* will return the number of Sprite instances that the mouse is over.

### ObjectTypeName

The name of the object type for the given object. For example *Sprite.ObjectTypeName* will return "Sprite". When used as a family expression, this returns the name of the actual object type, never the name of the family itself.

### IID

Return the instance's index ID (IID). See instances.

### UID

Return the instance's unique ID (UID). See instances.

# Size & Position

### BBoxLeft

### BBoxRight

### BBoxTop

### BBoxBottom

Return the layout co-ordinates of the object's axis-aligned bounding box. This is the smallest unrotated box that completely encloses the object, taking in to account any rotation or stretching.

### BBoxMidX

### BBoxMidY

Return the layout co-ordinates of the mid-point of the object's axis-aligned bounding box. This is not necessarily the same position as the object origin, such as if the origin is not exactly in the middle.

---

**Width**

**Height**

Return the size of the object in pixels.

---

**ImagePointX(nameOrNumber)**

**ImagePointY(nameOrNumber)**

**ImagePointZ(nameOrNumber)**

Return the position of one of the object's image points from its currently displaying animation frame in layout co-ordinates. Either the image point's name or its number can be passed. Note that when using a number, 0 refers to the origin, so the first image point is number 1.

---

**X**

**Y**

Return the object's position in the layout, in pixels. The origin (0,0) is at the top-left of the layout and the Y axis increments downwards.

## Tags

---

**Tags**

Get all the tags of an instance as a string.

---

**TagsCount**

Get the amount of tags an instance has.

---

**TagAt**

Get The tag at a specific index.

## Time

---

**dt**

Return delta-time according to the object's own timescale. See Delta-time and framerate independence for more information.

## Z Order

**LayerName**

The name of the layer the instance is currently on.

**LayerNumber**

The zero-based index of the layer the instance is currently on.

**ZElevation**

Return the current elevation on the Z axis for the instance relative to its layer. This is not affected by the layer's Z elevation.

**TotalZElevation**

Return the instance's Z elevation added to the layer's Z elevation, providing the total Z elevation the instance appears at.

**ZIndex**

Get the zero-based index of the Z order of this instance within its current layer. 0 is the bottom instance, increasing up to the top instance.

## Template

**TemplateName**

The name of the template used to create this instance. Returns an empty string if no template was used.

# 3D CAMERA

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/3d-camera

In Construct the standard view is restricted to 2D and scrolling to an X and Y co-ordinate. The **3D Camera** object can change the viewpoint of the game to anywhere in 3D, such as positioning the camera to an X, Y and Z co-ordinate, looking towards another X, Y and Z co-ordinate. See the First-person shooter example for a demonstration of what the 3D Camera object can do.

You can learn more about 3D Camera and how it works with other 3D features in the tutorial Using 3D in Construct.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the I3DCameraObjectType script interface.

## Using a 3D Camera

By default Construct will use the standard 2D view, using the standard scrolling features to pan the view in 2D. The *Look at position*, *Look parallel to layout* and *Set position* actions enable a 3D view, which overrides the standard 2D view - the standard scrolling features will no longer have any effect on the 3D view. The standard 2D view can be restored using the *Restore 2D camera* action.

The 3D Camera object will only apply to layers which have their **Rendering mode** property set to **3D**. Layers using a **2D** rendering mode will ignore the 3D Camera and remain using a standard 2D view using the standard scrolling features. This is useful for things like a 2D interface displayed on top of a 3D view.

## 3D Audio

When moving the camera in 3D, it can be useful to play positioned sounds which will adjust playback to make them sound like they are coming from a position in 3D space. This can be done by setting the listener orientation in the Audio object, and playing positioned sounds either at objects (which will also use their Z elevation as the Z co-ordinate of the sound) or at a position which allows specifying the X, Y and Z co-ordinates of the sound.

## 3D Camera properties

The 3D Camera object has two properties: **Z scale** and **Default camera Z**. These are related and are influenced by the *Z axis scale* project property. The properties are defined as follows.

**Z scale**

The number of pixels per unit on the Z axis. With a *Regular* Z axis scale, this is always 1, as the Z axis uses the same scale as the other axes.

---

**Default camera Z**

The default position of the camera on the Z axis, producing 2D rendering at 100% scale on the layout. With a *Normalized* Z axis scale, this is always 100.

The relationship between the properties and the *Z axis scale* property is as follows:

- With the **Normalized** Z axis scale (the default), the *Default camera Z* is set to 100. Then the *Z scale* tells you how many pixels each unit on the Z axis is equivalent to.
- With the **Regular** Z axis scale, the *Z scale* is always 1, as the Z axis uses the same scale as the X and Y axes. The *Default camera Z* is then adjusted ensure 2D rendering appears at 100% scale. Note the *Field of view* project property also affects the *Default camera Z*.

The *Normalized* Z axis scale is designed for 2D projects using simple 3D features like Z elevation. If your project uses 3D camera to employ a fully 3D view, it will likely benefit from using the *Regular* Z axis scale.

## 3D Camera conditions

The 3D Camera object has no conditions.

## 3D Camera actions

---

**Look at position**

Set the position and orientation of the 3D Camera using a camera position, a position for the camera to look towards, and an up vector. The camera and look-at positions are given as 3D co-ordinates. The up vector is a 3D vector specifying which way is up, as this is also necessary to determine how to orient the camera towards the look-at position. The default up vector is (0, 1, 0), i.e. up is the positive direction on the Y axis, suitable for a top-down view.

---

**Look parallel to layout**

Set the position and orientation of the 3D Camera using a camera position and a camera angle in degrees. Whereas the default view is top-down, this sets a camera position looking along the layout, such that the layout appears as the floor at the bottom of the screen. This is a shortcut for using the *Look at position* action looking towards a 2D angle with an up vector of (0, 0, 1).

---

**Move along camera axis**

Move the camera position, the look position, or both, a distance along an axis relative to the current camera orientation. The distance can be negative, for example to move left when the specified axis is *Right*.

> *Note the scale on the Z axis can be different to the X and Y axes.*

---

**Move along layout axis**

Move the camera position, the look position, or both, a distance along an axis relative to the layout. The distance can be negative, for example to move left when the specified axis is *X*.

> *Note the scale on the Z axis can be different to the X and Y axes.*

---

**Restore 2D camera**

Restore the camera to its default 2D behavior, using the standard scrolling features to move the view.

---

**Rotate camera**

Moves the camera look-at position according to X and Y rotations in degrees. Note that a 3D Camera must first have been enabled using the *Look at position*, *Look parallel to layout* or *Set position* actions, since these also define the starting orientation that this action rotates around. Typically the rotation values will be provided by the Mouse object's *MovementX* and *MovementY* expressions in an *On movement* trigger to achieve "mouse look" - see First-person platformer for an example of this.

---

**Set field of view**

Set the angle of the field of view in degrees. Note this only has an effect when *Z Axis scale* is set to *Regular* in Project Properties. The default field of view is also set in Project Properties.

---

**Set position**

Set the camera position or the look position to a 3D co-ordinate. This can be used to control the camera or look positions independently without having to always specify both, such as to move the camera while using mouse look to control the look direction.

## 3D Camera expressions

---

**CameraX**

**CameraY**

**CameraZ**

Get the current 3D position of the camera.

---

**LookX**

**LookY**

**LookZ**

Get the current 3D position of the position the camera is pointing at.

---

**LookVectorX**

**LookVectorY**

**LookVectorZ**

Get the current vector of the direction the camera is pointing in, including camera rotation (i.e. changes applied with the *Rotate camera* action for purposes like mouse look).

---

**ZScale**

The number of pixels per unit on the Z axis. See *Z scale* under *3D Camera properties* for more details.

---

**DefaultCameraZ**

The default position of the camera on the Z axis, producing 2D rendering at 100% scale on the layout. See *Z scale* under *3D Camera properties* for more details.

---

**FieldOfView**

The current field of view in degrees. This is only applicable when *Z axis scale* is set to *Regular*.

---

**CanvasToLayerX(layer, x, y, layerZ)**

**CanvasToLayerY(layer, x, y, layerZ)**

Transform a position in canvas co-ordinates to layer co-ordinates on a Z plane given by *layerZ*. This is similar to the system expressions of the same name, but working in 3D.

---

**LayerToCanvasX(layer, x, y, z)**

**LayerToCanvasY(layer, x, y, z)**

Transform a position in 3D layer co-ordinates to 2D canvas co-ordinates. This is similar to the system expressions of the same name, but working in 3D.

---

**LayerToLayerX(fromLayer, toLayer, x, y, z)**

**LayerToLayerY(fromLayer, toLayer, x, y, z)**

Calculate the 2D position on a second layer (*toLayer*) that corresponds to a 3D position given on a first layer (*fromLayer*). This is similar to the system expressions of the same name, but working in 3D.

---

**ViewportBottomLeftX(layer)**

**ViewportBottomLeftY(layer)**

**ViewportBottomRightX(layer)**

**ViewportBottomRightY(layer)**

**ViewportTopLeftX(layer)**

**ViewportTopLeftY(layer)**

**ViewportTopRightX(layer)**

**ViewportTopRightY(layer)**

Return the X and Y position in layer co-ordinates of the four corners of the visible viewport, taking in to account the layer's Z elevation. These expressions are similar to the viewport system expressions, but when using a 3D camera the viewport area can be an irregular quadrilateral instead of a simple 2D rectangle, so these expressions provide four separate positions.

> *These expressions can return NaN (Not A Number) if a corner of the viewport does not intersect the layer plane.*

---

**CameraXRotation**

**CameraYRotation**

Return the X and Y rotation of the camera in degrees, as set by the *Rotate camera* action.

---

**ForwardX**

**ForwardY**

**ForwardZ**

Returns a 3D unit vector pointing in the direction of the camera.

> *Note this does not include camera rotation. Use the LookVectorX/Y/Z expressions to get the vector of the direction the camera is pointing in including camera rotation.*

---

**RightX**

**RightY**

**RightZ**

Returns a 3D unit vector pointing to the right of the camera, perpendicular to the forward vector.

---

**UpX**

**UpY**

**UpZ**

Returns a 3D unit vector for the camera up vector, which helps determine the camera orientation. Note this is recomputed from the given camera and look positions, so may not be exactly the same as the up vector given in the *Look at position* action.

# 3D SHAPE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/3d-shape

The **3D shape** object can add basic elements of 3D to your project, such as a 3D box. Six images can be drawn to cover each face of the shape.

Try out the 3D shape tour example for a visual demonstration of what the 3D shape object can do. The Start Page has a number of other examples under the *3D* tag.

You can learn more about 3D shape and how it works with other 3D features in the tutorial Using 3D in Construct.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the I3DShapeInstance script interface.

## Size and position on Z axis

Construct's normal 2D editing features cover the X and Y co-ordinates and size. To adjust the position on the Z axis, change the **Z elevation** property (which is the same property used to move other 2D objects like Sprite up and down on the Z axis). To adjust how tall the shape is (i.e. its depth), change the **Z height** property of the 3D shape object.

## Transparency

Due to the way 3D rendering works, transparency may not work as expected in 3D shape objects. To correctly render 3D features, Construct must use a depth buffer, but this only fully supports opaque surfaces. Therefore **3D shape objects will work best with opaque images.**

Making individual faces invisible works as expected, as this means the face skips drawing entirely, rather than drawing content that is assumed to be opaque.

For more information, refer to the section on transparency in the tutorial Using 3D features in Construct.

## 3D shape properties

**Images**

Click the *Edit* link to edit the six images the object uses for face images, one for each face of a cube. Not all shapes use all six images. Some images may also be cropped according to the shape of the face it appears on. For example the *Prism* shape has triangular end faces, so the images at these ends are cropped to only show a triangular portion of the image.

### Shape

Select from one of a few pre-defined shapes that the object will use. Not all shapes use all six images, since some have fewer than six faces. The available shapes are:

- **Box:** a six-sided cube or cuboid shape.
- **Prism:** a five-sided prism shape, like a tent. The front face is not used.
- **Wedge:** a five-sided shape like a cuboid cut in half diagonally, sloping up to the right. The front face is not used.
- **Pyramid:** a five-sided square-based pyramid, with the peak of the pyramid centered over the base. The front face is not used.
- **Corner (out):** a five-sided shape similar to a pyramid with the peak aligned to the upper-right corner. The front face is not used. The name derives from the fact this shape can join rows of *Wedge* shapes at an outside corner.
- **Corner (in):** a six-sided shape similar to a box where the front and back faces join in the bottom-left. The front face is used, but slopes down in the bottom-left half of the image. The name derives from the fact this shape can join rows of *Wedge* shapes at an inside corner.

### Z height

Set the height of the shape on the Z axis, i.e. its depth. Note this must be positive. If you wish to display the object lower down, change its *Z elevation* instead.

### Initially visibile

Set whether the object is shown (visible) or hidden (invisible) when the layout starts.

### Face visibility (Back/Front/Left/Right/Top/Bottom)

Set whether each of the six faces of the shape is initially visible.

> *The Back face is hidden by default, as normally it cannot be seen and so may as well skip drawing. If you make another face invisible, you may wish to make the back face visible again.*

### Back face culling

Automatically hides faces of the shape that are pointing away from the camera. This helps optimize performance if the shape is fully opaque, as those faces will never be seen anyway. If the shape is semitransparent (e.g. the opacity is less than 100%) then this mode can be useful to stop the faces pointing away from the camera from still being visible. Whether or not that is desirable depends on your art style.

---

**Use object image for faces (Back/Front/Left/Right/Top/Bottom)**

Optionally choose a Sprite, Tiled Background or 9-Patch object to display instead of the 3D shape's own face images for a given face of the shape. An instance of the object must be placed in the same layout for this to work. The properties of this instance can also be used to control the appearance of the face on the 3D shape. This allows using animated face images (via Sprite), or varieties of tiled/repeating images for face images (via Tiled Background and 9-Patch).

---

**Z tiling factor**

By default the camera appears at Z=100 and looks down to Z=0, meaning the camera is normally 100 units above the layout. However when using tiled images for 3D shape faces, such as Tiled Background or 9-Patch, this can result in unexpected tiling results. For example a 3D shape that has a Z height of 25 will display a Tiled Background as if it was 25 pixels tall, which may be too small for the displayed size of the object. The Z tiling factor is a multiple for the Z height to use when tiling images. For example the default of 8 means a Z height of 25 will actually tile as if it was 200 pixels tall, which usually produces better tiling results.

## 3D shape conditions

For conditions in common to other objects, see Common conditions.

---

**Compare shape**

Compare the current shape that is in use. This can be changed by the *Set shape* action.

---

**Compare Z height**

Compare the current Z height (i.e. depth) of the shape.

---

**Is face visible**

Check if one of the six faces of the shape is currently set to visible. Note this only checks whether the visibility is currently enabled, either in the object's properties or with the *Set face visible* action - it does not test whether the face is really showing on-screen.

## 3D shape actions

For actions common to other objects, see Common actions.

### Set back face culling

Change the *Back face culling* property. See above for more details.

### Set face image

Change one of the shape faces to use one of the other face images. For example this allows swapping the front face image for the back face image. To restore the original image, use the same face for both parameters, e.g. set back face to use image of back face.

> *This also undoes Set face object, restoring the 3D shape's own face image instead of another object's image.*

### Set face object

Replace the image used for a face of the shape with the image used by a Sprite, Tiled Background or 9-Patch object. An instance of the given object must exist on the current layout. See the property *Use object image for faces* for more information.

> *This action can be undone with Set face image.*

### Set face visible

Enable or disable the visibility of one of the faces of the shape. See the *Face visibility* properties for more information.

### Set shape

Change the shape currently used by the object. See the *Shape* property for more information.

### Set Z height

Change the Z height, i.e. depth, of the 3D shape. This must be greater or equal to 0. See the *Z height* property for more information.

### Set Z tiling factor

Change the multiple used for tiling images along the Z height of the object. See the *Z tiling factor* property for more information.

## 3D shape expressions

For expressions common to other objects, see common expressions.

### FaceImagePointCount(Face)

**FaceImagePointX(Face, ImagePoint)**

**FaceImagePointY(Face, ImagePoint)**

**FaceImagePointZ(Face, ImagePoint)**

Retrieve the 3D position of an image point on any of the 3D shape's faces. The face is a zero-based index of the face as shown in the image editor, i.e.:

- 0: Back
- 1: Front
- 2: Left
- 3: Right
- 4: Top
- 5: Bottom

The image point is specified by the name or number of the image point. When using a number, note that as per other image point expressions, the first image point is number 1, as 0 refers to the origin.

---

**ZHeight**

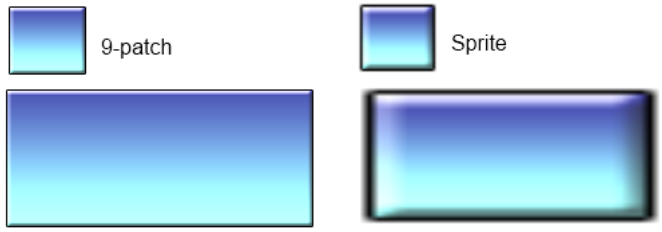
**ZTilingFactor**

Return the current Z height and Z tiling factor properties. See the documentation on the corresponding properties above for more information.

# 9-PATCH

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/9-patch

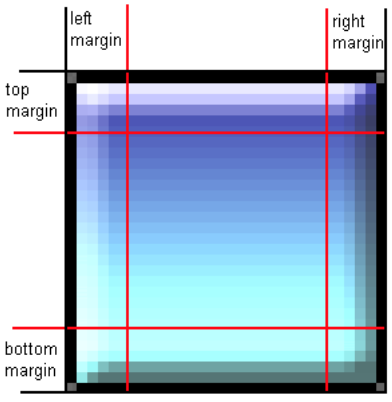
The **9-patch plugin** allows an image to be resized by stretching or tiling the edges, corners and fill separately. It is useful for conveniently making resizable windows or user interface elements that can take any size and still appear correctly. The image below demonstrates how a 9-patch looks compared to a Sprite object, which just stretches its entire image.




*Example of 9-patch scaling*

You can also click here to open an example of the 9-patch plugin.

The 9-patch object uses a single image, but automatically cuts it up in to nine sections using **margins** set in the object properties. By setting the left, top, right and bottom margins, the object then automatically works out the size and position of all nine sections.




*Sections in a 9-patch*

The edge and fill sections can be stretched or tiled independently, allowing for a better appearance when resized.

When resized very small, the bottom-right patches will be drawn on top of the top-left patches. This can be useful for representing things like progress bars with special artwork at the end of the bar. Note however that 9-patch objects do not support being mirrored or flipped, so the width and height should not be negative.

For an interactive demo of the 9-patch plugin, see the 9-patch example in the Example Browser.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the I9PatchInstance script interface.

## 9-patch properties

**Image**

Click the *Edit* link, or double-click the object in the Layout View, to edit the source image used for the 9-patch.

**Left margin**

**Right margin**

**Top margin**

**Bottom margin**

The margins of each side of the 9-patch, in pixels. See the image above for a visualisation of how these margins are used to determine the nine sections. Note these margins apply to the source image, ignoring the image scale properties.

**Edges**

Use *Stretch* to stretch each edge patch to the size of the object. Use *Tile* to repeat the edge patches instead.

**Fill**

Use *Stretch* to stretch the fill patch to the size of the object. Use *Tile* to repeat the fill patch inside the object instead, like a Tiled Background. Use *Transparent* if you don't want a fill image.

**Image scale X**

**Image scale Y**

Adjust the scale of the patches drawn inside the object. The margins are applied to the source (unscaled) image, but then the patches are drawn with the provided scale, increasing the visual size of the margins.

**Initial visibility**

Set whether the object is visible or invisible at the start of the layout.

**Origin**

Choose the location of the origin of the object relative to its bounding rectangle.

### Seams

To ensure seamless rendering under all circumstances, by default the patches internally overlap by 1 pixel (using the *Overlap* setting). However for semi-transparent patches this can cause a visible seam; in this case it is preferable to use the *Exact* setting instead.

## 9-patch conditions

### On image URL loaded

### On image URL failed to load

Triggered when *Load image from URL* finishes downloading the image and is ready to display it, or if the load fails.

## 9-patch actions

### Set edges

### Set fill

### Set image X scale

### Set image Y scale

### Set seams

Set the corresponding properties - see the section above for more details. Each of these settings can be changed on a per-instance basis.

### Set margins

Set the corresponding margin properties - see the section above for more details. By default all 9-patch instances share the same set of patch images, and so changing the margins will affect all instances, regardless of which instances were affected by the event's conditions. However the exception to this is after using the *Load image from URL* action, in which case the instance is using its own unique patch images, and the margins can then be changed for that instance only.

> *Be sure to wait for Load image from URL to finish using the Wait for previous actions to complete system action before setting margins. If you try to set margins without waiting for the image load to complete, then it will still be using the shared patch images, and setting the margins will still affect all instances.*

### Load image from URL

Load an image from a given URL. It is not shown until the image has finished downloading. Images loaded from different domains are subject to the same cross-domain restrictions as AJAX requests - for more information see the section on cross-domain in the AJAX object.

Data or blob URIs can also be passed as an image, e.g. from a canvas snapshot or camera image. Once this action has finished, the *Set margins* action can be used to set new margins for the loaded image, but note that to avoid affecting other instances, the image load must have first completed successfully (either triggering *On image URL loaded* or after using *Wait for previous actions to complete*).

## 9-patch expressions

**LeftMargin**

**RightMargin**

**TopMargin**

**BottomMargin**

Return the corresponding margin properties used by the instance.

> *See the notes about per-instance margins in the Set margins action.*

**ImageScaleX**

**ImageScaleY**

Return the current image scale properties as a percentage, with 100 representing 100% (normal) scale.

# ADVANCED RANDOM

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/advanced-random

The **Advanced Random** object provides expressions for advanced pseudo-random number generation (PRNG), including two- and three-dimensional noise functions like Perlin noise (referred to as "classic noise" in the plugin). These are useful for procedural generation, such as providing an endless supply of interesting and unique level designs.

It also provides seeded random functions, which provide the same pseudo-random number sequence when given the same seed. This can also be used to override the system random function (covering the *random()* expression, and any randomness used in behaviors) with a seeded random, which can provide deterministic random number generation for the whole runtime. By default the seed is always itself random, meaning random number generation is different between different runs of the game.

Click here to open an example of the Advanced Random plugin. This uses the Drawing Canvas object to display randomly generated textures using Advanced Random.

### Scripting

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

## Gradients

The Advanced Random object also provides tools to calculate custom gradients. This is useful for procedural generation. The random numbers it generates are in the range 0-1, but gradients allow you to specify values or colors for certain ranges of that. For example making the range 0-0.5 blue, and the range 0.5-1 green, allows you to look up the color using a random number in the range 0-1 and get either a water or land color. Combined with expressions like 2D perlin noise, this provides a way to randomly generate a level with mixed regions of water and land.

## Probability tables

Advanced Random can also create *probability tables*, which are a way of generating weighted random numbers. For example if you add three items with a weight of 1, and then a fourth item with a weight of 2, that item is twice as likely to be picked as any other item. The value of a probability table entry can be either a number or a string, but the weight must be a number. This is useful for doing things like random pickups with lots of common cheap items, but also some rare valuable items.

## Permutation tables

Another feature of Advanced Random is creating *Permutation tables*. These are simply a sequence of numbers that are randomly ordered. This is useful for retrieving random, non-repeating numbers. For example you could use the numbers 0-51 to represent a deck of cards, and create a permutation table to represent a shuffled deck of cards. Then if you read through the permutation table in order with the *Permutation* expression, it will return a random sequence of cards, like dealing from a shuffled deck.

## Advanced random properties

**Seed**

A string of characters used as the seed for random number generation. The same seed will always provide the same sequence of numbers. An empty string (the default) will use a random seed, ensuring the sequence of numbers is different every time.

**Replace system random**

If enabled, the system random function, which covers the *random* system expression as well as randomness in behaviors, is overridden to use the Advanced Random object's PRNG. Since the Advanced Random object can control the seed, this provides a way to seed the random number generation of the entire runtime.

## Advanced random conditions

The Advanced Random object has no conditions.

## Advanced random actions

**Set octaves**

Set the number of octaves used for coherent noise generation, from 1-16. The default is 1. This affects the Billow, Classic and Ridged expressions only. Using additional octaves adds layers of increasing detail to the noise functions, but is also slower to process.

**Update seed**

Set a new seed for random number generation, using a string.

> *Note if you pass an empty string, it still uses the empty string as the seed. If you want to go back to using a random seed, pass the RandomSeed expression as the seed to set.*

**Add gradient stop**

Adds a stop to the current gradient. Use after *Create gradient* to specify the gradient. The stop position can be any number, but is generally kept within the 0-1 range so it can be used with the random expressions. The stop value should be an expression of the form *rgbEx(r, g,*

*b)* or *rgba(r, g, b, a)* when the gradient uses color mode; otherwise it can be a simple number. The default gradient is a simple black to white gradient, using *rgbEx(0, 0, 0)* at position 0 and *rgbEx(100, 100, 100)* at position 1.

---

**Create gradient**

Create a new gradient. Multiple gradients can be managed by giving them different names. Creating a gradient also sets it as the current gradient, so this action can be immediately followed by *Add gradient stop* to specify the gradient. By default gradients work in color mode, which uses values based on the *rgbEx* or *rgba* expressions; however they can also be set to number mode which uses simple numbers.

---

**Set gradient**

Set the current gradient by its name. This allows switching between multiple gradients.

---

**Create permutation table**

Generate a randomly ordered sequence of numbers. *Length* is how many numbers to generate, and *Offset* is the first number in the sequence. For example a length of 3 with an offset of 1 will generate the numbers 1, 2 and 3, and then randomly shuffle them.

---

**Shuffle permutation table**

Re-shuffle an existing permutation table.

---

**Add probability entry**

Add an entry to the current probability table. The value can be a string or a number. The weight affects how likely the item is to be picked, relative to other item's weights.

---

**Create probability table**

Create a new probability table, using a string to identify it.

---

**Remove probability entry**

Remove an existing entry from the current probability table. If a weight of 0 is specified, the first entry with the given value is removed regardless of its weight. Otherwise an entry is only removed if it matches both the value and the weight.

---

**Set probability table**

Set the current probability table from which weighted random values are taken.

---

**Create probability table from JSON**

Create a new probability table from a JSON string. The input should be an array of `[weight: number, value: number|string]` tuples, e.g. `[[1, "Apple"], [2, "Banana"], [3, "Carrot"]]`.

## Advanced random expressions

---

**Billow2d(x, y)**

**Billow3d(x, y, z)**

Generate a random number using billow noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**Cellular2d(x, y)**

**Cellular3d(x, y, z)**

Generate a random number using cellular noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**Classic2d(x, y)**

**Classic3d(x, y, z)**

Generate a random number using classic (perlin) noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**RandomSeed**

Generate a random seed string that can be used to set the seed, restoring an unpredictable number sequence.

---

**Ridged2d(x, y)**

**Ridged3d(x, y, z)**

Generate a random number using ridged noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**Seed**

The currently set seed, as a string.

---

**Voronoi2d(x, y)**

**Voronoi3d(x, y, z)**

Generate a random number using Voronoi noise in the range 0-1, using either 2D or 3D co-ordinates.

**Gradient(Position)**

**GradientByName(Name, Position)**

Sample a gradient at the given position. This returns a color value for color mode gradients, otherwise a simple number. The *Gradient* variant refers to the current gradient, whereas the *GradientByName* variant refers to any gradient using a case-insensitive string of its name.

---

**Permutation**

Get a value at a zero-based index in the permutation table, from 0 (for the first item) up to but not including the length of the table.

---

**Weighted**

**WeightedByName(Name)**

Get a random value from a probability table. The relative likelihood of values is affected by their weight. The *Weighted* variant refers to the current probability table, whereas the *WeightedByName* variant refers to any probability table using a case-insensitive string of its name.

---

**ProbabilityTableAsJSON**

Get the current probability table as a JSON string. This can be read back using the **Create probability table from JSON** action.

# AJAX

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

The **AJAX plugin** allows you to fetch the content of a URL, or post data to a website. You can also use it to load project files. Its name derives from "Asynchronous JavaScript and XML", a technique familiar to most web developers.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Fetch API to make network requests.

## How to make a request

The basic usage of the AJAX object consists of:

1. Use the *Request* action to load a URL.
2. A moment later after the request completes, *On completed* triggers.
3. The *LastData* expression can be used to access the content of the response.

The *tokenat* system expression may be useful to split simple responses. Alternatively, you can read *LastData* in other formats by using other plugins, such as the XML object, loading Array data, and so on.

## Tags

A different tag can be provided for each request. This is a simple string you set to tell apart different requests. For example, on startup you may request both *foo.json* with tag *"foo"* and *bar.json* with tag *"bar"*. When the first request completes, *On "foo" completed* triggers; when the second request completes, *On "bar" completed* triggers. Requests can complete in a different order to the order they were made, so without tags it would be impossible to tell which request was completing.

## Making AJAX requests cross-domain or in preview

By default, browsers **block AJAX requests across domains**. This means, for example, a game on *construct.net* can request other pages on *construct.net*, but cannot request pages on *facebook.com*. This is an important security feature of web browsers (it is not specific to Construct or its AJAX object).

Also, when previewing in Construct the game runs on its own domain at *preview.construct.net*. Therefore AJAX requests to any other domain will typically fail during preview, unless the server explicitly allows cross-domain requests.

If you want AJAX requests to your server to work from any domain, or in preview, you can configure the server to send the following HTTP header:

```
Access-Control-Allow-Origin: *
```

This will enable AJAX requests from any domain, but you should still be aware of the possible security implications of this. You may need to ensure this is set for all HTTP methods used, including GET, POST, and also OPTIONS since cross-domain requests sometimes use that method for "preflighted" requests. For more information on cross-domain requests see HTTP access control (CORS) on MDN.

### Use HTTPS

Since *preview.construct.net* runs on a secure server (HTTPS), you cannot make AJAX requests in preview to insecure servers (HTTP). Browsers block this for security reasons. You may see warnings related to "mixed content", which refers to this problem.

Therefore for cross-domain AJAX requests to work in preview mode, you must also make sure your server is secure (using HTTPS). On the modern web this is best practice anyway, especially since many other features only work on secure servers.

## In NW.js

When exporting desktop applications with NW.js, the AJAX object can also load files from the application folder. Simply use the *Request URL* action and enter the name of a file in the same directory as the application, e.g. `"example.txt"`. Note if a project file exists with the same name, this will always load the project file instead.

## MIME types

AJAX requests for files on your own server requires that your server has the correct MIME types set up.

## Binary data

The AJAX object can receive resources as binary, and also post binary data, using the Binary Data object. This is also useful to fetch local resources like canvas snapshot URLs or video recording URLs, and load them in to a Binary Data object to do something else with them, like save it to storage or upload it to a server.

To request a resource that is received as a binary, use it as follows:

1. Use the *Set response binary* action to specify a Binary Data object to receive the next request's response.
2. Use the *Request* action to load a URL.
3. After the request completes and the response has finished downloading, *On completed* triggers.

4. Now the chosen Binary Data object has automatically been filled with the response data. (Note in this case the *LastData* expression is not used.)

## AJAX conditions

### On completed

Triggered when a request with the same tag has completed successfully. The *LastData* expression contains the response, unless the *Set response binary* action was used, in which case the selected Binary Data object now contains the response.

### On any completed

Triggered when any request has completed successfully. The *Tag* expression identifies the request, and *LastData* contains the response.

### On error

Triggered when a request with the same tag has failed. This can be for a number of reasons, such as the server being down or the request timing out. (The *LastData* expression is not set since there is no response.)

### On any error

Triggered when any request has failed. The *Tag* expression identifies the request.

### On progress

For long running downloads, *On progress* triggers periodically and updates the *Progress* expression with the state of the request. This is useful for making progress bars for things like large file downloads.

### On upload progress

For long running uploads, *On upload progress* triggers periodically and updates the *Progress* expression with the state of the request. This is useful for making progress bars for things like large POST data uploads.

## AJAX actions

### Override MIME type

In some cases you may wish to interpret the server's response with a different MIME type to the one the server indicates. For example a misconfigured server may return a text file with the wrong character set, and you want to force the response to be interpreted as UTF-8. In this case you could override the MIME type as `text/plain; charset=utf-8` to avoid garbling the text. This action only applies to the next AJAX request that is made, after which the

MIME type will be set back to the default setting of accepting what the server response indicates.

---

**Post to URL**

**Post binary to URL**

Send a request with data to a URL and retrieve the response. A tag is provided to match it up with the *On completed*, *On progress* and *On error* triggers. The *binary* variant can post the contents of a Binary Data object to the server; otherwise a string is used. Construct does not automatically URL encode the string - use the *URLEncode* system expression to ensure the data is in the correct format for posting. Note string data is in the same format as a query string, e.g. `"foo=1&bar=2"`. The method can also be specified: by default it is POST, but for some APIs you may need to change this to PUT, DELETE or another HTTP method.

---

**Request URL**

Send a GET request to retrieve the contents of a URL. A tag is provided to match it up with the *On completed*, *On progress* and *On error* triggers.

---

**Request project file**

Request the contents of a project file. A tag is provided to match it up with the *On completed*, *On progress* and *On error* triggers.

---

**Set request header**

Set a HTTP header on the next AJAX request that is made. After the next AJAX request all the headers set with this action are cleared again, so it only takes effect once.

---

**Set timeout**

Set the amount of time a request has to complete in seconds; if the timeout expires without the request completing successfully, it will instead fail and trigger *On error*. This action only affects subsequent requests, and does not affect any requests that have already started. If the timeout is set to -1 it restores the default browser timeout.

---

**Set with credentials**

Set the *with credentials* setting for the next AJAX request that is made. After the next AJAX request the setting will revert to its default (off), so it only takes effect once. When enabled, sending a request with credentials will cause cross-site requests to be made using credentials such as cookies and authorization headers. Internally this sets the `withCredentials` property of XMLHttpRequest. More details can be found at the MDN withCredentials documentation.

---

**Set response binary**

Use this action before a *Request* action to read the response in to a Binary Data object instead of returning it as a string in the *LastData* expression. This allows for non-text resources like images to be fetched and processed directly.

## AJAX expressions

**LastData**

The contents of the last response. This is set in the *On completed* trigger. Note if *Set response binary* was used, the response is in the chosen Binary Data object instead, and this expression will return an empty string.

**LastStatusCode**

The HTTP status code of the last response, e.g. 200 for OK or 404 for Not Found. This is set in the *On completed* trigger.

> *You can find a complete list of possible status codes and what they mean on the MDN page HTTP response status codes.*

**Progress**

Return the progress of the AJAX request in an *On progress* event. The progress is represented as a number from 0 to 1, e.g. 0.5 for half completed.

**Tag**

The tag of the AJAX request in a trigger. This is useful to identify requests in *On any completed* or *On any error*.

# ARRAY

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

The **Array** object stores lists of values (numbers or text). It is analogous to arrays in traditional programming languages.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IArrayInstance script interface. (JavaScript and TypeScript have built-in support for arrays, but this allows interacting with an array used in event sheets.)

## About Arrays

Array supports up to three dimensions. For example, a simple list of ten values would be a 10 x 1 x 1 array. Note that you should not set a size of 0 on any of the dimensions else the *entire* array becomes empty; it is correct to have a size of 1 on unused dimensions.

Each element of an array can store a number or some text. The number or text in an element can be changed with the *Set* actions, and accessed with the *At* expression. For example, a 10 x 10 x 1 array is analogous to a 2D grid with a total of 100 values. A number could be stored at the position (3, 7) with the action *Set at XY*, and accessed with `Array.At(3, 7)`. Note like the rest of Construct indices are zero-based, so the first element is at 0. In this example, `Array.At(0, 0)` would return the first number in the grid.

Array can store either text or a number in any of the elements. Numbers and text can also be mixed within an array.

Arrays do not automatically resize. If you access a value outside the array bounds, it returns the number 0. If you set a value outside the array bounds, it will have no effect.

## Designing arrays

You can use Construct's Array Editor Paid plans only to set the initial contents of an array. You can create a new array data file as a project file from the Project Bar. At runtime you can load the project file with the AJAX object and use the Array's *Load* action to read the data file from the AJAX's *LastData* expression.

## Manipulating arrays

A one-dimensional array, sized N x 1 x 1, serves as a simple list of N values. The actions in the *Manipulation* category (e.g. *Push*, *Pop*) allow one-dimensional arrays to be used like other data structures. (These actions work with multidimensional arrays, but are intended for the one-dimensional case.)

For example, the following scheme implements a queue (first in first out, or 'FIFO'):

- Add new items with **Push front**
- Retrieve the next value with `Array.Back`
- Remove the retrieved value with **Pop back**

The following scheme implements a stack (last in first out, or 'LIFO'):

- Add new items with **Push back**
- Retrieve the next value with `Array.Back`
- Remove the retrieved value with **Pop back**

## Array properties

**Width (X dimension)**

**Height (Y dimension)**

**Depth (Z dimension)**

The size of the array. If you want a one-dimensional array (i.e. a list of values), use A x 1 x 1. If you want a two-dimensional array (i.e. a grid of values) use A x B x 1.

**Elements Read-only**

This property indicates the total number of elements in the array. If it is 0, the array is completely empty and is unable to store any data. A common mistake is to set the Y or Z axis sizes to 0 which causes the entire array to be empty; this property helps you identify this mistake. Also using a huge array can cause very high memory use, and the element count helps you identify this case as well.

## Array conditions

**Compare at X**

**Compare at XY**

**Compare at XYZ**

Compare a value at a position in the array. Indices are zero-based. All values outside the array return the number 0. If *Compare at X* is used, the Y and Z indices are 0. If *Compare at XY* is used, the Z index is 0.

**Compare size**

Compare the size of one of the array dimensions, which is the number of elements on that axis.

**For each element**

A repeating condition that runs once for each element in the array. This therefore runs *width x height x depth* times.

**Compare current value**

Only valid in a *For each element* loop, either as a following condition or in a sub-event. This compares the current value being iterated in the loop.

**Contains value**

Searches the entire array to check if any of the elements contains the given value. For example, you can use this to test if the string *"sword"* is stored anywhere in the array.

**Is empty**

Test if the array is empty. The array is empty when the total number of elements is zero, calculated as *width x height x depth*. Therefore the array is empty when any axis has a size of zero. This can be useful when using Array as a data structure (e.g. when pushing and popping values).

## Array actions

**Clear**

Set every element in the array to the given value, which by default is the number 0.

**Set at X**

**Set at XY**

**Set at XYZ**

Write a value at a position in the array. Indices are zero-based. Writing to values outside the array has no effect. If *Set at X* is used, the Y and Z indices are 0. If *Set at XY* is used, the Z index is 0.

**Set size**

Set the dimensions of the array. Values are preserved, but if the new array is smaller it is truncated. If the new array is larger, new elements are set to store the number 0. If any of the dimensions are 0 the entire array is empty, so usually all the dimensions are at least 1.

**Download**

Invokes a browser download of a file containing the Array's contents in JSON format.

### Load

Load the contents of the array from a string in JSON format. This can be retrieved from either the *Download* action, the *AsJSON* expression, or the AJAX object loading a project file.

> *Note this does not allow loading arbitrary JSON data - the data must be in a special format for Construct. If you want to load data from a project file, create it using the New - Array menu option in the Project Bar.*

### Push

Add a new value either to the beginning (front) or end (back) of an axis. Since the Array is a 3D cube of values, technically this inserts a new 2D plane of elements all with the given value. However in 1D arrays this adds a single element, and in 2D arrays it inserts a new row of elements.

### Pop

Delete the value at either the beginning (front) or end (back) of an axis. Since the Array is a 3D cube of values, technically this removes a 2D plane of elements. However in 1D arrays this removes a single element, and in 2D arrays it removes a whole row of elements.

### Insert

Insert a new value at a specific index on an axis. Since the Array is a 3D cube of values, technically this inserts a new 2D plane of elements all with the given value. However in 1D arrays this adds a single element, and in 2D arrays it inserts a new row of elements.

### Delete

Delete the value at a specific index on an axis. Since the Array is a 3D cube of values, technically this removes a 2D plane of elements. However in 1D arrays this removes a single element, and in 2D arrays it removes a whole row of elements.

### Reverse

Reverse the order of elements on an axis. Note that in multidimensional arrays this only reverses one axis. For example reversing the X axis in a 2D array will reverse the order of the columns while preserving the contents of each column.

### Shuffle

Sort elements along a given axis into a random order. When using 2D or 3D arrays, then one-dimensional arrays are independently shuffled depending on the chosen axis. For example when choosing the X axis, every row is independently shuffled; when choosing the Y axis, every column is independently shuffled.

### Sort

Sorts the order of elements on an axis in ascending order. There are a variety of ways to sort multidimensional arrays. The options are:

- **X axis (by column):** sort the X axis by the first row of elements, moving entire columns when rearranging the ordering. Where elements match in the first row, it will refer to further elements down the Y axis as tie breakers.
- **X axis (separately):** sort the X axis independently in every row.
- **Y axis (by row):** sort the Y axis by the first column of elements, moving entire rows when rearranging the ordering. Where elements match in the first column, it will refer to further elements down the X axis as tie breakers.
- **Y axis (separately):** sort the Y axis independently in every column.
- **Z axis:** sort the Z axis independently in every 2D (X/Y) position in the array.

### Split string

Sets the array to a one-dimensional list of items based on splitting a string by a certain character. For example splitting the string `"1,2,3"` with the separator `","` and type *Auto* will set the array size to 3 x 1 x 1 with the numbers *1*, *2* and *3*. The *Type* parameter determines whether values are read as strings or numbers. The default mode *Auto* will set values as numbers if the token looks like a number, and a string if not. Setting the type to *String* or *Number* will ensure all values are consistently set as the given data type. The array can be converted back to a string with the *JoinString* expression.

> *Splitting string can work for simple cases but has limitations. For example it's not possible for the separator to appear inside values, and it is not possible to explicitly specify the data type of values. To handle more complex cases, use a more robust data format like JSON.*

## Array expressions

### At(x)

### At(x, y)

### At(x, y, z)

Retrieve a value at a position in the array. Indices are zero-based. Reading values outside the array returns the number 0. If the Y or Z indices are not provided then 0 is used.

**CurX**

**CurY**

**CurZ**

The current zero-based index for each dimension in a *For each element* loop.

---

**CurValue**

The current value in a *For each element* loop. This is a shortcut for `Array.At(Array.CurX, Array.CurY, Array.CurZ)`.

---

**Width**

**Height**

**Depth**

Return the size of each of the array's dimensions.

---

**Front**

Shortcut to access the first value in the array, which is the same as `At(0, 0, 0)`.

---

**Back**

Shortcut to access the last value on the X axis, which is the same as `At(Self.Width - 1, 0, 0)`.

---

**IndexOf**

**LastIndexOf**

Searches the array X axis for a given value and returns the index it is found at, or -1 if not found. *IndexOf* finds the first matching element, and *LastIndexOf* finds the last matching element.

---

**AsJSON**

Return the contents of the array as a string in JSON format. This can later be loaded in to the array with the *Load* action.

---

**JoinString(separator)**

Convert a one-dimensional array to a string by converting every value to a string and joining them together with a separator. For example if an array is sized 3 x 1 x 1 and stores the values *1*, *2* and *3*, then the expression `Array.JoinString(",")` will return the string `1,2,3`. This is effectively the reverse of the *Split string* action.

# AUDIO

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

The **Audio** object plays back audio files that have been imported to the project. The Audio object must be added to each project that needs to use audio playback. Audio files can be imported to a project by right-clicking the **Sounds** or **Music** folders in the Project Bar and selecting Import sounds or Import Music, which brings up the Import Audio dialog. See the relevant manual sections for more information on the steps involved to import audio files.

Construct comes with several examples of the Audio plugin; search for *Audio* in the Example Browser to find them.

### Scripting

When using JavaScript or TypeScript coding, this object can be accessed via the IAudioObjectType script interface. This mainly provides access to the underlying Web Audio API context.

## Categorise audio files correctly

It is important to organise audio files appropriately, because audio files in the **Sounds** folder are downloaded completely before playing, but files in the **Music** folder are streamed. This means if a Music track is accidentally put in the Sounds folder, it would have to download completely before it started playing, which could take a while. However, audio in the Music folder can start playing immediately since it is streamed from the server.

## Preloading sounds

The **Preload sounds** project property determines whether sounds (excluding music) are loaded while the loading bar is showing. Preloading sounds means there is more to load before the project can start, and memory use is higher due to having all sounds loaded, but all sound effects can play immediately once it starts. If sounds are not preloaded, the project starts sooner since it did not need to load any sounds, but sounds will be loaded on-demand during the project. In other words, nothing is loaded until a *Play* action starts playing an audio file. Then it starts loading and will play when ready. This also helps minimise memory usage since unused audio is never loaded. However, it can introduce a delay before audio plays for the first time. The delay on first play is a one-off, because after the first play the sound is already loaded and can be played immediately if played a second time.

If *Preload sounds* is disabled, the *Preload* action can be used to start loading an audio file without actually playing it. This can be done on *Start of layout* to start downloading a few important sound effects so there is no delay when they are played for the first time. The *Unload* actions can also be used to remove a loaded sound from memory. This allows you to manually

manage which sounds are loaded in to memory, which is important if you have a large library of sound effects which would use a lot of memory if they were all preloaded.

Music is never preloaded, since music tracks often involve a large download size, and it is not usually important to have music play with as little latency as possible. Music will still stream while playing, but if the latency is important, the *Preload* action can be used to load it in advance of playing.

## Audio tags

Some actions affect audio parameters such as the volume for sounds which are already playing. However there can often be many sounds playing at once in a project. In order to identify which sounds you want to affect, sounds are played with an associated *tag*. This is any string that identifies the sound. For example, the player's weapon sound effect could be played with the tag *"PlayerWeapon"* and an enemy's weapon with the tag *"EnemyWeapon"*. Then, the tag can be used in the *Set Volume* action to specify which sound to set the volume for. Tags are case insensitive.

Multiple sounds can also play at once using the same tag. In this case actions like *Set Volume* affect all the sounds playing with that tag.

A tag which is an empty string ("") has a special meaning: it refers only to the last sound played with one of the *Play* actions. This is convenient for playing a sound and immediately setting its volume and other parameters, without having to assign it a unique tag.

### Multiple tags

Providing the Audio object property *Enable multiple tags* is enabled (it is on by default), then the tag string can include multiple space-separated tags, such as "player weapon" and "enemy weapon". Then when changing sounds such as to set the volume, all sounds with all the provided tags will be updated. For example setting the volume for tag "weapon" will update sounds played with tags "player weapon" and "enemy weapon", and setting the volume for tags "player", "player weapon" or "weapon player" will update sounds played with the tags "player weapon".

There is one exception to using multiple tags, which is when using effects. Adding effects builds up an effect chain which audio playback can be routed through. However a sound can only be routed through one effect chain. Therefore when adding effects with multiple tags, the effect will be added to multiple effect chains - once for each provided tag. Then when playing sounds with multiple tags, the first tag determines which effect chain the sound is routed to.

If *Enable multiple tags* is disabled, then a tag with spaces is still treated as a single tag. This only exists for backwards compatibility with projects made before the multiple tags feature was introduced.

## Autoplay restrictions

Most modern browsers have a limitation in starting audio playback. To avoid annoying users generally browsing the web, audio playback cannot start until the user interacts with the page, such as touching or clicking it. This is a limitation in the browsers themselves and cannot be

worked around. As a result, if you play audio on the start of layout, you may find it does not actually start until the first user interaction.

Usually you do not need to handle this in your events. If audio cannot be played for any reason, the Audio object will automatically queue it up for playback at the soonest opportunity (usually when the user next clicks or touches). However you should be aware of this when designing your project. For example if the first touch changes layout or stops the music, then the music may never be heard. You may want to start playback then encourage the user to touch the screen with a 'Play' icon or something similar.

The limitation is specific to web browsers. It does not apply if you publish a mobile app. Further, browsers may lift the limitation in some circumstances, such as if you use the *Install* or *Add to home screen* options, or if they identify over time that your web page is regularly used for audio playback that the user wants.

## Audio properties

**Timescale audio**

The project timescale can be used to speed up or slow down playback of the project, for effects like slow-motion. See Delta-time and framerate independence for more information. This property controls whether or not audio is affected by the project's timescale.

- **Off** will play back audio the same regardless of the timescale.
- **On (sounds only)** will play back audio from the Sounds project folder at a different rate depending on the timescale, but will always play back audio from the Music project folder at the same rate.
- **On (sounds and music)** will play back all audio at a different rate depending on the timescale.
- Some browsers may not support audio timescaling at all; test on multiple browsers to establish support.

**Save/load**

When using savegames, what audio state should be saved and restored.

- **All** will save and restore all audio, so that music and sound effects rewind to the same point they were saved at when loading.
- **Sounds only** will only restore the sound effects playing at the time of the save, and allow music to keep playing through unaffected when loading a game.

- **Music only** conversely only restores the music playing at the time of the save, and allows sound effects to keep playing through unaffected when loading a game.

- **None** does not save or load any audio state at all. Audio will be completely unaffected when loading a game, and any playing music and sound effects will continue to play out to their end.

---

**Play in background**

If disabled, then switching browser tab, minimising the browser window, switching to a different mobile app, or otherwise hiding the window will pause all audio and resume it when switching back. This is intended to avoid annoying the user with continued music playback when deciding to do something else, and also helps save battery on mobile. However for some types of app such as music players it may be desirable to keep music playing in these cases, in which case enabling the property allows continued audio playback even when in the background.

---

**Latency hint**

Provide a hint to the audio engine about the preferred latency vs. power usage tradeoff. Typically interactive content like games will prefer a low latency mode, but other uses like music playback where latency is not important may prefer to use a more battery-efficient mode. The options are:

- **Interactive** (default): provides the lowest playback latency, but uses more battery power.

- **Balanced**: a middle-ground providing medium playback latency with medium power use.

- **Playback**: provides the highest playback latency, with the lowest power use, suitable for purposes like music playback where the latency is not important.

---

**Enable multiple tags**

If checked, multiple tags can be specified for sounds by separating them with spaces, e.g. "player weapon" would count as two separate tags "player" and "weapon". If disabled then sounds can only have one tag, i.e. "player weapon" would be counted as a single tag name including the space. This setting is enabled by default and only exists for backwards-compatibility with projects made before the introduction of the multiple tags feature. For more information see the section *Multiple tags* above.

---

**Panning model**

How positioned sounds are panned. *HRTF* uses a realistic model of human hearing, whereas *equal power* is a simple method that preserves the overall power in a stereo channel.

**Distance model**

The formula to determine volume reduction of positioned sounds relative to the distance to the listener. The options are:

- **Linear**, using the equation *1 - rolloffFactor * (distance - refDistance) / (maxDistance - refDistance)*
- **Inverse**, using the equation *refDistance / (refDistance + rolloffFactor * (distance - refDistance))*
- **Exponential**, using the equation *pow(distance / refDistance, -rolloffFactor)*

**Listener Z height**

The height of the listener above the layout, in layout pixels, used to determine relative volume and panning of positioned sounds. A low Z height will have intense changes over small distances, whereas a high Z height will have smaller changes over larger distances.

**Reference distance**

The distance at which the volume of positioned sounds begins to reduce. For best results this should be at least as much as the *Listener Z height*.

**Maximum distance**

The maximum distance in pixels beyond which positioned sounds no longer reduce their volume.

**Roll-off factor**

How quickly the volume reduces as positioned sounds move away from the listener. A high roll-off factor means sounds get quieter quickly, whereas a low roll-off factor means sounds will not lose much volume.

## Audio conditions

**Is any playing**

True if any audio is currently playing.

**Is silent**

True if the object has been set in to silent mode using the *Set silent* action.

**Is tag playing**

True if any audio with a given tag is currently playing.

---

**On ended**

Triggered when a sound with a given tag finishes playing. This is not triggered for looping sounds.

---

**On fade ended**

Triggered when a fade started by the *Fade volume* action finishes, for a given tag.

---

**Preloads complete**

True when all audio preloaded with one of the preload actions has finished loading.

## Audio general actions

---

**Play**

**Play (by name)**

Start playing an audio file with a given tag. The latter action gives you the opportunity to use an expression for the audio file name. The sound can optionally be set to looping when it starts playing. A volume can also be set, given in decibels (dB). A volume of 0 dB is original volume, and below 0 dB attenuates the sound. Note amplification is not supported. For example, entering a value of -10 plays the audio back 10 dB quieter (about half as loud). A stereo pan can also be provided, ranging from -100 (fully left) to 100 (fully right), with the default 0 being middle. Prefer setting the intended volume and pan in the *Play* action; even if followed immediately by a *Set volume* or *Set stereo pan* action, some platforms will momentarily play the audio at the volume and pan given in the *Play* action.

---

**Preload**

**Preload (by name)**

Start loading an audio file so it has no delay before playing. See the section *Preloading sounds* above for more information. Audio does not have to be preloaded before playing - it is optional and only serves to possibly reduce the delay before audio plays for the first time. Once all audio preloaded with this action finishes loading, the *On preloads complete* trigger fires. Note if the project *Preload sounds* property is enabled, there is no point preloading any sounds, since they will always be loaded before the project starts - in this case it only makes sense to preload music.

---

**Seek to**

Seek a currently playing sound to a different location in the audio file. The time to seek to is given in seconds.

### Set looping

Set a sound either looping (repeating when it finishes) or not looping (stopping when it finishes).

### Set master volume

Set the overall volume that is applied to all audio playback.

### Set muted

Set a sound either muted (silent) or unmuted (audible).

### Set paused

Pause or resume some audio by its tag.

### Set playback rate

Change the rate a sound plays back at. If the *Timescale audio* property is used, it combines with the playback rate set by this action.

### Set silent

Enable, disable or toggle *Silent mode*. In silent mode all currently playing sounds are muted and no new sounds will play. This is useful for quickly creating an audio toggle on a title screen.

### Set volume

Change the volume of a sound. The volume is given in decibels (dB). A volume of 0 dB is original volume, and below 0 dB attenuates the sound. Note amplification is not supported. For example, entering a value of -10 plays the audio back 10 dB quieter (about half as loud). Note it is best to set the initial volume in the *Play* action instead of setting it with this action immediately after playing, since that can sometimes cause a moment of playback at the wrong volume.

### Fade volume

Change the volume of a sound over time. This is typically used for fade-in and fade-out effects. The sound will fade from its current volume to the given level in decibels (dB), over the given time period in seconds. When the fade finishes, the sound can either automatically be stopped, or keep playing. For a fade-in, typically the sound will be initially played at a low volume (e.g. -100 dB) and faded in to a high volume (e.g. 0 dB), and keep playing when the fade finishes. For a fade-out, typically the sound will be playing at a high volume (e.g. 0 dB) and faded to a low volume (e.g. -100 dB), and then stopped when the fade finishes, so there

isn't an inaudible sound still playing. Note if *Set volume* is used during a fade, the fade is cancelled and the sound will be left at the volume specified by *Set volume*.

---

**Stop**

Stop a sound playing immediately.

---

**Stop all**

Stop all currently playing sounds.

## Audio effects actions

A selection of well-known audio effects can be added using the *Add effect* actions. Each tag has its own effect chain, and multiple effects can be added to a tag. All audio played with the given tag is then processed by the effect chain. This can be used to create environmental effects and other creative audio features. Audio signal processing is a complex topic and somewhat out of the scope of this manual, so it will not be detailed exhaustively here. Anyone with light experience in audio recording or production should already be familiar with all the effects available. For interactive examples, search for *Audio* in the Example Browser. A brief summary of each effect is provided below.

- **Analyser:** doesn't change the audio, but can report back frequency domain data
- **Compressor:** automatically boost or reduce volume to even out the overall volume level
- **Convolution:** an advanced effect using another sound as an impulse response to process the audio. This allows for real-world locations to be recorded and the environmental reverb applied
- **Delay:** a feedback loop with a delay, making a sort of simple echo effect.
- **Distortion:** a guitar-amplifier style signal distortion
- **Filter:** boost or reduce certain frequencies, such as a low-pass filter (which cuts out high frequencies). Useful for simple atmospherics, treble/bass adjustment, etc.
- **Flanger:** delays the sound by a few milliseconds then mixes it back in with itself. By oscillating the delay time a sweeping effect is created
- **Gain:** a volume control, which might be useful in longer effect chains. The **Mute** effect is also simply a zero gain effect, which can be useful to add after analysers (so the audio is analysed, but not heard).
- **Phaser:** phase-shifts the sound then mixes it back in with itself. By oscillating the phase shift another sweeping effect is created
- **Stereo pan:** a stereo pan control, allowing an entire group of sounds to be panned left or right together.
- **Tremolo:** automatically oscillates the volume up and down, also known as amplitude modulation. Some interesting amplitude modulation effects can be created by moving the

modulation frequency in to the audible range (above 20 Hz).

- **Ring modulator:** like tremolo, but oscillates all the way through to a full phase inversion

The **Remove all effects** action clears a tag's effects chain, allowing you to add a different selection of effects. The **Set effect parameter** action also allows effect parameters to by dynamically set or faded during playback. Each effect also has a wet/dry mix which can be used to fade in and fade out effects.

## Audio memory actions

**Unload all audio**

Release all loaded audio files from memory. Any subsequently played audio will need to be loaded again.

**Unload audio**

**Unload audio (by name)**

Release a specific loaded audio file from memory. If the audio file is not loaded, this has no effect. Any subsequent playback of the audio file will need to load it again. This allows manual control of which audio files are in memory.

## Audio panning/positioning actions

**Play at object**

**Play at object (by name)**

As per the ordinary *Play* actions, but the sound is positioned at an object. If the object moves (including changing Z elevation) or rotates during playback, the sound follows with it. A cone can be specified to create directional sounds, which follows the object's angle.

**Play at position**

**Play at position (by name)**

As per the *Play at object* action, but the sound does not move. It is just played at a fixed position and angle in the layout. A Z co-ordinate can also be specified to play a sound at a given Z elevation, or for full 3D audio support when used with *Set listener orientation*.

**Set listener object**

Set the object that positioned sounds are calculated relative to. Typically this is set to the object representing the player on *Start of Layout*.

**Set listener orientation**

By default the listener is oriented for a 2D game. However when using 3D Camera to move the view in 3D, it is also useful to change the listener orientation to match that of the camera for positioned sounds to play back correctly in 3D space. The listener orientation is specified using two vectors: a forwards vector (the direction the listener is facing in), and an up vector (which determines the orientation along the forwards vector). These can usually be set to the *LookVectorX/Y/Z* and *UpX/Y/Z* expressions of the 3D Camera object.

### Set listener Z

Set the *Listener Z* property of the audio object, which affects the calculation of positioned sounds.

### Set stereo pan

Change the stereo pan of a sound. The pan is a number ranging from -100 (fully left) to 100 (fully right), with 0 being middle. Note it is best to set the initial pan in the *Play* action instead of setting it with this action immediately after playing, since that can sometimes cause a moment of playback with the wrong pan.

> *Setting the stereo pan of a positioned sound will turn off positioning in order to apply the stereo pan.*

## Audio scheduling actions

### Schedule next play

This action causes the next *Play* action (all variants) to be delayed until the specified time. The delayed playback is sample-accurate. Typically events are only run every 16ms making it difficult to schedule sounds more accurately than that, but this action allows for perfectly scheduled playback, even in between ticks. If the specified time is in the past, it will play immediately. The time given must be relative to the audio hardware clock, which is returned by the *CurrentTime* expression, so typically sounds will be scheduled a short time ahead using an expression of the form *Audio.CurrentTime + N*. See the *Audio scheduling* example in the Start Page for a demonstration.

> *Note that audio scheduling works best if you set the Use worker project property to No. This is because worker mode does not have direct access to the audio clock, which adds a small amount of latency and variance to playback which may be noticeable when scheduling.*

## Audio web actions

### Add remote URL

Play audio from a URL by adding it first with this action. Specify the URL to be played, the type (aka the MIME type) of the audio to be played, and pick a name to use for this URL. On its own this action does nothing - no request will be made to the given URL. However you can then pass your chosen name to the other actions that work "by name", such as *Play (by name)* or *Preload (by name)*, it will then play from the URL you associated with that name. If these actions ask for the folder, it is not really used, but chooses whether the audio is fully downloaded and decoded before playback (for *Sounds*), or streamed (for *Music*). As with any other audio file, you can use the *Preload (by name)* action to preload the audio at the given URL, to ensure subsequent playback starts promptly.

> *WebM Opus is the only audio format that works across all browsers and platforms. To ensure your audio works everywhere, make sure the URL serves WebM Opus audio. Otherwise whether or not audio playback works will depend on the audio codec support of the current browser/platform.*

## Audio expressions

### AnalyserFreqBinAt(Tag, Index, Bin)

Get the magnitude of energy in an analyser's frequency bin. AA analyser effect must already be added to a tag. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on). *Bin* is the frequency bin number to retrieve from, up to *AnalyserFreqBinCount*.

### AnalyserFreqBinCount(Tag, Index)

Get the number of frequency bins returned by an analyser. An analyser effect must already be added to a tag. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on).

### AnalyserPeakLevel(Tag, Index)

Get the peak level of audio in the last FFT window from an analyser. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on). The value is returned in dBFS (0 dB for peak level, and negative values for lower). If you intend to use this value it is recommended to use an FFT size of 1024, because at a system sample rate of 44.1 KHz the value will update about 43 times a second. Projects usually run at 60 FPS, and smaller FFT sizes may cause FFT windows to be missed since they change faster than the framerate.

### AnalyserRMSLevel(Tag, Index)

Get the RMS level of audio in the last FFT window from an analyser (the square root of the average of the squared sample values). *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on).

The value is returned in dBFS (0 dB for peak level, and negative values for lower). If you intend to use this value it is recommended to use an FFT size of 1024, because at a system sample rate of 44.1 KHz the value will update about 43 times a second. Projects usually run at 60 FPS, and smaller FFT sizes may cause FFT windows to be missed since they change faster than the framerate.

---

**EffectCount(Tag)**

Get the number of effects in the effect chain for a tag.

---

**CurrentTime**

Get the audio clock time in seconds. Where supported, this is returned from the audio hardware, providing the correct time against which to schedule audio playback. It is important to use this value to calculate playback times in the *Schedule next play* action.

---

**Duration(Tag)**

Get the duration in seconds of an audio sample with a tag.

> *This expression will only return the correct value once the audio has finished loading and decoding. If you need to access the value immediately after starting playback, consider first preloading the sound.*

---

**NormalizedVolume(Volume, dbThreshold)**

This is a helper expression to convert a volume expressed in the range 0-100 (such as with a slider bar in a user interface) to a dB value suitable for actions such as *Set volume*. The *dbThreshold* parameter is the dB that will be set for a volume value of 10 (e.g. -40). The returned dB volume will start at 0 for volume value 100 and linearly drop to *dBThreshold* at volume value 10. Below that, the last 0-10 range is a linear dropoff in linear range, i.e. the dB level will drop off much more quickly until it reaches -Infinity at volume value 0 for silence.

---

**MasterVolume**

Return the current master volume set using the *Set master volume* action.

---

**OutputLatency**

The time in seconds reported by the system as the estimated output latency, i.e. the time between the system sending an audio buffer to play, and the time at which the first sample in the buffer is actually processed by the audio output device. This can vary between devices and can also change over time. If the value is unavailable or unknown, it will return 0.

> *The output latency will remain unknown until playback actually starts if autoplay*

*restrictions are in place.*

---

**PlaybackRate(Tag)**

Get the current playback rate of a sound with a tag. The default playback rate is 1, and is set with the *Set playback rate* action.

---

**PlaybackTime(Tag)**

Get the current playback time in seconds of a sound with a tag. This starts at 0 and counts up to the duration, except for looping sounds which keep counting up past the duration.

---

**SampleRate**

Return the audio output sample rate in Hz, typically 44100 or 48000.

---

**Volume(Tag)**

Get the volume set for a sound with a tag.

# BBC MICRO:BIT PLUGIN

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/bbc-micro-bit

The **BBC micro:bit plugin** allows communicating with the BBC micro:bit single-board computer via Bluetooth.

### Setup

For a guide on getting started using a BBC micro:bit in Construct, please refer to the tutorial Getting started with the BBC micro:bit. This also covers the system requirements for the feature. This manual entry is a reference only for Construct's BBC micro:bit plugin.

## BBC micro:bit properties

**Enable accelerometer**

**Enable buttons**

**Enable magnetometer**

**Enable temperature**

**Enable LED display**

Check the corresponding boxes to enable using the various features of the BBC micro:bit from the plugin. If any feature is disabled, note the corresponding conditions, actions and expressions for that feature will not work. Turning off unused features may improve the reliability of the Bluetooth connection. If you do turn off features, don't forget which ones, to avoid confusion in case you try to start using that feature later on as you will then need to re-enable it.

## BBC micro:bit conditions

**Is device connected**

True if a device is currently connected via Bluetooth.

**Is supported**

True if the current browser/platform supports Bluetooth features necessary for communicating with the device (the Web Bluetooth API).

**On device connected**

Triggered after the *Request device* action successfully connects to a device over Bluetooth.

### On device disconnected

Triggered when the Bluetooth connection to a device is lost. This may be intentional (e.g. using the *Disconnect device* action) or due to losing the Bluetooth signal.

### On request device failed

Triggered after the *Request device* action if it failed to establish a Bluetooth connection to a device. This includes the user cancelling the prompt to connect to a device.

### On accelerometer reading

Triggered while connected to a device whenever a reading from the accelerometer sensor is received via Bluetooth. This requires *Enable accelerometer* to be checked in the object's properties. The reading is available via the accelerometer expressions.

### On bearing reading

Triggered while connected to a device whenever a reading from the magnetometer sensor is received via Bluetooth. This requires *Enable magnetometer* to be checked in the object's properties. The reading is available via the *Bearing* expression.

> *Bearing readings may be unavailable unless the device is calibrated.*

### On button pressed

Triggered while connected to a device whenever one of the device buttons is pressed, released or held down for a moment (also known as "long pressed"). This requires *Enable buttons* to be checked in the object's properties.

### On temperature reading

Triggered while connected to a device whenever a reading from the thermometer is received via Bluetooth. This requires *Enable temperature* to be checked in the object's properties. The reading is available via the *Temperature* expression.

## BBC micro:bit actions

### Request device

Request to connect to a BBC micro:bit device over Bluetooth. This must be used in a user input trigger, such as a button click or a tap, as browsers block requests not started by user input. If a Bluetooth connection is established successfully, *On device connected* will trigger. If the user cancels or the request is otherwise unsuccessful, *On request device failed* will trigger.

### Disconnect device

Disconnect from a currently connected device. All communication with the device will stop and both the Construct project and the device will become available to connect to something else.

### Clear LED state

Reset the LED state stored in the plugin so that all LEDs are set to off. Note this does not affect the device until the *Update LEDs* action is used.

### Set LED state

Set one of the LED states on or off in the state stored in the plugin. The LED is specified by its zero-based column and row, so note that the first column and first row have the index 0. Also note that as the device has a 5x5 display, the last columna nd row index that can be used is 4. Note this does not affect the device until the *Update LEDs* action is used.

### Update LEDs

Send the current LED state stored in the plugin to the device via Bluetooth. This requires *Enable LED display* to be checked in the object's properties. Note that the *Clear LED state* and *Set LED state* actions do not affect the device: they only update the LED state stored in the plugin. This action then transmits the LED state stored in the plugin to the device for display, so the former actions can be used to set up the intended LED display, and then this action used to update the actual display on the device.

### Scroll text

Display a short message (up to 20 characters) on the device's LED display. This requires *Enable LED display* to be checked in the object's properties.

## BBC micro:bit expressions

### DeviceID

A string with a unique ID for the connected device.

### AccelerometerRawX

### AccelerometerRawY

### AccelerometerRawZ

The raw accelerometer readings received from the device, updated in *On accelerometer reading*. Each axis returns a number ranging from -1 to 1. The raw readings are the unmodified readings as determined by the sensor - these readings can be "noisy" and have a

lot of variation in them, so it may be preferable to use the smoothed readings, or implement your own smoothing calculation.

---

**AccelerometerSmoothedX**

**AccelerometerSmoothedY**

**AccelerometerSmoothedZ**

Smoothed accelerometer readings received from the device, updated in *On accelerometer reading*. Each axis returns a number ranging from -1 to 1. The smoothed readings employ a simple calculation to eliminate some of the noise in the raw readings, which can be helpful for a smoother and more reliable reading.

---

**Bearing**

The bearing reading in degrees from due north, updated in *On bearing reading*. This requires *Enable magnetometer* to be checked in the object's properties, and note the device must be calibrated before readings will be received.

---

**Temperature**

The thermometer reading in degrees celcius, updated in *On temperature reading*. This requires *Enable temperature* to be checked in the object's properties.

# BINARY DATA

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/binary-data

The **Binary Data plugin** allows raw access to an allocated section of memory (often referred to as a *buffer*). For example it could allocate 16 bytes of memory, and read and write anywhere in that buffer as individual bytes, 32-bit integers, floating point numbers, text, and so on.

A comprehensive description of how binary data/computer memory storage works is out of the scope of this manual. However there is lots of information on the Internet that covers it, and most computer science or computing courses will also cover it. The Binary Data object is also useful even if you do not access its contents: it integrates with other plugins like AJAX and Local Storage, allowing binary data such as images to be stored or transferred in useful ways. Despite the name, Binary Data can also store text, using the *Set from text* action and *GetAllText* expression, which can be useful with other plugins such as Cryptography.

*Construct expressions, like JavaScript, only use double-precision floating point numbers (Float64). When reading and writing other data types with Binary Data, they are converted to and from Float64 - no other types are used in expressions. Fortunately Float64 can store all other types losslessly.*

*The Binary Data object starts empty (with a zero byte buffer). You must set its length or load data from another source before reads and writes can be used.*

*Unlike unmanaged languages like C, the Binary Data object is implemented in the memory-safe language JavaScript. This means the binary data cannot be used unsafely: out-of-bounds writes are ignored, and out-of-bounds reads return 0.*

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IBinaryDataInstance script interface. (JavaScript and TypeScript have support for binary data built-in, but this allows for interacting with binary data used in event sheets.)

## Binary Data properties

#### Endian

The default endian to use when reading and writing binary values.

# Binary Data conditions

### Compare length

Compare the length of the data buffer, in bytes.

### Compare value

Read a value of a given data type at an offset in the data buffer, and then compare that value to another number.

# Binary Data actions

### Copy

Copy data from another (or the same) Binary Data object. This does not change the size of the buffer. The *Start* and *Length* specify the range of data in the source Binary Data object to copy, defaulting to copying the entire buffer. The *Target* is a byte offset to write the copy at.

### Fill

Fill a range of the buffer with copies of the same value. For example this can be used to fill an entire range of the buffer with zero bytes. The *Offset* and *Length* can be used to specify a byte range to fill, but they default to covering the entire buffer.

> *If you fill a value other than Int8 or Uint8, make sure the range size is a multiple of the size of the data type. For example if you fill with a Uint32 value, make sure the range is a multiple of 4 bytes. If the range is not a multiple, the end of the range will not be filled, e.g. filling a 5 byte range with a 4 byte value will only write the first 4 bytes and will not alter the last byte.*

### Set endian

Change the endian used when reading and writing values.

### Set from Binary Data

Set the contents of the Binary Data object to a copy of another Binary Data object. This changes the size of the buffer to be the same as the other object.

### Set from string

Set the contents of the Binary Data object from the contents of a string, changing the size of the data buffer to fit it. The string may be provided in three formats:

- **Text:** set the data to the provided text string encoded as UTF-8.
- **Base64:** set the data by decoding the provided base64-encoded string to binary.

- **Hex:** set the data by decoding the provided hexadecimal-encoded string to binary.

---

**Set length**

Set the size of the memory buffer in bytes. Note Binary Data initially has a zero sized buffer, so this must be used first before any reads or writes can complete. The existing data is preserved when changing the size; if the new size is smaller, data is truncated, and if the new size is larger, zero bytes are added.

---

**Set value**

Write a value of a given data type at a byte offset in the memory buffer. The entire value must be within the bounds of the buffer, otherwise the write is ignored. For example a 4-byte Uint32 value cannot be written anywhere in a 3-byte buffer, because all four bytes must be inside the buffer.

---

**Compress**

**Decompress**

Use a compression algorithm to compress or decompress the data stored in the Binary Data object. The supported compression algorithms are GZIP and DEFLATE. The same algorithm as used for compression must be used for decompression. Compression algorithms work by identifying and eliminating repeating patterns in data and can significantly shrink the amount of data. However the effectiveness depends on the size and type of data. Small amounts of data, or highly unpredictable data, generally does not compress well; large amounts of highly repetitive data generally compresses very well. See the Compression example for a demonstration.

> *These actions are asynchronous, which means they can take a moment to complete while working in the background. Use the system action Wait for previous actions to complete before using any further actions to work with the resulting data.*

## Binary Data expressions

---

**ByteLength**

Return the current length of the buffer in bytes.

---

**GetBase64**

Return the entire contents of the data buffer encoded as a base64 string. This is useful when binary data must be stored in a text-based format like JSON.

> *Base64 data is larger and slower to process than the equivalent binary data. It is more efficient to avoid converting to base64 where possible. For example instead of posting*

*an image to a server as a base64 string, the AJAX object is able to post a Binary Data object directly.*

---

**GetHex**

Return the entire contents of the data buffer encoded as a hexadecimal string. This always uses two hexadecimal digits per byte.

*Base64 is a more efficient text representation of binary data. Hexadecimal is only generally used for relatively small amounts of data such as hashes.*

---

**GetURL**

Return a URL that can be used locally to load the binary data. For example if the Binary Data represents an image, this URL can be passed the Sprite object's *Load image from URL* action to load the image from the Binary Data object.

*The URL is a `blob:` URL referring to data in memory. It can only be used in the same session, in the same browser, on the same device. Sharing the URL or saving it to be re-used in another session later will not work.*

*The provided URL will be valid until the next time the data stored in the Binary Data object changes. For example while the data stays the same, the GetURL expression continues to return the same URL. However if any part of the Binary Data object's stored data changes, the URL returned by the GetURL expression will change, and the old URL will become invalid and no longer work.*

---

**GetUint8(byteOffset)**

**GetInt8(byteOffset)**

**GetUint16(byteOffset)**

**GetInt16(byteOffset)**

**GetUint32(byteOffset)**

**GetInt32(byteOffset)**

**GetFloat32(byteOffset)**

**GetFloat64(byteOffset)**

Read a value of the corresponding type from the buffer at a byte offset. The entire value must be within the bounds of the buffer, otherwise it returns 0. For example a 4-byte Uint32 value cannot be read from a 3-byte buffer, because all four bytes must be inside the buffer.

*Construct expressions, like JavaScript, do not use different number types and instead treat all values as Float64. Therefore the read value is always converted to Float64 for use in expressions.*

---

**GetText(byteOffset, length)**

Decode text in the UTF-8 encoding in a range of the buffer in bytes, and return the string. If the UTF-8 encoding is invalid, or any part of the range is outside the buffer, an empty string is returned.

---

**GetAllText**

Decode all the data in the Binary Data object as text in the UTF-8 encoding. This is the same as using the *GetText* expression with a range that specifies all the data.

# BLUETOOTH

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

The **Bluetooth object** allows communicating with nearby Bluetooth devices via the Web Bluetooth API.

You can find a simple example of using the Bluetooth object in the Bluetooth - device name example.

*It's useful to have a basic understanding of the way Bluetooth works before starting to use the Bluetooth object. There is plenty of information covering the general technology if you search the web. In particular it is useful to know about services and characteristics. Familiarity with Binary Data will also be useful.*

*In Chrome, it can be useful to visit* `chrome://bluetooth-internals` *which provides a debug view of Bluetooth devices. This also helps identifying which services and characteristics a device exposes.*

*As of May 2022, the Bluetooth object is not supported in Android apps. This is because Google have not added support for the Web Bluetooth API in the Android WebView yet. (See this issue for updates). However it should work in the Chrome browser on Android.*

## Specifying services and characteristics

When specifying services and characteristics in the Bluetooth object, there are a few ways of identifying them:

- Use their standard e.g. `"heart_rate"` for the standard heart rate service. See the standard service names and standard characteristic names. Note the prefixes are omitted, e.g. *org.bluetooth.characteristic.gap.device_name* is specified just as *gap.device_name* in the Bluetooth object.
- Use their 16-bit hexadecimal IDs e.g. `"0x1802"`. These can also be found in the linked standard lists, or vendor-specific IDs used.
- Use the full UUID, e.g. `"c48e6067-5295-48d3-8d5c-0395f61792b1"`. This is usually used to refer to a vendor specific characteristic or service.

Where a standard service is specified as a UUID, Construct will turn it back in to its standard name, e.g. *gap.device_name*.

Device IDs can also usually be omitted: if left empty, the first connected device will be used.

## Bluetooth conditions

---

**Is device connected**

Test if a bluetooth device is currently connected by its device ID.

---

**Is supported**

True if bluetooth is supported on the current platform. The plugin will not work if this is false.

---

**On device connected**

**On device disconnected**

Triggered as devices connect (after *Request device* completes successfully) and disconnect. The *DeviceID* expression is set to the ID of the relevant device.

---

**On request device failed**

Triggered after the *Request device* action if the user cancels or a Bluetooth connection was unable to be established.

---

**On characteristic notification**

**On any characteristic notification**

Triggered after starting notifications for a characteristic whenever a notification is received. The notification data is loaded in to the provided Binary Data object. The *any* variant is triggered for all characteristics rather than a specific one, and sets the *CharacteristicID* expression accordingly

---

**On characteristic read**

**On any characteristic read**

**On any characteristic read error**

Triggered after the *Read value* or *Read binary* actions that attempt to read a characteristic value. If the read fails the error trigger fires. The *any* variant is triggered for any characteristic that successfully completes a read rather than a specific one, and sets the *CharacteristicID* expression accordingly. Once the read completes successfully, the characteristic value expressions are set, or if binary was read the data is now available in the chosen Binary Data object.

---

**On characteristic written**

**On any characteristic written**

**On characteristic write error**

Triggered after the *Write binary* action after the operation completes. If the write fails the error trigger fires. The *any* variant is triggered for any characteristic that successfully completes a write rather than a specific one, and sets the *CharacteristicID* expression accordingly.

---

**On charactersitics loaded**

**On characteristics error**

Triggered after the *Request characteristics* action depending on the outcome. If successful the characteristic list is available with the *CharacteristicCount* and *CharacteristicAt* expressions.

---

**On primary services loaded**

**On primary services error**

Triggered after the *Request primary services* action depending on the outcome. If successful the primary service list is available with the *PrimaryServiceCount* and *PrimaryServiceAt* expressions.

## Bluetooth actions

---

**Request device**

Prompt the user to choose a nearby Bluetooth device to connect to. This must be used in a user input trigger. The listed Bluetooth devices must be filtered by providing one of *Services* (or *Optional services*), *Device name* or *Device name prefix*; only matching devices will be shown. For possible ways to specify services, see *Specifying services and characteristics* above. *On device connected* triggers if successful, else *On request device failed*.

---

**Disconnect device**

Disconnect a connected Bluetooth device by its device ID.

---

**Read value**

**Read binary**

Read data from a Bluetooth characteristic. The device ID can be left blank to use the default device (the first device that is connected). For the service and characteristic, see *Specifying services and characteristics* above. One of the characteristic read triggers will run depending on the outcome. If successful, the selected Binary Data object will have the read data loaded for the *binary* variant, or the characteristic value expressions will be set for the *value* variant.

---

**Write binary**

Write data to a Bluetooth characteristic from a Binary Data object. The device ID can be left blank to use the default device (the first device that is connected). For the service and characteristic, see *Specifying services and characteristics* above. One of the characteristic write triggers will run depending on the outcome.

---

**Start/stop notifications**

Start or stop notifications for a Bluetooth characteristic. The device ID can be left blank to use the default device (the first device that is connected). For the service and characteristic, see *Specifying services and characteristics* above. While notifications are started, the notification triggers will run whenever a notification is received.

---

**Request primary services**

Request a list of primary services available for a device ID (which can be left blank to use the default device). If successful triggers *On primary services loaded*, and the primary service expressions can be used.

> *Browsers may filter the listed primary services depending on what was specified in Request device.*

---

**Request characteristics**

Request a list of characteristics available for a service. If successful triggers *On characteristics loaded*, and the characteristic expressions can be used.

## Bluetooth expressions

---

**DeviceID**

**PrimaryServiceID**

**CharacteristicID**

In a trigger, the ID of the relevant device, primary service, or characteristic, where applicable.

---

**DeviceName**

In *On device connected*, a string with the name of the device that connected.

---

**CharacteristicSignedInteger**

**CharacteristicUnsignedInteger**

**CharacteristicFloat**

**CharacteristicText**

After a successful *Read value* action, these expressions return the read value in a variety of different types. Construct will attempt to deduce the type from the length of the data, e.g. if the data is 1, 2 or 4 bytes long it will read it as both a signed and unsigned integer and set the expressions accordingly; if 4 or 8 bytes long set the float value; and always attempt to read the value as text. These values will only be meaningful if they match the type of value the characteristic really stores. For other data types, use the *Read binary* action instead.

---

**PrimaryServiceCount**

**PrimaryServiceAt(index)**

After *On primary services loaded*, use these expressions to retrieve the number of services and the service identifier at each index in the list.

---

**CharacteristicCount**

**CharacteristicAt(index)**

After *On characteristics loaded*, use these expressions to retrieve the number of characteristics and the characteristic identifier at each index in the list.

# BROWSER

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

The **Browser** object accesses features of the browser or platform running the project. It also provides features like switching to and from fullscreen mode, detecting if an update is available, and determining if the page is visible.

Security limitations sometimes prevent browser actions. For example to prevent abuse, the window *Close* action can only be used when the window was opened automatically by a web page and not when it was opened by the user.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in APIs to access the same features.

## Browser conditions

**Cookies enabled**

True if the user has cookies enabled in their browser.

**Is online**

True if the browser thinks it currently has an active connection to the Internet. Construct projects can work offline - see Offline games for more information.

> *The "is online" detection of the Browser plugin is more or less a guess. It is actually difficult to categorically determine if an Internet connection is available. For example the user may have intermittent signal, and online features sometimes work and sometimes fail unpredictably. In this case the "Is online" detection is probably wrong, as it can't actually tell whether a connection will work. Rather than using this condition to check if online features will work, usually it's better to just go ahead and use online features anyway as if the user is online, and handle any errors that occur as a result.*

**On went online**

**On went offline**

Triggered when the browser thinks the connection to the Internet has become available or unavailable during the running of the project. This is common on mobile devices which may be moving in and out of signal areas. The *Is online* condition also changes to reflect the connection status.

*See the caveat about online detection above.*

### Is portrait/landscape

Determine if the current display is portrait (height is greater than width) or landscape (width is greater than height). This is performed by making a simple check on the window size of the browser, so also returns accordingly on a desktop browser depending on its dimensions.

### On back button

Triggered when the user presses the device's 'Back' button. Note not all devices have this button (e.g. iOS devices only have a 'Home' button) and not all platforms support this trigger.

### On hash change

Triggered when the part of the URL after the hash character (e.g. `example.com/index.html#hashpart`) changes. The *Hash* expression will return the new value.

### On offline ready

Triggers the first time the project runs when it has finished downloading and is ready to use offline.

### On update found

Triggers when an update is detected. The update will download in the background and trigger *On update ready* when it is ready to be used.

### On update ready

Triggered when an updated version has finished downloading in the background. If the user is still on the project's menu or title screen, you may wish to prompt them to refresh the page (or just do it automatically) so the new version is loaded. See Offline games for more information.

### Compare display mode

Test the current display mode of the project. This corresponds to the CSS `display-mode` media query. Normal display in a web browser usually counts as the *browser* display mode. However if a web app install has been performed and the project is being displayed in its web app form without the usual browser interface elements like the address bar, then usually that counts as the *standalone* display mode.

### On install available

### Is install available

Even in browsers that support web app installs, the option to install may not be immediately available. *On install available* triggers when a web app install first becomes available, and after that *Is install available* remains true, indicating that the *Request install* action may be used.

---

### On install result

Triggered after the *Request install* action has been used to display an install prompt to the user. The possible install results are:

- Accepted: the user chose to install the web app.
- Dismissed: the user cancelled or otherwise declined to install the web app.
- Error: something went wrong with the install prompt and it was not able to complete successfully.
- Any: run the trigger regardless of the result. The *InstallResult* expression can be used to retrieve a string representing the result.

---

### On app installed

Triggered after the *Request install* action has been used to display an install prompt to the user, and the user chose to accept installation, and then the web app installation completed successfully. Usually this also corresponds to the browser changing the display mode of the project to its app form (typically *standalone* display mode).

---

### Has focus

### On focus

### On blur

*Has focus* is true if the window currently has focus, which means keyboard input will be received by the window. *On focus* is triggered when the window becomes focused, and *On blur* is triggered when the window loses focus.

> *In some cases focus refers to a frame instead of an actual window - for example inside an iframe, focus means that the user is interacting with the iframe rather than the parent frame.*

---

### Is fullscreen

True if the browser is running in fullscreen mode after using the *Request fullscreen* action.

*This condition does not detect if the user pressed F11 or used another browser-provided shortcut to enter fullscreen mode - it only checks if it's fullscreen due to using Request fullscreen.*

**On fullscreen error**

Triggered when a request to enter fullscreen mode was refused by the browser. This may be because the request was made outside of a user input trigger (e.g. mouse click or touch), or when running inside an iframe which blocks fullscreen permission.

**On resized**

Triggered when the browser window displaying the project is resized. This includes when changing orientation on a mobile device.

## Browser actions

**Start group**

**End group**

Start or end a group in the browser console. Groups appear indented, and the browser may give the option to expand/collapse the group easily. Groups can optionally be named. To create a group, use *Start group*, then a series of *Log* actions, then the *End group* action.

**Log**

Log a message, warning or error to the browser console. This can be useful for debugging, testing and diagnostics.

**Maximize window**

**Minimize window**

**Restore window**

Maximize or minimize the main app window, or restore it to its original size and position if it was previously maximized or minimized. These actions only work in desktop exports (Windows, macOS and Linux) as browsers do not allow controlling the window in this way, and mobile devices generally do not provide windowing capabilities.

**Vibrate**

Vibrate the device with a given pattern, if the device/platform supports vibration. The pattern is given as a comma-separated list of times in milliseconds, alternating between vibrate time and waiting. For example the string "200,100,200" specifies a 200ms vibration, 100ms pause, then another 200ms vibration. This allows a single action to specify a whole vibrate pattern.

### Load stylesheet

Load a Cascading Style Sheet (CSS) from a URL and applies its styles to the document. This can be useful for styling form controls and other DOM elements. The URL can also be the name of a project file, such as "myfile.css", to load a CSS file included in the project.

*CSS can only style DOM elements such as form controls. Note that most objects like Sprite and Text render directly in to the canvas and are not DOM elements, so cannot be styled with CSS.*

### Set CSS style

Set a CSS style on the style attribute of some HTML elements in the document, based on a CSS property name and a string for its value. Setting the value to an empty string will remove the property from the style attribute. The element to change the style for is set by a CSS selector, e.g. `".myclass"` will mean to update the CSS style of an element with the class *myclass*; if the *Type* is set to *all*, it will update the style of all elements matching the selector.

*This is useful for setting document-wide CSS variables that can be used with other HTML features like form controls or HTML Element.*

*This action is asynchronous because in worker mode the document cannot be accessed directly. When the action finishes, the change has been made to the document.*

### Get CSS style

Retrieve a string with the computed style value for a CSS property on an element given by a CSS selector. When the action finishes, the value is returned by the *CSSStyleValue* expression. A tag is used to identify different CSS style values.

*This is useful for retrieving the value of CSS variables from stylesheets in the project.*

*The action is asynchronous because in worker mode the document cannot be accessed directly. When the action finishes, the result is available via the CSSStyleValue expression, passing the same tag.*

### Go back

### Go forward

Move through the browser navigation history as if clicking the Back and Forward buttons on the browser.

---

**Go to URL**

Navigate to a given URL. Note this uses the same window/tab as is showing the project, so this action will end the project. The *Target* can be used to select which frame to redirect, which is only useful if the project is displayed within a frame (e.g. an iframe embed), and the frame has permission to redirect the parent frame (i.e. it is not sandboxed). Possible targets are:

- **Self:** redirect only the frame that is currently showing the project.
- **Parent:** redirect the parent frame.
- **Top:** redirect the top level frame (only different to the parent if more than one frame is used)

---

**Invoke download**

Invoke a URL as a file download in the browser. Even if this points to a web page or document, it will be downloaded as a file in the browser interface. The URL can point to any address on the Internet, or it can be the name of any imported project file, or it can be a data URL (useful for downloading canvas snapshots). The filename parameter allows you to choose the filename the browser gives to the download, which can be different to the name of the resource being downloaded.

> *Downloading is a browser feature and depends on the browser UI. Note that mobile apps don't run in browsers (there is no address bar etc), so the download feature isn't available there. Consider using the Share plugin to share the file instead.*

---

**Invoke download of string**

As with *Invoke download*, but instead of providing a URL to download, a string of the actual data to download as a file is used. A data URI combining the MIME type and data is created, then passed to the browser to download. This is convenient for downloading strings in JSON format as files, e.g. object data from the *AsJSON* expression.

---

**Open URL in new window**

Navigate to a given URL in a new window (or tab if the browser settings override). This continues to run the project in the old window or tab.

---

**Reload**

Force the page to refresh. This effectively restarts the project.

---

**Set hash**

Set the part of the URL after the hash character (e.g. `example.com/index.html#hashpart`). Note this does not reload the page, so the hash part of the URL can be used as a kind of mode or identifier that can be changed while the project is running.

---

**Set warn on close**

Enable a warning upon closing the browser tab or window. By default this is disabled, but when enabled, the user will receive a prompt to confirm closing the browser tab or window. A generic message will be shown, usually with text along the lines of "Leave site? Changes you made may not be saved." This provides the option to cancel closing the tab/window, which may help prevent accidental closure or losing unsaved progress. Note that browsers show this message at their discretion and in some circumstances may decide not to show a prompt for privacy or security reasons.

---

**Request install**

Where available, prompt the user to install the current page as a web app. This is normally only available in a web browser which supports Progressive Web App (PWA) installs. This is only allowed when installation is available, indicated by the *On install available* or *Is install available* conditions. Further it normally can only be used in a user input trigger, such as a button click or tap, and may only be used if not already installed. If the install prompt is shown, then *On install result* triggers afterwards with the outcome; if the install prompt was successful and the web app successfully installed, then *On app installed* triggers, and the display mode changes. Note that the style of the install dialog depends on how much information your project provides - see the section on *Installable web apps* in Publishing to the web for more details.

---

**Alert**

Bring up a simple 'alert' message box.

---

**Blur**

Unfocus the browser window.

---

**Cancel fullscreen**

Return to windowed mode if the browser is currently in fullscreen mode.

---

**Close**

Close the current window, if the script has permission to do so.

### Focus

Request to focus the window. Browsers may disregard this request to prevent abuse. In desktop exports, this will attempt to bring the main app window to the top, but similarly the system may not always allow this in order to prevent abuse.

### Lock orientation

### Unlock orientation

Lock the display of the project to a portrait or landscape mode only, if the current platform supports this. This only has effect on mobile devices. The project may have to already be displaying in fullscreen (using the *Request fullscreen* action) before the orientation can be locked. Unlocking the orientation restores whatever behavior was set before locking, such as automatically changing orientation depending on the way the device is being held.

### Request fullscreen

Request that the browser enter fullscreen mode. Note the browser may refuse this request unless the action is in a user-initiated event, such as a mouse click, key press, touch event or button press. It may also be refused if fullscreen permission is not available, such as if running inside an iframe which does not grant fullscreen permission. If the fullscreen request is ignored, *On fullscreen error* will trigger.

> *In desktop exports (Windows, macOS and Linux), fullscreen requests are always allowed at any time.*

The fullscreen modes that can be entered correspond to the *Fullscreen mode* project property. *Navigation UI* where supported sets whether the browser should show browser elements such as back buttons or the address bar, or hide them (for a true fullscreen experience). Typically this setting only affects mobile browsers.

### Set window size

### Set window position

Set the size and position of the main window. This is only applicable on desktop-style systems - mobile devices typically use fullscreen apps and therefore windows cannot be repositioned or resized.

> *Note that when running in a web browser, the browser may sometimes refuse to change the window size or position, as a measure to protect the user from unwanted changes. This can also include being unable to alter the main window when the project is running in an iframe.*

## Browser expressions

**Language**

Get the browser's current language setting, e.g. *en-US*.

**Platform**

Get the current platform the browser reports itself running on, e.g. *Win32* for Windows.

**UserAgent**

Return the full user agent string for the browser, e.g. *Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36*.

**CSSStyleValue(tag)**

Get the CSS style value retrieved from a prior call to the *Get CSS style* action. Once the action has finished (using *Wait for previous actions to complete*), pass the same tag as the action was called with to retrieve the result.

**Title**

The current HTML document's title.

**Domain**

The current domain, e.g. *construct.net*.

**Hash**

The string after the hash at the end of the URL, including the hash. For example, if the current URL is `example.com/index.html#hashpart`, this returns *#hashpart*.

**PathName**

The path relative to the domain in the URL. For example the path name of *https://construct.net/myproject/index.html#teapot* is */myproject/index.html*.

**Port**

A string of the port specified in the URL, if any. Note while ports are numbers, this expression returns a string, since if no port is specified in the URL it will return an empty string.

**Protocol**

The current protocol, usually either *http:* or *https:*.

**QueryParam**

Return a query string parameter by name. For example, if the URL ends with *index.html?foo=bar&baz=ban*, `QueryParam("foo")` returns *bar* and `QueryParam("baz")` returns *ban*.

---

**QueryString**

Return the full URL query string including the question mark. For example, if the URL ends with *index.html?foo=bar&baz=ban*, this returns *?foo=bar&baz=ban*.

---

**Referrer**

Get the previous page that linked to this page, if any.

---

**URL**

Get the complete current URL in the browser address bar, including the protocol.

---

**DisplayMode**

A string indicating the current browser display mode - see the *Compare display mode* condition for more details. This expression returns one of the strings "browser", "standalone", "minimal-ui" or "fullscreen".

---

**InstallResult**

In the *On install result* trigger, a string representing the install result. This may be one of the strings "accepted", "declined", "unavailable", "failed", or an empty string ("") if no install yet attempted.

# BUTTON

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

The **Button** object creates a button control which the user can click to perform an action. It can also be set to be a **checkbox**.

### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Styling buttons

As Button objects are HTML elements, their appearance can be customised using CSS (Cascading Style Sheets). The *ID* and *Class* properties can be used to identify the HTML element, and a CSS project file added to apply some styles to it.

## Button properties

**Type**

Either *Button* for a push-button or *Checkbox* for a checked/unchecked control.

**Text**

The text appearing on the button face or checkbox label.

**Tooltip**

A tooltip that appears in most browsers if the user hovers the mouse over the button and waits. Leave blank for no tooltip.

**Initially visibile**

Whether or not the button is shown on startup. If Invisible, the button must be shown with the Set visible action.

**Enabled**

Whether the button is initially enabled. If disabled, the button will be greyed out and cannot be pushed.

**Auto font size**

Automatically set the font-size property of the element according to the layout and layer scale. This will prevent the font-size CSS property being manually set with the *Set CSS style* action. Disable if you intend to use *Set CSS style* to adjust the *font-size* property.

**Checked**

If *Type* is *Checkbox*, this is the initial check state of the control.

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

**Class Optional**

An optional *class* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

## Button conditions

See common conditions for features shared between form control objects.

**Is checked**

If *Type* is *Checkbox*, is true if the control is currently checked.

**On clicked**

Triggered when the user pushes the button or checks/unchecks the control, either by keyboard, mouse or touch input.

## Button actions

See common actions for features shared between form control objects.

**Set checked**

If *Type* is *Checkbox*, set the current check state of the control.

**Set text**

Set the text on the button face.

---

**Set tooltip**

Set the text that appears for the button tooltip. Leave blank for no tooltip.

---

**Toggle checked**

If *Type* is *Checkbox*, toggles the check state of the control.

## Button expressions

The Button object does not have any of its own expressions.

# CLIPBOARD

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

The **Clipboard** object provides access to the system clipboard, such as copy and paste operations.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Clipboard API.

## Browser permissions

For security and privacy reasons, clipboard access - particularly paste operations - are usually controlled by permission prompts in browsers. Further, these operations are sometimes only allowed in a user input trigger, such as a button click or touch start. This prevents situations like web pages being able to immediately read from the user's clipboard before they've even interacted with the page. However it may mean you need to adjust the way your project accesses the clipboard.

## Clipboard conditions

**On copy success**

Triggered after a copy action when the copy has successfully completed, so the data is now on the user's clipboard.

**On copy error**

Triggered after a copy action fails. The user may not have granted permission or the attempt may not have been in a user input trigger.

**On paste text**

Triggered after a *Request paste text* action completes successfully. The text from the user's clipboard is available with the *PastedText* expression.

**On paste binary**

Triggered after a *Request paste binary* action completes successfully, and the data in the user's clipboard matches the MIME type specified in the trigger (e.g. "image/png" for a PNG image). This allows using different triggers to receive different types of clipboard data. When this triggers, the pasted data is placed inside the chosen Binary Data object.

### On paste error

Triggered after a paste action fails. The user may not have granted permission or the attempt may not have been in a user input trigger.

## Clipboard actions

### Copy text

Copy a string of text to the user's clipboard. This may require permission from the browser. *On copy success* or *On copy error* will trigger afterwards depending on the result.

### Copy binary

Copy the contents of a Binary Data object the user's clipboard. The MIME type of the data must also be specified, e.g. "image/png" for a PNG image, so other apps know whether they can paste the data. Copying may require permission from the browser. *On copy success* or *On copy error* will trigger afterwards depending on the result.

### Request paste text

Request to read text from the user's current clipboard. This usually involves a permission prompt in browsers and may only be allowed in a user input trigger. If successful then *On paste text* will trigger and the pasted text will be available in the *PastedText* expression; otherwise *On paste error* will trigger.

### Request paste binary

Request to read binary data from the user's current clipboard. This usually involves a permission prompt in browsers and may only be allowed in a user input trigger. If successful then *On paste binary* will trigger assuming the type of the clipboard data matches the type specified in the trigger; then the data is available in the chosen Binary Data object. Otherwise *On paste error* will trigger if the attempt to paste fails.

## Clipboard expressions

### PastedText

After *On paste text* triggers, this is set to the text that was copied to the user's clipboard.

# CONSTRUCT GAME SERVICES PLUGIN

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/construct-game-services

The **Construct Game Services** (CGS) plugin allows accessing game services hosted by Scirra, the makers of Construct. Features include authentication, cloud save, and submitting high scores both for the Construct Arcade and cross-platform leaderboards.

### Scripting

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

## Authentication

If you want to allow the player to sign in, you can make a login system with these steps.

1. In your Construct Game Services account, create a game. Take note of the game ID.
2. Use an action like *Sign in with provider* to provide a login using one of several existing account providers like Google, Microsoft or Steam. (You may want to allow multiple providers.) You'll need to specify the game ID to sign in to.
3. When the player chooses this option, a popup window will appear asking them to log in, or if they are already logged in, to confirm that they wish to proceed. (In desktop exports this will instead launch the system browser, and in mobile exports it will use an in-app browser so the user does not have to leave the app.)
4. If the player signs in successfully, then *On success* will trigger for *Sign in with provider*. You can then access the *PlayerName* and *PlayerID* expressions, and make use of other features like submitting scores to cross-platform leaderboards.

See the Log in with providers example for a demonstration of this, including support for handling popup blockers and persistent sign in.

### Handling popup blockers

When running in a web browser, it is possible the browser popup blocker will prevent the login popup window from opening. In this case *On sign in popup blocked* will trigger, and you can provide an option to retry opening the popup window with the action *Retry sign in popup*. You should display a message to the user explaining what happened and be sure to use the retry action in a user input trigger, such as a button click or mouse click, otherwise the second attempt may also be blocked.

You can test that your retry flow works correctly by checking the Construct Game Services plugin property *Simulate popup blocked*. When enabled this will not actually attempt to open a popup window on the first attempt, and will instead act as if the browser popup blocker prevented it from opening, triggering *On sign in popup blocked*. Then the second attempt made with *Retry sign in popup* will attempt to open the popup window normally. Turn this on to check your project handles popup blockers correctly, but be sure to turn it off again afterwards!

In desktop and mobile exports the popup will never be blocked - this only needs to be handled for web browsers.

### Sign in persistent

If the user signs in with *Allow persisting* enabled, then their sign in is remembered. Upon returning the *Can sign in persistent* condition will be true, and you can offer an option to log in again using the *Sign in persistent* action. This can continue with the previous sign in without showing any user interface.

If the user signs out, it will delete their remembered sign in, so *Can sign in persistent* will not be true when they return to your game.

## Leaderboards

Scores can be submitted to either the Construct Arcade or a cross-platform leaderboard.

### Construct Arcade leaderboards

To publish a game to the Construct Arcade with leaderboards, follow these steps:

1. Add the Construct Game Services plugin to your project
2. Add a *Submit score* action to submit the player's score at the appropriate time (such as when a 'Game over' screen appears). Leave the *Leaderboard ID* parameter empty.
3. Publish to the Construct Arcade following the normal process
4. Play your game until a score is submitted
5. Reload the page and a leaderboard will have appeared with your score

Note that you do not need to add any authentication features to your project to submit a score to the Construct Arcade, as it handles authentication automatically.

### Cross-platform leaderboards

To make use of leaderboards on other platforms, follow these steps:

1. In your Construct Game Services account, create a game (if you haven't already), and then create a leaderboard in that game. Take note of both the game ID and leaderboard ID.
2. Add a sign-in feature to your game using the authentication features, as players must be signed in to be able to submit a score. You'll need to specify the game ID to sign in to.

3. Once a player is signed in, you can use the *Submit score* action, specifying the leaderboard ID to submit to.

You should then be able to find the score on the leaderboard in your Construct Game Services account. You can display scores in your game using the *Get leaderboard scores* action.

See the Online leaderboards example to try it out and see how it works.

## Cloud save

Construct Game Services allows saving relatively small amounts of data for the signed in player. This can be used for a range of purposes, but is ideal for saving the user's progress online. This allows players to restore their progress on any device that they log in to the same account with, rather than only saving to the local device storage (which is what Construct's savegame feature does by default).

To set up a cloud save feature, you only need to have a user successfully signed in (see the *Authentication* section above). Then you can change Construct's savegame feature to save online by using the *Save game to JSON string* and *Load game from JSON string* actions, and saving and loading the JSON string with the Cloud Save *Create* and *Get* actions. See the Cloud Save example for a demonstration.

You can store data under different key names, such as "slot1", "slot2" etc. if you wanted to provide different save slots. However for the simple case of a single online save per user, you only need a single key name like "savegame".

You can also create separate storage *buckets* in your Construct Game Services account, which are accessible to all players. However in the simple case of using storage private to a player's own account, you can leave the bucket ID empty.

## CGS properties

**Simulate popup blocked**

When enabled, act as if the first attempt to open the login popup window is blocked, requiring the use of *Retry sign in popup*. See the section *Handling popup blockers* above for more information.

## CGS conditions

**Can sign in persistent**

True if the user previously successfully signed in when *Allow persisting* was enabled and did not sign out. This means the *Sign in persistent* action can be used to continue with the previous sign in.

**Is signed in**

True if the user is currently signed in.

---

**On sign in popup blocked**

Triggered during sign in if the browser prevented the login popup window from appearing. See the section *Handling popup blockers* above for more information.

---

**On create success**

**On create error**

Triggered after the cloud save *Create* actions with the corresponding bucket ID and key, depending on the outcome of the operation.

---

**On get success**

**On get binary success**

Triggered after the cloud save *Get* action with the corresponding bucket ID and key completes successfully. When the *Get* action *Type* is *text*, *On get success* triggers, and the downloaded data is provided by the *CloudSaveText* expression; when *Type* is *binary*, *On get binary success* triggers, and the downloaded data is placed in the specified Binary Data object.

---

**On get error**

Triggered after the cloud save *Get* action with the corresponding bucket ID and key fails to complete. This could occur if the user went offline, or if a network error occured.

---

**On success**

**On error**

Triggered after an action depending on whether it completed successfully or failed.

## CGS actions

---

**Sign in with provider**

Attempt to sign the user in with a third-party identity service like Google or Microsoft. The game ID to sign in to from your Construct Game Services account must be provided. If *Allow persisting* is enabled, a successful sign in is remembered, and may be re-used in future with the *Sign in persistent* action. The *Expiry* is the time in minutes that sessions remain active. Signing in with a provider will open a popup window, and you can also specify the size of this popup window. Upon completing a sign in, *On success* will trigger for *sign in with provider*. In some cases if the sign in is cancelled or fails, *On error* will trigger.

*Note that in some cases, sign in may be cancelled in a way that cannot be detected (neither On success nor On error will trigger). Be sure to design your project with this in mind - for example do not block the user interface until sign in finishes, as you cannot reliably detect that.*

---

**Sign in persistent**

Attempt to sign in re-using a previous successful sign in that allowed persisting. This can only be used when *Can sign in persistent* is true. This will trigger either *On success* or *On error* for *sign in persistent* depending on whether the persistent sign in completed successfully.

---

**Sign out**

Sign out of any account the user is currently signed in to, and also delete any remembered sign in if it allowed persisting. Locally this operation completes immediately and so there is no corresponding success or error trigger. For completeness signing out will send a request in the background to ensure the session is ended on the server-side as well, but that is optional and if it fails the session will time out anyway.

---

**Retry sign in popup**

After *On sign in popup blocked* triggers, attempt to open the login popup window again. See the section *Handling popup blockers* above for more information.

---

**Set player name**

Set the name of the currently signed in player. In some cases the player name is obtained from the authentication provider, but in others a generic name will be used, in which case the player will likely want to change their name for features like leaderboards. The current player name is provided by the *PlayerName* expression.

---

**Create (binary)**

**Create (text)**

Create or replace the data for a key. A game bucket ID can optionally be provided, but if left empty it will use player private storage. The *text* variant uploads a string for the data, whereas the *binary* variant uploads the contents of a Binary Data object. *On create success* or *On create error* will trigger depending on whether the upload completed successfully.

---

**Get**

Download the data previously uploaded for a key with one of the *Create* actions. A game bucket ID can optionally be provided, but if left empty it will use player private storage. If this fails, *On get error* will trigger. Otherwise when *Type* is *Text*, *On get success* will trigger, and the

downloaded data will be in the *CloudSaveText* expression; when *Type* is *binary*, *On get binary success* will trigger, with the downloaded data stored in the specified Binary Data object.

---

**Submit score**

Submit a score to a leaderboard. The *Score* must be an integer (fractional scores are not supported). The *Leaderboard ID* may be left empty to submit a score on the Construct Arcade, which does not require authentication. Otherwise it may be set to the leaderboard ID to submit the score to, in which case authentication is required.

---

**Get leaderboard scores**

Request a page of scores from a given leaderboard ID, with a specified zero-based page number and number of results per page. This does not require authentication. The returned scores can optionally be filtered with several options. An ISO 3166-1 alpha-2 country code filter can be provided, e.g. "US" to only return scores submitted in the United States of America. A time range can also be specified, such as *Daily* to return today's scores. Weekly leaderboards run from Monday to Sunday. A range offset can also be specified to return a prior range - for example specifying a *Daily* range with offset 1 will return yesterday's scores. The *Culture* can be set to a locale to use for returned values, or left empty to use the leaderboard's default culture. *On success* will trigger for *get scores* when scores are retrieved successfully, in which case the leaderboard expressions can be used to access them.

## CGS expressions

---

**GameID**

A string of the game ID currently signed in to.

---

**PlayerID**

A string with a unique ID to identify the currently signed in player.

---

**PlayerName**

The display name of the currently signed in player.

---

**CloudSaveBucketID**

**CloudSaveKey**

**CloudSaveName**

In a Cloud Save trigger, the bucket ID, key and name of the associated cloud save operation.

---

**CloudSaveText**

After *On get success* triggers (after the *Get* action with type *text*), the string of text data that was downloaded.

---

**ScoreCount**

After successfully retrieving leaderboard scores, the number of scores available.

---

**ScoreAt(index)**

**ScoreFormattedAt(index)**

**ScoreRankAt(index)**

**ScoreFormattedRankAt(index)**

**ScorePlayerIDAt(index)**

**ScorePlayerNameAt(index)**

**ScoreCountryAt(index)**

After successfully retrieving leaderboard scores, these expressions return information about a score at a given zero-based index. Scores and ranks are numbers, but the formatted versions return a string formatted according to the locale. Countries are returned as ISO 3166-1 alpha-2 country codes.

---

**TotalPageCount**

After successfully retrieving leaderboard scores, the total number of pages available.

# CRYPTOGRAPHY

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

The **Cryptography** object can perform cryptographic operations such as encryption and hashing.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Web Crypto API.

## Encryption does not guarantee security

> *Please read this section carefully if you intend to use encryption in your project.*

The Cryptography object can perform password-based encryption and decryption. This is intended for both educational purposes to help illustrate the principles of encryption, and also as a means to make it harder, although not impossible, to make unwanted modifications to sensitive data such as save game data (which could for example allow cheating).

Merely using encryption **does not mean your project is secure.** Designing truly secure systems involves careful design of the overall system and is usually done by people with training and expertise, and may well need to involve a server to host the sensitive data out of reach of clients. In particular, if you enter a long and supposedly secure password for the *Encrypt* action in the event sheet, then that password will appear in plain text in your project's exported data files. A sufficiently motivated person will likely be able to identify that password in the exported files and then be able to view or modify any data encrypted with the same password. Even if you use a dynamically generated password so that the password does not appear in the project's exported data files, a more advanced adversary will still be able to use debugging tools to identify the password at the moment it is used to encrypt data.

In short you should see encryption as a tool that makes it somewhat harder to view or modify data, but certainly not impossible. In some cases this is still sufficient. For example if you are concerned about people using browser developer tools to casually modify save game data and cheat, encrypting the data may be sufficient to deter casual modification, even if more skilled adversaries are still capable of it.

## HTTPS is already encrypted

Note that if you want to send data to a server securely, the best approach is to use a secure connection over HTTPS. This is already encrypted, and so there is no need to use the Cryptography object to encrypt data sent over a secure connection.

## Processing text

Since encrypted data is usually binary data that is not easily representible as text, the Cryptography object generally works with the Binary Data object for both its input and output. However this does not preclude the ability to use text: after all, text is just another kind of binary data. You can store text in a Binary Data object with the *Set from text* action, and retrieve text from it with the *GetAllText* expression. Using these allows performing tasks like encrypting and decrypting text or hashing text.

You can also reliably display and transmit binary data in a text format by encoding it as Base64 using both the *Set from base64* action and *GetBase64* expression.

## Examples

See the Encryption example for a basic demonstration of encryption and decryption of a message. The Hashing example also demonstrates hashing text or files.

## Encrypted data format

For advanced users who wish to interoperate encrypted data with other code or services, the encrypted binary data contains a small amount of metadata added at the start by Construct to aid in decryption. The format of the data is described below.

| Bytes | Description |
|---|---|
| 0-1 | Reserved (must be 0) |
| 1-17 | Salt (16 bytes) |
| 17-29 | Initialization vector (aka IV, 12 bytes) |
| 29-33 | Iterations (uint32, 4 bytes) |
| 33+ | Encrypted data payload |

## Cryptography conditions

**On encryption finished**

**On encryption failed**

**On any encryption finished**

**On any encryption failed**

Triggered after an encryption action depending on whether the operation completed successfully. A tag can be specified to distinguish different encryption operations. The "any" variants always trigger regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

**On decryption finished**

**On decryption failed**

**On any decryption finished**

**On any decryption failed**

Triggered after a decryption action depending on whether the operation completed successfully. Decryption will fail if the incorrect password is specified. A tag can be specified to distinguish different decryption operations. The "any" variants always trigger regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

---

**On hash finished**

**On any hash finished**

Triggered after a hash action when the operation completes. A tag can be specified to distinguish different hashing operations. The "any" variant always triggers regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

## Cryptography actions

---

**Encrypt binary**

Encrypt the contents of a given Binary Data object. The same provided password must be specified to successfully decrypt the data. The *Iterations* parameter indicates how many times to repeat a hash function when generating an encryption key; higher values use more processing power and so slow down encryption and decryption, but make it harder to use a brute-force attack to decrypt the data. An optional tag can be specified to distinguish different encryption operations. When the data has finished being encrypted, *On encryption finished* triggers.

> *Using encryption does not guarantee your project is secure. See the section Encryption does not guarantee security above.*

---

**Decrypt binary**

Decrypt the contents of a given Binary Data object. Decryption will only succeed if the same password that was used to encrypt the data is provided. An optional tag can be specified to distinguish different decryption operations. When the data has finished being decrypted, *On decryption finished* triggers; if decryption fails, including due to having the wrong password, then *On decryption failed* will trigger instead.

---

**Hash binary**

Compute a cryptographic hash function for the contents of a given Binary Data object. Currently supported hash functions include SHA-256, SHA-384 and SHA-512 from the SHA-2 family of functions. An optional tag can be specified to distinguish different hasing operations. When the hashing completes, *On hash finished* will trigger and the resulting hash will be available as a hexadecimal string in the *Hash* expression.

## Cryptography expressions

### Hash

After *On hash finished* triggers, a hexadecimal string of the resulting hash. For example the SHA-256 hash of the string "Construct" is `67e7c6e28e540b4fd412ad663b634a38572b98d3f4608fc5792f2600d32c1eb9`.

### Tag

In a trigger such as *On any decryption finished* or *On any hash finished*, this returns the associated tag of the action that resulted in the trigger.

### RandomUUID

Returns a random v4 UUID as a string using a cryptographically secure random number generator, e.g. `36b8f84d-df4e-4d49-b662-bcde71a8764f`.

# CSV

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

The **CSV** plugin supports reading and writing data in comma-separated values (CSV) format, as well as other similar formats that use a different delimiter such as tab-separated values (TSV).

Delimiter-based formats like CSV and TSV are simple and easy to use, but are limited in their ability to represent more complex data. For more advanced uses consider using JSON instead.

### Scripting

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

## Using CSV files

CSV and TSV files can be added as project files in the Project Bar and edited directly in Construct. As with other data formats, these can then be loaded by requesting them with the AJAX object.

## Using tab delimiters

Due to the use of the tab key as a shortcut in the editor, it can be difficult to type a tab in the Parameters Dialog for the delimiter parameter if you want to use tab-separated values. To make this easier the CSV plugin provides a *TabCharacter* expression which just returns a string with a single tab character in it, so it can be conveniently used as a delimiter parameter.

## Using array for storage

The CSV plugin merely parses and generates CSV. It uses an Array object to store the actual data, as CSV can conveniently be represented as a two-dimensional array, and it allows using the full features of the Array object to read and manipulate the data.

## CSV conditions

**On parse error**

Triggered when using the *Parse* action with invalid data, such as a quoted item that is missing its end quote. If a parse error occurs then no data will be read from the file and the specified Array object will be unmodified.

**On parse success**

Triggered when using the *Parse* action with valid data that is successfully parsed.

# CSV actions

### Generate CSV

Generate a string of CSV data using data from a specified two-dimensional Array object. A custom delimiter can be set for other delimiter-separated formats like tab-separated values. After this action the resulting string is available via the *GeneratedCSV* expression.

### Parse CSV

Read a string of CSV data in to a specified Array object. The resulting data will be stored as a two-dimensional array. A custom delimiter can be set for other delimiter-separated formats like tab-separated values. If any of the rows are different lengths, the width of the array will be the maximum row length. If the data is invalid for any reason, *On parse error* will be triggered; otherwise if valid *On parse success* will be triggered. The *Data type* parameter determines whether values are read as strings or numbers. Its possible values are:

- **Auto:** automatically determine whether values are strings or numbers, based on whether the string looks like a number. For example "hello" will be stored as a string, but "100" will be stored as a number.
- **String:** treat every value as a string. Note this means a number like "100" will be stored as a string with that sequence of characters rather than an actual number type.
- **Number:** treat every value as a number. Note this means strings like "hello", which are not valid numbers, will result in the special "Not A Number" (NaN) value.

# CSV expressions

### GeneratedCSV

After the *Generate CSV* action, this expression returns a string of the generated CSV data.

### TabCharacter

This expression merely returns a string with a single tab character in it. This can be useful to use as a delimiter parameter for tab-separated values (TSV), as a tab character can be difficult to type in the editor as it functions as a keyboard shortcut to move focus.

# DATE

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

The **Date** object provides a set of conditions and expressions for managing dates and times.

For an example of the Date object, open the Date & Time example.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Date object.

## Timestamps

The Date object mostly works with *timestamps*. These are the number of milliseconds since January 1st 1970 (also known as the Unix epoch), and is not affected by timezones or leap seconds. This is a standard way to reliably refer to a specific point in time in software development, and is sometimes referred to as the *Unix time*.

A timestamp is just a number, which makes it easy to store them in existing variables or compare them as you would any other number. However it's difficult for humans to make sense of a timestamp since it's just a single large number. The Date object provides expressions to convert between timestamps and human-readable dates and times, and other related tools.

## Date & time parts

A timestamp consists of the following parts, all of which are also numbers, with the given ranges and start points as noted:

- **Year:** the full four-digit year, e.g. 2020
- **Month:** the **zero-based** calendar month, e.g. 0 for January, 5 for June
- **Date:** the calendar date, i.e. the day of the month, from 1-31
- **Day:** the zero-based week day from 0-6, starting with Sunday, e.g. 2 for Tuesday
- **Hours:** the hours of the time, 0-23
- **Minutes:** the minutes of the time, 0-59
- **Seconds:** the seconds of the time, 0-59
- **Milliseconds:** the milliseconds of the time, 0-999

## Date Conditions

### Compare Timestamps

**Compare Timestamps**

Compare two timestamps. Equal timestamps refer to the same time, and a timestamp that is less than another precedes it in time.

---

**Compare Date Strings**

Compare two date strings with each other. The date strings are converted to their equivalent numeric timestamp before the comparison.

---

**Compare Timestamp parts**

Compare two timestamp parts with each other. The possible parts are Year (4 digit), Month (0 - 11), Date (1 - 31), Day (0 - 6), Hours (0 - 23), Minutes (0 - 59), Seconds (0 - 59), and Milliseconds (0 - 999).

---

**Compare date string parts**

Compare two date string parts with each other, they are converted to their equivalent numeric timestamp before the comparison. The possible parts are Year (4 digit), Month (0 - 11), Date (1 - 31), Day (0 - 6), Hours (0 - 23), Minutes (0 - 59), Seconds (0 - 59), Milliseconds (0 - 999).

## Date Expressions

---

**ToString(timeStamp)**

Convert a timestamp to a string representation including both date and time.

---

**ToDateString(timeStamp)**

Convert a timestamp to a string showing the corresponding date.

---

**ToTimeString(timeStamp)**

Convert a timestamp to a string showing the corresponding time.

---

**ToLocaleString(timeStamp)**

Convert a timestamp to a localized string representation including both date and time.

---

**ToLocaleDateString(timeStamp)**

Convert a timestamp to a localized string showing the corresponding date.

---

**ToLocaleTimeString(timeStamp)**

Convert a timestamp to a localized string showing the corresponding time.

**ToUTCString(timeStamp)**

Convert a timestamp to a string representation including both date and time in universal time.

**Parse(dateString)**

Parse a date string into the corresponding numeric timestamp. For the supported string formats, refer to the MDN documentation for Date.parse(), which is the underlying method used by this expression.

**ToTimerHours(milliseconds)**

Convert milliseconds to the equivalent amount of hours as they would be shown in a timer.

**ToTimerMinutes(milliseconds)**

Convert milliseconds to the equivalent amount of minutes as they would be shown in a timer (0-59).

**ToTimerSeconds(milliseconds)**

Convert milliseconds to the equivalent amount of seconds as they would be shown in a timer (0-59).

**ToTimerMilliseconds(milliseconds)**

Convert milliseconds to the equivalent amount of milliseconds as they would be shown in a timer (0-999).

**ToTotalHours(milliseconds)**

Convert milliseconds to the equivalent amount of hours, which may be a fractional value.

**ToTotalMinutes(milliseconds)**

Convert milliseconds to the equivalent amount of minutes, which may be a fractional value.

**ToTotalSeconds(milliseconds)**

Convert milliseconds to the equivalent amount of seconds, which may be a fractional value.

**Now**

Get the current timestamp (the number of milliseconds since January 1st 1970).

**ExportTimestamp**

Get the timestamp at the time the project was exported from Construct. In preview mode, this is the time the preview was launched.

---

**Get(year, month, day, hours, minutes, seconds, milliseconds)**

Return a timestamp by providing the individual parts of the date and time.

---

**TimezoneOffset**

Get the timezone offset of the local system.

---

**GetYear(timeStamp)**

Extract the 4 digit year from the provided timestamp in local time.

---

**GetUTCYear(timeStamp)**

Extract the 4 digit year from the provided timestamp in universal time.

---

**GetMonth(timeStamp)**

Extract the month (0 - 11) from the provided timestamp in local time.

---

**GetUTCMonth(timeStamp)**

Extract the month (0 - 11) from the provided timestamp in universal time.

---

**GetDate(timeStamp)**

Extract the date (1 - 31) from the provided timestamp in local time.

---

**GetUTCDate(timeStamp)**

Extract the date (1 - 31) from the provided timestamp in universal time.

---

**GetDay(timeStamp)**

Extract the day (0 - 6) from the provided timestamp in local time.

---

**GetUTCDay(timeStamp)**

Extract the day (0 - 6) from the provided timestamp in universal time.

---

**GetHours(timeStamp)**

Extract the hours (0 - 23) from the provided timestamp in local time.

**GetUTCHours(timeStamp)**

Extract the hours (0 - 23) from the provided timestamp in universal time.

**GetMinutes(timeStamp)**

Extract the minutes (0 - 59) from the provided timestamp in local time.

**GetUTCMinutes(timeStamp)**

Extract the minutes (0 - 59) from the provided timestamp in universal time.

**GetSeconds(timeStamp)**

Extract the seconds (0 - 59) from the provided timestamp in local time.

**GetUTCSeconds(timeStamp)**

Extract the seconds (0 - 59) from the provided timestamp in universal time.

**GetUTCMilliseconds(timeStamp)**

Extract the milliseconds (0 - 999) from the provided timestamp in local time.

**GetMilliseconds(timeStamp)**

Extract the milliseconds (0 - 999) from the provided timestamp in universal time.

**Difference(first, second)**

Calculate the difference between two timestamps.

**ChangeYear(timeStamp, year)**

Change the year (4 digit) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCYear(timeStamp, year)**

Change the year (4 digit) of the provided timestamp in universal, time and return as a new timestamp.

**ChangeMonth(timeStamp, month)**

Change the month (0 - 11) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCMonth(timeStamp, month)**

Change the month (0 - 11) of the provided timestamp in universal, time and return as a new timestamp.

**ChangeDate(timeStamp, date)**

Change the date (1 - 31) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCDate(timeStamp, date)**

Change the date (1 - 31) of the provided timestamp in universal time, and return as a new timestamp.

**ChangeDay(timeStamp, day)**

Change the day (0 - 6) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCDay(timeStamp, day)**

Change the day (0 - 6) of the provided timestamp in universal time, and return as a new timestamp.

**ChangeHours(timeStamp, hours)**

Change the hours (0 - 23) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCHours(timeStamp, hours)**

Change the hours (0 - 23) of the provided timestamp in universal time, and return as a new timestamp.

**ChangeMinutes(timeStamp, minutes)**

Change the minutes (0 - 59) of the provided timestamp in local time, and return as a new timestamp.

**ChangeUTCMinutes(timeStamp, minutes)**

Change the minutes (0 - 59) of the provided timestamp in universal time, and return as a new timestamp.

**ChangeSeconds(timeStamp, seconds)**

Change the seconds (0 - 59) of the provided timestamp in local time, and return as a new timestamp.

---

**ChangeUTCSeconds(timeStamp, seconds)**

Change the seconds (0 - 59) of the provided timestamp in universal time, and return as a new timestamp.

---

**ChangeMilliseconds(timeStamp, milliseconds)**

Change the milliseconds (0 - 999) of the provided timestamp in local time, and return as a new timestamp.

---

**ChangeUTCMilliseconds(timeStamp, milliseconds)**

Change the milliseconds (0 - 999) of the provided timestamp in universal time, and return as a new timestamp.

---

**FormatDateWithStyles(locale, timeStamp, dateStyle, timeStyle, hourFormat)**

Format the provided time stamp using the locale and optional styles. **"dateStyle"** can be any of *"full"*, *"long"*, *"medium"* or *"short"* and affects the date section of the final result, if an unsupported value is used, the date section will be omitted from the final output of the expression. **"timeStyle"** can be any of *"full"*, *"long"*, *"medium"* or *"short"* and affects the time section of the final result, if an unsupported value is used, the time section will be omitted from the final output of the expression. **"hourFormat"** can be either *"12"* or *"24"* and affects the formatting of the time section of the final result, if an unsupported value is used, the end result will be locale dependant.

---

**FormatDateWithComponents(locale, timeStamp, weekday, year, month, day, hour, minute, second, hourFormat)**

Format the provided time stamp using the locale and optional components. **"weekday"** can be any of *"long"*, *"short"* or *"narrow"*. **"year"** can be any of *"numeric"* or *"2-digit"*. **"month"** can be any of *"numeric"*, *"2-digit"*, *"long"*, *"short"* or *"narrow"*. **"day"** can be any of *"numeric"* or *"2-digit"*. **"hour"** can be any of *"numeric"* or *"2-digit"*. **"minute"** can be any of *"numeric"* or *"2-digit"*. **"second"** can be any of *"numeric"* or *"2-digit"*. **"hourFormat"** can be either *"12"* or *"24"* and affects the formatting of the time section of the final result, if an unsupported value is used the end result will be locale dependant.
Any of the components can be omitted from the final result by providing an unsupported value.

# DICTIONARY

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

The **Dictionary** object associates keys with values. Keys are string, and their associated value can be a number or a string. It is a data storage object - it does not do any spell checking or language-specific features.

Key names in the Dictionary object are always case sensitive. This means the key "SCORE" is considered different to the key "score".

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IDictionaryInstance script interface. (JavaScript and TypeScript have built-in support for Map which is a similar data structure, but this allows interacting with data used in event sheets.)

### Example

Suppose the number 100 is stored with the key "score", and the string "Joe" stored with the key "name". The result storage looks like the following table:

| Key | Value |
|---|---|
| name | Joe |
| score | 100 |

Retrieving the key "name" with `Dictionary.Get("name")` returns "Joe", and retrieving "score" likewise returns 100. Setting "score" to 50 will change the value for the key.

This is like storing data in instance variables or event variables, but since you can use strings as keys you can store any number of values.

Dictionaries are very efficient at retrieving values. Even if you have a dictionary with thousands of keys, it is still very fast to read a value. Arrays are typically much slower to search through (e.g. using the *IndexOf* expression), since they must scan through the entire array to locate elements.

## Designing dictionaries

You can use Construct's Dictionary Editor Paid plans only to set the initial contents of a dictionary. You can create a new dictionary data file as a project file from the Project Bar. At runtime you can load the project file with the AJAX object and use the Dictionary's *Load* action to read the data file from the AJAX's *LastData* expression.

## Dictionary conditions

**Compare value**

Compare the value stored for a key.

**Has key**

Check if a key exists in storage.

**Is empty**

True when there are no keys in storage.

**For each key**

Repeat the event once for each key in storage. The *CurrentKey* and *CurrentValue* expressions return the current key and its value respectively.

**Compare current value**

Only valid in a *For each key* event. Compare the value of the current key.

## Dictionary actions

**Add key**

Add a new key to storage, with a given value. If the key already exists, its value is updated.

**Clear**

Remove all keys from storage, making the object empty.

**Delete key**

Remove a key and its value from storage. If the key does not exist, this has no effect.

**Set key**

Update the value for a key which already exists. If the key does not exist, this has no effect. (Unlike *Add key*, the key will not be created.)

**Download**

Invokes a browser download of a file containing the Dictionary's contents in JSON format.

**Load**

Load all keys and values from JSON data previously retrieved from the Dictionary object using either the *Download* action, the *AsJSON* expression, or the AJAX object loading a project file.

> *Note this does not allow loading arbitrary JSON data - the data must be in a special format for Construct. If you want to load data from a project file, create it using the New - Dictionary menu option in the Project Bar.*

## Dictionary expressions

**Get(key)**

Return the value stored for a key, e.g. `Dictionary.Get("score")`. If the key does not exist, it returns 0.

**GetDefault(key, valueIfMissing)**

Return the value stored for a key, but if it is missing, return a different value instead. For example `Dictionary.GetDefault("name", "guest")` will return the value of the key "name" if it exists, otherwise it will return the string "guest".

**KeyCount**

Return the number of keys in storage.

**CurrentKey**

**CurrentValue**

In a *For each key* event, these return the key and its value (respectively) for the current key being iterated.

**AsJSON**

Return the contents of the Dictionary object in JSON format. This can be later loaded back with the *Load* action, sent to a server via AJAX, saved to disk, and so on.

# 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.

# FACEBOOK

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

The **Facebook** object allows you to integrate your game with Facebook. It can be used in any web-hosted game, not just in Facebook games. For example, you could have a game running on your own server. The Facebook object still allows you to integrate with user's Facebook accounts even when the user is playing the game on your website. However, it is still necessary to submit the app to Facebook so you have an App ID. Apps can be submitted to Facebook via Facebook Developers.

> *The Facebook plugin is **not compatible with Instant Games.** Use the Instant Games plugin instead.*

The Facebook object makes asynchronous requests. This means the game keeps running while operations which may take several seconds (like submitting a score) complete in the background. Therefore, the Facebook object works around the principle of an action which starts a task (like *Publish score*), which then triggers a condition when it has completed (like *On score submitted*). These may be a few seconds apart and the game continues to run in between, rather than freezing the game while it waits for completion. As with most asynchronous requests, actions may complete in a different order to that which they were made.

## Readiness

The Facebook object takes a moment to load on startup. Do not attempt to use the Facebook object before *On ready* has triggered or *Is ready* is true, otherwise the actions will be ignored.

You must enter the **App ID** for the Facebook object to successfully load, even in testing. If you don't enter it, the object will never become ready. This is because Facebook prevent you using the API unless you have a real App ID to give it.

## Logging in

The user must successfully log in with their Facebook account before they can use any other features of the Facebook object, such as posting to the user's wall. This is done with the *Log in* action, and since login can take a few seconds you must wait until *On user logged in* triggers before using any other features.

## Facebook Playable Ads

This plugin also provides the *On CTA click* action for Facebook Playable Ads. This action can be used independently of the other features, i.e. without having to log in first, or provide an app ID or any other properties.

# Facebook Properties

### App ID

The App ID given for your app in the Facebook Developers section.

### App Secret

This is only necessary if you plan to use the Scores API. Otherwise do not enter the app secret. The app secret can be found in the Facebook Developers section for your app and is currently necessary to enable the Scores API.

# Facebook conditions

### Is ready

True if the Facebook object is ready to be used. When false, the Facebook object is still loading the necessary scripts and all actions will be ignored.

### Is user logged in

True if the user is currently logged in with their Facebook account.

### On name available

Triggered shortly after *On user logged in*, when the expressions to get the user's name are available.

### On ready

Triggered when the Facebook object is ready to be used. Before this triggers, the Facebook object is still loading the necessary scripts and all actions will be ignored.

### On user logged in

Triggered when the user has successfully logged in. It also triggers on startup if the user has previously logged in, and they have returned to the app with a remembered login. This also triggers at the start of each layout throughout the game while the user is logged in.

### On user logged out

Triggered if the user logs out during the game.

### On hi-score

Triggers once for each hi-score after the *Request hi-score board* action. The *HiscoreName*, *HiscoreRank* and *Score* expressions contain the current hi-score information.

### On score submitted

Triggers when the *Publish score* action has successfully completed.

### On user top score available

Triggers when the *Request user top score* action has successfully completed. The *Score* expression has the user's top score.

## Facebook actions

### Log in

Open a popup window requesting the user to log in. This can only be called in a user-input event, such as a key press, button push, mouse click or touch screen tap. The user may cancel at the log in screen so logging in is not guaranteed to be successful. You must provide a comma-separated list of permissions to request. Some features of the Facebook plugin depend on the user approving certain permissions. Do not request permissions that your application does not need. You can find documentation on the available permissions in the Facebook developer's guide.

### Log out

Log the current user out from Facebook.

### On CTA click

**For Facebook Playable Ads.** In a Playable Ad, use this action when the user interacts with the call-to-action (CTA) in the ad. This action can be used independently of the rest of the plugin's features.

### Prompt to share link

Open a dialog prompting the current user to share a URL of your choosing on their wall. This does not require any permissions, but the user is free to cancel the action.

### Prompt to share this app

Open a dialog prompting the current user to share the URL to the current Facebook app on their wall. This does not require any permissions, but the user is free to cancel the action.

### Prompt wall post

Open a dialog prompting the current user to make a wall post. No content or links is added - it is an empty text box for the user to type anything they like. This does not require any permissions, but the user is free to cancel the action.

**Publish link**

Automatically publish a URL to the user's wall without any prompt. Descriptions and a thumbnail image can optionally be provided. This requires that the user has logged in with the *Publish to stream* permission.

**Publish wall post**

Automatically publish a message to the user's wall without any prompt. This requires that the user has logged in with the *Publish to stream* permission.

**Publish score**

Publish a numerical score to the hi-score board for the game. This requires that the user has logged in with the *Publish scores* permission. When the action completes, it triggers *On score submitted*.

**Request hi-score board**

Request the game's hi-scores. This requires that the user has logged in with the *Publish scores* permission. This action will repeatedly trigger *On hi-score* once per score on the hi-score board. It is up to you to find a way to display these scores; appending each hi-score to a text object is a simple way to display them.

**Request user top score**

Request the current logged in user's top score for this game. This requires that the user has logged in with the *Publish scores* permission. When the action completes, it triggers *On user top score available*.

## Facebook expressions

**FirstName**

The current user's first name. This is only set after *On name available* triggers.

**FullName**

The current user's full name. This is only set after *On name available* triggers.

**LastName**

The current user's full name. This is only set after *On name available* triggers.

**UserIDStr**

The current user's ID, which can be used to distinguish different users with the same name. This is only set if the user is logged in. Note: although this is a number, it is returned as a string since IDs can be larger numbers than Javascript can express.

---

**HiscoreName**

The current hi-score board entry name. This is only set in an *On hi-score* event.

---

**HiscoreRank**

The current hi-score board entry rank, from 1 down to the number of hi-score entries requested. This is only set in an *On hi-score* event.

---

**HiscoreUserIDStr**

Return the user ID of the current hi-score board entry. This is unique even if two people on the hi-score board have the same name. Note: although this is a number, it is returned as a string since IDs can be larger numbers than Javascript can express.

---

**Score**

Either the current hi-score in an *On hi-score* event, or the current user's score in *On user top score available*.

# FILE CHOOSER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/file-chooser

The **File chooser** plugin is a form control that allows the user to pick a file on their local system and load it. For example, a text file can be chosen and its contents read, or a picture chosen and the image displayed in-game. Some mobile browsers also allow a picture to be taken with the device camera instead of choosing an existing file, then the taken photo image file is passed as the chosen file.

The appearance of the file chooser button varies depending on the platform. Several browsers display a "Choose file" button.

### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Using files

Once a file has been selected, it can be accessed using a URL. While many URLs reference web resources on the Internet, the File Chooser plugin returns a special URL that refers to the local file. This URL typically starts with "blob:" (since the browser's term for an arbitrary set of data is a "blob") followed by some kind of unique ID. By accessing this URL the local file is accessed, without using any Internet connection, so this also works offline.

The file URL can be used like any other URL. For example, it can be requested by the AJAX object to load its contents as text, or passed to the Sprite object's *Load image from URL* action to display it.

## File Chooser properties

**Accept**

The file type filter to accept. This can be a comma-separated list of file extensions, e.g. ".png,.jpg,.jpeg", or a MIME type, such as *image/** for any image file, *video/** for any video file, and so on. If left empty, the file chooser will be able to choose any kind of file.

*For best cross-platform compatibility, where possible use a MIME type instead of a file extension. For example prefer using application/json instead of the .json file extension.*

**Select**

Whether to allow selecting a single file only, or multiple files in one go.

**Initially visibile**

Whether the control is initially visible or hidden.

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

**Class Optional**

An optional *class* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

## File Chooser conditions

See common conditions for features shared between form control objects.

**On changed**

Triggered when a file or set of files has been chosen from the input control.

## File Chooser actions

See common actions for features shared between form control objects.

**Clear**

Reset the control to its initial state with no selection, clearing any prior picked file.

**Click**

Acts as if the file chooser was clicked, which brings up the system file picker. This allows the button to be completely customised, as the actual File Chooser object can be moved offscreen, and a custom sprite or other object used to activate the *Click* action of File Chooser instead.

*Normally this action can only be used in a user input trigger, e.g. in a mouse click event, touch start event, etc.*

---

**Release file**

Release a previously returned file URL. This allows the browser to release memory for the file. If the user chooses lots of files or chooses them regularly this action should be used to release the files after they are no longer needed.

## File Chooser expressions

---

**FileCount**

The number of chosen files. If *Select* is *Single*, this is always 1. If *Multiple* then this returns the number of chosen files.

---

**FileNameAt(index)**

Return the name of the chosen file at an index. In *Single* select mode, the index should always be 0, since only one file can be chosen.

---

**FileSizeAt(index)**

Return the size of the chosen file in bytes at an index. In *Single* select mode, the index should always be 0, since only one file can be chosen.

---

**FileTypeAt(index)**

Return the MIME type of the chosen file at an index. The MIME type is the standardised Internet media type of the file contents. For example, this can be *image/png* for a PNG image, *image/jpeg* for a JPEG image, or *text/plain* for a plaintext file. In *Single* select mode, the index should always be 0, since only one file can be chosen.

---

**FileURLAt(index)**

The URL to access the local file's contents at an index. For more information see *Using files* above. In *Single* select mode, the index should always be 0, since only one file can be chosen.

# FILE SYSTEM

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

The **File System** plugin allows read and write access to files and folders on the local system.

For a demonstration of using file system features, see the Text Editor example.

### Scripting

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

## Browser/platform support

For the File System plugin features to work, the browser must support the full File System Access API. (Note that *Origin Private File System* or OPFS is not used by the File System plugin.) As of November 2024 this is only supported in Chrome and Edge (and most other Chromium-based browsers) on desktop. Notably this means Firefox and Safari are not currently supported. However it is also supported in the Windows WebView2, macOS WKWebView and Linux CEF exporters.

> *The macOS WKWebView export option is based on the Safari browser engine, but Construct has extended it to support the File System plugin features even though the Safari browser does not support the File System Access API. Therefore you can rely on File System plugin features working in macOS WKWebView exports.*

## Browser permissions model

Browsers impose security restrictions to ensure that general web browsing is safe. Therefore they start with **no access to local files or folders** at all. The only way to gain access to local files and folders in a browser is to show a **picker** - a "save file" dialog, an "open file" dialog, or a "pick folder" dialog.

**Desktop exporters** like Windows WebView2 **are an exception to this rule** and can have direct access to certain special folders like the user's documents folder without any dialogs or permission prompts - see below for more details. However in a browser, once the user has successfully completed a picker by making a selection and clicking OK, then permission to access the chosen files or folders is granted.

> *In browsers, pickers are also only allowed to be shown in a user input trigger, such as upon clicking a button, or starting a touch. This is another security measure to ensure pickers*

*can't be shown at unexpected times, such as immediately upon loading the page.*

In some circumstances browsers will also show a **permission prompt** to access a file. For example choosing a file with an "open file" picker allows reading the file without any further permission prompt, as the picker grants read access. However if later on an attempt is made to write to the opened file, browsers will normally show a permission prompt to verify that the user is willing to allow their previously chosen file to be modified. (This also only applies to browsers - desktop exporters won't show permission prompts.)

To help avoid the need to keep showing pickers, Construct is also able to **save the permission to access previously chosen files** and folders. The *Has picker tag* condition will be true if a picker completed in a prior session and the same chosen files or folders can still be accessed. Browsers may also show permission prompts when accessing files or folders from prior sessions.

## Picker tags and file tags

Tags are short pieces of text used to identify different uses of the same feature. The File System plugin uses two kinds of tags. **Picker tags** identify different picker dialogs, such as two different uses of an "open file" dialog. Each successfully completed picker grants access to the chosen files or folders, which are then referred to by the picker tag.

Reading or writing a file then uses a **file tag**. This identifies the actual file system operation of reading or writing to a chosen file or folder. The file tag is optional as it's only needed if you want to track when file system operations complete or encounter an error - if you don't care about the result, you can leave the file tag empty.

The reason two kinds of tags are used is because one picker may grant access to multiple files or folders. For example one folder picker can grant access to a folder, in to which two files are then written. In this case there is one picker tag identifying the folder the user chose, and then two file tags identifying the separate write operations.

In summary picker tags are required in order to identify the files and folders the user chose, but file tags are optional and are used to identify when reading or writing specific files or folders completes or encounters an error.

## Accessing known folders

In Windows WebView2, macOS WKWebView and Linux CEF exports, the browser permission model can be bypassed to directly access folders such as the user's documents folder. This is done by using pre-defined folder picker tags like "<documents>". These act like the folder was already picked on startup. You can check if these are available on startup by using the *Has picker tag* condition. If available then actions like *Write text file* can write to picker tag "<documents>" folder path "test.txt", in order to write "text.txt" in the user's Documents folder, without showing any picker dialog or permission prompts at all.

The full table of pre-defined picker tags for known folders is listed below. Note not all of these are guaranteed to be available: a system may not have some types of folders, in which case *Has picker tag* will be false even in a desktop export.

## macOS App Sandbox

Note that in macOS WKWebView exports, if you enable the App Sandbox, then macOS creates a *container* for your app. This effectively gives your app its own folders like 'Documents', 'Pictures' and so on, inside a special app-specific path; the app won't actually access the folders the user would normally see in Finder. This is a privacy measure to make sure a sandboxed app can only access files saved by the same application, rather than being able to access all their files. When the sandbox is enabled, you can enable read-only or read-write permission for the Pictures, Movies and Downloads folder which provides access to the real folders shown in Finder. Alternatively if you disable the sandbox then the app will access all the known folders normally, with access to all the user's files.

## Known folders table

Note the example paths shown in this table are for illustrative purposes only - the specific paths accessed may be different depending on the username, and may be in entirely different locations depending on the application install directory or system configuration.

| Picker tag/platform | Description/path |
|---|---|
| **<app>** | Folder main application executable belongs to. |
| *Windows* | C:\Program Files\My app |
| *macOS* | /path/to/executable |
| *Linux* | /path/to/executable |
| **<web-resource>** | The folder that contains web resource files such as index.html, typically in a *www* subfolder to the app folder. |
| *Windows* | C:\Program Files\My app\www |
| *macOS* | /path/to/executable/www |
| *Linux* | /path/to/executable/www |
| **<current-app-data>** | The local app data folder for this specific application. |
| *Windows* | C:\Users\username\AppData\Local\myapp |
| *macOS* | /Users/username/Library/Application Support/com.myapp.id |
| *Linux* | /home/username/.local/share/com.myapp.id |
| **<local-app-data>** | The main system local app data folder for the current user. |
| *Windows* | C:\Users\username\AppData\Local |
| *macOS* | /Users/username/Library/Application Support |
| *Linux* | /home/username/.local/share |
| **<roaming-app-data>** | The main system roaming app data folder for the current user. |
| *Windows* | C:\Users\username\AppData\Roaming |
| *macOS* | *Not supported* |
| *Linux* | *Not supported* |
| **<desktop>** | User's desktop folder. |
| *Windows* | C:\Users\username\Desktop |
| *macOS* | /Users/username/Desktop |
| *Linux* | /home/username/Desktop |
| **<documents>** | User's documents folder. |
| *Windows* | C:\Users\username\Documents |
| *macOS* | /Users/username/Documents |
| *Linux* | /home/username/Documents |
| **<downloads>** | User's downloads folder. |
| *Windows* | C:\Users\username\Downloads |
| *macOS* | /Users/username/Downloads |
| *Linux* | /home/username/Downloads |

| Picker tag/platform | Description/path |
|---|---|
| **<pictures>** | User's pictures/photos folder. |
| *Windows* | C:\Users\username\Pictures |
| *macOS* | /Users/username/Pictures |
| *Linux* | /home/username/Pictures |
| **<profile>** | Parent folder for all files and folders relating to the current user. |
| *Windows* | C:\Users\username |
| *macOS* | /Users/username |
| *Linux* | /home/username |
| **<saved-games>** | User's folder for games to save their current progress to. |
| *Windows* | C:\Users\username\Saved Games |
| *macOS* | *Not supported* |
| *Linux* | *Not supported* |
| **<screenshots>** | User's folder for games and applications to save screenshots to. Note this may be a subfolder to the pictures folder. |
| *Windows* | C:\Users\username\Pictures\Screenshots |
| *macOS* | *Not supported* |
| *Linux* | *Not supported* |
| **<videos>** | User's folder for games and applications to save recorded videos to. |
| *Windows* | C:\Users\username\Videos |
| *macOS* | /Users/username/Movies |
| *Linux* | /home/username/Videos |
| **<dropped-files>** | Used only to reference files dragged and dropped in to the window, when *On dropped files* triggers. This does not actually refer to a folder on disk, but allows accessing the dropped files as if the user picked a read-only folder containing them. |

# File system conditions

### Desktop features supported

Detect whether desktop-specific features of this plugin are available. This is only the case when running after using a supported desktop export option. When true, the known folder pickers like "<documents>" can be used (providing they are available - use the *Has picker tag* condition to check), as well as the desktop-specific actions like *Run file* and *Shell open*.

### Has picker tag

True if a given picker tag has been remembered from a previous session, or if a known folder tag is available. In this case the picker tag can still be referred to for file system operations, such as to read a previously chosen file, without having to show a picker again.

### On picker complete

### On picker error

Triggered after the *Show folder picker*, *Show open file picker* or *Show save file picker* actions with a matching picker tag, depending on the result of the picker. A picker is completed when the user successfully chooses a file or folder and access is then granted to the selection. Cancelling a picker, or otherwise being unable to show a picker (e.g. due to not being in a user input trigger), will trigger *On picker error*.

### Is supported

True if file system features are supported. This depends on support for the File System Access API in the browser. If false then none of the features of the plugin will work.

**On file operation complete**

**On file operation error**

Triggered after any file or folder operation such as reading a file or creating a folder, with a matching file tag completes/fails.

**On any file operation complete**

**On any file operation error**

Triggered after any file or folder operation such as reading a file or creating a folder completes/fails, regardless of its file tag. The associated file tag can be retrieved with the *FileTag* expression.

**On dropped files**

Triggered when the user drags and drops files in to the window. The dropped files can be accessed using the special picker tag `<dropped-files>`. This allows accessing the dropped files as if the user picked a folder containing the files. The *FileCount* and *FileNameAt* expressions can also be immediately used to identify the number and names of the dropped files without having to use the *List content* action. The dropped files are read-only.

**Has file/folder**

Use this after the *List content* action to check whether a given file or folder exists in the returned list of files and folders. Note that this checks for the specified type of item - so for example if checking whether a folder exists, and the given name does exist but is actually a file, then the condition will be false. You must choose whether to check for the file name case-sensitively; note that typically on Windows and macOS file names are not case sensitive, but on Linux they are.

# File system actions

**Add accept type**

Add a file type to be shown in the next open file or save file picker. By default the pickers will show all kinds of files; adding an accept type defaults to filtering by those kinds of files, and adding multiple accept types allows the user to switch between different filters. The accept type must specify a MIME type, such as "text/plain" for text files or "image/png" for PNG images. A list of file extensions can also be provided in case they cannot be deduced from the MIME type. These must begin with a dot and multiple file extensions can be specified separated by semicolons, e.g. ".png;.jpg" indicates to allow both .png and .jpg file extensions. A description can also be provided which may be shown in the picker. When the next picker is shown, the list of added accept types is cleared and must be added again for another picker; typically you should use this action immediately prior to showing a picker.

**Show folder picker**

.

Show a folder picker allowing the user to choose a folder on their local system. The *Picker tag* identifies the chosen folder if it is completed successfully. The *Mode* determines what kind of permission prompt the user is shown.

> *Files can be written to a read-only folder, but the browser will show another permission prompt. Obtaining read & write permission initially will avoid later permission prompts.*

The *Picker ID* is an optional extra identifier for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

---

### Show open file picker

Show an open file picker allowing the user to choose one or multiple files on their local system to open. The *Picker tag* identifies the chosen file or files if it is completed successfully, and grants read access. (The files can later be written to, but the browser will show another permission prompt.) *Show accept all* sets whether to show an accept type that shows all kinds of files; if disabled then the *Add accept type* action must be used beforehand. *Multiple* can be enabled to allow the user to select multiple files in the open file picker.

> *When opening multiple files, the selection is treated like an empty folder with just the chosen files in it. The FileCount and FileNameAt expressions will return the list of chosen files, and these names can be used in the Folder path parameter when reading or writing files.*

The *Picker ID* is an optional extra identifier for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

---

### Show save file picker

Show a save file picker allowing the user to choose a file on their local system to save to. The *Picker tag* identifies the chosen file if it is completed successfully, and grants write access.

> *Note that the chosen file is erased, so cannot be read from. It is expected that the file contents will only be written to.*

*Show accept all* sets whether to show an accept type that shows all kinds of files; if disabled then the *Add accept type* action must be used beforehand. The *Suggested name* is used as the initial filename choice to save to in the picker. The *Picker ID* is an optional extra identifier for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

### Read text file

### Read binary file

Read the contents of a file from a previously completed picker. The file or folder to read from is identified by the *Picker tag*. The *Folder path* is optional and is only used for folder pickers or from a open picker with *Multiple* enabled, and specifies the filename to be read, e.g. "file1.txt" or "subfolder/file2.txt" when using a folder picker. When using a save file picker or an open file picker for a single file, leave the *Folder path* empty as it is not used, as it will read the single file chosen by the picker. The *File tag* is optional and allows identifying when the read operation completes or fails with the *On file operation complete/error* triggers. In the case of reading a text file, the *FileText* expression is set to the read text content when completed. In the case of reading a binary file, the read file contents will be placed in the chosen Binary Data object when completed.

### Write text file

### Write binary file

Write text or binary data to a file from a previously completed picker. The file or folder to write to is identified by the *Picker tag*. The *Folder path* is optional and is only used for folder pickers or from a open picker with *Multiple* enabled, and specifies the filename to be written to, e.g. "file1.txt" or "subfolder/file2.txt" when using a folder picker. When using a save file picker or an open file picker for a single file, leave the *Folder path* empty as it is not used, as it will write to the single file chosen by the picker. The *File tag* is optional and allows identifying when the write operation completes or fails with the *On file operation complete/error* triggers. In the case of writing a text file, the given *Text* will be written to the file, and the text content can optionally be appended to the end of the existing file.

> *Note that save file pickers erase the chosen file, so appending is only useful after using an open file picker.*

In the case of writing a binary file, the contents of the chosen Binary Data will be written to the file.

### Copy file

Only applies to folder pickers. Copy a file within a previously picked folder. The *Source path* specifies an existing file to copy, e.g. "subfolder/file1.txt". The *Destination path* specifies where to create a copy; this will either create a new file if it doesn't exist, or overwrite an existing file if it already exists. The *File tag* is optional and allows identifying when the copy operation completes or fails with the *On file operation complete/error* triggers.

### Create folder

Only applies to folder pickers. Create a subfolder within a previously picked folder. The *Folder path* specifies the folder to create. This can refer to multiple subfolders which will all be

created, e.g. "subfolder/otherfolder". The *File tag* is optional and allows identifying when the folder creation operation completes or fails with the *On file operation complete/error* triggers.

---

**Delete**

Only applies to folder pickers. Delete a file or folder within a previously picked folder. The *Folder path* specifies the file or folder to delete, e.g. "subfolder/file1.txt" or "subfolder". The *Recursive* option applies only when *Folder path* identifies a folder: if enabled it will delete all files and folders within the identified folder as well as the folder itself, but if disabled it will only successfully delete the folder if it is already empty. The *File tag* is optional and allows identifying when the delete operation completes or fails with the *On file operation complete/error* triggers.

---

**List content**

Only applies to folder pickers. Retrieves a list of all files and folders within a previously picked folder. The *Folder path* identifies a subfolder to list the contents for, or can be left empty to list the contents of the originally picked folder. Enabling *Recursive* will also list all files and folders through subfolders of the specified folder.

> *Recursive listing may be slow with a very large folder that contains thousands of files/subfolders.*

The *File tag* is optional and allows identifying when the listing operation completes or fails with the *On file operation complete/error* triggers. Once completed, the available files and folders can be accessed with the *FileCount*, *FileNameAt*, *FolderCount* and *FolderNameAt* expressions. When *Recursive* was enabled, the listed file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.

---

**Move file**

Only applies to folder pickers. Move a file within a previously picked folder. The file can be moved from one subfolder to another, or if it stays in the same folder, it renames the file. The *Source path* specifies the file to move, e.g. "subfolder/file1.txt". The *Destination path* specifies where to move (or rename) the file to. The *File tag* is optional and allows identifying when the copy operation completes or fails with the *On file operation complete/error* triggers.

---

**Run file**

Only supported in desktop exports when the *Desktop features available* condition is true. This action attempts to execute the provided file path. Command-line arguments can also be optionally provided. This can be used to run another program. Uniquely for this action the folder picker parameter can be left empty, in which case the provided file path is run directly; this can be useful to run system executables like "cmd.exe" on Windows.

### Shell open

Only supported in desktop exports when the *Desktop features available* condition is true. This action requests that the operating system shell (the component that handles the system user interface) opens the specified file. Typically this launches the default app associated with the file type and then opens the file in it, similar to double-clicking the file in the system file explorer.

## File system expressions

### FileCount

### FileNameAt(index)

Retrieve the number and filenames of available files. The available list of files is updated in the following situations:

- After an open file picker, to identify the available filenames (which can be multiple filenames if the *Multiple* option was enabled, otherwise it is just one file)
- After the *List content* action for listing the contents of a folder. If the *Recursive* option was enabled, file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.
- After *On dropped files* triggers, to identify the available dropped files
- After a save file picker to identify the chosen saved filename (which is just one file)

### FolderCount

### FolderNameAt(index)

Retrieve the number and names of available folders. The available list of folders is updated in the following situations:

- After the *List content* action for listing the contents of a folder. If the *List content* action *Recursive* option was enabled, file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.
- After a folder picker has completed, to identify the name of the chosen folder (which is just one folder).

### FileText

After a *Read text file* action completes successfully, the contents of the read file.

*Note that this is replaced when the next Read text file action completes, so it is best to use it immediately after reading a file.*

---

**FileTag**

In a file system operation trigger such as *On any file operation complete*, this returns the file tag of the associated file operation.

---

**FolderPath**

In a file system operation trigger working within a folder, this returns the folder path of the associated file operation.

---

**PickerTag**

In a trigger, this returns the picker tag associated with the picker or file system operation.

# FLOWCHART CONTROLLER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/flowchart-controller

The **Flowchart Controller** plugin allows flowcharts to be controlled in event sheets.

Some of the common uses of the plugin include the following:

- Create instances of flowcharts using the **Start flowchart** or **Start flowchart (by name)** actions.
- Use the **Set flowchart** action to set an existing flowchart instance as the current one, affecting which flowchart other actions, conditions and expressions act upon.
- Use the **Go to** actions to traverse a flowchart.
- Use the **Node entered** and **Node exited** triggers to perform other project-specific actions depending on the current state of a flowchart.
- Query information from the current node of the current flowchart using expressions.

### Flowchart instances

Use the *Start flowchart* action to create an *instance* of a flowchart. This is essentially the current state of a flowchart, including the current node. Sometimes only one instance of a flowchart is necessary, which is simpler to use. However it is possible to create multiple instances of the same flowchart, and have each instance track its current node separately. This allows for more advanced uses, such as having every instance of a Sprite object have its own flowchart instance and be able to proceed through the flowchart independently, rather than sharing one flowchart state for all Sprite objects.

The *End flowchart* action is the counterpart of *Start flowchart*, and destroys a flowchart instance so it no longer consumes memory.

> *Look for the Flowcharts and Containers example for a quick look at how to combine Flowcharts with Containers to give individual instances their own flowchart instance.*

## Flowchart Controller Conditions

**On any flowchart entered**

Triggered when any flowchart is entered. This will happen after using the **Set Flowchart** action or after using one of the actions to start a new flowchart and using the parameter to immediately set it as the current one.

**On any node entered**

Triggered when any node is entered in the current flowchart.

**On tagged node entered**

Triggered when a node with the specified tags is entered in the current flowchart. All the passed tags must match for the condition to be fulfilled.

**On any node entered in flowchart**

Triggered when any node is entered in a flowchart with the specified tag.

**On tagged node entered in flowchart**

Triggered when a node with the specified tags is entered in a flowchart with the specified tag. All the passed node tags must match for the condition to be fulfilled.

**On any flowchart exited**

Triggered when any flowchart is exited.

**On any node exited**

Triggered when any node in the current flowchart is exited.

**On tagged node exited**

Triggered when a node with the specified tags in the current flowchart is exited. All the passed tags must match for the condition to be fulfilled.

**On any node exited in flowchart**

Triggered when any node in the specified flowchart is exited.

**On tagged node exited in flowchart**

Triggered when a node with the specified tags in the specified flowchart is exited. All the passed node tags must match for the condition to be fulfilled.

**Is at start node**

Check if the current flowchart is at the start node.

**Is at start node in flowchart**

Check if the specified flowchart is at the start node.

### Has flowchart

Check if a flowchart with the specified tag was already created by either **Start flowchart** or **Start flowchart (by name)**.

> *Note this is false if the flowchart was already released by **End flowchart** or **End flowchart by tag** actions.*

### Compare output count

Compare the output count of the current node in the current flowchart.

### Compare output name

Compare the output name in the current node of the current flowchart, using an index to choose the output.

### Compare default output name

Compare the output name in the current node of the current flowchart, checking against the default output. Only works if a default output has been set.

### Compare output value

Compare the output value in the current node of the current flowchart, using an index or name to choose the output.

### Compare default output value

Compare the output value in the current node of the current flowchart, checking against the default output. Only works if a default output has been set.

### Compare node tags

Compare the tags of the current node in the current flowchart. In the case of passing multiple tags as a space separated string, the current node must contain all the passed tags for the condition to be fulfilled.

### Compare node tag count

Compare the tag count of the current node in the current flowchart.

### Compare flowchart tag

Compare the tag of the current flowchart.

**Has output**

Check if the current node of the current flowchart has an output, using an index or a name to choose the output.

**Has default output**

Check if the current node of the current flowchart has a default output.

**Output name match regex**

Check if an output's name in the current node of the current flowchart matches a regular expression, using an index to choose the output.

**Output value match regex**

Check if an output's value in the current node of the current flowchart matches a regular expression, using an index or name to choose the output.

**For Each Output**

Run a loop for each output in the current node of the current flowchart.

## Flowchart Controller Actions

**Start flowchart**

**Start flowchart (by name)**

Starts a new flowchart instance. Use the **Start node tag** parameter so the flowchart starts in that node, if left empty the flowchart's starting node set in the editor is used. The **Flowchart tag** parameter assigns the flowchart instance a tag to later identify it elsewhere in the event sheet. The **Set as current** parameter is a shortcut to immediately set the new flowchart as the current one.

> *Starting a flowchart with a previously used tag will reset the existing flowchart!*

> *In the case of passing multiple tags as a space separated string to **Start node tag**, all the tags must match for a node to be used as the starting position.*

> *If there is more than one node that matches **Start node tag**, the first one found will be used.*

**Set flowchart**

**Set flowchart**

Set the flowchart with the specified tag as the current one. All actions, conditions and expressions used after this will refer to the specified flowchart, except for the case a tag is provided to get information from a specific flowchart instance other than the current one.

---

**End flowchart**

Release the current flowchart.

---

**End flowchart by tag**

Release the flowchart with the specified tag.

---

**Reset flowchart**

Reset the current flowchart, making the default initial node the current one.

---

**Reset flowchart by tag**

Reset the specified flowchart, making the default initial node the current one in that flowchart.

---

**Go to next node**

Go to the next node, using an index or a name to choose the output to follow.

---

**Go to default node**

Go to the next node, following the default output. If no default output has been set, nothing will happen.

---

**Go to node**

Go to an arbitrary node in the current flowchart using tags to find it. When passing multiple tags as a space separated string, a node must match all the tags before it can be visited.

---

**Go to previous node**

Go to the previous node of the current flowchart.

---

**Go to parent node**

Go to the parent node of the current node in the current flowchart, using an index or tags to find the parent node. If the node only has one parent the action will just move the flowchart to that node. When passing multiple tags as a space separated string, the parent node must match all the tags before it can be visited. If there are multiple matches, the first valid one is used.

## Flowchart Controller Expressions

**FlowchartTag**

Return the tag of the current flowchart.

**NodeTag**

Return the full tag string of the current node in the current flowchart.

**NodeTagAt(Index)**

Return a tag of the current node in the current flowchart using a 0-based index to look it up.

**NodeTagCount**

Return the amount of tags of the current node in the current flowchart.

**OutputCount**

Return the output count of the current node in the current flowchart.

**OutputNameAt(Index)**

Return the output name at the specified index in the current node of the current flowchart.

**OutputNameDefault**

Return the default output name in the current node of the current flowchart. If no default output is set, an empty string is returned.

**OutputValue(IndexOrName)**

Return the output value at the specified index or name in the current node of the current flowchart.

**OutputValueDefault**

Return the default output value in the current node of the current flowchart. If no default output is set, an empty string is returned.

**ParentCount**

Return the parent count of the current node in the current flowchart.

**ParentTag(ParentIndex)**

Return the full parent tag string of the current node in the current flowchart, specifying a parent index.

---

**ParentTagAt(ParentIndex, TagIndex)**

Return a parent tag of the current node in the current flowchart, specifying a parent index and a 0-based index to look up the tag.

---

**ParentIndex(ParentTag)**

Return the parent node index of the current node in the current flowchart, specifying a parent tag.

---

**NodeTagInFlowchart(FlowchartTag)**

Return the full tag string of the current node in the specified flowchart.

---

**NodeTagAtInFlowchart(FlowchartTag, TagIndex)**

Return a tag of the current node in the specified flowchart using a 0-based index to look up the node tag.

---

**NodeTagCountInFlowchart(FlowchartTag)**

Return the tag count of the current node in the specified flowchart.

---

**OutputCountInFlowchart(FlowchartTag)**

Return the output count of the current node in the specified flowchart.

---

**OutputNameAtInFlowchart(Index, FlowchartTag)**

Return the output name at the specified index in the current node of the specified flowchart.

---

**OutputNameDefaultInFlowchart(FlowchartTag)**

Return the default output name in the current node of the specified flowchart. If no default output is set, an empty string is returned.

---

**OutputValueInFlowchart(IndexOrName, FlowchartTag)**

Return the output value at the specified index or name in the current node of the specified flowchart.

---

**OutputValueDefaultInFlowchart(FlowchartTag)**

Return the default output value in the current node of the specified flowchart. If no default output is set, an empty string is returned.

---

**ParentCountInFlowchart(FlowchartTag)**

Return the parent count of the current node in the specified flowchart.

---

**ParentTagInFlowchart(ParentIndex, FlowchartTag)**

Return the full parent tag string of the current node in the specified flowchart using an index to pick the parent.

---

**ParentTagAtInFlowchart(ParentIndex, FlowchartTag, tagIndex)**

Return a parent tag of the current node in the specified flowchart using an index to pick the parent and a 0-based index to look up the parent tag.

---

**ParentIndexInFlowchart(ParentTag, FlowchartTag)**

Return the parent index of the current node in the specified flowchart using a tag to pick the parent.

---

**CurOutputName**

Return the current output name in a **For Each Output** loop.

---

**CurOutputValue**

Return the current output value in a **For Each Output** loop.

---

**CurIsDefault**

Check if the current output in a **For Each Output** loop is the default one.

*The expressions that accept a flowchart tag are useful because they offer a way to get information from a specific flowchart instance without executing any existing **node change** or **flowchart change** triggers.*

# FUNCTION

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

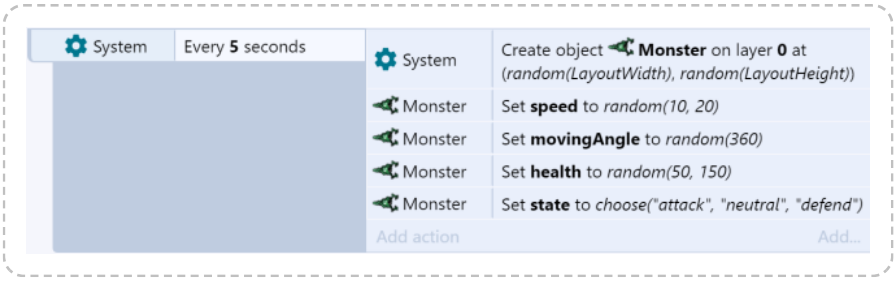
***The Function plugin has been replaced with a new built-in functions feature and is now deprecated. This manual entry is provided for archival reasons only and may be removed in future.***

The **Function** object can run a different event (*On function*) in an action (*Call function*). This is analogous to functions in traditional programming languages. Using functions can help you organise events and avoid having to duplicate groups of actions or events.

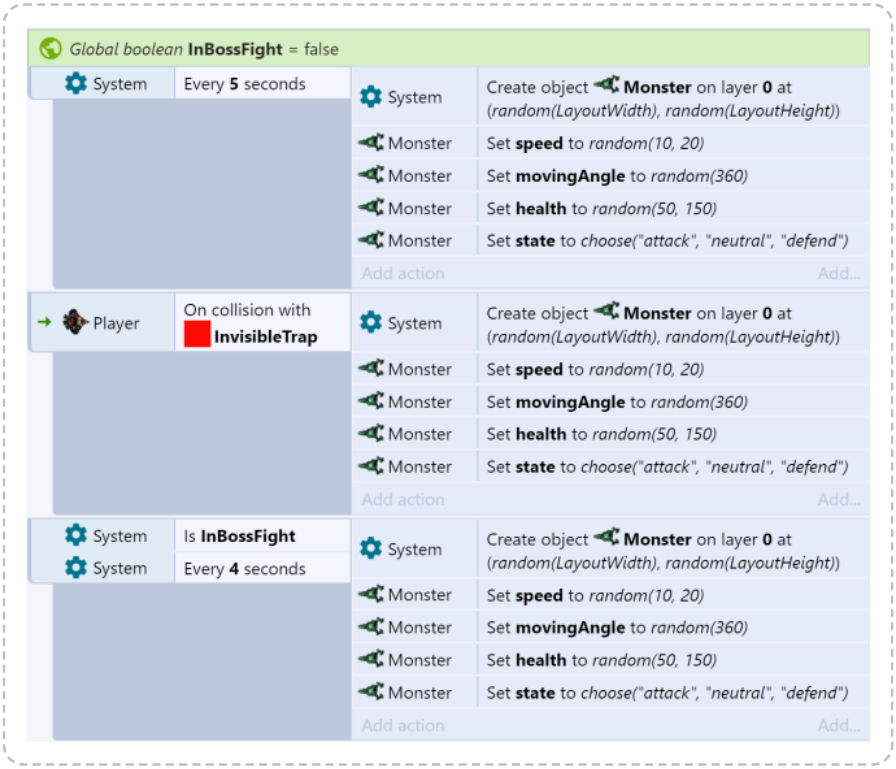
## About functions

The main purpose of the Function object is using the *Call function* action. This takes the name of a function (e.g. *Call function "CreateEnemy"*). The action then triggers the corresponding *On function* event (e.g. *On function "CreateEnemy"*), running the event's actions and any sub-events, before returning to the original *Call function* action and continuing from where it was.

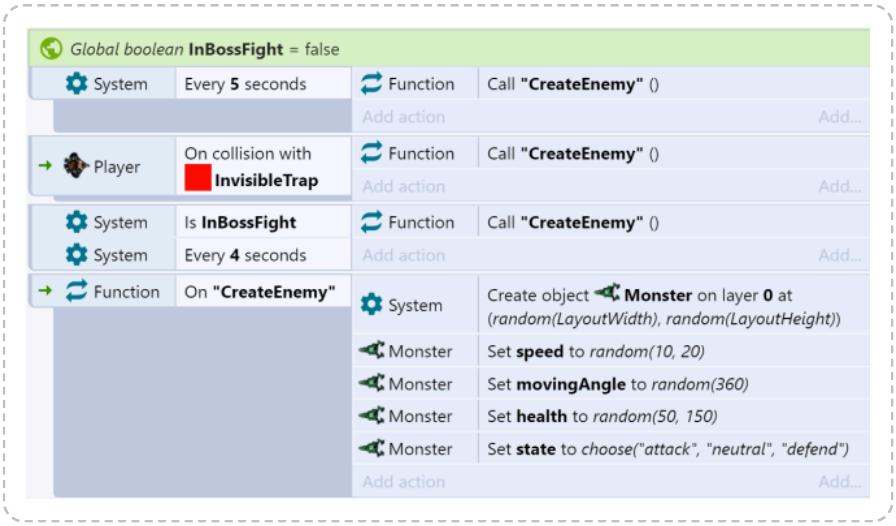
As another example, suppose you create an enemy with random stats in a game every 5 seconds using this event:




Suppose there are two other events where you want to create an enemy the same way: one when a player walks in to a trap, and another one every 4 seconds when in a boss fight. Without functions, you may have to copy-and-paste the actions multiple times, like this:




Notice this is becoming inconvenient. There may be times you need to repeat the actions in even more places. If you want to make a change, you then have to find every place you repeated the actions, and repeat the change. We can remove the repetition using functions. By creating a *CreateEnemy* function which has the repeated actions, we can replace all the repeated actions with a *Call function* action like this:




This works identically to the previous events, but is much shorter and more convenient. We can use *Call "CreateEnemy"* action anywhere we want to create an enemy, and it uses the same set of actions in the *On "CreateEnemy"* event.

It is often useful to split many parts of your events in to functions like this, so they can be conveniently re-used across event sheets.

## Parameters

When calling a function, you can also pass *parameters*. These are simply numbers or strings that are made available to the function. For example, the *CreateEnemy* function from the previous example could be modified to take two parameters: the X and the Y co-ordinates at which to create the enemy. This helps functions to be made more general purpose by using extra information from the action calling the function.

To add a parameter to a function call, click the **Add parameter** link that appears in the Parameters dialog when editing the *Call function* action. This is a special link that only appears for this action in the Function object. Inside an *On function* event, you can then use the *Param* expression with the zero-based index of the parameter to retrieve the corresponding value.

## Advanced function features

Like in programming languages, the Function object supports the following:

- Functions calling other functions
- Functions calling themselves (recursion)
- Returning values from functions
- Calling functions from expressions (which also returns the return value)

Note that functions calling other functions or recursing create a new "stack" of local variables. In other words, like in programming languages, local variables are unique at each level of function call. This does not apply to static local variables or global variables.

Also note the Function object logs to the browser console if it is used incorrectly, such as calling a non-existent function or accessing a parameter that was not passed. This can help identify problems using functions in large projects.

## Returning values from functions

Functions can also return a result. For example, a *factorial* function could calculate the mathematical result and return it. In an *On function* event, the return value can be set using the *Set return value* action.

If the event was called using the *Call function* action, the returned value is afterwards available using the *ReturnValue* expression. Functions can also be called directly from an expression using the *Call* expression; in this case the return value is automatically returned as the result of the *Call* expression.

## JavaScript integration

It is strongly recommended to use the Addon SDK to integrate JavaScript code with Construct. However it is possible to trigger a function in the Function object from JavaScript code using the following function:

```
if (self.c2_callFunction)
    self.c2_callFunction("name", ["param1", "param2"]);
```

(The name still refers to C2 for legacy reasons.) Note if the Function object is not included in a project, the `c2_callFunction` function will not exist, so the if check is necessary before using it. The function with the given "name" is triggered synchronously. Parameters are optional and can be omitted, but must be provided as an array in the second argument, and parameters may only be string or number values (any other types will return as 0 in Construct). The `c2_callFunction` method also returns the return value set in Construct (if any), and also can only return a string or number.

## Function conditions

### Compare parameter

Compare the value of one of the parameters to a function call. This condition should only be used in an *On function* event, since outside of function calls there are no parameters set.

### On function

Triggered when the corresponding *Call function* action is used.

## Function actions

### Call expression

This is an alternative to the *Call function* action. It simply provides a parameter to enter an expression, and the result is ignored. You can use this to call a function via the *Function.Call(...)* expression, which may be more convenient if using a very large number of parameters.

### Call function

Trigger the corresponding *On function* events. Additional parameters can be passed that are accessed by the *Param* expression.

### Set return value

In a function event, set the value to be returned to the caller. This is either returned by the *Call* expression or accessed later using the *ReturnValue* expression.

## Function expressions

### Call

Call a function directly from an expression. The expression returns the return value that was set in the function, or 0 if no return value was set. Additional parameters can optionally be added after the name of the function, e.g. `Function.Call("CreateEnemy", 123, 456)`.

---

**Param**

Retrieve a parameter passed to a function call by its zero-based index. For example, `Function.Param(0)` returns the value of the first parameter.

---

**ParamCount**

Return the number of parameters passed to a function call.

---

**ReturnValue**

Return the value set using the *Set return value* action from the last function call. If *Set return value* is not used in a function, it returns 0.

# GAMEPAD

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

The **Gamepad** object allows you to detect input from a connected console controller, PC gamepad or joystick. While a range of devices ought to work with the Gamepad object, it is designed for and works best with the XBox controller, or similarly designed controllers (with the same button/stick layout). Click here to open an example of gamepad control.

Unfortunately whether or not a specific device is supported depends on a lot of factors, including the operating system, available drivers, and the browser's support. This makes it difficult to know in advance if a specific device will work. The XBox controller works out of the box on Windows systems with the Google Chrome browser. The Playstation controllers do not typically work without installing third party drivers since it is not officially supported.

Some mobile devices also support gamepad input, either by special hardware accessories, or by connecting a gamepad via a cable or wirelessly.

To prevent allowing websites to track you by your available controllers, most browsers supporting Gamepad input will report that there are no controllers connected until a button is pressed on one of the devices.

## Key mapping

Different controllers have different button layouts, or the buttons have different names. For example, the Playstation 3 controller uses *square*, *circle*, *triangle* and *X* buttons, whereas the XBox controller uses *A*, *B*, *X* and *Y* (note that *X* appears in a different position in each controller too). For consistency, the Gamepad object refers to the XBox layout only.

## Controlling behaviors with Gamepad

To control behaviors with Gamepad input, use the *Simulate control* action. An example is shown below for controlling a Platform behavior with the left analog stick and A to jump.




For more information, see the section on *Custom controls* in the Behavior reference.

## Multiple gamepads

Multiple gamepad devices can be connected to a single computer. To differentiate between them, most actions, conditions and expressions in the Gamepad object also take a *Gamepad* parameter. This is a zero-based index of the controller. For example, 0 identifies the first controller, 1 identifies the second, and so on. This allows you to make multiplayer gamepad-controlled games.

# Gamepad properties

### Analog deadzone

Devices with analog joysticks are extremely sensitive to input. This allows fine control in games, but also means a joystick in rest position can still register a fairly large amount of movement. If this wasn't ignored, a joystick-controlled player could start moving around erratically even when the player is not touching the control. To solve this all values close to rest position are ignored. Joystick movement is in the range -100 to 100 on each axis, and if the distance from the center is below the *Analog deadzone* value, it will return 0. For example, the default is 25, so values inside a circle with the radius 25 will still count as zero. This is the recommended value to ensure even ageing controllers with highly erratic input do not cause unintended player movement.

# Gamepad conditions

### Gamepad supports vibration type

Check if a currently connected gamepad supports a particular kind of vibration effect, such as dual rumble or trigger rumble. Note both the browser and the hardware must support the vibration type - for example it is possible a gamepad has hardware support for trigger rumble, but one browser supports it where another browser does not.

### Has gamepads

True if any gamepad is connected and activated. To prevent websites tracking you based on the available gamepads, most browsers supporting Gamepad input will report that no controllers are connected until a button is pressed on one of the devices.

### On gamepad connected

Triggered when a gamepad device is connected to the computer. To prevent websites tracking you based on the available gamepads, most browsers supporting Gamepad input will report that no controllers are connected until a button is pressed on one of the devices, when *On gamepad connected* will also run.

> *In this trigger, the GamepadIndex expression gives the index of the gamepad that was connected.*

### On gamepad disconnected

Triggered when a gamepad device is disconnected from the computer, such as by pulling out its cable.

> *In this trigger, the GamepadIndex expression gives the index of the gamepad that was disconnected.*

### Compare axis

Compare the position of an analog joystick on a specific gamepad. Values within the *Analog deadzone* are returned as 0. Axes values range from -100 to 100.

### Is button down

True if a given button is currently down on a specific gamepad. The buttons are always referred to according to the XBox controller layout.

### Is button index down

True if a given button by its numerical index is currently down on a specific gamepad.

### On any button pressed

Triggered when any button is pressed on a specific gamepad. The *ButtonIndex* expression is set with the index of the button. The special value -1 can also be used for the gamepad, in which case the trigger will fire when any button on any gamepad is pressed, and the *GamepadIndex* expression will also be set to the index of the gamepad where the button was pressed.

### On any button released

Triggered when any button is released on a specific gamepad. The *ButtonIndex* expression is set with the index of the button. The special value -1 can also be used for the gamepad, in which case the trigger will fire when any button on any gamepad is released, and the *GamepadIndex* expression will also be set to the index of the gamepad where the button was released.

### On button index pressed

Triggered when a given button by its numerical index is pressed on a specific gamepad.

### On button index released

Triggered when a given button by its numerical index is released on a specific gamepad.

### On button pressed

Triggered when a given button is pressed on a specific gamepad. The buttons are always referred to according to the XBox controller layout.

### On button released

Triggered when a given button is released on a specific gamepad. The buttons are always referred to according to the XBox controller layout.

## Gamepad actions

### Vibrate (dual rumble)

### Vibrate (trigger rumble)

Initiate vibration (also known as "rumble") of a specific gamepad for a period of time given in milliseconds. Each kind of vibration uses different rumble motors in the gamepad hardware. The types of vibration that are supported can be checked with the *Gamepad supports vibration type* condition. Each kind of vibration has different parameters for the hardware motors, allowing for different types of rumble effects to be produced. If a vibration is already active when this action is used, it will be replaced by this action.

### Reset vibration

Stop any active vibration started by the *Vibrate* action for a specific gamepad. The vibration will be immediately cancelled, so it will not fulfil the duration it was started for. If no vibration is active, this has no effect.

## Gamepad expressions

### Axis(Gamepad, Index)

Retrieve the current position of an analog joystick on a specific gamepad. *Index* specifies left analog X and Y or right analog X and Y axes. Axes range from -100 to 100. Axis values within the *Analog deadzone* are returned as 0.

### Button(Gamepad, Index)

Retrieve the current button press value of a button on a specific gamepad. *Index* specifies the zero-based index of a button from the dropdown list in the *Is button down* condition (e.g. 0 returns the value for the *A* button). The returned value depends on the features of the button: if the button is pressure sensitive, it can return any value from 0 to 100 depending on the pressure; otherwise it returns 0 for not pressed and 100 for pressed. Buttons which are not pressure sensitive are easier to detect using the *Is button down* condition.

### GamepadCount

Return the number of currently connected and active gamepad devices. To prevent websites tracking you based on the available gamepads, most browsers supporting Gamepad input will report that no controllers are connected until a button is pressed on one of the devices.

---

**GamepadID(Gamepad)**

A string intended to represent the device manufacturer and model for a specific gamepad, e.g. "XBox 360 controller". However in practice this varies depending on the system and browser in use.

---

**GamepadIndex**

In an *On gamepad connected/disconnected* trigger, or when using *On any button pressed/released* with a gamepad of -1, this expression indicates the index of the corresponding gamepad.

---

**ButtonIndex**

Return the numerical index of the last button pressed on a specific gamepad. This is useful with the *On any button pressed* and *On any button released* triggers to set up custom controls.

---

**RawAxis(Gamepad, Index)**

Retrieve raw axis input for a specific gamepad and axis index. This returns the value without keymapping, applying the analog deadzone, or multiplying the returned value by 100. Axis values range from -1 to 1.

---

**RawAxisCount(Gamepad)**

Return the number of axes available in the raw input for a specific gamepad. This returns the value without keymapping.

---

**RawButton(Gamepad, Index)**

Retrieve raw button input for a specific gamepad and button index. This returns the value without keymapping. Button values range from 0 to 1 (pressure sensitive buttons can return values in between).

---

**RawButtonCount(Gamepad)**

Return the number of buttons available in the raw input for a specific gamepad. This returns the value without keymapping.

# GEOLOCATION

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

The **Geolocation** object allows the user's current geographical location to be estimated. Note not all devices support geolocation, and of the devices that do, the accuracy can vary significantly. For example a desktop computer might not come with any location-tracking equipment, and only be able to report a location accurate to a 50 kilometer radius based on their internet connection. However this at least allows for the user's timezone, country, or possibly city or town to be determined. On the other hand many mobile phones and tablets come equipped with GPS equipment and can report their location as accurately as within a few meters, and track movements in real-time.

Click here to open an example of the Geolocation plugin.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Geolocation API.

## Permission prompts

When requesting the user's location, for privacy reasons most platforms will prompt the user for permission. Each platform tends to have its own specific way of asking for permission. The user may decline the permission request, in which case *On error* will trigger. Your projects should handle such a case gracefully if possible. Normally each browser or platform has a way to grant permission when it was previously declined, but it either involves prompting again, or making changes in the browser or app platform settings. Some platforms will fail without even prompting the user after a single declined permission request.

## Battery usage

It should be noted that tracking the user's location may involve activating GPS hardware in a phone or tablet, which can drain the battery more quickly. Requesting high-accuracy location information is also likely to use more power. Try to only request the user's location if absolutely necessary, use low accuracy if suitable, and request one-off positions rather than watching the position for a long time.

## Geolocation conditions

**Is supported**

True if the current device supports reporting the user position with geolocation. If false, none of the features of the object will work.

### Is watching location

True after a successful *Watch location* action, until the *Stop watching* action is used.

### On error

Triggered if an error occurs when requesting permission for, or retrieving, the user's location. The *ErrorMessage* expression contains more information about the problem in this trigger.

### On location update

Triggered after a successful *Request location* or *Watch location* action, when the position has been updated. This only triggers once after a successful *Request location action*, but can trigger regularly after a successful *Watch location* action whilst the position is tracked and updated.

## Geolocation actions

### Request location

Make a one-off request for the user's current location. The user may see a permission prompt which they must approve before any information is returned; if they decline, *On error* will trigger. *Accuracy* can be set to *High* to get more accurate results, but it may take longer to calculate and consume more battery. *Timeout* is the maximum time in seconds the device may take before it must return a position or trigger *On error*. *Maximum age* is the maximum age of a cached result that can be returned. If zero, the device will actively try to determine the user's position at that time. However if it is nonzero, and the operating system had previously requested the user's position within that time, the previous result may be returned immediately instead. This is faster and can save battery, but the result will not be as close to real-time. If a result is successfully determined, *On location update* will trigger.

### Watch location

As with *Request location*, but the location will be tracked. *On location update* will trigger whenever new position information is available, until the *Stop watching* action is used. Watching the location can consume more battery on mobile devices than one-off requests.

### Stop watching

Stop a previous successful request to watch the user's location. The position will no longer be updated.

## Geolocation expressions

The position-related expressions only update when *On location update* triggers, which in turn can only occur after a successful *Request location* or *Watch location* action.

**Accuracy**

**AltitudeAccuracy**

The estimated accuracy in meters of the latitude and longitude (for *Accuracy*) or the altitude (for *AltitudeAccuracy*). The accuracy may be more or less a guess, or if it is not known the expression returns 0.

**Altitude**

The estimated altitude in meters relative to sea-level, or 0 if not known.

**ErrorMessage**

In *On error*, a string with some additional information about the error.

**Heading**

**Speed**

While watching a position, the direction of travel in degrees relative to due north and speed in meters per second if available, else 0 of not available.

**Latitude**

**Longitude**

The latitude and longitude that has been determined, subject to the *Accuracy* (which may not be known).

**Timestamp**

A timestamp of the time at which the current details were retrieved. This is measured in milliseconds since midnight, January 1, 1970 (also known as a UNIX timestamp).

# GOOGLE PLAY

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/google-play

The **Google Play** plugin allows you to integrate your game with **Google Play Game Services**. Users can log in, submit and view scores, and unlock achievements.

## Setting up Google Play Game Services

Using Google Play Game Services requires that you have an account registered with the Google Play Developer service. There may be a small one-time registration fee if you need to sign up.

For each game that you wish to use Google Play Game Services for, find the **Play Games Services** section in the developer console and follow the steps to set it up.

Once added you can edit the game details such as its description and any associated images. You can also add achievements and leaderboards - which must be added before you can use them in the Google Play plugin - as well as configure testing and publishing.

To ensure your finished game has permission to access the Google Play Game Services, you need to link your "app" to your "game". Click the *Linked apps* section while editing the game in the developer console, the following steps vary depending on if you are publishing as a web app or a android app.

## Publishing as a Web app

Click the button to link a Web app. You will need to fill out details such as the URL it is played from and some authorisation details. Once completed, you should be given a client ID. This should be in a format similar to:

**12345678987-abcdefghijklmnopqrstuvwxyz1234567890.apps.googleusercontent.com**

**Copy and paste** this client ID in to the Google Play plugin's *Client ID* property. It is not actually necessary to fill in the *Application ID* property, but you can add it anyway: the application ID is the number that appears beside the game name in the header of the Developer Console, e.g. "My Super Game - 12345678". The application ID in this case is 12345678.

## Publishing as an Android app

Click the button to link an Android app. You will need to select an application that you have already registered with Google Play and provide the SHA-1 signature for the certificate you will use to sign your application. It is important that you have provided the signature and that when you are testing that you sign your application with the correct certificate. Your application will be unable to login unless you do this!

Once completed, you should be given a client ID and application ID. For an Android app you **only need you application ID**. This should be in a format similar to:

**12345678987**

**Copy and paste** this application ID in to the Google Play plugin's *Game ID* property.

## Enabling access in preview mode

When testing your project, your game will run from *https://preview.construct.net*. Since this is is different to the published URL of your game, Google Play Game Services will block access unless you add the preview URL as a permitted location.

To do this, in the Developer Console under *Game details*, find the header that says *API CONSOLE PROJECT* and follow the link to the API console project for your game. In the API console, select *APIs & auth*, then *Credentials*. Notice the *Javascript Origins* field contains only your final URL; this also needs to contain any preview URLs to be allowed access. Click **Edit settings**. Under *Authorized Javascript Origins*, make sure the final URL and the preview URL appear on separate lines, e.g.:

mywebsite.com

preview.construct.net

Click **Update**. Now your game should be authorised to be accessed from preview mode as well as when you publish it.

> *Note that an origin is the scheme and domain only, excluding any path or filenames.*

## Adding a Test User

When working with an unpublished Android application you will only be able to log in with the registered test users for that application. To add a test user click the *Testing* section while editing the game in the developer console, click **Add Testers** and type in the email address associated with the Google Account registered on your testing device.

## Basic usage

On startup, the plugin automatically attempts to load Google Play Services. If this is successful, then *Is loaded* will be true when the project starts running. If it failed then *Is loaded* will be false and none of the plugin's features will work.

Next the Google Play plugin will automatically try to sign the user in. If they have successfully signed in before, *On signed in* will trigger soon after startup. If they have never signed in before, *On auto-sign in failed* will trigger; in this event you should show a button that allows the user to sign in. When they click this button, use the *Sign in* action; if it succeeds, *On signed in* will be triggered.

Once the user is signed in, you can make use of the other plugin features such as requesting leaderboards, submitting high-scores, and unlocking achievements.

## Asynchronous actions

Most actions in the Google Play plugin are asynchronous. This means they are not completed immediately. Instead, the action starts a request which is sent off to the Google Play servers. A few moments later the server will respond, and a corresponding trigger will run in the Google Play plugin.

For example, the *Request player details* action does not complete immediately. The player details are not available until *On player details* received triggers, which should happen shortly after running the action. Only after the trigger runs can the player details be accessed.

## Non-immediate requests and caching (Android only)

Google Play Games Services offers **Immediate** and **Non-immediate** versions of most methods. Immediate methods will attempt to update the user's achievements / leaderboard information on the server immediately, whereas non-immediate methods will only update the local Play Games application on the next server sync. In line with official advice we use non-immediate methods where available. To the users eyes it will behave nearly exactly the same, but during testing you may experience some issues with caching. For instance, if you are adding new leaderboards and achievements they may take up to an hour to appear on your device. This doesn't mean it will take an hour for a hour to unlock an achievement though! Local changes appear within the Play Games app immediately, even if it isn't synced with the server.

## Google Play properties

**Application ID**

This is not currently necessary, but can be filled out with the application ID from the Google Play Developer Console.

**Client ID**

The client ID for the game from the Google Play Developer Console. This is only required for web applications. For more information see the section *Setting up Google Play Game Services* above.

**Game ID**

The Game ID is only used for Android applications. It is the same as the application ID. For more information see the section *Setting up Google Play Game Services* above.

## Google Play conditions

**Compare achievement state**

Compare whether an achievement at an index is revealed, hidden or unlocked. The achievement list must have already been successfully received.

---

**On achievement list success**

**On achievement list fail**

Triggered after the *List achievements* action, depending on whether the request succeeded or failed. If successful, achievement information for the current player is then available.

---

**On achievement metadata success**

**On achievement metadata fail**

Triggered after the *Get metadata achievements* action, depending on whether the request succeeded or failed. If successful, achievement metadata (such as the achievement names and icons) is then available.

---

**On achievement revealed**

**On achievement unlocked**

Triggered after the *Reveal*, *Unlock* or *Increment* actions when an achievement has successfully been revealed or unlocked. When incrementing achievements, the achievement is unlocked when it has incremented through every step.

---

**Is loaded**

True if the Google Play plugin has loaded and is ready to use. If false, no features of the plugin will work.

---

**Is signed in**

True if the user has been successfully signed in (possibly automatically).

---

**On auto-sign in failed**

Triggered upon the first visit, when the user cannot be automatically logged in. It is necessary to display a 'Sign in' button and use the *Sign in* action to get the user to sign in.

---

**On sign in failed**

Triggered when an attempt to sign in fails, either due to an error or because the user cancelled the attempt.

---

**On error**

Triggered if an error occurs. The *ErrorMessage* expression will contain information about the error.

**On player details received**

Triggered after the *Request player details* action, when the player details have been successfully received. The *Player details* category of expressions now are set to their correct values for the currently signed in user.

**On signed in**

**On signed out**

Triggers when the user is signed in or signed out from Google Play Game Services.

**On hi-score request success**

**On hi-score request fail**

Triggered after the *Request hi-scores* action depending on whether the request succeeded or failed. If successful, the hi-scores list is then available.

**On score submit success**

**On score submit fail**

Triggered after the *Submit score* action, depending on whether the submission succeeded or failed. If successful the score should then appear in hi-score lists.

## Google Play actions

**Get metadata**

Request metadata for the achievements list, such as the achievement names, descriptions and icons. If successful, *On achievement metadata* success is triggered.

**Increment**

Add to, or set, the number of steps in an incremental achievement. Once the full number of steps has been reached, the achievement is automatically unlocked.

**List achievements**

List the achievements for the currently signed in player. Optionally the list of achievements can be filtered to only those in a given state (e.g. revealed). If successful, *On achievement list success* triggers.

**Show achievements (Android only)**

Shows the native achievements dialog for the currently signed in player.

---

**Reveal**

If an achievement is hidden, set its state to 'revealed' for the currently signed in player. If revealing for the first time, *On achievement revealed* will be triggered.

---

**Unlock**

If an achievement is not already unlocked, set its state to unlocked for the currently signed in player. If unlocking for the first time, *On achievement unlocked* will be triggered.

---

**Request player details**

Request the details of the current player, such as their name and avatar image. If successful *On player details received* triggers and the *Player details* category of expressions can be used.

---

**Sign in**

If the player is not already signed in, pop up a window that allows them to sign in. Due to popup blockers, this may only work in a user input event, such as *On button clicked* or *On touch start*.

---

**Sign out**

If the player is already signed in, sign them out. This also allows for a different user to then sign in.

---

**Request hi-scores**

Request a hi-score list for a given leaderboard. Scores can be returned for public results, or "social" (from users connected to the currently signed in player), and a time limit can be applied such as to return only the day's best scores so far. The *top* type returns the very highest scores, and the *window* type returns the scores around the current player's own best score, allowing them to see where they appear in the rankings.

---

**Submit score**

Submit a new hi-score to a leaderboard. A *tag* can be provided, which is just a short string (up to 64 characters) associated with this score board entry, e.g. a short comment or an alternative alias for the player. If successfully submitted, *On score submit success* triggers.

---

**Show leaderboards(Android only)**

Shows the native leaderboards dialog for the currently signed in player.

**Show leaderboard (Android only)**

Shows the native leaderboard dialog with the specified leaderboard for the currently signed in player.

## Google Play expressions

**AchievementsCount**

The total number of achievements available. The achievements list must already have been successfully requested.

**AchievementNameAt(index)**

**AchievementDescriptionAt(index)**

**AchievementIDAt(index)**

**AchievementStepsAt(index)**

**AchievementTotalStepsAt(index)**

**AchievementTypeAt(index)**

Retrieve information about a given achievement in the achievements list. The achievements list must already have been successfully requested.

**AchievementUnlockedIconURLAt(index)**

**AchievementRevealedIconURLAt(index)**

Retrieve the icon image URL for a given icon in either its unlocked or revealed state. This can be displayed using the Sprite object's *Load image from URL* action.

**ErrorMessage**

In *On error*, the relevant error message if available.

**HiScoreCount**

The number of hi-scores in the current returned list of results.

**HiScoreTotalCount**

The total number of scores in the leaderboard, which may be greater than the number of returned results (*HiScoreCount*).

**HiScoreAt(index)**

**HiScoreFormattedAt(index)**

Return a numerical value, or formatted string, for a score at a given index.

**HiScoreRankAt(index)**

**HiScoreFormattedRankAt(index)**

Return the numerical rank, or formatted string of the rank (e.g. "1st"), at a given index.

**HiScoreNameAt(index)**

Return the name of the player associated with the score at an index.

**HiScoreTagAt(index)**

Return the tag (a short string) that was submitted along with the score at an index.

**HiScoreMyBest**

**HiScoreMyFormattedBest**

**HiScoreMyBestRank**

**HiScoreMyBestFormattedRank**

**HiScoreMyBestTag**

Return the details for the current player's own best score, including the numerical and formatted versions of the score and rank.

# HTML ELEMENT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/html-element

The **HTML Element** plugin allows you to include custom HTML content in your project. The HTML content can also be styled with CSS to alter its appearance. This is a good way to display content with more complex layout, such as tables, grids, and menus.

Construct creates a single HTML Element for each instance of the object, and sizes and positions the element to match the object's size and position in the project. Beyond that the content is entirely determined by the HTML you provide.

This guide will not teach how HTML and CSS work. However there is lots of information on the web about them. For example take a look at the HTML basics article on the MDN Web Docs.

You can find several examples of using the HTML Element object in the Example Browser. Add the HTML Element plugin as a filter and the list will display projects using it.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IHTMLElementInstance script interface. This allows access in worker mode; when worker mode is disabled, you can also use all the browser standard DOM APIs.

## Layering HTML objects

As the name suggests, this object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## CSS styling

If you only need a small number of custom CSS styles for HTML Element, these can be entered in to the *Style attribute* property. However it's often more convenient to use a CSS file to write styles in a more readable and easily editable way.

CSS styles can be added to the project by adding a stylesheet to the *Files* folder in the Project Bar. See Project files for more details.

The HTML Element object has *ID* and *Class* properties. These can be filled in and then CSS styles added for the corresponding ID or class, which will allow altering the appearance of the main HTML element created by Construct. For example if the class is entered as *myelement*, then CSS styles can be added for it with the rule `.myelement { ... }`.

Note that Construct sets the following styles on the HTML element via its style attribute. It is best not to override these as they may cause the element to work incorrectly.

- `position`
- `left`, `top`, `width` and `height`
- `box-sizing`
- `display` if made invisible
- `color` and `background-color` if *Set color* and *Set background color* are enabled
- `font-size` if *Auto font size* is enabled
- `user-select` if *Allow text selection* is disabled

One useful technique is to put a wrapper element inside the HTML element that is sized to fill its container. Then any custom styles or appearances can be used on that wrapper element without accidentally affecting the element created by Construct. This would mean the HTML content is something like this:

```
<div class="wrapper">
        <!-- Content goes here -->
</div>
```

This can then be styled to fill its container. There are several ways to do this in CSS; here is one approach using absolute positioning.

```
.wrapper {
        position: absolute;
        left: 0;
        top: 0;
        right: 0;
        bottom: 0;
}
```

Now further styles can be added to the wrapper, such as `font-size`, without conflicting with the styles added by Construct.

### Scaling with Construct's canvas

Construct sets a CSS variable named `--construct-scale` on the root `html` element of the document, with a number representing the canvas scale as a multiplier. You can use this with `calc()` to scale CSS properties to match the displayed canvas size, e.g.:

```
.mybutton {
        height: calc(var(--construct-scale) * 2em);
}
```

This sets `.mybutton` to have a height of 2em at 100% scale, but also adjusts the height to follow Construct's fullscreen scaling.

## Managing long HTML content

If you have a lot of HTML content to display, consider adding it as a HTML project file. Then use the AJAX object to fetch the project file, and set the content of the HTML Element to the content of the project file.

## Scaling with the canvas

By default *Auto font size* is enabled. This means Construct sets a `font-size` style on the main HTML element to an `em` size that scales with the displayed canvas size. For example at 100% scale it will set `font-size: 1em`. Then if the window is resized to display twice as large, it will set `font-size: 2em`.

This is useful for scaling your HTML content with the displayed size of the game. The key is to **use `em` units for sizing**. Then everything will automatically scale with the canvas.

For example if you set `border-radius: 20px`, that will use a fixed 20px size regardless of how large or small the window size is, meaning it will look wrong at some sizes. However if you set `border-radius: 2em`, it will scale the size to match the canvas size, ensuring it looks consistent at all sizes.

If you want to change the `font-size` for your HTML content, note that Construct sets that property already. The best approach is to use a wrapper element as described above, and change the `font-size` of the wrapper instead.

## Using BBCode

HTML Element also sets supporting its content using simple formatting tags known as "BBCode". This involves using square bracket tags such as [b] and [/b] around text to make bold, e.g. `[b]bold text[/b]`. The tags you can use for BBCode in HTML Element are listed below.

> *Note these BBCode tags are different to those supported by the Text or SpriteFont objects.*

| Tag | Description |
|---|---|
| [b]...[/b] | **Bold text** |
| [i]...[/i] | *Italic text* |
| [s]...[/s] | ~~Strikethrough text~~ |
| [u]...[/u] | Underline text |
| [sub]...[/sub] | Subscript text |
| [sup]...[/sup] | Superscript text |
| [small]...[/small] | Smaller text |
| [mark]...[/mark] | Mark text with highlight |
| [code]...[/code] | Format as code snippet |
| [h1]...[/h1] | Header 1 |
| [h2]...[/h2] | Header 2 |
| [h3]...[/h3] | Header 3 |
| [h4]...[/h4] | Header 4 |
| [item] | Item bullet point: • |

## Security

*Be careful when adding user-provided values in to HTML. Read this section carefully if you do this.*

By default the HTML Element object interprets content as HTML. This is safe with content you've written yourself. However it's common to insert user-provided values, such as the user's name, in to HTML. **This is a potential security risk** and must be handled carefully.

Inserting a string like `"<p>Your name is " & PlayerName & "!</p>"` is dangerous, because the player could enter HTML content for their name - which can even include script tags. For example the player could enter their name as `<script>runDangerousCode()</script>`. Then in the previous example the inserted HTML content will be `<p>Your name is <script>runDangerousCode()</script>!</p>` which will execute the given JavaScript code, which may be malicious.

The solution is to make sure all user-provided values are *escaped*. This makes sure that HTML tags are displayed as text, rather than interpreted as HTML, replacing characters like `<` and `>` with `<` and `>`. This is done with the *EscapeHTML* expression. For example the following expression is now safe as it escapes the player name:

```
"<p>Your name is " & HTMLElement.EscapeHTML(PlayerName) & "!</p>"
```

Another alternative is to use BBCode or plain text as the content type. In both cases it is impossible for users to insert unexpected HTML tags, so they are fundamentally safe.

## HTML Element properties

**Tag**

The name of the HTML tag that Construct creates. By default this is `div` for a `<div>` element, but it can be changed to any other HTML element.

**Content**

The text to use as the initial content of the HTML Element. The way this is interpreted is set by the *Content type* property, which defaults to HTML. If you have a lot of content consider using a project file instead - see *Managing long HTML content* above.

**Content type**

Choose how the initial content text is interpreted. The default is *HTML*. It can also be set to *BBCode* (see the section on *Using BBCode* above), or *Plain text* to avoid interpreting any tags on the text at all.

**Initially visible**

Set whether the HTML Element is visible upon creation. The element is made invisible by setting the style `display: none`.

---

**ID**

The ID to set for the HTML Element.

---

**Class**

The class to set for the HTML Element. Like with the HTML *class* attribute, multiple classes can be specified separated by spaces.

---

**Allow context menu**

Whether the browser's default context menu should be allowed on the HTML Element. Typically this should be allowed for input elements, but often disallowing it is appropriate for games, which may use controls like right-clicking for other purposes. If disallowed, Construct will call `preventDefault()` on the `"contextmenu"` event.

---

**Stop input events**

Construct sometimes stops input events on HTML elements from reaching the rest of the project. For example using arrow keys to move the caret inside a text input should not also move the player if they are controlled by arrow keys. This also applies to mouse or touch input. This is optional with the HTML Element object and has three modes:

- **No** - no input events are stopped. All mouse, touch or keyboard input on the element will reach the rest of the project.
- **Child elements only** - input events reaching child elements of the main HTML element will be stopped from reaching the rest of the project. However the main HTML element will not stop any input.
- **Entire element** - all input events on the entire HTML element will be stopped from reaching the rest of the project.

---

**Origin**

Choose the position of the origin of the object relative to its unrotated bounding rectangle.

---

**Set color**

**Default color**

Enable *Set color* to set the `color` CSS style on the HTML element to the given color. This changes the text color. Usually this is desirable as otherwise the text color may not be visible

against the background.

**Set background color**

**Default background color**

Enable *Set background color* to set the `background-color` style on the HTML element to the given color.

**Auto font size**

Automatically set a `font-size` CSS style on the HTML element according to the display scale of the canvas. This provides a convenient way to size HTML content with the canvas by using `em` units. See the section *Scaling with the canvas* above for more details.

**Allow text selection**

Set whether dragging over text is allowed to create a selection. By default this is disabled, which adds the CSS style `user-select: none`.

**Style attribute**

Additional CSS styles for the main HTML element can be added here, separated by semicolons. This will be set via the `style` attribute of the HTML element. Consider using CSS files if there are more than a couple of simple styles - see the section *CSS styling* above for more details.

# HTML Element conditions

**On clicked**

Triggered when any part of the HTML element is clicked. The *TargetID* and *TargetClass* expressions are set to the ID and class of the clicked element.

**On clicked class**

Triggered when an element with a given list of CSS classes is clicked. This includes a click on any child elements contained by the element with the given CSS classes. A single class name can be specified, or multiple classes given separated by spaces, in which case the clicked element must match all the given classes.

> *This does not use a CSS selector, so don't prefix class names with a dot.*

**On clicked ID**

Triggered when an element with a given ID is clicked. This includes a click on any child elements contained by the element with the given ID.

> *This does not use a CSS selector, so don't prefix the ID with* `#`.

---

### On CSS animation ended

Triggered when a CSS animation for any contained element finishes (i.e. the `"animationend"` event). The animation name is specified by the `@keyframes` rule. The *TargetID* and *TargetClass* expressions are set to the ID and class of the element that finished a CSS animation. This trigger is useful for things like destroying the HTML element after a fade out CSS animation has finished.

## HTML Element actions

> *Note that some actions have the same name. Actions in the HTML element group are common to all HTML-based plugins like Button and Text Input, and will only affect the main HTML element. Actions in the HTML content group are unique to the HTML Element plugin and can update the content of the object.*

---

### Create sprite image element

Creates an `<img>` element with the content of a given Sprite object's current image, and inserts it to the HTML element. The location to insert is set by a CSS selector. This can be left blank to insert in to the main HTML element. Alternatively a selector like `".myclass"` will mean to insert content to the child element with the class *myclass*. The element can be set to inserted at the start or the end of the given element, or alternatively replacing all the content of the given element with the image. The inserted image element can optionally also be given an ID and class, which helps make it easy to style the inserted image with CSS.

> *This action provides a simple way to show a Sprite image on top of a HTML element, since normally HTML elements always show on top of Sprites.*

---

### Insert content

Insert the given string of content inside the HTML element. The string is interpreted according to the content type (HTML, BBCode or plain text). The position indicates whether to insert at the start or the end of the given location. The location to insert is set by a CSS selector. This can be left blank to insert in to the main HTML element. Alternatively a selector like `".myclass"` will mean to insert content to the child element with the class *myclass*; if the *Type* is set to *all*, it will insert the same content to all elements matching the selector.

*Be careful when inserting user-provided values as HTML. See the section on Security above.*

---

### Position object at element

Sets the position and size of a given object to match the position and size of a specific HTML element. The element is chosen by a CSS selector, e.g. `".myclass"` to position an object at an element with the class *myclass*.

*This action provides a way to use invisible HTML and CSS for complex layouts, while displaying the actual content with other objects, allowing for full use of Z order, effects and so on.*

---

### Remove content

Either removes elements entirely, or clears their content to make them empty, according to the given mode. The content to remove or clear is given by a CSS selector. This can be left blank to clear the main HTML element. (Note the main HTML element cannot be removed, so in this case it will be cleared instead.) Alternatively a selector like `".myclass"` will mean an element with the class *myclass* will be removed or cleared; if the *Type* is set to *all*, this will remove or clear all elements matching the selector.

---

### Set attribute

Either sets or removes a named attribute on a HTML element, according to the given mode. The value is ignored if removing. The HTML element to alter attributes for is given by a CSS selector. This can be left blank to alter attributes for the main HTML element. Alternatively a selector like `".myclass"` will update attributes for an element with the class *myclass*; if the *Type* is set to *all*, this will update attributes for all elements matching the selector.

---

### Set class

Either adds, toggles or removes classes on a HTML element, according to the given mode. Multiple classes can be updated at once by separating their names with spaces. The HTML element to alter classes for is given by a CSS selector. This can be left blank to alter classes for the main HTML element. Alternatively a selector like `".myclass"` will update the classes for an element with the class *myclass*; if the *Type* is set to *all*, this will update classes for all elements matching the selector.

*Note that the Class parameter is not a selector so does not need class names to be prefixed with a dot; however the Selector parameter is a CSS selector and so needs class names to be prefixed with a dot.*

### Set content

Replaces some content inside the HTML element with the given string. The string is interpreted according to the content type (HTML, BBCode or plain text). The location to replace content is set by a CSS selector. This can be left blank to replace the content of the entire main HTML element. Alternatively a selector like `".myclass"` will mean to replace the content of the child element with the class *myclass*; if the *Type* is set to *all*, it will replace the content of all elements matching the selector.

> *Be careful when inserting user-provided values as HTML. See the section on Security above.*

### Set CSS style

Set a single CSS style on the style attribute inside the HTML element, based on a CSS property name and a string for its value. Setting the value to an empty string will remove the property from the style attribute. The element to change the style for is set by a CSS selector. This can be left blank to update the style attribute of the main HTML element. Alternatively a selector like `".myclass"` will mean to update the CSS style of the child element with the class *myclass*; if the *Type* is set to *all*, it will update the style of all elements matching the selector.

### Set scroll position

Set the horizontal or vertical scroll position of an element. The HTML element to scroll is given by a CSS selector. This can be left blank to scroll the main HTML element, or use a CSS selector string like `".myclass"` to scroll an element with the class *myclass*. This action only scrolls one element matching the selector. The *Direction* specifies whether to set the scroll top (vertical) or left (horizontal) position, and the *Position* value is the scroll position to set in CSS pixels.

## HTML element expressions

### EscapeHTML(string)

Escapes any characters with meaning in HTML from the given string, such as replacing `<` with `<`. This is important for securely inserting user content in to HTML strings. See the *Security* section above for more details.

### HTMLContent

### TextContent

A string with the complete content of the HTML element, either as a full HTML string, or plain text only (with HTML tags removed).

*Note that this does not update immediately after actions that change the HTML content - use the Wait for previous actions to complete system action to ensure any modifying actions have completed before reading updated content from these expressions.*

---

**TargetClass**

**TargetID**

In a trigger like *On clicked*, these contain the class and ID of the affected HTML element. Note that *TargetClass* can be a space-separated list of multiple classes, as per the *class* HTML attribute.

# IFRAME

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

The **iframe** object can display another web page, or a string of some HTML content, in your project. It is named after the <iframe> element, which is what the object uses.

> *If you display a HTML string in the iframe instead of loading a URL, it is recommended to check the Enable sandbox property, especially if the content includes any user-provided content. This makes it safe to display untrusted HTML content, avoiding security problems like XSS (cross-site scripting) where attackers can control or abuse your site.*

This object can also be used to show embedded content, such as YouTube videos. For example if you choose to share a YouTube video and select the "embed" option, it provides some HTML code for an *iframe* element. The `src` attribute is the embed URL for the video, e.g. https://www.youtube.com/embed/pWiC5Ln_0yA. This can be used in the *URL* property of Construct's iframe object to show the video in your game.

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## iframe properties

**URL**

The URL of a web page to load inside the iframe.

**HTML content**

A string of HTML content to display inside the iframe. This loads the HTML locally and does not request a separate web page. This is only used if the *URL* property is left empty.

**Initially visible**

Whether the object is initially visible at runtime.

**ID**

An optional `id` attribute to set on the iframe element. This may be useful if you have other styles or JavaScript code that you want to use with the iframe.

**Allow**

An optional feature policy string to set in the iframe's `allow` attribute, which specifies what the displayed page is allowed to do. The default is designed to allow embedded video playback on services like YouTube, granting the video permission to enter fullscreen, autoplay, and use encrypted media.

---

**Enable sandbox**

**Sandbox**

Check *Enable sandbox* to add the `sandbox` attribute on the iframe element, providing enhanced security. The *Sandbox* property is then the string to use in the `sandbox` attribute. Enabling the sandbox starts by removing a wide range of capabilities, and then each capability can be re-enabled by adding it to the sandbox string. By default the sandbox string allows JavaScript execution, but blocks forms, popups, same-origin access, top-document navigation, and more. If you only display static HTML content, you can also remove the default `allow-scripts` to block any JavaScript execution at all. For more information see the iframe sandbox attribute on MDN.

> *Do not use both the `allow-scripts` and `allow-same-origin` in the sandbox string. This allows scripts to remove the sandbox protection.*

## iframe conditions

The iframe object does not have any of its own conditions. See common conditions for features shared between form control objects.

## iframe actions

See common actions for features shared between form control objects.

---

**Display HTML string**

Load a string of HTML content in the iframe. This is similar to using the *HTML content* property. If the iframe was previously displaying a URL, it will switch to the HTML content instead.

---

**Navigate to URL**

Load a new URL in the iframe. If the iframe was previously displaying a string of HTML, it will switch to loading this URL instead.

---

## iframe expressions

The iframe object does not have any of its own expressions.

# INSTANT GAMES

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

The Instant Games plugin allows you to create games that integrate with the **Facebook Instant Games** platform. These games can be played in Messenger, the Facebook news feed, and so on.

*Don't use the Facebook plugin in Instant Games. You cannot publish Instant Games using the Facebook API (the Instant Games API works separately). Only use the Instant Games plugin.*

## Instant Games official documentation

For information specifically relating to the Instant Games platform, please also refer to the official Instant Games developer documentation. Construct's Instant Games plugin handles many of the details for you, but the official documentation also includes useful information on setting up an app, managing web hosting and publishing, setting up ads, tips and best practices, additional ways to test the app, and more.

## Exporting for Instant Games

Make sure you use the **Facebook Instant Games** export option when publishing an Instant Game. This will ensure additional files required by Instant Games are included with the export. The resulting zip file can be directly uploaded to Facebook's web hosting service.

## Instant Games properties

**Navigation menu**

The type of the navigation menu used by Instant Games to display. This corresponds to the `navigation_menu_version` in the bundle configuration.

## Instant Games conditions

**On load ad success**

**On load ad error**

Triggered after the *Load ad* action when an ad of the same type either successfully finishes loading or has an error loading.

**On show ad success**

**On show ad error**

Triggered after the *Show ad* action when the user finishes viewing an ad of the same type or if it fails to be shown.

---

**Is available**

True if the Instant Games platform is available. This will be false if the game is hosted outside of the Instant Games platform or was not exported using the *Facebook Instant Games* export option. Note Instant Games is also unavailable in preview mode.

---

**On context changed**

**On context change cancelled**

Triggered after the *Change context* action when the user changes context or cancels the dialog. If the context changed, the context ID will change too.

---

**On error**

Triggered if an error occurs at any point using the Instant Games platform. Usually looking in the browser console will provide more information about the problem.

---

**On pause**

Triggered when the Instant Games platform wants the game to pause in order to handle an interruption, e.g. due to an incoming phone call.

---

**On shortcut created**

**On shortcut failed**

Triggered after the *Create home screen shortcut* action depending on whether the shortcut was successfully created.

---

**On leaderboard loaded**

Triggered after the *Load leaderboard scores* action when the leaderboard score data has been loaded. The leaderboard expressions can then be used to access the data.

---

**On player score loaded**

Triggered after the *Load player score* action when the player's score data has been loaded. The player score expressions can then be used to access the data.

---

**On score submitted**

Triggered after the *Set score* action once the score has been submitted to the Instant Games platform.

### On connected players loaded

Triggered after the *Load connected players* action when the connected player's data has been loaded. The connected player expressions can then be used to access the data.

### On loaded player data

Triggered after the *Load player data* action when the player data has been loaded. The *PlayerData* expression can then be used to access the data.

## Instant Games actions

### Load ad

Load an interstitial or rewarded video ad. The ad must have been created on Facebook's advertising platform and the placement ID provided to this action. Ads must be loaded before they can be shown. Only one ad of each type can be loaded at a time. *On load ad success/error* will trigger depending on the outcome.

### Show ad

Show an ad that has previously been successfully loaded. Note only the last successfully loaded ad of the given type will be shown with this action. *On show ad success/error* will trigger depending on the outcome.

### Log event

Log an event to Facebook's analytics platform. This can help you learn more about how players are interacting with your game. An optional value can be provided that will be summed with the prior event value.

### Change context

Opens a dialog allowing the player to change to a different Instant Games context. This action corresponds to the `chooseAsync()` SDK method. If the user successfully changes context, *On context changed* will trigger and the context ID will change. If the user cancels the dialog, *On context change cancelled* will trigger.

### Create home screen shortcut

(Android only) Request to create a home screen shortcut to the game on the user's device. *On shortcut created/failed* will trigger depending on the outcome.

### Custom update

Post a custom update in to a message thread. This can use a custom image and text for the call-to-action (CTA) and other fields. This action corresponds to the `updateAsync()` SDK

method with a `"CUSTOM"` action.

---

**Quit**

Quit the game if it is being played in a context where exiting is possible.

---

**Share**

Show a dialog to invite, request, challenge or share, in a chat or timeline. An image must be provided in the form of a Sprite object, whose currently showing image will be used.

> *There must be an instance of the chosen Sprite object on the layout, otherwise its image will not be loaded and the share will not work.*

The text to use in the share message can be provided. The content of the *Data* parameter will be set to the *EntryPointData* expression if another user loads the game through that particular share. This can be used to automatically join the other player to the sharing player's game.

---

**Subscribe to bot**

Prompt the user to subscribe to a bot associated with the game, if one has been configured.

---

**Switch game**

Request that the client switch to a different Instant Game according to its App ID. The game must belong to the same business as the current game. This action corresponds to the `switchGameAsync()` SDK method.

---

**Load leaderboard scores**

Load a list of scores from a leaderboard. The leaderboard must have first been created for the app. *On leaderboard loaded* will trigger once the scores are available, after which the leaderboard expressions can be used to access the data.

---

**Load player score**

Load the current player's own score and rank. *On player score loaded* will trigger once available, after which the player score expressions can be used to access the data.

---

**Set score**

Set the player's current score on a leaderboard. *On score submitted* will trigger once submitted.

---

**Share leaderboard update**

### Share leaderboard update

Post an update in a chat or timeline indicating the player's current score status for a given leaderboard. This action corresponds to the `updateAsync()` SDK method with a `"LEADERBOARD"` action.

---

### Load connected players

Load a list of connected players (other users who have also interacted with this player in the game). *On connected players loaded* will trigger once available, after which the connected player expressions can be used to access the data.

---

### Load player data

Load the last set data for the current player. *On loaded player data* will trigger once available, after which the *PlayerData* expression can be used to access it.

---

### Set player data

Store some data to associate with the current player on the Instant Games platform. This data can later be requested using the *Load player data* action. Note the data should be kept as short as possible, ideally under 1000 characters.

## Instant Games expressions

---

### ContextID

A unique identifier that identifies the current game context. Note this can change after the *Change context* action completes.

---

### ContextType

The type of the current game context. This can be one of `"POST"`, `"THREAD"`, `"GROUP"` or `"SOLO"`. Note this can change after the *Change context* action completes.

---

### EntryPointData

If the game is launched from a share, update or switch, then this expression returns the data that was associated with the share.

---

### Locale

The locale of the current player.

---

### Platform

A description of the current platform the user is playing on. This can be one of `"IOS"`, `"ANDROID"`, `"WEB"` or `"MOBILE_WEB"`.

**SDKVersion**

The version of the Instant Games SDK in use.

**LeaderboardResultCount**

**LeaderboardPlayerIDAt**

**LeaderboardPlayerNameAt**

**LeaderboardPlayerPhotoURLAt**

**LeaderboardRankAt**

**LeaderboardScoreAt**

In *On leaderboard loaded*, return the number of results, and the player ID, name, photo URL, rank and score for each entry in the list.

**PlayerRank**

**PlayerScore**

In *On player score loaded*, the current player's rank and score on the leaderboard.

**ConnectedPlayerCount**

**ConnectedPlayerIDAt**

**ConnectedPlayerNameAt**

**ConnectedPlayerPhotoURLAt**

In *On connected players loaded*, return the number of results, and the connected player ID, name and photo URL for each entry in the list.

**PlayerData**

In *On loaded player data*, the string of the last set player data that was stored to the Instant Games platform.

**PlayerID**

**PlayerName**

**PlayerPhotoURL**

Retrieve basic information about the current player, including their unique ID, their name and a URL to their profile picture.

# INTERNATIONALIZATION

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

The **Internationalization** plugin provides tools to manage the localization of a project. This includes facilitating translations, pluralization, formatting dates for the user's locale, and so on.

See the built-in example project for a thorough example showing the various internationalization features.

> *Internationalization is sometimes written as the shorthand i18n, referring to the fact the word starts with an I, ends with an N, and has 18 other letters in between.*

### Scripting

When using JavaScript or TypeScript coding, the string lookup features of this plugin can be accessed via the Internationalization script interface. Note however many other features of this plugin merely access the browser-provided Intl APIs, so those are not duplicated in the script interface, as you can access them directly instead.

## Locales

Locales - which specify a region, language or dialect - are specified using standardized BCP 47 language tag, also known as just a *language tag*. For example `en-US` refers to US English, `en-GB` refers to British English, `pt-BR` refers to Brazilian Portuguese, and so on.

## The translation file

The plugin can be loaded with translation information through event sheets, but the most common use case is to have all the strings for a given language in a separate file and load that using the *Load from JSON* action. A bare-bones example of such file would be as follows:

```
{
	"locale": "en-GB",
	"strings": {
		"foo": "example localized string",
		"bar": {
			"baz": "more localized text"
		}
	}
}
```

A language file is a JSON file, with two distinct keys.

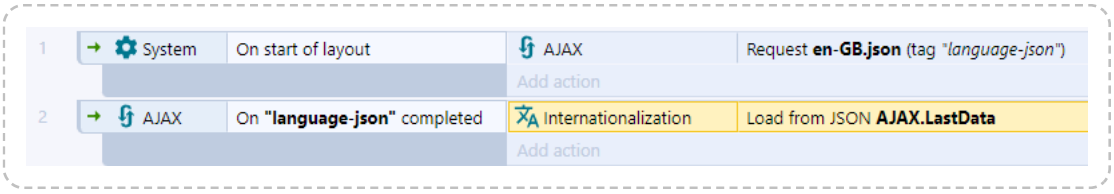
The `"strings"` key is mandatory, all the localized text should be nested inside it. It can contain any valid JSON - how it is organized is entirely up to the user to decide what is more convenient. Trying to load a file without this key will result in an error.

The `"locale"` key is optional and is used when loading to indicate to what language the file corresponds to. If the key is not present then the content of the file will be associated with the locale the Internationalization plugin is currently set to. If the key contains a value which can not be parsed into a valid locale, an error will be thrown when loading.

## Loading an Internationalization file

To load an internationalization file do the following:

1. Use the AJAX plugin to request the file.
2. In the AJAX *On Complete* trigger, use the *Load from JSON* action of the Internationalization plugin to load the AJAX data.



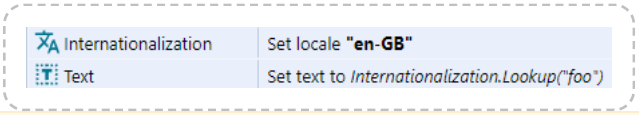

Following those steps it is possible to load a localization file for each locale that needs it.

> *When loading files like in the example above, the file needs to have the **"locale"** key so the plugin can know to which locale the data belongs to without any further event blocks.*

## Looking up localized strings

Once the plugin has the data for the required locales, the next step is to look up the strings to use them where needed. To do that do the following:

1. Use the *Set locale* action
2. Use the *Lookup* expression with the appropriate path




> *If the example localization file at the beginning of this document was loaded, that look up would yield the string **"example localized string"**.*

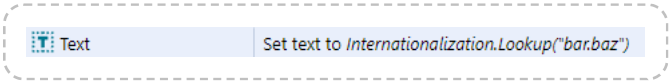
# Nesting, Push and Pop

In the case of having a few strings that need to be localized it can be easy enough to just have all of them at the root of the **"strings"** key in the localization file. As the amount of text that needs localization increases it can be useful to group related strings together to keep the localization files tidy.

To make the lookup for a nested string you can do two things:

## Absolute path

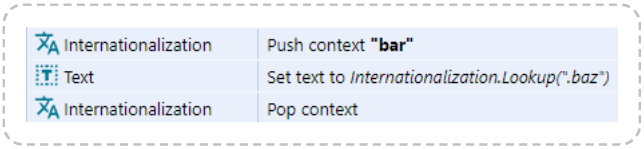
Include the absolute path to the string you are looking for in the argument of the *Lookup* expression. Dots can be used to navigate to nested JSON keys.




This option is straight forward and can be convenient in isolated cases.

## Relative path

Push the context where the strings you are looking for are in the localization file and then use relative paths when using the *Lookup* expression. A relative path begins with a dot, and is appended to the current context.




In the case of having many strings in the same context it can be useful to use relative paths to avoid duplication.

When using relative paths always remember to have a corresponding *Pop context* action for each *Push context* action.

> *The previous examples assume the example localization file at the beginning of the document has been loaded.*

> *Note that absolute paths never have a preceding "." and that relative paths always have a preceding "."*

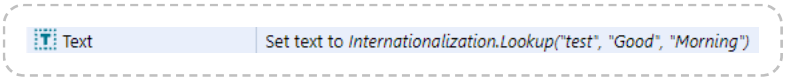
# Substitutions

By using substitutions it is possible to produce dynamic results at runtime, take the following example:

```
{
	"locale": "en-GB",
	"strings": {
		"test": "Hello {0} {1}"
	}
}
```

The string has two placeholders that can be replaced at runtime if additional values are provided when using the *Lookup* expression. There can be any amount of placeholders and the number in between the curly braces refers to the order of the arguments passed to *Lookup*.

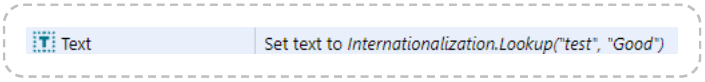
**Example 1**




The output of that example is the string **"Hello Good Morning"**.

If there are more placeholders than arguments, like in the example bellow, then the placeholders with no matching argument are left unchanged in the final result

**Example 2**




The output of that example is the string ***"Hello Good {1}"***.

# Plural rules

Following is a simple example of a localization file set up with a plural rule for British English.

```
{
	"locale": "en-GB",
	"strings": {
		"plural-example": {
			"one": "{0} Pig",
			"other": "{0} Pigs"
		}
	}
}
```

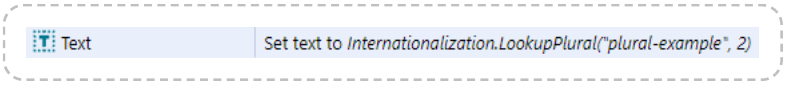
In order to look up the plural form the *LookupPlural* expression is used passing in an additional value which is used to decide which plural form will be used.

**Example 1**



That example would yield the string **"1 Pig"**.

**Example 2**




That example would yield the string **"2 Pigs"**.

> *The placeholder **{0}** in plural strings is always substituted by the second argument of the expression. The placeholder is optional - if it is removed from the localization string the correct plural form will still be picked.*

> *LookupPlural can also have any amount of additional placeholders for substitution just like Lookup*

### Pluralization in other languages other than English

English is a relatively simple language when it comes to pluralization, as there are only two plural categories: **"one"** and **"other"** which refer to single units and more than one (or zero) respectively.

Other languages might be more complex in this regard. To find out how many categories and the name each category has, use the following expressions to print out the values and see what keys your localization files need.

*PluralCategoryCount* will return the total amount of categories for the current locale.

*PluralCategoryAt(index)* will return the name of the plural category with the provided zero-based index.

> *Make sure to use the Set locale action with the locale you want that information before using the expressions.*

As an example, the locale **"ar-EG"** (Egypt Arabic) has six different plural categories, "few", "many", "one", "two", "zero" and "other". So a localization file for this language which uses plural rules should provide translations for each category.

## Internationalization conditions

**Compare to current locale**

Test the passed in locale value against the locale the plugin is currently configured with.

## Internationalization actions

### Set locale

Set the locale the plugin will use.

### Set context

Set a context to get strings from. An absolute context overwrites any existing context and a relative context is appended to the existing one.

### Push context

Push a context to get strings from. This helps to avoid duplication in the case of having to look up multiple strings in the same context.

### Pop context

Remove the existing context from the stack.

### Add string

Dynamically add a localized string to the provided context in the localization data for the current locale.

### Load from JSON

Load the plugin with localization data for the current locale or the locale defined in the passed in JSON.

## Internationalization expressions

### Locale

Returns the locale string the plugin is currently using.

### DefaultLocale

Return the default locale for the current system. This can be used to select the default language.

### Lookup(Context [, ...])

Looks up a localized string based on the current locale and a passed in context. Supports a variable list of arguments for substitution.

### LookupPlural(Context, Count [, ...])

Looks up the plural form of a localized string based on the current locale a passed in context and a number to decide which plural form to return. Supports a variable list of arguments for substitution.

---

**CurrentContext**

Return the current context of the plugin.

---

**SelectPlural(number)**

Returns the name of the plural form based on the current locale and the passed in number.

---

**PluralCategoryCount**

Returns the total amount of plural categories for the current locale.

---

**PluralCategoryAt(index)**

Returns the plural category name based on the current locale and the provided zero-based index.

---

**SaveToJSON**

Returns a JSON representation of the plugin internal state.

---

**FormatNumberAsDecimal(number)**

Format the passed in number as a decimal based on the current locale.

---

**FormatNumberAsPercent(number)**

Format the passed in number as a percent based on the current locale.

---

**FormatNumberAsCurrency(Number, Currency, CurrencyDisplay)**

Format the passed in number as a currency in the current locale. The **currency** argument must be a valid 3 letter ISO currency code - see this table for supported currencies. If an unsupported currency is passed in, a warning will be printed to console indicating all the supported currencies. The **currencyDisplay** argument can be any of *"symbol"*, *"narrowSymbol"*, *"code"*, *"name"*. Using an unsupported value will default to *"symbol"*.

---

**FormatNumberWithUnit(Number, Unit, UnitDisplay)**

Format the passed number as a unit in the current locale. The **unit** argument can take any of the values defined in this table. The **unitDisplay** argument can be any of *"long"*, *"short"* or *"narrow"*. Using an unsupported value will default to *"short"*.

**RegionName(RegionLocale)**

Returns the region name of the passed in locale in the language of the current locale of the plugin.

**LanguageName(LanguageLocale)**

Returns the language name of the passed in locale in the language of the current locale of the plugin.

**CurrencyName(CurrencyCode)**

Returns the currency name of the passed 3 letter ISO currency code, in the language of the current locale of the plugin.

# JSON

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

The **JSON** object can parse and read strings in JavaScript Object Notation (JSON) format, as well as writing data and converting the result back to a JSON string. A description of the JSON format is out of the scope of this manual; however there are some free tutorials you can search for on the web.

Click here to open an example of the JSON plugin.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IJSONInstance script interface. (JavaScript and TypeScript have built-in support for JSON, but this allows for interacting with data used in an event sheet.)

## Loading a JSON file

JSON must be loaded as a string with the *Parse* action. If you have a small snippet of JSON, you can paste it directly in to the action parameter - but note in expressions a double-quote character ( `"` ) must be repeated twice ( `""` ) to avoid ending the string, which can be inconvenient. Instead it is recommended to request a JSON project file using the AJAX object. When the AJAX request completes, pass `AJAX.LastData` in to the *Parse* action. Then the data from the file can be used.

## JSON paths

Construct only supports numbers and strings in expressions. To allow you to use structures like nested objects and arrays, the JSON object uses a special *path string* which identifies keys in the JSON data. The path is similar to the JavaScript syntax that would be used to access the JSON data, but note it is not actually evaluated as JavaScript code.

A path is essentially a list of nested keys separated by dots. For example consider the following JSON data:

```
{
        "foo": {
                "bar": 42
        }
}
```

The path `foo.bar` refers to the inner "bar" key, which will return the value 42.

### Escaping

If a JSON key actually has a dot in it, e.g. `"my.key"`, then the dot needs to be escaped in a path string otherwise it will try to look for a key named `my` with another object key named `key` inside it. Dots preceded by a backslash ( `\` ) will be interpreted as part of the key name, e.g. `my\.key` will look for a key named `"my.key"`. To actually use a backslash in a key name, then use a double backslash in a path, e.g. `my\\key` will look for a key named `"my\key"`.

### Types

Since Construct expressions only support numbers and strings, only number and string type properties can be directly returned in expressions. Booleans are returned as a number (0 for false and 1 for true). However using paths, loops and conditions, there are a variety of tools to identify what kind of data is available and access values held within objects and arrays (even when nested), as well as detecting special values like `null`.

### Arrays

Array elements can be accessed as if their elements were numbered properties (which is actually how JavaScript specifies arrays internally). For example consider the following JSON data:

```
{
        "array": [123, 456]
}
```

Like most of Construct JSON arrays use zero-based indices, so the path `array.0` refers to the first element (123) and `array.1` refers to the second element (456), and so on.

### Relative paths

The *Set path* action changes the current path, making it more convenient to access deeply nested keys. A relative path can then be used to continue from the current path. Relative paths begin with a dot, e.g. `.bar`. If a path does not begin with a dot, it is always treated as absolute (starting from the root), regardless of the current path.

For example suppose you want to access multiple keys under a common path, like `foo.bar.baz.first` and `foo.bar.baz.second`. You can first use *Set path* to set `foo.bar.baz` as the current path, and then the paths `.first` and `.second` refer to `foo.bar.baz.first` and `foo.bar.baz.second` respectively. Even with a current path set, the path `abc.def` refers to top-level keys because it does not start with a dot, so is treated as absolute.

The *For each* looping condition also sets the current path to the full path to the current key being iterated, making it convenient to retrieve data in a loop.

## JSON conditions

**Compare type**

Test the type of a value at a given path. This can also detect the special `null` value that cannot be returned in a Construct expression, as well as identifying arrays and objects.

---

**Compare value**

Compare the value at a given path. This can only be used with number or string values.

---

**For each**

Repeats once for each key at a path in the JSON data. This can be used with either object or array types; in the case of arrays, the keys are the array indices (e.g. 0, 1, 2...) represented as a string. Inside the loop, the current path is set to the current key being iterated, so relative paths can be used to retrieve data from the current key. The *CurrentKey*, *CurrentValue* and *CurrentType* expressions return information about the current key-value pair being iterated.

---

**Has key**

Determine if a key exists at a given path.

---

**Is boolean set**

Determine if a given path contains a boolean "true" value.

---

**On parse error**

Triggered after a *Parse* action if there was invalid syntax in the JSON string resulting in an error trying to parse it.

---

**On parse success**

Triggered after a *Parse* action if the JSON syntax was valid and parsing completed successfully.

## JSON actions

Note that when setting values, nonexistent keys are created as necessary. For example if the JSON file is empty but you set the number 5 at the path `foo.bar`, the `foo` and `bar` keys are automatically created, resulting in the data `{ "foo": { "bar": 5 } }`.

---

**Delete key**

Delete the key at a path so it is no longer present in the JSON data.

> *Note this action cannot remove elements from an array. Use the array modifying actions instead.*

**Parse**

Parse a string of JSON data and load it in to the object so it can be accessed. If the data is valid and is parsed successfully, *On parse success* is triggered; otherwise if the data is invalid and parsing fails, *On parse error* is triggered.

**Pop value**

Remove an element at the start or end of an array located at a path. If the path does not specify an array, this does nothing.

**Push value**

Add an element with the given value at the start or end of an array located at a path. If the path does not specify an array, this does nothing.

**Insert value**

Inserts an element with the given value into an array located at a path, increasing the size of the array by 1. The element is inserted at a specified index, if any elements existed at or after that index they pushed forward by 1.

**Remove values**

Removes a specified number of elements from an array located at a path, reducing the size of the array. Elements are removed starting at a specified index, if there are less elements after the array than requested to be removed then only the available number will be removed.

**Set array**

Create an array with a given number of elements at a path. If an array already exists at the given path, it is resized to the given number of elements. In both cases, any new elements are initialised to 0.

**Set boolean**

Set a true or false value at a path.

**Set JSON**

Parse a string of JSON data, and set the value at a path to the resulting JSON. This is useful to merge data from different sources in to the same JSON object.

**Set null**

Set the special `null` value at a path.

**Set object**

Set an empty object at a path. If there is already an object at the given path, it is replaced with an empty object.

**Set path**

Set the current path. This allows relative keys to continue from this path.

**Set value**

Set a number or string value at a path.

**Toggle boolean**

Toggle a boolean value at a path. If the value at the path is not a Boolean, do nothing.

**Add to**

Add a value to the numerical value at a path. If the value at the path is not numerical, do nothing.

**Subtract from**

Subtract a value from the numerical value at a path. If the value at the path is not numerical, do nothing.

## JSON expressions

**ArraySize**

Return the length of an array at a path. If there is not an array at the given path, returns -1.

**Back**

**Front**

Return the element at the start (front) or end (back) of an array at a given path. If there is not an array at the given path, returns -1.

**CurrentKey**

In a *For each* loop, a string of the current key name. If an array is being looped, the current key is a string of the current index, e.g. "0", "1"...

**CurrentType**

In a *For each* loop, a string representing the type of the current value, which can be one of `"null"`, `"array"`, `"object"`, `"boolean"`, `"number"` or `"string"`.

---

**CurrentValue**

In a *For each* loop, the current value. This only returns numbers or strings, or booleans as a number (0 for false and 1 for true). All other types will return 0.

---

**Get**

Get the value at a given path. The path can be relative to the current path or the current key in a *For each* loop. This only returns numbers or strings, or booleans as a number (0 for false and 1 for true). All other types will return 0.

---

**Type**

Get a string representing the type of a value at a given path, which can be one of `"null"`, `"array"`, `"object"`, `"boolean"`, `"number"` or `"string"`. The path can be relative to the current path or the current key in a *For each* loop.

---

**Path**

Return the current path.

---

**ToBeautifiedString**

**ToCompactString**

Return the current JSON data either as a formatted string with line breaks and indentation ("beautified"), or as a minimal string excluding any line breaks or indentation ("compact"). Beautified strings are easier to read, but compact strings are more efficient for storing, sending over the Internet, and loading. Both beautified and compact strings always represent identical data, and there are a range of third-party tools available that can convert between beautified and compact representation.

---

**GetAsBeautifiedString**

**GetAsCompactString**

Return the JSON data at a specified location either as a formatted string with line breaks and indentation ("beautified"), or as a minimal string excluding any line breaks or indentation ("compact"). These expressions are conceptually similar to "ToBeautifiedString" and "ToCompactString" respectively, but for a specific part of the current data instead of everything. In that way they are the opposite half of the "Set JSON" action, which allow you to set a value from a JSON string at a given location.

# KEYBOARD

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

The **Keyboard** object allows projects to respond to keyboard input.

### Scripting

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

## About keyboards

When designing your project, you cannot assume everyone has a keyboard. Many users browse the web with touch-screen devices that have no keyboard. (The Keyboard object also does not respond to input from on-screen keyboards on any modern touch devices.) Therefore if your project uses exclusively mouse or keyboard control, it is impossible to use on touch devices. See the Touch controls tutorial for an alternative control system.

Also note that there are a variety of keyboard layouts used internationally. For example if you only provide "WASD" as direction controls, your project may be difficult to control on AZERTY keyboards. "ZQSD" controls covers the AZERTY layout, but there are many other possible keyboard layouts. In this case, also supporting arrow keys for direction controls will cover most international keyboards, but remember the same problem applies for any other controls depending on a specific key layout.

## Key codes

It's possible to detect key presses by numerical *key codes* with the Keyboard object. A key code is simply a number assigned to every possible key on the keyboard. This can be useful for implementing custom controls, since key codes can be stored in variables.

## Key ghosting

You may notice you cannot reliably detect three or more simultaneous key presses on the keyboard. This is a limitation of common keyboard hardware, not Construct. The circuitry in common keyboards exhibits an effect called *key ghosting* where only certain combinations of a certain number of keys can be reliably detected. You can get special gamer keyboards that support anti-ghosting, but since these are rare it's probably a better idea to design your project around the limitations of common keyboards, such as by avoiding having to hold down lots of keys.

## Keyboard conditions

**Key code is down**

True if a given key by its key code is currently being held down.

**On key code pressed**

Triggered when a specific key code is pressed.

**On key code released**

Triggered when a specific key code is released.

**Key is down**

True if a given keyboard key is currently being held down.

**On any key pressed**

Triggered when any keyboard key is pressed. Useful for title screens or cutscenes. The corresponding key code is set in the *LastKeyCode* expression.

**On any key released**

Triggered when any keyboard key is released. The corresponding key code is set in the *LastKeyCode* expression.

**On key pressed**

Triggered when a specific keyboard key is pressed.

**On key released**

Triggered when a specific keyboard key is released.

**Is keyboard lock supported**

True if the current browser/platform supports using keyboard lock with the *Lock keyboard* action.

**On keyboard locked**

**On keyboard lock error**

Triggered after the *Lock keyboard* action depending on whether the attempt to lock the keyboard completed successfully. If the keyboard is now locked, *On keyboard locked* is triggered; if locking the keyboard failed (including the user declining a permission prompt), *On keyboard lock error* is triggered.

**Left/right key is down**

**On left/right key pressed**

**On left/right key released**

As per *Is key down*, *On key pressed* and *On key released*, but is able to identify the left or right Shift, Control, Alt or Meta keys separately.

# Keyboard actions

**Lock keyboard**

Request to lock the keyboard so key presses are always received by the project, rather than the browser/application (such as to activating browser keyboard shortcuts). For example if WASD controls are used to move an object, and the Control key is also used for another purpose, it is easy to accidentally press Ctrl+W, which is a keyboard shortcut to close the current browser tab. If the WASD keys are locked, then Ctrl+W will act as a normal keyboard input to the project and not activate the browser shortcut. The keys to lock are specified by a string of key names separated by commas, e.g. "KeyW,KeyA,KeyS,KeyD" for the WASD keys - a full list of key names can be found on this MDN page. Pass an empty string to request to lock all keys. Requesting to lock the keyboard may show a permission prompt to the user, as it is a security-sensitive feature. Further, it only works in fullscreen mode, i.e. after using *Request fullscreen* in the Browser plugin. If keyboard lock is successfully enabled, *On keyboard locked* triggers; otherwise *On keyboard lock error* triggers.

> *The condition Is keyboard lock supported checks if this action is supported by the current browser/platform.*

**Unlock keyboard**

If the keyboard was previously locked, this action unlocks it again, restoring the normal handling of key presses (where the browser/application may handle some key presses).

# Keyboard expressions

**LastKeyCode**

Retrieve the key code of the last key press. This is useful in *On any key pressed* or *On any key released* to determine the key code of the key the user pressed, which is useful when setting up custom controls.

**StringFromKeyCode**

Convert a numerical key code back in to a string representation. For example this turns the key code 65 in to the string "A".

**TypedKey**

Return the last key press as the character that would have been entered in to a text field. For example when pressing A, this could be "a", "A", "á" or something else, depending on which other keys are held down. If the last key press is not a typed character, like Shift, then the expression is set to the name of the key.

# LIST

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

The **List** object creates either a **dropdown list** or **list box** form control. A dropdown list only shows one item but can be expanded to show the full list; a list box shows multiple items at once. The image below shows a dropdown list on the left (which has been expanded), and a list box on the right.




*List types*

### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Styling lists

As List objects are HTML elements, their appearance can be customised using CSS (Cascading Style Sheets). The *ID* and *Class* properties can be used to identify the HTML element, and a CSS project file added to apply some styles to it.

## List properties

**Items**

A list of the initial items to display in the list, with one item per line. Click the button to the right of the property to open a multi-line text box in a dialog to edit this more conveniently. The property field cannot show line breaks,
so instead represents lines with the special **\n** escape sequence.

**Tooltip**

A tooltip that appears in most browsers if the user hovers the mouse over the button and waits. Leave blank for no tooltip.

---

**Initially visibile**

Whether or not the control is initially visible in the layout.

---

**Enabled**

Whether the control is initially enabled. If disabled, the control will appear greyed out and the selection cannot be modified.

---

**Type**

Choose between the *List box* and *Dropdown list* control styles. An image displaying the two types is shown above.

---

**Multi-select**

Allow more than one item to be selected when *Type* is *List box*. This has no effect for dropdown lists.

---

**Auto font size**

Automatically set the font-size property of the element according to the layout and layer scale. This will prevent the *font-size* CSS property being manually set with the *Set CSS style* action. Disable if you intend to use *Set CSS style* to adjust the *font-size* property.

---

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

---

**Class Optional**

An optional *class* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

## List conditions

See common conditions for features shared between form control objects.

---

**Compare item text at**

Compare the text of a given item in the list.

**Compare selected item text**

Compare the text of the currently selected item in the list.

**Compare selection**

Compare the zero-based index of the currently selected item.

**On clicked**

Triggered when the control is clicked.

**On double-clicked**

Triggered when the control is double-clicked.

**On selection changed**

Triggered whenever the chosen selection in the control is changed. This can be by any means of input (such as a mouse click, keyboard press, or touch input on mobile).

## List actions

See common actions for features shared between form control objects.

**Add item**

Append a new item to the end of the list of available choices.

**Add item at**

Insert a new item to the list of available choices at a certain zero-based index.

**Clear**

Remove all the available choices from the list.

**Remove**

Delete an item at a specific index.

**Set item text**

Change the text of an item at a specific index.

**Set selection**

Set the item at a specific index as selected.

---

**Set tooltip**

Set the *Tooltip* property of the object, displayed by most browsers when hovering the mouse over the control.

## List expressions

---

**ItemCount**

The current number of items in the list.

---

**ItemTextAt**

Return the text of an item at a zero-based index in the list.

---

**SelectedCount**

The number of currently selected items. This will always be either 0 or 1 unless a list box with *Multi-select* enabled is used.

---

**SelectedIndex**

The zero-based index of the currently selected item. For multi-select lists, use *SelectedIndexAt* instead.

---

**SelectedIndexAt**

The index of a selected item out of all the selected items. In other words, *SelectedIndexAt* with numbers 0 to `SelectedCount - 1` gives the indices of all the selected items.

---

**SelectedText**

The text of the currently selected item. For multi-select lists, use *SelectedTextAt* instead.

---

**SelectedTextAt**

The text of a selected item out of all the selected items. In other words, *SelectedTextAt* with numbers 0 to `SelectedCount - 1` gives the text of each selected item.

# LOCAL STORAGE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/local-storage

The **Local Storage** plugin can **store data locally** on the user's device. For example it can be used to store a personal best high-score. Note if you want a full-state save and load feature, see How to make savegames.

Local Storage works offline, since it stores data to the device itself. Different browsers use different storages, so data is not shared between different browsers on the same computer - each browser has its own unique storage. This is also separate to the browser cache, which is temporary storage to avoid needlessly re-downloading the same files over and over again. However the user can usually still choose to clear local storage data from their browser (with an option perhaps named something like "clear offline website data"). Non-browser export options like desktop exports and mobile apps are not affected by the user clearing any amount of data from any browser, they also use separate storage. Finally for security reasons browsers use separate storage per domain. For example all content on *construct.net* shares the same storage, but content on *facebook.com* uses a different storage and cannot access any data saved from *construct.net*.

### Scripting

When using JavaScript or TypeScript coding, you can use the IStorage interface to access the same storage as Construct uses for this object (there is no dedicated script interface for the Local Storage plugin itself). Further, you can use the browser built-in storage APIs such as IndexedDB for more advanced cases.

## Storage quotas

To prevent abuse, most browsers implement a storage quota, which is a maximum amount of data that can be saved locally. On most modern browsers this is defined as a proportion of the free storage space on the device. You can check the available quota on a device by loading Construct and checking the About dialog which shows the quota available. If the quota is exceeded, the *On error* trigger will fire when writing to storage.

## Using Local Storage

Local Storage uses a very simple storage system: values are stored under named *keys*, similar to how the Dictionary object works. For example the value 100 could be stored under a key named *score*.

Local Storage is asynchronous. This means reading and writing data does not complete immediately. The actions only start the process of reading or writing a value, and the project continues to run in the interim. This ensures that slow or busy storage systems do not impact the performance of the project. When the read or write is complete, a trigger fires (*On item get* or

*On item set*) which indicates either the value is available to read (with the *ItemValue* expression) or that the value was successfully written.

For example here is a flow to read the value of the key "score":

1. Use the action *Get item "score"*
2. A moment later, *On item "score" get* triggers
3. In this trigger, use the *ItemValue* expression to read the item

Here is a flow to set the new value of "score":

1. Use the action *Set item "score" to 100*
2. A moment later, *On item "score" set* triggers
3. You may not need to do anything in this trigger, but it indicates the value has been successfully written. The *Key* and *ItemValue* expressions are still set in this trigger in case you need them.

Note that you must be careful to avoid "races" when using asynchronous storage. For example the *Clear storage* action may take a moment to complete before it fires *On storage cleared*. It is possible to write more values to storage in between, while *Clear storage* is still processing. This is like "racing" the *Clear storage* and *Set item* actions: the end result depends on what order they complete in, which is unpredictable. In this case it is more or less random what happens: the written keys may be cleared, or they may not be - you cannot rely on any specific result. Therefore you should be careful to avoid this case.

## Simplifying usage with Dictionary

Although it improves performance, dealing with asynchronous reads and writes can sometimes be difficult. One simple way to conveniently have synchronous storage is to store an entire Dictionary object's contents to Local Storage, by saving its *AsJSON* string. Then you can load this content from Local Storage with the *Load* action. This means only saving and loading the dictionary contents is asynchronous, and the rest of the time you can use the Dictionary object's features to synchronously access data, such as simply using its *Get* expression to immediately read a value. However you must remember to save the Dictionary again at some point before the user quits the project.

## Storage location on disk

The Local Storage plugin stores all saved data in an internal browser database. This does not produce any easily discoverable files on disk. If you want to save data to easily identifiable files on disk, such as storing save data in a file in the user's Documents folder, consider using the File System plugin instead.

## Redirecting storage

If you have a complex existing project and you decide you want to change how data is saved - such as by saving it to a file with the File System plugin, or using a cloud service - updating all

usages of the Local Storage plugin can be tricky. To help with this you can enable the *In memory only* property of the object. This essentially disables the use of storage and only holds data in memory, and the data will be lost when reloading the page or app. However the data can then be accessed using the *MemoryStorageAsJSON* expression for saving elsewhere, and loaded with the *Load memory from JSON* action, to persist storage somewhere else. Note that if you use this to save to a cloud service, you may want to check the size of the saved data - usually it is not a problem to save a large amount of data locally, but it may turn out to be too much to upload and download.

## Local Storage properties

**In-memory only**

Check to enable in-memory only storage mode. In this mode data will not be persisted if the page or app is closed, but can be stored elsewhere instead - see *Redirecting storage* above for more details.

## Local Storage conditions

**On any item get**

Triggered after any *Get item* action completes.

**On any item removed**

Triggered after any *Remove item* action completes.

**On any item set**

Triggered after any *Set item* action completes.

**On item exists**

Triggered after the *Check item exists* action completes if the key checked does indeed exist. In this trigger the *ItemValue* expression is also set to the value of this key, so there is no need to use another *Get item* action to read it.

> *Note ItemValue is not set if binary data was stored.*

**On item get**

Triggered after a *Get item* action completes for a given key. The *ItemValue* expression is set to the value of the key, except for when reading binary data.

**On item missing**

Triggered after the *Check item exists* action completes if the key checked does not exist.

---

**On item removed**

Triggered after the *Remove item* action completes for a given key.

---

**On item set**

Triggered after the *Set item* action completes for a given key. This indicates the data is now in storage.

---

**Compare key**

Compare the current value of the *Key* expression, which is the name of the current key in a trigger. This can be useful in the *On any item...* triggers.

---

**Compare value**

Compare the current value of the *ItemValue* expression, which is set to the item value when getting an item or in *On item exists*.

---

**Is persistent**

True if the browser has granted the current domain persistent storage permission. See the *Request persistent storage* action for more details.

---

**On all key names loaded**

Triggered after the *Get all key names* action completes. In this trigger the *KeyCount* and *KeyAt* expressions give the list of all the key names.

---

**On error**

Triggered at any time while using Local Storage if an error occurs, such as if a write failed, or the maximum storage quota was exceeded. The *ErrorMessage* expression is set to the error message if available.

---

**On storage cleared**

Triggered after the *Clear storage* action completes and storage is now empty.

---

**Is processing gets**

True if any *Get item* actions are still processing, i.e. any *On item get* trigger is yet to fire for a *Get item* action.

### Is processing sets

True if any *Set item* actions are still processing, i.e. any *On item set* trigger is yet to fire for a *Set item* action.

### On all gets complete

Triggered when all outstanding *Get item* actions are completed, i.e. when *Is processing gets* first becomes false. In other words if 10 *Get item* actions are all used at the same time, *On all gets complete* triggers when all 10 items have been read and fired their *On item get* triggers.

### On all sets complete

Triggered when all outstanding *Set item* actions are completed, i.e. when *Is processing sets* first becomes false. In other words if 10 *Set item* actions are all used at the same time, *On all sets complete* triggers when all 10 items have been written and fired their *On item set* triggers.

## Local Storage actions

### Check item exists

Check if a key exists in storage. This triggers either *On item exists* if the key exists, or *On item missing* if the key does not exist. If the item exists, the *ItemValue* expression is set to the key value in the *On item exists* trigger, so there is no need to use a subsequent *Get item* action to read the value.

> *Note ItemValue is not set if binary data was stored.*

### Get item

### Get binary item

Read the value of a key in storage. This triggers *On item get* when the value has been read. When reading binary data, the data will be written to the chosen Binary Data object; otherwise the *ItemValue* expression is set to the value of the key.

### Remove item

Remove (delete) a key from storage. This triggers *On item removed* when the key has been removed.

### Set item

### Set binary item

Set the value of a key in storage. This triggers *On item set* when the value has been written. When setting binary data, the contents of a Binary Data object are written; otherwise the text or number provided is used.

---

**Clear storage**

Remove (delete) all items from storage, reverting it back to the empty state. This triggers *On storage cleared* when completed.

---

**Get all key names**

Retrieve a list of all the key names that currently exist in storage. This triggers *On all key names loaded* when the list has been loaded, where the *KeyCount* and *KeyAt* expressions can be used to access the list.

---

**Request persistent storage**

Request that storage be made persistent for the current domain. This typically only applies to web browsers, as all other export options already use persistent storage. In the context of a web browser, all storage is preserved on a "best effort" basis, but may be erased in various situations such as if too much storage space is being used, if the site has not been visited for too long, and so on. If persistent storage permission is granted, then the browser will avoid automatically deleting the storage wherever possible; it may also avoid deleting the storage even if the user chooses to clear storage in browser settings, unless they specifically acknowledge they want to delete the site's data. Requesting persistent storage may show a visible prompt to the user asking them to accept permission, and this action may only be allowed to be used in a user input trigger (such as mouse click or touch). The action is asynchronous, meaning it can be used with *Wait for previous actions to complete*, after which the permission will have been granted or refused. The *Is persistent* condition reflects the current persistent permission state.

---

**Load memory from JSON**

Only applicable when the property *In-memory only* is enabled. Load the entire state of the storage - including all keys and values - from a string of JSON data previously returned by the *MemoryStorageAsJSON* expression. For more details on usage, see *Redirecting storage* above.

## Local Storage expressions

---

**ItemValue**

The value for a key that has been read or written in a Local Storage trigger, such as *On item get* or *On item exists*. This can be either a string or a number. This returns 0 if used outside of a Local Storage trigger, or if binary data was stored instead of text or a number.

**Key**

The name of the key that was modified in any Local Storage trigger, such as *On item get*, *On item set* or *On any item set*. This returns an empty string if used outside of a Local Storage trigger.

**ErrorMessage**

In *On error*, the text of the error message if any is available.

**KeyAt(index)**

In *On all key names loaded*, the name of the key at the given zero-based index in the list.

**KeyCount**

In *On all key names loaded*, the number of key names in the list.

**MemoryStorageAsJSON**

Only applicable when the property *In-memory only* is enabled. Returns the entire state of the storage - including all keys and values - as a string of JSON data. This can then be loaded again later with the *Load memory from JSON* action. For more details on usage, see *Redirecting storage* above.

# MIDI PLUGIN

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

The **MIDI plugin** allows communicating with connected MIDI (Musical Instrument Digital Interface) devices via the Web MIDI API. Typically MIDI devices are digital musical instruments such as a MIDI piano keyboard or synthesizer. The MIDI plugin can both receive MIDI input from devices, such as detecting when notes are pressed, as well as send MIDI output to devices, such as to create an arpeggiator.

In order to use a MIDI device it will need to be connected to the device that the Construct project is running on. Some MIDI devices may require a special 5-pin MIDI cable, but others may be able to use another standard such as USB.

You can find two examples of using the MIDI plugin in the MIDI input and MIDI output examples.

> *Familiarity with the MIDI protocol may be useful when using MIDI. You can search the web for the full specification, but there's also a concise unofficial third-party reference here.*

> *As of February 2025, Safari does not support the Web MIDI API and so the MIDI plugin will not work in web exports loaded in Safari, or in macOS WKWebView exports. Use the Is supported condition to check if MIDI is supported on the current browser/platform.*

## MIDI access

For privacy and security reasons, browsers do not provide access to MIDI devices by default. Therefore the first thing your project must do before using MIDI is use the *Request MIDI access* action. This may show a permission prompt to the user. This action may also only be allowed in a user input trigger, such as in a button click or touch event, although some browsers may allow it to be used on startup (in *On start of layout*).

The user must approve any permission prompt - if they do so, *On request access success* is triggered, and MIDI devices may then be used. If the user declines the permission prompt *On request access failed* will be triggered and access to MIDI will be blocked. In this case you may wish to show a message to the user.

There are two options when requesting MIDI access:

- **System exclusive:** when enabled, this allows for sending and receiving system exclusive (sysex) MIDI messages. These are usually device or vendor specific. As these messages have potentially unlimited capability, such as wiping data stored on the device or overwriting

its firmware, there may be stricter security requirements for using this mode; therefore it is recommended to leave it disabled if you don't need it.

- **Software:** when enabled then virtual or software MIDI devices running on the same system as the Construct project may be listed as available. If you are only interested in using separate physical MIDI hardware, you should leave this disabled.

## MIDI devices

By default, after successfully gaining MIDI access, no MIDI messages will be sent or received. To use a MIDI device, it must first be *opened*. This means the device becomes active and messages may be sent or received. If the device is no longer needed it may then be *closed* to return it to an inactive state that will not send or receive MIDI messages.

There are two types of MIDI devices: input, from which messages are received, and output, to which messages may be sent. If a single piece of MIDI hardware supports both input and output, it will appear as two devices - both an input device and an output device. Sometimes this will also require two cables, one for input and one for output. Be sure to make sure the appropriate MIDI sockets are used - for example to receive MIDI input from a device, you may need to connect a cable from the device's MIDI output socket to the MIDI input socket of the device the Construct project is running on.

The MIDI plugin supports input and output to multiple devices simultaneously. To allow for this, devices are identified by an ID string provided by the browser or system. Usually the ID string can be left empty to refer to either the first available device, or the last opened device, which is a convenient way to use a single device.

## MIDI notes

In the MIDI specification, notes are identified by a number from 0-127, with number 60 referring to middle C. With Construct's MIDI plugin, for convenience notes may also be referred to by a string of the note name. For example number 60 (middle C) may be referred to by the string "C4", meaning note C in octave 4.

Note names may also use a sharp ("#") or flat ("b") to adjust the note by a semitone in either direction, for example "D#4" or "Db4". This means both "C#4" and "Db4" refer to the same note. Construct also treats "Cb" as equivalent to "B", and "B#" as equivalent to "C".

The *NoteNumberToName* and *NoteNameToNumber* expressions allow converting between note number and note name representations. Note that when converting from a note number to a note name, Construct will always use a sharp notation. For example note 61 is both "C#4" and "Db4", and Construct will choose the name "C#4" in this case.

## MIDI conditions

---

**Has access**

True if MIDI access is available, i.e. requesting MIDI access has completed successfully.

**Is supported**

True if MIDI access is supported on this browser/platform. This depends on support for the Web MIDI API.

**On devices state changed**

Triggered when any MIDI devices are connected, disconnected, or changed. If your project displays a list of available MIDI devices, it should update the list in this trigger.

> *This may trigger multiple times in quick succession - for example if a single MIDI hardware device that provides both an input and an output is removed, then this trigger may fire twice as both the input and output become unavailable.*

**On request access success**

**On request access failed**

Triggered after the *Request MIDI access* action, depending on the outcome of the request.

**On any device opened**

**On device opened**

Triggered after the *Open device* action when the device has been opened and can now send or receive MIDI messages. With the *any* variant, the relevant MIDI device can be identified by expressions like *CurrentDeviceID*.

**On any device closed**

**On device closed**

Triggered after the *Close device* action when the device has been closed and so will no longer send or receive MIDI messages. With the *any* variant, the relevant MIDI device can be identified by expressions like *CurrentDeviceID*.

**On any device error**

**On device error**

Triggered if an error occurs when accessing a MIDI device. With the *any* variant, the relevant MIDI device can be identified by expressions like *CurrentDeviceID*.

**Compare parameter**

Compare an aspect of a MIDI message such as the note number or velocity.

**On any binary message**

Triggered when any MIDI message is received from any device. The raw binary data of the MIDI message is stored in a specified Binary Data object. The associated MIDI device can be identified by expressions like *CurrentDeviceID*. Using the binary data will require knowledge of the MIDI specification - for example a three-byte message in hexadecimal notation 0x90 0x3C 0x40 refers to Note On, MIDI channel 0, note 60 (middle C), velocity 64.

### On any message

Triggered when a MIDI message of a specific type, e.g. Note On, is received from any MIDI device. The associated MIDI device can be identified by expressions like *CurrentDeviceID*. Details about the message can be compared with *Compare parameter* or retrieved via expressions.

### On message

Triggered when a MIDI message of a specific type, e.g. Note On, is received from a specific MIDI device given by its ID. The device ID may be left as an empty string to refer to the last opened input device. Details about the message can be compared with *Compare parameter* or retrieved via expressions.

## MIDI actions

### Request MIDI access

Request to access MIDI devices. This must be done before any MIDI devices can be used, and may show a permission prompt to the user. For more details see the section on *MIDI access* above.

### Open device

Open a MIDI device specified by its device ID and type (input/output). The device ID may be left as an empty string to open the first available device; however note if multiple devices are available, this may not select the device you really want, so generally you should provide a way for the user to pick the device (see the MIDI input/output examples in Construct). Once a device is opened, it may start sending or receiving messages (depending on whether it is an input or output device).

### Close device

Close a MIDI device specified by its device ID and type (input/output). The device ID may be left as an empty string to close the last opened device (useful when only one device is opened at a time). Once a device is closed, it will no longer send or receive messages.

### Schedule next message

This action will cause the next *Send* action to actually send the message after a given delay in seconds. This can be used to schedule messages to be sent with higher accuracy than ticks in Construct, which typically run approximately every 16ms (for a 60 Hz display).

---

**Send binary message**

Send raw binary data as a MIDI message to an output device ID. The ID may be left as an empty string to send a message to the last opened MIDI output device. The message contents is taken from a chosen Binary Data object. Using the binary data will require knowledge of the MIDI specification - for example a three-byte message in hexadecimal notation 0x90 0x3C 0x40 refers to Note On, MIDI channel 0, note 60 (middle C), velocity 64.

---

**Send control change message**

Send a MIDI Control Change message to an output device ID. The ID may be left as an empty string to send a message to the last opened MIDI output device. Refer to the MIDI specification for possible controller numbers; a common one to use is controller number 1 to refer to the modulation wheel (coarse). The controller value is a number in the range 0-127.

---

**Send message**

Send a Program Change, Channel Pressure or Pitch Wheel message to an output device ID. The ID may be left as an empty string to send a message to the last opened MIDI output device. The value is a number in the range 0-127 for Program Change or Channel Pressure messages, or 0-16383 for Pitch Wheel messages.

---

**Send note message**

Send a Note Off, Note On or AfterTouch message to an output device ID. The ID may be left as an empty string to send a message to the last opened MIDI output device. The note may be specified by either its MIDI note number (0-127, e.g. 60 for middle C) or a note name e.g. "C4". The note velocity is given as a number 0-127.

> *Some devices will treat a Note On message with velocity 0 as Note Off.*

---

**Set MIDI channel**

Set the MIDI channel of outgoing messages, from 0-15. The default is channel 0. Some devices will be configured to only receive messages on a certain channel and will have a setting to select the MIDI channel number of the device; note that many consumer devices will use 1-based indexing for the MIDI channel (so the first MIDI channel is displayed as channel 1, rather than channel 0).

## MIDI expressions

**InputCount**

**InputIDAt(index)**

**InputManufacturerAt(index)**

**InputNameAt(index)**

Retrieve the list of available input devices. These expressions provide the number of input devices, and the device ID, manufacturer, and name of the device at a given zero-based index. Note this list first becomes available in *On request access success*, and may change in *On devices state changed*.

---

**OutputCount**

**OutputIDAt(index)**

**OutputManufacturerAt(index)**

**OutputNameAt(index)**

Retrieve the list of available input devices. These expressions provide the number of input devices, and the device ID, manufacturer, and name of the device at a given zero-based index. Note this list first becomes available in *On request access success*, and may change in *On devices state changed*.

---

**CurrentDeviceID**

**CurrentDeviceManfucaturer**

**CurrentDeviceName**

In a trigger, such as *On any message*, these expressions contain information about the device that sent the message.

---

**Channel**

When a MIDI message is received, this returns the MIDI channel of the message from 0-15.

---

**ControllerNumber**

**ControllerValue**

When a Control Change message is received, these return the controller number (0-127) and controller value (0-127) specified in the message.

---

**NoteName**

**NoteNumber**

When a note message is recieved (Note Off, Note On or After Touch), these return the associated note name as a string (e.g. "C4") or MIDI note number (0-127, e.g. 60 for middle C).

**PitchWheelValue**

When a Pitch Wheel message is received, this returns pitch wheel position in the range 0-16383.

**Pressure**

When an After Touch or Channel Pressure message is received, this returns the pressure value (0-127).

**ProgramNumber**

When a Program Change message is received, this returns the program number (0-127).

**Velocity**

When a Note On or Note Off message is received, this returns the velocity in the range 0-127. If the device does not support detecting note velocity, a default value such as 64 may be sent instead. Some devices send a Note On message with velocity 0 to indicate Note Off; in this case the velocity will be returned as 64 as a default value given the velocity is not actually specified in that case.

**NoteNameToNumber(NoteName)**

**NoteNumberToName(NoteNumber)**

Convert between a note name (e.g. "C4" for middle C) and a MIDI note number in the range 0-127 (e.g. 60 for middle C). See the section on *MIDI notes* above for more information.

> *NoteNumberToName will always use the sharp variant, e.g. note number 61 will always return "C#4" and never "Db4".*

# MOBILE ADVERT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/mobile-advert

The **Mobile Advert** plugin allows a game to use **Admob** Banner, Interstitial, Rewarded and Rewarded Interstitial adverts on **Android** and **iOS** exports.

Web exports use Google's Ad Placement API to show Interstitial and Rewarded adverts.

> ***Admob** adverts can only be shown in **Android** or **iOS** exports. Adverts shown in local preview, remote preview, or with any export option other than Android or iOS will use **Google's Ad Placement API**.*

## Advert types

Construct supports the following advert types:

**Banner Advert**

Displays an advert that partially overlays the bottom of the screen, the size of the advert can be specified when it's created. **(Mobile export only)**

**Interstitial Advert**

Displays a full screen advert, useful for transitioning between layouts. **(Mobile & Web exports)**

**Rewarded Advert**

Similar to an interstitial advert in that it is a full screen advert. If the user watches this advert for the designated amount of time, they will get the reward payload. If the advert is dismissed before that, the reward is not given. **(Mobile & Web exports)**

**Rewarded Interstitial Advert**

Similar to rewarded adverts, but users aren't required to opt in to view a rewarded interstitial. Instead of the opt-in prompt in rewarded ads, rewarded interstitials require an intro screen that announces the reward and gives users a chance to opt out if they wish to do so. **(Mobile & Web exports)**

## Getting started with Admob

Follow these steps to quickly get started with Admob on mobile exports:

1. Create an Admob account - see: Sign up for AdMob
2. Create an **application** on your Admob account - see: Set up an app in Admob
3. Create **ad units** for your **application** - see: Ad units
4. Copy your application IDs and paste them to the Mobile Advert **Android Application ID** and **iOS Application ID** properties. Finding your Application IDs
5. Submit your business and payment details to your AdMob account and wait for them to verify (which may take 24 hours). This must be done before ads can be served.

## Getting started with web adverts

Follow these steps to quickly get started with adverts on web exports:

1. Sign up for an Adsense account if you already don't have one
2. You will need a website which is whitelisted to show these type of ads
3. Fill in this form to indicate interest in using Google's Ad Placement API

### How do I get my site whitelisted?

1. Visit the **Sites** options in the left hand side menu of your Adsense account
2. Find the **Add site** button and click it to start adding your website
3. Follow the steps to add your website
4. You will need to modify the root of your website by adding a small script to it
5. Request a review

After sometime, hopefully your website will be approved and your game will be able to show ads.

### Considerations for web adverts

1. You will need to have your website be a real one, that means that it should have real content and generally speaking look like it is serving a purpose. If you just setup a test website with no content, it is less likely it will be approved.
2. Since the parent website your game will be hosted in needs to be whitelisted so the game can show ads, the ideal use case is you have control of the hosting website.
3. If a game is shown in a cross origin iframe, it won't be allowed to show ads.

## User Consent (Mobile exports only)

In order for the **Show User Consent Dialog** action to show **GDPR** consent forms and **IDFA** messages (iOS only), it is required to set them up in your Admob account.

### GDPR message

Follow this guide to Create a GDPR message

### IDFA message

Follow this guide to Create an IDFA message

> ***IDFA*** *messages are not strictly necessary, but they can help in getting consent from the user by making the purpose of the dialog clearer.*

> *If you have an* ***IDFA*** *message set up and the application is opened by a device in the* ***EEA*** *or* ***UK****, the* ***App Tracking Transparency*** *dialog will be shown immediately after the* ***EU*** *consent form. This is how the underlying SDK works and it is mentioned in Google's own quick start guide.*
> *However, if you disable your* ***IDFA*** *message or don't have one at all, then the* ***App Tracking Transparency*** *dialog won't be shown automatically and you will be able to control when it appears by using the* ***Request IDFA*** *action.*

> *Note when you first set up your account and submit your business/payment details, it can take up to 24 hours to complete verification. Wait for this to complete before trying to test your adverts, since it may fail to work until verification has completed.*

## Mobile exports required properties

If you intend to show ads in mobile exports, either **Android** or **iOS**, you will need to specify your application IDs on the Mobile Advert object. Without these values the plugin will not be able to start.

## Web exports required properties

If you intend to show ads in a web export, you will need to specify your publisher ID in the Mobile Advert object. Without that value, the plugin will not be able to show web adverts.

That's all the configuration you need for now. You can create new applications and edit your advert units later if you need.

## Initialization

The plugin will automatically initialize when the app starts. However this may involve showing a user consent prompt to the user for regulatory reasons, depending on the user's region. Therefore you should wait for the **On configuration complete** trigger to fire before attempting to create or show any ads. You should also use the **On configuration failed** trigger, and perhaps the **Is configured** condition, to check if configuration failed for any reason, such as being on an invalid platform or one of your properties being incorrect.

*Don't want to show the user consent dialog at startup? By default users in the **EEA & UK** (European Economic Area and United Kingdom) will see the user consent dialog before your title screen the first time they start the game. If you don't want this to happen then uncheck the property **Show on startup**. Then use the **Show user consent dialog** action to show the dialog at a time of your choosing.*

### Using IDFA on iOS

As of iOS 14+, the *identifier for advertisers* or **IDFA** is no longer available to apps by default. This can affect the performance of ads. To use the **IDFA**, you must prompt the user for permission to access it with the **Request IDFA** action. You may also wish to do this on startup. The **On IDFA request complete** trigger fires when the user makes a decision, and the **IDFAStatus** expression indicates the outcome.

*Requesting the **IDFA** requires using Xcode 12+ with iOS 14+.*

*Currently no prompt is required on Android, so the **Request IDFA** action will immediately trigger **On IDFA request complete** with an **IDFAStatus** of "not-determined".*

### The App Tracking Transparency framework on iOS

In order to support the **Request IDFA** action, the Mobile Advert plugin includes an iOS library called **App Tracking Transparency**. To comply with Apple's App Store review process you may be required to indicate to the reviewers where your app uses the **IDFA** prompt. Therefore you may be required to add the **Request IDFA** action somewhere in your project. You do not have to always request it on startup - it could for example be a setting in a menu, and then you indicate to the App Store reviewers where to find the option.

## Advert Loading

Adverts need to load over the network before you can use them. When creating an advert you can choose to "show" the advert immediately after it has loaded. However if it's more suitable for your game you can choose to load your advert ahead of time by creating the advert, but not showing it, then later displaying the advert when you need it. This will prevent your user experiencing a lag while the advert loads. Be sure to check your advert has actually loaded before trying to display it, loading times for rewarded adverts in particular can be quite long. It's also worth remembering that you are not guaranteed to receive an advert when you request one, so take that into account when you are making your events.

## Banner Advert Sizes

To display a banner advert you must specify a size for it, generally smart portrait/landscape will work for most situations but here are the specifiable sizes. Using a size that is larger than the display will prevent it from loading. Smart banner will vary in height depending on the available screen height. With a smart banner if the base image isn't wide enough to fill the screen black bars are displayed on either side.

**Smart portrait**

Screen width x 32/50/90

**Smart landscape**

Screen width x 32/50/90

**Standard**

320 x 50

**Large**

320 x 100

**Medium**

300 x 250

**Full (tablet only)**

468 x 60

**Leaderboard (tablet only)**

728 x 90

*All banner sizes are in device pixels. That means that a banner will have an actual size according to the pixel density of the device it is being shown in.*

## Mobile Advert Properties

**Testing mode**

Changes the displayed adverts units to testing adverts. Should be used during development. **(Mobile & Web exports)**

**Android application ID**

The Admob application ID for the Android version of your game. This has the format "ca-app-pub-0000000000000000~0000000000". On Android this value is required, and your application will fail to start unless it is provided and correct. Find your app ID **(Mobile exports only)**

**iOS application ID**

The Admob application ID for the iOS version of your game. This has the format "ca-app-pub-0000000000000000~0000000000". On iOS this value is required, and your application will fail to start unless it is provided and correct. Find your app ID **(Mobile exports only)**

**Spoof location (debug)**

Fakes the device location to allow testing the user consent dialog in different scenarios. Testing mode must be enabled for this to work. **(Mobile exports only)**

**Show on start up**

Choose wether to show the consent form automatically on startup or not. This option doesn't do anything in web exports. **(Mobile exports only)**

**Publisher ID**

Adsense publisher ID, found in your Adsense account. This is required if you intend to show adverts in a web export. Find your publisher ID **(Web exports only)**

### *Test Ads*

*It is important that when you're developing and testing your application you use test adverts. Viewing real adverts during your testing may result in your Admob account being suspended or closed! Testing mode can be enabled by setting the Testing mode property on the plugin. Testing mode is enabled by default so remember to turn it off when you publish your game!*

### *GDPR Consent Form in Testing Mode*

*When Testing mode is enabled, the GDPR consent form will be shown every time the application starts, regardless of previous choices. When you are ready to deploy, disable*

*Testing mode so the form can behave normally and only be shown the first time the application is executed after installing it.*

### *Spoof location*

*Unfortunately at the moment it is not possible to use this feature in a real device with iOS version 14 or greater. This is because to enable the feature, it is necessary to have access to the **IDFA** beforehand, which is not possible without first showing the **App Tracking Transparency** dialog. Since the dialog can only be shown one time, enabling the feature makes it impossible to later test how the application would behave upon first installing it. The feature can still be used in a simulator though.*

### *Ads in preview*

*When previewing a project the application will behave as a web export, so it will show test ads as if it was a web export. Additionally it will simulate frequency capping, alternating between a valid ad request and an invalid ad request.*

## Differences between mobile and web adverts

### Ad units

If you only want to show web ads, it is not necessary to provide ad units in your actions, the field can be left blank. If you are planning to use the same project for web and mobile though, you will still need to provide them.

### Rewarded ads

There is a slight difference between rewarded ads in mobile and rewarded ads in web that should be considered.

In mobile when the ad is viewed you can use the **RewardType** and **RewardValue** expression to get information about the reward and this information is set in your Admob account through the ad unit.

In a web export, because you don't use ad units, the rewarded ads will always return the value *"Reward"* for the **RewardType** expression and the value of *"1"* for the **RewardValue** expression.

For this reason it is recommended to use the Platform Info plugin to decide what to do after a rewarded ad is viewed, depending on what platform the game is running on.

## Mobile Advert Conditions

**On Banner Ready**

Triggers when a banner advert has loaded.

**On Interstitial Ready**

Triggers when a interstitial advert has loaded.

**On Rewarded Ready**

Triggers when a rewarded advert has loaded.

**On Rewarded Interstitial Ready**

Triggers when a rewarded interstitial advert has loaded.

**On Banner Failed to Load**

Triggers when a banner advert fails to load.

**On Interstitial Failed to Load**

Triggers when a interstitial advert fails to load.

**On Rewarded Failed to Load**

Triggers when a rewarded advert fails to load.

**On Rewarded Interstitial Failed to Load**

Triggers when a rewarded interstitial advert fails to load.

**On Banner Shown**

Triggers when a banner advert has been displayed.

**On Interstitial Complete**

Triggers when a interstitial advert has closed.

**On Rewarded Complete**

Triggers when a rewarded advert has closed, and the user has been rewarded.

**On Rewarded Interstitial Complete**

Triggers when a rewarded interstitial advert has closed, and the user has been rewarded.

**On Banner Hidden**

Triggers when a banner advert has been hidden.

**On Interstitial Cancelled**

Triggers when a interstitial advert has been cancelled.

**On Rewarded Cancelled**

Triggers when a rewarded advert has been cancelled.

**On Rewarded Interstitial Cancelled**

Triggers when a rewarded interstitial advert has been cancelled.

**On Configuration Complete**

Triggers when the application ID has been successfully set.

**On Configuration Failed**

Triggers when the application ID failed to be set.

**On IDFA request complete**

Triggered after the *Request IDFA* action once the user has made a decision. The outcome of their decision is reflected in the *IDFAStatus* expression.

**Is Configured**

True if the plugin has been succesfully configured.

**Is Showing Banner**

True if a banner advert is being shown.

**Is Showing Interstitial**

True if a interstitial advert is being shown.

### Is Showing Rewarded

True if a rewarded advert is being shown.

### Is Showing Rewarded Interstitial

True if a rewarded interstitial advert is being shown.

### Is Banner Loaded

True if a banner advert is ready to be shown.

### Is Interstitial Loaded

True if a interstitial advert is ready to be shown.

### Is Rewarded Loaded

True if a rewarded advert is ready to be shown.

### Is Rewarded Interstitial Loaded

True if a rewarded interstitial advert is ready to be shown.

### Is in EEA or unknown

True if the SDK consider the device to be inside the EEA or is unable to detect the location. As it is a requirement to show the dialog inside the EEA "unknown" is considered effectively the same as being in the EEA. In web exports this condition always evaluates to "true".

## Mobile Advert Actions

### Create Banner

Create a banner advert with a AdMob advert unit ID and size. Optionally show when loaded. Only one banner may be created at a time. If the **reload** parameter is set to **true** an existing banner will be reloaded with the new settings. Setting **overlap** to **true** causes the banner to overlap the main Webview, If **false** the main Webview will be resized so both it and the banner fit in the screen without overlapping. In the case of setting **overlap** to **true** the **offset** parameter can be used to specify an amount by which the banner should be offset from it's initial position.

### Create Rewarded

Create a rewarded advert with a AdMob advert unit ID. Optionally show when loaded. Only one rewarded advert may be created at a time.

**Create Interstitial**

Create a interstitial advert with a AdMob advert unit ID. Optionally show when loaded. Only one interstitial may be created at a time.

**Create Rewarded Interstitial**

Create a rewarded interstitial advert with a AdMob advert unit ID. Optionally show when loaded. Only one rewarded interstitial may be created at a time.

**Show User Consent Dialog**

Shows the modal user consent dialog

**Show Banner**

Show a loaded banner advert. Will only display if there is a loaded banner ready.

**Show Rewarded**

Show a loaded rewarded advert. Will only display if there is a loaded video ready.

**Show Interstitial**

Show a loaded interstitial advert. Will only display if there is a loaded interstitial ready.

**Show Rewarded Interstitial**

Show a loaded rewarded interstitial advert. Will only display if there is a loaded rewarded interstitial ready.

**Hide Banner**

Hide an active banner advert (destroys the advert). Does nothing if no banner is available.

**Request IDFA**

Prompt the user for permission to use the *identifier for advertisers* (IDFA). When the user makes a decision, *On IDFA request complete* triggers, and the outcome is indicated in the *IDFAStatus* expression. See the section on using the IDFA above.

**Set Max Advert Content Rating**

Filter the viewed adverts based on the digital content label classifications for various age groups. This action is not supported in web exports.

**Tag For Child Directed Treatment**

Request advert content that is child-directed for the purposes of the Children's Online Privacy Protection Act (COPPA). This action is not supported in web exports.

---

**Tag for under age of consent**

Indicate the user should be treated as under the age of consent as per the restrictions in the European Economic Area(EEA) General Data Protection Act(GDPR). This action is not supported in web exports.

## Mobile Advert Expressions

---

**ErrorMessage**

When in a failure condition, an error message related to it.

---

**RewardType**

When in a **On Rewarded Complete** condition, the type of the related reward. In web exports this expression always returns the value "Reward".

---

**RewardValue**

When in a **On Rewarded Complete** condition, the value of the related reward. In web exports this expression always returns the value "1".

---

**RewardInterstitialType**

When in a **On Rewarded Interstitial Complete** condition, the type of the related reward. In web exports this expression always returns the value "Reward".

---

**RewardInterstitialValue**

When in a **On Rewarded Interstitial Complete** condition, the value of the related reward. In web exports this expression always returns the value "1".

---

**ConsentStatus**

The current user consent status as a string, can take the values of: UNKNOWN, NOT_REQUIRED, REQUIRED or OBTAINED. In web exports this expression always return the UNKNOWN value.

---

**IDFAStatus**

Indicates the outcome of requesting to use the IDFA. If no request has been made this defaults to "not-determined". After a request it is either "authorized" or "denied" depending on

the choice the user made. In web exports this expression always return the "not-determined" value.

# MOBILE IAP

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/mobile-iap

The **Mobile IAP** plugin allows a game to use consumable and non-consumable **in app purchases** (IAP) on Android and iOS.

> *IAP can only be used in Android or iOS apps. IAP will not work in local preview, remote preview, or with any export option other than Android or iOS.*

## Consumable vs Non-consumable

Non-consumable purchases can be bought once per user. A good use for this type of purchase is to distribute your game for free but with the majority of the content locked from the user. To unlock the rest of the content the user can purchase a premium upgrade. The plugin keeps track of these types of purchases for you across multiple user devices.

Consumable purchases can be bought multiple times by each user. A good use for this type of purchase is to enable a timed bonus or virtual currency. You need to keep track of these types of purchase yourself.

## Setup

Before adding purchases to you game you will need to set up your application on Google Play and/or the App Store.

To register an app on Google Play visit https://play.google.com/apps/publish. To enable in app purchases you will also need to setup a Google payments merchants account at http://www.google.com/wallet/merchants.html.

To register an app on the App Store visit https://itunesconnect.apple.com/login

## Registration stage

When making IAP events, the first thing you need to do is to complete the plugin's registration stage. You won't be able to make any purchases or check if a user owns a product until this is done. For each product you have you must call the *Add Product ID* action with the product ID and type (consumable or non-consumable). To finish registration you must call the *Complete Product Registration* action. If registration succeeds then *On Registration Success* triggers. After this you are free to check the state of products and make purchases.

*Owned products may appear as "available" instead of "owned" in the "On Registration Success" trigger, but will resolve soon afterwards. So it's worth loading store logic before you need it. You can use the "On product owned" trigger to observe when a product becomes owned.*

You only need to register the products once per session. Using the *Add product ID* or *Complete product registration* actions after calling registration will have no effect.

## Displaying products

When offering a product to a user it is important to first ensure that it is available for purchase, otherwise the user will not be able to purchase it. A product may not be available if it is already owned, is invalid, has been flagged as unavailable by the store or (if consumable) is in the process of being purchased. This plugin also includes expression to get the name and description of a product from the store. It's important that your application uses these where appropriate rather than text that you have included in your application, as the app store is known to reject applications that don't follow this rule.

## Keeping track of purchases

It's important to consider how you're going to keep track of user purchases before you start adding IAP events. Non-consumable purchases are tracked by the app stores, so you can easily find out if the user owns it. However, consumable products are a bit more tricky. The app stores don't track these, so you need to store information about these yourself. Initially it may seem appealing to store this information in Local Storage or similar, and while this works quite well for local testing it has a big issue: synchronization. If a user shares their account across multiple devices only the device that performed the purchase will know of it. This is also a pretty big issue if they move to a new device or reinstall your game, as they will lose content **they have paid for** and this tends to upset users. So best practice is to store this information in a remote database somewhere, that you can connect to from your game to check the purchases a user has made.

Exactly how you set this up very much depends on the game you are making and if you are offering a short timed bonus you may not even need a database. A simple setup would be a single number representing a user's balance of "magic gems" or similar. With a more complicated game you may choose to store the entire game state on the server; the world, balance of virtual currency, purchases of virtual goods made with virtual currency and purchases of virtual currency with real currency. A complete setup like this has the bonus of a user not losing virtual goods when moving device, and being able to play the game on multiple devices, but does increase your network dependence.

## Testing

Both the app store and the play store offer mechanisms for testing purchases without spending real money. You should use these to confirm that your products have been correctly configured,

and that your purchase flow works correctly for successful and unsuccessful purchases as well as checking ownership on subsequent runs of your application. While the plugin unifies purchases for iOS and Android there are subtle differences in timings and product behaviour that you may experience. These timing differences can upset your purchase flow, so it's important to check any timing assumptions you make for one platform hold for the other.

## Trigger order

Here are the sequence that triggers fire in for various scenarios.

### Product not owned

On startup:

1. On registration success
2. On product available

On performing a successful purchase:

1. On purchase success
2. On transaction finished
3. On product owned

On performing a failed purchase:

1. On purchase failed

### Product already owned

On startup:

1. On registration success
2. On product owned

## IAP properties

**Validator URL**

Optional URL of a receipt validation service to verify in-app purchases with. This is for advanced users to implement a server to ensure purchases are valid. The URL is passed to the underlying `cordova-plugin-purchase`; refer to the Cordova plugin documentation on store.validator for technical details on how to set up or implement a validation service.

## IAP conditions

**On Purchase Success**

Triggers when a specific product purchase succeeds.

**On Any Purchase Success**

Triggers when any product purchase succeeds.

**On Purchase Failed**

Triggers when a specific product purchase fails.

**On Any Purchase Failed**

Triggers when a any product purchase fails.

**On Registration Success**

Triggers when registration has been completed (after the *Complete registration* action). This is a good time to check if a product is owned. You should wait for this trigger before attempting any purchases.

**On Registration Failure**

Triggers when registration failed. If this occurs then you won't be able to make any purchases.

**On Product Available**

Triggers when a specific product becomes available to purchase.

**On Any Product Available**

Triggers when any product becomes available to purchase.

**On Product Owned**

Triggers when a specific product becomes owned. This triggers both after a first purchase for it, and on startup when the product was previously purchased, allowing easily identifying if the purchased product can be made use of.

**On Any Product Owned**

Triggers when any product becomes owned.

**Product Owned**

True if the current user owns the product.

**Product Available**

True if the current user can purchase the product.

**Store Registered**

True if the registration stage successfully completed.

**On transaction finished**

Triggered when the store transaction for a purchase has finished. The transaction ID is available in the *TransactionID* expression.

## IAP Actions

**Add product ID**

Add a new product to the plugin by specifying the ID and type (consumable or non-consumable). This can only be called in the registration stage.

**Complete product Registration**

Ends the registration stage. After this has been called you will no longer be able to register products. This must be called before you can purchase products. *On registration success* will trigger if successful.

**Restore Purchases**

Restores user purchases. This is not necessary on Android.

**Purchase Product**

Triggers the purchase of a product with a specific ID. This product must be available to purchase. *On Purchase Success/Failed* will trigger depending on the outcome of the purchase. If the purchase is successful, *On product owned* will also trigger, as well as on startup in future sessions while the product is still owned.

## IAP Expressions

**ProductName(ProductID)**

Get the name of a product from its ID. This is the localized name provided by the store, you should use it instead of a hard-coded string.

**ProductPrice(ProductID)**

Get the price of a product from its ID. This is the localized value provided by the store, you should use it instead of a hard-coded value.

---

**ProductDescription(ProductID)**

The description of a product from its ID. This is the localized string provided by the store, you should use it instead of a hard-coded string.

---

**ProductID**

The ID of the current product in a trigger.

---

**TransactionID**

In *On transaction finished*, the ID of the transaction that finished, using the ID provided by the store.

---

**ErrorMessage**

In an error trigger, the relevant error message, if any.

# MOUSE

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

The **Mouse** object allows projects to respond to mouse input.

When designing your project, you cannot assume everyone has a mouse. Many users browse the web with touch-screen devices that have no mouse. Therefore if your project uses exclusively mouse or keyboard control, it may be impossible to use on touch devices. See the Touch controls tutorial for an alternative control system.

If you only use left clicks, consider instead using the Touch object with *Use Mouse Input* enabled. This will allow your project to work on touchscreen devices without any further changes.

### Mouse hardware buttons

Most modern mouse hardware has three buttons: left, middle and right. The middle button usually is also a scroll wheel that allows more conveniently scrolling up and down (and can be detected in Construct with the *On mouse wheel* trigger). However some specialist devices have an additional two buttons, named simply button 4 and button 5, which are usually positioned on the side of the device (but some hardware may differ). You can also detect these buttons with the Mouse object, but note that not all users will have those buttons on their mouse hardware. You could use the additional buttons to provide shortcuts which can be achieved another way, such as with key presses, but you should avoid using them for unique features that cannot be accessed any other way.

### Scripting

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

## Mouse conditions

**Cursor is over object**

True if the mouse cursor is hovering over an object.

**Mouse button is down**

True if a given mouse button is currently being held down.

**On any click**

Triggered when any mouse button is clicked. This can be useful for title screens or cut-scenes where any input will continue to the next step.

### On button released

Triggered when a given mouse button is released.

### On click

Triggered when a given mouse button is pressed. This can also be used to detect double-clicks.

> *Double-clicks only support the primary (left) mouse button.*

### On mouse wheel

Triggered when the mouse wheel is scrolled up or down a notch. Choose *any* to detect the mouse wheel being scrolled in either direction; in this case the WheelDeltaX/Y/Z expressions are useful to determine the direction and magnitude of the scroll.

> *Some non-mouse devices, like laptop trackpads, can also fire wheel events. These inputs can have different behavior to mouse wheels, such as momentum-based scrolling with slowly reducing delta values.*

### On object clicked

Triggered when a given mouse button is pressed while the mouse cursor is over an object. This can also be used to detect double-clicks on objects.

### Has pointer lock

True if pointer lock is currently active, i.e. after *On pointer locked* triggered.

### On movement

Triggered every time the mouse moves, or any touch input moves. The *MovementX* and *MovementY* expressions provide the amount of movement. This can be used without pointer lock, but mouse movement will still be limited by the boundaries of the window or screen. Since this trigger also works with touch input, it means movement input also works with touch input on mobile devices. Usage with touch input does not require pointer lock, since that only affects the mouse cursor.

> *When using On movement for 3D camera mouse look, the support for touch input also means that the view can be rotated by touch input on mobile.*

### On pointer locked

Triggered when the pointer is successfully locked after the *Request pointer lock* action. The mouse cursor will disappear and only mouse movement values will be available via the *On movement* trigger.

---

**On pointer unlocked**

Triggered when pointer lock is released, restoring the normal mouse cursor behavior. This can happen either by the *Release pointer lock* action, or at any time if the user manually exits pointer lock, such as by pressing `Escape`.

---

**On pointer lock error**

Triggered if an error occurs attempting to use pointer lock, such as attempting to acquire pointer lock when security restrictions disallow it.

## Mouse actions

---

**Set cursor from sprite**

Set the cursor image from a Sprite object. This is preferable to setting a sprite to the mouse co-ordinates, because the input lag is significantly lower. Various limitations apply: the sprite image is used as it appears in the image editor, not taking in to account size or rotation in the layout; the image cannot be too large (64x64 is usually the limit); the cursor may not be applied close to the edges of the browser window; and support varies depending on browser and OS.

---

**Set cursor style**

Set the type of mouse cursor showing. The cursor can be hidden completely by choosing *None*.

---

**Request pointer lock**

Request to acquire pointer lock, which hides the mouse cursor and only reports mouse movement values via the *On movement* trigger and *MovementX* and *MovementY* expressions. This allows continuous input without the boundaries of the window or screen stopping movement. This type of input is useful for "mouse look" with the 3D Camera object (see First-person platformer for an example). Normally this can only be used in a user input trigger, such as *On click*. *On pointer locked* is triggered if the request is successful, but the request may be denied, for example due to security restrictions, in which case *On pointer lock error* will trigger. Also note the user can also exit pointer lock at any time, triggering *On pointer unlocked*. The *Unadjusted movement* parameter can be checked to disable system adjustments to the mouse movement, such as acceleration, instead accessing the raw mouse input.

---

**Release pointer lock**

If pointer lock is active, this ends pointer lock and restores the normal mouse cursor behavior.

## Mouse expressions

---

**AbsoluteX**

**AbsoluteY**

Return the position of the mouse cursor over the canvas area in the page. This is (0, 0) at the top left of the canvas and goes up to the window size. It is not affected by any scrolling or scaling in the project.

---

**X**

**Y**

Return the position of the mouse cursor in layout co-ordinates. This is (0, 0) at the top left of the layout. It changes to reflect scrolling and scaling. However, if an individual layer has been scrolled, scaled or rotated, these expressions do not take that in to account - for that case, use the layer versions below.

---

**X(layer)**

**Y(layer)**

Return the position of the mouse cursor in layer co-ordinates, with scrolling, scaling and rotation taken in to account for the given layer. The layer can be identified either by a string of its name or its zero-based index, e.g. `Mouse.X("HUD")`.

---

**WheelDeltaX**

**WheelDeltaY**

**WheelDeltaZ**

In *On mouse wheel*, these values provide the direction and magnitude of the wheel movement. Typically this will alter *WheelDeltaY* representing vertical movement, with the value being positive or negative depending on whether it is up or down movement. The magnitude can also vary depending on the speed of the movement, the system settings, and the type of device. The other values support other kind of hardware devices, such as an additional horizontal scrolling wheel on some mouse devices, or pan scrolling on a laptop trackpad.

---

**MovementX**

**MovementY**

In the trigger *On movement*, these values provide the amount of horizontal and vertical movement.

# MULTIPLAYER

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

The **Multiplayer object** provides features to develop real-time online multiplayer games. It uses WebRTC DataChannels, a peer-to-peer networking feature of modern browsers, to transmit gameplay data.

### Scripting

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

## Multiplayer features

The Multiplayer object supports a number of features to enable low-latency gameplay over the Internet:

- UDP-based transmission for minimal latency avoiding head-of-line blocking, with optional reliable modes
- NAT traversal to connect through common router/network setups
- Compensation for poor quality connections with high latency, packet delay variation (PDV), and packet loss
- Interpolation and extrapolation modes to ensure smooth in-game motion
- Scirra-hosted signalling server to connect peers to each other
- Scirra-hosted TURN server to improve connectivity across some types of networks
- Authoritative host model to prevent cheating, with local input prediction to hide input latency
- LAN game support for near-zero latency gameplay, including support for mixed LAN/Internet games
- Automatic data compression to save bandwidth
- Automatic bandwidth reduction when objects are not changing, avoiding redundant repeated transmissions of identical data
- Binary transmission with control over specific datatypes to minimise bandwidth
- Large binary transfers, such as to send an entire file between peers
- Latency and packet loss simulation for realistic local testing
- Support for lag compensation
- Support for both peer-to-peer games (not needing a server) and centrally-hosted games (using a server)

## Learning to make Multiplayer games

Designing Multiplayer games is challenging. It is essential to follow the introductory tutorial series to gain an understanding of how to use the Multiplayer object correctly. You are likely to struggle if you try to skip ahead and start making a game without fully understanding how various aspects of multiplayer games work. Mistakes can also result in degraded gameplay quality, with unnecessary lag.

The four introductory tutorials are:

1. Multiplayer tutorial 1: concepts
2. Multiplayer tutorial 2: chat room
3. Multiplayer tutorial 3: pong
4. Multiplayer tutorial 4: real-time game

You can also find examples of the Multiplayer features by searching for *Multiplayer* in the Start Page.

## Signalling and hosting

The signalling server is a central server where players go to find each other. Scirra host an official signalling server at **wss://multiplayer.construct.net**. The signalling server does not transmit any gameplay data; it serves only to connect peers to the game host by relaying connection information like IP addresses. Players must connect and log in to the signalling server before they can join any rooms.

The first player in to a room becomes the host. The host acts as the server for the game, transferring actual gameplay data. Any player can be the host. This means games can run without needing any server hosting, saving you from having to pay bandwidth bills to run your multiplayer game.

If you have a large or particularly latency-sensitive game you can still run your game with a dedicated server host to take advantage of its better quality connection. This can be achieved by starting a browser on the server, starting the game and make sure the server is the first to join the room so it becomes the host. Now the server connection will be hosting the game. You can host multiple games on a server by opening multiple browser tabs (a hosted game can continue to work in a background tab). Note this means your server is genuinely running the game - as mentioned, the signalling server only helps peers connect to the room host, and actual gameplay data will be transmitted through your server if it is the room host. It is not necessary to run your own signalling server to achieve this.

## Peer IDs

The signalling server assigns every player who connects a *Peer ID*. This is a short string of random characters that uniquely identifies them, such as "ABCD". When designing multiplayer

games, it is best to identify peers by their peer ID instead of their alias (display name), since their alias could potentially change but their peer ID never changes so long as they remain connected.

## Sending messages

Peers only connect to the host. In other words, the host has a connection to every peer, and peers only have one connection to the host. In order for two peers to communicate, the information must be relayed via the host.

Consequently, the host can directly message any peer in the room, but peers can only directly message the host. See the chat tutorial for an example of how to relay messages through the host so other peers can receive messages from another peer.

The host also has the unique ability to *broadcast* a message. This sends the same message to every peer in the room.

Messages can be sent with different reliability modes. These are:

- **Reliable ordered**: all messages are guaranteed to arrive, and the order received will be the same as the order sent. However this can have the highest latency since a lost message will need retransmitting and will hold up every message sent after it. This mode is suitable for chat messages.
- **Reliable unordered:** all messages are guaranteed to arrive, but the order received can be different from the order sent. This allows improved latency since a lost message will be held up as it is retransmitted, but subsequent messages can still get through quickly without being held up. This mode is suitable for gameplay events that occur independently of each other, such as "door opened" or "explosion occurred".
- **Unreliable:** messages are not guaranteed to arrive, and the order received can be different from the order sent. The network will make a "best effort" attempt to deliver the message, but the message may be dropped and the recipient will never receive it and will have no indication it was even attempted to be sent. This mode is suitable for high-bandwidth or regularly-sent messages, where if a message is dropped it is likely to be shortly followed up by another one. Note the Multiplayer object uses this mode internally for object positions and data when using *Sync object*; avoid re-implementing this functionality with this mode.

## Updating games

The host is authoritative for both gameplay data and its settings. If you change any aspect of your game, such as the synced objects or variables, the client input values, or bandwidth profile, the host's values are authoritative when there is a mismatch between the peer and the host. In order to avoid confusion or broken games, use a different game instance name when distributing an update, so only peers using the same version end up connecting to each other.

## Binary transfers

The Multiplayer object supports sending binary messages by transmitting the contents of a Binary Data object. However due to the underlying networking technologies, a single binary

message cannot be too large (common limits are in the range 64 KB - 256 KB), which poses a problem if you want to send a larger file to another peer.

The Multiplayer object provides a **Binary transfer** feature to help send larger amounts of data. See the Multiplayer file transfer example for a demonstration of how it works. When making a binary transfer, the process works like this:

1. The sending peer starts a binary transfer. The *receive started* event fires for the receiving peer, where the receiver can access metadata like the filename and transfer size.
2. The Multiplayer object internally splits the data up into smaller chunks that can each be safely transmitted. The sending peer then starts sending these chunks over the network. Both sides will fire progress triggers to be able to observe how the transfer is progressing.
3. Once all chunks have been successfully transmitted, the transfer completes and the receiving peer as an identical copy of the binary data that was transmitted.

Binary transfers can have an optional tag assigned to identify different transfers. For convenience each transfer can also specify a filename and MIME type that the receiver can also access for purposes like transferring a named file. Transfers can also be cancelled while still in progress, such as if a user aborts a transfer. Transfers also benefit from automatic data compression to save bandwidth. Binary data transfers don't allow specifying a reliability mode - they always implicitly send with *reliable ordered* mode as the feature is designed to guarantee transmission of an identical copy of the data.

As noted in the section above, peers only connect to the host and so direct communication is only possible between the host and peers. However the binary transfer feature supports an automatic relay through the host. This means two non-host peers can transfer binary data to each other. In this case each chunk will be sent first to the host, and then the host will send it on to the destination peer. This saves you having to implement a relay yourself. However it is important to note the privacy and security implications of this. While data transmitted over the network is automatically encrypted, the host will see unencrypted binary transfers while relaying them. Bear this in mind if transferring any privacy or security sensitive data between non-host peers. Also note this will increase the bandwidth used by the host, especially if multiple binary transfers occur simultaneously.

## Multiplayer conditions

**On (binary transfer) complete**

**On any (binary transfer) complete**

Triggered on the receiving end of a binary transfer when the transfer has completed. The full transferred data is then available in the chosen Binary Data object. The *On complete* trigger fires only for the specified transfer tag, whereas the *any* variant trigger fires for any transfer (with the tag available in the *Tag* expression).

**On (binary transfer) event**

### On any (binary transfer) event

These triggers fire at various stages of a binary transfer, either for a transfer with a specific tag, or for any transfer (with the tag available in the *Tag* expression). The types of events are:

- *receive started*: triggered for the receiving peer when a binary transfer starts. Metadata is available through expressions like *Filename* and *TransferSize*.
- *progress*: triggered for both sending and receiving peers as the transfer progresses, with the progess available in the *TransferProgress* expression.
- *cancelled*: triggered for both sending and receiving peers when a transfer is explicitly cancelled by the sender with the *Cancel binary transfer* action.
- *send complete*: triggered for the sending peer when the transfer has finished being sent (as the *On complete* trigger is only fired by the receiver).

---

### Compare peer count

Compare the number of peers currently in the room, if a room has been joined. The peer count includes the host so is at least 1 if in a room.

---

### Is host

True if in a room and acting as the host. The host of the room is effectively the server for the game. Peers only connect to the host, and the host must relay data if two other peers are to communicate.

---

### On any peer message

Triggered when a message with any tag is received. The *Message*, *Tag*, *FromID* and *FromAlias* expressions can be used to retrieve information about the received message. The order messages are received, or whether a sent message is received at all, depends on the reliability mode chosen when the message was originally sent.

---

### On kicked

Triggered if kicked from the current room. This can occur if the host quits, the connection to the host could not be established, or the host otherwise decides to forcibly remove you from the room. After *On kicked* the player is no longer in the room and must re-join a room to be able to participate in a game.

---

### On peer connected

Triggered when another peer joins the same room. It also triggers once per peer already in the room when joining an existing room, including the host. The *PeerID* and *PeerAlias* expressions identify the relevant peer.

---

**On peer disconnected**

Triggered when a peer disconnects from the room. The *PeerID* and *PeerAlias* expressions identify the peer that left. The *LeaveReason* expression can indicate why the peer left, such as if they intentionally quit or timed out.

---

**On host disconnected**

Triggered when a peer is disconnected from the host. This trigger works the same as using *On peer disconnected* and checking that the disconnected peer ID is the host's ID. When the host disconnects, it ends all communication as the host was the peer's only point of contact, and typically represents the end of the session.

---

**On peer message**

Triggered when a message sent with a specific tag is received. The *Message*, *FromID* and *FromAlias* expressions can be used to retrieve information about the received message. The order messages are received, or whether a sent message is received at all, depends on the reliability mode chosen when the message was originally sent.

---

**On peer binary message**

As with *On peer message*, but triggered when receiving a binary message sent with the *Send binary message* action. The received data is stored in the chosen Binary Data object. Note that binary messages do not have tags, but this data can be included in the transmitted binary data.

---

**Is ready for input**

True when a peer is ready to send input to the host. This means *On client update* has triggered at least once, or is about to trigger. Do not allow players using input prediction to move or act before this condition is true or *On client update* has triggered: doing so will simply cause an input prediction error since the host is not yet ready to receive input.

---

**On client update**

Triggered when a peer is about to send its input state to the host. The input state should be updated in this trigger using the *Set client state* action.

---

**Is connected**

True if currently connected to the signalling server. It is not necessary to be connected to a signalling server once connected to the room host.

---

**Is in room**

True if currently in a room on the signalling server.

---

**Is logged in**

True if currently connected to the signalling server and successfully logged in.

---

**On connected**

Triggered after successfully connecting to the signalling server. In order to join rooms, it is necessary to next log in to the server.

---

**On disconnected**

Triggered after disconnecting from the signalling server.

---

**On error**

Triggered if an error occurs with the signalling server. The *ErrorMessage* expression indicates the type of error that occurred.

---

**On game instance list**

Triggered after *Request game instance list* when the list has been received from the signalling server. The *List...* expressions can be used to retrieve the list details.

---

**On joined room**

Triggered after the *Join room* or *Auto-join room* actions when the room has been successfully joined. The *Is host* condition can be used to determine if the player is the first joining peer and has been assigned the room host.

---

**On left room**

Triggered after the *Leave room* action when the room has been left. The room is also left if *On kicked* triggers.

---

**On logged in**

Triggered after the *Log in* action if the login is successful. Once logged in it is possible to join rooms. Note the signalling server may have assigned a different alias to the one requested if it was already taken; use the *MyAlias* expression to determine the actual alias in use.

### On room list

Triggered after the *Request room list* action when the room list has been successfully received. The room list expressions can then be used to inspect the received list.

## Multiplayer actions

### Transfer peer binary

Begin a binary data transfer to a peer identified by their peer ID. The data to send is in a specified Binary Data object. An optional tag, filename and MIME type may be specified which the receiver can also use. For more details see the section on *Binary transfers* above.

### Cancel binary transfer

Cancel a currently in progress binary transfer specified by the peer ID it is being sent to and the tag of the transfer. Both the sender and receiver will fire the *cancelled* event and no further data will be transmitted.

### Add client input value

Use on startup to add a value that peers send to the host to indicate their input state. Each client input value has a tag to identify it; use this tag to update the value with the *Set client state* action. To avoid wasting bandwidth, use the lowest *Precision* that can still hold all the values that need to be set. If using *setbit* / *getbit* to send key states, you must use *None* for *Interpolation*; otherwise use *Linear* for values like positions, or *Angular* if representing an angle.

### Associate object with peer

If *Sync object* is used on objects which represent peers in the game, use this action to associate an instance of an object with a given peer. Typically this is used in the object's *On created* trigger. Both the host and peers must associate objects with peers. Associated objects are automatically destroyed when the corresponding peer leaves.

### Disconnect

Disconnect from the room. If the room host, all players are kicked; otherwise the peer disconnects from the host. The room is also left on the signalling server, so another room can be joined afterwards.

### Send message

Send a message to a specific peer with a given reliability mode. Peers can only message the host (and the *Peer ID* field must be left empty), but the host can send a message to any peer. Message tags can be used to identify messages for different purposes, such as "chat" or "gameplay-event". The message must be a text string, but could also be JSON data such as

from an Array or Dictionary *AsJSON* expression; however be sure to avoid wasting bandwidth. The order messages arrive, or whether it is guaranteed to arrive at all, depends on the reliability mode.

### Send binary message

As with *Send message*, but the data to be sent is specified by a Binary Data object. Binary messages do not use a tag, although this data can be incorporated in the transmitted data.

> *The underlying networking technologies usually impose a limit on the maximum size of binary data that can be transmitted in a single message. Commonly this is in the range 64 KB - 256 KB (although this limit applies after automatic data compression, so sometimes larger messages may be allowed if they compress to within the allowed size). To send larger amounts of data such as for file transfers, use the binary transfer feature instead.*

### Set bandwidth profile

Switch between *Internet* or *LAN* (Local Area Network) bandwidth modes. The bandwidth profile must be set before joining a room, and only the host's setting is used for all peers in the room. The default mode is *Internet*, which sends updates 30 times a second with an 80ms buffer. *LAN* mode sends updates 60 times a second with a 40ms buffer. *LAN* mode will use about double the bandwidth of *Internet* mode and will degrade gameplay quality more if there is latency or PDV in the connection. *LAN* mode should never be used for Internet games - it is intended for networks where bandwidth is effectively unlimited and latency effectively zero, which is typically only the case with local area networks, and taking advantage of this can improve gameplay quality. *Internet* mode should however work well over LANs, so if in doubt leave it on that.

### Simulate latency

Simulate latency, PDV and packet loss on all inbound and outbound messages. This can be useful for making local testing more realistic, since unlike the Internet latency is effectively non-existent. For local testing it is only necessary to simulate latency on the host, since that guarantees every message in the game will have delay added; it is not necessary to also simulate latency on the peers. The latency for an individual message is calculated as the latency plus a random value from zero to the PDV. The packet loss indicates the chance an unreliable message is lost entirely, or in the case of reliable messages that retransmission is necessary and the latency is multiplied.

### Sync object

Automatically sync an object. The host sends information about synced objects to peers. This is one-way transmission; peers sync with what is happening on the host. As synced objects are created, moved and destroyed on the host, they are correspondingly created, moved and destroyed on all connected peers. It is important to disable any behaviors and

deactivate any events on the peers that may attempt to move the objects themselves; this will conflict with what *Sync object* is trying to do, and will not have any effect on the host. Peers should use their client input values as their sole way of influencing the game.
Synced objects can optionally include their position and/or angle with a given precision. If no position or angle is selected, then it simply ensures the same numbers of objects are created.

> *Syncing without a position, or with only one axis, causes objects to be created at a co-ordinate of -1000, as there is not full information about the synced object's position being sent. In that case its up to your project to correctly position the object.*

*Bandwidth* can be used to reduce the number of updates it is necessary to send for a synced objects.

- *Normal bandwidth (unpredictable)* will send updates for the object at most every update (30 times a second in Internet mode) and is suitable for objects with unpredictable movement.

- *Low bandwidth (highly predictable)* will send updates at most 10 times a second, which should only be used for highly predictable motion such as moving in a straight line at the same speed (it is not enough to handle changes in motion smoothly).

- *Very low bandwidth (essentially static)* will send updates at most twice a second, which should only be used for objects which are not expected to move but nevertheless can occasionally be created or destroyed, such as scenery.

Note that even in *Normal bandwidth* mode, objects which are not changing gradually reduce their bandwidth to twice a second anyway, so static objects will still end up using *Very low bandwidth* mode. Therefore it is not normally necessary to change this, and it is suitable to use *Normal bandwidth* even for objects which rarely change.

> *Note Sync object does not support objects in containers. You should make sure any synced objects are not in a container, sync each object separately, and if necessary associate them through events.*

---

### Sync instance variable

Add an instance variable to sync with an object. The host sends the values of the instance variable for each object to the peers, keeping them up to date. The chosen object must already first be synced with *Sync object*.
The *Precision* corresponds to the precision for *Sync object*, with an additional *Very low (uint8, 1 byte)* option, useful for bitwise flags.
*Interpolation* can be *None* (updates in steps), *linear* (linearly smoothed interpolation between values, suitable for positions), or *angular* (rotational interpolation between angles).
The *Client value tag* should be the name of the corresponding client input value, if any, to help

ensure the host can sync the instance variable with minimal latency.
Note: text instance variables cannot be synced, only numbers. To share text data between peers, use messages instead.

---

**Broadcast message**

As with *Send message*, but can only be used by the host. This sends a message to every peer in the room. *From ID* can be used to indicate the message is being sent on behalf of another peer; if it is used, when peers receive the message the *FromID* and *FromAlias* will be set to this peer. Also the message will *not* be sent to the specified *From ID* peer, since usually this is redundant. If it is empty, it will be sent to all peers and received as from the host.

---

**Kick peer**

When host, forcibly disconnect a peer from the room so they are no longer participating. The kicked peer will be notified that they have been disconnected and optionally the kick reason can be displayed. Peers cannot kick anyone, only the host can.

---

**Enable local input prediction**

Enable local input prediction on an object representing the local player. The object must be associated with the local peer's ID, and the game logic must correctly attempt to move both the local player and the player on the host in exactly the same way with correct use of client input values. This allows local controls to have immediate effect, but apply correction if the host position starts to deviate from the local position. See the fourth multiplayer tutorial for more information. There are two additional options for customizing how local input prediction is handled:

- **Avoid solids:** if enabled, input prediction will avoid moving the object into any solid. If input prediction would have moved the object so that it overlaps a solid, it instead leaves the object in its current position.
- **Platform mode:** if enabled with an object using the Platform behavior, this detects if the Platform behavior is currently standing on the floor, and when it is then input prediction on the Y axis is disabled. This can help avoid a jittery movement when landing on the floor.

The precise settings to use depend on the type of game and movement in use - some experimentation may be required to determine the best input prediction settings.

---

**Set client state**

In *On client update*, set the value of a client input value by its tag. The value must be a number, and *Add client input value* must have been used on startup to add a value with the given tag in advance.

---

**Add ICE server**

Add a custom Interactive Connectivity Establishment (ICE) server used by WebRTC to establish connections between peers. There are a couple of built-in public STUN servers used, but you can also provide your own TURN servers to enable connectivity through certain kinds of NAT. A username and credential can also be optionally provided if the server requires them. This action should be used on startup, before any connections are made.

---

**Auto-join room**

Join the first available room with the given game, instance and first room name. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. When rooms are full, the signalling server will create a new room. For example if "myroom" is full, it will try "myroom2", "myroom3", etc. This effectively arranges all joining peers in to games of a particular size. If the room is locked when full, then late-joiners are not allowed; if left unlocked and a peer leaves after the game starts, a newly joining peer may be added back to the game to top it up to the *Max peers* again. Upon joining, *On joined room* triggers.

---

**Connect**

Connect to a signalling server. The official Scirra signalling server is at **wss://multiplayer.construct.net**. Upon successful connection, *On connected* will trigger.

---

**Disconnect**

Disconnect from the signalling server. This can be done once peer-to-peer connections are established if the signalling server is no longer necessary, but note that will prevent any new peers from joining the game late.

---

**Join room**

Join a specific room in the given game instance. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. *Max peers* can be used to limit the number of peers that join. Only the host's value is used. If the room is full, subsequently joining peers will receive a "room full" error. The peer count includes the host, so 2 is the minimum value, or it can be left as 0 to allow an unlimited number of peers to join. Upon successfully joining, *On joined room* triggers.

---

**Leave room**

If in a room, leaves the room on the signalling server. *On left room* triggers upon the server acknowledging the request to leave. Note the room has not really been left until that trigger runs.

---

**Log in**

Once connected, log in to the signalling server. Players must log in before they can join rooms. The *Alias* is the requested display name to use. Note that if the requested alias is already taken, the server will automatically assign an alternative; be sure to use the *MyAlias* expression after logging in to determine the actual alias in use. Upon a successful login, *On logged in* triggers.

---

**Request game instance list**

Request a list of active game instances within the given game. When the response is received *On game instance list* triggers and the name and total number of peers in the returned instances can be listed using the *List...* expressions.

---

**Request room list**

Request a list of active rooms within a given game instance. The returned list can include all rooms, only rooms which are unlocked, or only rooms which are available to join (unlocked and not full). When the requested list is received, *On room list* triggers.

## Multiplayer expressions

---

**Filename**

**MIMEType**

In a binary transfer, the filename and MIME type specified by the sender. These expressions are available on the receiving end in both the *receive started* and *completed* triggers.

---

**TransferProgress**

During a binary transfer and in a *progress* event (fired for both senders and receivers), this returns the current transfer progress on a scale of 0 to 1 (e.g. 0.5 being 50% complete).

---

**TransferSize**

In a binary transfer, the size in bytes of the data to be transmitted. This is set on the receiving end in the *receive started* event so the receiver knows ahead of time how big the data will be. (Once the transfer has completed, the size can also be obtained from the chosen Binary Data object.)

---

**ListInstanceCount**

After *On game instance list* triggers, the number of game instances in the received list.

---

**ListInstanceName(index)**

**ListInstancePeerCount(index)**

Get the name and peer count of a given game instance in the returned instance list.

**ListRoomCount**

After *On room list*, the number of rooms in the received list.

**ListRoomName(index)**

**ListRoomPeerCount(index)**

**ListRoomMaxPeerCount(index)**

**ListRoomState(index)**

After *On room list*, retrieve information for a room at an index in the received list. The state can be one of "available", "locked" or "full".

**FromAlias**

**FromID**

The alias and ID of the peer a message is from in *On message received* or *On any message received*.

**HostAlias**

**HostID**

When in a room, the alias and ID of the host of the room.

**LeaveReason**

A string identifying a reason for leaving in *On peer disconnected*, if known, e.g. "quit", "timeout", "network error"...

**Message**

The contents of the received message in *On message received* or *On any message received*.

**PeerAlias**

**PeerID**

The alias and ID of the relevant peer in a trigger like *On peer connected* or *On peer disconnected*.

**PeerCount**

The number of peers in the current room, including the host.

**PeerAliasAt(index)**

**PeerIDAt(index)**

The alias and ID of the nth peer in the current room, up to *PeerCount*.

---

**PeerAliasFromID(peerid)**

Get the alias of a peer in the current room from their peer ID.

---

**PeerIDFromAlias(alias)**

Get the ID of a peer in the current room by their alias.

---

**PeerLatency(peerid)**

**PeerPDV(peerid)**

Get the latency and packet delay variation (PDV) of a peer from their peer ID. Peers can only use this to get the stats for the host, since that is the only connection they have, but the host can use it for any peer.

---

**Tag**

The tag of the received message in *On any message received*. This expression is also used by binary transfers, returning the tag of the current binary transfer in any of the binary transfer triggers.

---

**LagCompensateAngle(movingPeerID, fromPeerID)**

**LagCompensateX(movingPeerID, fromPeerID)**

**LagCompensateY(movingPeerID, fromPeerID)**

Return the lag-compensated position and angle for *movingPeerID* as seen by *fromPeerID*. In other words, this returns the past position of *movingPeerID* going back by the amount of time that *fromPeerID* is delayed by, given their latency. For example this can be used to perform a lag-compensated hit-test when *fromPeerID* shoots a laser. This is covered in more detail in the fourth multiplayer tutorial.

---

**PeerState(peerid, tag)**

When host, retrieve the latest client state value with the given tag, for a given peer ID. The peer will have set this with the *Set client state* action to indicate their input state.

---

**CurrentGame**

**CurrentInstance**

**CurrentRoom**

Retrieve the current game, instance and room names, if joined on the signalling server.

---

**ErrorMessage**

**ErrorMessage**

In *On signalling error*, the error message if available.

---

**MyAlias**

**MyID**

The current player's own alias and ID, once connected and logged in to the signalling server.

---

**SignallingMOTD**

**SignallingName**

**SignallingOperator**

**SignallingURL**

**SignallingVersion**

Once connected to the signalling server, retrieve the Message Of The Day (MOTD), server name, server operator, website URL and server version for the connected server.

---

**ClientXError**

**ClientYError**

The input prediction error for peers, used for debugging.

---

**HostX**

**HostY**

The position the host has for the current peer, used for debugging.

---

**StatInboundBandwidth**

**StatOutboundBandwidth**

Return the total estimated inbound and outbound bandwidth transmitted over the network, in bytes per second. When automatic data compression is in use, this measures the compressed size of the data actually sent over the network.

---

**StatInboundDecompressedBandwidth**

**StatOutboundDecompressedBandwidth**

Return the total estimated decompressed inbound and outbound bandwidth for all data transmission through the Multiplayer object, in bytes per second. When automatic data compression is in use, this measures the size of the decompressed messages, which may be significantly larger than the data actually sent over the network, measured by the *StatInboundBandwidth* and *StatOutboundBandwidth* expressions. Together these expressions can also be used to identify the compression ratio (i.e. how much bandwidth is saved by compression).

**StatInboundCount**

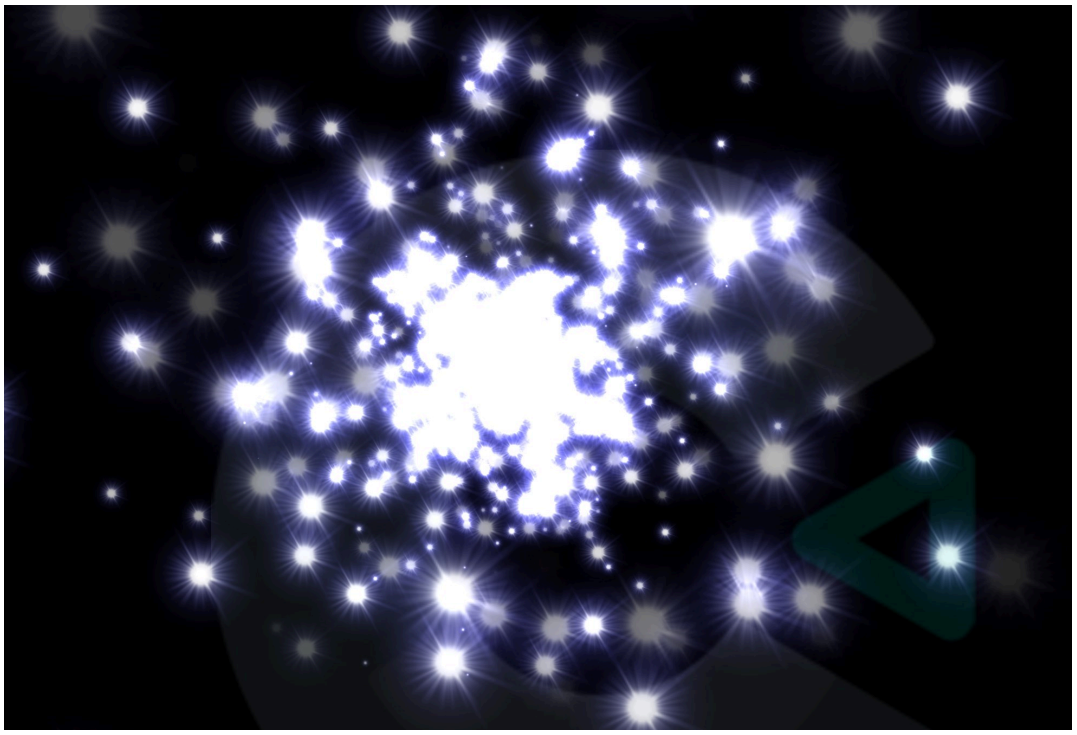
**StatOutboundCount**

Return the total number of separate inbound and outbound messages sent and received by the Multiplayer object. This includes internally-used messages for things like ping and synchronisation; generally the bandwidth is the more practically useful statistic.

# PARTICLES

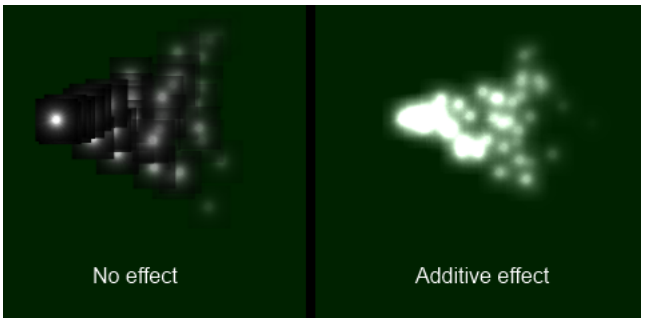
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/particles

The **Particles** object can quickly create visual effects by creating and moving many small images independently. It is a versatile object capable of many different kinds of visual effects. There are several examples in the Example Browser, ranging from fire to fountains; search for *Particles* to find them. The image below shows an example of one of the particle effects possible with the object.



The Particles object has many parameters to change the behavior of each particle. Also, it requires a texture used to draw each particle. Often a simple white spot on a black background is sufficient.

The **Additive** blend mode works especially well with the Particles object. It makes each particle brighten the background rather than pasting its image over the background, and allows particles to blend in to each other as well rather than simply overlapping. This makes particles look more like light sources. The below image shows what the effect does when the texture is a white spot on a black background.




*Using Additive blend on particles*

Colored effects can be created using colored particle images. Note that since the Additive effect brightens the background towards white, any objects using an Additive effect will not show up on a white background. The effect works best on dark backgrounds.

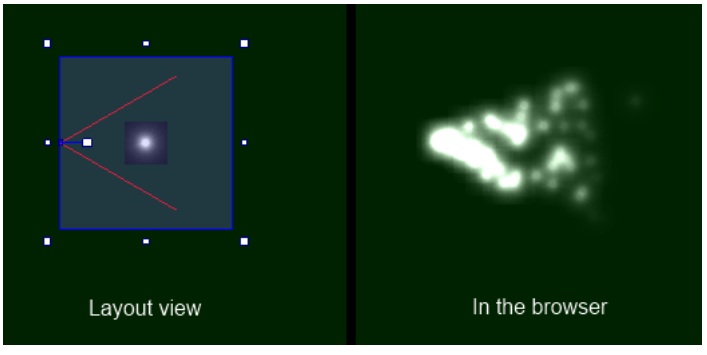
For more information about blend modes and effects, see the manual section on Effects.

### Scripting

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

## Particles in the Layout View

The Particles object is represented in the Layout View by two red lines which represent the spray cone (the angle through which particles are fired), with the particle texture in the middle. The Particle object's origin is where particles are created from. An example is shown below on the left, with the effect at runtime on the right.




The size of the particles object in the layout view is not important. The object will automatically size itself at runtime to fit all the particles it has created.

### Previewing particle effects

Paid plans only Enable the **Preview** property to run the particle effect directly in the Layout View. As you edit the particle object's properties, the preview will update in real-time. This is a much faster way to get the exact effect you want compared to having to preview each time.

## How particle effects work

The particle effect works similarly to using the Bullet behavior on each particle. Initially particles are fired forwards at a given speed and at an angle within the spray cone. Each particle is then individually controlled with different alterations to its speed, angle, opacity and size during its lifetime. The fact particles move independently of the others is often what makes the visual effect interesting. The various properties of the Particles object control exactly how the particles change over time and what random alterations are made. It is worth spending some time changing parameters to see the effect they have for yourself.

There are three different settings for when particles are destroyed, set by the **Destroy mode** property. The default *Fade to invisible* will fade each particle's opacity to zero over the *Timeout*, destroying the particle when it becomes invisible. *Timeout expired* will simply destroy the particle after an amount of time, without changing its opacity. *Particle stopped* will destroy the particle when its speed reaches zero, but you must take care to ensure particles slow down with a negative acceleration otherwise they will never be destroyed!

## Advanced particle effects

By default, the Particles object draws particles using its own image. However you can specify an object instead by choosing another object, such as a Sprite, in the *Object* property. In this case the Particles object will then spawn and move instances of that object instead of drawing its own particles. This provides a similar visual result, but allows for much more flexibility. For example you can rotate the objects that are spawned, use behaviors and effects on them, or even test them for collisions.

> *Note that using objects for particles is slower than letting the Particles object draw its own particles.*

## Optimisation

When the Particles object is drawing its own particles (i.e. not using an object), it is more efficient than creating the same effect with objects, but not by a large margin. Just like with sprites, you should be aware that creating a large number of particles can have a serious performance impact on your project. Use the **ParticleCount** expression to monitor how many particles are being created. Creating more than a few hundred particles may start to impact the framerate.

To reduce particle counts, try reducing the rate or shortening the timeout. To compensate, you can try making the particle size larger so the effect does not get thinner.

When using objects for particles, note that this is slower, and the performance can get worse if every object is different (e.g. using effects with different parameters) or mixed up in the Z order (e.g. other objects appear in between particles in the Z order). For best performance keep all the particle objects as similar as possible and ensure they aren't Z ordered individually.

# Particle Properties

The Particles object has a relatively many properties, which are split in to three groups: particle spray properties (relating to the Particles object itself), initial particle properties (relating to the creation of each individual particle) and particle lifetime properties (relating to how particles behave after creation).

## General

### Rate

The number of particles created per second. If *Type* is *One-shot*, this is the total number of particles fired. Note that in *Continuous spray* mode, the overall particle count may be significantly more than the rate depending on the other properties. Also note that in *One-shot* mode, the rate can only be changed immediately after the object has been created; after the first tick, using the *Set rate* action will have no effect.

### Spray cone

The number of degrees through which particles are fired. This is represented by the red lines in the Layout View. Use 360 to fire particles in all directions.

### Type

The Particles object can work in two modes:

- **Continuous spray** will create a constant spray of particles (the default).
- **One-shot** will create a single blast of particles, the total number set by *Rate*. Once all particles have been destroyed, the Particles object then destroys itself. This is useful for one-off effects like explosions or impacts.

### Image

Click to open the Animations editor to edit the particle image. Try a spot on a transparent background, or on a black background with the Additive effect. Note the image is not used if an object is set instead.

### Object

Create instances of an object for each particle instead of drawing the particle image. For more information see the section *Advanced particle effects* above. Note this mode is slower than using a particle image.

> *To unselect an already chosen object, open the object picker, click Clear selection, and*

*then click OK.*

**Initially visible**

Set whether the object is shown (visible) or hidden (invisible) when the layout starts.

**Preview Paid plans only**

Enable to run a preview of the particle effect directly in the Layout View. You can change the other particle properties and see their effect in real-time.

## Initial particle properties

**Speed**

The initial speed each particle is fired at, in pixels per second.

**Size**

The initial size of each particle, in pixels. Particles are always shown as squares, but the shape can be customised with the particle image.

**Opacity**

The initial opacity of each particle, from 0 (transparent) to 100 (opaque).

**Grow rate**

The initial grow rate (change in size over time) for each particle, in pixels per second. 0 means the particle will always stay the same size. A positive value will make particles grow, and a negative value will make particles shrink.

**X randomiser**

**Y randomiser**

The initial offset to the particle's position. You can make particles created along a line or in a box with these properties.

**Speed randomiser**

A random adjustment to each particle's initial speed on creation. For example, a value of 100 will change each particle's initial speed by up to 50 pixels per second faster or slower.

**Size randomiser**

A random adjustment to each particle's size on creation. For example, a value of 20 will change each particle's initial size by up to 10 pixels larger or smaller.

**Grow rate randomiser**

A random adjustment to each particle's grow rate on creation. For example, a value of 10 will change each particle's initial grow rate by up to 5 pixels per second greater or less.

## Particle lifetime properties

**Acceleration**

Change in particle speed over time, in pixels per second per second. A positive value will make particles speed up, and a negative value will make them slow down.

**Gravity**

The acceleration downwards caused by gravity, in pixels per second per second. Useful for making fountain or other falling particle effects. Set to 0 to prevent gravity having any effect on particle movement.

**Angle randomiser**

A random change to each particle's angle to apply during its lifetime. For example, set to 0 to prevent particles ever changing direction, or set to 10 to allow particles to randomly change direction a little over time.

**Speed randomiser**

A random change to each particle's speed to apply during its lifetime. For example, set to 0 to prevent the speed changing, or set to 100 to allow particles to speed up or slow down somewhat over time.

**Opacity randomiser**

A random change to each particle's opacity to apply during its lifetime. Useful for creating "twinkling" effects.

**Destroy mode**

How each particle is destroyed. There are three modes available:

- **Fade to invisible** will fade each particle's opacity to zero over the *Timeout*. When the particle becomes invisible, it is destroyed.

- **Timeout expired** simply destroys each particle after the *Timeout* has expired, without altering the opacity.

- **Particle stopped** destroys each particle when its speed reaches zero. You must take care to use a negative *Acceleration*, or particles will never be destroyed!

---

**Timeout**

The time in seconds particles last for before being destroyed, depending on the *Destroy mode*.

# Particle conditions, actions and expressions

Most of the Particle object's actions and expressions just set or get the above properties. See the above properties for a reference. The other conditions, actions and expressions not relating to the above properties are documented below.

For features in common to other objects, see Common features.

## Particle conditions

---

**Is spraying**

True if the particle spray is currently enabled.

## Particle actions

---

**Set spraying**

Enable or disable the spray, when in *Continuous spray* mode. When disabled, no new particles are created.

---

**Fast-forward**

Skip ahead the particle effect by a time in seconds. For example fast-forwarding by 3 seconds will cause the Particles object to instantly spawn, move and destroy particles as if 3 seconds had gone by. This is useful for making sure particle effects appear ready immediately, rather than taking a few seconds to move their particles out from the spawn point.

## Particle expressions

---

**ParticleCount**

The number of particles the Particles object currently has. This is important to ensure you are not creating too many particles and slowing the project down; see the *Optimisation* section above. Note that due to the way Construct expressions work, if you have multiple

Particle object instances, this will only return the particle count for one of the instances - use a *For Each* loop to count multiple instance's total particle count.

# PLATFORM INFO

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/platform-info

The **Platform info** object returns information about the system, device and browser.

### Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IPlatformInfo script interface, typically accessed via the `runtime.platformInfo` property. Note that this interface is always available in script regardless of whether the Platform Info object has been added to the project.

## Platform info conditions

**Is Cordova export**

True when running after a Cordova-based export. Cordova is used to export Android and iOS apps, and implies that the platform is mobile and the OS is either Android or iOS.

**Is Linux wrapper export**

True when running after a Linux wrapper export, and implies the platform is desktop and the OS is Linux.

**Is macOS wrapper export**

True when running after a macOS wrapper export, and implies the platform is desktop and the OS is macOS.

**Is web export**

True when running after a web-based export. This is also true in preview mode, so that any features based on these export conditions will act as if it's in a web-based export in preview mode.

**Is Windows wrapper export**

True when running after a Windows wrapper export, and implies the platform is desktop and the OS is Windows.

**On network change**

Triggered when the network connection changes, e.g. when moving from a Wifi network to a cellular data network. The network-related expressions will update in this trigger to reflect

values for the new network.

---

**Is on Android**

**Is on Chrome OS**

**Is on iOS**

**Is on Linux**

**Is on macOS**

**Is on Windows**

These conditions check which operating system (OS) the project is currently running on. For example in Chrome on Windows, *Is on Windows* will be true; in an Android app, *Is on Android* will be true; and so on. Note these conditions are not exhaustive: there may be platforms where none of these six conditions are true.

> *macOS was formerly known as OS X.*

---

**Is on mobile**

Test if the current device reports itself as a "mobile" device.

> *This condition may not work like you expect. Many devices lie about their device class - for example modern iPads tell websites they are macOS desktop devices in order to get the desktop experience. Further it is probably unwise to rely on this detection anyway, as there is no clear definition of "mobile device". For example tablets can be used with the touchscreen only and act like a mobile device, or have a keyboard and/or mouse attached and be used like a desktop. It would also be wrong to use this condition to check if the device will use touch input (see the detecting input method for a better approach). This condition just reports whatever the device indicates to web content, which may or may not be what you want.*

---

**Is wake lock active**

True if a *Request wake lock* action has successfully completed and the screen is currently being kept on.

---

**Is wake lock supported**

True if the current browser/platform supports using wake locks to keep the screen on.

---

**On wake lock acquired**

Triggered after the *Request wake lock* action if the lock was successfully acquired. The screen will be kept on until *On wake lock released* triggers.

### On wake lock error

Triggered if an error occurs while attempting to request a wake lock. The screen will not be kept on.

### On wake lock released

Triggered when an existing wake lock is released, meaning the screen may once again automatically turn off due to inactivity. This can happen after a *Release wake lock* action, or automatically in some circumstances, such as if the browser tab or app goes in to the background. To keep the screen on after this trigger, another wake lock must be requested.

## Platform info actions

### Request wake lock

Request a wake lock to keep the screen on permanently, preventing it automatically dimming or switching off due to inactivity. Note this increases power usage so will drain battery more quickly. If successful, *On wake lock acquired* triggers; otherwise *On wake lock error* triggers.

> *A wake lock might only be granted in a user input trigger, such as On touch end.*

> *The wake lock could be released at any time after being acquired. Use the On wake lock released trigger to identify if the wake lock ended.*

### Release wake lock

If a wake lock is currently active, release it, so the screen may once again dim or turn off due to inactivity.

## Platform info expressions

### DeviceMemory

Return the approximate amount of device memory (RAM) in gigabytes. For privacy reasons some platforms will round the result, so it may not match the exact amount installed on the system. Additionally this value is not available on some browsers or platforms, in which case it will return 0.

### HardwareConcurrency

Return the number of hardware threads supported by the CPU. This is normally at least the number of CPU cores. Many modern CPUs support multiple hardware threads on a single CPU (e.g. Hyper-threading), and a common case is for each CPU to support two hardware

threads, so this is often double the number of CPU cores. For privacy reasons some platforms will round the result, so it may not match the exact number of hardware threads/CPU cores available on the system. If the value is not available, it will return 0. However almost all consumer devices have at least two CPU cores.

---

**CanvasCssWidth**

**CanvasCssHeight**

Return the size of the main display canvas in CSS pixels. This does not correspond exactly to device (physical display) pixels, but is the appropriate size from a web design perspective in `px` units.

---

**CanvasDeviceWidth**

**CanvasDeviceHeight**

Return the size of the main display canvas in device (physical display) pixels. Unlike the CSS size, this size reflects the number of actual pixels used in the display.

---

**DevicePixelRatio**

Return the number of device (physical display) pixels per CSS pixel. For example a high-DPI display may have a device pixel ratio of 2, meaning there are two device pixels per CSS pixel. This value also reflects the browser zoom level, which works by adjusting the device pixel ratio. The concept of the device pixel ratio also allows web pages using `px` units to appear the same size on higher density displays where physical pixels are much smaller.

---

**Renderer**

Return a string indicating the graphics rendering technology in use. This can be `"webgl1"`, `"webgl2"` or `"webgpu"`. It can also have a `-software` suffix if a "major performance caveat" is detected, which normally indicates slower software rendering, typically due to unreliable GPU drivers. For example, software-rendered WebGL 1 support would return `"webgl1-software"`.

---

**RendererDetail**

Return a string describing system-specific detail about the graphics hardware in use. This typically describes the GPU manufacturer and model name, and sometimes some hardware capabilities.

---

**SafeAreaInsetLeft**

**SafeAreaInsetTop**

**SafeAreaInsetRight**

**SafeAreaInsetBottom**

The inset around the edges of the screen in CSS pixels of a rectangular area that is always visible (hence safe to use for displaying anything important without it risking being cut off). This only applies for devices with non-rectangular screens, notably mobile devices with a notch, or in some cases devices with rounded edges in the corners of the screen. Devices with a standard rectangular display will return 0 for these values.

---

**ScreenWidth**

**ScreenHeight**

Get the size of the current display screen. Note this often includes areas not available to applications, such as a desktop taskbar, or mobile status bar.

---

**WindowInnerWidth**

**WindowInnerHeight**

Get the inner size of the current window. This is the size of the window content area that is available to the application.

---

**WindowOuterWidth**

**WindowOuterHeight**

Get the outer size of the current window. This includes the window browser, caption, browser address bar etc. which is not generally available to the application.

---

**ConnectionEffectiveType**

Return a string rating the effective type of the connection based on the comparable cellular data connection generation, e.g. `"2g"`, `"3g"`, `"4g"`.

---

**ConnectionRTT**

Return the estimated round-trip time (latency) of the connection in milliseconds. This is the time it takes for a network message to be sent to the remote host and a reply to be received back.

---

**ConnectionType**

The type of network connection technology in use, e.g. `"cellular"`, `"wifi"` or `"ethernet"`. If the connection type cannot be detected or the platform does not support this feature, returns `"unknown"`.

---

**Downlink**

The estimated effective download bandwidth in megabits per second. Returns 0 if unable to detect.

### DownlinkMax

The maximum downlink speed in megabits per second of the underlying connection technology. This is normally the theoretical maximum the current network technology's specification allows, for example 4G cellular allows a higher maximum speed than 3G under ideal signal conditions; however the actual available bandwidth will depend on other factors like signal quality and other parts of the network. Returns 0 if unable to detect.

### FramesPerSecond

How many frames per second (FPS) the project is rendering. The most common display refresh rate is 60 Hz, so typically an efficiently designed project will render at 60 FPS. Note however if nothing is changing on-screen, then nothing is rendered, and so the FPS measurement may fall to 0 or display a lower result; this does not indicate poor performance, only that fewer frames are necessary to render. The *TicksPerSecond* expression indicates how frequently the engine is stepping, which may be different to the frames rendered per second.

### TicksPerSecond

How many ticks per second (TPS) the project is running at. Each tick processes the logic of the game. Usually a new frame is also rendered every tick, but if nothing changes then rendering a frame is skipped; further, depending on the framerate mode, stepping the engine and drawing frames may happen at different rates. Therefore the ticks per second may produce a different measurement to the frames per second. Usually the project will continually tick even if nothing is visually changing, and only stop ticking if the project is suspended, such as by being minimized or going in to the background.

# PROGRESS BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/progress-bar

The **Progress bar** object displays a bar which can be used to indicate the progress of a long-running operation or goal.



The progress bar is styled differently depending on the platform or browser, designed to match the style of the system. If a custom style is desired, it may instead be preferable to use a Tiled Background which has its width set depending on the progress.

### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Progress bar properties

**Value**

The initial progress display to show, from 0 to the maximum.

**Maximum**

The maximum progress value, at which point the progress bar is shown full indicating a completed operation.

**Tooltip**

An optional tooltip to show while hovering the mouse over the control.

**Initially visibile**

Whether the control is initially visible or invisible in the page.

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

**Class Optional**

An optional *class* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

## Progress bar conditions

See common conditions for features shared between form control objects.

**Compare progress**

Compare the currently set progress amount.

**On clicked**

Triggered when the progress bar control is clicked.

## Progress bar actions

See common actions for features shared between form control objects.

**Make indeterminate**

Set the progress bar in to an indeterminate mode, intended to indicate that it is working, but the progress is unknown. The display of this mode depends on the browser and platform. Not all browsers may support an indeterminate mode for progress bars.

**Set maximum**

Set the maximum progress value for the progress bar.

**Set progress**

Set the current progress value displayed by the progress bar, from 0 to the maximum.

**Set tooltip**

Set the tooltip that appears when the mouse hovers over the control.

## Progress bar expressions

**Maximum**

The currently set maximum progress value.

---

**Progress**

The currently set progress amount, if the bar is not in indeterminate mode.

# QR CODE

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

The **QR code** object allows displaying a QR code in your project. This is a type of two-dimensional barcode that most modern mobile phones are able to conveniently scan, such as to open a URL without having to type it in.

Note that QR codes have limits on how much data they can represent. If too much text is specified, the QR code may disappear or turn black. The nature of a QR code means it is best suited to short snippets of information such as web addresses.

See the QR code maker example project for a demonstration of using QR codes in Construct.

## QR code properties

**Text**

The text to encode within the QR code. This can also be a URL, and most devices will offer to open a browser if scanning a QR code that contains a URL.

**Correction level**

QR codes contain redundant additional information for error recovery purposes. This improves the reliability of scanning the QR code, particularly if the image is in any way obscured, distorted or damaged. The correction level allows choosing how much additional error recovery information to include. Higher levels mean more data is stored in the QR code (and so it will also appear more detailed) but increase the scanning reliability.

**Initially visible**

Whether the object is initially visible in the layout at runtime.

## QR code conditions

The QR code object does not have any of its own conditions. However see Common conditions for features shared with other objects.

## QR code actions

**Set text**

Change the text encoded within the QR code. This allows dynamically creating a QR code, such as to represent a unique link for joining a multiplayer game.

**Set correction level**

Change the correction level of the QR code. See the *Correction level* property above for more details.

## QR code expressions

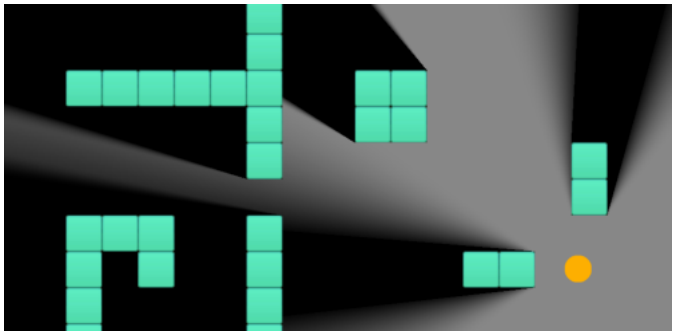
The QR code object does not have any of its own expressions. However see Common expressions for features shared with other objects.

# SHADOW LIGHT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/shadow-light

The **Shadow light** object can render real-time shadows from other objects with the Shadow caster behavior.

Construct comes with a number of examples of shadow-casting effects. Search for *Shadows* in the Example Browser to locate them.



*Example of a shadow-casting effect*

### Scripting

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

## Shadow rendering

The Shadow light object renders shadows adjacent to objects with the *Shadow caster* behavior, using the object's collision polygon and the relative location of the light. Shadows are filled in over the background, as opposed to rendering sections of light. The Z order of the Shadow light object determines whether the shadows appear above or below other objects.

The light can be set to have a radius. If the radius is 0, it acts like a point source, and shadows have hard edges. If the radius is larger, it accurately renders penumbras (the transition from lightness to darkness) at the edges of shadows. However in this mode the light height is ignored and all shadows extend off the screen.

## Shadow limitations

Due to the shadow rendering algorithm, there are a couple of limitations:

- The shadow casters can only use convex collision polygons. Shadows will not render correctly if they use concave polygons. If you need a concave shape, this can always be

achieved by placing multiple shadow caster objects next to each other to compose a concave shape out of convex parts.

- Antumbras (beyond where the umbra converges to a point) are not rendered. To avoid the umbra converging to a point on-screen, avoid using shadow casters smaller than the light radius. Stick to large shadow casters and a small light radius.
- When using a light radius, avoid placing the light very close to or directly over a shadow caster. Shadows can fail to render correctly in these circumstances.

## Tags

By default, all shadow lights cast shadows off all shadow casters. In some cases it is desirable to only have certain shadow lights cast shadows off certain shadow caster objects. Each shadow light object can be assigned a tag, and the *Cast from* property set to only cast shadows from that object off shadow caster behaviors with the same (or different) tags.

## Shadow light properties

**Light height**

The height of the light, used with the shadow caster object heights to calculate the length of shadow to cast. This property only has an effect if the *Light radius* is 0, otherwise shadows always extend offscreen.

**Light radius**

The radius of the light. If the radius is 0, the light acts like a point source and shadows are hard-edged. If the radius is larger the object will render penumbras at the edges of shadows. The larger the radius, the wider the penumbras will be. For correct rendering avoid using a large radius, and especially avoid making the radius larger than any of the shadow caster objects. If the radius is not 0, the light height is ignored and shadows always extend offscreen.

**Cast from**

Which shadow caster objects to render shadows for from this object. The options are:

- **All:** every shadow caster object will get a shadow rendered for this light.
- **Same tag:** shadows will only be rendered for shadow casters with the same Tag property.
- **Different tag:** shadows will only be rendered for shadow casters with a different Tag property.

**Tag**

Used to determine which shadow casters to render shadows for, depending on the *Cast from* mode. If *Cast from* is set to *All*, the tag is ignored.

**Preview Paid plans only**

Enable to run a preview of the shadow casting effect directly in the Layout View.

## Shadow light conditions

The Shadow light object does not have any of its own conditions.

## Shadow light actions

**Set cast from**

**Set light height**

**Set tag**

Set the corresponding object properties. For more information see *Shadow light properties*.

**Set light position**

Sets the position in the layout from which shadows are cast from.

> *Note that using the normal Set position action will also update the light position. However the Shadow Light object automatically positions itself in the middle of the viewport in order to draw over the whole screen. Using Set position to set the light position in the middle of the viewport may conflict with its automatic positioning, so this action can be used as a more reliable way to guarantee the light position is placed at the given location.*

**Set shadow color**

Set the color of the shadows that are rendered by the light. The default is black. Use an expression of the form `rgb(red, green, blue)`. To set the opacity of the shadows, change the opacity of the *Shadow light* object.

## Shadow light expressions

**LightX**

**LightY**

The X and Y co-ordinates of the light source in the layout. Note a quirk: the light source is moved using the ordinary *Set position* actions, but due to the way the object rendering works

the ordinary X and Y expressions always return a position relative to the viewport instead. The *LightX* and *LightY* expressions return the actual position of the light source.

---

**Tag**

Return the current tag of the object.

# SHARE

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

The **Share** object can bring up the system share feature, allowing the user to share something via another app.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Web Share API.

## Sharing files

The Share plugin can add files to a share, such as images or videos, if *Is sharing files supported* is true. The contents of the file is taken from a Binary Data object. Often Construct can provide a screenshot or video as a local URL. The process to share this as a file is as follows:

1. Use the AJAX object to load the URL in to a Binary Data object. This works like any other AJAX request, except using the *Set response binary* action before the request action.
2. Once the AJAX request completes successfully, the data from the URL is in the Binary Data object.
3. Now you can use the *Add file* action to attach the contents of the Binary Data object to the next share. Use the *Add file* action immediately before the *Share* action to attach the file to the next share.

For an example of this, open the 'Taking screenshots' example which demonstrates taking a canvas snapshot and sharing it as a file.

## Share conditions

**Is supported**

Check whether sharing is supported on the current platform. Sharing will only work if this is true.

**Is sharing files supported**

Check whether sharing files with the *Add file* action is supported on the current platform. The *Add file* action will only work if this is true. If it is false but *Is supported* is still true, then the *Share* action can still be used to share text and a URL.

**On share completed**

Triggered after a share action once the user completes the share process.

> *This does not necessarily mean anything was shared - this can be triggered if the user cancels the share.*

---

### On share failed

Triggered if a share action is not successfully completed or an error otherwise occurs.

## Share actions

---

### Add file

Attach a file to the next share using the contents of a Binary Data object, with a given filename. This can be used multiple times before a *Share* action to attach multiple files to be shared, such as a series of screenshots. The type of the data must also be specified, which is normally `"image/png"` for a screenshot, or `"video/webm"` for a video, but can also be other types (see MIME Types). For more information see *Sharing files* above.

---

### Share

Use the system share feature to share some text via another app. *Text* is the text to share; *Title* is an optional title to use (which can be used for other fields, such as the subject of an email if shared to an email app); and *URL* is a link to share. All three fields are optional, but at least one must be provided. If *Is sharing files supported* is true and any *Add file* actions were used before this action, those files are attached to the share.

> *To avoid annoying the user, browsers may only allow this action in a user input event, such as On button clicked, On touch start, etc. So to ensure sharing works, only use this action in a user input trigger.*

---

### Request app rating

Display a dialog to the user requesting that they rate your application. Where available this uses the platform's built-in request app rating dialog, which has predefined text. On Android if this is not supported it falls back to using a custom system dialog, the content of which is defined by the parameters to the action. You should request review at an appropriate time, and infrequently, as so not to frustrate your users. For iOS see Apple's guide on ratings, and for Android see the Google Play In-App Review API guide.

> *This action is not guaranteed to show anything to the user. For the best user experience app stores limit how often the dialog is allowed to appear. Check the linked guides for more details.*

**Display store page**

Display the native store page (Google Play/App Store) for an application with a given identifier. Note the app ID can be left empty to use the app ID specified in Project Properties.

## Share expressions

The *Share* object has no expressions.

# SLIDER BAR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/slider-bar

The **Slider bar** object provides a simple form control allowing the user to pick a value between a minimum and maximum by moving a slider along a bar.



### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Slider bar properties

**Minimum**

The lowest value that can be picked, when the slider is all the way to the left.

**Maximum**

The highest value that can be picked, when the slider is all the way to the right.

**Step**

The increment of possible values. For example if the step is 10, then the slider will jump in units of 10 as it is moved, and only a multiple of 10 can be chosen as a value.

**Tooltip**

An optional tooltip to show while hovering the mouse over the control.

**Initially visibile**

Whether the control is initially visible or invisible in the page.

**Enabled**

Enabled

Whether the control is initially enabled and usable, or disabled so that it cannot be interacted with.

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

**Class Optional**

An optional *class* attribute for the element in the DOM (Document Object Model). This can be useful for CSS styling.

## Slider bar conditions

See common conditions for features shared between form control objects.

**Compare value**

Compare the currently chosen value from the slider bar.

**On changed**

Triggered when the user finishes changing the chosen value on the slider bar. Typically this only triggers when the user releases a mouse button or touch after moving the slider.

**On changing**

Triggered repeatedly as the user changes the chosen value on the slider bar. Unlike *On changed* this will reflect the current value of the slider as the user is still dragging it.

**On clicked**

Triggered when the user clicks the slider bar.

## Slider bar actions

See common actions for features shared between form control objects.

**Set maximum**

Set the maximum value that can be chosen from the slider bar.

**Set minimum**

Set the minimum value that can be chosen from the slider bar.

### Set step

Set the increment step of the slider bar.

### Set tooltip

Set the tooltip that appears when the mouse hovers over the slider bar.

### Set value

Set the currently selected value of the slider bar. This must be between the currently set minimum and maximum values.

## Slider bar expressions

### Maximum

Return the currently set maximum slider value.

### Minimum

Return the currently set minimum slider value.

### Step

Return the currently set slider step value (increment).

### Value

Return the current value chosen by the user, between the minimum and maximum values.

# SPEECH RECOGNITION

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/speech-recognition

The **Speech recognition** object can transcribe text from the audio of the user talking in to a microphone.

Speech recognition may not be supported by all browsers or platforms. Use the *Supports speech recongition* condition to check if speech synthesis can be used.

> *Starting speech recognition requires access to the user's microphone, which normally requires a permission prompt for security reasons. To avoid annoying the user, the permission prompt may also only be allowed to start in a user input trigger, such as On button clicked or On touch started.*

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Web Speech API.

## Speech recognition conditions

**Is recognising speech**

True if a speech recognition request has been approved, and speech input through a microphone is actively being recognised.

**On end**

Triggered after the *Stop speech recognition* action, or after the user stops speaking in *Single phrase* mode speech recognition.

**On error**

Triggered if there is an error approving speech recognition, or during speech recognition. The *SpeechError* expression is set to a string which describes the type of problem, e.g. "not-allowed" if permission was declined.

**On result**

Triggered during active speech recognition when the interim or final transcript has changed. Use either the *FinalTranscript* and/or the *InterimTranscript* expressions to get the updated

result.

### On start

Triggered after *Request speech recognition* when the user has also approved any prompt for permission.

### Supports speech recognition

True if the current browser or platform supports speech recognition. If false, none of the speech recognition features of the object will work.

## Speech recognition actions

### Request speech recognition

If *Supports speech recognition* is true, initiates speech recognition. Usually a permission prompt will appear asking the user if they want to allow the page to use their microphone input. The user must approve the permission prompt before *On start* triggers. If there is a problem or permission is denied, *On error* is triggered.
*Language* specifies the spoken language to recognize, as an IETF language tag. Use a tag like *en* for English, *en-US* for US English, *en-GB* for British English, and so on.
*Mode* can be *continuous*, which keeps recognising speech until the page is closed or the *Stop speech recognition* is used; or *single phrase*, which recognises speech until the user stops talking, then automatically stops speech recognition and triggers *On end*.
*Results* can be *Interim* to allow interim (unconfirmed) results which can change, accessed by the *InterimTranscript* expression; or *Final* to only allow confirmed final results of speech recognition to be returned which will not change, accessed by the *FinalTranscript* expression.

### Stop speech recognition

If speech recognition is currently active, ends the speech recognition. *On end* will trigger.

## Speech recognition expressions

### FinalTranscript

If speech recognition is active, returns the final transcript of confirmed results. This does not change, other than to add newly spoken words which have also been confirmed.

### InterimTranscript

If speech recognition is active, returns the interim transcript of results. The *Request speech recognition* action must have specified *Interim* for the *Results* parameter. The text of this expression can change, as the speech recognition engine uses the sound input in real-time to refine the results and correct any misinterpreted words. Once the user has spoken far

enough for the speech recognition engine to be confident of a final result, the word will disappear from *InterimTranscript* and be appended to *FinalTranscript*.

---

**SpeechError**

In *On speech recognition error*, contains a string which identifies the type of error. Possible values are: `"no-speech"`, `"aborted"`, `"audio-capture"`, `"network"`, `"not-allowed"`, `"service-not-allowed"`, `"bad-grammar"`, or `"language-not-supported"`. The most common errors are `"not-allowed"` if the user declined the permission prompt; `"audio-capture"` if no microphone is present; or `"network"` if the speech recognition is implemented by a remote server over the Internet which is currently unavailable.

# SPEECH SYNTHESIS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/speech-synthesis

The **Speech synthesis** object can automatically speak some text using a synthetic voice, also known as text-to-speech (TTS).

Speech synthesis may not be supported by all browsers or platforms. Use the *Supports speech synthesis* condition to check if speech synthesis can be used.

> *Starting speech synthesis is treated similarly to audio playback by some browsers. This means in order to avoid annoying the user it may not be able to autoplay on startup. It may also only be allowed to start in a user input trigger, such as On button clicked or On touch started.*

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Web Speech API.

## Speech synthesis conditions

**Is speaking**

True if the speech synthesis engine is currently reading out some text.

**On speech ended**

Triggered when the speech started by *Speak text* finishes being read out.

**On speech error**

Triggered if an error occurs during speech synthesis.

**Supports speech synthesis**

True if the current browser/platform supports speech synthesis, so the *Speak text* action can be used

## Speech synthesis actions

**Pause speaking**

**Resume speaking**

Pause or resume text being read out by speech synthesis from the *Speak text* action.

---

**Speak text**

Read out some text using speech synthesis. The language, volume, rate and pitch of the voice that reads out the text can be customised. The *Voice URI* can be used to select a different kind of voice (e.g. male vs. female) from a list of the supported voices, if any alternatives are available. The list of possible voices can be retrieved using the *VoiceCount* and *VoiceURIAt* expressions.

---

**Stop speaking**

Stop reading out text from a previous *Speak text* action. The speech cannot be resumed.

## Speech synthesis expressions

---

**VoiceCount**

Return the number of voices available for use with speech synthesis.

---

**VoiceLangAt(index)**

**VoiceNameAt(index)**

**VoiceURIAt(index)**

Return the language, name, or URI of the voice at the given zero-based index. This can be used to show the user a list of possible voices to choose. To select a different voice, pass the appropriate voice URI to the *Speak text* action.

# SPRITE

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

The **Sprite** object is an animatable image that appears in the project. It is one of the most important objects for most Construct projects. It is used to make most visual elements in a project, such as the player, enemies, projectiles, explosions and non-tiling scenery. (Tiled scenery is much better done with the Tiled Background object.)

If a Sprite has a single animation with a single frame, it just shows an image without animating. However, multiple animations can be added to Sprite objects with the Animations editor.

All instances of Sprite objects share their animations. In other words, there is a single set of images comprising the animations which belongs to the object type, and these images are referenced by instances.

Sprites can have effects applied. For more information, see Effects.

### Scripting

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

## Sprite properties

**Animations**

Click the *Edit* link to open the Animations editor for the object. All instances of the object type share a single set of animations.

**Size**

Click the *Make 1:1* link to size the selection at original size (100%). This makes the width and height of the object the same as its first animation frame image.

**Initially visibile**

Set whether the object is shown (visible) or hidden (invisible) when the layout starts.

**Initial animation**

Set the initially displaying animation.

**Initial frame**

Set the initially displaying animation frame from the object's initial animation. This is a zero-based index, so the first frame is 0.

---

**Enable collisions**

Enable or disable collisions for the object. Disabling collisions means no collision events will register for the object nor will any behaviors on the object register collisions with solids or jump-thrus. Disabling collisions does not improve performance unless there are some events or behaviors that test collisions.

---

**Preview Paid plans only**

Enable to run a preview of the initial animation directly in the Layout View.

## Sprite conditions

For conditions in common to other objects, see Common conditions.

---

**Compare frame**

Compare the current animation frame number, which is a zero-based index (the first frame is 0).

---

**Compare frame tag**

Compare the current animation frame tag.

---

**Compare speed**

Compare the speed of the current animation, in animation frames per second. Animations which are playing backwards (e.g. with ping-pong animations) have a negative speed.

---

**Is flipped**

**Is mirrored**

True if the object has been flipped or mirrored with the *Set flipped* or *Set mirrored* actions.

---

**Is playing**

True if a given animation is currently set. Animations are identified by their name (case insensitive).

---

**On any finished**

Triggered when any animation reaches the end. Looping animations do not finish.

**On finished**

Triggered when a given animation reaches the end. Looping animations do not finish. Animations are identified by their name (case insensitive).

**On frame changed**

Triggered whenever the animation switches to another frame while the animation is playing.

**Collisions enabled**

True if the object's collisions are currently enabled.

**On image URL loaded**

**On image URL failed to load**

Triggered when *Load image from URL* finishes downloading the image and is ready to display it, or if the load fails.

## Sprite actions

For actions common to other objects, see Common actions.

**Add/remove animation**

Add or remove an animation with the provided animation name. When adding a new animation, the animation name must be unique, and the new animation will have a single transparent frame sized 100x100. The last animation cannot be removed, as Sprite objects must have at least one animation.

> *Animations are shared between all instances of the same object type, so this action will affect all instances regardless of which are picked in the conditions.*

**Add/remove animation frame**

Add or remove an animation frame in the animation with the given name. When adding a new frame, it will be transparent and sized 100x100. The last animation frame cannot be removed, as an animation must have at least one frame. The *Where* parameter is the zero-based index, or animation frame tag, of the location to modify, and can be -1 to refer to the last frame in the animation. When adding an animation frame not at the end, it is inserted just before the given frame.

> *Animations are shared between all instances of the same object type, so this action will affect all instances regardless of which are picked in the conditions.*

### Set animation

Change the currently playing animation to another animation. Animations are identified by their name (case insensitive). The new animation can either play from the *beginning* or from the same frame number as the last animation was on (*current frame*).

> *Note that if the set animation is already playing, this action does nothing, even if set to play from the beginning. If you intend to restart the animation, use the Start action and choose from beginning.*

### Set flipped

Set whether the object image appears vertically flipped or normal. This also affects image points and the collision polygon. This is a shortcut for inverting the height (a flipped Sprite's height is a negative size).

### Set mirrored

Set whether the object image appears horizontally mirrored or normal. This is useful for platform games. Mirroring also affects image points and the collision polygon. This is a shortcut for inverting the width (a mirrored Sprite's width is a negative size).

### Set frame

Set the current animation frame that is showing, either by its zero-based index, or a string of the animation frame tag. If a tag is provided and there are multiple animation frames with the same tag, then it will use the first one. After this action the animation will continue to play at its current speed.

### Set repeat-to frame

Set the frame to return to when looping in the current animation, either by its zero-based index, or a string of the animation frame tag. This essentially changes the *Repeat to* property of the animation in the Animation Editor. It is especially useful when reversing animations, since the default of repeating to the first frame is no longer suitable. Instead when playing looping animations in reverse it is more useful to repeat to the last frame of the animation (so the animation actually repeats, instead of getting stuck on the first frame).

### Set speed

Set the playback rate of the current animation, in animation frames per second. Instances can have different animation speeds. You can also use negative speeds, which causes the animation to play backwards. Note in this case repeating animations should set the *Repeat to* frame at the *end* of the animation, otherwise by default it repeats to frame 0 (the start of the animation), causing the animation to stop after playing in reverse.

### Start

If the current animation is stopped, start playing the animation again. Playback can either resume from the *current frame*, or restart from the *beginning*.

### Stop

Stop the current animation from playing. The object will be left showing the current animation frame.

### Spawn another object

Create a new instance of a given object type. The new instance is created at the current object's position and also set to the same angle. The created object can be on any layer (chosen by its name or its zero-based number), and it can be positioned by an image point instead of the object's origin (chosen by its name or number). If a Family Paid plans only is created, a random object type in the family is picked. 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.*

### Set scale

Sets the width and height to a multiple of the object's original size, similar to zooming the object proportionally. For example, if the object is 50x100, *Set scale to 2* will set its size to 100x200, and *Set scale to 0.1* will set its size to 5x10.

### Load image from URL

Load an image from a given URL. The current animation frame will be replaced with the image. It is not shown until the image has finished downloading, and *On image URL loaded* triggers. Images loaded from different domains are subject to the same cross-domain restrictions as AJAX requests - for more information see the section on cross-domain in the AJAX object. Data URIs can also be passed as an image, e.g. from a canvas snapshot or webcam image. The *Size* parameter sets whether the Sprite object will be set to the image size when it loads, or whether to keep its current size and stretch the image.

> *Note that as all Sprite instances share the same set of animations and frames, loading an image will replace the image for all instances of the object type. If you want to dynamically load images for individual instances, try adding animation frames and loading a different image in to each frame.*

---

**Set collisions enabled**

Enable or disable collisions for the object. Disabling collisions means no collision events will register for the object nor will any behaviors on the object register collisions with solids or jump-thrus. Disabling collisions does not improve performance unless there are some events or behaviors that test collisions.

---

**Set solid collision filter**

Enable or disable collisions with the Solid behavior according to tags. Specify tags using a string of space-separated tag names. In *Inclusive* mode, collisions are only enabled with solids that match any of the given tags; if no tags are specified, collisions are disabled with all solids. In *Exclusive* mode, collisions are disabled with solids that match any of the given tags; if no tags are specified, collisions are enabled for all solids (the default).

## Sprite expressions

For expressions common to other objects, see common expressions.

---

**AnimationFrame**

The currently displaying zero-based animation frame number.

---

**AnimationFrameTag**

The string tag of the currently displaying animation frame.

---

**AnimationFrameCount**

The number of animation frames in the current animation.

---

**AnimationName**

A string containing the name of the currently playing animation.

---

**AnimationSpeed**

The current playback rate of the current animation, in animation frames per second. If the animation is playing backwards (e.g. ping-pong animations), the animation speed is negative.

---

**OriginalAnimationSpeed**

The speed of the current animation as specified in the Animations Editor. This does not change if the animation speed is altered at runtime. It is useful for setting the animation speed to a multiplier of the original speed.

---

**ImageWidth**

**ImageHeight**

The original dimensions of the object (its current animation frame image size), in pixels. Since objects can be stretched at runtime causing the normal *Width* and *Height* expressions to return different values, these can be used to get the original size regardless of the stretched size.

---

**ImagePointCount**

Return the number of image points on the currently displaying animation frame of the object.

> *The count excludes the origin.*

---

**ImagePointX(nameOrIndex)**

**ImagePointY(nameOrIndex)**

Retrieve the position of an image point on the currently displaying animation frame of the object. You can pass either the zero-based index of the image point, or a string of its name.

> *When using an index, the origin is excluded, so 0 means the first added image point.*

---

**PolyPointCount**

Return the number of collision polygon points on the currently displaying animation frame of the object.

---

**PolyPointXAt(index)**

**PolyPointYAt(index)**

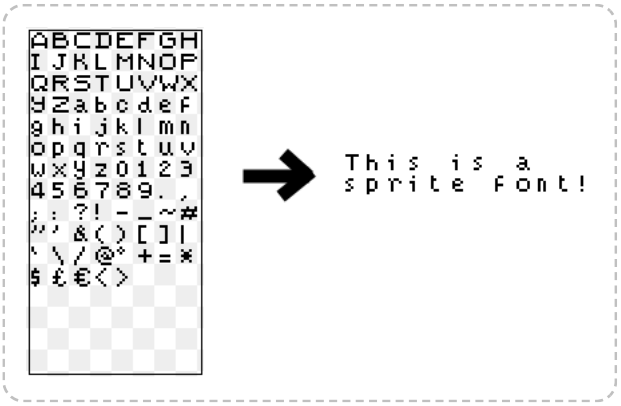
Retrieve the position of a collision polygon point on the currently displaying animation frame of the object, by its zero-based index.

*The first poly point is repeated again at the end (at the index PolyPointCount) since it makes it easier to iterate through each edge of the collision polygon.*

# SPRITE FONT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/sprite-font

The **Sprite Font** object uses an image to display text. The "sprite font" is the object image, which contains a grid of every character that can be drawn. By drawing parts of this image in sequence, strings of text can be rendered. This allows complete artistic control over the appearance of text, rather than having to choose from existing fonts.




### Scripting

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

## Comparison with Text object

Both the Sprite Font and Text objects can display text in the project. Typically the Text object shows monochrome characters from an existing font or web font, which can use a range of sizes and possibly also bold and italic options. On the other hand Sprite Font uses images for each character. While this means any kind of artwork can be used for text, notably allowing for multi-colored text, it has the trade-off that it only really supports one font size and one bold/italic setting (those that it is drawn with).

Another important difference is that traditional fonts as used by the Text object often have good support for unicode characters. This allows them to display a wide range of characters, including many alphabets and character sets from many different languages, as well as emoji. Sprite Fonts however can only use the characters for which an image has been drawn. It is very difficult to make a sprite font that covers much of the tens of thousands of possible unicode characters. If a Sprite Font is set to show some text which contains a character that has not been drawn, it will simply show an empty space for that character. If the entire string is in a different language which the sprite font does not cover, nothing will render at all. Bear in mind that if you allow user-inputted text, such as the player's name, or you wish to translate the project in future, then Text objects are probably more suitable.

## Re-coloring SpriteFonts

SpriteFont objects have a *Color* property that can be used to conveniently re-color the text (and the `[color]` BBCode tag works similarly; see below). This works by applying a color filter, which works best if the SpriteFont is drawn with white text, since that can be filtered to any other color. For this reason the default SpriteFont is drawn white. The *Color* property is set to blue by default in order to help identify that the color of the SpriteFont is set by this property. So long as the image is drawn white, the color of the text can be set to any color with this property.

## Using BBCode

By default the SpriteFont object allows the use of BBCode, a simple way of marking up text for formatting. If you don't want such tags to affect the formatting of the text, you can opt-out of it by unchecking the *Enable BBCode* property.

BBCode uses "tags" in square brackets to mark the start and end of formatting. For example to hide a word, wrap it in `[hide]` and `[/hide]`, e.g. `[hide]Hello[/hide]`. Some tags take a parameter, such as the scale, which is specified after an equals sign in the opening tag, e.g. `[scale=2]Hello[/scale]`.

The following tags are supported. Note that due to the fact SpriteFonts render images for text, the supported BBCode tags differ from those used by the Text object.

- `[scalex=2]stretch wider[/scalex]`
- `[scaley=2]stretch taller[/scaley]`
- `[scale=2]stretch both axes[/scale]`
- `[color=#ff0000]change text color[/color]` - the color can be specified in the same way CSS colors are specified, e.g. hexadecimal, using `rgb()`, etc. Note that for SpriteFont, the color is applied as a tint. To ensure you can use any color text, use a SpriteFont with the characters drawn in a white color.
- `[opacity=50]change text opacity[/opacity]`
- `[hide]invisible text[/hide]` - this is useful for flashing effects, since the text still takes up the same width while invisible
- `[background=#ff0000]change background color[/background]`
- `[offsetx=10]offset X[/offsetx]` and `[offsety=10]offset Y[/offsety]` - move text by a number of pixels on each axis, useful for animated effects
- `[tag=mytag]tag a range of text[/tag]`, assigns the tag "mytag" to a range of text, which can then be referred to in events (e.g. the *Has tag at position* condition, or expressions to get its size and position). Note see the section *Tagged range fragmentation* in the Text object manual entry for more details (as fragmentation works the same for both SpriteFont and Text objects).

## Sprite font properties

**Text**

The initial text to display.

**Sprite font**

Click the Edit link to edit the source image that text characters are rendered from. The image can be any size, but it should fit the characters it contains exactly. Characters start in the top-left and the sequence moves to the right, wrapping down to the next line when it reaches the right edge of the image. If the character is narrower than the cell, and you change its width using *Spacing data* or the *Set character width* action, the image should be drawn left-aligned in the cell.

**Character width**

**Character height**

The size of each character's cell in the sprite font image. Individual characters can be displayed with a different width using *Spacing data* or the *Set character width* action. In this case, the character should be drawn left-aligned within its cell.

**Character set**

A string of characters that describes the sequence of letters in the sprite font image. This is used to map text to images. While the default starts with the English alphabet, it could be changed to another language or sequence and the image updated accordingly. Note however the Sprite Font can only display characters that are in the character set; any characters not in the character set with a corresponding image will appear as an empty space.

**Spacing data**

Some data in JSON format that lists the widths of individual characters. This allows improved text layout by using narrower spaces for narrower characters. The spacing data also affects the display of the text in the Layout View. The data is an array of pairs. Each pair is a width, and then a string of all the characters that width applies to. For example the pair `[10, "aeou"]` will set the width of the characters *a*, *e*, *o* and *u* to 10 pixels. The characters are case-sensitive, allowing you to choose different widths for uppercase characters. You can also set the width of the space character. Each pair must be listed in an array, e.g. `[[10, "aeou"], [12, "mvw"]]`.

**Scale**

A multiplier to scale the rendered text with, such as 0.5 for half as big or 2 for twice as big. This can be used to "fake" different font sizes, but remember it's only stretching images; you may want to draw the font again at a different size instead of using a scale.

**Character spacing**

Extra space in pixels to add horizontally between characters.

**Line height**

Extra space in pixels to add vertically between lines. 0 is the default size, negative values make lines closer together, and positive values space lines out further apart.

**Horizontal alignment**

The horizontal alignment of the text within the object bounding rectangle.

**Vertical alignment**

The vertical alignment of the text within the object bounding box.

**Wrapping**

Choose how text wraps at the end of a line. *Word* will wrap entire words separated by spaces or hyphens. *Character* will wrap to the next line on any character, even punctuation. *CJK* works similarly to *Character*, but has special handling for Chinese, Japanese and Korean punctuation characters, intended to ensure punctuation wraps appropriately.

**Initially visible**

Whether the object is initially visible or invisible when the layout starts.

**Origin**

Choose the position of the origin relative to its unrotated bounding rectangle.

## Sprite font conditions

**Compare text**

Compare the current text the object is showing.

**Has tag at position**

Test if there is text with a specific tag at the given position (case insensitive). For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then testing if the tag "mytag" is at a given position will check if that position is over just the part of the text that says "world".

**Is running typewriter text**

True while text is being written out using the *Typewriter text* action.

**On typewriter text finished**

Triggered when text being written out using the *Typewriter text* action finishes writing out all the text.

## Sprite font actions

**Append text**

Add some text to the end of the existing text.

**Set character spacing**

**Set line height**

**Set scale**

**Set horizontal alignment**

**Set vertical alignment**

**Set wrapping**

Set the corresponding object properties. For more information, see *Sprite font properties*.

**Set character width**

Set the width of certain characters. Normally it is preferable to use the *Spacing data* property, since it displays with correct spacing in the Layout View. In this action you can specify multiple characters at the same time to set their widths simultaneously, including the space character.

**Set text**

Replace the current text with a new string.

**Typewriter text**

Set the text over time by starting with an empty string and gradually adding characters until the full text is written out, over a duration specified in seconds. Once the full text is written out, *On typewriter text finished* triggers. Note using *Set text* or *Append text* while text is being written out will cancel the effect.

> *You can use a speed in characters per second instead of an overall time by using an expression like* `len(Self.PlainText) / 10` *for the time. In this case it will write out 10 characters per second regardless of the length of the string.*

**Finish typewriter**

If text is being written out with the *Typewriter text* action, force it to finish immediately.

## Sprite font expressions

---

**CharacterHeight**

Return the sprite font cell height.

---

**CharacterScale**

**CharacterSpacing**

**LineHeight**

Return the corresponding object properties. For more information, see *Sprite font properties*.

---

**CharacterWidth(char)**

Return the width of a character. A character must be passed (as a string) so the *Spacing data* or *Set character width* action can be taken in to account. Since the expression can only return one value, if there are multiple characters in the string, only the first is used.

---

**Text**

Return the object's current text.

---

**PlainText**

Return a string containing the object's current text, with any BBCode tags stripped out. For example if the text is `[b]Hello[/b]`, the *Text* expression will return that (with BBCode tags included), but the *PlainText* expression will return just `Hello`.

---

**TagAtPosition(x, y)**

Look up the tag for a part of the text at a given position and return the tag if any, else return an empty string if no tag is specified. For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then the tag at a position over the word "world" is "mytag", and the tag at a position over the word "Hello" is "" (an empty string).

---

**TagCount(tag)**

**TagX(tag, index)**

**TagY(tag, index)**

**TagWidth(tag, index)**

**TagHeight(tag, index)**

Identify the size and position of all ranges of the text with a given tag. Note the count and the index actually refers to *fragments*, as a single tagged range may be broken up in to multiple pieces - see the section *Tagged range fragmentation* in the Text object manual entry for more details (as fragmentation works the same for both SpriteFont and Text objects).

---

**TextWidth**

**TextHeight**

Return the size of the actual text content within the Sprite Font object's rectangle.

# SVG PICTURE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/svg-picture

The **SVG Picture** object can display a Scalable Vector Graphics (SVG) file in your project.

SVG Picture currently does not support animations and only shows a static image.

> *Animated SVGs can be displayed with the HTML Element object.*

### Scripting

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

## Importing an SVG image

The easiest way to add an SVG file you your project is to **drag-and-drop an SVG file in to the Layout View**. This will automatically import the SVG file as a project file, create an SVG Picture object, and set the SVG Picture to display the imported file.

Alternatively you can follow this process manually:

1. Import the SVG file as a project file
2. Create an SVG Picture object
3. Set the *SVG file* property to the imported SVG file

## Other ways to use SVG

You can also import SVG files in the Animations Editor for use in other objects like Sprite. This will rasterize them, which means converting them to a bitmap image at a fixed size. This loses some of the benefits of scaling SVGs, but allows them to be used in other objects, including as a part of Sprite animations, and provides other features such as customizing the collision polygon.

## SVG Picture properties

**SVG file**

Choose the SVG file to display. The SVG file must have been imported as a project file in the *Files* folder.

**Image**

When an SVG file is selected, the *View* link provides a shortcut to preview or edit it. This is the same view you get when double-clicking the SVG file in the Project Bar.

---

**Initially visible**

Choose whether the object is shown (visible) or hidden (invisible) when the layout starts.

---

**Origin**

Choose the position of the origin of the object relative to its unrotated bounding rectangle.

## SVG Picture conditions

SVG Picture does not have any of its own conditions. For conditions in common to other objects, see Common conditions.

## SVG Picture actions

---

**Set image**

**Set image (by name)**

Set the SVG file being displayed by the object, either by a dropdown list or by an expression of the filename.

## SVG Picture conditions

SVG Picture does not have any of its own expressions. For expressions in common to other objects, see Common expressions.

# TEXT

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

The **Text** object can display text using a font in your project. Note that there are not many built-in fonts common to all computers. Instead you can import **web fonts** for use with the Text object.

Note the Text object is used for displaying text only. Don't confuse it with the Text input object, which is a form control used for entering text in to.

### Scripting

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

## Using web fonts

Follow these steps to use a custom web font in the Text object.

1. Locate a web font to use, in WOFF or WOFF2 format. There are some web services that list web fonts. (Be sure to check the web font license to ensure you use it correctly.)

   > *Be sure to use WOFF or WOFF2 format fonts, as these are the only formats that work consistently across all platforms. If you use other formats like TTF or OTF, they may appear to work, but then stop working when you export your project to a different platform.*

2. Download the **.woff** or **.woff2** file for the web font.
3. In the Project Bar, **right-click** the **Fonts** folder and select Import fonts.
4. Import the .woff or .woff2 file you downloaded previously. This will add the web font file as a project file.
5. Select a Text object in the Layout View, and click the button next to the *Font* property in the Properties Bar.
6. In the Font Picker dialog, pick the web font you imported from the second drop-down list (under *Or pick a web font from this project*), and click OK.

The Text object will now be displaying the custom web font in the Layout View. Since the web font is bundled with your project, it will be available on any platform.

Note: the Free Edition of Construct is limited to only importing one web font.

## Text rendering

The Text object does not display anything if its bounding rectangle is too small to fit a single letter of text. If text objects appear to go invisible, try resizing them larger.

Different browsers render text in different ways. This means you should expect the appearance of the Text object to vary slightly across browsers. You should test your project in a range of browsers to ensure text objects display how you intend for all users. For more information see Best practices.

## Using BBCode

By default the Text object allows the use of BBCode, a simple way of marking up text for formatting like **bold** and *italic*. If you don't want such tags to affect the formatting of the text, you can opt-out of it by unchecking the *Enable BBCode* property.

BBCode uses "tags" in square brackets to mark the start and end of formatting. For example to make a word bold, wrap it in `[b]` and `[/b]`, e.g. `[b]Hello[/b]`. Some tags take a parameter, such as the font name to use, which is specified after an equals sign in the opening tag, e.g. `[font=Arial]Hello[/font]`.

The following tags are supported:

- `[b]bold text[/b]`
- `[i]italic text[/i]`
- `[u]underline text[/u]`
- `[s]strikethrough text[/s]`
- `[size=20]change font size (in pt)[/size]`
- `[font=Arial]change font face[/font]` - you can also use any web font imported to the project.
- `[color=#ff0000]change text color[/color]` - the color can be specified in the same way CSS colors are specified, e.g. hexadecimal, using `rgb()`, etc.
- `[opacity=50]change text opacity[/opacity]`
- `[hide]invisible text[/hide]` - this is useful for flashing effects, since the text still takes up the same width while invisible
- `[background=#ff0000]change background color[/background]`
- `[offsetx=10]offset X[/offsetx]` and `[offsety=10]offset Y[/offsety]` - move text by a number of pixels on each axis, useful for animated effects. The offset can optionally also be specified as a percentage of the line height, e.g. `[offsety=50%]` means offset downwards by half the line height.
- `[stroke]stroke text[/stroke]`, drawing an outline rather than a solid fill
- `[outline=#ff0000]outlined text (on top)[/outline]`, which adds an outline with a different color drawn on top of the text (as opposed to stroke, which removes the fill)

- `[outlineback=#ff0000]outlined text (underneath)[/outline]`, which adds an outline with a different color drawn behind the text. This variant can look better with very thick outlines.
- `[lineThickness=2]change line thickness[/lineThickness]`, affecting the line thickness used for stroke, outline, strikethrough and underline
- `[icon=0]` or `[icon=tag]` - insert an icon to the text. The icons are taken from the animation frames of a Sprite object set in the *Icon set* property. The icon can be referred to by its zero-based frame index, or by the *Tag* property of the animation frame.
- `[iconoffsety=50%]icon offset Y[/iconoffsety]` - change the vertical alignment of any icons between the tags. A percentage uses an amount relative to the line height. This is relative to the alphabetic baseline, so a value of 0 will align the bottom of the icon with the bottom of the character 'x'. The default icon offset Y is 20% so icons are normally aligned slightly below the alphabetic baseline.
- `[tag=mytag]tag a range of text[/tag]`, assigns the tag "mytag" to a range of text, which can then be referred to in events (e.g. the *Has tag at position* condition, or expressions to get its size and position). Note see the section *Tagged range fragmentation* below for more details.
- `[insert]inserted text[/insert]` does not change the text style, but inserts the given text separately to the rest of the string. This is useful when inserting right-to-left (RTL) text in to left-to-right (LTR) strings: normally RTL text inside an LTR string may change the direction of other text in the string, but if you put the RTL text inside these tags, it will ensure it does not change the direction of any text outside of the tags. It will also prevent other text merging features, such as kerning and ligatures, across the tag boundary.

See the Text formatting example for a demonstration of the various formatting tags, and the Icons in text example for a demonstration of using icon tags.

## Tagged range fragmentation

> *This section also applies to the Sprite Font object.*

As noted above, certain ranges of text can be tagged, e.g.:

```
The [tag=mytag]quick brown fox[/tag] jumps over the lazy dog
```

This will assign the tag "mytag" to the words "quick brown fox". This works straightforwardly with the *Has tag at position* condition and *TagAtPosition* expression. However when accessing the size and position of the tag with the *TagCount*, *TagX* etc. expressions, the tagged range can in fact be split in to multiple *fragments*. There are two reasons this can happen:

1. Word wrap - if the tagged range is broken across two or more lines
2. Changing styles within the tagged range

In the normal case you may expect the above tagged range to count as a single tag with a single size and position. However it may be broken across lines by word wrap, such as shown below:

```
The quick brown
fox jumps over
the lazy dog
```

Notice how the tagged range is now in two separate places across two lines: the first line having "quick brown", and the second line having "fox". Each of the two parts is referred to as a *fragment*. There are now two positions and two sizes for the two fragments that the tagged range was broken in to. Therefore *TagCount("mytag")* will return 2. The other expressions to retrieve the size and position of the tagged range will return each of the fragments separately, identified by the index parameter. Note that long tagged ranges could be broken across three or even more lines, further increasing the number of fragments.

Changing the style of text also fragments the text. For example consider the following BBcode:
`The [tag=mytag]quick [b]brown[/b] fox[/tag] jumps over the lazy dog`

Now inside the tagged range, just the word "brown" is made bold. This also fragments the tagged range, this time in to three fragments: the first with "quick " (including a space), the second "brown" in bold, and the third " fox" (including a space). This time *TagCount("mytag")* will return 3 and each fragment's size and position will be returned separately based on the index parameter.

## Text properties

**Text**

The text for the object to initially be showing.

**Enable BBCode**

Whether to enable the use of BBCode formatting in the text. See above for a list of allowed tags. If disabled, any BBCode tags will simply be displayed as plain text.

**Font**

The font the text object uses to display its text. Click the button to the right of the font name to open a font picker dialog.

> *You may see a permission prompt to access the full list of fonts installed on your system.*

Remember local fonts may not be available on other devices - consider using a web font, as described in *Using web fonts* above.

**Size**

The size of the text to display, in points (pt).

**Line height**

Amount to change the space between each line of text, in pixels. Use 0 for the default amount, -5 for 5 pixels shorter than default, 10 for 10 pixels taller than default, and so on.

**Bold**

Whether to use the **bold** variant of the font, if available.

**Italic**

Whether to use the *italic* variant of the font, if available.

**Color**

Choose the color of the text object's text.

**Horizontal alignment**

Choose whether the text displays left, center or right aligned within its bounding rectangle.

**Vertical alignment**

Choose whether the text displays top, center or bottom aligned within its bounding rectangle.

**Wrapping**

Choose how text wraps at the end of a line. *Word* will wrap entire words separated by spaces or hyphens. *Character* will wrap to the next line on any character, even punctuation. *CJK* works similarly to *Character*, but has special handling for Chinese, Japanese and Korean punctuation characters, intended to ensure punctuation wraps appropriately.

**Text direction**

Set the direction of the text flow. The default is left-to-right (LTR). For some languages the right-to-left (RTL) direction is more appropriate, such as Arabic and Hebrew. The text direction is particularly significant when using BBcode formatting, as it affects the order of formatted fragments.

**Icon set**

Choose a Sprite object to use for BBcode icon tags. Each animation frame of the Sprite object is treated as an individual icon. Icons can then be referred to by the animation frame

index, or the animation frame *Tag* property. See the section *Using BBcode* above for more details.

---

**Initially visibile**

Whether or not the object is shown (visible) or hidden (invisible) when the layout starts.

---

**Origin**

Choose the position of the origin of the object relative to its unrotated bounding rectangle.

---

**Read aloud**

Check to indicate to screen readers that the contents of this text object ought to be read aloud when it appears or changes. By default text objects are not read out by screen readers as they are rendered in to a canvas, which is essentially a large image and so not accessible to screen readers. Further, Text objects are not all automatically read aloud as this can provide a poor screen reader experience, such as constantly reading out a changing score instead of more helpful information. Checking this option for important text objects improves the accessibility of projects to ensure the contents of text objects can be understood by people who cannot necessarily read them visually. The text object does not need to be on-screen, so a dedicated text object for screen readers with this option checked can also be used. See also the Speech Synthesis plugin which can be used for similar purposes.

## Text conditions

For conditions common to other objects, see common conditions.

---

**Compare text**

Test whether the text object is currently displaying a certain string of text. The comparison can be either case sensitive ("TEXT" is different to "text") or case insensitive ("TEXT" is considered the same as "text"). To test if the text object is not showing some text, invert the condition.

---

**Has tag at position**

Test if there is text with a specific tag at the given position (case insensitive). For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then testing if the tag "mytag" is at a given position will check if that position is over just the part of the text that says "world".

---

**Is running typewriter text**

True while text is being written out using the *Typewriter text* action.

---

**On typewriter text finished**

**On typewriter text finished**

Triggered when text being written out using the *Typewriter text* action finishes writing out all the text.

## Text actions

For actions common to other objects, see common actions.

---

**Set font color**

Set the color of the text. Use an expression in the form `rgb(red, green, blue)`.

---

**Set font face**

Change the font used to display the text. This must be the name of a web font imported to the project, or a local font that is pre-installed on the user's device.

---

**Set font size**

Set the size of the text in points (pt).

---

**Set horizontal alignment**

**Set vertical alignment**

**Set line height**

**Set read aloud**

**Set text direction**

**Set wrapping**

Change the corresponding properties. See *Text properties* above for more information.

---

**Set resolution mode**

By default Text objects use *Automatic* resolution mode, which means the resolution of the text adjusts according to the 2D display scale. This produces the best quality display, but only works with 2D display, and can also cause the text to constantly re-render when being smoothly scaled, which can sometimes have a significant performance overhead. *Fixed* resolution mode causes the text to render at a fixed resolution according to a provided scale factor, and ignore the display scale. This usually results in a reduced display quality but better performance.

---

**Append text**

Add some text to the end of the current text. For example, if the text object contains *Hello* and has *World* appended, the text object then contains *HelloWorld*.

### Set text

Set the text the object is currently displaying. Use the **&** operator to combine text and numbers. For more information, see expressions.

### Change icon set

Changes the *Icon set* property, replacing the Sprite used for BBcode icons. This can be used to change the set of icons displayed by the Text object. Note if the new Sprite object does not have the same number of animation frames, or the same animation frame tags, then some icons may disappear.

### Typewriter text

Set the text over time by starting with an empty string and gradually adding characters until the full text is written out, over a duration specified in seconds. Once the full text is written out, *On typewriter text finished* triggers. Note using *Set text* or *Append text* while text is being written out will cancel the effect.

> *You can use a speed in characters per second instead of an overall time by using an expression like* `len(Self.PlainText) / 10` *for the time. In this case it will write out 10 characters per second regardless of the length of the string.*

### Finish typewriter

If text is being written out with the *Typewriter text* action, force it to finish immediately.

### Update HTML

This action converts the contents of the Text object, including any icons, in to a string of HTML. This can then be displayed in the HTML Element object. This action is *asynchronous*: it takes a moment complete, so you need to use it with the *Wait for previous actions to complete* system action before you can use the result in the *AsHTML* expression. See the Text icons to HTML example for a demonstration.

## Text expressions

For expressions common to other objects, see common expressions.

### FaceName

### FaceSize

### LineHeight

Return the corresponding object's properties. See *Text properties* above for more details.

**Text**

Return a string containing the object's current text.

**PlainText**

Return a string containing the object's current text, with any BBCode tags stripped out. For example if the text is `[b]Hello[/b]`, the *Text* expression will return that (with BBCode tags included), but the *PlainText* expression will return just `Hello`.

**AsHTML**

Returns a string of HTML code that represents the content of the Text object, including formatting and icons. This is only available after, and only updated by, the *Update HTML* action completing.

**TagAtPosition(x, y)**

Look up the tag for a part of the text at a given position and return the tag if any, else return an empty string if no tag is specified. For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then the tag at a position over the word "world" is "mytag", and the tag at a position over the word "Hello" is "" (an empty string).

**TagCount(tag)**

**TagX(tag, index)**

**TagY(tag, index)**

**TagWidth(tag, index)**

**TagHeight(tag, index)**

Identify the size and position of all ranges of the text with a given tag. Note the count and the index actually refers to *fragments*, as a single tagged range may be broken up in to multiple pieces - see the section *Tagged range fragmentation* above for more details.

**TextWidth**

**TextHeight**

Return the size of the actual text content within the text object's rectangle.

# TEXT INPUT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/text-input

The **Text input** object is a form control providing a text field the user can type text in to. This is used for getting data from the user; don't confuse it with the Text object, which is for displaying text.

### Scripting

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

## Layering HTML objects

This object displays using a HTML element rather than drawing in to the canvas. This means its layering works differently to other objects. To learn more about how to layer HTML objects, see HTML layers.

## Styling text inputs

As Text Input objects are HTML elements, their appearance can be customised using CSS (Cascading Style Sheets). The *ID* and *Class* properties can be used to identify the HTML element, and a CSS project file added to apply some styles to it.

## Text Input properties

**Text**

The initial text entered in to the field.

**Placeholder**

Some text that appears faintly when the field is empty. This can be used for hints for what the field is for, e.g. *Username*.

**Tooltip**

A tooltip that appears if the user hovers the mouse over the text box and waits. Leave blank for no tooltip.

**Initially visibile**

Whether or not the text box is shown on startup. If invisible, the field must be shown with the *Set visible* action.

**Enabled**

Whether the text box is initially enabled. If disabled, the field will be greyed out and cannot be modified.

**Read-only**

Set whether the field is read-only, which means the text cannot be modified but can still be selected. This is different to disabling the field, which changes the appearance of the input.

**Spell check**

Enable spell-checking on the text entered in to the field, if the browser supports it. If enabled, spelling errors are underlined with a squiggly red line.

**Type**

Set the type of content being entered in to the text field, which can be:

- **Text:** any text content
- **Password:** any content but characters hidden
- **Email:** intended for strings in the format of an email, e.g. *joe@bloggs.com*
- **Number:** numerical digits only
- **Telephone number:** telephone number characters only
- **URL:** web addresses in the general format *https://example.com*
- **Textarea:** a multi-line text input, usually displayed with a monospace font
- **Search:** text content intended as a search query

The *email*, *number*, *telephone number* and *URL* types are generally most useful for mobile devices, since they change which type of on-screen keyboard appears when the field is focused. For example, *Text* will show a general purpose on-screen keyboard, whereas *Number* may show a simple number pad, making it more convenient for the user to enter the content.

**Auto font size**

Automatically set the *font-size* property of the element according to the layout and layer scale. This will prevent the *font-size* CSS property being manually set with the *Set CSS style* action. Disable if you intend to use *Set CSS style* to adjust the *font-size* property.

**ID Optional**

An optional *id* attribute for the element in the DOM (Document Object Model). This can be useful for accessing the element's value from external scripts, or styling with CSS in the HTML page.

## Text input conditions

See common conditions for features shared between form control objects.

**Compare text**

Compare the text currently entered in to the field. The comparison can either be case sensitive ("TEXT" is different to "text") or case insensitive ("TEXT" is the same as "text").

**On clicked**

Triggered when the user clicks the field.

**On double-clicked**

Triggered when the user double-clicks the field.

**On text changed**

Triggered whenever the text in the field is modified, by typing, backspace/delete, cut/paste etc.

## Text input actions

See common actions for features shared between form control objects.

**Append text**

Add some text to the end of the current text. For example, if the text object contains *Hello* and has *World* appended, the text object then contains *HelloWorld*.

**Scroll to bottom**

Scroll to the bottom of the control. Only has an effect when set to the *textarea* type, since it is the only multiline mode. This is useful for chat or log style textareas.

### Set max length

Set the maximum number of characters allowed to be entered in the field. Set to -1 to disable any limit and allow an unlimited number of characters (which is the default).

### Set placeholder

Set the text that appears faintly when the field is empty. This can be used for hints for what the field is for, e.g. *Username*.

### Set read-only

Set whether the field is read-only, which means the text cannot be modified but can still be selected. This is different to disabling the field, where text cannot be selected.

### Set text

Set the text currently entered in to the field.

### Set tooltip

Set the text that appears for the field tooltip. Leave blank for no tooltip.

## Text Input expressions

### MaxLength

Return the maximum number of characters allowed to be entered in to the field, as set by the *Set max length* action. If there is no maximum length (the default), this returns -1.

### Text

Get a string containing the text currently entered in to the field.

# TILED BACKGROUND

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/tiled-background

The **Tiled Background** object can display an image in a repeating pattern, as shown below.



*A Tiled Background object*

This pattern can be achieved with a single Tiled Background object, and it is much faster (and more convenient to edit) than using multiple Sprite objects arranged in a grid. Always prefer using Tiled Background objects wherever an image repeats.

Tiled Backgrounds cannot have a collision polygon. They always collide according to their bounding rectangle.

Tiled Backgrounds can have effects applied. For more information, see Effects.

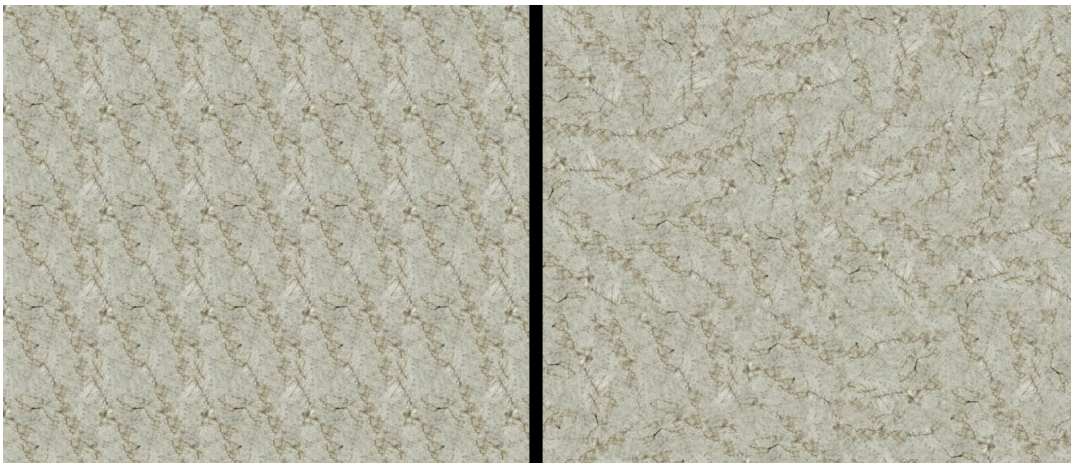
If you wish to have a tile-based project where each tile can be different, consider using a Tilemap object instead.

### Scripting

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

## Tile randomization

If you tile a small image, or zoom out, then the repetitive appearance can become obvious. One easy way to solve this is to enable **Tile randomization** in the Tiled Background properties. The image below shows the effect this can have. See also the interactive Tile Randomization example.



*Left: standard tiling. Right: randomized tiling.*

## Tiled Background properties

**Image**

Click the *Edit* link to edit the Tiled Background's image in the Animations Editor.

**Initially visibile**

Choose whether the object is shown (visible) or hidden (invisible) when the layout starts.

**Origin**

Choose the position of the origin of the object relative to its unrotated bounding rectangle.

**Wrap horizontal**

**Wrap vertical**

Choose how the image repeats on each axis. *Mirrored repeat* alternately mirrors/flips the image for each repeat. *Clamp to edge* can be used to prevent wrapping on one axis, for example to wrap an image vertically only, choose *Clamp to edge* for *Wrap horizontal*. This is useful to avoid unwanted artefacts due to wrapping on the other axis.

**Image offset X**

**Image offset Y**

Offset the displayed Tiled Background image by a number of pixels on each axis.

**Image scale X**

**Image scale Y**

Stretch the displayed Tiled Background image by a percentage on each axis.

**Image angle**

Rotate the displayed Tiled Background image by a number of degrees, relative to the offset position.

**Enable tile randomization**

By default a Tiled Background repeats its image identically in a grid. However this can make the image repetition obvious. Enabling tile randomization helps avoid this by randomly offsetting and rotating each individual tile, and blending the edges to keep the appearance seamless. There are several tile randomization properties that can be changed to adjust the effect and better suit different types of artwork.

**X random**

**Y random**

When using tile randomization, adjust the amount of random horizontal and vertical offset for each tile. This is set as a percentage from 0% (no offset) to 100% (offset up to a full tile's width/height).

**Angle random**

When using tile randomization, adjust the amount of random rotation for each tile. This is set as a percentage from 0% (no rotation) to 100% (rotate up to 360°).

**Blend margin X**

**Blend margin Y**

When using tile randomization, adjust the area over which tiles will blend in to the adjacent tile. If these are set to 0% then there is no edge blending and so usually hard edges are visible where tiles join. Using a value like 5% means that the first and last 5% of the tile's width/height will fade in to the adjacent tile. Using 50% will provide a full blend across the entire tile, as each half will be blending in to an adjacent tile, but high values can make the artwork look blurry. Usually the lowest value that does not produce visible edges is the best setting.

## Tiled Background conditions

For conditions in common to other objects, see Common conditions.

**Is tile randomization enabled**

True if tile randomization is currently enabled. See the *Enable tile randomization* property for more details.

**On image URL loaded**

**On image URL failed to load**

Triggered when *Load image from URL* finishes downloading the image and is ready to display it, or if the load fails.

## Tiled Background actions

For actions common to other objects, see Common actions.

---

**Set image angle**

Change the *Image angle* property, rotating the displayed Tiled Background image by a number of degrees.

---

**Set image X offset**

**Set image Y offset**

Change the *Image offset X* and *Image offset Y* properties, offsetting the displayed tiled background image.

> *Avoid indefinitely increasing the image offset, such as by always adding to it. In some cases a very large image offset can start to exhibit rendering glitches due to precision issues on the GPU. You can avoid this by wrapping the image offset back to 0 after it exceeds the image size.*

---

**Set image X scale**

**Set image Y scale**

Change the *Image scale X* and *Image scale Y* properties, stretching the displayed tiled background image by a percentage on each axis.

---

**Set tile randomization enabled**

Set whether tile randomization is currently enabled. See the *Enable tile randomization* property for more details.

---

**Set angle random**

**Set position random**

**Set tile blend margin**

When tile randomization is enabled, set the relevant tile randomization properties. See the corresponding properties above for more details.

---

**Load image from URL**

Load an image from a given URL. It is not shown until the image has finished downloading, and *On image URL loaded* triggers. Images loaded from different domains are subject to the same cross-domain restrictions as AJAX requests - for more information see the section on cross-domain in the AJAX object. Data URIs can also be passed as an image, e.g. from a canvas snapshot or camera image.

## Tiled Background expressions

For expressions common to other objects, see common expressions.

---

**ImageWidth**

**ImageHeight**

The original dimensions of the tiled background's current image in pixels. Since tiled backgrounds can be extended over large areas causing the normal Width and Height expressions to return different values, these can be used to get the original size of the source image regardless of the object size.

---

**ImageAngle**

Return the *Image angle* property, in degrees.

---

**ImageOffsetX**

**ImageOffsetY**

Return the *Image offset X* and *Image offset Y* properties, in pixels.

---

**ImageScaleX**

**ImageScaleY**

Return the *Image scale X* and *Image scale Y* properties, as a percentage.

---

**TileAngleRandom**

**TileBlendMarginX**

**TileBlendMarginY**

**TileXRandom**

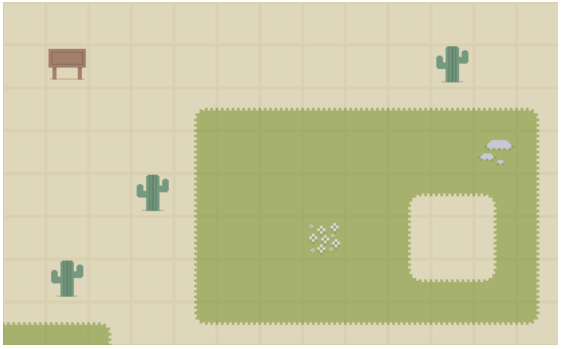
**TileYRandom**

These expressions return the current tile randomization settings. See the corresponding properties above for more details.

# TILEMAP

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

The **Tilemap** object allows tile-based projects to be designed more easily. The object's tilemap can also be edited in the layout view using the Tilemap Bar.



*An example tilemap*

Tilemaps also have significant performance benefits over achieving the same results with other kinds of objects, such as arranging a grid of Sprites. The Tilemap object can optimise collision detection and rendering in a way that scales well even with extremely large Tilemap objects.

For information about editing tilemaps in Construct, see the manual entry for the Tilemap Bar.

A useful behavior to use to move objects around on top of the Tilemap object is the Tile movement behavior.

### Scripting

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

## Tilemap object image

The image used for the Tilemap object is the *tileset*. This is an image that contains every different tile that can be used in the tilemap. The tiles can also can be offset and spaced, but this is not normally necessary. The tileset image appears in the Tilemap Bar after selecting the object, allowing you to choose which tiles to draw with.

## Collisions

When testing for collisions with a Tilemap object, empty (erased) tiles count as not colliding, and by default all other tiles count as colliding. A custom collision polygon can be set, or collisions disabled, for individual tiles by double-clicking a tile in the Tilemap bar. The image editor will appear for the tile, where the collision polygon can be modified, or disabled completely by

unticking the *Use collision* property, or right-clicking on the image and selecting *Toggle collision polygon*.

## Tile IDs

Each tile in the tileset has a zero-based index to identify it. This starts with the top-left tile and increments horizontally in rows. The tile ID can easily be seen by hovering the mouse over a tile in the Tilemap Bar. The tile ID is useful for comparing or setting tiles at runtime with the object's conditions, actions and expressions.

## Tile positions

When using tiles in the object's conditions, actions and expressions, positions are generally given in *tiles* instead of layout co-ordinates. You can convert between tile positions and layout co-ordinates using the *PositionToTileX/Y* and *TileToPositionX/Y* expressions.

## Inappropriate uses of Tilemaps

Don't use tilemaps to display large images where every tile in the tilemap is different. This makes it needlessly less efficient to render the image, since it is rendered one tile at a time when you could have just used a Sprite.

## Tilemap Properties

**Image**

Click the *Edit* link to edit the tileset image from which tiles are drawn.

**Initially visibile**

Choose whether the object is visible or invisible at the start of the layout.

**Tile width**

**Tile height**

The size of tiles in the tilemap, in pixels.

**Tile X offset**

**Tile Y offset**

The offset in pixels of the top-left tile in the tileset image. This is not normally necessary and is provided mainly for compatibility with existing tileset images that have the tiles drawn at an offset.

**Tile X spacing**

**Tile Y spacing**

The spacing in pixels between tiles in the tileset image. This is not normally necessary and is provided mainly for compatibility with existing tileset images that have the tiles drawn apart from each other.

## Tilemap conditions

**Compare tile at**

Compare the tile ID at a position in the tilemap.

**Compare tile state at**

Test whether a tile at a position in the tilemap is flipped or rotated from its normal state.

**On image URL loaded**

**On image URL failed to load**

Triggered when *Load image from URL* finishes downloading the image and is ready to display it, or if the load fails.

**Brush exists**

Check if a tilemap brush exists

## Tilemap actions

**Download**

Invoke a download of the current tilemap data (from the *TilesJSON* expression) as a JSON file. This can be useful for in-game level editors.

**Load**

Load the current tiles from a string of JSON data from a previous use of the *TilesJSON* expression.

**Erase tile**

Erase the tile at a position.

**Erase tile range**

Erase a rectangular area of tiles in the tilemap.

**Erase tile with brush**

Erase tiles using a brush created in the Tilemap Brush Editor. When using an auto tiling brush, the specified position will be modified, along with the surrounding eight positions. If a patch brush is used the whole patch will be erased.

**Erase tile with brush (by name)**

Like **Erase tile with brush**, but allows you to specify the name of the brush using a string.

**Erase tiles with patch brush**

Erase tiles using a patch brush. This is similar to **Erase tile with brush**, with the additional options to specify to *mirror*, *flip* and *rotate* the brush.

**Erase tiles with patch brush (by name)**

Like **Erase tiles with patch brush**, but allows you to specify the name of the brush using a string.

**Set tile**

Set the tile at a position in the tilemap by its tile ID. The tile that is set can also optionally be flipped or rotated.

**Set tile range**

As with *Set tile*, but sets a rectangular area of tiles in the tilemap.

**Set tile state**

Set the tile flipped or rotated state at a position in the tilemap. The tile ID is not changed.

**Set tile state range**

Set the flipped or rotated state for a rectangular area of tiles in the tilemap. None of the tile IDs in the rectangular area are changed.

**Set tile with brush**

Set tiles using a brush created in the Tilemap Brush Editor. When using an auto tiling brush, the specified position will be modified, along with the surrounding eight positions. Using a patch brush modifies all the tiles covered by the patch, the selected position corresponds to the top left of the patch.

**Set tile with brush (by name)**

Like **Set tile with brush**, but allows you to specify the name of the brush using a string.

**Set tile with patch brush**

Set tiles using a patch brush. This is similar to **Set tile with brush**, with the additional options to specify to *mirror*, *flip* and *rotate* the brush. The **Invalid indexes** parameter allows you to choose what should happen when a patch brush doesn't have a valid tile set at a given position, **Erase** deletes the corresponding tile, while **Ignore** leaves the tile unchanged.

**Set tile with patch brush (by name)**

Like **Set tile with patch brush**, but allows you to specify the name of the brush using a string.

**Load image from URL**

Load a new tilemap image from a given URL. It is not used until the image has finished downloading, and *On image URL loaded* triggers. Images loaded from different domains are subject to the same cross-domain restrictions as AJAX requests - for more information see the section on cross-domain in the AJAX object. Data URIs can also be passed as an image, e.g. from a canvas snapshot or camera image.

# Tilemap expressions

**TilesJSON**

Retrieve the tile data in JSON format, which can be loaded in again later using the *Load* action. Note this differs from the built-in *AsJSON* expression, which returns the entire object state (including position, size, behaviors etc), whereas *TilesJSON* returns only the tile data.

**MapDisplayWidth**

**MapDisplayHeight**

The size of the displayed tilemap in tiles. For example if a Tilemap is 320px wide with tiles 32px wide, the display width is 10 as at this size it can fit 10 tiles in to the width.

**PositionToTileX(x)**

**PositionToTileY(y)**

Convert an X or Y layout co-ordinate in to the corresponding tile number in the tilemap. For example, this can be used to get the tile position under the mouse.

**SnapX(x)**

**SnapY(y)**

Snap an X or Y layout co-ordinate to the nearest tile. This also returns a layout co-ordinate, but aligned to the nearest tile in the tilemap.

**TileAt(x, y)**

Return the tile ID at a position in the tilemap. Note the position is given in tiles, not layout co-ordinates. If the tile at the given position is empty (has been erased), the expression returns -1.

---

**TileWidth**

**TileHeight**

The width and height of each tile, as specified in the Tilemap properties.

---

**TileToPositionX(x)**

**TileToPositionY(y)**

Convert a tile position to layout co-ordinates. For example, this can be used to position a Sprite object on top of a given tile.

---

**BrushWidth**

**BrushHeight**

The width and height (in tiles) of a patch brush. Auto tiling brushes always return 0.

# TIMELINE CONTROLLER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/timeline-controller

The **Timeline controller** object allows timelines to be controlled in event sheets.

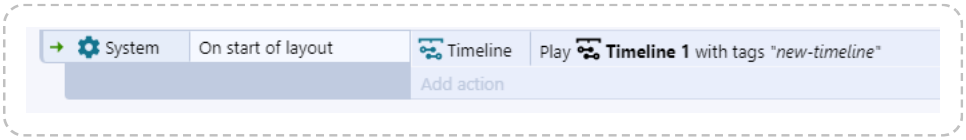
## Tagging

Much like tweens, timelines can be optionally tagged when they are played using one of the **Play** actions. Tags are useful to later control a timeline (or multiple timelines sharing the same tags) with some of the other actions, conditions or expressions.

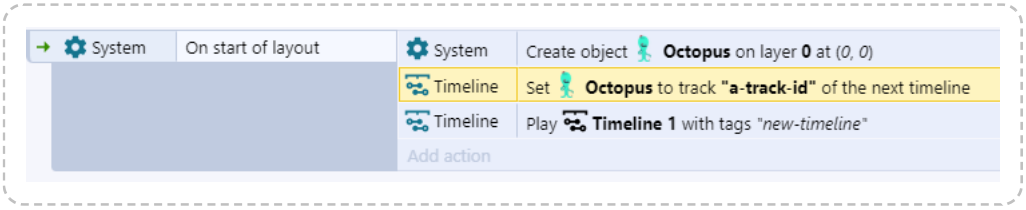
## Setting instances to play

In the simplest case, a timeline will affect the instances that were used to create the timeline in the editor. Using the **Set Instance** action it is possible to use different instances to the ones used in the editor. Below are some short examples to help illustrate how this action works.

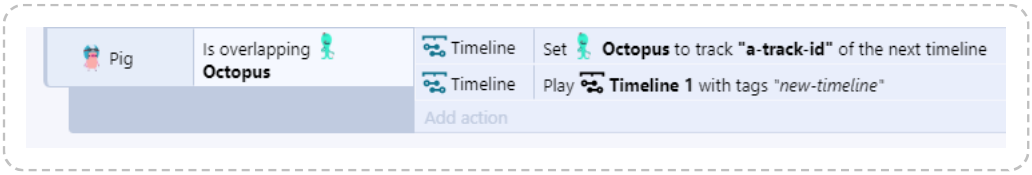
In the below example the timeline plugin **Play** action is used by itself on startup. This plays the timeline affecting the instances used in the editor to create the timeline. The timeline is tagged **"new-timeline"**.




In this example the timeline plugin **Play** action is used together with the system plugin **Create Object** action and the timeline plugin **Set Instance** action. This plays the timeline affecting the newly created instance. The new instance will be used in the track with ID **"a-track-id"** and the timeline is tagged **"new-timeline"**




This example is similar to the last one, but instead of creating a new instance from scratch, the one picked by a collision event is used.



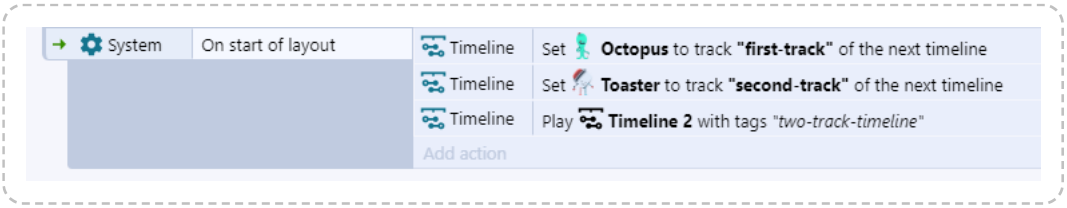
# About picking

When using the **Play** action after one or more **Set Instance** actions, it is possible that one or more similar timelines will start playing. This will depend on the amount of currently picked instances for each given object type.

In the example below a timeline will be played for each group of instances.

This is the preview of the timeline in the editor. It has two different tracks and placeholder instances.

Using the **Set Instance** action we specify to play a timeline for each picked group of instances at the start of the layout




Lastly the preview of the layout shows that two different timelines where created to accommodate for the four instances found at the start of the layout.

# Omitting the track ID

There are a couple of cases in which it is possible to ignore using the **track ID** property of a track as well as omit using the **track ID** parameter of the **Set Instance** action.

## A timeline with only one track

In this case it is possible to not use the **track ID**, as there is only one instance so there is no need to make any choice. The track ID must be empty in both the timeline track and the **Set Instance** action.

## A timeline that will have all of its instances replaced





First instance properties

| Master track | |
|---|---|
| Name | Sprite - 5 |
| Animation mode | Default |
| Result mode | Default |
| Ease | Default |
| Path mode | Default |
| Enabled | ☑ |
| Track ID | |
| **Editor** | |
| Visible | ☑ |
| Locked | ☐ |
| Show UI elements | ☑ |
| More information | Help |

Second instance properties

| Master track | |
|---|---|
| Name | Sprite2 - 6 |
| Animation mode | Default |
| Result mode | Default |
| Ease | Default |
| Path mode | Default |
| Enabled | ☑ |
| Track ID | |
| **Editor** | |
| Visible | ☑ |
| Locked | ☐ |
| Show UI elements | ☑ |
| More information | Help |

Events

| | | | |
|---|---|---|---|
| System | On start of layout | Timeline | Set **octopus** to track "" of the next timeline |
| | | Timeline | Set **toaster** to track "" of the next timeline |
| | | Timeline | Play **Timeline 1** with tags *"new-timeline"* |
| | | | Add action |

In this particular case the **octopus** sprite is used in the first track while the **toster** sprite is used in the second track. Because no track IDs are used, order is assumed to be the same as that defined in the editor.

## Unsetting all instances

Sometimes it might be necessary to clear the state the **Timeline Controller** uses in order to use different instances in a timeline. You might find that some state was previously set but no timeline was played to use it. Such a scenario is likely to cause subsequent uses of **Set instance** and **Play** to produce unexpected results. In those cases use the **Unset instances** action to clear the **Timeline Controller** before attempting to use **Set Instance** and **Play** together again.

## Timeline controller conditions

### Is any playing

**Is any playing**

True if any timeline is playing.

---

**Is playing**

True if a specified timeline is playing, given by its tag.

---

**Is any paused**

True if any timeline is paused.

---

**Is paused**

True if a specified timeline is paused, given by its tag.

---

**On started**

Triggered when a timeline starts playing, given by its tag. Use the **"Type"** parameter to specify when the trigger should take place. *"Any"* will execute the trigger any time a timeline is started, *"Starting"* will execute the trigger only on the initial playback of the timeline and *"Resuming"* will execute the trigger only when the timeline is resumed after being paused.

---

**On any started**

Triggered when any timeline starts playing. Use the **"Type"** parameter to specify when the trigger should take place. *"Any"* will execute the trigger any time a timeline is started, *"Starting"* will execute the trigger only on the initial playback of the timeline and *"Resuming"* will execute the trigger only when the timeline is resumed after being paused.

---

**On finished**

Triggered when a timeline finishes playback, given by its tag.

---

**On any finished**

Triggered when any timeline finishes playback.

---

**On keyframe reached**

Triggered when a master keyframe with certain tags is reached during playback. The keyframe can be identified by whether it matches any of the given tags, or if it has all of the given tags. Separate tags with spaces.

---

**On any keyframe reached**

Triggered when any master keyframe is reached during playback. The *KeyframeTags* expression has a string of the keyframe's *Tags* property.

**On time set**

Triggered when the time of a timeline is set with the **Set Time** action.

## Timeline controller actions

**Play**

Start playing a timeline, with tags to identify this playback.

**Play all**

Play all the timelines in the layout.

**Pause**

Pause a timeline by its tag. Paused timelines can subsequently be resumed.

**Pause all**

Pause all currently playing timelines.

**Resume**

Resume a paused timeline by its tag.

**Resume all**

Resume all paused timelines.

**Stop**

Stop a timeline and reset it to its initial state.

**Stop all**

Stop all timelines, resetting them all to their initial state.

**Set time**

Set the current playback time of a timeline in seconds. **Note:** you can also use a string with a keyframe tag for the *Time* parameter, in which case the time is set to the position of that keyframe. If the timeline is playing when its time is set, playback is stopped.

**Set playback rate**

Set the playback rate of a timeline. 1 is normal speed, 0.5 is half speed, etc. Negative numbers will play in reverse.

**Set instance**

Set an instance to be used for the next timeline playback. The instance can be of a different type to the one used in the editor. The instance will be set to the track with the corresponding track ID. The track ID can also be left empty in which case it uses the first track in the timeline. It can also be used repeatedly with an empty ID to keep setting the tracks in the timeline in sequence. When the timeline is played it will then affect this instance instead of the one used in the editor. Playback can be controlled by using unique tags when playing the timeline.

**Unset instances**

Clears all state associated with calls to **Set instance**.

## Timeline controller expressions

**Time(nameOrTags)**

Retrieve the current time of the first matching timeline by either name or tags.

**Progress(nameOrTags)**

Retrieve the progress of the first matching timeline by either name or tags, returning a value in the range [0, 1].

**TotalTime(nameOrTags)**

The total time of the first matching timeline by either name or tags.

**KeyframeTags**

In a On keyframe reached or On any keyframe reached trigger, a string with the Tags property of the keyframe that was reached.

**KeyframeTime**

Get the time of a keyframe providing a timeline name (or tags) and keyframe tags.

**TimelineName**

In a trigger, a string with the name of the relevant timeline.

**TimelineTags**

In a trigger, a string with the tags of the relevant timeline.

---

**Value(timelineNameOrTags, valueTrackNameOrId)**

Retrieve the value of a value track by specifying a timeline name or tags and a value track name or track id. If no matching value track is found, the expression returns 0.

---

**Ease(easeName, value)**

Return the result of an ease function at a given value in the range 0-1. The ease name can be either a built-in ease, or the name of a custom ease in the project. A list of the names of built-in eases is included below.

## Built-in ease names

These are the names for built-in ease functions that can be used with the TimelineController *Ease* expression.

| | | |
|---|---|---|
| `"linear"` | | |
| `"in-sine"` | `"out-sine"` | `"in-out-sine"` |
| `"in-elastic"` | `"out-elastic"` | `"in-out-elastic"` |
| `"in-back"` | `"out-back"` | `"in-out-back"` |
| `"in-bounce"` | `"out-bounce"` | `"in-out-bounce"` |
| `"in-cubic"` | `"out-cubic"` | `"in-out-cubic"` |
| `"in-quadratic"` | `"out-quadratic"` | `"in-out-quadratic"` |
| `"in-quartic"` | `"out-quartic"` | `"in-out-quartic"` |
| `"in-quintic"` | `"out-quintic"` | `"in-out-quintic"` |
| `"in-circular"` | `"out-circular"` | `"in-out-circular"` |
| `"in-exponential"` | `"out-exponential"` | `"in-out-exponential"` |

# TOUCH

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

The **Touch** object detects input from touchscreen devices like phones and tablets, as well as desktops or laptops with touch-sensitive displays.

The Touch object also provides input from the **accelerometer** (motion) and **inclinometer** (tilt/compass direction) if the device supports them. The user must grant permission for these before they can be used though; see the *Request permission* action for more details. Note also some low-end devices lack the necessary hardware to measure these values.

It is best to support touch input wherever possible. On the web many users browse on mobile devices with touch input only and no mouse or keyboard. If your project does not support touch controls, many users will be unable to play your project at all. For a guide on how to implement on-screen touch controls, see the tutorial on Touch controls.

For a number of examples of using Touch input, search for Touch in the Example Browser.

### Scripting

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

## Multi-touch

The Touch object supports multi-touch. This is most useful with the *On touched* object and *Is touching object conditions*, which can for example detect if multiple on-screen touch controls are being used. This is sufficient for many projects.

For more advanced uses, the *TouchID*, *XForID* and *YForID* expressions can be used to track individual touches for different purposes. Each touch has a unique ID (which is an arbitrary number), and can be accessed using the *TouchID* expression in an event like *On any touch start*. The touch ID can then be stored in a variable and tracked using the *XForID* and *YForID* expressions. Finally comparing the *TouchID* in *On any touch end* indicates when that touch has been released.

## Touch properties

**Use mouse input**

If enabled, mouse clicks will simulate touch events. Clicking and dragging the left mouse button will simulate a touch along where mouse dragged, and single clicks will simulate taps. This can be very useful for testing touch events work properly on a desktop computer with no touch input supported. However, only single-touch input can be simulated with a mouse,

and a mouse is much more precise than a touch, so it is still best to test on a real touchscreen device.

## Touch conditions

**On double-tap**

**On double-tap object**

Triggered when two tap gestures are performed in quick succession in the same location. The *On double-tap object* variant triggers when this gesture is performed over an object.

---

**On hold**

**On hold over object**

Triggered when a touch is held (pressed and not moved) for a short time period. The *On hold over object* variant triggers when this gesture is performed over an object.

---

**On tap**

**On tap object**

Triggered when a tap gesture is performed, which is defined as a touch and release in quick succession in the same location. The *On tap object* variant triggers when this gesture is performed over an object.

---

**Compare acceleration**

Requires motion permission. Compare the current device's motion as its acceleration on each axis in m/s^2 (meters per second per second). The effect of gravity can be included or excluded, but note that some devices only support accelerometer values including the effect of gravity and will always return 0 for acceleration excluding gravity.

---

**Compare orientation**

Requires orientation permission. Compare the device's current orientation, if the device has a supported inclinometer. *Alpha* is the compass direction in degrees. *Beta* is the device front-to-back tilt in degrees (i.e. tilting forwards away from you if holding in front of you). A positive value indicates front tilt and a negative value indicates back tilt. *Gamma* is the device left-to-right tilt in degrees (i.e. twisting if holding in front of you). A positive value indicates right tilt and a negative value indicates left tilt.

---

**On permission granted**

**On permission denied**

Triggered after the *Request permission* action depending on the outcome of the permission request. These can be triggered without an actual permission prompt being shown to the user, such as if a similar prompt was already shown.

### Compare touch speed

Compare the speed of a specific touch (given by its zero-based index). Touch speed is measured in canvas pixels per second, so is not affected by scaling the display.

### Has Nth touch

True if a given touch number is currently in contact with the screen. For example, *Has touch 1* will be true if there are two or more touches currently in contact with the screen (given that it is a zero-based index).

### Is in touch

True if any touch is currently in contact with the screen.

### Is touching object

True if any touch is currently touching a given object.

### On any touch end

Triggered when any touch releases from the screen.

### On any touch start

Triggered upon any touch on the screen.

### On Nth touch end

Triggered when a given touch number releases from the screen. For example, *On touch 1 end* will trigger when releasing the second simultaneous touch (given that it is a zero-based index).

### On Nth touch start

Triggered when a given touch number touches the screen. For example, *On touch 1 start* will trigger upon the second simultaneous touch (given that it is a zero-based index).

### On touched object

Triggered when a given object is touched. The *Type* parameter defaults to *start*, which means it will trigger when a touch starts inside the given object. Changing the *Type* parameter to *end* instead means it will only trigger when a touch is released while inside the given object.

## Touch actions

### Request permission

Request permission to access the device accelerometer (motion) or inclinometer (orientation). The acceleration and orientation expressions may not return any values until permission has been granted by the user. This must be used in a user input event, normally **On touch end** (note that *On touch start* may not work). Some systems merge both requests in to one, so if you request only one permission, the device will grant access to both. *On permission granted* or *On permission denied* is triggered depending on whether the user approved or declined the permission prompt. These can also trigger automatically without a prompt if the user recently approved or declined a similar permission prompt in the same browser session.

## Touch expressions

---

**AccelerationX**

**AccelerationY**

**AccelerationZ**

Requires motion permission. Get the current device's motion as its acceleration on each axis in m/s^2 (meters per second per second) excluding the effect of gravity. The expressions which include gravity (below) are more widely supported; these will return 0 at all times on devices which do not support them.

---

**AccelerationXWithG**

**AccelerationYWithG**

**AccelerationZWithG**

Requires motion permission. Get the current device's motion as its acceleration on each axis in m/s^2 (meters per second per second) including the acceleration caused by gravity, which is about 9.8 m/s^2 down at all times. For example, at rest, the device will report an acceleration downwards corresponding to the force of gravity. These expressions are more commonly supported than the expressions returning acceleration without G (above). However, devices are still not guaranteed to support motion detection, in which case these will return 0 at all times.

---

**CompassHeading**

**Alpha**

**Beta**

**Gamma**

Requires orientation permission. Return the device's orientation if supported, or 0 at all times if not supported. *Alpha* is the compass heading in degrees. In some circumstances this is relative to the compass heading of the device when the app started instead of the true compass heading relative to due North; the *CompassHeading* expression returns the true compass heading, where supported. *Beta* is the device front-to-back tilt in degrees (i.e. tilting forwards away from you if holding in front of you). A positive value indicates front tilt and a

negative value indicates back tilt. *Gamma* is the device left-to-right tilt in degrees (i.e. twisting if holding in front of you). A positive value indicates right tilt and a negative value indicates left tilt.

---

**AbsoluteX**

**AbsoluteY**

**AbsoluteXAt(index)**

**AbsoluteYAt(index)**

**AbsoluteXForID(id)**

**AbsoluteYForID(id)**

Return the current position of a touch over the canvas area. This is (0, 0) at the top left of the canvas and goes up to the window size. It is not affected by any scrolling or scaling in the project. The *At* expressions can return the absolute position of any touch given its zero-based index, and the *ForID* expressions return the position of a touch with a specific ID.

---

**X**

**Y**

**XAt(index)**

**YAt(index)**

**XForID(id)**

**YForID(id)**

Return the current position of a touch in layout co-ordinates. It changes to reflect scrolling and scaling. However, if an individual layer has been scrolled, scaled or rotated, these expressions do not take that in to account - for that case, use the layer versions below. The *At* expressions can return the position of any touch given its zero-based index, and the *ForID* expressions return the position of a touch with a specific ID.

---

**X(layer)**

**Y(layer)**

**XAt(index, layer)**

**YAt(index, layer)**

**XForID(id, layer)**

**YForID(id, layer)**

Return the current position of a touch in layer co-ordinates, with scrolling, scaling and rotation taken in to account for the given layer. The layer can be identified either by a string of its name or its zero-based index (e.g. `Touch.X("HUD")`). The *At* expressions can return the position of any touch on a layer given its zero-based index, and the *ForID* expressions return the position of a touch with a specific ID.

**TouchCount**

Number of touches currently in contact with the device's screen.

**TouchID**

Return the unique ID of a touch (which is an arbitrary number) in an event like *On any touch start* or *On any touch end*.

> *Touch IDs are arbitrary numbers. The only guarantee is that all simultaneous touches have a unique ID. Do not depend on touches having any particular value for their IDs.*

**TouchIndex**

Return the zero-based index of the touch in an event like *On any touch start* or *On any touch end*.

**AngleAt(index)**

**AngleForID(id)**

Get the angle of motion of a specific touch in degrees by its zero-based index or unique ID. A touch must be moving across the device screen for this expression to contain a useful value.

**WidthForID(id)**

**HeightForID(id)**

Return the width and height of a touch with a given ID in pixels. This allows the app to determine the approximate size of the touch area. Note some platforms do not support this and will always return 0 for the touch size.

**PressureForID(id)**

Return the pressure of a touch with a given ID, as a number from 0 (least detectable pressure) to 1 (most detectable pressure). This is useful for devices with pressure-sensitive displays. Note however not all devices have pressure-sensitive displays, and so will always return 0 for the pressure.

**SpeedAt(index)**

**SpeedForID(id)**

Get the speed of a specific touch by its zero-based index or unique ID. Touch speed is measured in canvas pixels per second, so is not affected by scaling the display.

# USER MEDIA

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/user-media

The **User Media** object allows retrieving camera or microphone input from a user. This requires appropriate hardware being installed on the user's system, such as a webcam for a PC, a phone camera on a mobile, or a microphone. Camera snapshots can be taken and transferred in to Sprite or Tiled Background objects, and microphone input can be analysed with the Audio object.

For security reasons, most browsers will prompt the user for permission before allowing user media input, and will display clear notifications that the media device is currently being used, such as a recording icon in the system tray or tab icon.

The User Media object has common features, including the ability to have effects applied for video feeds.

For several examples of what the User Media object can do, search for *User Media* in the Start Page.

## In the layout

The User Media object appears as a rectangle in the layout view, represented by a red cross with its icon in the middle. This represents where the video feed will be displayed in the layout. If you don't need video input, place the User Media object outside the layout.

## User Media conditions

**On media request approved**

Triggered when the user confirms a security prompt after the *Request camera* or *Request microphone* actions, indicating their approval to allow the application to use media input.

**On media request declined**

Triggered when the user cancels a security prompt after the *Request camera* or *Request microphone* actions, indicating they do not approve the application's request to use media input.

**On retrieved media sources**

Triggered after the *Get media sources* action completes, and the list of media sources is available with the *AudioSource* and *CameraSource* expressions.

**On snapshot ready**

Triggered after the *Snapshot* action, when the snapshot is ready to use with the *SnapshotURL* expression.

## User Media actions

### Get media sources

Request a list of media sources that can be used with the *Request camera* or *Request microphone* actions. For example a mobile device may have both front-facing and back-facing cameras, or multiple microphones. Using the media source list allows the specific camera or microphone input to be selected. This does not complete immediately; the media source list is only available after the *On retrieved media sources* trigger fires. The browser also may not support listing the media sources, in which case the trigger will never fire.

### Request camera

Show a security prompt to the user requesting that they give the application permission to use camera input. Either *On media request approved* or *On media request declined* will trigger depending on their decision. If approved, the User Media object in the layout will start displaying a video feed from the user's camera device. The specific camera source to use can be chosen with the *Source* parameter, if media source listing is supported and a media source list has been requested. Otherwise the *Preferred direction* setting can be used to select the user-facing (front/selfie) or environment-facing (back) camera if the device has two, which is common on mobile devices. If the preferred width/height are not zero, the nearest supported resolution that the input device supports will be picked. Microphone input can also be optionally included, with a given microphone source (see *Request microphone*), which is useful in case you want to use Game recorder to record the user's camera and include audio.

### Request microphone

Show a security prompt to the user requesting that they give the application permission to use microphone input. *On media request approved* or *On media request declined* will trigger depending on their decision. The Audio object must also be in the project for this to be useful. A tag is given for the microphone input, and the audio input from the microphone is routed the same way as playing a sound with that tag. This means you can assign effects from the Audio object to the microphone input by adding the effects to the same tag assigned to the microphone. A useful combination is to add an analyser effect then a mute effect to microphone input. This prevents the user hearing their own voice, but allows peak, RMS and spectrum monitoring with the analyser. The specific microphone input to use can be chosen with the *Source* parameter, if media source listing is supported and a media source list has been requested; otherwise the default microphone input is used.

### Snapshot

If the user has approved a camera request and the User Media object is showing a video feed, then snapshots the current frame. The still image is available after *On snapshot ready* triggers in the *SnapshotURL* expression. The image can be loaded in to a Sprite or Tiled Background object using the *Load image from URL* action and passing *SnapshotURL*, or downloaded using the Browser object's *Invoke download* action. This action optionally takes parameters allowing you to specify the compression format, which is useful if you intend to upload or save the image and a smaller file size would be advantageous.

---

**Stop**

Ends any active video feed or microphone input. Media input must be requested again before it can be used.

## User Media expressions

---

**AudioSourceCount**

After *On retrieved media sources* triggers, the number of audio sources available.

---

**AudioSourceLabelAt(index)**

After *On retrieved media sources* triggers, the label of the audio source at the given index. The label is normally the name of the input or recording device, but it may be empty for security reasons (such as if the user has not yet approved a media request).

---

**CameraSourceCount**

After *On retrieved media sources* triggers, the number of camera sources available.

---

**CameraSourceFacingAt(index)**

After *On retrieved media sources* triggers, a string indicating which way a camera source is facing. This can be "user" (the camera is facing the user, such as the front-facing camera on a phone), "environment" (the camera is facing away from the user, such as the back-facing camera on a phone), "left", "right", or empty if unknown or withheld for security reasons.

---

**CameraSourceLabelAt(index)**

After *On retrieved media sources* triggers, the label of the camera source at the given index. The label is normally the name of the input device, but it may be empty for security reasons (such as if the user has not yet approved a media request).

---

**SnapshotURL**

A data URI representing the snapshotted image after a *Snapshot* action once *On snapshot ready* has triggered, otherwise an empty string.

**VideoWidth**

**VideoHeight**

If a video feed is approved and active, this returns the size in pixels of the feed from the device (which may not be the same size as the object in the layout). If no feed is active then 0 is returned.

# VIDEO

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

The **Video** object can play a video inside a project. The video renders into the project canvas itself, allowing other objects to appear on top of it and effects to be applied, unlike with HTML elements.

## Video formats

The supported video formats varies depending on the browser, operating system, and sometimes hardware. The most widely supported video format is probably MP4 H.264 but as a patent-encumbered format it is sometimes not possible to support in open source platforms. As of 2024 the modern and open codec AV1 is increasingly widely supported, but may not be supported on some older devices.

As a result to guarantee that video playback will work on all browsers and on all platforms, it may be necessary to encode your videos in two different formats. The Video plugin allows you to set two sources for a video: a **primary source** which is played if supported, otherwise it falls back to the **secondary source** (which is also only played if supported, but two formats should be enough to achieve universal support). This also allows choosing a modern efficient format which isn't universally supported as a primary source, and an older and less efficient but universally supported format as a secondary source, ensuring optimal playback on modern devices while still supporting older devices.

### Identifying video formats

Video files actually encompass multiple different sub-formats. These include:

- **Container format:** this is usually WebM or MPEG-4 (MP4).
- **Video codec:** the codec used to compress video data, such as VP9, H.264 or AV1.
- **Audio codec:** if the video has an audio track, then this is the codec used to compress audio data, such as Opus or AAC.

Use a third-party tool to identify the container, video codec and audio codec used by a video file. For example VLC media player provides codec information via the menu Tools▶Codec information while playing a video.

## Importing video

Be sure to import video files to the **Videos project folder**. If video files are added in any other project folder, e.g. *Files*, they may be exported to a different folder and fail to load.

Due to the complexities of video compression and the patent-encumbrance of some codecs like H.264, Construct does not provide a video importer like it does with audio. You must encode your video files yourself, and then import them as project files. WebM and its codecs VP8, VP9 and AV1 are open formats and you should be able to find free encoders. However H.264/H.265 encoders may involve a fee.

When publishing a project using video playback to the web, be sure that your server has the correct MIME types set up otherwise video playback may fail after export.

## Playback restrictions

In some cases, video playback cannot begin unless triggered by a user input event. The *Play* action will work in a user input trigger like *On touch start*, but if done outside of that it cannot play right away. To work around this the video plugin will wait until the next touch event to start playing the video. This also applies to autoplaying videos: it will not start until the first touch.

This restriction generally only applies to web browsers. Usually if you publish an app, the restriction is able to be removed because the app can adjust the permissions.

## Video properties

**Primary source**

**Primary format**

The name of the project file to play if the primary format is supported. Make sure *Primary format* matches the actual container and codec used by the specified project file. See *Video formats* above for more details.

**Secondary source**

**Secondary format**

The name of the project file to play, if the primary format is *not* supported. Make sure *Secondary format* matches the actual container and codec used by the specified project file. See *Video formats* above for more details.

**Autoplay**

The autoplay or preload mode. This can be:

- **No:** nothing is done until the video is requested to be played.
- **Preload:** on startup the video will start downloading the video data, but will not start playing it yet. This can allow video playback to start more quickly when requested. Some platforms (e.g. mobile devices on cellular data connections) may ignore this.

- **Yes:** on startup the video will start downloading the video data, and also start playing it as soon as it determines the progress and transfer rate are sufficient to play through to the end without stalling for buffering. Some mobile platforms will not start playing until the first touch event - see *Compatibility* for more information.

---

**Play in background**

If disabled, then switching browser tab, minimising the browser window, switching to a different mobile app, or otherwise hiding the window will pause the video and resume it when switching back. This is intended to avoid annoying the user with continued audio playback when deciding to do something else, and it also helps save battery on mobile devices. However for some types of app it may be desirable to keep playing in the background, in which case enabling this allows continued playback even when in the background.

---

**Initially visibile**

Whether the video is initially visible or invisible. Note that if it is invisible, audio playback may still be heard when playing, so it may be desirable to also mute the video.

## Video conditions

---

**Has ended**

True if the video playback has reached the end of the video and stopped.

---

**Is muted**

True if the audio playback from the video has been muted.

---

**Is paused**

True if the video playback has been paused.

---

**Is playing**

True if the video playback is actively playing.

---

**On playback event**

Triggers when a playback event occurs. This can be one of:

- *Can play:* triggered when enough data is available to play at least a couple of frames, but there may not be enough data to play through to the end.

- *Can play through:* triggered when the browser determines that the load progress and transfer rate are sufficient for playback through to the end without stalling for buffering.

However this is not a guarantee, since the transfer rate could drop or be cut off completely.

- *Ended:* triggered when playback reaches the end of the video.
- *Error:* triggered if an error occurs during video loading, decoding or playback.
- *Started loading:* triggered when the browser begins loading video data.
- *Played:* triggered when playback begins.
- *Paused:* triggered upon pausing the video playback.
- *Stalled:* triggered if the video download rate is too slow to keep up the current playback rate. This will cause the video to pause while it finishes loading the rest of the video, also known as buffering.

## Video actions

**Pause**

Pause the video playback if it is currently playing.

**Play**

Start playing the video. On some platforms this can only happen in a user input event. For more information, see the section on *Compatibility*.

**Set looping**

Set whether the video is looping, so that it restarts from the beginning when it reaches the end.

**Set muted**

Set whether the audio playback from the video is muted (inaudible) or unmuted.

**Set playback rate**

Set how fast the video playback proceeds, as a multiplier of its original speed. That means 1 is the original speed, 2 is twice as fast, 0.5 is half as fast, etc.

**Set playback time**

Set the video playback time to a specific time in seconds (i.e. seek to the given time). Due to the way video encoding technologies work, the video may only be able to seek close to but not exactly on the specified time.

---

**Set source**

Set a different video file to play. As with the object properties, different primary and secondary formats can be specified. Setting the source does not automatically start playing the video; use the *Play* action to start it after changing the source.

---

**Set volume**

Set the volume of the audio playback from the video, in decibels attenuation. 0 is full volume, -10 dB is approximately half as loud, etc. The audio cannot be amplified: positive volume values will be treated as 0.

## Video expressions

---

**Duration**

The video duration in seconds, if the video has loaded enough for this to be determined.

---

**PlaybackRate**

The current playback rate as set by the *Set playback rate* action, as a multiplier of the original rate (e.g. 1 is original speed, 2 is twice as fast, etc).

---

**PlaybackTime**

The current playback time in seconds.

---

**VideoWidth**

**VideoHeight**

The dimensions of the source video, in pixels.

---

**Volume**

The current audio playback volume in dB attenuation.

# VIDEO RECORDER

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/video-recorder

The **Video recorder** object can record videos of your project in real-time. It can record either the main display canvas, the browser tab, the entire screen, or even a different app (where supported).

Note not all browsers or platforms record the same types of audio or video. The codecs used and the resulting file format can vary; use *Is audio/video format supported* conditions to check which are supported. By default the recording actions use *Auto* as the format, which means a supported format will be picked.

> *Screen recording, when supported, often requires a permission prompt from the user for security reasons. Further, to avoid annoying the user, this may only be supported inside a user input event, such as On button clicked, On touch start, etc.*

## Video recorder conditions

**Is audio format supported**

**Is video format supported**

Check if particular audio and video codecs are supported for the recorded video.

**Is recording**

True while currently recording the canvas or screen.

**Is recording supported**

Check if the current browser or platform supports recording. If this is not true, video recording will not work.

**Is screen recording supported**

Check if the *Start screen recording* action is supported. This action can record the user's entire screen, and is not always supported even when the *Start recording* action is supported.

**On recording error**

Triggered if an error occurs while recording a video. The recording is unlikely to be available after an error.

### On recording ready

Triggered after recording finishes, e.g. after the *Stop recording* action, when the video recording has finished encoding and is available to download with the *RecordingURL* expression. This can be downloaded using the Browser object's *Invoke download* action.

## Video recorder actions

### Start recording

Start recording a video of the main display canvas. Note this excludes form controls like buttons and text inputs, because these are HTML elements that "float" above the main display canvas and aren't actually part of the canvas, so are excluded from the recording. (If it is important to record these, some browsers support screen recording and can capture the entire browser tab.) Specific video and audio formats can be chosen, but it is recommended to leave them at *Auto* to ensure a supported format is used, or set to *None* to omit either the video or audio from the recording. A custom framerate can be set but the default 0 indicates the display rate. The quality in kilobits per second (kbps) can also be set, determining the video quality vs. size tradeoff. Use *Stop recording* to end the recording, after which *On recording ready* will trigger so the video can be downloaded.

### Start screen recording

Start recording a video of the user's entire screen. This only works when the *Is screen recording supported* condition is true. For security reasons, browsers will prompt the user before the recording starts. To avoid the prompt annoying users, this action may only be allowed in a user input trigger, e.g. *On button clicked*, *On touch start* etc. Some browsers also provide options in the prompt to record either the browser tab (which will include non-canvas elements like form controls), a different app, or the screen. Specific video and audio formats can be chosen, but it is recommended to leave them at *Auto* to ensure a supported format is used, or set to *None* to omit either the video or audio from the recording. Note that if audio is included, the video can only include audio played by the project itself, unless *System audio* is checked, which allows for recording all audio output from the system including other apps; note however the user may still need to opt-in to including system audio in the permission prompt. The quality in kilobits per second (kbps) can also be set, determining the video quality vs. size tradeoff. Use *Stop recording* to end the recording, after which *On recording ready* will trigger so the video can be downloaded.

### Start User Media recording

Start recording video and/or audio from a User Media object's input. *On media request approved* must trigger before this can be used.

### Stop recording

Stop any active recording. When the video has finished encoding, *On recording ready* will trigger so the video can be downloaded.

# Video recorder expressions

### RecordingURL

In *On recording ready*, a URL that can be used to download the recorded video file. Use the Browser object's *Invoke download* action to download this.

### RecordingType

The content type (also known as MIME type) of the recording that was made, e.g. *"video/webm;codecs=vp9"*. This is useful if you need to know the type of an *Auto* format recording, such as when sharing it.

### RecordingFileExtension

The file extension of the recording that was made including the dot, e.g. *".webm"*. This is useful if you need to know the file extension of an *Auto* format recording, such as when download it.

# WEBSOCKET

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

The **WebSocket** plugin is a simple wrapper around the standardised WebSocket protocol. It allows for low-overhead bi-directional communication in real-time. Since WebSockets are standards-based, it should be compatible with any standards-compliant WebSocket server.

Using the WebSocket plugin requires a WebSocket server. Construct does not provide a server nor can the WebSocket plugin be used to make a server. If you don't already have a WebSocket server set up, you will need to create one yourself using a technology like node.js with WebSocket support. This can be a significant undertaking and require server-side programming knowledge.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in WebSocket API.

## Secure WebSockets and preview mode

Construct runs and previews your projects on a secure (HTTPS) site. Some browsers block connections from secure to insecure sites. This means if you use an insecure WebSocket ( `ws:` rather than `wss:` ) the connection may be blocked in Construct's preview mode.

On today's web, it is best practice to use secure connections for all services. Your site should also host your project on HTTPS, and any WebSockets you use should use secure connections as well. HTTPS hosting is actually required for many features to work, secure WebSockets are more likely to connect successfully, and obviously it makes your content much more secure, so this is a good idea anyway. In some browsers you may be able to temporarily unblock the use of insecure WebSockets by clicking a button in the browser or changing settings, but it is a much better idea to ensure all your services are secure. In future, browsers may block the use of insecure sites and connections entirely.

## WebSockets and multiplayer games

It may be tempting to use WebSockets to design real-time multiplayer games. Despite the fact they communicate in real-time, WebSockets are not a suitable choice for latency-sensitive real-time games. The underlying transport uses reliable transmission, meaning a single dropped packet can hold up all transmission until the packet is retransmitted successfully. For games with demanding real-time requirements, this can cause unplayable levels of latency. It is usually difficult to design around this without changing the transmission mode, which WebSockets do not support. Consider using the specially-designed Multiplayer plugin Paid plans only instead.

On the other hand, WebSockets should be suitable for games without such a demanding real-time requirement, like turn-based games. It should also be useful for application services, like chat rooms. Note this will still require you to create your own WebSocket server.

## WebSocket properties

The WebSocket object has no properties.

## WebSocket conditions

**Is connecting**

True if currently in the process of establishing a connection to a server. The connection is not yet successfully established; there may still be an error.

**Is open**

True if a connection has been successfully established and the communication channel is currently open.

**Is supported**

Use before attempting any connections to verify the current browser or platform supports WebSockets.

**On closed**

Triggered when the connection is closed, either deliberately or due to an error. The *CloseCode* and *CloseReason* expressions can indicate why the connection was closed.

**On error**

Triggered when an error occurs in the WebSocket connection. Use the *ErrorMsg* expression to get the error message text.

**On opened**

Triggered when the connection is successfully established and the communication channel is now open.

**On binary message**

Triggered when a binary message arrives from the server over an open connection. The specified Binary Data object has the contents of the message written to it when the trigger fires, allowing access to the message content.

**On text message**

Triggered when a text message arrives from the server over an open connection. Use the *MessageText* expression to retrieve the content of the message.

## WebSocket actions

### Close

Close any active connection. No more messages can be sent or will be received after closing.

### Connect

Connect to a WebSocket server. WebSocket server addresses typically start with **ws://** for non-secure transmission and **wss://** for secure transmission. Note some network configurations may require secure transmission in order to function correctly.
The *Protocol* parameter may be optionally set to a required sub-protocol (sent with the *Sec-WebSocket-Protocol* header in the WebSocket handshake). If the server does not indicate it supports the chosen sub-protocol, the connection will fail to be established. This can be used to prevent the client connecting to WebSocket servers that do not understand your application's specific messages.

### Send binary

Send the contents of a Binary Data object as a binary message to the server. This is ignored if the connection is not currently open.

### Send text

Send a text string to the server. This is ignored if the connection is not currently open.

## WebSocket expressions

### CloseCode

In the *On closed* trigger, returns the numeric code of the close reason. This can be one of the standard-specified return values, or a user-defined value.

### CloseReason

In the *On closed* trigger, returns a string describing the reason the connection was closed. This is optional and may be empty.

### ErrorMsg

In *On error*, the error message text.

**MessageText**

In *On text message*, the text content of the message just received from the server.

# XML

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

The **XML** plugin can parse and read data from XML documents. It uses **XPath** to access the XML document. XPath is a kind of query language for XML, similar to how SQL is a query language for databases. A description of how to use XPath is out of the scope of this manual; there are some free tutorials you can search for on the web.

Click here to open an example of the XML plugin.

Currently the XML plugin is read-only. You can read data but not change any values in the XML document.

### Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in APIs for parsing and serializing XML.

## Loading an XML document

XML must be loaded as a string with the *Load* action. If you have a small snippet of XML, you can paste it directly in to the action parameter - but note in expressions a double-quote character ( `"` ) must be repeated twice ( `""` ) to avoid ending the string, which can be inconvenient. Instead it is recommended to load an XML project file using the AJAX object. When the AJAX request completes, pass `AJAX.LastData` in to the *Load* action. Then the data from the file can be used.

## XML conditions

**For each node**

Repeat the event once for each node returned by an XPath query. Typically this will be used with a query that returns multiple nodes, e.g. `"/bookstore/book"` to select all "book" nodes under "bookstore". In the *For each node* event, the current node is set to the one currently being iterated. This means relative XPaths, like `"title/text()"`, work relative to the current node (in this case returning the text of the child "title" tag). *For each node* can also be nested, so you can iterate another list relative to the current node.

## XML actions

**Load**

Load an XML document from a string. See 'Loading an XML document' above.

## XML expressions

**NodeCount**

Return the number of nodes returned by an XPath expression. For example, this can count the number of elements with a given name. In a *For each node* event, the XPath is relative to the current node.

**NumberValue**

Return a number from an XPath expression. If multiple values are returned, only the first value is retrieved. In a *For each node* event, the XPath is relative to the current node.

**StringValue**

Return a string from an XPath expression. If multiple values are returned, only the first value is retrieved. In a *For each node* event, the XPath is relative to the current node.

# SYSTEM OBJECT REFERENCE

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

The **System object** is the only object built in to Construct. It provides features to access the runtime engine, and utilities that are useful for most projects. For more information on the System object see Project Structure from the Overview section of the manual.

Since the System object provides a lot of features, it is documented in separate sections for its conditions, actions and expressions.

- System conditions
- System actions
- System expressions

# SYSTEM CONDITIONS

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

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

## Angles

Note angles in Construct start with 0 degrees facing right and increment clockwise.

### Is between angles

True if a given angle is between the two other angles in degrees. The first and second angles must be in clockwise order. That is, *X is between 0 and 45 degrees* is true if the angle X is in a 45 degree area, but *X is between 45 and 0 degrees* is true if X is in the 315 degree area from 45 degrees through 0 degrees. The first angle is inclusive, but the second angle is exclusive, to ensure adjacent ranges are handled correctly.

### Is clockwise from

True if a given angle is clockwise from another angle in degrees, or in other words, if it is 180 degrees or less in a clockwise direction from Angle 2 to Angle 1. Invert to test if anticlockwise instead. For example, 45 degrees is clockwise from 0 degrees, but 0 degrees is anticlockwise from 45 degrees. Angle 1 is the angle to test, and Angle 2 is the reference angle to test whether Angle 1 is clockwise from.

### Is within angle

True if an angle is within a number of degrees of another angle. This is more reliable than testing if an angle exactly equals an angle, e.g. *X within 0.5 degrees of 90 degrees* is probably better than X equals 90 degrees, since there are many cases an angle can be very close to, but not exactly, 90 degrees.

## General

### Compare two values

Compare any two expressions (which can either numbers or text) with each other. They can be compared as *Equal*, *Not equal*, *Less*, *Less or equal*, *Greater* or *Greater or equal*.

### Evaluate expression

True if the given expression is a non-zero number, or a non-empty string. This is useful with boolean expressions like `score < 0 | health < 0`.

---

**Every tick**

A condition which is always true. Used on its own, this has the same effect as running every time it is checked, which is once per tick, hence the name "Every tick". This is about 60 times a second on most devices; see how events work for more information. Adding *Every tick* to an event with other conditions is redundant and has no effect at all.

---

**Is between values**

Test if a number is between two values (greater or equal to a lower value and less or equal to a higher value).

---

**Is group active**

Test if a group of events is active or inactive. The name of the group is used to identify it.

---

**Is number NaN**

Test if a number is equal to *NaN* (Not A Number), a special value returned by calculations which cannot be represented as a real number, such as the square root of -1.

---

**Is value type**

Check if a value is a number or a string.

---

**Object UID exists**

Test if an object exists with the given Unique ID (UID). For more information on UIDs, see instances.

---

**Test regex**

Test if a given string matches a regular expression with flags. This only returns a true or false result, so to make more advanced use of regular expressions, see the *Regex...* system expressions.

# Global & local variables

---

**Compare variable**

Compare the value of a *number* or *text* type event variable (a global variable or local variable in scope). The comparison can be made *Equal*, *Not equal*, *Less*, *Less or equal*, *Greater* or *Greater or equal*.

### Is boolean set

Test if a boolean event variable is set to *true*. To test if it is false, invert the condition.

## Layers

### Compare opacity

Compare the opacity (or semitransparency) of a layer, from 0 (transparent) to 100 (opaque). A layer's opacity cannot be outside this range.

### Layer is empty

Test if a layer currently has zero instances on it. This counts any objects anywhere at all on the layer, so even one instance far outside the viewport will make this condition false.

### Layer is HTML

Test if a layer is currently set to be a HTML layer. For more information see HTML layers.

### Layer is interactive

Test if a layer is currently interactive, allowing its content to respond to mouse and touch input. This can be changed with the *Set layer interactive* action.

### Layer is visible

Test if a layer is currently set to be visible.

> *This check includes parent layers: if any parent layer is invisible, then its sub-layers count as invisible too.*

### Layer name exists

Test if the current layout includes a layer with the given name (case insensitive). This can be useful when using dynamic layers (i.e. the *Add layer* and *Remove layer* actions).

## Layout

### On canvas snapshot

Triggered after the *Snapshot canvas* system action, when the snapshot is ready. It can then be accessed with the *CanvasSnapshot* system expression.

## Loops

Loops can be stopped with the *Stop Loop* system action.

---

**For**

Repeat the event a number of times, using an index variable over a range of values. The index can be retrieved with the *LoopIndex* system expression and passing the name of the loop.

---

**For Each**

**For Each (ordered)**

Repeat the event once per picked instance. This only repeats for instances that have been picked by prior conditions. See how events work for more information on picking. *For Each* is commonly mis-used or used redundantly - actions already apply *for each* instance picked by conditions, so it often is simply not needed. However, if you fully understand how the event system works, it can be used to force the event to apply once per instance where the event system would not normally do that. The 'ordered' variant allows the order that the instances are iterated in to be defined by an expression. For example, ordering by `Sprite.Y` ascending will iterate the top instances on the screen first, moving downwards.

---

**Repeat**

Simply repeat the event a given number of times. This tests any conditions following it on every repeat, and if those conditions are met also runs the actions and any sub-events on every repeat.

---

**While**

Repeat the event until one of the other following conditions in the event becomes false or a *Stop loop* action is used. Be careful not to create infinite loops which will cause the project to hang.

# Memory management

---

**Is loading images**

True while any of the memory management 'Load' actions are in the process of loading images.

---

**On image loading complete**

Triggered when all memory management 'Load' actions that were started have finished loading their images.

# Pick instances

### Pick all

Reset the picked objects back to all of them. Subsequent conditions will pick from all instances again instead of filtering from only those meeting all the conditions so far. See How events work for more information on how instances are picked in events. Useful in subevents to start affecting different instances again.

### Pick by comparison

Pick the individual instances of an object type that meet a comparison. For example, it is possible to pick all instances where `Object.X * 2` is less than `Object.Y + 100`, which is not possible with either the *Compare X* or *Compare Y* conditions.

### Pick by evaluate

Pick the individual instances of an object type where the expression evaluates to a nonzero value. In other words, for each instance if the expression is 0, it is not picked, else it is picked. This is most useful with the comparison and logical operators (see Expressions). For example, it's possible to pick instances using the following expression (where & means "and" and | means "or"): `(Object.X > 100 & Object.Y > 100) | (Object.X < -100 & Object.Y < -100)`

### Pick by highest/lowest

Pick a single instance of a given object type for which the given expression evaluates to the highest or lowest number. This evaluates the provided expression repeatedly for each individual instance of the object type, and uses the resulting values to determine which instance had the highest or lowest result, and then picks that instance. For example picking the instance with the highest expression `Object.X` will pick the single instance furthest to the right. Where there is a tie, an arbitrary single instance is chosen - it will never pick more than once instance.

### Pick last created

Pick the most recently created instance of an object type or family. This is useful with the *Create object (by name)* system action. For example if you know the created object must belong to a family, then you can use *Pick last created* to pick the created instance from the family.

### Pick Nth instance

Pick the instance at a given place in the internal list of picked objects. This is most useful used in sub-events to act on separate instances. For example, in a "Sprite collided with Sprite" event, *Pick 0th instance* and *Pick 1st instance* can be used to act on each instance involved in the collision separately.

If all objects are currently picked, this condition can also be used to pick an object by its index ID (IID).

---

**Pick overlapping point**

Pick all instances of a given object type that are overlapping a point in the layout. The given X and Y position in the layout will be tested against the instance's collision polygons.

---

**Pick random instance**

Pick a random instance from the currently picked objects. In other words, if *Pick random instance* follows another condition, it will pick a random instance from the instances meeting the prior condition. Otherwise it picks a random instance from all the instances.

## Save & Load

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

---

**On load complete**

Triggered after the *Load* system action successfully completes.

---

**On load failed**

Triggered after the *Load* system action fails to complete, usually because the slot has not been saved to yet.

---

**On save complete**

Triggered after the *Save* or *Save to JSON* system actions successfully complete. In both cases, a string of the resulting JSON data can be accessed in this trigger with the *SaveStateJSON* system expression.

---

**On save failed**

Triggered if the *Save* system action fails to complete. This can occur if the browser has reached the limit of its storage quota, or the user has withdrawn permission for the page or app to write to storage.

## Special conditions

---

**Else**

Run if the previous event did not run. Note that this condition does not pick any objects: if it follows an event that picks objects, in the *Else* event all instances revert to picked again. *Else* can only follow normal (non-triggered) events. It can also follow another *Else* event with other conditions to make an "if - else if - else" chain.

### Is in preview

True when running the project from a preview in Construct, and false when running after being exported. Useful to add debug or diagnostic features for previewing only.

### Trigger once while true

Turn an ordinary event (which is tested every tick) in to a trigger. For example, if an event plays a sound when lives equals 0, normally this event runs every tick. This plays about 60 sounds a second and would sound pretty bad. Adding *Trigger once while true* **after** the other conditions makes the event run just once when it first becomes true. This makes the previous example only play a sound once the first time your lives reaches 0. It should be the last condition in an event.

## Start & end

### Is suspended

True if the runtime is currently suspended, i.e. in between *On suspended* and *On resumed*. This normally means the browser/app is in the background and is not currently running.

### On suspended

### On resumed

Triggered when the browser/app suspends and resumes execution. Normally when the app goes in to the background (e.g. minimized, or switched back to the home screen), execution of the app is suspended to conserve system resources and save battery power, triggering *On suspended*. When the app is reopened, *On resumed* triggers and execution resumes.

### On end of layout

Triggered when the layout is ending. This can happen when the project goes to a different layout or when the project closes.

### On loader layout complete

Triggered on a loader layout when the progress reaches 100%. For more information see How to use loader layouts to make custom loading screens.

### On start of layout

Triggered when the layout begins. Use this event to run any actions that need to be done on start-up.

## Time

### Compare time

Compare the time, in seconds, since the project started. For example, events can be set up to run when the time reaches (equals) 10 seconds.

### Every X seconds

Run the event regularly at a given time interval in seconds. This can also be used beneath other conditions to only run the event at a given time interval while the other conditions are true, e.g. "Player is holding spacebar AND every 0.5 seconds: fire laser".

> *Every X seconds does not run more than once per tick. This means very short time periods may not work like you expect. For example many displays use a 60 Hz refresh rate, in which case Every 0.01 seconds will run 60 times a second, not 100 times a second. To avoid such unexpected results, don't use this condition with short time periods - prefer to just use Every tick instead, and if necessary check delta-time (dt) to see how much time passed in the last tick.*

### On signal

Triggered when the *Signal* system action is run with a matching tag (case insensitive).

## Templates

### Template exists

Check if an object type has a template with the provided name.

# 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.

# SYSTEM EXPRESSIONS

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

This section outlines the expressions in the built-in System object in Construct. Many are common mathematical operations, and they can be listed with descriptions in the Expressions dictionary, but they are included here for completeness.

This section does not list the operators or syntax that can be used in expressions - just the expressions specific to the System object. For more general information on how to use expressions in Construct, see Expressions.

## Display

**OriginalViewportWidth**

**OriginalViewportHeight**

Get the original values of the *Viewport size* project property. The value of these expressions does not change when using fullscreen modes like *Scale inner* and *Scale outer* that change the viewport size depending on the available screen area. Note however that the system *Set canvas size* action does modify the values returned by these expressions when in any fullscreen mode other than *Off*, as in this case it has the effect of changing the *Viewport size* project property.

## Layers

In expressions where a layer is required, either its name (as a string) or index (as a number, zero-based) can be entered.

**CanvasToLayerX(layer, x, y)**

**CanvasToLayerY(layer, x, y)**

Calculate the layout co-ordinates underneath a position in canvas CSS co-ordinates for a given layer.

> *The 3D Camera plugin also provides expressions with the same name that work in 3D.*

**LayerToCanvasX(layer, x, y)**

**LayerToCanvasY(layer, x, y)**

Calculate the canvas CSS co-ordinates above a position in layout co-ordinates for a given layer.

> *The 3D Camera plugin also provides expressions with the same name that work in 3D.*

---

**LayerToLayerX(fromLayer, toLayer, x, y)**

**LayerToLayerY(fromLayer, toLayer, x, y)**

Calculate the position on a second layer (*toLayer*) that corresponds to a position given on a first layer (*fromLayer*). This is a shorthand for converting a layer position to canvas CSS co-ordinates, and then back to a position on a different layer.

> *The 3D Camera plugin also provides expressions with the same name that work in 3D.*

---

**LayerAngle(layer)**

Get the angle, in degrees, of a layer.

---

**LayerIndex(layer)**

Get the zero-based index of a layer from its name.

---

**LayerOpacity(layer)**

Get the opacity (or semitransparency) of a layer, from 0 (transparent) to 100 (opaque).

---

**LayerParallaxX(layer)**

**LayerParallaxY(layer)**

Get the current parallax X and Y components of a layer.

---

**LayerScale(layer)**

Get the current scale of the layer, not including the overall layout scale.

---

**LayerScaleRate(layer)**

Get the current scale rate of the layer, which defines how quickly it scales (if at all).

---

**LayerScrollX(layer)**

**LayerScrollY(layer)**

Get the current scroll position of a specific layer. Note this is always the same as the layout scroll position (given by the *ScrollX* and *ScrollY* system expressions) unless a layer was independently scrolled using the *Set layer scroll* system action.

**LayerZElevation(layer)**

Get the current elevation of the layer on the Z axis.

**ViewportBottom(layer)**

**ViewportLeft(layer)**

**ViewportRight(layer)**

**ViewportTop(layer)**

Return the viewport boundaries in layout co-ordinates of a given layer. Not all layers have the same viewport if they are parallaxed, scaled or rotated separately.

> *The 3D Camera plugin also provides viewport expressions that work in 3D.*

**ViewportMidX(layer)**

**ViewportMidY(layer)**

Return the mid-point of the viewport area in layout co-ordinates for a given layer.

**ViewportWidth(layer)**

**ViewportHeight(layer)**

Return the size of the viewport in layout co-ordinates for a given layer.

# Layout

**CanvasSnapshot**

Contains the resulting image from a *Snapshot canvas* action after *On canvas snapshot* has triggered. (Note this expression is not available immediately after the *Snapshot canvas* action - you can only use it after *On canvas snapshot* triggers.) The expression returns a data URI of the image file. This can be loaded in to a Sprite or Tiled Background object via *Load image from URL*, sent to a server or stored locally, or opened with the Browser object in a new tab to save to disk.

**LayoutAngle**

Get the angle, in degrees, of the current layout. This does not include the rotation of individual layers.

**LayoutScale**

Get the current scale of the entire layout set by the *Set layout scale* action. This does not include the scaling of individual layers.

---

**LayoutWidth**

**LayoutHeight**

Get the size of the current layout in pixels.

---

**LayoutName**

Get the name of the current layout.

---

**ScrollX**

**ScrollY**

Get the current position the view is centered on.

---

**VanishingPointX**

**VanishingPointY**

Get the current vanishing point in the layout, where the range 0-100 represents the viewport. For more information refer to the *Vanishing point* layout property.

# Math

These expressions are simply ordinary math functions like you find on calculators. However, note that all functions using an angle take it in **degrees**, not radians. Angles start with 0 degrees facing right and increment clockwise.

**sin(x)**, **cos(x)**, **tan(x)**, **asin(x)**, **acos(x)**, **atan(x)** Trigonometric functions using angles in *degrees*.

**abs(x)** Absolute value of x e.g. `abs(-5)` = 5

**angle(x1, y1, x2, y2)** Calculate angle between two points

**anglelerp(a, b, x)** Linearly interpolate the angle *a* to *b* by *x*. Unlike the standard *lerp*, this takes in to account the cyclical nature of angles.

**anglediff(a1, a2)** Return the smallest difference between two angles

**anglerotate(start, end, step)** Rotate angle *start* towards *end* by the angle *step*, all in degrees. If *start* is less than *step* degrees away from *end*, it returns *end*.

**ceil(x)** Round up x e.g. `ceil(5.1)` = 6

**cosp(a, b, x)** Cosine interpolation of *a* to *b* by *x*. Calculates *(a + b + (a - b) * cos(x * 180°)) / 2*.

**cubic(a, b, c, d, x)** Cubic interpolation through *a*, *b*, *c* and *d* by *x*. Calculates *lerp(qarp(a, b, c, x), qarp(b, c, d, x), x)*.

**distance(x1, y1, x2, y2)** Calculate distance between two points

**exp(x)** Calculate e^x

**floor(x)** Round down x e.g. `floor(5.9)` = 5

**infinity** A floating point number value representing infinity.

**lerp(a, b, x)** Linear interpolation of *a* to *b* by *x*. Calculates *a + x * (b - a)*.

**unlerp(a, b, y)** Reverse linear interpolation: if lerp(a, b, x) = y, then unlerp(a, b, y) = x. Calculates *(y - a) / (b - a)*.

**ln(x)** Log to base e of x.

**log10(x)** Log to base 10 of x.

**max(a, b [, c...])**, **min(a, b [, c...])** Calculate maximum or minimum of the given numbers. Any number of parameters can be used as long as there are at least two.

**pi** The mathematical constant pi (3.14159...)

**qarp(a, b, c, x)** Quadratic interpolation through *a*, *b* and *c* by *x*. Calculates *lerp(lerp(a, b, x), lerp(b, c, x), x)*.

**round(x)** Round x to the nearest whole number e.g. `round(5.6)` = 6

**roundToDp(x, digits)** Round x to a given number of decimal places, e.g. `roundToDp(1.666666, 2)` = 1.67

**sign(x)** Retrieve the sign of *x*: -1 for any negative number, 1 for any positive number, or 0 if *x* is zero.

**sqrt(x)** Calculate square root of x e.g. `sqrt(25)` = 5

---

**getbit(x, n)**

Get the nth bit of x represented as a 32-bit integer. For example `getbit(7, 0)` will get the least significant bit of the number 7 when represented as a 32-bit integer. Returns either 0 or 1.

---

**setbit(x, n, b)**

Set the nth bit of x represented as a 32-bit integer to b (either 0 or 1). The resulting 32-bit integer is returned.

---

**togglebit(x, n)**

Toggle the nth bit of x represented as a 32-bit integer. If that bit is 0, it is set to 1; if it is 1, it is set to 0. The resulting 32-bit integer is returned.

# Memory management

**ImageLoadingProgress**

Return the current loading progress of any memory management *Load* actions that are currently busy, on a scale of 0-1.

## Save & Load

**SaveStateJSON**

In *On save complete* or *On load complete*, returns a string of JSON data representing the savegame data. This can later be loaded using the *Load from JSON* action. For more information see How to make savegames.

## System

**CurrentEventNumber**

**CurrentEventSheetName**

Return the number of the current event being run, and the name of the event sheet it belongs to. These are useful for testing purposes, such as logging the current event number to the browser console.

**ImageMemoryUsage**

Returns the estimated total memory usage, in megabytes, of all the currently-loaded images. Note image memory is sometimes also referred to as "VRAM", but this is not strictly correct since not all devices have video-specific memory. Also remember this expression does not include the memory use of sounds, code, or other non-image resources.

**LoadingProgress**

Return the current load progress on a loader layout. The progress is returned as a number from 0 to 1, e.g. 0.5 for half complete. For more information, see the tutorial how to make a custom loading screen.

**LoopIndex**

Get the index (number of repeats so far) in any currently running loop.

**LoopIndex(name)**

Get the index (number of repeats so far) of the loop with the given name. Useful for getting indices in nested loops.

**ObjectCount**

The total number of objects currently created.

**ProjectFileCount**

**ProjectFileNameAt(index)**

Retrieve the list of project files in this project at the time of preview or export. Files in the Sounds, Music, Videos, Fonts, Icons & Screenshots and Files folders are included. Each entry is a string with the relative path to the file, e.g. `subfolder/mydata.json`.

> *This file list only reflects the state of the project at the time it was previewed or exported. If you export the project and then change some of the files, this list will not update to reflect the change. If you need a list that updates to post-export changes, consider another approach, such as using File System to list folder contents on desktop, or a separate data file to list files in which can also be updated after export.*

**ProjectID**

Return the project ID as it appears in Project Properties.

**ProjectName**

Return the name of the project as it appears in Project Properties.

**ProjectUniqueID**

Return a string of random characters that Construct automatically generated to uniquely identify this specific project.

**ProjectVersion**

Return the version entered in to Project Properties. Note that this is always returned as a string, not a number.

# Text

**find(src, text)**

**findCase(src, text)**

Find the first index within *src* that *text* occurs, else returns -1. *find* is case-insensitive, and *findCase* is case-sensitive.

**left(text, count)**

Return the first *count* characters of *text*.

**len(text)**

Return the number of characters in *text*.

---

**lowercase(text)**

Convert the given text to all lowercase.

---

**mid(text, index, count)**

Return the *count* characters starting from *index* in *text*. Note *count* can be set to -1 to return characters from *index* to the end of the string.

---

**newline**

A string containing a line break. Use to insert line breaks in to strings, e.g. `"Hello" & newline & "World"`

---

**RegexMatchAt(String, Regex, Flags, Index)**

Process the regular expression *Regex* on *String* with *Flags*, and in the list of results, return the entry at *Index*.

---

**RegexMatchCount(String, Regex, Flags)**

Process the regular expression *Regex* on *String* with *Flags*, and return the number of entries in the list of results.

---

**RegexReplace(String, Regex, Flags, Replace)**

In *String* substitute matches for the regular expression *Regex* (with *Flags*) with the string *Replace*. The replacement string can contain the following special characters: **$$** (inserts a $), **$&** (inserts the matched substring), **$`** (inserts the portion of the string that precedes the matched substring), or **$'** (inserts the portion of the string that follows the matched substring).

---

**RegexSearch(String, Regex, Flags)**

Return the index of the first character in *String* where a match for *Regex* with *Flags* could be found.

---

**replace(src, find, rep)**

Find all occurrences of *find* in *src* and replace them with *rep*.

---

**right(text, count)**

Return the last *count* characters of *text*.

### StringSub(text, sub1 [, sub2...])

Substitute placeholders of the form `{n}` in the given string. This expression accepts a variable number of parameters. For example `StringSub("Hello {0}!", "Sam")` returns *Hello Sam!*, as the placeholder `{0}` is replaced with the first provided additional parameter. Further parameters can be provided by increasing the number in the placeholder, e.g. `StringSub("Hi {0}, your score is {1}!", "Sam", 100)` returns *Hi Sam, your score is 100!*. Note the substitutions can be either strings or numbers. If there are multiple occurrences of the same placeholder, they are all replaced. If a placeholder is used for which a parameter is not provided, then it is left as-is, e.g. `StringSub("Hi {0}, your score is {1}!", "Sam")` returns *Hi Sam, your score is {1}!* as there is only one substitution provided.

### tokenat(src, index, separator)

Return the Nth token from *src*, splitting the string by *separator*. For example, `tokenat("apples|oranges|bananas", 1, "|")` returns *oranges*.

> *Use the Array object's Split string action for more flexibility. To better handle more complex data, use a more robust data format like JSON.*

### tokencount(src, separator)

Count how many tokens occur in *src* using *separator*. For example, `tokencount("apples|oranges|bananas", "|")` returns 3.

### trim(src)

Return *src* with all whitespace (spaces, tabs etc.) removed from the beginning and end of the string.

### uppercase(text)

Convert the given text to all uppercase.

### URLEncode(str)

### URLDecode(str)

Convert to and from a string in a format suitable for including in a URL or POST data.

### zeropad(number, digits)

Pad *number* out to a certain number of *digits* by adding zeroes in front of the number, then returning the result as a string. For example, `zeropad(45, 5)` returns the string "00045".

## Time

### CPUUtilisation

The percentage of the last second that was spent in logic, such as running events or processing behaviors. This is for performance measurements. Note on most devices the rendering happens on the separate GPU and so is not counted by this measurement; for that *fps* or *GPUUtilisation* is a better measure. Also note this measurement is based on timers so should be treated as an approximation, and it only measures time on the main thread.

> *This measurement can be unreliable, especially when the system is largely idle. Most modern devices deliberately slow down the CPU if not fully loaded in order to save power. This means work takes longer to get done, and this expression will misleadingly return a higher measurement, since it's based on timing how long the work takes. It will generally only be reliable in the device's maximum performance mode, i.e. under full load.*

### dt

Delta-time in seconds. See Delta-time and framerate independence.

### fps

How many frames per second (FPS) the project is rendering. The most common display refresh rate is 60 Hz, so typically an efficiently designed project will render at 60 FPS. Note however if nothing is changing on-screen, then nothing is rendered, and so the FPS measurement may fall to 0 or display a lower result; this does not indicate poor performance, only that fewer frames are necessary to render. The *TicksPerSecond* expression of the Platform Info object indicates how frequently the engine is stepping, which may be different to the frames rendered per second.

### GPUUtilisation

The percentage of the last second that was spent rendering graphics. This represents how busy the graphics processing unit (GPU) is, which is useful for performance measurements. This measurement is based on timers so should be treated as an approximation. The GPU utilisation is only affected by the amount of rendering work to be done, such as the number of objects visible on-screen, and also increases if the window size is larger. Note this measurement is only available on certain systems. If it is not supported, it will return NaN (a special value representing Not A Number) to indicate there is no value available.

> *See the note under CPUUtilisation about possibly unreliable measurements in some circumstances. This also applies to the GPU and can affect this measurement too.*

### tickcount

The number of ticks that have run since the project started.

### time

The number of seconds since the project started, taking in to account the time scale.

### timescale

The current time scale.

### wallclockdt

This is the same as the *dt* expression, but is not affected by the time scale. This can be useful to still obtain the real-world delta-time value when the project is paused by setting the time scale to 0, which causes *dt* to also become 0.

### wallclocktime

The number of seconds since the project started, not taking in to account the time scale (i.e. the real-world time).

> *Unlike other time expressions, wallclocktime can update during the same tick, such as within a long loop. This means it can be used for things like performance measurements, or doing work for a fixed period of time.*

## Values

### choose(a, b [, c...])

Choose one of the given parameters at random. E.g. `choose(1, 3, 9, 20)` randomly picks one of the four numbers and returns that. This also works with strings, e.g. `choose("Hello", "Goodbye")` returns either *Hello* or *Goodbye*. Any number of parameters can be used as long as there are at least two.

### chooseindex(index, value0[, ...])

Choose one of the given parameters by a zero-based index. For example:

- `chooseindex(0, "foo", "bar", "baz")` returns *"foo"*
- `chooseindex(1, "foo", "bar", "baz")` returns *"bar"*
- `chooseindex(2, "foo", "bar", "baz")` returns *"baz"*

Any number of parameters can be included after the index (but there must be at least one). If the index is out of range, it returns either the first or last value, e.g. in the above example an index of -1 will still return *"foo"*.

**clamp(x, lower, upper)**

Return *lower* if *x* is less than *lower*, *upper* if *x* is greater than *upper*, else return *x*.

**float(x)**

Convert the integer or text *x* to a float (fractional number). If *x* is text, non-numeric characters are allowed after the number, but not before. For example `float("3.1xx")` returns *3.1*, but `float("xx3.1")` returns 0.

**int(x)**

Convert the float or text *x* to an integer (whole number). If *x* is text, non-numeric characters are allowed after the number, but not before. For example `int("33xx")` returns 33, but `int("xx33")` returns 0.

**random(x)**

Generate a random float from 0 to *x*, not including *x*. E.g. `random(4)` can generate 0, 2.5, 3.29293, but not 4. Use `floor(random(4))` to generate just the whole numbers 0, 1, 2, 3.

**random(a, b)**

Generate a random float between *a* and *b*, including a but not including *b*.

**rgbEx(r, g, b)**

**rgbEx255(r, g, b)**

**rgba(r, g, b, a)**

**rgba255(r, g, b, a)**

Generate a single number containing a color with the given red, green, blue and optionally alpha components. *rgbEx* and *rgba* use components in the range 0-100, whereas *rgbEx255* and *rgba255* use components in the range 0-255. When an alpha is not provided, the resulting color is opaque. These are useful for conditions or actions taking a color parameter.

**str(x)**

Convert the integer or float *x* to a string. Generally this is not necessary since strings can be built using the & operator, e.g. `"Your score is " & score`

# SCRIPTING OVERVIEW

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/overview

Construct supports writing **JavaScript** or **TypeScript** code in the place of actions and events, as well as in separate JavaScript or TypeScript files. This is a great way to learn to code, as well as taking advantage of the full power of modern industry-standard programming languages in your projects.

## JavaScript is not Java

Note **Java** is a completely different programming language to **JavaScript**. This is a common point of confusion. Try not to get the two mixed up. For example if you search the web for help on JavaScript, be sure to specifically search for *JavaScript* and not *Java*, otherwise you will get results for the wrong programming language!

## JavaScript vs. TypeScript

TypeScript is essentially an extension of JavaScript that adds static type features. Whereas JavaScript is a dynamic language that allows variables and properties to hold any data and change at any time, TypeScript uses type annotations to more strictly validate code. To illustrate the difference, here is a sample JavaScript function:

```
function add(a, b)
{
        return a + b;
}
```

This function is intended to add two numbers, but can actually be called with strings, objects, or anything else - and it may then produce unintended results. The same function in TypeScript looks like this:

```
function add(a: number, b: number)
{
        return a + b;
}
```

The only difference is the addition of `: number` in the parameters, which specifies they must be numbers. Then TypeScript will enforce this restriction, helping to avoid any mistakes.

Browsers can only directly run JavaScript, so TypeScript code is converted (or "compiled") to JavaScript essentially just by deleting the type annotations. This illustrates how closely related the languages are - TypeScript is essentially adding a layer of additional checks on top of JavaScript.

Some of the advantages and disadvantages of JavaScript vs. TypeScript are:

- **Ease of learning**: TypeScript may take a little longer to learn as you have to also learn about type annotations.
- **Coding accuracy**: TypeScript's type checks can catch lots of kinds of errors for you before you even try to run your project. JavaScript on the other hand allows many kinds of incorrect code to execute, and you may only be able to find out the problem from either the browser console or using a debugger.
- **Coding flexibility**: JavaScript allows writing a wider range of code styles, allowing things that TypeScript would not normally allow due to its type checking (but as noted previously that also includes incorrect code).
- **Autocomplete**: TypeScript allows providing an exact autocomplete feature. As JavaScript is a dynamic language it is not possible to always accurately identify which properties and methods are available, and so often the autocomplete list will include guesses which are not actually valid, or miss out possible entries.
- Other developer tools: TypeScript also allows other tools like **Go to definition** and **Find references** to work accurately thanks to its type system, whereas much like with autocomplete in JavaScript these may resort to guesses or miss out some results.

If you are new to coding and wondering which to learn, we would suggest to consider starting with TypeScript. Despite the fact there is a little more to learn, its extra checks mean it catches lots of kinds of beginner mistakes, and the precise autocomplete helps you more quickly identify what kind of code you can write.

## Terminology

JavaScript is often shortened to *JS* and normally uses files with the extension *.js*. TypeScript is often shortened to *TS* and normally uses files with the extension *.ts*. The terms *script* and *code* refer to the same thing: the programming code that you or someone else has written. Similarly *scripting*, *coding* and *programming* are used interchangeably, and all refer to the process of writing code in a programming language. In the context of Construct these terms refer to JavaScript or TypeScript code, as those are the two languages it supports.

## Quick start guide

If you already know JavaScript, check out the Construct for JavaScript developers quick start guide for a summary of what you might need to know when working in Construct. If you already know TypeScript, you can also check out the Construct for TypeScript developers quick start guide.

If you'd also prefer to use an external code editor like VS Code, see Using an external editor.

## Learning JavaScript

If you're interested in learning JavaScript from scratch using Construct, see our 13-part tutorial series Learn JavaScript in Construct.

JavaScript is one of the most popular languages in the world, and is widely used in the technology industry, especially in web development. As a result there are many more materials available across the web for teaching the JavaScript language. The tutorial series linked to above includes lots of links towards the end where you can continue learning more about JavaScript.

This manual section focuses on the unique details of using JavaScript within Construct. We recommend the MDN web docs as a good reference to use for the JavaScript language itself. This manual links to it when referring to specific parts of the JavaScript programming language so you can learn more about it.

### Introductory video

For a video introduction to using JavaScript in Construct, see the video below.



Subscribe to Construct videos now

## Learning TypeScript

If you're interested in learning TypeScript from scratch using Construct, see our 14-part tutorial series Learn TypeScript in Construct.

As with JavaScript, TypeScript is one of the most popular languages in the world, and is widely used in the technology industry, especially in web development. The tutorial series linked to above also includes lots of links towards the end where you can continue learning more about TypeScript.

The official TypeScript website provides lots of useful guides and information, including:

- TypeScript for the New Programmer
- TypeScript for JavaScript Programmers
- The TypeScript Handbook

See also the guide Using TypeScript in Construct.

## Examples

In the Example Browser you can filter for JavaScript or TypeScript examples under the *Coding* header. You'll find a variety of example projects making use of JavaScript/TypeScript coding in Construct, including many examples that come with both JavaScript and TypeScript variants. These cover everything from simple beginner examples to sophisticated fully-coded games.

The reference section of the manual also includes some code samples in places, with snippets demonstrating how to use specific features. These will usually need to be pasted in to a project with appropriately named objects and behaviors to work correctly, or otherwise edited as necessary for use in your own projects.

# CODING IN CONSTRUCT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/coding-in-construct

This section covers some details about how JavaScript and TypeScript code is run in Construct.

## Expressions aren't exposed to script

Construct's event sheets fundamentally work differently to code. Consequently object expressions that you may enter in to conditions and actions are not directly accessible from script. Refer to the scripting reference for the methods and properties that are directly accessible from scripts. You can also still indirectly access things that are not directly exposed to scripts by passing values between scripts and event sheets - for an example of that see integrating events with script.

## Case sensitivity

Many features in Construct are case insensitive, such as "Player" being considered equivalent to "player". However like most programming languages, JavaScript and TypeScript are case sensitive. Therefore when referring to objects, variables and other items in your project from code, you must use the same case as it was written with in Construct.

## Modules

All scripts in Construct are Modules (also known as ES Modules, or ESM). This allows use of the `import` and `export` syntax. It also has a few differences to legacy "classic" mode scripts, notably:

1. Each module (script file) has its own top-level scope. This means top-level declarations are not globally accessible. Instead prefer to use imports and exports, or if you have to, explicitly use properties of `globalThis` to make them global.
2. Modules run code in strict mode. This eliminates common problems such as silent errors, typos accidentally creating variables, and fixes some aspects of the language considered mistakes. In particular it helps avoid confusing problems that beginners often run in to. Since all code is already run in strict mode, there is no need to add the `"use strict"` directive to any of your code.

See the Imports & exports example for a demonstration of using modules in Construct.

## Worker mode

When the *Use worker* project property is *Yes*, the Construct runtime is hosted in a Web Worker instead of in the DOM, and renders using an OffscreenCanvas. This is generally good for

performance since the runtime can run independently of the browser main thread, which can be blocked by browser tasks like layout. However Web Workers have a reduced set of APIs available. Notably there are no `window` or `document` objects available, and so the DOM cannot be directly accessed. However there are many other APIs still available, such as fetch (and the older XMLHttpRequest), IndexedDB, WebSocket and others - see Functions and classes available to workers for more details.

The default mode for *Use worker* is *Auto*. This means Construct automatically decides whether to run your project in a Web Worker or the DOM. Currently this means it will run in a worker unless your project uses any JavaScript code, in which case it will run on the DOM on the assumption your code will need to make use of DOM APIs. Note however that this loses the performance benefits of worker mode. If your JavaScript code can run in a Web Worker, you can change the setting back to *Yes* and gain the performance benefits.

### Worker mode with TypeScript

When using TypeScript, Construct will update type definitions to match the environment. In other words if your project runs in a worker, it will use type definitions for a worker, so accessing things like `document` will be an error. Otherwise if the project runs in the DOM then it uses type definitions for the DOM, permitting the use of such DOM-specific APIs.

### Accessing global scope

Sometimes the `window` object is used to refer to global scope, such as with `window.myGlobal = 1;`. Note however that the `window` object is not available in worker mode, so this won't work there.

The standard way of accessing the global scope is with `globalThis`, such as with `globalThis.myGlobal = 1;`. Since this is available in both the DOM and Web Workers, this code will work everywhere.

## Browser/platform support

When using the very latest JavaScript features, you may need to check which browsers support it. For example if you use a JavaScript feature that not all browsers or platforms support, you may get an error trying to run your game on that platform. We recommend the MDN web docs as a good place to check compatibility.

This also applies when using TypeScript. Since TypeScript essentially just deletes type annotations and then runs the code as JavaScript, you still need to be aware of browser support for any newer JavaScript features you use.

You can also rely on modern support for JavaScript being available, because Construct only supports relatively modern browsers. For example the C3 runtime does not support Internet Explorer so you do not need to worry about supporting it at all. Note many older code examples across the web use an older style designed to support defunct browsers. For example many old code examples use `var` to declare variables, whereas in modern JavaScript `let` and `const`

are preferred. Bear this in mind when looking at other code examples, and note there may be modern features that can considerably simplify the code. It can be a good idea to update any code snippets you use in your projects to a modern style.

## Undocumented features

> *Do not use undocumented features in your JavaScript code.*

If you explore the functions and variables available in a debugger, you may find undocumented APIs specific to the Construct engine. The only reason these can be found is because the way JavaScript works makes it difficult to hide them. **Do not use any such undocumented features in your JavaScript code.** These are internal details of the Construct engine and can change at any time, and such changes can easily break your code. **No support will be provided for undocumented APIs**, even if engine changes break your code. Responsible developers know to only use documented and officially supported APIs. These can be found in the scripting reference. If new engine functionality is essential to you, please file a feature request.

Note this only applies to the Construct engine - all other browser APIs are of course available for use.

When using TypeScript, type definitions are only provided for documented features, so it will prevent you using undocumented features. This can be circumvented with the `any` type, but naturally we would strongly advise against that.

# TYPESCRIPT IN CONSTRUCT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/typescript-construct

There are a few details specific to using TypeScript in Construct. This section outlines what you need to know.

## TypeScript in the Construct editor

If you are writing TypeScript code in Construct itself, you should only have TypeScript files (.ts) in your project. Construct will automatically compile these to JavaScript (.js) files for you behind the scenes, so you don't need to worry about having .js files in your project. The only times you'll see the JavaScript output of your TypeScript code is when debugging or after exporting your project.

If you do have both a .ts and a .js file with the same name in your project, Construct gives priority to the .js file. In other words it will ignore your .ts file and just use the contents of the .js file.

## Using an external editor

If you use an external editor to write TypeScript code, you most likely want to keep the .ts files external, and only import the .js output files to your Construct project. This is because your external code editor will use its own version of TypeScript. Using two different versions of TypeScript is likely to cause awkward compatibility problems, so the best workflow is to let your external editor compile TypeScript to JavaScript, and then just use the .js files in Construct so it uses them as-is. You can set your code editor to automatically compile TypeScript to JavaScript when saving a file, and set Construct to auto-reload the script files on preview, so then you can have an easy workflow where saving a TypeScript file will automatically reflect the changes in the next Construct preview.

Using an external editor also allows you to use a newer or different version of TypeScript to the one built in to Construct. Construct's built-in TypeScript version is logged to the browser console the first time a code editor is opened.

## Suggested workflows

In short the two suggested workflows for TypeScript coding are:

1. Code TypeScript in the Construct editor: only have .ts files in your Construct project, and no .js files.
2. Code TypeScript using an external editor: don't have .ts files in your Construct project, only .js files.

## TypeScript definition files

You can import .d.ts files - aka TypeScript definition files - to your Construct project. This will allow TypeScript to use the type definitions in those files. As per the design of TypeScript, definition files are not compiled to JavaScript and have no effect on how code is run - they only define types to be used by TypeScript's code validator.

## Converting a project to TypeScript

You can switch an existing project using JavaScript code to TypeScript by right-clicking the *Scripts* folder in the Project Bar and choosing TypeScript▶Switch project to TypeScript. This will automatically switch all JavaScript code in event sheets to use the TypeScript language instead, and rename all .js project files to .ts. Note however it does not change any code, as it is not possible to automatically add type annotations. Therefore you will still need to go through your code and add type annotations until there are no more TypeScript errors (see the next section for advice on that). However this option should still save you some time if you want to switch your project's language.

Similarly a project using TypeScript can be switched back to JavaScript. Again you'll need to then remove type annotations for it to be valid JavaScript code.

### Converting a folder project to TypeScript

Construct has special handling for switching a folder-based project to TypeScript. This applies when there are both .js and .ts files in the project folder, but only .js files in the Construct project (i.e. our recommended workflow for using an external editor). In this case choosing the *Switch project to TypeScript* option will instead replace .js files in the Construct project with the corresponding .ts files from the project folder. This provides a handy way to switch a project from using an external editor back to using Construct's built-in TypeScript support.

## Adding types to existing JavaScript code

If you convert a JavaScript project to TypeScript, you'll probably see a lot of TypeScript errors. That's because in some places TypeScript requires type annotations, and leaving them out is marked as an error. You'll need to go through all the code adding type annotations until there are no more TypeScript errors reported. This section has some advice on how to do that. Note this is not an exhaustive list of all changes you'll need to make, but the common changes that you're likely to need to make are covered, as well as some advice specific to Construct.

As a starting example, the default *main.js* code file includes this function:

```
async function OnBeforeProjectStart(runtime)
{
	// ...
}
```

TypeScript will identify the `runtime` parameter as an error, because it does not have a required type annotation. Construct's runtime interface type is IRuntime. So the parameter must be marked as having the type `IRuntime`, as shown below.

```
async function OnBeforeProjectStart(runtime: IRuntime)
{
	// ...
}
```

In many cases TypeScript can automatically infer types from existing code (see Type inference) so a type annotation is not always needed in every place in the code. However there are several places like function parameters that will need type annotations to be added. Familiarity with Construct's built-in class names is useful as they are sometimes needed as types, such as `IRuntime` above - all the class names are included in the scripting reference section of the manual.

## Instance types

When using TypeScript, Construct generates a special class representing an instance for every object type and family in the project. This includes type definitions for things like the instance variables, behaviors and effects specific to that object. These classes are all in the `InstanceType` namespace with the name of the object. For example `InstanceType.Player` is the type for an instance of the *Player* object type.

> *Construct's naming rules are more permissive than JavaScript/TypeScript's naming rules, so in some cases the generated class name may be different to the object type name. The best approach is to make sure all your object names are valid JavaScript identifiers, which generally means starting with an alphabetic letter.*

## Optional types

Many of Construct methods, such as `objectType.getFirstInstance()`, can return `null` (in this case, if no instances exist at all). This means the method's return type can optionally be `null`. TypeScript will show an error if you try to use something that could be `null`. An example of this is shown below.

```
const playerInst = runtime.objects.Player.getFirstInstance();
playerInst.x += 10; // Error: 'playerInst' is possibly 'null'
```

If you know for sure that there is always an instance of the object and so it will never return `null`, you can add an exclamation mark `!` after the expression to tell TypeScript you know it won't be `null`.

```
// Note '!' added to line below
const playerInst = runtime.objects.Player.getFirstInstance()!;
playerInst.x += 10; // OK
```

This is known as the non-null assertion operator.

## Subclassing

If your project uses subclassing to customize the instance class for Construct objects, then you'll find some Construct APIs still return instances of the default type. For example instances of a Globin sprite object will be typed as the default *InstanceType.Goblin* instead of a custom *GoblinInstance* class, e.g.:

```
const inst = runtime.objects.Goblin.getFirstInstance()!;
// 'inst' is of type InstanceType.Goblin - so it won't have any of
// the properties or methods of the custom GoblinInstance class
```

To solve this, the methods available on `IObjectType` are in fact generic, so you can make them return the correct type. This means adding the `<Type>` generic syntax like so:

```
const inst = runtime.objects.Goblin.getFirstInstance<GoblinInstance>()!;
// 'inst' is now of type GoblinInstance and so can use the properties
// and methods of the custom class
```

*Note that you may then have to import the module that defines `GoblinInstance`, as otherwise TypeScript doesn't know about its type.*

## Object literals

Sometimes it's useful to write an object literal, which the *Spell Caster Code* example does for sharing global variables from a module, similar to this:

```
const Globals = {
        score: 0,
        playerInstance: null
};
```

In this case, TypeScript will correctly infer the type of `score` as *number*, but it will infer the type of `playerInstance` as *null*. The type *null* means the variable can only ever have the value `null` and assigning anything else to it will be an error! Due to the syntax of object literals, which already use a colon, it's not always obvious at first how to add a specific type to this property. The solution is to use the generic-style syntax `<Type>` like so:

```
const Globals = {
        score: 0,
        playerInstance: <InstanceType.Player | null> null
};
```

## Class properties

Commonly in JavaScript, class properties are added in the constructor.

```
class MyClass {
        constructor()
        {
                this.prop1 = "hello";
                this.prop2 = 123;
        }
}
```

TypeScript does not infer the class properties from the constructor, so it will show an error for `prop1` and `prop2`, e.g. *Property 'prop1' does not exist on type 'MyClass'*. Instead you must declare the class properties, and their types, at the class-level like so:

```
class MyClass {

        prop1: string;
        prop2: number;

        constructor()
        {
                this.prop1 = "hello";
                this.prop2 = 123;
        }
}
```

Note JavaScript does allow class property definitions in a similar way, with the feature known as class fields. Using this feature in JavaScript should make it easier to switch to TypeScript, as you can then just add type annotations to the existing class fields.

## Imports

Typically when importing other JavaScript files in your project, you'd write a relative import for another .js file like so:

```
import Globals from "./globals.js";
```

How do you write the import for TypeScript? The answer is: exactly the same way! Even though the import ends with .js, TypeScript knows the file is really generated from the .ts file, and so everything just works. Don't try to change it, as otherwise it won't work after it's compiled to JavaScript.

## Example

The Spell Caster TypeScript example demonstrates the Spell Caster JavaScript example but updated to use TypeScript. In the Example Browser you can find a number of examples that have both JavaScript and TypeScript variants, which allows you to compare them and identify what changes were made. You can also try practicing by taking a JavaScript example, using the

*Switch project to TypeScript* option, and adding type annotations; if you get stuck, check the TypeScript version to see what the solution is. Usually if a non-obvious change was needed for TypeScript, the project will contain a special comment explaining what was done.

# SCRIPTS IN EVENT SHEETS

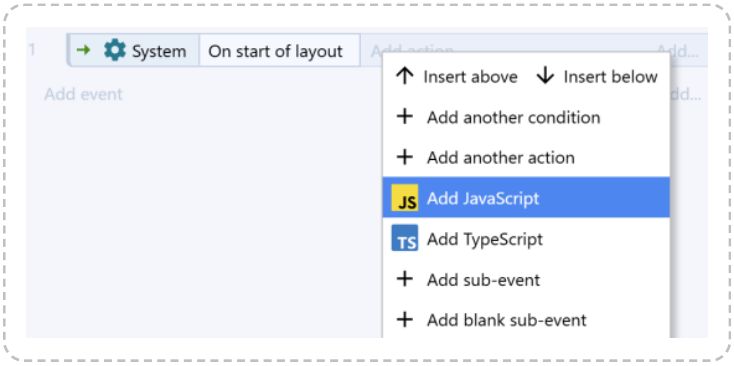
**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/scripts-in-event-sheets

A great way to get started with JavaScript or TypeScript is writing snippets of code in the event sheet. You can write some code to run in the place of an action or an event block.

> *In the context of JavaScript/TypeScript, an event commonly refers to a callback function that runs when something happens, similar to a trigger in Construct's event sheets. However the term event is also used in Construct to refer to a block in an event sheet. To clarify the difference the scripting section of the manual refers to event blocks or event sheets to refer to Construct's event system.*

## Scripts in actions

JavaScript or TypeScript code can be added as an action. The code runs whenever the action is run, i.e. after all the event's conditions are met, and all previous actions have run. To add some code as an action, use the *Add...* menu to the right of the *Add action* link, and select *Add JavaScript* or *Add TypeScript*. Alternatively you can use the `J` or `T` keyboard shortcuts respectively when an action is selected.




A code editor will appear for you to enter the code to run.

## Scripts in blocks

Alternatively JavaScript or TypeScript code can be added as a block, in the same place as other event blocks appear. The code runs whenever an event block in that place would be run. You can add a script block using any of the following methods:

1. Right-click in the margin of an existing event block and select Add▶Add JavaScript or Add▶Add TypeScript
2. Click the *Add...* menu at the end of a group, or the end of the event sheet, and select *Add JavaScript* or *Add TypeScript*

3. Use the `J` or `T` keyboard shortcuts when an event block is selected (note if an action is selected, a script action will be inserted instead)

Remember that the event sheet runs all events in top-to-bottom order every tick, so a script in a block at the top level of an event sheet is run every tick (as if it was an action in an *Every tick* event). Often it is more useful to use script blocks as sub-events, which only run if their parent event block was true.

## Using 'await'

Code written in either a script action or script block is actually run inside an async function. This means your code can use the `await` keyword. For example you can await the result of a fetch over the network.

Note that the rest of the event sheet will continue to run while `await` waits for its result. In other words, `await` pauses the script execution and any actions after the code will immediately run. Then once the awaited promise resolves, any code after the `await` will then run - which will be *after* the following actions have run. You can change this using the System action *Wait for previous actions to complete*: this will wait for any code blocks before it to finish before running the following actions.

## Using imports

Since code in a script action or script block is run inside a function, you cannot use `import` or `export` statements inside scripts in event sheets, since the JavaScript/TypeScript programming languages do not permit these inside functions.

Instead you can import things in to a script file with the purpose *Imports for events*, normally named *importsForEvents.js*. For example if you add the following JavaScript line in the *Imports for events* script:

```
// importsForEvents.js
import * as Utils from "./utilities.js";
```

...then all your JavaScript code in script actions and script blocks can use the exports in *Utils*, e.g. `Utils.myExport()`. This provides a useful way to write a library of functions that can be used by code anywhere else in the project, and is another good way to closely integrate event sheets with code.

*Do not `export` anything in your Imports for events script. This does not work because Construct combines this script with other content and then renames it. Therefore nothing else can import the Imports for events script file as it does not really exist, and so there's no point exporting anything from it either. If you need to share content between scripts, put it in a separate script and import it to the Imports for events script.*

Note that ***Imports for events* scripts are language-specific.** If you add a JavaScript *Imports for events* file, then it only applies to JavaScript code in event sheets - it will not apply to TypeScript code in event sheets - and vice versa. You can have both a JavaScript and TypeScript *Imports for events* file in the same project, but it is recommended to use either all JavaScript or all TypeScript to avoid creating confusing situations where the two *Imports for events* files have different contents.

## Using the runtime interface

All scripts in the event sheet can access a special `runtime` variable which refers to the runtime script interface. This provides functions and properties that lets your code access and control Construct's runtime. This also includes ways to closely integrate code and events, such as iterating the instances picked by the current event.

A very simple example is shown below, which can be used to show a dialog box with the project name in it.

```
alert(runtime.projectName);
```

## Accessing local variables

A useful way to pass values between scripts and the event sheet is to use local variables in the event sheet. These can be accessed by both script actions and script blocks using the variable name `localVars`. This is set to an object with a property for each local variable in scope. The available local variables are the same as are available to a *Set value* action in the same place. This includes any parameters for event sheet functions.

For example a script in an event group with a local variable named *temp* can access the local variable using `localVars.temp`. A useful pattern is to use an action to set a local variable to an expression, and then read from it in a following script action. Alternatively a script could calculate a value and assign it to a local variable, to subsequently be used in the event sheet. It could also be used both ways at once, both reading the variable and then assigning it. See also the Integrating events with script example which shows one of the ways this can be used.

Note that `localVars` excludes global variables, which are available via runtime.globalVars instead. `localVars` is also unique to scripts in the event sheet - script files cannot access it, because they do not have a scope in the event sheet.

> *In some cases, event variables may have names that aren't valid JavaScript/TypeScript identifiers. In this case you can use the string property syntax, e.g.* `localVars["temp"]`.

## Errors

Any exceptions, or rejected promises, arising from a script in an event sheet will be caught by the Construct engine and logged to the console with information about where the error came from. This means unhandled exceptions or rejections will not crash the game (since browsers stop

running scripts if they encounter an unhandled error). However you should keep an eye on the browser console to check for any unexpected errors. For more information see the section on debugging script.

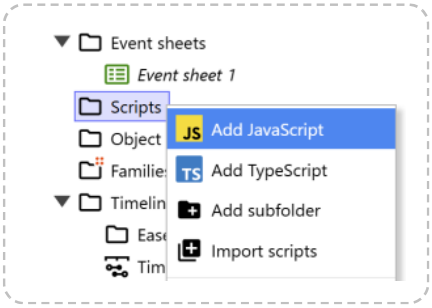
# SCRIPT FILES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/script-files

For those more familiar with coding, script files provide a code editor to write a JavaScript or TypeScript file which works independently of an event sheet.

## Adding script files

Script files can be added in the *Scripts* folder of the Project Bar. Existing JavaScript files (.js) and TypeScript files (.ts) can also be imported using the *Import scripts* option instead.




Once you add a script a code editor appears. The first script added will have some default code added to help you get started.

## Execution of the main script

Construct loads only the *main script* just after the Construct engine scripts run on startup. This is before any loading screen appears. It is a good place to write any initialization code and imports/exports the entire project will use. Since the Construct engine has still not yet initialized at this point, there is no `runtime` variable (representing the runtime script interface) available at the top level. Instead Construct provides a special `runOnStartup` function that runs a callback once the runtime is ready, and provides the `runtime` variable as a parameter.

```
runOnStartup(async runtime =>
{
	// This code runs on startup with
	// the 'runtime' variable available
});
```

The callback can be an `async` function, meaning the `await` keyword can be used inside it, so this is a convenient place to run any asynchronous initialization while also making use of the runtime script interface.

Note that the callback runs **when the loading screen appears.** The project is not yet fully loaded and no objects exist yet. Therefore any code that attempts to interact with objects in your project

in that callback won't work. Instead you'll need to wait for the `"beforeprojectstart"` event, which fires just before the first layout starts running.

```
runOnStartup(async runtime =>
{
	// Wait for "beforeprojectstart" event
	runtime.addEventListener("beforeprojectstart", () => OnBeforeProjectStart(runtime));
});

async function OnBeforeProjectStart(runtime)
{
	// Now the project is loaded and objects exist.
	// You can interact with objects etc. here.
}
```

Note that the function for the `"beforeprojectstart"` event can be async, and Construct will wait for it to complete before the project starts running. This means you can also use `await` in case you need to do any asynchronous initialization there.

### Continuing execution

The only code in the main script that is automatically executed is the top-level scope and any `runOnStartup` callbacks. Beyond that no code in your script will run any more, unless you add event listeners to run callbacks, as previously shown with `"beforeprojectstart"`. The main other event to listen for is the runtime `"tick"` event. Since this fires every tick it provides a good place to keep running code throughout your game. The JavaScript code example below demonstrates a typical way to use this.

```
runOnStartup(async runtime =>
{
	runtime.addEventListener("beforeprojectstart", () => OnBeforeProjectStart(runtime));
});

async function OnBeforeProjectStart(runtime)
{
	runtime.addEventListener("tick", () => Tick(runtime));
}

function Tick(runtime)
{
	// Code to run every tick.
	// Note 'runtime' is passed.
}
```

## Using additional scripts

As noted previously, the only script Construct automatically loads and runs is the *main script*. This appears in bold in the Project Bar.

When you select script files, the Properties Bar shows a *Purpose* property for the script. The main script has the purpose set to *Main script*, and you can only have one main script in your project.

Since you can only have one main script, you must choose whether the main script is JavaScript or TypeScript. There is some support for mixing JavaScript and TypeScript code, but you may find it easier to choose one language to use throughout your project.

To use any other script files in your project, you must `import` them in the main script. This also lets you control the order they are loaded and run. These other script files should have the *Purpose* set to *'(none)'* (indicating Construct won't use it automatically) and also `export` the things it wants to be used by other scripts. See the Imports & exports example for a basic demonstration of using modules in Construct.

To learn more about imports and exports, refer to the MDN guide on JavaScript Modules.

### Using external scripts

Sometimes you want to load a separate script file that is external and not loaded via an import. A good example of this are Web Workers - you may use something like `new Worker("myworker.js")`, where "myworker.js" must always be in a separate file.

You should place these script files in the **Files** folder of the Project Bar instead of the **Scripts** folder. The reason for this is Construct has special processing for everything in the Scripts folder, including moving the files to a different folder on export, or minifying all the scripts there, and these steps can cause the worker script file to stop working after export. On the other hand script files in the Files folder are not processed and are just copied as-is on export, so things like worker scripts will work consistently both in preview and export.

## Integration with scripts in events

Construct loads all script files as modules. Unlike legacy "classic" mode scripts, modules have their own top-level scope. This means things like a top-level function declaration is *not* available in other script files.

Instead you can add imports to the script file with the purpose *Imports for events* which then become available for scripts in events. See the section *Using imports* in Scripts in event sheets for more details.

Another option is to write globals as explicit properties of `globalThis`, e.g. `globalThis.myFunction = function () { ... }` and call it via `globalThis.myFunction()`, but using modules is preferable.

## Errors

Unlike scripts in event sheets, errors arising from the top level of script files are not automatically handled by Construct. If an unhandled exception is thrown, the browser will halt any further execution of script in that file. Typically this causes the rest of your code to stop working, and is considered a crash. See the section debugging scripts to find out how to deal with such issues.

Note one difference is exceptions or rejections in a `runOnStartup` callback are automatically handled by Construct. The error will be logged to the browser console and the runtime will continue to start up and run the game - but note if an error occurred it may not run as expected.

# DEBUGGING SCRIPT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/debugging-script

Browsers provide comprehensive developer tools ("dev tools" for short) to help debug and profile JavaScript code. Construct is designed to allow you to use these industry-standard tools to also debug code used in your project.

Developer tools are complex and sophisticated tools used by professional developers. For full documentation you should refer to each browser's own dev tools documentation, such as Google's Chrome DevTools documentation. However an overview is provided here, focusing on how to use dev tools with Construct specifically. Screenshots of dev tools are taken from Chrome's DevTools, as it is the most widely-used browser. Other browser's dev tools will look different, but generally they work in similar ways.

## Debugging TypeScript

Browsers only directly support running JavaScript code. The way TypeScript works is it compiles the code in to JavaScript before running it, which generally amounts to deleting the type annotations. When debugging code written in TypeScript you'll actually find the compiled JavaScript output of your TypeScript code in the browser developer tools, all using the *.js* file extension rather than *.ts*. However this should not be a significant hindrance - it should look pretty much the same, including with all your comments, and only be missing the type annotations that TypeScript uses.

As TypeScript compiles to JavaScript for running in the browser, the rest of this section refers to JavaScript code.

## Construct's debugger

It should be noted that Construct's debugger cannot debug JavaScript - only the browser dev tools can. In an event sheet Construct's debugger can only stop just before running a script block or script action, and continuing will run all the JavaScript in that block to completion and then advance to the next block. Construct's debugger has no way to stop on code in script files at all. Therefore it is not generally useful for debugging JavaScript code.

Using both the Construct debugger and browser dev tools simultaneously is possible, but is likely to be very confusing. Therefore it is recommended to only use one or the other at a time.

## Opening dev tools

First make sure you have a preview window running and focused before opening developer tools. Otherwise you will open developer tools for the Construct editor, which is not useful, and due to security issues will show a warning in the console.

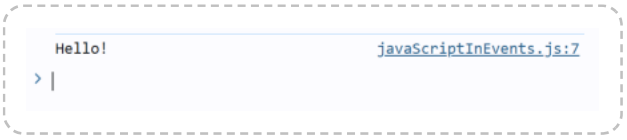
Once a preview is running, in most browsers you can press `F12` to open developer tools. In Safari, use `⌥⌘I`.

Normally the dev tools window can be undocked, so it appears in its own window. This is the easiest way to see the largest possible dev tools window, which is important for more advanced tasks like debugging JavaScript code.

For more information, including using remote debugging to open developer tools for a mobile device, see the tutorial Checking for errors in browsers.

## The console

The *Console* tab provides a list of messages, used for development purposes. Calling `console.log("Hello!")` will add the message *Hello!* here. For more console features available in code, see the Console API. Adding console messages to indicate which parts of the code have been reached, and the contents of any important variables, can be a useful way to diagnose code.




You can also directly type in JavaScript code in the console, which runs when you press `Enter`. This is a useful way to try out snippets of code or quickly write and test a function to use in your project. If you are stopped on a breakpoint, code run in the console can also use any variables in the scope of the breakpoint. Often this is useful if you break in some code with the `runtime` variable available, allowing you to call runtime script interface functions from the console.

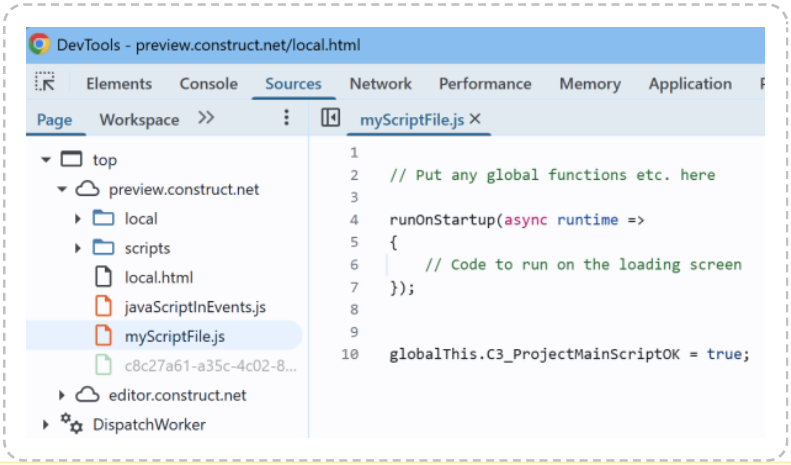
> *Remember that browsers can only directly run JavaScript code. Therefore you can only type JavaScript code in to the console - you can't enter TypeScript code there.*

## Debugging JavaScript code

A more sophisticated tool is the JavaScript debugger. This is an incredibly powerful tool that allows you to pause JavaScript execution at any point, and inspect and alter the full state of your program. Normally this can be found in the *Sources* tab of dev tools.

### Finding your scripts

You can find any script files in your project in this section of the debugger. For example if you add *MyScript.js* to your project, it will be listed here (normally on the left). Sometimes a unique ID may be appended to the name, but you can ignore that. The file should appear under the origin *preview.construct.net*, since that is the domain that project previews run on.




*In worker mode, the scripts will instead appear under a Web Worker named Runtime.*

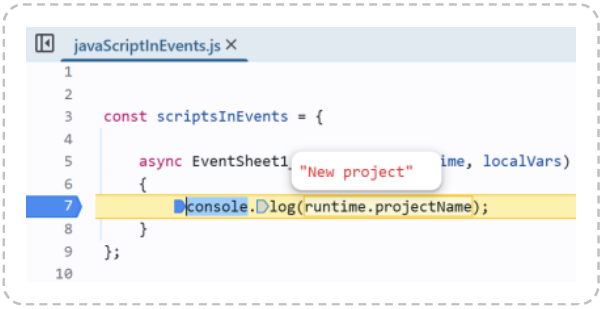
There may also be a special script file named *javaScriptsInEvents.js*. As the name suggests, all JavaScript code used in event sheets appear in this file. Each code block appears as a function whose name is derived from its location in the event sheet, such as *EventSheet1_Event1_Act1*. There may also be *typeScriptInEvents.js* which is the same but for all TypeScript code used in event sheets, but note this is also the JavaScript output of compiling the original TypeScript file.

Messages logged to the browser console also identify where they came from. You can click the location that logged the message to open that script in the debugger, which is a useful way to jump to the relevant code.




## Using breakpoints

Once you have opened a script file in the debugger, you can add a breakpoint by clicking in the margin of the file, where the line numbers appear. The breakpoint can be removed by clicking it again. When the breakpoint is present, just before that line of code runs all script execution will pause and the debugger will highlight that line, allowing you to inspect the state of your code. In this state you can hover the mouse cursor over variables to see their values, look at the call stack, and even run code in the console to alter the state of the program or call functions.




Execution will not continue until you resume the debugger. Normally there is a *Continue* button for this. There are also features to step one line at a time, jump in to or out of functions, and so on. With keyboard shortcuts experienced developers can comfortably control execution of their code through a debugger, watching how the program runs at every step.

You can also use the special *debugger* statement in code:

```
debugger;
```

Whenever this line of code runs, the debugger will automatically stop as if there was a breakpoint on that line. However it will always do this until you delete the statement from your code, whereas normal breakpoints can be removed at any time.

### Breakpoints on startup

Breakpoints, including the `debugger` statement, only ever stop when dev tools are open. If you want to stop on a breakpoint run on startup, this means by the time you open dev tools, the code has already run without stopping. This can be solved by reloading the preview window once you have opened dev tools. Press `F5` with the preview window focused to refresh it, or right-click the caption and select *Reload*. (Some debuggers also provide a reload button.) Since the preview window reloads with dev tools open, all breakpoints in startup code will be able to stop.

### Avoiding stepping in to runtime code

While debugging code that makes lots of calls to the runtime APIs, you may find yourself regularly stepping in to the internal runtime code. This is not normally useful as you only really want to debug your own code.

You can avoid the debugger ever stepping in runtime code by ignoring the runtime scripts. The debugger will avoid ever showing you an ignored script, even if you try to step inside it. In Chrome you can ignore a runtime script if you step in to it, right-click the code, and then select *Add script to ignore list*.

## More tools and techniques

Browsers also provide advanced capabilities like profiling performance, tracking all network requests, and more. Many other debugging tools are also provided. This section has only scratched the surface; over time you'll find and learn useful debugging techniques that will help you learn to quickly diagnose problems in your code and fix them. These are invaluable tools that

experienced developers cannot live without! For beginners, learning to use a debugger and step your code one line at a time is also a very useful technique. It will show you, step-by-step, exactly what your code is doing. This helps you learn the flow of a program and often helps reveal the cause of a problem in an obvious way, where you'd be completely stuck without it.

# USING AN EXTERNAL EDITOR

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/guides/using-external-editor

Construct provides its own built-in code editor based on Monaco to help you conveniently write code in your project. However you may wish to use an external code editor, such as an industry-standard tool like Visual Studio Code (aka VS Code). Construct has special features to help you use external editors, and this guide describes how they work.

Using TypeScript requires some extra steps which are detailed below. However the basic setup is explained first, which works well for JavaScript projects.

## Step 1: save a folder project

First of all, make sure you save your project to a folder. This means all your project files, including your script files, are saved as individual files within the project folder, rather than all contained within a .c3p file. For more information see the section on project folders in Saving projects.

## Step 2: enable auto-reload

Once saved to a folder, right-click the main *Scripts* folder in the Project Bar, and select the *Auto reload all on preview* menu option. After doing that the menu option will appear with a tick to indicate the auto-reload mode is enabled.

Enabling this setting means that every time you preview in Construct, it will re-load all your script files from the project folder, ensuring any changes made by an external editor are loaded.

## Step 3: open external editor

Now you can open your script files in your favorite external code editing tool like VS Code. If the tool has an option to open a folder, you probably want to open the *scripts* subfolder in your project folder, as that is the folder that contains all your JavaScript code. Then you should be able to view and edit your project's script files.

## Step 4: make changes

Now try making a change in the external editor, save the change, and go to Construct and preview the project. The changes made in the external editor should be immediately reflected in preview.

When auto-reload mode is enabled, Construct also shows notifications when you preview indicating how many script files were changed. If you also have a script file open in Construct when you preview, Construct will also reload the file from the project folder to show its latest contents.

*Be warned that in auto-reload mode, if you make a change to a script file in Construct, don't save the changes, and then preview the project, your changes will be lost as they will be replaced with the contents of the file in your project file. It's best to consistently edit your scripts from the same editor to avoid this.*

## Using an external editor with TypeScript

The previous steps are the basic setup for being able to edit code with an external editor, which works well with JavaScript. For TypeScript there are some extra steps involved.

To help you get started, this section describes the necessary steps to set up TypeScript with VS Code, but note the steps may be different for other code editors - refer to their documentation for specific instructions.

### Installation

With some external editors like VS Code, you may need to install TypeScript separately. You will only need to do this once to set up your code editor.

1. First install VS Code if you haven't installed it already
2. Next, install Node.js if you haven't installed it already
3. Then in a terminal run run the command `npm install -g typescript` to install TypeScript

You can check the TypeScript compiler, or `tsc` for short, is installed by running `tsc --version` in the terminal. It should print the version installed.

*Modern versions of Windows use PowerShell for the terminal, and running some of the above commands could return an error like tsc.ps1 cannot be loaded because running scripts is disabled on this system due to the security restrictions set by default. To fix this, run the command* `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser` *to allow permission to run the command, and then try the original command again.*

For more details refer to TypeScript in Visual Studio Code in VS Code's official documentation.

### Setting up the Construct project

Assuming you've followed the steps above, you've already got a project saved to a folder. There are some extra steps you need to do to make sure you can use TypeScript with an external editor though. This includes exporting the TypeScript definitions that Construct uses. To do this, right-click on the *Scripts* folder in the Project Bar, and select TypeScript ▶ Set up TypeScript for external editor. This only needs to be done once per project and performs the initial setup for using TypeScript in your project. It does three things:

1. It creates a TypeScript (.ts) file copy of every JavaScript (.js) file in your project. The TypeScript files aren't shown in the Project Bar, but they'll appear in your project folder. If the project doesn't have any JavaScript files, Construct creates the initial two default script files *main.js* and *importsForEvents.js*, and then creates TypeScript copies of those.
2. A file named *tsconfig.json* is created in the project folder. This is a configuration file for TypeScript. This also does not appear in the Project Bar, as it's just for the external code editor.
3. A subfolder named *ts-defs* is created with lots of *.d.ts* files. These are TypeScript definition files, which tells TypeScript about all of Construct's built-in APIs, as well as some types that are specific to your project. This also doesn't appear in the Project Bar.

If you run *Set up TypeScript* again, it won't overwrite any existing tsconfig.json or .ts files. This means it's safe to run again later on, for example if you add more .js files and want a quick way to make .ts copies of them.

## TypeScript workflow

Once you are up and running, you will likely want to make repeated changes to your TypeScript code, and easily be able to preview the result in Construct. To make this process work smoothly, in VS Code, press `Ctrl` + `Shift` + `B`, and then select *tsc: watch*. This enables a mode where VS Code will automatically compile your .ts files to .js whenever you save the file. Note this must be done once per session.

Then, assuming that you have enabled *Auto reload all on preview* as described above, the workflow goes like this:

1. You make a change to a TypeScript file and save the change
2. TypeScript then automatically compiles the .ts file to .js (or reports errors if you made a mistake)
3. Then preview the project in Construct, at which point the .js file is re-loaded from the project folder

Sometimes you may make changes to the project that affect the TypeScript definition files Construct generated for you. Alternatively when updating to new versions of Construct, the TypeScript definition files could change too. To make sure the TypeScript definition files are up-to-date, right-click on the *Scripts* folder in the Project Bar, and select TypeScript▶Update TypeScript definitions. This essentially does only step 3 from the *Set up TypeScript* steps.

## Don't add .ts files in Construct

Note that we recommend using the workflow described here, where you only have the .js files in your Construct project, and the .ts files are only used by the external editor. This basically out-sources compilation of TypeScript to JavaScript to your external editor, with Construct only using the compiled JavaScript output.

If your Construct project has both a .ts and .js version of the same file, Construct gives precedence to the .js file. This should mean your external editor workflow keeps working as expected, but there's no reason to keep the .ts file in the project - Construct isn't actually using it

at all, and so to avoid confusion, we would recommend not having it in your project. In particular this could be confusing if you try to edit the .ts file from within Construct: it would not change the .js file and so not affect the way your project works, and auto-reload mode would overwrite the file on the next preview, so your changes would be lost anyway.

For more advice about TypeScript workflows, see TypeScript in Construct.

# SUBCLASSING INSTANCES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/guides/subclassing-instances

Construct uses the IInstance and IWorldInstance interfaces to provide access to instances. Many plugins provide derivatives of these to provide plugin-specific APIs. For example the Text plugin adds a `text` property with an `ITextInstance` class that derives from `IWorldInstance`. The Plugin interfaces section covers these.

However in your projects it is often desirable to have a further customised class to represent instances. For example all your Sprite instances will provide an ISpriteInstance interface, but this is still a fairly generic class to represent many different aspects of your project, such as both the player and enemies. *Subclassing* allows you to use your own custom classes like `PlayerInstance` or `EnemyInstance` to represent different objects in your game. Then any time you ask Construct for instances, such as with getAllInstances(), you'll get references to your custom classes instead of a generic `ISpriteInstance` or `IWorldInstance` class.

To use subclassing, follow the steps provided here. The Spell Caster code scripting example also demonstrates how to use this, with a custom `GoblinInstance` class to represent the enemy goblins in the game. A TypeScript version is also available. The code samples here use JavaScript, but subclassing works similarly in TypeScript with the necessary type annotations - see TypeScript in Construct.

## Step 1: create your class

First write a class that extends from the class normally used by the instance. To help make this easy, Construct creates a special kind of class for every kind of instance in the project in the `InstanceType` namespace. For example when extending an object named *Player*, extend from `globalThis.InstanceType.Player`.

> *Make sure your object names are valid JavaScript identifiers. Construct has more permissive rules around naming objects, such as allowing them to start with a number, but JavaScript names do not allow that. Invalid names may be adjusted by Construct, and the easiest approach is to just make sure all your object names are also valid in JavaScript.*

In this example we'll extend a Sprite instance for the object named *Player*, so the class extends from `globalThis.InstanceType.Player`.

```
class PlayerInstance extends globalThis.InstanceType.Player
{
        constructor()
        {
```

```
                super();
        }
}
```

Often it is sensible to organize code by using a separate script file for the class.

## Step 2: set the instance class

Next, use the IObjectClass.setInstanceClass() method to set your custom class. This must be done before any instances in the project are created, to make sure they all use the right class. Therefore this must be called in `runOnStartup`, which runs before the runtime has finished loading, so no instances exist yet.

```
runOnStartup(async runtime =>
{
        runtime.objects.Player.setInstanceClass(PlayerInstance);
}
```

## Step 3: customise the class

Now you can add custom properties and methods to your class. For example the Player class may need to use an ammo counter, and a *shoot* method to fire one of their bullets. You can write these as you would with a normal JavaScript class.

```
class PlayerInstance extends globalThis.InstanceType.Player
{
        constructor()
        {
                super();

                // Start with 5 bullets
                this.ammo = 5;
        }

        shoot()
        {
                // Decrement ammo count
                this.ammo--;

                // create a bullet instance, etc.
        }
}
```

Note that here `ammo` is not an instance variable or anything else associated with Construct's event system: it's just a normal JavaScript object property.

*When using TypeScript, you'll need to use class fields to declare the property type. See TypeScript in Construct for more details.*

Since IInstance has a `runtime` property, within your class you can always use `this.runtime` to refer to the runtime script interface.

You may also wish to make use of private properties and methods to ensure some details remain internal to your class.

*Over time as the Construct engine is improved, there are likely to be more properties and methods added to the base classes. Your derived class's properties and methods override any in base classes, so could potentially hide new APIs added in future. To avoid this causing problems, try to use as specific names as possible that only apply to your project, and avoid generic terms used elsewhere by Construct. If you want to be completely safe, use a different naming scheme for your own additions, such as beginning every property with an underscore. Also, name clashes will never occur with private properties or methods (starting with `#`), so it's a good idea to use those where you can.*

## Step 4: use your custom features

Now whenever you retrieve instances of the player from the existing APIs, you'll get `PlayerInstance` classes instead of the default based on `ISpriteInstance`. Then you can read your custom properties and call custom methods.

```
// Assume called in "beforelayoutstart" event
function OnBeforeLayoutStart(runtime)
{
	// Get player instance from Construct
	const playerInstance = runtime.objects.Player.getFirstInstance();

	// Example uses of custom class
	console.log("Ammo = " + playerInstance.ammo);
	playerInstance.shoot();
}
```

## Conclusion

Subclassing is straightforward to set up, and lets you use custom classes for different objects in your project. This can make your code a lot clearer, and helps you to use the full power of JavaScript/TypeScript classes with instances in Construct's runtime.

# EXPORTING WITH ADVANCED MINIFICATION

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/guides/advanced-minification

When exporting your project, there is an option to *Minify script*. This compresses all the JavaScript code used both in the Construct engine and in your own scripts, and can help obfuscate the project to make it more difficult to reverse-engineer. However if you choose the *Advanced* mode, you may need to adjust how you write JavaScript code in Construct.

When using TypeScript the same adjustments also need to be made. The TypeScript compiler essentially just removes type annotations and outputs regular JavaScript code which is then what gets minified, so the rest of this guide focuses on JavaScript.

The minify mode *None* skips minifying scripts so they are not altered. *Bundle only* combines multiple scripts into a single file without minifying. *Simple* mode eliminates whitespace and does simple adjustments like renaming local variables to shorter names. This does not affect how any of the code is run so is always safe to use. *Advanced* mode renames everything else, including object properties, class method names, and so on. This process is referred to as *property mangling*. It provides the best compression and obfuscation, but in some cases it can affect how the code is run. Therefore you have to be aware of how it works and write your code accordingly to safely use it.

> *Everything in the Construct engine supports Advanced mode. You only need to be careful about using Advanced mode if you use the scripting feature to write JavaScript code in Construct.*

## How property mangling works

Property mangling renames object properties to shorter names. For example consider the following code:

```
const obj = {
    apples: 1,
    oranges: 2
};
console.log(obj.apples, obj.oranges);
```

This will log the numbers *1* and *2* as they correspond to the object properties *apples* and *oranges*. After advanced minification the properties are renamed to shorter names, e.g.:

```
const obj = {
    a: 1,
    b: 2
};
console.log(obj.a, obj.b);
```

This is shorter code (which is faster to load) and harder to understand (which is harder to reverse-engineer). It also works identically to how it did previously.

## Avoiding renaming

In some cases there are specific names you *don't* want renamed. For example if you load an external library at runtime and call a method like this:

```
externalLibrary.doSomethingUseful();
```

Property renaming will then change the name of the function, e.g.:

```
externalLibrary.a();
```

This then breaks the code, since it switched to calling a function that doesn't exist. The problem is that the minification process doesn't know about code that comes from outside your own scripts, and ends up renaming things it shouldn't.

To avoid this, you can use the string property syntax instead, as shown below.

```
externalLibrary["doSomethingUseful"]();
```

Advanced minification never renames string properties, so the name *doSomethingUseful* is preserved even after minification, and the code continues to work.

To write a global name as a string property, access it as a property of `globalThis`, e.g.:

```
globalThis["myGlobalFunction"]();
```

### Built-in names

Construct uses an internal list of built-in browser API names which the minifier uses to avoid renaming built-in functions and properties. For example the names *console* and *log* are on the list, so it never renames the properties in built-in calls like `console.log("Hello")`.

In some cases a browser API may not be on the internal list, for example if it was only just added in the latest browser release. In this case you can access it with string properties to ensure it's not renamed.

## Object and variable names from Construct

Some runtime APIs use the names of things from the Construct editor, such as `runtime.objects.Sprite` (which takes the name *Sprite* from the name of an object). These properties are all consistently renamed with everything else in your code, so you should access them normally as dot properties, to make sure your code still works after minifying.

> *Only object names that are valid JavaScript identifiers will be renamed. For example `Sprite` will be renamed, but `0Sprite` will not. The reason for that is because `0Sprite` is not a valid JavaScript identifier, it cannot be used as a dot property (e.g. `runtime.objects.0Sprite` is invalid). It must always be referred to as a string property (e.g. `runtime.objects["0Sprite"]`), and string properties are not changed by the minifier. Consider always using valid JavaScript property names for object names to avoid any confusion.*

## Don't mix property styles

The main mistake that breaks code with advanced minification is mixing normal properties (which are renamed) with string properties (which are not). For example consider the following code which will be broken:

```
const obj = {
    "apples": 1
};
console.log(obj.apples);
```

This uses both string syntax and normal syntax for the property name *apples*. Normally this code works (logging the number *1*), but after advanced minification only one of the properties is renamed, resulting in code like this:

```
const obj = {
    "apples": 1
};
console.log(obj.a);
```

Notice how `"apples"` had its name preserved (because it uses string syntax), but `obj.apples` was renamed to `obj.a`. That property does not exist on the object, so now the code logs `undefined` instead of *1*.

The solution is to always use the same syntax consistently. So long as a property name consistently never uses string syntax, or consistently always does use string syntax, it will work correctly. Code is only broken if the two types are mixed.

There are several less-obvious ways string properties can end up mixed up. For example `Object.defineProperty()` only takes a string of a property name. Since strings are never renamed, if you use this method, that property name must always be referred to with string syntax.

## Global variables

One extra requirement of *Advanced* minifying is that global variables must always be referred to as a property of the global object. For example the code below shows a problem due to a global variable not being referred to as a property.

```
globalThis.myGlobal = 1;
console.log(myGlobal);
```

The use of `console.log(myGlobal)` normally does refer to the global variable. However this will fail to minify in advanced mode. (This is due to the difficulty of automatically renaming global variables that are written the same way as local variables.) The solution is to always refer to global variables as a property of the global object, i.e. with a dot, as shown below:

```
globalThis.myGlobal = 1;
console.log(globalThis.myGlobal);
```

## Debugging problems

If *Advanced* minification appears to be breaking your code and you're finding it difficult to identify the problem, one option that can help is to choose the *Debug advanced* minify option. In this mode all properties that get renamed are changed to something that preserves the original name - for example `obj.apples` may be renamed to `obj._$apples$_`. This means anything that is broken by properties being renamed will still be broken, but you can see in the source code which property it is attempting to read. For example if an error occurs because `obj._$apples$_` is undefined, and you can see in the debugger that `obj.apples` exists, it would indicate that the property `apples` is inconsistently using both dot and string syntax. This mode also "beautifies" the output to help make it more readable.

## Conclusion

Script minification helps compress the size of the code and makes reverse engineering more difficult. Advanced minification provides the best compression and protection, but requires some care in writing code to ensure it won't be broken by property mangling. Often code will work just fine without any specific changes in advanced mode. The only cases to be aware of are:

1. Calling external library functions (or new browser APIs that aren't on Construct's built-in names list)
2. Mixing string property syntax with normal property syntax
3. Consistently referring to global variables as global properties

If you make sure your code handles the above correctly, then you should be able to safely use advanced minification mode with your own code in Construct.

# USING IMPORT MAPS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/guides/using-import-maps

Construct allows using import maps to customize how you write `import` statements.

Normally you must import a script by its file path, such as:

```
// Standard path-based import
import * as MyModule from "./mymodule.js";
```

However with an import map you can write something like this instead:

```
// Standard path-based import
import * as MyModule from "mymodule";
```

The import map then determines what path to use for "mymodule".

## Adding an import map

Import maps are defined as JSON files. To add an import map to your project, follow these steps.

1. In the Project Bar, right-click on *Files* and select New▶JSON
2. Name the file something like *importMap.json*.
3. Click the file in the Project Bar to show its properties in the Properties Bar.
4. Change the *Purpose* property to *Import map*.

For the contents of the import map, use some JSON data like the following. Refer to the MDN article linked above for full details on the format.

```
{
        "imports": {
                "mymodule": "./mymodule-v1.js"
        }
}
```

Now if you write this in a script file:

```
// Standard path-based import
import * as MyModule from "mymodule";
```

...the import map means that importing "mymodule" will load the path "./mymodule-v1.js".

## Why to use import maps

You don't have to use import maps, but they can be a handy way to organize your imports. To illustrate this, using the example above, suppose you import "mymodule" in lots of places. You can then change that from importing "./mymodule-v1.js" to importing "./mymodule-v2.js" merely by changing the entry in the import map - you don't have to go through every place it's imported and update it. So import maps can help manage complex sets of imports more easily.

Some third-party libraries like three.js also rely on import maps, and so supporting them makes it easier to use these libraries. You can find an example of this in the Integrated 3D engine example.

## Limitations

As of September 2024, browsers don't support using import maps in web workers. Therefore if you use import maps, your project will run in DOM mode and not be able to use worker mode.

Currently Construct only supports the `"imports"` field in an import map. The `"scopes"` and `"integrity"` fields are supported by browsers, but are not yet supported in exported projects, as these need special handling for cases like minified exports.

# SCRIPTING REFERENCE

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

This section of the manual has a comprehensive reference covering all the APIs available in the scripting feature. You can use this to explore the available APIs or learn more about APIs of particular interest.

## Commonly used APIs

The following classes may be a good place to start to learn more about Construct's scripting APIs.

- IRuntime is the runtime script interface representing the Construct engine and is typically the entry point to all other APIs. It's typically accessed by a variable named `runtime`.
- IInstance is the base class of all object instances, and its derivative IWorldInstance is the base class of all object instances that appear in the layout.
- See also the **Plugin interfaces** and **Behavior interfaces** sections, which cover APIs for things like the Sprite plugin and Bullet behavior.

# RUNTIME SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/iruntime

The `IRuntime` script interface provides a way for JavaScript code in Construct to interact with the engine.

## Accessing the runtime script interface

The runtime script interface is typically accessed with the name `runtime`. Note however this is not in a global variable: it is only passed in specific places.

All scripts in event sheets have the runtime interface passed to them as a parameter named `runtime`. For more information see Scripts in event sheets.

In script files, the runtime interface is only passed to the `runOnStartup()` method. Outside of that, it is your responsibility to pass it along wherever else it is needed. For more information see Script files.

## Runtime events

The following events can be listened for using the `addEventListener` method.

> *Many input events are also fired on the runtime interface. This is so they can be used in worker mode, where the `window` and `document` objects are not available, and so input events cannot be listened for. The runtime passes copies of the event objects, since they may have had to be posted over a MessageChannel to the worker.*

**"resize"**

Fires when the display size changes, such as upon window resizes. The event object has `cssWidth` and `cssHeight` properties with the size of the main canvas in CSS units, and `deviceWidth` and `deviceHeight` properties with the size of the main canvas in device units.

**"window-maximized"**

**"window-minimized"**

Fires when the main app window is maximized or minimized. These events are only supported in desktop exports (Windows, macOS and Linux).

**"pretick"**

**"tick"**

**"tick2"**

These events fire every tick, at different points during ticking the engine. Each tick, `"pretick"` fires first, then behaviors are ticked, then `"tick"` fires, then it runs event sheets, then `"tick2"` fires. Use the `runtime.dt` property to access delta-time and step the game's logic by that amount of time.

---

**"beforeprojectstart"**

**"afterprojectstart"**

Fired once only when the first layout in the project starts. `"beforeprojectstart"` fires before the ILayout event `"beforelayoutstart"` on the first layout, which in turn is before *On start of layout* triggers. `"afterprojectstart"` fires after the ILayout event `"afterlayoutstart"` on the first layout, which in turn is after *On start of layout* triggers. In both events, all instances on the first layout are created and available to modify.

> *These events can use async handler functions, and the runtime will wait for them to finish before continuing.*

---

**"beforeanylayoutstart"**

**"afteranylayoutstart"**

**"beforeanylayoutend"**

**"afteranylayoutend"**

Fired whenever any layout starts or ends. The *start* events fire before and after the *On start of layout* trigger, and the *end* events fire before and after the *On end of layout* trigger. The event object has the property `layout` which is the ILayout for the layout that is starting or ending.

> *These events can use async handler functions, and the runtime will wait for them to finish before continuing.*

---

**"keydown"**

**"keyup"**

Fired when keys are pressed and release on the keyboard. These pass copies of a KeyboardEvent.

---

**"mousedown"**

**"mousemove"**

**"mouseup"**

**"dblclick"**

Fired when mouse input is received. These pass copies of a MouseEvent.

> *You can use pointer events like `"pointermove"` instead of mouse events to cover both mouse and touch input.*

---

**"wheel"**

Fired when mouse wheel input is received. This passes a copy of a WheelEvent.

---

**"pointerdown"**

**"pointermove"**

**"pointerup"**

**"pointercancel"**

Fired when pointer input is received. This covers mouse, pen and touch input. These pass copies of a PointerEvent. Construct also adds a `lastButtons` property for `"mouse"` type pointers with the previous `buttons` state, which makes it easier to detect if different mouse buttons have been pressed or released in this event.

> *See also the Tracking pointers example for a demonstration of how multiple simultaneous pointers can be tracked with JavaScript code.*

---

**"deviceorientation"**

**"devicemotion"**

Fired when device orientation or motion sensor input is received. These pass copies of a DeviceOrientationEvent or DeviceMotionEvent respectively.

> *These events require permission to be granted before they will fire. See the `requestPermission()` method in the Touch script interface.*

---

**"suspend"**

**"resume"**

Triggered when the browser/app suspends and resumes execution. Normally when the app goes in to the background (e.g. minimized, or switched back to the home screen), execution of the app is suspended to conserve system resources and save battery power, firing the `"suspend"` event. When the app is reopened, the `"resume"` event is fired and execution resumes. The `isSuspended` property also reflects the current suspend state.

**"save"**

**"load"**

Fired when the savegame system saves or loads the state of the game. The `saveData` property of the event object is extra JSON data to save in the "save" event, and the corresponding last saved data in the "load" event. This allows custom data stored in scripts to be integrated with the savegame system. Note this is serialized to JSON so things like object references and complex types cannot be saved directly.

> *These events can use async handler functions, and the runtime will wait for them to finish before continuing.*

**"instancecreate"**

Fired whenever any new instance is created. The event object has an `instance` property referring to the IInstance (or derivative) that was created.

**"hierarchyready"**

Fired for the root instance in a hierarchy after all instances have finished creating - see the IWorldInstance event for more details. When fired on IRuntime, the event object has an `instance` property referring to the IWorldInstance (or derivative) that was created.

**"instancedestroy"**

Fired whenever any instance is destroyed. After this event, all references to the instance are now invalid, so any remaining references to the instance should be removed or cleared to `null` in this event. Accessing an instance after it is destroyed will throw exceptions or return invalid data. The event object has an `instance` property referring to the IInstance (or derivative) that was destroyed. It also has an `isEndingLayout` property to indicate if the object is being destroyed because it's the end of a layout, or destroyed for other reasons.

**"loadingprogress"**

Fired during a loader layout when the value of the `loadingProgress` property changes.

> *This will still fire even if the loader layout is completely empty, with a progress of 1 to indicate fully complete.*

## Runtime APIs

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for an event. See *Runtime events* above for the available events.

> *See the Handling multiple events example for a way to conveniently handle events.*

---

**objects**

An object with a property for each object class in the project. For example if the project has an object named *Sprite*, then `runtime.objects.Sprite` will refer to the IObjectClass interface for *Sprite*.

> *In some cases, objects may have names that aren't valid JavaScript identifiers. In this case you can use the string property syntax, e.g.* `runtime.objects["Sprite"]`.

---

**getInstanceByUid(uid)**

Get an instance (an IInstance or derivative) by its UID (Unique ID), a unique number assigned to each instance and accessible via its `uid` property.

---

**globalVars**

An object with a property for each global variable on an event sheet in the project. For example if the project has a global variable on an event sheet named *Score*, then `runtime.globalVars.Score` provides access to the global variable from script.

> *In some cases, event variables may have names that aren't valid JavaScript identifiers. In this case you can use the string property syntax, e.g.* `runtime.globalVars["Score"]`.

---

**mouse**

A shorthand reference to the Mouse script interface. This is only set if the Mouse plugin is added to the project.

---

**keyboard**

A shorthand reference to the Keyboard script interface. This is only set if the Keyboard plugin is added to the project.

---

**touch**

A shorthand reference to the Touch script interface. This is only set if the Touch plugin is added to the project.

**timelineController**

A shorthand reference to the Timeline Controller script interface. This is only set if the Timeline Controller plugin is added to the project.

**platformInfo**

A shorthand reference to the IPlatformInfo script interface. Note this always available regardless of whether the Platform Info plugin has been added to the project.

**collisions**

The ICollisionEngine interface providing access to collision APIs.

**layout**

An ILayout interface representing the current layout.

**getLayout(layoutNameOrIndex)**

Get an ILayout interface for a layout in the project, by a case-insensitive string of its name or its zero-based index in the project.

**getAllLayouts()**

Return an array of ILayout interfaces representing all the layouts in the project, in the sequence they appear in the Project Bar.

**goToLayout(layoutNameOrIndex)**

End the current layout and switch to a new layout given by a case-insensitive string of its name, or its zero-based index in the project (which is the order layouts appear in the Project Bar with all folders expanded). Note the layout switch does not take place until the end of the current tick.

**renderer**

The IRenderer script interface representing Construct's renderer, which is used for drawing content. Note that content can only be drawn in certain events, such as ILayer's `"beforedraw"` and `"afterdraw"` events. However you may use the renderer to load resources such as textures at any time, including on startup.

**assets**

An IAssetManager interface providing access to project assets like sound and music files or other project files, as well as audio decoding utilities.

**storage**

An IStorage interface providing access to storage from scripts. Storage is shared with the Local Storage plugin.

**projectId**

A string of the project ID, as specified in Project Properties.

**projectName**

A string of the project name, as specified in Project Properties.

**projectUniqueId**

A string of random characters that Construct automatically generated to uniquely identify this specific project.

**projectVersion**

A string of the project version, as specified in Project Properties.

**viewportWidth**

**viewportHeight**

**getViewportSize()**

Read-only numbers with the project viewport size, as specified in Project Properties. The method returns both values at the same time.

**loadingProgress**

Return the current load progress on a loader layout in the range 0 to 1. This is the same as the *LoadingProgress* system expression. For more information, see the tutorial how to make a custom loading screen.

**imageLoadingProgress**

Return the current loading progress of any memory management *Load* system actions that are currently busy, on a scale of 0-1. This is the same as the *ImageLoadingProgress* system expression.

**sampling**

A read-only string indicating the project's *Sampling* property with one of the strings `"nearest"`, `"bilinear"` or `"trilinear"`.

**isPixelRoundingEnabled**

A boolean representing the project's *Pixel rounding* setting.

**gameTime**

Return the in-game time in seconds, which is affected by the time scale. This is equivalent to the *time* system expression.

**wallTime**

Return the in-game time in seconds, which is not affected by the time scale.

> *Note this is not exactly equivalent to the wallclocktime system expression, as it only measures time elapsed in-game, rather than since the project started.*

**tickCount**

A read-only number with the number of ticks that have run since the project started.

**timeScale**

Set or get a number that determines the rate at which time passes in the project, e.g. 1.0 for normal speed, 2.0 for twice as fast, etc. This is useful for slow-motion effects or pausing.

**dt**

Return the value of delta-time, i.e. the time since the last frame, in seconds.

**dtRaw**

Return the wall-clock time in seconds since the last frame. Unlike *dt*, the "raw" value is not affected by the game time scale or the min/max delta-time clamping.

**minDt**

**maxDt**

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.*

---

### framerateMode

Change the project *Framerate mode* property at runtime. This can be one of the following strings: `"vsync"`, `"unlimited-tick"` or `"unlimited-frame"`. For more details, see the corresponding project property.

---

### framesPerSecond

A read-only number indicating how many frames per second (FPS) the project is rendering. The most common display refresh rate is 60 Hz, so typically an efficiently designed project will render at 60 FPS. Note however if nothing is changing on-screen, then nothing is rendered, and so the FPS measurement may fall to 0 or display a lower result; this does not indicate poor performance, only that fewer frames are necessary to render. The `ticksPerSecond` property indicates how frequently the engine is stepping, which may be different to the frames rendered per second.

---

### ticksPerSecond

A read-only number indicating how many ticks per second (TPS) the project is running at. Each tick processes the logic of the game. Usually a new frame is also rendered every tick, but if nothing changes then rendering a frame is skipped; further, depending on the framerate mode, stepping the engine and drawing frames may happen at different rates. Therefore the ticks per second may produce a different measurement to the frames per second. Usually the project will continually tick even if nothing is visually changing, and only stop ticking if the project is suspended, such as by being minimized or going in to the background.

---

### cpuUtilisation

A timer-based estimate of Construct's main thread CPU usage over the past second, as a percentage in the range 0-1. This can help with performance measurements. Note however this only measures time in function calls that Construct knows about, so it may not include time spent running custom JavaScript code. Timer-based measurements can also be unreliable as most modern CPUs deliberately slow down if not fully loaded in order to save power, so the reading can be misleadingly high unless the system is under full load.

---

### gpuUtilisation

A timer-based estimate of the GPU usage over the past second, as a percentage in the range 0-1. Not all devices support this, in which case this returns `NaN`. Timer-based measurements can also be unreliable as most modern GPUs deliberately slow down if not fully loaded in order to save power, so the reading can be misleadingly high unless the system is under full load.

---

### isSuspended

A read-only boolean indicating if the runtime is currently suspended. When suspended, the runtime stops ticking and drawing anything, and remains inactive. This is normally done to save resources while in the background, such as when a browser tab is in the background or the application window minimized. The `"suspend"` and `"resume"` events are fired when this property changes.

---

### exportDate

A read-only Date object representing the time the project was exported from Construct. In preview mode, this is the time the preview was launched.

---

### callFunction(name, ...params)

Call a function in an event sheet, by a case-insensitive string of its name. Each parameter added after the name is passed to the function. There must be at least as many parameters provided as the function uses, although any additional parameters will be ignored. If the function has a return value, it will be returned from this method, else it returns `null`.

---

### setReturnValue(value)

This can only be called from a script in an event sheet function with a return type other than *None*. It is essentially equivalent to the *Set return value* action. However the fact this method can be called from script can make it easier to return a value from script from an event sheet function. For example an event sheet function could contain a single script action with the code `runtime.setReturnValue(getMyValue())`, which means anywhere the event sheet function is called it returns the value of calling `getMyValue()` in JavaScript.

---

### signal(tag)

Triggers *On signal*, resumes any events waiting for a signal with the given tag, and resolves any promise returned by `waitForSignal().`

---

### waitForSignal(tag)

Returns a Promise that resolves when the given tag is signalled. It may be signalled by either the script API or an event sheet.

### random()

Return a random number in the range [0, 1). This is similar to `Math.random()`, but can produce a deterministic sequence of values if the Advanced Random object overrides the system random.

### sortZOrder(iterable, callback)

Sort the relative Z order of all the IWorldInstances given by *iterable*, using a custom sort function as the *callback* which receives two *IWorldInstance* to compare as arguments. An example using a *myZOrder* instance variable for sorting a Sprite object's instances is given below.

```
runtime.sortZOrder(runtime.objects.Sprite.instances(),
        (a, b) => a.instVars.myZOrder - b.instVars.myZOrder);
```

### saveCanvasImage(format, quality, areaRect)

Take a screenshot of the current display canvas. 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.

### invokeDownload(url, filename)

Invoke a download of the resource at the given *url*, downloading with the given *filename*. Locally-generated content can be downloaded with this method using either a data URI or blob URL for *url*.

### isInWorker

A read-only boolean indicating if the runtime is currently running in the context of a Web Worker. This is controlled by the *Use worker* project property. In worker mode, a more limited set of browser APIs is available. See Functions and classes available to Web Workers.

### getHTMLLayer(index)

Retrieve the HTML element used to contain all HTML content on a given HTML layer in the project. The index is the zero-based index of the HTML layer (not the Construct layer). This method can only be used in DOM mode - if the project property *Use worker* is *Yes* then calling this method will throw an exception. This method is useful to obtain a parent element in which to insert custom HTML content such that it appears layered according to one of Construct's HTML layers. For more information see HTML layers.

### async alert(message)

Show an alert message prompt using the alert() method. This is provided as a runtime method since it forwards the call to the DOM in worker mode. Note that unlike the standard browser `alert()` method, this is an async method - in worker mode it returns a promise that is resolved when the alert is closed, and execution in the worker will continue while the alert is showing. In DOM mode, the alert is blocking and will stop all execution while the alert is showing (but it still returns a promise that resolves when the alert is closed).

> *This method is also made available as a global `alert()` function in worker mode. This is to help make sure simple test code works as expected, even if the code is unintentionally run in the context of a Web Worker, where the browser `alert()` method is not normally available.*

## Runtime SDK events and APIs

These events and APIs are normally only needed for use with the Addon SDK and should not normally be used when using JavaScript coding in Construct. However as they are also part of the `IRuntime` interface they are also documented here.

### SDK events

### "afterload"

Fired after the `_loadFromJson()` call after the rest of the runtime has finished loading some saved state. This means all objects are available and can be looked up by their UID. For example to save a reference to an instance, save its UID to JSON, load its UID and store it in `_loadFromJson()`, and then look it up with `getInstanceByUid()` in the "afterload" event.

### SDK APIs

### sdk

Access the ISDKUtils interface which implements more APIs specific to the addon SDK.

# 8 DIRECTION BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/direction

---

The `I8DirectionBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the 8 direction behavior.

## 8 direction behavior APIs

---

**stop()**

Stop the movement, setting the speed to 0.

---

**reverse()**

Invert the direction of motion. Useful as a simple way to bounce the object off an obstacle.

---

**simulateControl(control)**

Simulate one of the movement controls being held down. Useful when *isDefaultControls* is disabled. The control is provided as a string and must be one of `"left"`, `"right"`, `"up"`, `"down"`.

---

**speed**

Set or get the current speed in pixels per second. Note this cannot exceed `maxSpeed`.

---

**maxSpeed**

Set or get the maximum speed in pixels per second.

---

**acceleration**

**deceleration**

Set or get the acceleration/deceleration of the movement in pixels per second per second.

---

**vectorX**

**vectorY**

**setVector(vectorX, vectorY)**

**getVector()**

Set or get the X and Y components of the movement in pixels per second. The methods allow setting or getting both values at the same time.

---

**isAllowSliding**

A boolean indicating if the behavior is allowed to slide along solids (corresponding to the *Allow sliding* property).

---

**isDefaultControls**

A boolean indicating if the default controls (using the arrow keys) are enabled.

---

**isIgnoringInput**

A boolean indicating if input is currently being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# ANCHOR BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/anchor

The `IAnchorBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Anchor behavior.

## Anchor behavior APIs

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# BULLET BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/bullet

The `IBulletBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Bullet behavior.

## Bullet behavior APIs

### speed

Set or get the current speed in pixels per second.

### acceleration

Set or get the acceleration of the movement in pixels per second per second.

### gravity

Set or get the downwards acceleration caused by gravity in pixels per second per second.

### angleOfMotion

Set or get the current angle of movement in radians.

### bounceOffSolids

A boolean indicating if the *Bounce off solids* property is enabled.

### distanceTravelled

A number indicating the distance the bullet has travelled so far in pixels. Note this can also be set, such as to reset the counter.

### isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# CAR BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/car

The `ICarBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Car behavior.

## Car behavior APIs

### stop()

Stop the movement, setting the speed to 0.

### simulateControl(control)

Simulate one of the movement controls being held down. Useful when *isDefaultControls* is disabled. The control is provided as a string and must be one of `"left"`, `"right"`, `"up"`, `"down"`.

### speed

Set or get the current speed in pixels per second. Note this cannot exceed `maxSpeed`.

### maxSpeed

Set or get the maximum speed in pixels per second.

### acceleration

### deceleration

Set or get the acceleration/deceleration of the movement in pixels per second per second.

### vectorX

### vectorY

### getVector()

Get the read-only X and Y components of the movement in pixels per second. The method returns both values at the same time.

### angleOfMotion

The read-only current angle of the movement in radians.

**steerSpeed**

Set or get the rate the car rotates at when steering, in radians per second.

**driftRecover**

Set or get the rate the car recovers from drifts, in radians per second. In other words, this is the rate the angle of motion catches up with the object angle. The angle of motion can never be more than 90 degrees off the object angle. If the drift recover is greater or equal to *steerSpeed*, no drifting ever occurs. The lower the drift recover, the more the car will drift on corners.

**friction**

Set or get the amount of speed lost when colliding with a solid, from 0 (stop dead) to 1 (speed not affected at all). For example, 0.5 will slow the speed down by half when colliding with a solid.

**turnWhileStopped**

A boolean indicating if the *Turn while stopped* behavior property is enabled. When enabled, it allows rotation while not moving.

**isDefaultControls**

A boolean indicating if the default controls (using the arrow keys) are enabled.

**isIgnoringInput**

A boolean indicating if input is currently being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# DRAG & DROP BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/drag-drop

The `IDragDropBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Drag & Drop behavior.

## Drag & Drop behavior events

See behavior instance event for standard behavior instance event object properties.

**"dragstart"**

Fired when the object starts being dragged.

**"drop"**

Fired when the object is dropped after being dragged.

## Drag & Drop behavior APIs

**axes**

Set or get a string indicating if the dragging is locked to a specific axes, which must be one of `"horizontal"`, `"vertical"` or `"both"`.

**drop()**

Call while dragging to force the object to be dropped.

**isDragging**

A read-only boolean indicating if the object is currently being dragged.

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# FADE BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/fade

The `IFadeBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Fade behavior.

## Example

Below is a sample code snippet demonstrating starting a fade for `inst` (assumed to be an instance with the Fade behavior) and logging to the console when the fade finishes.

```
// Handle "fadeoutend" event which logs to console when
// the fade has finished
inst.behaviors.Fade.addEventListener("fadeoutend", e =>
{
        console.log("Fade finished!");
});

// Start the Fade effect running
inst.behaviors.Fade.startFade();
```

## Fade behavior events

See behavior instance event for standard behavior instance event object properties.

**"fadeinend"**

Fired when the fade in stage finishes, moving on to the wait stage.

**"waitend"**

Fired when the wait stage finishes, moving on to the fade out stage.

**"fadeoutend"**

Fired when the fade out stage finishes. The object may also be destroyed immediately after this event if the *Destroy* property of the behavior is enabled.

## Fade behavior APIs

**startFade()**

Start the fade effect running if it is not already running.

---

**restartFade()**

Force the fade effect to restart from the beginning.

---

**fadeInTime**

Set or get the fade in time in seconds. Set to 0 to skip this stage.

---

**waitTime**

Set or get the wait time, in between the fade in and fade out, in seconds. Set to 0 to skip this stage.

---

**fadeOutTime**

Set or get the fade out time in seconds. Set to 0 to skip this stage.

# FLASH BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/flash

The `IFlashBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Flash behavior.

## Example

Below is a sample code snippet demonstrating flashing `inst` (assumed to be an instance with the Flash behavior) and logging to the console when the flash finishes.

```
// Handle "flashend" event which logs to console when
// the flash has finished
inst.behaviors.Flash.addEventListener("flashend", e =>
{
        console.log("Flash finished!");
});

// Start flashing with the Flash behavior for 2 seconds
inst.behaviors.Flash.flash(0.1, 0.1, 2);
```

## Flash behavior events

See behavior instance event for standard behavior instance event object properties.

### "flashend"

Fired when a flash finishes.

## Flash behavior APIs

### flash(onTime, offTime, duration)

Start flashing for the given `duration` with the object shown visible for the `onTime` and invisible for the `offTime`, with all times in seconds.

### stop()

Stop any currently active flash effect, returning the object to a visible state.

### isFlashing

A read-only boolean indicating if the object is currently flashing.

# FOLLOW BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/follow

---

The `IFollowBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Follow behavior.

### Built-in property strings

The Follow behavior uses the following strings to refer to built-in properties in some APIs: `"x"`, `"y"`, `"z-elevation"`, `"width"`, `"height"`, `"angle"`, `"opacity"`, `"visibility"`, `"destroyed"`.

### Interpolation strings

The following strings are used to refer to interpolation modes: `"step"`, `"linear"`, `"angular"`.

## Follow behavior APIs

---

**startFollowing(inst, fromCurrentPosition = false)**

Begin following the specified IInstance. This starts recording the changes over time of the specified object, and after the delay period has passed, it will then start following the changes for the enabled properties. Until the delay has passed, `hasFollowData` will be false as there is not yet any data to follow. Alternatively the `fromCurrentPosition` parameter can be set to `true`, which allows immediate following. When enabled this creates an initial history entry based on the following object's current state in the past at the delay time. Therefore `hasFollowData` is immediately true and the object is able to immediately start updating. This has the effect of interpolating from the following object's starting position to the followed object's starting position over the delay time.

---

**stopFollowing()**

Stops recording the history of a followed object. Any recorded history of the object that was followed is still preserved, and it will still continue following changes up until the time that this action was used.

---

**followInstance**

A read-only reference to the IInstance being followed, or `null` if none.

---

**mode**

Set or get a string corresponding to the follow mode, which may be one of `"time"` or `"distance"`.

---

**delay**

**maxDelay**

**historyRate**

Set or get the corresponding behavior properties. See the Follow behavior manual entry for more details.

---

**clearHistory()**

Erases any recorded history about the object being followed. This will cause `hasFollowData` to become false and stop updating the object until enough data has been collected again. This is useful for resetting the behavior.

---

**rewindHistory(time)**

Rewinds the follow time by the given time in seconds, deletes history entries past that time, and then continues recording history. Note that it is not possible to rewind further than the max delay, as data beyond that time is erased. This action allows for implementing a 'rewind time' feature where an object can go backwards in time and then continue from a different location, while preserving the history prior to the time it continued from.

---

**hasFollowData**

A read-only boolean indicating if the behavior has enough data to be able to start following on a delay. For example if the behavior starts following an object on a 5 second time delay, then for the first 5 seconds there is no follow data and so the object will not be updated, and `hasFollowData` will be false. Once 5 seconds has elapsed it then starts updating and `hasFollowData` will be true. Note that if following starts with `fromCurrentPosition` set to `true`, then that counts as having follow data immediately.

---

**setFollowingProperty(prop, isEnabled)**

**isFollowingProperty(prop)**

Set or get whether a built-in property is being followed. See the section *Built-in property strings* above for the strings that can be specified for `prop`.

---

**setPropertyInterpolation(prop, interpolation)**

**getPropertyInterpolation(prop)**

Set or get the interpolation mode of one of the built-in properties. See the section *Built-in property strings* above for the strings that can be specified for `prop`, and *Interpolation*

*strings* for the strings that can be specified for `interp`. Generally this is used to change one of the built-in properties from smooth interpolation to step interpolation. For example mirroring an object with the Platform behavior should always update the width instantly, and not interpolate any in-between values.

---

**startFollowingCustomProperty(customProperty, interpolation)**

**stopFollowingCustomProperty(customProperty)**

**isFollowingCustomProperty(customProperty)**

Start, stop, or test if following a custom property. This allows the Follow behavior to track a custom value other than one of the built-in properties such as the X and Y position. Multiple custom properties can be followed, each identified by a case-insensitive string. The `interpolation` parameter determines how values in between history entries are determined (see the section *Interpolation strings* for the strings that can be specified). Step does not interpolate and just uses the previous history entry. Linear interpolation is suitable for linear values like position and size. Angular interpolation is suitable for rotational values like angles. Note that if a custom property value is a string, then only step interpolation is supported. The value to be recorded must be set with `setCustomPropertyValue()`, and then the value to be followed can be retrieved with `getDelayedCustomPropertyValue()`.

---

**setCustomPropertyValue(customProperty, value)**

Set the current value of a custom property that is being followed. The custom property is identified by a case-insensitive string. The `value` parameter can be either a string or a number, but if it is a string then it will only use step interpolation mode. This method should be used every tick while following a custom property, so that the latest value is available when the behavior decides to add a history entry.

---

**getDelayedCustomPropertyValue(customProperty)**

Retrieve the current value to be followed for a custom property, specified by a case-insensitive string.

---

**isPaused**

Set or get whether following is paused. While paused, no further history is recorded, but it also stops advancing the follow time. Upon resuming, the behavior will restart recording the history of the followed object. If the followed object has substantially changed while following was paused, then it will skip to the new position as it follows the history, as no changes in between will have been saved.

---

**saveHistoryToJSON(maxDelay = 0)**

Save the current recorded history of the followed object to a JSON object. This can then be loaded again later using `loadHistoryFromJSON()`. The `maxDelay` parameter can be used to save only a portion of the most recent history, rather than all the recorded history, suitable for

a record/replay feature when the recording duration is less than the behavior's max delay. If `maxDelay` is 0, then it saves all the history.

---

**loadHistoryFromJSON(json)**

Load the recorded history of the object being followed from a JSON object previously saved by `saveHistoryToJSON()`. This also sets the delay to the time of the oldest history entry loaded, so it then immediately follows the amount of data originally saved. This allows for creating a record/replay feature.

---

**isEnabled**

Set or get a boolean indicating whether the behavior is enabled. If disabled it will be completely inactive, without recording any history or updating the object at all.

# JUMP-THRU BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/jump-thru

The `IJumpthruBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Jump-thru behavior.

## Jump-thru behavior APIs

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# LINE-OF-SIGHT BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/line-of-sight

---

The `ILOSBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Line-of-sight behavior.

An additional `ILOSBehaviorRay` interface is returned by the `castRay()` method.

## Examples

See the Scripting raycasting example for a demonstration of using the `castRay()` method to perform raycasting.

## Line-of-sight behavior APIs

---

### range

Set or get the maximum distance in pixels that line-of-sight can reach. If an object is further away than this distance, the object will never have line-of-sight to it, even if the intervening space is clear.

---

### coneOfView

Set or get the angle in radians of the cone of view in which the object can have line-of-sight to other objects, relative to the current angle of the object.

---

### addObstacle(iObjectClass)

If the *Obstacles* property is *Custom*, adds the given IObjectClass as another kind of object to count as an obstruction to line-of-sight.

> *Note that while this is a method for the instance, it affects the entire behavior.*

---

### clearObstacles(iObjectClass)

If the *Obstacles* property is *Custom*, clears all obstacles added with the `addObstacle()` method.

> *Note that while this is a method for the instance, it affects the entire behavior.*

**hasLOStoPosition(x, y)**

Returns a boolean indicating if the object currently has line-of-sight to a position in layout co-ordinates, respecting the range and cone of view.

**hasLOSBetweenPositions(fromX, fromY, fromAngle, toX, toY)**

Returns a boolean indicating if there is line-of-sight between any two positions in the layout, instead of using the object's own position. This respects the range and cone of view, based on *fromAngle*, which is in radians.

**castRay(fromX, fromY, toX, toY, useCollisionCells = true)**

Check for obstacle intersection between any two positions in the layout, returning a `ILOSBehaviorRay` interface representing the result. Check the `didCollide` property of the returned interface to identify if an intersection was found. The other properties of the interface indicate the hit position, normal and reflection angle, if an intersection was found. For more information see the documentation of the `ILOSBehaviorRay` interface below. The *useCollisionCells* parameter specifies whether to use the collision cells optimisation when testing line of sight. Usually this is faster, but in some cases over extremely long distances it can be slower.

> *This method ignores the range and cone of view, to allow raycasting anywhere in the layout.*

**ray**

Returns the `ILOSBehaviorRay` interface representing the result of the last `castRay` method call.

## ILOSBehaviorRay interface

This interface is used to represent the result of a call to `castRay()`. All its properties are read-only.

**didCollide**

Read-only boolean indicating whether an intersection was found.

> *The rest of the properties on this interface are only set if `didCollide` is `true`.*

**hitX**

**hitY**

**getHitPosition()**

If *didCollide* is true, the read-only position of the first obstacle the ray intersected, in layout co-ordinates. The method returns both values at the same time.

---

**hitDistance**

If *didCollide* is true, the read-only distance between the ray start point and the hit position.

---

**hitUid**

If *didCollide* is true, the read-only UID of the instance that was the first obstacle the ray intersected.

---

**getNormalX(length)**

**getNormalY(length)**

**getNormal(length)**

If *didCollide* is true, returns a position at a given distance along the surface normal vector. The `getNormal()` variant returns both values at the same time.

---

**normalAngle**

If *didCollide* is true, the read-only angle of the surface normal at the point of intersection, in radians.

---

**getReflectionX(length)**

**getReflectionY(length)**

**getReflection(length)**

If *didCollide* is true, returns a position at a given distance along the reflection vector. The `getReflection()` variant returns both values at the same time.

---

**reflectionAngle**

If *didCollide* is true, the read-only angle of the reflection at the point of intersection, in radians.

# MOVE TO BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/move

The `IMoveToBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Move To behavior.

## Example

Below is a sample code snippet demonstrating moving `inst` (assumed to be an instance with the Move To behavior) to a position and logging to the console when it arrives.

```
// Handle "arrived" event which logs to console when
// movement has finished
inst.behaviors.MoveTo.addEventListener("arrived", e =>
{
        console.log("Arrived!");
});

// Start the Move To behavior moving to (100, 100)
inst.behaviors.MoveTo.moveToPosition(100, 100);
```

## Move To behavior events

See behavior instance event for standard behavior instance event object properties.

**"arrived"**

Fired when the object arrives at its destination.

**"hitsolid"**

If *Stop on solids* is enabled, fired when the object hits a solid and stops.

## Move To behavior APIs

**moveToPosition(x, y, isDirect = true)**

Start moving the object to a target position in layout co-ordinates. If `isDirect` is true, any existing waypoints will be cleared so the object moves directly to this position; otherwise it will add a waypoint to the queue.

**getTargetX()**

**getTargetY()**

**getTargetPosition()**

Return the current target position in layout co-ordinates that the object is moving to. The `getTargetPosition()` variant returns `[x, y]`.

**getWaypointCount()**

Return the number of waypoints that have been added.

**getWaypointX(index)**

**getWaypointY(index)**

**getWaypoint(index)**

Return the position in layout co-ordinates of a waypoint at a given zero-based index. The `getWaypoint()` variant returns `[x, y]`.

**stop()**

Stop any current movement, and clear all waypoints.

**isMoving**

Read-only boolean indicating whether the object is currently moving.

**speed**

Set or get the current movement speed in pixels per second.

**maxSpeed**

Set or get the maximum movement speed in pixels per second.

**acceleration**

**deceleration**

Set or get the acceleration and deceleration of the movement in pixels per second per second.

**angleOfMotion**

Set or get the current angle the object is moving at, in radians.

**rotateSpeed**

Set or get the rate the object can turn at, in radians per second.

---

**isStopOnSolids**

Set or get a boolean indicating whether the *Stop on solids* property is enabled. When enabled the object will stop its movement, clear all waypoints and fire the `"hitsolid"` event if it touches an object with the Solid behavior during movement.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# ORBIT BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/orbit

The `IOrbitBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Orbit behavior.

## Orbit behavior APIs

**setTargetPosition(x, y)**

Set the position in the layout that the movement will orbit around.

**getTargetPosition()**

Return the current target position in the layout as a two-element array in the form `[x, y]`.

**pin(iWorldInst)**

Pass an IWorldInstance to set the behavior to always orbit around that object's position.

**speed**

Set or get the current rotation speed in radians per second.

**acceleration**

Set or get the current rotation acceleration in radians per second per second.

**rotation**

Set or get the current orbit position by its angle relative to the target position in radians.

**offsetAngle**

For elliptical orbits, set or get the rotation of the ellipse in radians. For circular orbits, this does not affect the orbit path (since rotating a circle has no effect), but it changes the initial angle the orbit starts from.

**primaryRadius**

**secondaryRadius**

Set or get the distance of the orbit from its target position. The primary radius is in the direction of the offset angle, and the secondary radius is perpendicular to the offset angle. For a circular orbit set both values to the same radius; for an elliptical orbit set them to different values.

---

**isMatchRotation**

Set or get a boolean indicating if the behavior will also alter the object's angle to match the direction of travel.

---

**totalRotation**

**totalAbsoluteRotation**

Set or get the total accumulated rotation in radians. These values do not wrap upon completing a full rotation. The `totalRotation` value will decrease for counter-clockwise rotation, whereas `totalAbsoluteRotation` increases regardless of the direction of rotation.

---

**getDistanceToTarget()**

Return the distance from the object to the target position.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# PATHFINDING BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/pathfinding

---

The `IPathfindingBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Pathfinding behavior.

An additional `IPathfindingMap` interface is also used to represent the pathfinding map, i.e. grid of obstacles, which is shared between all Pathfinding behavior instances using the same cell size and cell border settings.

## Examples

See the Scripting pathfinding example for a demonstration of using these APIs to find and display a path around obstacles.

## Pathfinding behavior events

See behavior instance event for standard behavior instance event object properties.

---

**"arrived"**

Fired when a moving object comes to a stop at its destination.

## Pathfinding behavior APIs

---

**map**

The `IPathfindingMap` interface representing this behavior instance's pathfinding map, such as where obstacles are. See the documentation on `IPathfindingMap` below.

---

**async findPath(x, y)**

Starts calculating a path to the given position in layout co-ordinates, and returns a promise that resolves when the path has been calculated. The promise resolves with a boolean indicating whether a path was found.

---

**async calculatePath(fromX, fromY, toX, toY)**

Calculates a path between any two positions, and returns a promise that resolves when the path has been calculated. The promise resolves with a boolean indicating whether a path was found. This can be used for pure pathfinding calculations without taking in to account

the state of the instance: `findPath()` is designed for use with the instance movement and so will only allow one path starting at the object position to be calculated at a time, but `calculatePath` allows any number of paths to be calculated for any positions at any time.

> *Note that when the promise resolves the path nodes should be read immediately, as the path will be overwritten when a subsequent call to `findPath()` or `calculatePath()` resolves.*

---

**startMoving()**

Automatically start moving the object along the found path. This can only be used after a path has been successfully found.

---

**stop()**

If the object is moving along its path, causes it to stop.

---

**maxSpeed**

Set or get the maximum speed in pixels per second the object can move at, for use with *startMoving()*.

---

**speed**

Set or get the current speed of the object if it is currently moving along its path, in pixels per second. This cannot be negative or greater than *maxSpeed*.

---

**acceleration**

**deceleration**

Set or get the acceleration and deceleration rates in pixels per second per second, for use with *startMoving()*.

---

**rotateSpeed**

Set or get the rate at which the object can rotate in radians per second, for use with *startMoving()*. Note this can affect the speed of the object: if the rotation speed is low, the object will have to slow down on tight corners.

---

**isCalculatingPath**

A read-only boolean indicating if a path is being calculated, e.g. via *findPath()*.

---

**isMoving**

A readonly boolean indicating if the object is currently moving along its path after calling *startMoving()*. It is set back to false after the object arrives at its destination.

---

**currentNode**

A read-only number indicating the zero-based index of the node the object is currently moving towards, while *isMoving* is true. This may skip ahead just before the object actually reaches the next node, in order to help it round corners.

---

**getNodeCount()**

Returns the number of nodes in the path that was found, after a path has been successfully found.

---

**getNodeXAt(i)**

**getNodeYAt(i)**

**getNodeAt(i)**

Return the position of a node in the path that was found, in layout co-ordinates, using the zero-based index of the node. This is only available after a path has been successfully found. The `getNodeAt()` variant returns `[x, y]`.

---

***nodes()**

Iterates all nodes in the path that was found. This returns the same information as `getNodeAt()` but as a generator, yielding values of the form `[x, y]`.

---

**directMovementMode**

Set or get a string of one of `"none"`, `"to-destination"` or `"anywhere-along-path"` reflecting the direct movement mode. For more information about the effect of each mode, see the *Direct movement* property in the Pathfinding behavior manual entry.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

## IPathfindingMap interface

This interface is accessed via the `map` property of the Pathfinding behavior script interface, providing access to details such as the grid of obstacles.

---

**cellSize**

**cellBorder**

Read-only numbers with the corresponding Pathfinding behavior properties for this map.

---

**widthInCells**

**heightInCells**

Read-only numbers with the current size of the pathfinding map in cells.

---

**isCellObstacle(x, y)**

Returns a boolean indicating if a cell in the obstacle grid is marked as an obstacle. This is useful for debugging or displaying the obstacle grid. Note the position is taken in cell co-ordinates rather than layout co-ordinates.

---

**isDiagonalsEnabled**

Set or get a boolean indicating whether paths moving along diagonals are allowed. If disabled, the result nodes along paths will only ever change at 90-degree angles (up, right, down and left). If enabled nodes can move along diagonals as well.

---

**moveCost**

Set or get an integer of the base path cost for moving a single cell. This affects the relative cost of additional costs added by other features. The default is 10. The move cost is rounded to an integer, and it is multiplied by the square root of 2 for the diagonal move cost if diagonals are enabled.

---

**async regenerateMap()**

Determine whether each cell in the obstacles grid is an obstacle again. This is a very CPU intensive action and should not be used regularly. If only part of the obstacle map has changed, prefer to use *regenerateRegion()* or *regenerateObjectRegion()*. Returns a promise indicating when the regeneration has finished. Note finding paths before the promise has resolved will not use the updated map.

---

**async regenerateRegion(startX, startY, endX, endY)**

**async regenerateObjectRegion(objectClass)**

As with *regenerateMap()*, but only the specified area is updated. This is usually considerably faster than regenerating the entire map. However as with regenerating the entire obstacle map, changes only take effect after the returned promise resolves. *regenerateRegion()* takes a rectangle in layout co-ordinates to regenerate. *regenerateObjectRegion()* similarly regenerates the rectangle in the layout given by the bounding boxes of the instances of an IObjectClass. Note this can cover multiple rectangles if there are multiple instances.

---

**startPathGroup(baseCost = 1, cellSpread = 1, maxWorkers = 1)**

### endPathGroup()

Start and end a *path group*, which can be used to spread out the paths found inside the group. For more information refer to the corresponding *Start path group* and *End path group* actions in the Pathfinding behavior manual entry.

# PHYSICS BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/physics

The `IPhysicsBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Physics behavior.

The Physics behavior also provides a `IPhysicsBehavior` interface deriving from IBehavior, which specifies the global settings affecting the entire Physics world. This interface can be accessed through the `behavior` property of a Physics behavior instance.

## Examples

See the Physics scripting example for a demonstration of using physics from JavaScript code.

## IPhysicsBehavior APIs

The behavior script interface specifies the properties of the physics world. It is typically accessed through the `behavior` property. Below shows an example of this to change the physics world gravity.

```
const behaviorInst = spriteInst.behaviors.Physics;
const behavior = behaviorInst.behavior;
behavior.worldGravity = 0;
```

### worldGravity

Set or get the force of gravity affecting all Physics objects. By default this is a force of 10 downwards.

### steppingMode

Set or get a string of either `"fixed"` or `"variable"` indicating the Physics time stepping mode. Variable mode uses delta-time for framerate independent simulation, but may be non-deterministic due to variance in timer measurements. Fixed mode uses exactly the same time step every frame regardless of the framerate. This is not recommended since modern devices have a range of refresh rates, and it can cause physics to run too fast or too slow depending on the device. However it also makes the physics simulation deterministic (reproducing identical results every time). For more information see the tutorial Delta-time and framerate independence.

**velocityIterations**

**positionIterations**

Set or get the number of velocity iterations and position iterations used in the physics engine. The default is 8 and 3 respectively. Lower values run faster but are less accurate, and higher values can reduce performance but provide a more realistic simulation.

**setCollisionsEnabled(iObjectClassA, iObjectClassB, state)**

Set whether collisions are enabled between object types using the Physics behavior. The object types are specified by IObjectClass, and *state* is a boolean indicating whether collisions between these types are enabled. Note this affects all instances of the given object types.

# IPhysicsBehaviorInstance APIs

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object, and the corresponding physics body will be destroyed.

**applyForce(fx, fy, imgPt = 0)**

**applyForceTowardPosition(f, px, py, imgPt = 0)**

**applyForceAtAngle(f, a, imgPt = 0)**

Apply a force on the object, either with custom X and Y components, towards a position (in layout co-ordinates), or at an angle (in radians). The latter two are just convenience methods that internally calculate the X and Y components. Applying a force causes the object to accelerate in the direction of the force.
Forces can be applied at an image point with the `imgPt` parameter, which normally also causes the object to rotate. Using `0` (the default) for the image point uses the object's center of mass, which does not cause rotation. Use `-1` to use the object's origin, which may be different to the center of mass and cause rotation. A string of the image point name can also be used.

**applyImpulse(ix, iy, imgPt = 0)**

**applyImpulseTowardPosition(i, px, py, imgPt = 0)**

**applyImpulseAtAngle(i, a, imgPt = 0)**

Apply an impulse on the object, either with custom X and Y components, towards a position (in layout co-ordinates), or at an angle (in radians). The latter two are just convenience methods that internally calculate the X and Y components. Applying an impulse simulates the object being struck, e.g. hit by a bat.
Impulses can be applied at an image point with the `imgPt` parameter, which normally also causes the object to rotate. Using `0` (the default) for the image point uses the object's

center of mass, which does not cause rotation. Use `-1` to use the object's origin, which may be different to the center of mass and cause rotation.

---

**applyTorque(m)**

**applyTorqueToAngle(m, a)**

**applyTorqueToPosition(m, px, py)**

Apply a torque (rotational acceleration) to the object, either directly, or towards an angle or position. The torque and angle are specified in radians.

---

**setVelocity(vx, vy)**

Set the object's current velocity, providing a speed in pixels per second for the X and Y axes.

---

**getVelocityX()**

**getVelocityY()**

**getVelocity()**

Get the X or Y components of the object's current velocity, in pixels per second. `getVelocity()` returns both as `[x, y]`.

---

**teleport(x, y)**

Set the object position preserving the Physics velocity. Normally changing the position of the object will reposition it, but alter the velocity to try to ensure the physics simulation remains realistic even though some external change was made to the object position. Using the `teleport()` method will reposition the object but not alter its Physics velocity in any way, which is sometimes desirable for purposes such as if a Physics object goes through a portal and is meant to appear somewhere else but with the same velocity.

---

**angularVelocity**

Set or get the angular velocity, in radians per second.

---

**isImmovable**

**isPreventRotation**

**density**

**friction**

**elasticity**

**linearDamping**

**angularDamping**

**isBullet**

These are setters and getters for the various properties of the Physics behavior. For more details, refer to the section *Physics properties* in the Physics behavior manual entry.

---

**mass**

Read-only number representing the mass of the physics object, as calculated by the physics engine. This is the area of the object's collision mask multiplied by its density.

---

**getCenterOfMassX()**

**getCenterOfMassY()**

**getCenterOfMass()**

Get the X and Y position of the center of mass of the physics object, as calculated by the physics engine. This depends on the *collision mask* property, and is not necessarily in the middle of the object. `getCenterOfMass()` returns both components as `[x, y]`.

---

**isAwake**

Set or get a boolean indicating whether the Physics object is awake or asleep. Physics simulations are relatively CPU intensive, requiring a lot of calculations. To save processor time, the Physics engine will make objects that have come to a complete stop go in to "sleep" mode so they no longer require processing. However sometimes changes like repositioning an adjacent object will leave the object in "sleep" mode so it will not respond properly. In this situation setting `isAwake` to `true` can be used to force a sleeping object to resume simulation. (It can also be used to force an object to go in to "sleep" mode, but note this is normally done automatically when possible.)

---

**isSleeping**

Deprecated Returns true when `isAwake` is false. Only provided for backwards compatibility; use `isAwake` instead.

---

**createDistanceJoint(imgPt, iOtherInst, otherImgPt, damping, freq)**

Fix two physics objects at a given distance apart, as if connected by a pole. The other instance must be an IWorldInstance which also uses the Physics behavior. An image point can be specified for each with `imgPt` to connect to a specific part of the object. Note that an image point of `0` specifies the center of gravity of the object - if you intend to connect to the object origin, use `-1`. `damping` is the joint damping ratio from 0 to 1, and `freq` is the mass-spring-damper frequency in Hertz.

---

**createRevoluteJoint(imgPt, iOtherInst)**

**createLimitedRevoluteJoint(imgPt, iOtherInst, lower, upper)**

Hinge two physics objects together, so they can rotate freely as if connected by a pin. Limited revolute joints only allow rotation through a certain range of angles (given in radians), like the clapper of a bell. The other instance must be an IWorldInstance which also uses the Physics behavior. An image point can also be specified to connect to a specific part of the object. Note that an image point of `0` specifies the center of gravity of the object - if you intend to connect to the object origin, use `-1`.

---

**createPrismaticJoint(imgPt, iOtherInst, axisAngle, enableLimit, lowerTranslation, upperTranslation, enableMotor, motorSpeed, maxMotorForce)**

Restrict the movement of two physics objects along a specific axis, given by `axisAngle` in radians. An image point can also be specified to connect to a specific part of the object. Note that an image point of `0` specifies the center of gravity of the object - if you intend to connect to the object origin, use `-1`. The other instance must be an IWorldInstance which also uses the Physics behavior. `enableLimit` is a boolean specifying whether there is a lower and upper movement limit; if enabled these are given by the lower and upper translation (in pixels), otherwise unlimited movement is allowed along the axis. A motor can also be enabled by `enableMotor` to provide a continuous force along the axis with `motorSpeed` in radians per second, and `maxMotorForce` the maximum torque.

---

**removeAllJoints()**

Remove all joints from the object. Any objects this object was attached to via joints is also affected. Note some joints automatically disable collisions between the objects, so you may want to manually disable collisions again after removing joints otherwise overlapping objects will "teleport" apart (as the physics engine will try to prevent them overlapping).

---

**getContactCount()**

Return the number of locations the physics engine has identified this object as touching other physics objects.

---

**getContactX(index)**

**getContactY(index)**

**getContact(index)**

Return the position of a contact with another physics object, in layout co-ordinates, given by the zero-based index of the contact. The `getContact` variant returns `[x, y]`.

# PLATFORM BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/platform

The `IPlatformBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Platform behavior.

## Platform behavior APIs

### fallThrough()

If the player is currently standing on a jump-thru platform, this method will make them fall through it.

### resetDoubleJump(allow)

Change whether a double-jump is allowed during the current jump. If `false` passed, a double-jump will no longer be allowed in the current jump, even if it is the first jump. If `true` passed a double-jump will be allowed again, even if a double-jump was already made in the current jump.

### simulateControl(control)

Simulate one of the movement controls being held down. Useful when *isDefaultControls* is disabled. The control is provided as a string and must be one of `"left"`, `"right"`, `"jump"`.

### speed

Get the current speed in pixels per second (read-only).

### maxSpeed

Set or get the maximum speed in pixels per second.

### acceleration

### deceleration

Set or get the acceleration/deceleration of the movement in pixels per second per second.

### vectorX

### vectorY

**setVector(vectorX, vectorY)**

**getVector()**

Set or get the X and Y components of the movement in pixels per second. The methods allow setting or getting both values at the same time.

---

**jumpStrength**

Set or get the initial vertical speed of a jump in pixels per second when the jump key is pressed.

---

**maxFallSpeed**

Set or get the maximum speed in pixels per second the object can accelerate to when in free-fall.

---

**gravity**

Set or get the acceleration caused by gravity, in pixels per second per second, at the angle given by *gravityAngle*.

---

**gravityAngle**

Set or get the angle that gravity accelerates the object, in radians. By default this points down.

---

**isDoubleJumpEnabled**

A boolean indicating if the player may make one extra mid-air jump before landing on the ground.

---

**jumpSustain**

Set or get the maximum time in seconds that the jump strength is sustained at while the jump control is being held before the effect of gravity takes over.

---

**isMoving**

A read-only boolean indicating if the object is currently moving. This checks if either `vectorX` or `vectorY` are non-zero.

---

**isOnFloor**

A read-only boolean indicating if the object is currently standing on a solid or jump-thru.

---

**isByWall(side)**

Test if a solid blocking horizontal movement is immediately to the object's left or right, returning a boolean indicating if a wall was found. Jump-thrus do not count as walls. Pass either `"left"` or `"right"` as the *side*.

---

**isJumping**

A read-only boolean indicating if the object is currently moving upwards.

---

**isFalling**

A read-only boolean indicating if the object is currently in free-fall.

---

**ceilingCollisionMode**

A string of either `"stop"` (the default) or `"preserve-momentum"` (keep the vertical speed) indicating how to handle ceiling collisions.

---

**isDefaultControls**

A boolean indicating if the default controls (using the arrow keys) are enabled.

---

**isIgnoringInput**

A boolean indicating if input is currently being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# ROTATE BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/rotate

The `IRotateBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Rotate behavior.

## Rotate behavior APIs

### speed

Set or get the rotation speed in radians per second. Positive values rotate clockwise and negative values rotate counter-clockwise.

### acceleration

Set or get the rotation acceleration rate in radians per second per second.

### isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# SHADOW CASTER BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/shadow-caster

The `IShadowCasterBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Shadow Caster behavior.

## Shadow Caster behavior APIs

### height

Set or get the simulated height of the object, which adjusts the length of shadow it casts.

### tag

Set or get a string with a tag for this shadow caster. This is used to match the object with different Shadow Light objects, depending on their properties.

### isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# SINE BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/sine

The `ISineBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Sine behavior.

## Sine behavior APIs

### movement

Set or get the movement type of the behavior as a string, which must be one of `"horizontal"`, `"vertical"`, `"forwards-backwards"`, `"size"`, `"width"`, `"height"`, `"angle"`, `"opacity"`, `"z-elevation"` and `"value-only"`. For a full description of each, refer to the Sine behavior manual entry.

### wave

Set or get the wave function used to calculate the movement as a string, which must be one of `"sine"`, `"triangle"`, `"sawtooth"`, `"reverse-sawtooth"` and `"square"`. For a visualisation see this Wikipedia diagram.

### period

Set or get the duration, in seconds, of one complete back-and-forth cycle.

### magnitude

Set or get the maximum change in the object's position, size or angle. This is in pixels for position or size modes, or radians for the angle mode.

### phase

Set or get the progress through one cycle of the chosen wave, from 0 (the beginning of the cycle) to 2π (the end of the cycle). For example setting the cycle position to `Math.PI` will put it half way through the repeating motion.

### value

A read-only number returning the current value of the offset applied by the sine behavior. This is intended to be used when the `movement` property is `"value-only"`.

### updateInitialState()

The Sine behavior records the object's initial state upon its creation, and always oscillates relative to that, even if it is deactivated and later activated after the object has been modified. If the object changes and you wish for the Sine behavior to oscillate relative to the new state instead of its state upon creation, use this method to reset the initial state to the object's current state.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# SOLID BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/solid

The `ISolidBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Solid behavior.

## Solid behavior APIs

### tags

Set or get a list of tags for this solid behavior instance as a space-separated string. This allows for solid collision filtering in combination with the `setSolidCollisionFilter()` method of ISpriteInstance.

### isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# TILE MOVEMENT BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/tile-movement

The `ITileMovementBehaviourInstance` interface derives from IBehaviorInstance to add APIs specific to the Tile Movement behavior.

## Tile Movement behavior APIs

### isIgnoringInput

A boolean indicating if input is currently being ignored. If input is ignored, pressing any of the control keys has no effect. However, unlike disabling the behavior, the object can continue to move.

### isDefaultControls

A boolean indicating if the default controls (using the arrow keys) are enabled.

### simulateControl(control)

Simulate one of the movement controls being held down. Useful when `isDefaultControls` is disabled. The control is provided as a string and must be one of `"left"`, `"right"`, `"up"`, `"down"`.

### isEnabled

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

### setSpeed(x, y)

Modify the speed property of the behavior. X and Y are numerical values that define the speed on the respective axes.

### getSpeed()

Read the speed property of the behavior. The speed is returned as an array with 2 numerical elements, representing the speed for the X and Y axes respectively.

### setGridPosition(x, y, immediate)

Modify the target grid position of the behavior. X and Y are numerical values that define the new target in grid space. Immediate is a boolean flag that indicates if the instance should immediately move to the target position, or move there as if the user was controlling the instance.

---

**getGridPosition()**

Read the current instance position in grid space. Returns an array of 2 numerical elements, being the position in the X and Y axes respectively.

---

**modifyGridDimensions(width, height, xOffset, yOffset)**

Redefine the dimensions of the grid. The Width and height parameters are numbers that specify the new size for the grid cells. The xOffset and yOffset parameters a numbers that specify the grids offset from the world space. If xOffset and yOffset are a multiple of width or height respectively they will have no effect on the alignment of the grid, but will change the grid space position of each cell.

---

**isMoving()**

Returns a boolean indicating if the instance is currently trying to move to a new target position

---

**isMovingDirection(direction)**

Returns a boolean indicating if the instance is currently trying to move to a new target position, in the given direction. Useful for deciding which animation to show for your character. The direction parameter is a string, which must be one of the following `"left"`, `"right"`, `"up"`, `"down"`.

---

**canMoveto(x, y)**

Allows you to check if the instance would collide with an object if was moved to the grid cell specified by the numerical parameters x and y. The parameters being the co-ordinates of the cell in grid space. This does not check for obstructions between the current position and the target position.

---

**canMoveDirection(direction, distance)**

Allows you to check if the instance would collide with an object if it moved a number of cells in a given direction. The direction is specified by the parameter direction, which must be a string of the value `"up"`, `"left"`, `"right"` or `"down"`. The distance is specified by the parameter distance, which must be a number indicating the number of cells to travel ( not the distance in world space ). This will check for any obstructions between the current position and the target. This is quite useful for character AI.

### getTargetPosition()

Returns the current target position in world space as an array of 2 numerical elements. These elements being the X and Y positions respectively.

### getGridTargetPosition()

Returns the current target position in grid space as an array of 2 numerical elements. These elements being the X and Y positions respectively.

### toGridSpace(x, y)

Convert a given co-ordinate in world space, to grid space. The return value is an array of 2 numerical elements. These elements being the X and Y positions in grid space respectively.

### fromGridSpace(x, y)

Convert a given co-ordinate in grid space, to world space. The return value is an array of 2 numerical elements. These elements being the X and Y positions in world space respectively.

# TIMER BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/timer

The `ITimerBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Timer behavior.

## Example

Below is a sample code snippet demonstrating setting two timers on `inst` (assumed to be an instance with the Timer behavior). One is a regular timer and the other is a one-off timer.

*Note that many browsers combine identical log messages in the console, so repeat messages may just increment a number by the message rather than showing it again.*

```
// Handle "timer" event and log to the console that the timer ran.
inst.behaviors.Timer.addEventListener("timer", e =>
{
        console.log(`Timer "${e.tag}" ran!`);
});

// Set a regular timer running every 1 second
inst.behaviors.Timer.startTimer(1, "myRegularTimer", "regular");

// Set a one-off timer to run after 2.5 seconds
inst.behaviors.Timer.startTimer(2.5, "myOneOffTimer", "once");
```

## Timer behavior events

See behavior instance event for standard behavior instance event object properties.

**"timer"**

Fired when a timer period has elapsed. The event object has a `tag` property which is a string of the tag for the timer that has elapsed.

## Timer behavior APIs

**startTimer(duration, tag, type = "once")**

Set a new timer, or if a timer with the same `tag` exists, re-start it with new options. `duration` is the time in seconds until the `"timer"` event fires. If `type` is `"once"`, then the timer event will fire once and not again until `startTimer` is called again. If `type` is set to `"regular"`, then the timer event will keep firing every `duration` seconds. The `tag` is a string that allows identifying different timers.

---

**setTimerPaused(tag, isPaused)**

Set a currently running timer either paused or resumed. When a timer is paused, it will stop firing timer events. When resumed, it will continue firing timer events, resuming from the time that it was paused at. In other words if a timer is set for 1 second, it is paused after 0.5 seconds, and then after some time it is resumed again, the next timer event will fire 0.5 seconds after resuming.

---

**setAllTimersPaused(isPaused)**

This does the same thing as the `setTimerPaused()` method, but affecting all existing timers rather than only one with a given tag.

---

**stopTimer(tag)**

Stop a timer with a specific tag. The timer event will no longer fire for the stopped timer after this call.

---

**stopAllTimers()**

Stop all currently running timers regardless of their tags. The timer event will no longer fire for any timer after this method unless a new timer is started.

---

**isTimerRunning(tag)**

Returns a boolean indicating if a timer with the given tag has been started. Once stopped, the timer no longer counts as running. Paused timers also count as running - use `isTimerPaused()` to identify these timers separately.

---

**isTimerPaused(tag)**

Returns a boolean indicating if a timer with the given tag has been started and then subsequently paused with `setTimerPaused()`.

---

**getCurrentTime(tag)**

Returns the time in seconds since the `"timer"` event last fired, for a timer with a specific tag.

### getTotalTime(tag)

Returns the time in seconds since a timer with a specific tag was started. This is only useful with regular timers, since it will always equal `getCurrentTime()` for one-off timers (after which they fire and the timer no longer exists, so these expressions return 0).

### getDuration(tag)

Returns the duration in seconds for a timer with a specific tag.

### getNormalizedProgress(tag)

Returns the current progress for a timer with a specific tag represented as a number between 0 and 1, regardless of the duration of the timer. For example a normalized progress of 0.5 means half way through the duration.

### hasFinished(tag)

Returns a boolean that is true for the tick that the `"timer"` event fires in. This is the way the *On timer* condition of the Timer behavior checks for finished timers, and this method allows script callers to make the same check, such as by polling the timer in the `"tick"` event.

# TURRET BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/turret

The `ITurretBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Turret behavior.

## Turret behavior events

See behavior instance event for standard behavior instance event object properties.

**"targetacquired"**

Fired when the turret behavior acquires a target. The `targetInst` property of the event object contains a reference to the instance that was acquired as a target.

**"shoot"**

Fired when the turret behavior has a target acquired in range and that it is aiming at, with a frequency up to the given rate of fire. The `targetInst` property of the event object contains a reference to the instance that is the current target.

## Turret behavior APIs

**currentTarget**

A reference to the instance currently acquired as a target, else `null` if o target is acquired. This property can also be assigned in order to ask the behavior to switch to targeting that specific instance; however this may have no effect if the assigned target is out of range.

**range**

Set or get the range that the turret can detect targets in. Any targets further away from the turret than this distance will be ignored.

**rateOfFire**

Set or get the rate in seconds at which the `"shoot"` event fires, when the turret has both acquired a target and rotated to point in the direction of the target.

**isRotateEnabled**

A boolean indicating whether the behavior will control the angle of the object.

---

**rotateSpeed**

Set or get the speed at which the turret can rotate towards targets, in radians per second.

---

**targetMode**

A string of either `"first"` or `"nearest"` indicating the targeting mode. `"first"` will always track the same target until it is destroyed or leaves range; `"nearest"` may switch to a different target if a new target comes closer than the current target.

---

**isPredictiveAimEnabled**

**projectileSpeed**

A boolean indicating whether predictive aim is enabled. If it is enabled, the projectile speed must also be specified in pixels per second. For more information see the section on predictive aim in the Turret behavior manual entry.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# TWEEN BEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/behavior-interfaces/tween

The `ITweenBehaviorInstance` interface derives from IBehaviorInstance to add APIs specific to the Tween behavior.

An actively running tween is represented by ITweenState, which also derives from ITimelineState. These interfaces can be used to control playback, including identifying when tweens end via the `finished` promise.

## Examples

See the Scripting tweens example for a demonstration of using tweens from JavaScript code.

### Basic tween usage

A code sample is shown below of starting a tween and waiting for it to finish.

```
async function doTween(runtime)
{
	// Get a Sprite instance with the Tween behavior
	const inst = runtime.objects.Sprite.getFirstInstance();

	// Create a tween that moves it to (300, 300) over 2 seconds
	const tween = inst.behaviors.Tween.startTween("position", [300, 300], 2, "in-out-sine");

	// Wait for the tween to finish
	await tween.finished;

	// Log to the console now the tween has finished
	console.log("Tween finished");
}
```

### More examples

Some examples of valid calls to `startTween` are shown below (assuming `Tween` represents this behavior).

```
// Tween X position to 300 over 2 seconds linearly
Tween.startTween("x", 300, 2, "linear");
```

```
// Tween position to (300, 300) over 2 seconds with ease "in-out-sine"
Tween.startTween("position", [300, 300], 2, "in-out-sine");

// Looping ping-pong tween to size 200x200 every 0.5 seconds
Tween.startTween("size", [200, 200], 0.5, "out-sine", {
        loop: true,
        pingPong: true
});

// Tween color to blue over 1.5 seconds linearly
Tween.startTween("color", [0, 0, 1], 1.5, "linear");

// Value tween from 100 to 200 linearly over 3 seconds
const t = Tween.startTween("value", 200, 3, "linear", {
        startValue: 100
});
// (then read t.value over time)
```

## Tween properties

When using the `startTween` method, the `prop` parameter must be one of the strings given in the table below. Each property also lists how many values are expected for the `endValue` parameter; if more than 1, they should be passed as an array.

| Property | Number of values |
|---|---|
| `"x"` | 1 |
| `"y"` | 1 |
| `"position"` | 2 |
| `"width"` | 1 |
| `"height"` | 1 |
| `"x-scale"` | 1 |
| `"y-scale"` | 1 |
| `"size"` | 2 |
| `"scale"` | 2 |
| `"angle"` | 1 *(in radians)* |
| `"opacity"` | 1 *(in 0-1 range)* |
| `"color"` | 3 *(RGB values in 0-1 range)* |
| `"z-elevation"` | 1 |
| `"value"` | 1 |

## Ease names

When using the `startTween` method, the `ease` parameter must be one of the strings given in the table below, or the name of a custom ease in the project.

| | | |
|---|---|---|
| `"linear"` | | |
| `"in-sine"` | `"out-sine"` | `"in-out-sine"` |
| `"in-elastic"` | `"out-elastic"` | `"in-out-elastic"` |
| `"in-back"` | `"out-back"` | `"in-out-back"` |

| | | |
|---|---|---|
| `"in-bounce"` | `"out-bounce"` | `"in-out-bounce"` |
| `"in-cubic"` | `"out-cubic"` | `"in-out-cubic"` |
| `"in-quadratic"` | `"out-quadratic"` | `"in-out-quadratic"` |
| `"in-quartic"` | `"out-quartic"` | `"in-out-quartic"` |
| `"in-quintic"` | `"out-quintic"` | `"in-out-quintic"` |
| `"in-circular"` | `"out-circular"` | `"in-out-circular"` |
| `"in-exponential"` | `"out-exponential"` | `"in-out-exponential"` |

## Tween behavior APIs

---

**startTween(prop, endValue, time, ease, opts)**

Start a tween running for a property to a given end value, over a `time` given in seconds, with an ease function specified by `ease`. Returns an ITweenState representing the running tween.

- `prop` must be a string of one of the property names given in the table in the section *Tween properties* above.
- `endValue` must be either a number, or an array of numbers, depending on `prop`. In the table of properties above, where the *Number of values* is 1, this must be a number; where it is greater than 1, it must be an array with that many values.
- `time` is the duration the tween will run for in seconds.
- `ease` is a string of the name of one of the built-in eases in the section *Ease names* above, or the name of a custom ease in the project.

The `opts` parameter is optional for providing further parameters via object properties. The following properties can be used:

- `tags` : a list of tags to assign to the tween, specified either as a space-separated string, or an array of strings
- `destroyOnComplete` : a boolean indicating whether to automatically destroy the instance once the tween completes (default false)
- `loop` : a boolean indicating whether to repeat the tween when it reaches the end (default false)
- `repeatCount` : the number of times to repeat the tween (default 1).
- `pingPong` : a boolean indicating whether to alternate the playback direction when repeating (default false)

- `startValue`: for value tweens only, specifies the start value (default 0).

See above for some code examples demonstrating some of the ways this method can be called.

---

***allTweens()**

Iterates all actively running tweens created by the behavior, represented with ITweenState.

---

***tweensByTags(tags)**

Iterates all actively running tweens matching the given set of tags, represented with ITweenState. The tags may be specified as either a space-separated string, or an array of strings.

---

**isEnabled**

A boolean indicating if the behavior is enabled. If disabled, the behavior no longer has any effect on the object.

# IRENDERER SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/graphics-interfaces/irenderer-interface

The `IRenderer` script interface provides access to Construct's renderer in the runtime. The same interface can be used regardless of the underlying rendering technology (e.g. WebGL or WebGPU). The interface's methods provide high-level drawing commands implemented by Construct, so you don't need to handle low-level concerns like vertex buffers.

`IRenderer` can be used both in project code, such as with the ILayer events `"beforedraw"` and `"afterdraw"`, and in the addon SDK for drawing plugins.

## Renderer state

`IRenderer` uses a persistent rendering state. Therefore to correctly render something, all the intended state must be specified, otherwise it will use an undefined previous state. `IRenderer` simplifies the renderer state to:

1. A blend mode. Typically a normal alpha blend mode is used.
2. A fill mode (internally, the current fragment shader). The fill modes can be *color fill* (draw a solid color), *texture fill* (draw a texture), and *smooth line fill* (for drawing smooth lines).
3. A color set by `setColor()` or `setColorRgba()`. The alpha component of the color is used as the opacity in texture fill mode.
4. A texture set by `setTexture()`. This is only used in texture fill mode.

There are two other states that are more applicable to 3D content:

- The face culling mode, set by `setCullFaceMode()`
- The front face winding that is also used by the face culling mode, set by `setFrontFaceWinding()`

A drawing method should begin by at least specifying the blend mode, the fill mode, the color, and the texture (if texture fill mode is used), before continuing to draw. The renderer efficiently discards redundant calls, so if the state does not actually change then these calls have minimal performance overhead.

Once all state is set up, quads can be issued using one of the `rect()` or `quad()` method overloads. These methods draw using the currently set state.

## Texture options

Some texture methods accept the same options objects to specify the texture parameters. To avoid repeating them, the common options that can be specified are documented below.

- `wrapX` : the texture horizontal wrap mode: one of `"clamp-to-edge"` , `"repeat"` , `"mirror-repeat"`
- `wrapY` : as with `wrapX` but for the vertical wrap mode
- `sampling` : the texture sampling mode, one of `"nearest"` , `"bilinear"` or `"trilinear"` (default)
- `mipMap` : boolean indicating if mipmaps should be used for this texture, default true

# Methods

---

**setAlphaBlendMode()**

Set the blend mode to a premultiplied alpha blending mode.

---

**setBlendMode(blendMode)**

Set the blend mode by a string which must be one of `"normal"` , `"additive"` , `"copy"` , `"destination-over"` , `"source-in"` , `"destination-in"` , `"source-out"` , `"destination-out"` , `"source-atop"` , `"destination-atop"` . Passing `"normal"` is equivalent to calling `setAlphaBlendMode()` .

---

**setColorFillMode()**

Set the fill mode to draw a solid color, specified by the current color.

---

**setTextureFillMode()**

Set the fill mode to draw a texture, specified by the current texture, and using the alpha component of the current color as the opacity.

---

**setSmoothLineFillMode()**

Set the fill mode to draw smooth lines using the current color.

---

**setColor(color)**

Set the current color from a four-element array representing the RGBA components in [0, 1] range, e.g. `[1, 0, 0, 1]` for opaque red.

---

**setColorRgba(r, g, b, a)**

Set the current color by directly passing the RGBA components. in [0, 1] range.

**setOpacity(o)**

Set only the alpha component of the current color in [0, 1] range. Note this does not affect the RGB components.

**resetColor()**

Set the current color to opaque white (1, 1, 1, 1).

**setCullFaceMode(mode)**

**getCullFaceMode()**

Set or get the face culling mode, which may be one of the following strings:

- `"none"`: all faces are rendered
- `"back"`: back faces are culled
- `"front"`: front faces are culled

Whether a face counts as front or back depends on the front face winding (see `setFrontFaceWinding()`). The default mode is `"none"`.

> *Note that mirrored or flipped sprites are in fact showing a back face, which is why Construct defaults to `"none"`.*

**setFrontFaceWinding(mode)**

**getFrontFaceWinding()**

Set or get the front face winding, which is used to determine whether a triangle is front facing or back facing depending on the order ("winding") of the vertices. This is used by the cull face mode (see `setCullFaceMode()`). The mode may be either the string `"cw"` for clockwise winding, or `"ccw"` for counter-clockwise winding. The default mode is `"cw"`, because most content that Construct renders itself uses clockwise winding.

**setCurrentZ(z)**

**getCurrentZ()**

Set and get the current Z component used for all 2D drawing commands that don't specify Z components, such as the `rect2()` and `quad3()`.

**rect(rect)**

Draw a rectangle given by an DOMRect.

**rect2(left, top, right, bottom)**

Draw a rectangle by directly passing the left, top, right and bottom positions.

**quad(quad)**

Draw a quad given by a DOMQuad.

**quad2(tlx, tly, trx, try_, brx, bry, blx, bly)**

Draw a quad by directly passing the positions of each of the four points in the quad.

**quad3(quad, rect)**

Draw a quad given by a DOMQuad, using a DOMRect for the source texture co-ordinates to draw from.

**quad4(quad, texQuad)**

Draw a quad given by an DOMQuad, using another `DOMQuad` for the source texture co-ordinates to draw from.

**quad5(quad, texQuad, colorArr)**

As with `quad4`, but adds a `colorArr` parameter for per-vertex colors. This must be a Float32Array with 16 elements in the order r, g, b, a for the top-left, top-right, bottom-right and bottom-left vertices, in that order.

**quad3D(tlx, tly, tlz, trx, try_, trz, brx, bry, brz, blx, bly, blz, rect)**

**quad3D2(tlx, tly, tlz, trx, try_, trz, brx, bry, brz, blx, bly, blz, texQuad)**

**quad3D3(tlx, tly, tlz, trx, try_, trz, brx, bry, brz, blx, bly, blz, texQuad, colorArr)**

Draw a 3D quad, specifying all four points of the quad with X, Y and Z co-ordinates. The first method accepts texture co-ordinates via a DOMRect *rect*. The second method accepts texture co-ordinates via a DOMQuad *texQuad*. The third is the same as the second, but adds a `colorArr` parameter for per-vertex colors. This must be a Float32Array with 16 elements in the order r, g, b, a for the top-left, top-right, bottom-right and bottom-left vertices, in that order.

**drawMesh(posArr, uvArr, indexArr, colorArr)**

Draw an array of textured triangles based on the given position, texture co-ordinate and index arrays, and an optional per-vertex color array. The `posArr` parameter must be a Float32Array of vertex positions in the sequence x, y, z (and therefore its size must be a multiple of 3). The `uvArr` parameter must be a Float32Array of texture co-ordinates in the sequence u, v (and therefore its size must be a multiple of 2). The `indexArr` parameter must

be a Uint16Array of indices of vertices and texture co-ordinates, in the sequence i, j, k with each set defining a single triangle to be drawn (and therefore its size must be a multiple of 3). Note that indices refer to the index of a vertex, rather than a direct index in to either array, e.g. a position array with elements x1, y1, z1, x2, y2, z2 has six elements but defines two vertices, and so index 1 refers to the second vertex. If `colorArr` is specified, it must be a Float32Array of colors in the sequence r, g, b, a (and therefore its size must be a multiple of 4). These colors override the renderer's current color. If `colorArr` is not specified, it uses the renderer's current color for every vertex. For a code sample, see the section *Drawing meshes* below.

> *Note that for performance reasons, this method does not take in to account the layer Z elevation - all Z co-ordinates are used exactly as-is. If you wish to apply the layer Z elevation, you must offset all the position Z components yourself.*

---

**convexPoly(pointsArray)**

Draw a convex polygon using the given array of points, in alternating X, Y order. Therefore the size of the array must be even, and must contain at least six elements (to define three points).

---

**line(x1, y1, x2, y2)**

Draws a quad from the point (x1, y1) to (x2, y2) with the current line width.

---

**texturedLine(x1, y1, x2, y2, u, v)**

Draws a quad from the point (x1, y1) to (x2, y2) with the current line width, and using (u, 0) as the texture co-ordinates at the start, and (v, 0) as the texture co-ordinates at the end.

---

**lineRect(left, top, right, bottom)**

Draws four lines along the edges of a given rectangle.

---

**lineRect2(rect)**

Draws four lines along the edges of a given DOMRect.

---

**lineQuad(quad)**

Draws four lines along the edges of a given DOMQuad.

---

**pushLineWidth(w)**

**popLineWidth()**

Set the current line width for line-drawing calls. This must be followed by a `popLineWidth()` call when finished to restore the previous line width.

---

**pushLineCap(lineCap)**

**popLineCap()**

Set the current line cap for line-drawing calls. This must be followed by a `popLineCap()` call when finished to restore the previous line cap. The available line caps are `"butt"` and `"square"`.

---

**setTexture(texture)**

Set the current texture to a given ITexture.

---

**createStaticTexture(data, opts)**

Create an ITexture with the content specified by `data`. This method is asynchronous and so returns a Promise that resolves with the created `ITexture`. The `data` parameter may be one of `HTMLImageElement`, `HTMLCanvasElement`, `OffscreenCanvas` or `ImageBitmap`. (Note that in worker mode, only `OffscreenCanvas` and `ImageBitmap` are available.) `opts` specifies options for the texture - see the section *Texture options* above for more details.

> *Static textures do not support changing their content, and are optimized accordingly. If you want to be able to change the content of a texture, use `createDynamicTexture()`.*

---

**createDynamicTexture(width, height, opts)**

Create a new empty ITexture for dynamic use, i.e. expecting the texture content to be replaced using `updateTexture()`. The size of the texture is given by `width` and `height` which must be positive integers. `opts` specifies options for the texture - see the section *Texture options* above for more details.

---

**updateTexture(data, texture, opts)**

Upload *data* as the new texture contents for the ITexture *texture*. This can only be used for textures created with `createDynamicTexture()` and managed by your addon.
*data* can be one of the following types: `HTMLImageElement`, `HTMLVideoElement`, `HTMLCanvasElement`, `ImageBitmap`, `OffscreenCanvas` or `ImageData`. Note in worker mode the DOM types cannot be used (`HTMLImageElement`, `HTMLVideoElement`, `HTMLCanvasElement`); in this case use `ImageBitmap` or `OffscreenCanvas` instead. This method cannot resize an existing texture, so the data must match the size the texture was created with; if the size needs to change, destroy and re-create the texture.
*opts* specifies options for the texture upload which is an object that can include the following properties:

- `premultiplyAlpha`: a boolean indicating whether to premultiply alpha of the image content specified by *data* (default true). Construct always renders using premultiplied alpha so this is normally necessary; however if the data is known to already be premultiplied, set this to false.

---

**deleteTexture(texture)**

Delete an ITexture, releasing its resources. This can only be used for textures created with `createDynamicTexture()` and managed by your addon. Do not attempt to delete textures managed by the Construct engine.

---

**async loadTextureForImageInfo(imageInfo, opts)**

For use with the addon SDK. Load a texture for a given IImageInfo. Returns a promise that resolves with the loaded ITexture. `opts` specifies options for the texture - see the section *Texture options* above for more details.

---

**releaseTextureForImageInfo(imageInfo)**

For use with the addon SDK. Release a texture for a given IImageInfo that was previously loaded with `loadTextureForImageInfo()`.

---

**getTextureForImageInfo(imageInfo)**

For use with the addon SDK. Returns the existing ITexture for a given IImageInfo that was previously loaded with `loadTextureForImageInfo()`, or returns `null` if no texture is loaded (or the texture is still asynchronously loading).

---

**createRendererText()**

Return a new IRendererText interface. This manages text wrapping, drawing text, and uploading the results to a WebGL texture.

---

**setDeviceTransform()**

Set the co-ordinate system to be in device transform mode, which is in units of device pixels and relative to the screen. This can be useful for achieving pixel-perfect rendering.

---

**setLayerTransform(layer)**

Set the co-ordinate system to match the given ILayer. This is the default mode - this method is normally called after `setDeviceTransform()` to restore normal rendering.

## Drawing meshes

The `drawMesh()` method allows passing typed arrays with vertex, texture co-ordinate, index, and optionally color data, for efficient rendering of entire meshes. Note that the mesh drawn with this method uses the same renderer state (blend mode, fill mode, color and texture) for all triangles. To draw parts of the mesh with different renderer state, you will need to make multiple calls to `drawMesh()` with other calls to change state in between. You can create typed arrays over different ranges of the same ArrayBuffer to draw sections of a mesh - see the MDN guide on TypedArray for more details.

Here is some sample code for using `drawMesh()`. This draws four triangles arranged as two quads with the single call to `drawMesh()`. This code assumes `quad` is a DOMQuad of the first quad to draw, and the second quad is drawn shifted 200px to the right; and that `rcTex` is the texture co-ordinates to use (which is repeated for both quads).

```
// Vertex positions as sequence of x, y, z
const posArr = new Float32Array([
        quad.p1.x, quad.p1.y, 0,
        quad.p2.x, quad.p2.y, 0,
        quad.p3.x, quad.p3.y, 0,
        quad.p4.x, quad.p4.y, 0,

        quad.p1.x + 200, quad.p1.y, 0,
        quad.p2.x + 200, quad.p2.y, 0,
        quad.p3.x + 200, quad.p3.y, 0,
        quad.p4.x + 200, quad.p4.y, 0
]);

// Texture co-ordinates as sequence of u, v
const uvArr = new Float32Array([
        rcTex.left, rcTex.top,
        rcTex.right, rcTex.top,
        rcTex.right, rcTex.bottom,
        rcTex.left, rcTex.bottom,

        rcTex.left, rcTex.top,
        rcTex.right, rcTex.top,
        rcTex.right, rcTex.bottom,
        rcTex.left, rcTex.bottom
]);

// Indices of vertices in sequence i, j, k
const indexArr = new Uint16Array([
        0, 1, 2,
        0, 2, 3,

        4, 5, 6,
        4, 6, 7,
```

```
]);

renderer.drawMesh(posArr, uvArr, indexArr);
```

Also note that the usual drawing methods (e.g. quad methods) automatically apply the current layer's Z elevation. For performance reasons, drawing meshes does not do this automatically, and will just use the Z components in the positions array as absolute values. If you want the content to move with layer Z elevation, you'll need to offset the Z components in the positions array yourself.

# IRENDERERTEXT SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/graphics-interfaces/irenderertext

The `IRendererText` interface manages text wrapping, drawing text to a canvas, and then uploading the result to a texture. This makes it easy to display text in Construct's renderer. It is created via the IRenderer method `createWebGLText()`.

## IRendererText APIs

### release()

Destroy the object and its resources. `IRendererText` must be released when it is no longer needed; do not simply drop references, otherwise not all of its resources will be collected. If you are developing a plugin that creates an IRendererText, it should release any it still uses in its own `_release()` method.

### fontFace

Set or get a string of the name of the font face used for drawing text.

### sizePt

Set or get the size of the font, in points, used for drawing text.

### lineHeight

Set or get the extra line height spacing, in pixels, used for drawing text. Note 0 is the default, indicating no offset to the default line height.

### isBold

Set or get a boolean indicating whether the text is bold.

### isItalic

Set pr get a boolean indicating whether the text is italic.

### setColor(color)

Set the color of the text using an array with three elements for the RGB components in the range [0, 1].

### SetColorRgb(r, g, b)

Set the color of the text using separate RGB components in the range [0, 1].

### setCssColor(str)

Set the color of the text using a string with a CSS color, e.g. "red", "#00ffee", "rgb(0, 128, 192)" etc.

### horizontalAlign

Set or get the horizontal alignment of the text within its bounding box. This can be one of `"left"`, `"center"` or `"right"`.

### verticalAlign

Set the vertical alignment of the text within its bounding box. This can be one of `"top"`, `"center"` or `"bottom"`.

### wordWrapMode

Set the word wrapping mode. This can be one of `"word"` (for space-delimited word wrapping) or `"character"` (for wrapping on any character).

### textDirection

Set or get the text writing direction, which can be one of `"ltr"` or `"rtl"`.

### text

Set or get the text string to be drawn.

### setSize(width, height, zoomScale)

Set the size of the area that text can be drawn in. The size is specified in CSS pixels. The `zoomScale` can be increased to render the text at a higher resolution, which can be useful when zooming in.

### getTexture()

Get an ITexture interface representing the texture with the requested text rendered on to it. **Note:** the texture is generated asynchronously, so can return `null` when first requested. Use `setTextureUpdateCallback()` to get a callback when the texture has updated.

### getTexRect()

Return a DOMRect representing the content area of the text on the texture. This is the subset of the texture that ought to be rendered. Note: this is only valid when `getTexture()` returns a non-null result.

---

**setTextureUpdateCallback(callback)**

Set a function to call when the texture containing the rendered text is updated. Since the texture is generated asynchronously, this is necessary to know when to redraw anything that may be displaying the text, so they can redraw with the updated texture.

---

**releaseTexture()**

Release the underlying texture. This can be used to save memory. However the texture will be re-created the next time `getTexture()` is called.

---

**textWidth**

**textHeight**

Return the size of the text bounding box after processing word wrap. This allows determining the size of the actual visible text, rather than the box used for word wrap bounds.

# ITEXTURE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/graphics-interfaces/itexture

The `ITexture` interface represents a texture for use with the renderer.

This interface cannot be directly constructed. It is only available through other APIs.

## ITexture APIs

**width**

**height**

Return the width or height of the texture. Note this refers to the source texture. Construct's in-editor spritesheeting engine means the texture could be significantly larger than an object's image to be rendered from it.

# LAYOUT SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/layout-interfaces/ilayout

---

The `ILayout` script interface represents a layout in the project.

## Getting an ILayout

The `ILayout` interface is typically accessed via the IRuntime `layout` property, e.g. `runtime.layout`. This represents the current running layout. Other layouts can be accessed via the IRuntime methods `getLayout()` and `getAllLayouts()`.

## Layout events

The following events can be listened for using the `addEventListener` method.

---

**"beforelayoutstart"**

**"afterlayoutstart"**

Fired when the layout starts. `"beforelayoutstart"` fires just before *On start of layout*, and `"afterlayoutstart"` fires just after. In both events, all instances on the layout are created and available to modify.

> *These events can use async handler functions, and the runtime will wait for them to finish before continuing.*

---

**"beforelayoutend"**

**"afterlayoutend"**

Fired when the layout ends due to changing to another layout. `"beforelayoutend"` fires just before *On end of layout*, and `"afterlayoutend"` fires just after. In both events, all instances on the layout are still available, but all non-global instances are destroyed immediately after the `"afterlayoutend"` event.

> *These events can use async handler functions, and the runtime will wait for them to finish before continuing.*

## Layout APIs

---

**runtime**

A reference back to the IRuntime interface.

---

**name**

A read-only string of the layout name.

---

**index**

A read-only number of the zero-based index of the layout in the order it appears in the Project Bar.

---

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for an event. See *Layout events* above for the available events.

---

**width**

**height**

**setSize(width, height)**

**getSize()**

Set or get the size of the layout. The methods allow setting and getting both values at the same time.

> *Note a layout cannot have a zero or negative size.*

---

**scrollX**

**scrollY**

**scrollTo(x, y)**

**getScrollPosition()**

Set or get the scroll position in layout co-ordinates. `scrollTo()` is a shorthand for setting both `scrollX` and `scrollY`, and `getScrollPosition()` returns both scroll co-ordinates at the same time.

---

**isUnboundedScrolling**

A read-only boolean reflecting the state of the *Unbounded scrolling* layout property.

---

**scale**

Set or get the layout scale, with `1` being the default scale, `2` being 2x scale, etc. This scales all the layers in the layout, taking in to account their scale rate property.

---

**angle**

Set the layout angle in radians. This rotates all the layers in the layout.

---

**projection**

Set or get a string specifying the current layout projection, which must be one of `"perspective"` or `"orthographic"`. For more details see *Projection* in Layout Properties.

---

**setVanishingPoint(vpX, vpY)**

**getVanishingPoint()**

Set or get the *Vanishing point* layout property, with each component in the range 0-1. The getter returns an array with two elements in the form `[vpX, vpY]`.

---

**effects**

An array of IEffectInstance representing the effect parameters of the effects on this layout.

## Layer APIs

These APIs relate to the set of layers on the layout.

---

**getLayer(layerNameOrIndex)**

Get an ILayer interface for a layer on the layout, by a case-insensitive string of its name or its zero-based index. When passing a number, an out-of-range number is clamped to the valid range and the nearest layer returned. When passing a string, if no layer with the given name is found, the method returns `null`.

---

***allLayers()**

Iterates ILayer interfaces representing all the layers on the layout, in increasing Z order.

---

**getAllLayers()**

Return an array of ILayer interfaces representing all the layers on the layout, in increasing Z order.

---

**addLayer(layerName, insertBy, where)**

Create a new layer and insert it to the layer tree at runtime (also known as a *dynamic layer*). `layerName` is a string of the name to use for the added layer, which must be different to all

existing layers already added, including other dynamic layers. `insertBy` is an ILayer of another layer to insert the new layer relative to. `where` specifies where to insert the new layer relative to the `insertBy` layer, which may be one of the following strings:

- `"above"` or `"below"`: insert adjacent to the `insertBy` layer, above or below it in Z order.
- `"top-sublayer"` or `"bottom-sublayer"`: insert as a sub-layer of the `insertBy`, at the top or bottom of its existing sub-layers. `insertBy` may also be `null`, in which case the new layer is added at the top or bottom at the root level of the layer tree.

---

**moveLayer(layerToMove, insertBy, where)**

Remove and re-insert a layer to a new location in the layer tree. This works similarly to `addLayer()`, except `layerToMove` refers to an ILayer that already exists; otherwise the `insertBy` and `where` parameters are used in the same way.

---

**removeLayer(layer)**

Remove a given ILayer from the layer tree. 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.

---

**removeAllDynamicLayers()**

Removes all layers added using the `addLayer()` method, 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.

# LAYER SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/layout-interfaces/ilayout/ilayer

---

The `ILayer` script interface represents a layer on a layout.

## Examples

See the Input event position example for a demonstration of using the layer `cssPxToLayer()` method, which is useful when handling input events. The Custom Layer Drawing example also demonstrates using code to draw custom content to a layer with the `"afterdraw"` event.

## Layer events

The following events can be listened for using the `addEventListener` method.

---

**"beforedraw"**

**"afterdraw"**

Fired when the layer is drawn, allowing custom drawing code to draw on this layer. As Construct uses a back-to-front renderer, content drawn in the `"beforedraw"` event will appear underneath other content on the layer, and content drawn in the `"afterdraw"` event will appear on top of other content on the layer. The event object has the following properties:

- `renderer`: the IRenderer script interface with which you can draw content.
- `layer`: the `ILayer` of the layer being drawn.

> *Only draw content inside this event. Construct does not expect you to draw anything outside of this event (which includes code after an `await`), and if you attempt to do so anyway, it may not appear correctly.*

## Layer APIs

---

**runtime**

A reference back to the IRuntime interface.

---

**name**

A read-only string of the layer name.

**index**

A read-only number with the zero-based index of the layer on its layout. The bottom layer has an index of 0, with the index increasing upwards in Z order.

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for an event. See *Layer events* above for the available events.

**layout**

The ILayout interface representing the layout this layer belongs to.

**parentLayer**

A reference to the layer's parent `ILayer` if it is a sub-layer, else `null` if it is a top-level layer.

***parentLayers()**

Iterates all the layer's parent layers, moving up towards the top of the hierarchy.

***subLayers()**

Iterates the layer's own sub-layers in increasing Z order. This does not iterate any sub-layers at lower levels in the hierarchy.

***allSubLayers()**

Iterates the layer's sub-layers and further sub-layers beneath those recursively, in increasing Z order.

**isInteractive**

A boolean indicating if the layer is interactive, allowing its content to respond to mouse and touch input.

> *Note that this returns the layer's own interactive state. If it has a non-interactive parent layer, this property can be `true` but the layer will still not be interactive. Use `isSelfAndParentsInteractive` to check if the layer and all its parents are interactive.*

**isSelfAndParentsInteractive**

A read-only boolean indicating if both this layer and all its parent layers are set to be interactive. If this is true the layer content will respond to mouse and touch input.

**isVisible**

A boolean indicating if the layer is visible. When invisible, the layer skips drawing entirely.

> *Note that this returns the layer's own visibility state. If it has an invisible parent layer, this property can be* `true` *but the layer will still not be visible. Use* `isSelfAndParentsVisible` *to check if the layer and all its parents are visible.*

**isSelfAndParentsVisible**

A read-only boolean indicating if both this layer and all its parent layers are set to visible. If this is true the layer will be drawn.

**isTransparent**

A boolean indicating if the layer background is transparent. When transparent, the background color is ignored.

**backgroundColor**

Set or get the background color of a layer as an array with 3 elements specifying the red, green and blue components with values in the 0-1 range. Note this is ignored if the layer is transparent.

**isHTMLElementsLayer**

A boolean indicating if this layer acts as a HTML layer. For more information see HTML layers.

**scrollX**

**scrollY**

**scrollTo(x, y)**

**getScrollPosition()**

**restoreScrollPosition()**

Independently scroll a layer, regardless of where the layout is scrolled to. By default layers all follow the layout scroll position. Upon setting a layer's scroll position, the layer will stop following the layout scroll position, and remain scrolled at the position specified. The `restoreScrollPosition()` method reverts the layer to the default mode where it follows the layout scroll position. When not independently scrolling a layer, the `scrollX` and `scrollY` getters return the layout scroll position.

**parallaxX**

**parallaxY**

Set or get the horizontal and vertical parallax rates of a layer.

---

**opacity**

The opacity of the layer, as a floating point number in the range [0, 1], where 0 is fully transparent and 1 is fully opaque. Note that changing the opacity to a value other than 1 will force the layer to render via its own texture.

---

**scale**

Set or get the layer scale, taking in to account its scale rate property.

---

**scaleRate**

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

---

**angle**

Set or get the layer angle in radians.

---

**zElevation**

Set or get the Z elevation of the 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).

---

**renderingMode**

Set or get the *Rendering mode* layer property as a string of either `"2d"` or `"3d"`. This allows dynamically changing a layer between rendering modes at runtime.

---

**getViewport()**

Return a DOMRect representing the bounds of the viewport on this layer in layout co-ordinates.

---

**isForceOwnTexture**

A boolean indicating the layer's *Force own texture* property. For more information see the property in the Layers manual entry.

---

**blendMode**

A string indicating the blend mode of the layer, controlling how it draws over the other layers behind it. This must be one of `"normal"`, `"additive"`, `"copy"`, `"destination-over"`,

`"source-in"`, `"destination-in"`, `"source-out"`, `"destination-out"`, `"source-atop"`, `"destination-atop"`.

---

**effects**

An array of IEffectInstance representing the effect parameters of the effects on this layer.

---

**cssPxToLayer(clientX, clientY, z = 0)**

**layerToCssPx(layerX, layerY, z = 0)**

Convert between positions in CSS pixels, such as the `clientX/Y` properties of an input event, and layer co-ordinates within the project. An optional Z value can be provided to do the conversion taking in to account Z elevation to a certain height on the layer. This is useful for purposes like identifying what position in a layer was clicked in an input event, or positioning a HTML element in layer co-ordinates. Both methods return a pair of co-ordinates in the form `[x, y]`.

---

**drawSurfaceToLayer(dsX, dsY, z = 0)**

**layerToDrawSurface(layerX, layerY, z = 0)**

Convert between positions in layer co-ordinates and the draw surface in units of device pixels. An optional Z value can be provided to do the conversion taking in to account Z elevation to a certain height on the layer. Both methods return a pair of co-ordinates in the form `[x, y]`. These methods are intended for the Addon SDK and are for identifying the draw surface co-ordinates that correspond to an object.

---

**renderScale**

A read-only number with the render scale factor for this layer. This is intended for the Addon SDK where plugins need this value to render taking in to account Construct's scale factor.

# IANIMATION SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ianimation

The `IAnimation` interface represents an animation, typically from a Sprite object via its ISpriteInstance interface. Each frame of the animation is represented by the IAnimationFrame interface.

## IAnimation APIs

**name**

A read-only string of the animation name.

**speed**

A read-only number with the animation playback speed in animation frames per second.

**isLooping**

A read-only boolean indicating if animation playback repeats when it reaches the end.

**repeatCount**

A read-only number indicating how many times to repeat the animation.

**repeatTo**

A read-only number of the zero-based frame index to go back to when repeating the animation.

**isPingPong**

A read-only boolean indicating if the animation will reverse when reaching the start or end of the animation.

**frameCount**

A read-only number of frames in this animation.

**getFrames()**

Return an array of IAnimationFrame representing all the frames in this animation in sequence.

***frames()**

Iterates all IAnimationFrame in this animation in sequence.

# IANIMATIONFRAME SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ianimationframe

The `IAnimationFrame` interface represents a single animation frame within an IAnimation. It derives from the IImageInfo script interface.

> *Note that when accessing the origin, image points, or collision polygon points on `IAnimationFrame`, the positions are returned in normalized form, in the range 0-1, representing the position set in the editor. For example (0, 0) is the top-left point, (1, 1) is the bottom-right, and (0.5, 0.5) is the middle. This differs from the ISpriteInstance methods which return positions in layout co-ordinates based on the current location and orientation of the Sprite instance it is called on.*

## IAnimationFrame APIs

**duration**

A read-only number with the relative duration of this animation frame, i.e. 1 for standard speed, 2 for twice as long, etc.

**tag**

A string of the tag assigned for this frame in the Animation Editor.

**originX**

**originY**

**getOrigin()**

Read-only numbers with the normalized position of the origin within this animation frame, ranging from 0-1. The method returns both values at the same time.

**getImagePointCount()**

Return the number of image points on the animation frame.

**getImagePointX(nameOrIndex)**

**getImagePointY(nameOrIndex)**

### getImagePoint(nameOrIndex)

Return the location of an image point on the animation frame in normalized co-ordinates, i.e. ranging from 0-1. Image points are identified either by a case-insensitive string of their name, or their index. If the image point is not found, this returns the origin instead. The `getImagePoint` variant returns `[x, y]`.

---

### getPolyPointCount()

Return the number of collision polygon points on the animation frame.

---

### getPolyPointX(index)

### getPolyPointY(index)

### getPolyPoint(index)

Return the location of a collision polygon point on the animation frame in normalized co-ordinates relative to the origin, by its zero-based index. The `getPolyPoint` variant returns `[x, y]`.

> *Note that the returned positions are both normalized and relative to the origin. For example when the origin is at (0.5, 0.5), a collision poly point in the top-left corner has the co-ordinates (-0.5, -0.5).*

# IBEHAVIOR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ibehavior

The `IBehavior` interface represents a kind of behavior, such as Solid, Physics or Pin. Some behaviors derive from this class to add extra options that are global to the entire behavior, such as the physics world properties in the Physics behavior. This interface is usually accessed through the IBehaviorInstance `behavior` property.

## IBehavior APIs

### runtime

A reference back to the IRuntime interface.

### id

A read-only string with a unique identifier for this behavior as specified by the behavior developer.

### getAllInstances()

Return an array of all instances that have this kind of behavior, for example every object with the Solid behavior. Note the returned instances may come from a range of different object types.

### static getByConstructor(ctor)

Return an `IBehavior` from its constructor in the `C3.Behaviors` namespace, or `null` if it does not exist. For example `IBehavior.getByConstructor(C3.Behaviors.Bullet)` will return the `IBehavior` interface representing the Bullet behavior, or `null` if the behavior is not used in the project.

# IBEHAVIORINSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ibehaviorinstance

The `IBehaviorInstance` interface represents a behavior on an IInstance.

Many behaviors return a more specific class deriving from IBehaviorInstance to add APIs specific to the behavior. See the Behavior instances reference for more information.

## Getting an IBehaviorInstance

Behavior instances are typically accessed via the `behaviors` property of IInstance, followed by the name of the behavior. Some example code is shown below.

```
const mySpriteInst = runtime.objects.Sprite.getFirstInstance();
const myBehaviorInst = mySpriteInst.behaviors.Platform;
// ... do something with myBehaviorInst ...
```

## IBehaviorInstance APIs

### addEventListener(type, func, capture)

### removeEventListener(type, func, capture)

Add or remove an event handler for a particular type of event fired by an addon's script interface. An event object is passed as a parameter to the handler function. See behavior instance event for standard event object properties. For information on which events are fired by specific addons and which additional event object properties are available, see the documentation on each addon's script interfaces.

### dispatchEvent(e)

Dispatch an event, firing any handler functions that have been added for the event type. You can use `new C3.Event(eventName, isCancellable)` to create an event object that can be dispatched (e.g. `new C3.Event("arrived", true)`), and add any extra properties relevant to your event to that object. This can also be used by the addon SDK to cause your addon to fire an event in the script interface, e.g.:

```
const e = new C3.Event("arrived", true);
this.GetScriptInterface().dispatchEvent(e);
```

### instance

A reference to the IInstance representing the object instance this behavior instance is affecting.

---

**behavior**

A reference to the IBehavior representing the kind of behavior, e.g. Solid or Physics.

---

**behaviorType**

A reference to the IBehaviorType the behavior instance belongs to.

---

**runtime**

A reference back to the IRuntime interface.

# IBEHAVIORTYPE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ibehaviortype

The `IBehaviorType` interface represents a behavior added to an object class. For example if SpriteA and SpriteB both have the Bullet behavior added to them, then there are two behavior types, one for each object type.

## IBehaviorType APIs

### runtime

A reference back to the IRuntime interface.

### behavior

A reference to the IBehavior of the associated behavior.

### name

A read-only string of the behavior type name.

# DOMINSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/idominstance

The `IDOMInstance` script interface represents a single instance of an object type (represented by IObjectClass) that appears in a layout and represents a DOM element such as a button or other form control at runtime. It derives from the IWorldInstance script interface.

Note these methods can still be used in worker mode, since it does not directly access a DOM element.

## DOMInstance APIs

**focus()**

**blur()**

Focus or blur the DOM element represented by this instance.

**setCssStyle(prop, val)**

Apply a CSS style to the DOM element, using a string of the property name (in CSS format, e.g. `"background-color"` and a string of the property value (e.g. `"red"`).

**getElement()**

Return the HTML element used to represent the object.

*Since the DOM APIs are not available in worker mode, this will throw an exception when running in a Web Worker.*

# EFFECT INSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ieffectinstance

---

The `IEffectInstance` script interface represents the parameters for a single effect on a IWorldInstance, ILayer or ILayout. It is typically accessed through the `effects` property.

## Examples

See the Scripting effect parameters example for a demonstration of using the `setParameter()` method to modify effect parameters.

## Effect APIs

---

**index**

The zero-based index of this effect, which is its index in the `effects` array.

---

**name**

A read-only string of the effect name.

---

**isActive**

A boolean indicating whether this effect is enabled or not. Inactive effects act the same as the effect being deleted, but the effect can later be reactivated if it is needed again. Note making effects inactive if they are not needed improves performance.

---

**setParameter(index, value)**

**getParameter(index)**

Set or get an effect parameter by the zero-based parameter index. Most parameters use a number as the value. Note however that color parameters are represented by an array with three elements, i.e. `[r, g, b]`. The R, G and B values are normalized to floats in the [0, 1] range.

# IFAMILY SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/ifamily

---

The `IFamily` script interface represents a family in the project, e.g. a family of Sprite object types created in the Project Bar.

`IFamily` derives from IObjectClass which is a base class shared with object types (IObjectType).

## Getting an IFamily

As with object types, references to the project's families are typically accessed through the IRuntime interface `objects` property. For example `runtime.objects.Family1` would refer to the `IFamily` interface for the family *Family1*, assuming it was added to the project.

## Family APIs

---

**getAllObjectTypes()**

Returns an array of all the IObjectType members of this family.

---

***objectTypes()**

Iterates over all the members of this family (as IObjectType).

---

**hasObjectType(objectType)**

Returns a boolean indicating whether the specified `IObjectType` is a member of this family.

# IIMAGEINFO SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iimageinfo

The `IImageInfo` interface represents an image in the project. It is also the base class of IAnimationFrame.

## IImageInfo APIs

**width**

**height**

**getSize()**

Read-only numbers specifying the dimensions of the image in pixels. The method returns both values at the same time.

# INSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iinstance

---

The `IInstance` script interface represents a single instance of an object type. Instances that appear in the layout have a IWorldInstance interface instead, but it derives from `IInstance`, so these methods and properties are available for any type of instance.

Many objects return a more specific class deriving from `IInstance` or `IWorldInstance` to add APIs specific to the plugin. See the Plugin interfaces reference for more information.

## Getting an IInstance

Instances are typically accessed through IObjectClass methods like `getFirstInstance()`. For example, `runtime.objects.Sprite.getFirstInstance()` will return the first instance of the Sprite object type.

## Instance events

The following events can be listened for on any instance using the `addEventListener` method. See instance event for standard event properties. Note many more kinds of addon-specific events can be fired. See the documentation on each addon's script interfaces for more information.

---

**"destroy"**

Fired when the instance is destroyed. After this event, all references to the instance are now invalid, so any remaining references to the instance should be removed or cleared to `null` in this event. Accessing an instance after it is destroyed will throw exceptions or return invalid data. The event object also has an `isEndingLayout` property to indicate if the object is being destroyed because it's the end of a layout, or destroyed for other reasons.

## Instance APIs

---

**addEventListener(type, func, capture)**

**removeEventListener(type, func, capture)**

Add or remove an event handler for a particular type of event fired by an addon's script interface. An event object is passed as a parameter to the handler function. See instance event for standard event object properties. For information on which events are fired by specific addons and which additional event object properties are available, see the documentation on each addon's script interfaces.

### dispatchEvent(e)

Dispatch an event, firing any handler functions that have been added for the event type. You can use `new C3.Event(eventName, isCancellable)` to create an event object that can be dispatched (e.g. `new C3.Event("click", true)`), and add any extra properties relevant to your event to that object. This can also be used by the addon SDK to cause your addon to fire an event in the script interface, e.g.:

```
const e = new C3.Event("click", true);
this.GetScriptInterface().dispatchEvent(e);
```

### runtime

A reference back to the IRuntime interface. (This is particularly useful when subclassing instances, since in a custom class's methods you can always refer to the runtime with `this.runtime`.)

### objectType

The IObjectType interface for this instance's object type.

### plugin

The IPlugin interface (or derivative) for this instance's plugin.

### instVars

If the object has any instance variables, they can be accessed by named properties under this property. For example if an object has an instance variable named *health*, it can be set and retrieved using `instance.instVars.health`. Note if the object has no instance variables, the instance won't have an `instVars` property at all.

> *In some cases, instance variables may have names that aren't valid JavaScript identifiers. In this case you can use the string property syntax, e.g.* `instance.instVars["health"]`.

> *You don't have to use instance variables to add custom properties to instances. In JavaScript you can simply assign new properties to existing objects, or use instance subclassing to use your own custom class with your own properties and methods.*

### behaviors

If the object has any behaviors, they can be accessed by named properties under this property. For example if an object has a behavior named *Bullet*, it can be accessed using `instance.behaviors.Bullet`. Each behavior has its own properties and methods, which can be found in the Behavior interfaces reference section. Note if the object has no behaviors, the instance won't have a `behaviors` property at all.

> *In some cases, behaviors may have names that aren't valid JavaScript identifiers. In this case you can use the string property syntax, e.g.* `instance.behaviors["8Direction"]`.

---

**uid**

The unique ID of this instance, as a number. Note instances can be looked up by their UID using the runtime `getInstanceByUid()` method.

---

**iid**

The index ID (IID) of this instance, as a number. See instances for more details.

---

**templateName**

Read-only string of the name of the template used to create this instance, or an empty string if no template was used.

---

**destroy()**

Destroy the instance, removing it and releasing any memory associated with it.

> *Do not make any further calls or access any properties after the* `destroy()` *call. The instance is no longer valid and any attempts to use it may throw exceptions.*

---

**getOtherContainerInstances()**

Return an array of `IInstance` (or derivatives) representing other instances in the same container as this instance. This excludes the instance the method is called on.

---

***otherContainerInstances()**

Iterates over `IInstance` (or derivatives) representing other instances in the same container as this instance. This excludes the instance the method is called on.

---

**dt**

Return delta-time according to the object's own timescale. See Delta-time and framerate independence for more information.

### timeScale

### restoreTimeScale()

The `timeScale` property sets or gets the current instance-specific time scale, e.g. 1.0 for normal speed, 2.0 for twice as fast, etc. Note that once set, the instance uses its own time scale instead of runtime time scale, e.g. allowing an instance to keep moving when the runtime time scale is 0. Calling the `restoreTimeScale()` method will then switch it back to following the runtime time scale.

### signal(tag)

Triggers *On signal* for this instance, and resolves any promise returned by `waitForSignal()`. Any events waiting for a signal with the given tag will resume when all picked instances are signalled.

### waitForSignal(tag)

Returns a Promise that resolves when the given tag is signalled for this instance. It may be signalled by either the script API or an event sheet.

### hasTags(tagsArray)

Pass in an array of strings and check if the instance has all the passed in tags or not. Returns true or false.

### setAllTags(tagsSet)

Pass in a Set of tags and set them all to the instance. Replaces all existing tags.

### getAllTags()

Return a Set with all the tags an instance has.

# OBJECT CLASS SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iobjectclass

The `IObjectClass` script interface represents an object class in the project, which can be either an object type or a family. It is the base class for both IObjectType and IFamily.

For the most part, object types and families work the same and can be treated equally, hence most APIs being available on the `IObjectClass` base class. However there are some differences, such as only being able to create an instance of a specific object type, hence the `createInstance()` method only being available for `IObjectType`.

## Getting an IObjectClass

References to the project's object classes are typically accessed through the IRuntime interface `objects` property. This covers both object types and families, hence everything under that property is an `IObjectClass`. For example `runtime.objects.Sprite` would refer to the `IObjectType` interface for the *Sprite* object type, and `runtime.objects.Family1` would refer to the `IFamily` interface for the family *Family1*, assuming they were added to the project.

> *Try not to confuse object classes with object instances. A common mistake is to try to use something like `runtime.objects.Sprite.x` to get the X co-ordinate of a Sprite instance. However `runtime.objects.Sprite` is an IObjectClass, which does not have a position. First add another call to get an instance before trying to read instance properties, for example `runtime.objects.Sprite.getFirstInstance().x`.*

## Object class events

The following events can be listened for using the `addEventListener` method.

**"instancecreate"**

Fired whenever a new instance belonging to this object class is created. The event object has an `instance` property referring to the IInstance (or derivative) that was created.

**"hierarchyready"**

Fired for the root instance in a hierarchy after all instances have finished creating - see the IWorldInstance event for more details. When fired on an IObjectClass, the event object has an `instance` property referring to the IWorldInstance (or derivative) that was created.

**"instancedestroy"**

Fired whenever any instance belonging to this object class is destroyed. After this event, all references to the instance are now invalid, so any remaining references to the instance should be removed or cleared to `null` in this event. Accessing an instance after it is destroyed will throw exceptions or return invalid data. The event object has an `instance` property referring to the IInstance (or derivative) that was destroyed. It also has an `isEndingLayout` property to indicate if the object is being destroyed because it's the end of a layout, or destroyed for other reasons.

## Object class APIs

**runtime**

A reference back to the IRuntime interface.

**plugin**

A reference to the IPlugin interface (or derivative) this object class was created from.

**name**

A read-only string of the object class's name.

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for an event. See *Object class events* above for more information.

**getAllInstances()**

Return an array of all instances of this object class.

**getFirstInstance()**

Return the first instance in the array returned by `getAllInstances()`, or `null` if no instances exist.

***instances()**

Iterates over all the object class's instances.

**getPickedInstances()**

Return an array of instances that have been picked by the event's conditions. This is only useful with scripts in event sheets.

### getFirstPickedInstance()

Return the first instance that has been picked by the event's conditions, or `null` if none. This is only useful with scripts in event sheets.

### *pickedInstances()

Iterates over the instances that have been picked by the event's conditions. This is only useful with scripts in event sheets.

### getPairedInstance(inst)

Get an instance of this object class at the same index (IID) as the given instance. If there are fewer instances of this object class than the given instance's object class, the index is allowed to wrap around. This allows identifying the same instance that the event system would pair with the given instance.

# IOBJECTTYPE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iobjecttype

The `IObjectType` script interface represents an object type in the project, e.g. a Sprite object type. An object type can have multiple instances created, which are represented by the IWorldInstance script interface if it appears in a layout, otherwise the IInstance interface.

`IObjectType` derives from IObjectClass which is a base class shared with families (IFamily).

## Getting an IObjectType

References to the project's object types are typically accessed through the IRuntime interface `objects` property. For example `runtime.objects.Sprite` would refer to the `IObjectType` interface for the *Sprite* object type, assuming one was added to the project.

> *Try not to confuse object classes with object instances. A common mistake is to try to use something like `runtime.objects.Sprite.x` to get the X co-ordinate of a Sprite instance. However `runtime.objects.Sprite` is an IObjectType, which does not have a position. First add another call to get an instance before trying to read instance properties, for example `runtime.objects.Sprite.getFirstInstance().x`.*

## Examples

The following examples demonstrate using some features of `IObjectType`:

- Instance destroy event demonstrates the use of the `"instancedestroy"` event
- Iterating instances demonstrates the use of the `instances()` iterator to modify all instances

## Object type APIs

**setInstanceClass(Class)**

Set a custom class to be used to represent instances of this object type. The class must derive from the default class. This can only be called in `runOnStartup`, before any instances have been created. For more information see the guide on subclassing instances.

**createInstance(layerNameOrIndex, x, y, createHierarchy, template)**

Create a new instance of the object type at a position. The layer to create on is specified either by a case-insensitive string of the layer name or its zero-based index. The position is

given in layout co-ordinates. If `createHierarchy` is true, all children of the created instance in the scene-graph hierarchy will also be created automatically with their connections in place. If `template` is a valid template name then the new instance will be based on the template rather than an arbitrary instance. The return value is the script interface for the created instance.

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

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

---

**getAllFamilies()**

Return an array of IFamily listing all the families this object type belongs to.

---

***families()**

Iterates over all the families this object type belongs to (as `IFamily`).

---

**isInFamily(family)**

Returns a boolean indicating if this object type belongs to the specified `IFamily`.

# IPLUGIN SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iplugin

The `IPlugin` script interface represents a plugin, such as the Sprite plugin. The plugin only exists once in the project: if there are multiple Sprite objects in a project, then each object type is represented with a different IObjectClass.

## IPlugin APIs

**runtime**

A reference back to the IRuntime interface.

**id**

A read-only string with a unique identifier for this plugin as specified by the plugin developer.

**isSingleGlobal**

A read-only boolean indicating whether this plugin is a 'single global' type, meaning it is added to the project once and then is available globally, such as with the Mouse object.

**getSingleGlobalObjectType()**

**getSingleGlobalInstance()**

These calls are only permitted when `isSingleGlobal` is true. These return the single IObjectClass and IInstance respectively that have been created for this plugin.

**isWorldType**

A read-only boolean indicating whether this plugin is a world type, meaning it appears in layouts.

**isHTMLElementType**

A read-only boolean indicating whether this plugin creates a HTML element.

**isRotatable**

A read-only boolean indicating whether instances of the plugin may be rotated.

**hasEffects**

A read-only boolean indicating whether the plugin may use effects.

---

**is3d**

A read-only boolean indicating whether the plugin is 3D (i.e. has depth on the Z axis).

---

**supportsHierarchies**

A read-only boolean indicating whether instances of the plugin may be used in hierarchies.

---

**supportsMesh**

A read-only boolean indicating whether instances of the plugin may use the mesh distortion feature.

---

**static getByConstructor(ctor)**

Return an `IPlugin` from its constructor in the `C3.Plugins` namespace, or `null` if it does not exist. For example `IPlugin.getByConstructor(C3.Plugins.Audio)` will return the `IPlugin` interface representing the Audio object, or `null` if the plugin is not used in the project.

# WORLDINSTANCE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/object-interfaces/iworldinstance

---

The `IWorldInstance` script interface represents a single instance of an object type (represented by IObjectClass) that appears in a layout. It derives from the IInstance script interface.

Many objects return a more specific class deriving from `IInstance` or `IWorldInstance` to add APIs specific to the plugin. See the Plugin interfaces reference for more information.

## Getting an IWorldInstance

Instances are typically accessed through IObjectClass methods like `getFirstInstance()`. For example, `runtime.objects.Sprite.getFirstInstance()` will return the first instance of the Sprite object type.

> *Try not to confuse object classes with object instances. A common mistake is to try to use something like `runtime.objects.Sprite.x` to get the X co-ordinate of a Sprite instance. However `runtime.objects.Sprite` is an IObjectClass, which does not have a position. First add another call to get an instance before trying to read instance properties, for example `runtime.objects.Sprite.getFirstInstance().x`.*

## World instance events

The following events can be listened for on any `IWorldInstance` using the `addEventListener` method. See instance event for standard event properties. Note many more kinds of addon-specific events can be fired. See the documentation on each addon's script interfaces for more information.

---

**"hierarchyready"**

Fired for the root instance in a hierarchy after all instances have finished creating. During creation of a hierarchy it is uncertain whether other instances in the hierarchy have been created yet, which can sometimes complicate initializing hierarchies. When this event fires all instances in the hierarchy have been created, including triggering *On created* in event sheets, and so it is a suitable time to perform initialization of an entire hierarchy. As this only fires for the root instance, if you wish to iterate the rest of the instances in the hierarchy, use the `allInstances()` generator method of the root instance.

## General APIs

**layout**

An ILayout interface representing the layout the instance is on.

**layer**

An ILayer interface representing the layer the instance is on.

**x**

**y**

**setPosition(x, y)**

**getPosition()**

The position of this instance, in layout co-ordinates. The methods allow setting or getting both co-ordinates at the same time.

**offsetPosition(dx, dy)**

Adjust the position by adding `dx` to the X co-ordinate and `dy` to the Y co-ordinate.

**zElevation**

The Z elevation of the instance, relative to the layer it is on.

**totalZElevation**

A read-only value indicating the Z elevation of the instance including its layer's Z elevation.

**originX**

**originY**

**setOrigin(originX, originY)**

**getOrigin()**

The current origin point of this instance, represented as a normalized position in the range [0, 1]. For example the origin (0, 0) is in the top-left corner, and (0.5, 0.5) is in the middle regardless of the size of the object. The methods allow setting or getting both co-ordinates at the same time.

> *With Sprite objects, changing the animation frame will also update the origin according to the origin placement in the Animations Editor.*

**width**

**height**

### setSize(width, height)

### getSize()

The size of this instance, in layout co-ordinates. The methods allow setting or getting both values at the same time.

---

### angle

The angle of the instance in radians. If this is changed, `angleDegrees` updates accordingly.

---

### angleDegrees

The angle of the instance in degrees. If this is changed, `angle` updates accordingly.

---

### getBoundingBox(ignoreMesh = false)

Return a DOMRect representing the axis-aligned bounding box of the instance in layout co-ordinates. By default this takes in to account any changes from the mesh distortion feature - specifying `false` for the `ignoreMesh` parameter will return a bounding box as if the object did not use mesh distortion.

> *This returns a copy of the bounding box. The returned DOMRect does not change if the instance changes, nor does changing the DOMRect affect the instance.*

---

### getBoundingQuad(ignoreMesh = false)

Return a DOMQuad representing the bounding quad of the instance in layout co-ordinates. This is always a rectangle, but unlike the bounding box can represent rotation. By default this takes in to account any changes from the mesh distortion feature - specifying `false` for the `ignoreMesh` parameter will return a bounding quad as if the object did not use mesh distortion.

> *This returns a copy of the bounding quad. The returned DOMQuad does not change if the instance changes, nor does changing the DOMQuad affect the instance.*

---

### isVisible

A boolean indicating whether the instance is visible in the layout.

---

### isOnScreen()

Returns true if any part of the object's bounding box is within the screen area (performing the same check as the *Is on-screen* condition). This is not affected by the object's visibility or opacity.

**opacity**

The opacity of the instance, as a floating point number in the range [0, 1], where 0 is fully transparent and 1 is fully opaque.

**colorRgb**

An array with 3 elements specifying the red, green and blue color filter of the instance, with color values as floats in the 0-1 range.

**blendMode**

A string indicating the current blend mode of the instance, controlling how it draws over the background. This must be one of `"normal"`, `"additive"`, `"copy"`, `"destination-over"`, `"source-in"`, `"destination-in"`, `"source-out"`, `"destination-out"`, `"source-atop"`, `"destination-atop"`.

**effects**

An array of IEffectInstance representing the effect parameters for each effect on the instance.

## Z order APIs

**moveToTop()**

**moveToBottom()**

Move the instance to the top or the bottom of its current layer in the Z order.

**moveToLayer(layer)**

Move the instance to the top of a different layer given by its ILayer.

**moveAdjacentToInstance(other, isAfter)**

Move the instance adjacent to `other` (another `IWorldInstance`) in the Z order. If necessary this also moves the instance to the same layer as `other`. If `isAfter` is true, it moves it just above the given instance, else just below.

**zIndex**

A read-only integer indicating the instance's current index in the Z order on its current layer, starting at 0 for the back of the current layer, and increasing as it moves to the front.

## Collision APIs

See also the ICollisionEngine interface for more collision APIs.

---

**isCollisionEnabled**

Set or get a boolean indicating whether collisions are enabled for this instance. If disabled, the instance will always fail all overlap or collision checks.

---

**containsPoint(x, y)**

Test if a point intersects this instance, using its collision polygon if any, and return a boolean indicating if the point is inside the instance's collision area.

---

**testOverlap(wi)**

Test if this instance overlaps another world instance given by an `IWorldInstance`, returning `true` if they overlap, else `false`. This uses the object's collision polygons if any. If either instance has collisions disabled, this will always return `false`.

---

**testOverlapSolid()**

Test if this instance overlaps any instance with the Solid behavior. This returns the instance interface class for the first instance with the solid behavior that was found to overlap this instance, or `null` if none. This uses the object's collision polygons if any and respects solid collision filtering.

> *The return value of this method is truthy when an overlap is found and falsey when not, so this can be used directly in an `if` statement.*

## Mesh distortion APIs

---

**createMesh(hsize, vsize)**

Create a mesh for deforming the appearance of the object with the given number of mesh points horizontally and vertically. The minimum size is 2.

---

**releaseMesh()**

Releases any mesh that has been created, reverting back to default rendering of the object with no mesh distortion. Ignored if no mesh created.

---

**setMeshPoint(col, row, opts)**

Alter a given point in a created mesh given by its zero-based column and row. `opts` is an object that may specify the following properties:

- `mode`: a string of `"absolute"` (default) or `"relative"`, determining how to interpret the `x`, `y`, `u` and `v` options.

- `x` and `y`: the mesh point position offset, in normalized co-ordinates [0, 1] across the object size. These are allowed to go outside the object bounds. In relative mode these are added to the mesh point's current position.

- `u` and `v`: the texture co-ordinate for the mesh point, in normalized co-ordinates [0, 1]. These are not allowed to go outside the object bounds. These can be omitted, or in absolute mode be set to -1, to indicate not to change the texture co-ordinate from the default.

- `zElevation`: the Z elevation of the mesh point, allowing for distortion in 3D. Similarly to Z elevation of entire objects, this moves the mesh point up and down on the Z axis.

---

**getMeshPoint(col, row)**

Return an object describing the currently set mesh point at its zero-based column and row. The returned object has the same properties as the `opts` argument of `setMeshPoint()` uses, except for `mode`. The returned values are all absolute values (as relative mode is only relevant when applying changes).

---

**getMeshSize()**

Return the size of the mesh as `[hsize, vsize]` (corresponding to the size passed to `createMesh()`) if one is created. If no mesh has been created, returns `[0, 0]`.

## Scene graph APIs

---

**getParent()**

Return the parent `IWorldInstance` of this instance in the scene graph hierarchy if any, else `null`.

---

**getTopParent()**

Return the top parent of this instance in the scene graph hierarchy (which by definition has no parent itself) if any, else `null`.

---

***parents()**

A generator method that can be used to iterate all the instance's parents, up to the top parent.

### getChildCount()

Returns the number of children that have been added to this instance in the scene graph hierarchy.

### getChildAt(index)

Of the children that have been added to this instance, return the child instance at the given zero-based index. If the index is out of bounds, returns `null`.

### *children()

A generator method that can be used to iterate all the instance's added children.

### *allChildren()

A generator method that can be used to iterate all the instance's children recursively, i.e. including children of children, down to the bottom of the scene graph hierarchy.

### addChild(wi, opts)

Add another world instance given by an `IWorldInstance` as a child of this instance in the scene graph hierarchy. This instance becomes its parent in the scene graph hierarchy. The child will move, scale and rotate with this instance according to the provided options specified in the object `opts`, which supports the following properties:

- `transformX`: move the child with this instance's X position
- `transformY`: move the child with this instance's Y position
- `transformWidth`: scale the child with this instance's width
- `transformHeight`: scale the child with this instance's height
- `transformAngle`: rotate the child with this instance's angle
- `transformZElevation`: move the child with this instance's Z elevation
- `transformOpacity`: change the child's opacity according to the parent's opacity
- `transformVisibility`: make the child invisible if the parent is also invisible

- `destroyWithParent`: automatically destroy the child if this instance is destroyed

Each option is a boolean which defaults to `false` if omitted, so only `true` properties need to be specified.

> *Instances can only have one parent. If the given instance is already added as a child of something else, this method will have no effect.*

---

### getHierarchyOpts()

Return an object with properties representing the options specified for this child when it was added to its parent. The returned object has the same properties as the `opts` argument of `addChild()` uses, such as `transformX`, with boolean values indicating whether they are enabled. Therefore the returned object can be passed directly to another call to `addChild()` to re-use the same options.

---

### removeChild(wi)

Remove an existing child given by an `IWorldInstance` that was previously added with `addChild()`. The child is detached from the scene graph hierarchy and this instance will no longer act as its parent. The removed child still keeps its own children, if it has any.

---

### removeFromParent()

Shorthand method for `wi.getParent().removeChild(wi)`, i.e. removes this instance from its parent if it has any. If the instance has no parent, the method has no effect.

# BEHAVIOR INSTANCE EVENT

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/behavior-instance-event

Events fired on IBehaviorInstance (or its derivatives) pass an event object as a parameter to the handler function, and this event object has the following standard properties. Each type of event may add other properties - refer to the documentation for each event to identify any further properties that are available.

## Standard instance event properties

**instance**

A reference to the IInstance (or derivative) associated with the behavior instance which fired the event.

**behaviorInstance**

A reference to the `IBehaviorInstance` which fired the event.

# ASSETMANAGER SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/iassetmanager

---

The `IAssetManager` interface provides access to the project's assets, such as audio files and other project files added to your project. It is typically accessed via `runtime.assets`.

On most modern platforms, assets can be directly retrieved as if over the network with standard APIs like `fetch()` or `XMLHttpRequest`, even in non-web export platforms like mobile or desktop apps. However in some circumstances, generally with unusual export options like Playable Ads or with legacy projects, it may be necessary to use Construct's IAssetManager interface to fetch resources, as it implements workarounds that prevent standard fetches working in those environments.

## AssetManager APIs

---

**async fetchText(url)**

**async fetchJson(url)**

**async fetchBlob(url)**

**async fetchArrayBuffer(url)**

Retrieve the contents of a given URL as a string, JSON object, Blob or ArrayBuffer. Returns a promise that resolves when the resource has been loaded.

---

**async getProjectFileUrl(url)**

Retrieve a URL that can be fetched directly for a given resource. Returns a promise that resolves to a string with a URL that may be the same as the original URL, or a different URL (e.g. `blob:` URL) if direct fetching is not supported. This is intended for using with local files where the other fetch methods are not appropriate, such as assigning the `src` attribute of a video.

---

**async getMediaFileUrl(url)**

As with `getProjectFileUrl` but for sound and music files, which are exported to a *media* subfolder.

---

**mediaFolder**

A string of the subfolder media files are in, including sound and music files. In preview this is an empty string, and after export it is the media subfolder followed by a forward slash, e.g.

`"media/"`.

---

### projectFileList

A read-only array with the list of project files in this project at the time of preview or export. Files in the Sounds, Music, Videos, Fonts, Icons & Screenshots and Files folders are included. Each entry is a an object with the properties `name` being the relative path to the file, e.g. `subfolder/mydata.json`, and `size` which is the file size in bytes.

> *This file list only reflects the state of the project at the time it was previewed or exported. If you export the project and then change some of the files, this list will not update to reflect the change. If you need a list that updates to post-export changes, consider another approach, such as using File System to list folder contents on desktop, or a separate data file to list files in which can also be updated after export.*

async loadScripts(...urls)
Fetch and run the JavaScript files at the given URLs. This can load scripts in the *Files* folder of the Project Bar, none of which are automatically loaded by Construct. When loading multiple scripts, they will run in the order they are provided, e.g. `loadScripts("script1.js", "script2.js")` will always run script1.js first and script2.js second. For best efficiency, try to load all the scripts you need in a single call, rather than repeated calls.

---

### async compileWebAssembly(url)

Fetch and compile a WebAssembly.Module from the given URL, which is typically a .wasm file. This uses streaming compilation where supported. Note this does not instantiate the module, which needs to be done before any calls can be made. Pass the module resulting from this call to WebAssembly.instantiate() to get a WebAssembly.Instance from the module.

---

### async loadStyleSheet(url)

Fetch a stylesheet at the given URL and attach it to the current document, applying its styles. Returns a Promise that resolves when the stylesheet has been applied to the document.

# ICOLLISIONENGINE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/icollisionengine

The `ICollisionEngine` interface provides access to Construct's collision engine, such as to identify if two objects are intersecting. It is typically accessed via `runtime.collisions`.

## Collision APIs

### runtime

A reference back to the IRuntime script interface.

### testOverlap(instA, instB)

Returns a boolean that indicates if two instances, which must be derivatives of IWorldInstance, are overlapping at their current positions.

```
// Example code
if (runtime.collisions.testOverlap(instA, instB))
{
    console.log("Collision found!");
}
```

### testOverlapAny(inst, iterable)

Test if an IWorldInstance is overlapping any of the other `IWorldInstance`s provided by an iterable (which can be any kind of iterable object, such as an array). If an overlap is found, it returns the `IWorldInstance` of the first instance in `iterable` that overlaps `inst`. If no overlap is found with any of the provided instances, it returns `null`.

> *Note the return value can also be used as truthy or falsey, such as in an `if` statement.*

### testOverlapSolid(inst)

Test if an IWorldInstance is overlapping any other instance with the Solid behavior. If an overlap is found, it returns the `IWorldInstance` of the first found solid instance that overlaps `inst`. If no overlap is found it returns `null`.

*Note the return value can also be used as truthy or falsey, such as in an `if` statement.*

---

### setCollisionCellSize(width, height)

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. The collision cell size will also affect how many instances are returned by `getCollisionCandidates()`.

---

### getCollisionCellSize()

Returns `[width, height]` indicating the current collision cell size.

---

### getCollisionCandidates(iObjectClasses, domRect)

Efficiently retrieve only instances of the given object classes that are near the specified rectangular area in the layout. This uses Construct's "collision cells" optimization and allows substantially reducing the number of collision checks that need to be performed. For example instead of checking for collisions against all instances of a given object class, which could involve thousands of instances distributed across a large layout, this method allows efficiently retrieving only a smaller number of instances in the area which are possible candidates for collision checking. The parameters work as follows:

- `iObjectClasses` is either an IObjectClass, or an array of IObjectClass, specifying the object classes of interest, e.g. `runtime.objects.Enemy`. Only instances belonging to these object classes will be returned. Families can also be specified and all instances belonging to any of the object types in the family will be included.

- `domRect` is a DOMRect specifying a rectangular area in the layout. Instances near this area will be returned. All instances inside the specified rectangle will be returned, but some instances near to but outside the rectangle may also be returned, as the area checked is done at the resolution of the collision cell size; however instances far away from the area will not be included, which is the main purpose of this method.

The method returns an array of IWorldInstance with the instances near the specified area. Note also that due to the algorithm used, the returned array of instances **may include duplicates** - i.e. the same instance may appear more than once in the array. These can be filtered by creating a Set with the returned array, as a Set only stores unique items; however this will add a performance overhead. Eliminating duplicates may not be necessary depending on the algorithm used. For example if the aim is to identify any collision and then destroy the object, it does not matter if the same check is performed twice. However if the

aim is to add to the score for every overlap, then duplicates must be eliminated for the correct intended result.

# INSTANCE EVENTS

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/instance-event

Events fired on IInstance (or its derivatives) pass an event object as a parameter to the handler function, and this event object has the following standard properties. Each type of event may add other properties - refer to the documentation for each event to identify any further properties that are available.

## Standard instance event properties

**instance**

A reference to the `IInstance` (or derivative) which fired the event.

# PLATFORM INFO SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/iplatforminfo

The `IPlatformInfo` interface provides access to Construct's detected platform details, such as the browser, operating system, and export type. It provides similar information to the Platform info plugin, but note that plugin does not need to be in the project to use this interface. It is typically accessed by the IRuntime `platformInfo` property.

```
// Example: log detected operating system to console on startup
runOnStartup(async runtime =>
{
	console.log("The detected OS is: " + runtime.platformInfo.os);
});
```

## Platform info APIs

### isMobile

A read-only boolean indicating whether Construct detects the current device as "mobile".

> *This detection may be unreliable. See the warning for the Is on mobile condition in the Platform Info manual entry.*

### os

A read-only string with the operating system Construct has detected, which is one of `"windows"`, `"macos"`, `"linux"`, `"chrome-os"`, `"android"`, `"ios"` or `"unknown"`.

### osVersion

A read-only string with the operating system version Construct has detected, or the string `"(unknown)"` if it failed to identify a version.

### browser

A read-only string with the browser Construct has detected, which is one of `"chrome"`, `"chromium"`, `"edge"`, `"opera"`, `"firefox"`, `"safari"` or `"unknown"`.

*Consider using feature detection or referring to* `browserEngine` *rather than relying on the identified browser.*

---

**browserVersion**

A read-only string with the browser version Construct has detected, or the string `" (unknown)"` if it failed to identify a version.

---

**browserEngine**

A read-only string with the browser engine Construct has detected, which is one of `"chromium"`, `"gecko"`, `"webkit"`, or `"unknown"` if failed to identify any.

---

**exportType**

A read-only string indicating the export option used to export the project, which is one of `"preview"` (when previewing prior to export), `"html5"`, `"scirra-arcade"`, `"cordova-android"`, `"cordova-ios"`, `"windows-webview2"`, `"macos-wkwebview"`, `"linux-cef"`, `"xbox-uwp-webview2"`, `"instant-games"` or `"playable-ad"`.

---

**renderer**

A read-only string indicating the graphics rendering technology in use. This can be `"webgl1"`, `"webgl2"` or `"webgpu"`. It can also have a `-software` suffix if a "major performance caveat" is detected, which normally indicates slower software rendering, typically due to unreliable GPU drivers. For example, software-rendered WebGL 1 support would return `"webgl1-software"`.

---

**rendererDetail**

A read-only string describing system-specific detail about the graphics hardware in use. This typically describes the GPU manufacturer and model name, and sometimes some hardware capabilities.

---

**canvasClientX**

**canvasClientY**

Read-only numbers with the position of the main display canvas in CSS pixels. In other words these are the CSS co-ordinates of the top-left point of the main display canvas relative to the page.

---

**canvasCssWidth**

**canvasCssHeight**

Read-only numbers with the size of the main display canvas in CSS pixels. This does not correspond exactly to device (physical display) pixels, but is the appropriate size from a web design perspective in `px` units.

---

**canvasDeviceWidth**

**canvasDeviceHeight**

Read-only numbers with the size of the main display canvas in device (physical display) pixels. Unlike the CSS size, this size reflects the number of actual pixels used in the display.

---

**devicePixelRatio**

Return the devicePixelRatio property.

> *This property can also be read in worker mode, which means it may be preferable to read this rather than the `devicePixelRatio` global.*

# STORAGE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/istorage

The `IStorage` interface provides access to storage for the project. It essentially wraps a simple key-value storage engine based on IndexedDB. This means any data that can be stored in IndexedDB can be stored with these methods, such as numbers, strings, Blobs, etc. It is typically accessed by the IRuntime `storage` property.

This interface accesses the same storage as the Local Storage plugin. Therefore an item stored from the event sheet can also be read from script, and vice versa. Note however that Construct expressions can only be strings or numbers, so if a script stores a different type it cannot be used in the event sheet.

As with the Local Storage plugin, the storage is unique to the specific project. It is not shared with any other projects or other website storage, even on the same origin.

## Examples

See the Local storage - script example for a demonstration of using these storage APIs to track a high score.

## Storage APIs

### async getItem(key)

Read an item from storage. Returns a promise that resolves to the value of the item if it exists in storage, else `null` if the item does not exist in storage.

> *If an error occurs when reading from storage, this resolves with `null` instead of throwing an exception.*

### async setItem(key, value)

Write an item to storage. Returns a promise that resolves when the write has completed.

> *If the write fails - most commonly due to using up all available storage space - the promise will reject. To ensure this does not crash the game, ensure calls are in a `try...catch` block.*

### async removeItem(key)

Delete an item from storage. Returns a promise that resolves when the removal has completed.

---

**async clear()**

Delete all items from storage. Returns a promise that resolves when the clear has completed.

---

**async keys()**

Retrieve a list of all keys in storage. Returns a promise that resolves to an array of key names.

# TIMELINE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/itimelinestate

The `ITimelineState` represents an actively running timeline. It derives from ITimelineStateBase which implements APIs in common between timelines and tweens. Many general playback APIs are part of `ITimelineStateBase`; the `ITimelineState` interface only provides APIs specific to timelines that do not also apply to tweens.

Timelines can be created using the `play()` method on the Timeline Controller script interface.

> *Once a timeline finishes, this interface is destroyed and all its properties will throw exceptions upon access. The only exception to this is the `isReleased` property which provides a read-only boolean that indicates if the interface has been released and is now invalid.*

## Timeline APIs

**name**

A read-only string with the name of the timeline.

# ITIMELINESTATEBASE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/itimelinestatebase

The `ITimelineStateBase` is the base class of ITimelineState and ITweenState, allowing common APIs to be shared between timelines and tweens, as tweens are a kind of temporary timeline. Unless otherwise stated, where this documentation refers to a timeline, it also applies to a tween.

> *Once a timeline finishes, this interface is destroyed and all its properties will throw exceptions upon access. The only exception to this is the `isReleased` property which provides a read-only boolean that indicates if the interface has been released and is now invalid.*

## ITimelineStateBase APIs

**finished**

A promise that resolves when the timeline finishes playing. This can be awaited in order to wait until the timeline finishes before proceeding to do something else.

**pause()**

**resume()**

Pause or resume playback of the timeline.

**isPlaying**

**isPaused**

Read-only booleans indicating whether playback is active or paused.

**time**

Set or get the current playback time of the timeline in seconds.

**totalTime**

Set or get the total time (i.e. the duration) of the timeline in seconds.

**progress**

A read-only number representing the playback progress from 0 to 1 (i.e. the time divided by the total time).

**isLooping**

Set or get a boolean indicating whether playback will repeat when it finishes.

**isPingPong**

Set or get a boolean indicating whether the playback direction will alternately reverse when repeating.

**playbackRate**

Set or get the speed of playback as a multiplier, e.g. 1 is normal speed, 2 is twice as fast, etc.

**tags**

A read-only array of strings representing the tags for this timeline.

**hasTags(tags)**

Return a boolean indicating if the timeline matches all the provided tags, given by a space-separated string.

**isReleased**

A read-only boolean indicating if the interface was released, which happens after the timeline finishes. Once released accessing any other property (apart from this one) will throw an exception, as the underlying timeline state no longer exists.

# TWEEN SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/interfaces/itweenstate

The `ITweenState` interface represents an actively running tween. It derives from ITimelineStateBase which implements APIs in common between timelines and tweens, as a tween is a kind of temporary timeline. Many general playback APIs are part of `ITimelineStateBase`; the `ITweenState` interface only provides APIs specific to tweens that do not also apply to timelines.

Tweens can be created using the `startTween()` method on the Tween behavior script interface.

> *Once a tween is released, this interface is destroyed and all its properties will throw exceptions upon access. The only exception to this is the `isReleased` property (inherited from `ITimelineState`) which provides a read-only boolean that indicates if the interface has been released and is now invalid.*

## Tween APIs

**stop()**

Stops playback and immediately ends the tween. The interface is released in this call and so it cannot be used any further after this call.

**instance**

A read-only property with a reference to the IWorldInstance the tween is running on.

**isDestroyOnComplete**

Set or get a boolean indicating whether the corresponding instance will be automatically destroyed once the tween finishes.

**value**

A read-only number providing the current value of a value tween.

**setEase(easeName)**

Set the ease function used for the tween by a string of its name. Refer to the Tween behavior script interface for a list of valid built-in ease names, or use the name of a custom ease.

**released**

A promise that resolves when the tween is destroyed. Afterwards the rest of the API can no longer be used except for `isReleased` property (inherited from `ITimelineState`) which provides a read-only boolean that indicates if the interface has been released and is now invalid.

A promise that resolves when the timeline finishes playing. This can be awaited in order to wait until the timeline finishes before proceeding to do something else.

# 3D CAMERA SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/3d-camera

The `I3DCameraObjectType` interface derives from IObjectClass to add APIs specific to the 3D Camera plugin.

Note this class derives from the object class interface, not the instance interface. Its default name is not a valid JavaScript identifier as it starts with a number, meaning it needs to be accessed with the syntax `runtime.objects["3DCamera"]`. You may wish to rename the object to a valid JavaScript identifier such as *Camera3D*, so that it can be accessed more conveniently with the syntax `runtime.objects.Camera3D`.

## 3D Camera APIs

### lookAtPosition(camX, camY, camZ, lookX, lookY, lookZ, upX, upY, upZ)

Set the position and orientation of the 3D Camera using a camera position, a position for the camera to look towards, and an up vector. The camera and look-at positions are given as 3D co-ordinates. The up vector is a 3D vector specifying which way is up, as this is also necessary to determine how to orient the camera towards the look-at position. The default up vector is (0, 1, 0), i.e. up is the positive direction on the Y axis, suitable for a top-down view.

### lookParallelToLayout(camX, camY, camZ, lookAngle)

Set the position and orientation of the 3D Camera using a camera position and a camera angle in radians. Whereas the default view is top-down, this sets a camera position looking along the layout, such that the layout appears as the floor at the bottom of the screen. This is a shortcut for using the *Look at position* action looking towards a 2D angle with an up vector of (0, 0, 1).

### restore2DCamera()

Restore the camera to its default 2D behavior, using the standard scrolling features to move the view.

### moveAlongLayoutAxis(distance, axis, which)

Move the camera position, the look position, or both, a distance along an axis relative to the layout. The distance can be negative to move in the opposite direction to the given axis. *axis* must be a string of `"x"`, `"y"` or `"z"`. *which* must be a string of `"camera"`, `"look"` or `"both"`.

> *Note the scale on the Z axis is different to the X and Y axes.*

---

### moveAlongCameraAxis(distance, axis, which)

Move the camera position, the look position, or both, a distance along an axis relative to the current camera orientation. The distance can be negative to move in the opposite direction to the given axis. *axis* must be a string of "forward", "up" or "right". *which* must be a string of `"camera"`, `"look"` or `"both"`.

> *Note the scale on the Z axis is different to the X and Y axes.*

---

### rotateCamera(rotateX, rotateY, minPolar, maxPolar)

Moves the camera look-at position according to X and Y rotations in radians. Note that a 3D Camera must first have been enabled using the `lookAtPosition()` or `lookParallelToLayout()` methods, since these also define the starting orientation for the camera to rotate around. Typically the rotation values will be provided by the mouse movement for "mouse look". The min/max polar values are also in radians and limit the Y rotation, stopping the camera rotating too close to directly upwards or downwards.

---

### getCameraPosition()

Return an array of `[x, y, z]` with the current 3D position of the camera.

---

### getLookPosition()

Return an array of `[x, y, z]` with the current 3D position of the position the camera is pointing at.

---

### getLookVector()

Return an array of `[x, y, z]` with the current vector of the direction the camera is pointing in, including camera rotation.

---

### getForwardVector()

Return an array of `[x, y, z]` with a 3D unit vector pointing in the direction of the camera.

> *Note this does not include camera rotation. Use `getLookVector()` to get the vector of the direction the camera is pointing in including camera rotation.*

---

### getRightVector()

Return an array of `[x, y, z]` with a 3D unit vector pointing to the right of the camera, perpendicular to the forward vector.

---

**getUpVector()**

Return an array of `[x, y, z]` with the camera up vector, which helps determine the camera orientation. Note this is recomputed from the given camera and look positions, so may not be exactly the same as the up vector given in *lookAtPosition()*.

---

**zScale**

A read-only number with the number of pixels per unit on the Z axis. See the *Z scale* property in the 3D Camera manual entry for more details.

---

**fieldOfView**

Set or get the angle of the field of view in radians. Note this only has an effect when *Z Axis scale* is set to *Regular* in Project Properties. The default field of view is also set in Project Properties.

# 3D SHAPE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/3d-shape

The `I3DShapeInstance` interface derives from IWorldInstance to add APIs specific to the 3D shape plugin.

## 3D Shape APIs

### shape

Set or get a string representing the the current shape being displayed. This must be one of `"box"`, `"prism"`, `"wedge"`, `"pyramid"`, `"corner-out"` and `"corner-in"`.

### zHeight

Set or get the current Z height (i.e. depth) of the 3D shape. Note the Z height must be greater or equal to 0.

### setFaceVisible(face, visible)

Set whether a face is visible or invisible based on the boolean *visible*. The *face* parameter is a string identifying the face which must be one of `"back"`, `"front"`, `"left"`, `"right"`, `"top"` and `"bottom"`.

### isFaceVisible(face)

Return a boolean indicating whether a given face is visible. The *face* parameter is a string identifying the face which must be one of `"back"`, `"front"`, `"left"`, `"right"`, `"top"` and `"bottom"`.

### setFaceImage(face, image)

Change one of the shape faces to use one of the other face images. For example this allows swapping the front face image for the back face image. To restore the original image, use the same face for both parameters. Both parameters are strings identifying the face to use, which must be one of `"back"`, `"front"`, `"left"`, `"right"`, `"top"` and `"bottom"`.

> *This also undoes `setFaceObject`, restoring the 3D shape's own face image instead of another object's image.*

### setFaceObject(face, objectClass)

Replace the image used for a face of the shape with the image used by a Sprite, Tiled Background or 9-Patch object. An instance of the given object must exist on the current layout. The *face* parameter is a string identifying the face which must be one of `"back"`, `"front"`, `"left"`, `"right"`, `"top"` and `"bottom"`. The *objectClass* parameter is an IObjectClass referencing the object to set for the given face. Only Sprite, Tiled Background and 9-Patch object types are supported.

> *This method can be undone with* `setFaceImage`.

---

**isBackFaceCulling**

Set or get a boolean corresponding to the *Back face culling* property. When enabled, back face culling automatically hides faces of the shape that are pointing away from the camera.

---

**zTilingFactor**

Set or get the Z tiling factor property of the 3D shape. For more information, refer to the 3D shape plugin manual entry.

---

**getImagePointCount()**

Return the number of image points on the back face.

---

**getImagePointX(nameOrIndex)**

**getImagePointY(nameOrIndex)**

**getImagePoint(nameOrIndex)**

Return the location of an image point on the back face in layout co-ordinates. Image points are identified either by a case-insensitive string of their name, or their index. Note image point 0 is the origin, so index 1 is the first image point. If the image point is not found, this returns the origin instead. The `getImagePoint` variant returns `[x, y]`.

---

**getFaceImagePointCount(face)**

**getFaceImagePointX(face, nameOrIndex)**

**getFaceImagePointY(face, nameOrIndex)**

**getFaceImagePointZ(face, nameOrIndex)**

**getFaceImagePoint(face, nameOrIndex)**

Return the count and 3D location of image points on any of the six possible faces of the 3D shape. The face is identified by a string which must be one of `"back"`, `"front"`, `"left"`, `"right"`, `"top"`, `"bottom"`. Image points are identified either by a case-insensitive string of their name, or their index. Note as image point 0 refers to the origin, index 1 is the first

image point. If the image point is not found, this returns the origin instead. The `getFaceImagePoint` variant returns `[x, y, z]`.

# 9-PATCH SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/9-patch

The `I9PatchInstance` interface derives from IWorldInstance to add APIs specific to the 9-patch plugin.

## 9-patch APIs

**leftMargin**

**rightMargin**

**topMargin**

**bottomMargin**

Set or get the margins of each side of the 9-patch in pixels. Note these margins apply to the source image, ignoring the image scale properties. By default all 9-patch instances share the same set of patch images, and so changing the margins will affect all instances, regardless of which instance was modified. However the exception to this is after the `replaceImage()` method completes, in which case the instance is using its own unique patch images, and the margins can then be changed for that instance only.

**edges**

Set or get the style of the edge patches as a string, which must be one of `"tile"` or `"stretch"`.

**fill**

Set or get the style of the central patch as a string, which must be one of `"tile"`, `"stretch"` or `"transparent"`.

**seams**

Set or get the seams mode as a string, which must be one of `"exact"` or `"overlap"`.

**imageScaleX**

**imageScaleY**

Set or get the scale of the patches drawn inside the object, as a scale factor (e.g. 1 represents 100% scale).

**async replaceImage(blob)**

Replace the current image with the contents of a Blob representing an image file such as a PNG image. The blob can be locally generated or retrieved from a URL, for example:

```
// Loading an image from a URL
const response = await fetch(url);
const blob = await response.blob();
await inst.replaceImage(blob);

// (adjust margins if necessary here, e.g.:)
// inst.leftMargin = 20;
```

Once this method has completed (after `await`, or the Promise resolving), the margins can then be modified for the specific instance without affecting all other instances. Be sure to only modify the margins after the returned promise resolves, otherwise it will still be using the shared patch images and so affect all other instances.

# ADVANCED RANDOM SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/advanced-random

---

The `IAdvancedRandomObjectType` interface derives from IObjectClass to add APIs specific to the Advanced Random plugin.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.objects.AdvancedRandom`.

## Advanced Random APIs

---

**seed**

Set or get a string with the current seed for the pseudo-random number generator. The same seed will produce the same sequence of pseudo-random numbers.

---

**octaves**

Set or get the number of octaves used for coherent noise generation, from 1-16. The default is 1. This affects the Billow, Classic and Ridged noise functions only. Using additional octaves adds layers of increasing detail to the noise functions, but is also slower to process.

---

**billow2d(x, y)**

**billow3d(x, y, z)**

Generate a random number using billow noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**cellular2d(x, y)**

**cellular3d(x, y, z)**

Generate a random number using cellular noise in the range 0-1, using either 2D or 3D co-ordinates.[/dd classic2d(x, y)
classic3d(x, y, z)
Generate a random number using classic (perlin) noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**ridged2d(x, y)**

**ridged3d(x, y, z)**

Generate a random number using ridged noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**voronoi2d(x, y)**

**voronoi3d(x, y, z)**

Generate a random number using Voronoi noise in the range 0-1, using either 2D or 3D co-ordinates.

---

**createGradient(name, mode)**

Create a new gradient with a given string for its name. The `mode` must be one of `"float"` or `"rgb"`.

---

**setCurrentGradient(name)**

Set the current gradient that is the default to sample from.

---

**addGradientStop(position, value)**

Adds a stop to the current gradient. The stop position can be any number, but is generally kept within the 0-1 range so it can be used with the random expressions.

---

**sampleGradient(name, position)**

Sample a gradient at the given position. The `name` can be omitted (pass `null`) to use the current gradient; otherwise it specifies a case-insensitive string of the gradient to sample.

---

**createProbabilityTable(name)**

Create a new probability table, using a string of its name to identify it.

---

**createProbabilityTableFromJSON(name, jsonStr)**

Create a new probability table with a name, using a string of JSON data from a prior call to `getProbabilityTableAsJSON()` for its entries.

---

**getProbabilityTableAsJSON()**

Return a string of JSON data representing the current probability table.

---

**setCurrentProbabilityTable(name)**

Set the current probability table by a string of its name.

---

**addProbabilityTableEntry(weight, value)**

Add an entry to the current probability table with the given weight and value.

---

**removeProbabilityTableEntry(weight, value)**

Remove an existing entry from the current probability table. If a weight of 0 is specified, the first entry with the given value is removed regardless of its weight. Otherwise an entry is only removed if it matches both the value and the weight.

---

**sampleProbabilityTable(name)**

Get a random value from a probability table. The relative likelihood of values is affected by their weight. The `name` can be omitted (pass `null`) to use the current probability table; otherwise it specifies a case-insensitive string of the probability table to sample.

---

**createPermutationTable(length, offset)**

Generate a randomly ordered sequence of numbers. The `length` parameter is how many numbers to generate, and `offset` is the first number in the sequence. For example a `length` of 3 with an `offset` of 1 will generate the numbers 1, 2 and 3, and then randomly shuffle them.

---

**shufflePermutationTable()**

Re-shuffle an existing permutation table.

---

**getPermutation(index)**

Get a value at a zero-based index in the permutation table.

# ARRAY SCRIPT INTERFACE

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

The `IArrayInstance` interface derives from IInstance to add APIs specific to the Array plugin.

> *The Array object can only store strings and numbers, since these are the only types Construct uses in expressions. Use normal JavaScript arrays to store other types.*

## Array APIs

**width**

**height**

**depth**

Retrieve the current dimensions of the array.

**setSize(w, h = 1, d = 1)**

Set the size of the array in up to three dimensions. For one or two dimensional arrays, the later parameters can be omitted as they default to 1. (Note passing 0 for any dimension will create an array with zero elements that cannot store any data.) If the array grows, new elements have the value 0. If the array shrinks, elements are removed.

**getAt(x, y = 0, z = 0)**

Retrieve an element from the array at the given X, Y and Z co-ordinates. For one or two dimensional access, the later parameters can be omitted as they default to 0.

**setAt(val, x, y = 0, z = 0)**

Set an element in the array at the given X, Y and Z co-ordinates. `val` must be a number or string. For one or two dimensional arrays, the later parameters can be omitted as they default to 0.

# AUDIO SCRIPT INTERFACE

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

The `IAudioObjectType` interface derives from IObjectClass to add APIs specific to the Audio plugin.

The script interface essentially just provides access to the underlying AudioContext (part of the Web Audio API) that the Audio plugin uses internally for audio playback. However this is sufficient to provide complete control over audio playback, including setting up complex graphs of audio processing nodes. It is also convenient, since Construct manages some awkward details such as ensuring playback is enabled as soon as possible given most browsers impose autoplay restrictions. Adding the Audio object and using its script interface saves you from having to re-implement these details yourself.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.objects.Audio`.

> *Since the Web Audio API is not available in Web Workers, the Audio object's script interface can only be used in DOM mode, i.e. with the project Use worker option turned off.*

## Examples

The following examples demonstrate using the Web Audio API for audio playback.

- Audio scripting demonstrates loading audio files and playing them
- Sound synthesis demonstrates generating sounds with code

## Audio APIs

**audioContext**

The Audio plugin's internal AudioContext used for audio playback.

**destinationNode**

The destination node to connect any additional audio nodes to.

> *While `AudioContext` has its own `destination` property, in some cases Construct redirects the audio output to another destination, such as when recording playback. To achieve this it creates its own destination node which it can redirect the output from. By*

*connecting your own nodes to this destination your script's audio output will properly integrate with other Construct features like recording.*

---

**isSilent**

Set or get a boolean indicating whether *Silent mode* is enabled. In silent mode all currently playing sounds are muted and no new sounds will play.

---

**masterVolume**

Set or get the current master volume, which is applied to all audio playback, in dB.

---

**stopAll()**

Stop all currently playing sounds.

# BINARY DATA SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/binary-data

The `IBinaryDataInstance` interface derives from IInstance to add APIs specific to the Binary Data plugin. This makes it possible to read and alter the binary data from scripts, which is often more convenient than trying to do so from events.

Binary Data stores its data as an ArrayBuffer, which cannot be directly modified. In JavaScript, data can be read and written to an ArrayBuffer using typed arrays or DataView.

## Binary Data APIs

### setArrayBufferCopy(viewOrBuffer)

Set the contents of the Binary Data object by an ArrayBuffer or typed array which is copied. This means it is safe to continue using the passed data after this call.

### setArrayBufferTransfer(arrayBuffer)

Set the contents of the Binary Data object by an ArrayBuffer which the Binary Data object takes ownership of. You must not use the passed ArrayBuffer after this call, since it is now managed by the Binary Data object. Since this method does not copy the passed ArrayBuffer, it is more efficient if the ArrayBuffer won't be used again in your code. Note this method does not accept a typed array.

### getArrayBufferCopy()

Return the contents of the Binary Data object as an ArrayBuffer which is a copy of the internal ArrayBuffer. This means it is safe to modify the returned ArrayBuffer without affecting the state of the Binary Data object.

### getArrayBufferReadOnly()

Return the contents of the Binary Data object as a read-only reference to the internal ArrayBuffer. This ArrayBuffer must not be modified since it is managed by the Binary Data object. However it is more efficient to use this method if the data is only read from, e.g. to send over the network, since it does not copy the ArrayBuffer.

# BUTTON SCRIPT INTERFACE

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

The `IButtonInstance` interface derives from IDOMInstance to add APIs specific to the Button plugin.

## Button events

See instance event for standard instance event object properties.

**"click"**

Fired when the button is clicked, or the checkbox state is toggled.

## Button APIs

**text**

The string currently displayed as the button or checkbox label.

**tooltip**

The string used as the tooltip for the button or checkbox.

**isEnabled**

A boolean indicating if the control is enabled or disabled.

**isChecked**

A boolean indicating if the checkbox is checked. For button style controls this is always false.

# CONSTRUCT GAME SERVICES SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/construct-game-services

The `ICGSObjectType` interface derives from IObjectType to add APIs specific to the Construct Game Services plugin. Construct Game Services is also written as CGS for short.

## Using the REST APIs

Construct Game Services have a comprehensive set of REST APIs available. When writing code it is possible to use those entirely independently of this plugin. This option may be of interest to advanced users - see the Construct Game Services manual for details.

The CGS script interface also provides the `sessionKey` when signed in. This can be used to make REST API calls with the user's current authentication. This allows using the CGS script interface solely for authentication, and then using the resulting `sessionKey` to make REST API calls.

For completeness the CGS script interface does provide equivalent features to the plugin, but note this is a subset of all the available features in the REST API.

## CGS events

**"signinpopupblocked"**

Triggered during sign in if the browser popup blocker prevented the sign in popup window from appearing. Use the `retryOpenSignInPopup()` method in a user input event to retry.

## CGS APIs

**isSignedIn**

Read-only boolean indicating if the player is currently signed in.

**canSignInPersistent**

Read-only boolean indicating if persistent sign in with `signInPersistent()` is available.

**playerId**

A read-only string of the unique player ID when signed in.

### playerName

A read-only string of the player display name when signed in.

### gameId

A read-only string of the game ID signed in to when signed in.

### sessionKey

When signed in, a string with the session key for this sign in. This can be used to make REST API calls using the same authentication.

### async signInWithProvider(provider, gameId, options)

Attempt to sign the user in with a third-party identity service like Google or Microsoft. `provider` must be one of: `"Apple"`, `"BattleNet"`, `"BattleNetChina"`, `"Discord"`, `"Facebook"`, `"Github"`, `"Google"`, `"ItchIO"`, `"Microsoft"`, `"Reddit"`, `"Steam"`, `"X"`, `"Yandex"`. The game ID to sign in to from your Construct Game Services account must be provided. `options` is an optional object that may provide the following properties:

- `allowPersisting`: if `true` a successful sign in is remembered, and may be re-used in future with `signInPersistent()`. The default is `true`.
- `expiryMins`: the time in minutes that sessions remain active. The default is 1440 (24 hours).
- `popupWindowWidth` and `popupWindowHeight`: Signing in with a provider will open a popup window, and you can also specify the size of this popup window with these properties. The default is 800x800.

The method returns a Promise that resolves when sign in has completed successfully, or rejects if sign in fails or is cancelled.

> *Note that in some cases, sign in may be cancelled in a way that cannot be detected, and so the returned promise will not settle. Be sure to design your project with this in mind - for example do not block the user interface until sign in finishes, as you cannot reliably detect that.*

### retryOpenSignInPopup()

After the `"signinpopupblocked"` event fires, make another attempt to open the sign in popup window to continue the sign in process. This should be used in a user input event such as a button click or mouse click, otherwise the browser popup blocker may still block the second attempt.

### async signInPersistent(gameId)

Attempt to sign in re-using a previous successful sign in that allowed persisting. This can only be used when `canSignInPersistent` is true. This returns a Promise that resolves when the persistent sign in has completed.

### async signOut()

Sign out of any account the user is currently signed in to, and also delete any remembered sign in if it allowed persisting. Locally this operation completes immediately. For completeness signing out will send a request in the background to ensure the session is ended on the server-side as well, but that is optional and if it fails the session will time out anyway. This method returns a Promise that resolves when the background request completes, but failure should not be considered significant.

### async setPlayerName(name)

Set the name of the currently signed in player. In some cases the player name is obtained from the authentication provider, but in others a generic name will be used, in which case the player will likely want to change their name for features like leaderboards. Returns a Promise that resolves when the operation completes.

### async submitScore(score, leaderboardId)

Submit a score to a leaderboard. `score` must be an integer (fractional scores are not supported). `leaderboardId` may be omitted to submit a score on the Construct Arcade, which does not require authentication. Otherwise it must be set to the leaderboard ID to submit the score to, in which case authentication is required. This method returns a Promise that resolves when the score has been successfully submitted.

### async getLeaderboardScores(leaderboardId, options)

Request a page of scores from a given leaderboard ID. This does not require authentication. The returned scores can optionally be filtered with several options, which may include:

- `resultsPerPage`: the number of results to fetch. The default is 20.
- `page`: zero-based index of the page to fetch. The default is 0 to retrieve the first page.
- `country`: an ISO 3166-1 alpha-2 country code filter can be provided, e.g. "US" to only return scores submitted in the United States of America.
- `range`: a time range to filter results by, which if provided must be one of `"Daily"`, `"Weekly"`, `"Monthly"` or `"Yearly"`. For example using `"Daily"` will return today's

scores. Weekly leaderboards run from Monday to Sunday. If this property is omitted it returns all scores (which is the default).

- `rangeOffset`: an offset to return a prior time range when `range` is specified. For example specifying a `"Daily"` range with a `rangeOffset` of 1 will return yesterday's scores.

- `culture`: a locale to use for returned values. By default it will use the leaderboard's default culture.

This method returns a Promise that resolves with the following object upon success:

- `totalPageCount`: the total number of pages available

- `scores`: an array of returned scores, each being an object with the following properties:
  - `score`: a number representing this score.
  - `formattedScore`: a string with the score formatted according to the culture.
  - `rank`: a number representing the score rank.
  - `formattedRank`: a string with the rank formatted according to the culture.
  - `country`: a string of the ISO 3166-1 alpha-2 country code the score was submitted from.
  - `playerId`: a string of the unique player ID who submitted the score.
  - `playerName`: a string of the player display name who submitted the score.

---

**async createCloudSave(opts)**

Create or replace the data for a cloud save key. The `opts` parameter is an object which uses the following properties to specify options:

- `key` (required): the storage key to save to.

- `bucketId`: the bucket ID to save to. This can be omitted to use player private storage.

- `name`: an optional name to associate with the data, such as a file name.

- `data` (required): a string or Blob for the data to upload.

The method returns a Promise that resolves when the operation has completed.

---

**async getCloudSave(opts)**

Download the data previously uploaded for a key in a game bucket with `createCloudSave`. The `opts` parameter is an object which uses the following properties to specify options:

- `key` (required): the storage key to save to.
- `bucketId`: the bucket ID to save to. This can be omitted to use player private storage.
- `type`: a string of `"text"` or `"blob"` that determines the data type the returned Promise resolves with. The default is `"text"`.

The method returns a Promise that returns with a string when `type` is `"text"`, or a Blob if `type` is `"blob"`.

# CSV SCRIPT INTERFACE

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

---

The `ICSVObjectType` interface provides APIs specific to the CSV plugin. This allows reading and writing CSV data (and other delimiter-based formats like TSV) from JavaScript code.

The script interface uses JavaScript data types directly and so does not need to use an Array object for storage, which is necessary when using CSV in event sheets.

## CSV APIs

---

**parseCsv(str, delimiter = ",", dataType = "auto")**

Parse a given string of CSV data using the provided delimiter. The `dataType` parameter must be one of `"auto"`, `"string"` or `"number"`. The return value is an array of arrays with values, representing the two-dimensional array of values in the data.

---

**generateCsv(arr, delimiter = ",")**

Generate a string of CSV data using the provided delimiter, from a two-dimensional array of values.

# DICTIONARY SCRIPT INTERFACE

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

The `IDictionaryInstance` interface derives from IInstance to add APIs specific to the Dictionary plugin.

> *The Dictionary object can only use strings as keys, and strings/numbers as values, since these are the only types supported by the plugin. Use your own independent JavaScript Maps to use other types.*

## Dictionary APIs

### getDataMap()

Return the Map which is used as the underlying data storage for the Dictionary object. This allows access to add, change, remove and iterate items.

> *Only use string keys, and only store number or string primitives as key values, or the plugin will cease to work correctly.*

# 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.

# FACEBOOK SCRIPT INTERFACE

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

The `IFacebookObjectType` interface derives from IObjectClass to add APIs specific to the Facebook plugin. Note this class derives from the object class interface, not the instance interface. It would typically be accessed via `runtime.objects.Facebook`.

This script interface exists mainly to provide the access token so you can write code to access the Facebook SDK for JavaScript using the existing authentication provided by Construct's Facebook plugin. This may also be useful for third-party addons developed with Construct's addon SDK.

## Facebook APIs

### getAccessToken()

A string of the access token provided by the Facebook SDK after a successful login. If the user is not logged in, this will return an empty string.

# FILE CHOOSER SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/file-chooser

---

The `IFileChooserInstance` interface derives from IDOMInstance to add APIs specific to the File Chooser plugin.

## File Chooser events

See instance event for standard instance event object properties.

---

**"change"**

Fired when a file or set of files has been chosen from the input control.

## File Chooser APIs

---

**click()**

Acts as if the file chooser was clicked, which brings up the system file picker. This allows the button to be completely customized, as the actual File Chooser object can be moved offscreen, and a custom sprite or other object used to activate the `click()` method instead.

> *The browser may only allow this method in a user gesture (e.g. mouse click).*

---

**clear()**

Reset the control to its initial state with no selection, clearing any prior picked file.

---

**getFiles()**

Return an array of File objects representing the currently chosen files.

# FILE SYSTEM SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/file-system

---

The `IFileSystemObjectType` interface derives from IObjectClass to add APIs specific to the File System plugin. Refer to the plugin documentation for details about permissions and known folders.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.objects.FileSystem`.

## Common types

The following types are used across multiple APIs and so are described once here.

### Accept types

File open and save pickers allow specifying file type filters, also known as the accept types. Each accept type is specified by an object with the following properties:

- `description`: a description of this file type that may be shown to the user.
- `accept`: an object whose keys are MIME types, and values are an array of the corresponding file extensions, including the dot.

Here is an example for an accept type for common image file formats:

```
{
        description: "Images",
        accept: {
                "image/*": [".png", ".jpg", ".webp", ".avif"],
        }
}
```

### Start in locations

An initial folder location can be specified using a string which may be one of the following values: `"default"`, `"desktop"`, `"documents"`, `"downloads"`, `"music"`, `"pictures"`, `"videos"`.

## File System events

---

**"drop"**

Fired when the user drags and drops files in to the window. The event object has the property `files` which has an array of File objects representing the dropped files.

# File System APIs

---

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for a File System event. See *File System events* above for more information.

---

**isSupported**

A read-only boolean indicating if file system features are supported. This depends on support for the File System Access API in the browser. If false then none of the file system APIs will work.

---

**desktopFeaturesSupported**

A read-only boolean indicating whether desktop-specific features are available. This is only the case when running after using a supported desktop export option. When true, the known folder pickers like `"<documents>"` can be used (providing they are available - use the `hasPickerTag()` method to check), as well as the desktop-specific methods like `runFile()` and `shellOpen()`.

---

**hasPickerTag(pickerTag)**

Return a boolean indicating if a given picker tag has been remembered from a previous session, or if a known folder tag is available. In this case the picker tag can still be referred to for file system operations, such as to read a previously chosen file, without having to show a picker again.

---

**async showSaveFilePicker(opts)**

Show a save file picker allowing the user to choose a file on their local system to save to. The `opts` parameter is an object which uses the following properties to specify options:

- `pickerTag` (required): a string of a tag to identify the chosen folder.
- `acceptTypes`: an array of objects used to filter the listed files. See *Accept types* above for more information.
- `showAcceptAll`: a boolean indicating whether to show an accept type that shows all kinds of files.

- `suggestedName`: a string used as the initial filename choice to save to in the picker.
- `id`: a string used as an extra identifier for remembering the picker settings, such as the last viewed folder.
- `startIn`: a string specifying the initial folder to show (when not remembered based on the `id`). See *Start in locations above for more information*.

The method returns a Promise that resolves with an array of strings for the selected file names, which in the case of a save file picker, will contain a single entry.

---

**async showOpenFilePicker(opts)**

Show an open file picker allowing the user to choose one or multiple files on their local system to open. The `opts` parameter is an object which uses the following properties to specify options:

- `pickerTag` (required): a string of a tag to identify the chosen folder.
- `acceptTypes`: an array of objects used to filter the listed files. See *Accept types* above for more information.
- `showAcceptAll`: a boolean indicating whether to show an accept type that shows all kinds of files.
- `multiple`: a boolean indicating whether to allow the user to select multiple files.
- `id`: a string used as an extra identifier for remembering the picker settings, such as the last viewed folder.
- `startIn`: a string specifying the initial folder to show (when not remembered based on the `id`). See *Start in locations above for more information*.

The method returns a Promise that resolves with an array of strings for the selected file names.

---

**async showFolderPicker(opts)**

Show a folder picker allowing the user to choose a folder on their local system. The `opts` parameter is an object which uses the following properties to specify options:

- `pickerTag` (required): a string of a tag to identify the chosen folder.
- `mode`: a string of either `"read"` or `"readwrite"` to specify whether read-only or read-write permission is to be granted for the chosen folder, which also affects the permission

prompt shown to the user.

- `id` : a string used as an extra identifier for remembering the picker settings, such as the last viewed folder.

- `startIn` : a string specifying the initial folder to show (when not remembered based on the `id` ). See *Start in locations above for more information*.

The method returns a Promise that resolves with an array of strings for the selected folder names, which in the case of a folder picker, will contain a single entry.

---

**async writeFile(opts)**

Write data to a previously chosen file. The `opts` parameter is an object which uses the following properties to specify options:

- `pickerTag` (required): a string of the picker tag used to choose the file or folder.

- `data` (required): a string or ArrayBuffer with the data to be written. Strings are written as text with UTF-8 encoding.

- `folderPath` : when the picker tag refers to a folder, the relative path to the file inside that folder, e.g. "subfolder/file.txt".

- `mode` : a string of either `"overwrite"` or `"append"` for the file write mode. The default is to overwrite, replacing the entire file contents with the provided data. Append mode is only supported when writing text (passing a string for `data` ).

- `fileTag` : a file tag used by triggers in the event system.

The method returns a Promise that resolves when the write has completed.

---

**async readFile(opts)**

Read data from a previously chosen file. The `opts` parameter is an object which uses the following properties to specify options:

- `pickerTag` (required): a string of the picker tag used to choose the file or folder.

- `mode` (required): a string of either `"text"` or `"binary"` to specify the type of the data returned. In text mode the returned Promise will resolve with a string, and in binary mode it will resolve with an ArrayBuffer.

- `folderPath` : when the picker tag refers to a folder, the relative path to the file inside that folder, e.g. "subfolder/file.txt".

- `fileTag`: a file tag used by triggers in the event system.

The method returns a Promise that resolves with either a string when `mode` is `"text"`, or an ArrayBuffer when `mode` is `"binary"`.

---

**async createFolder(pickerTag, folderPath, fileTag)**

Only applies to folder pickers. Create a subfolder within a previously picked folder. `folderPath` specifies the folder to create. This can refer to multiple subfolders which will all be created, e.g. "subfolder/otherfolder". `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the folder has been created.

---

**async copyFile(pickerTag, srcFolderPath, destFolderPath, fileTag)**

Only applies to folder pickers. Copy a file within a previously picked folder. `srcFolderPath` specifies an existing file to copy, e.g. "subfolder/file1.txt". `destFolderPath` specifies where to create a copy; this will either create a new file if it doesn't exist, or overwrite an existing file if it already exists. `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the file has been copied.

---

**async moveFile(pickerTag, srcFolderPath, destFolderPath, fileTag)**

Only applies to folder pickers. Move a file within a previously picked folder. The file can be moved from one subfolder to another, or if it stays in the same folder, it renames the file. `srcFolderPath` specifies the file to move, e.g. "subfolder/file1.txt". `destFolderPath` specifies where to move (or rename) the file to. `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the file has been moved.

---

**async delete(pickerTag, folderPath, isRecursive, fileTag)**

Only applies to folder pickers. Delete a file or folder within a previously picked folder. `folderPath` specifies the file or folder to delete, e.g. "subfolder/file1.txt" or "subfolder". `isRecursive` applies only when Folder path identifies a folder: if true it will delete all files and folders within the identified folder as well as the folder itself, but if false it will only successfully delete the folder if it is already empty. `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the deletion has completed. Note that deleting a folder recursively may take a long time if it contains a large number of files/folders.

---

**async listContent(pickerTag, folderPath, isRecursive, fileTag)**

Only applies to folder pickers. Retrieves a list of all files and folders within a previously picked folder. `folderPath` identifies a subfolder to list the contents for, or can be left empty to list the contents of the originally picked folder. If `isRecursive` is true it will also list all files and folders through subfolders of the specified folder.

> *Recursive listing may be slow with a very large folder that contains thousands of files/subfolders.*

`fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves upon completion with an object with the following properties:

- `files`: an array of strings with file names in the specified folder.
- `folders`: an array of strings with folder names in the specified folder.

When `isRecursive` was true, the listed file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.

---

### async shellOpen(pickerTag, filePath, fileTag)

Only supported in desktop exports when `desktopFeaturesSupported` is true. This action requests that the operating system shell (the component that handles the system user interface) opens the specified file. Typically this launches the default app associated with the file type and then opens the file in it, similar to double-clicking the file in the system file explorer. `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the operation has completed.

---

### async runFile(pickerTag, filePath, args, fileTag)

Only supported in desktop exports when `desktopFeaturesSupported` is true. This action attempts to execute the provided file path. Command-line arguments can also be optionally provided as the string `args`. This can be used to run another program. Uniquely for this action the folder picker parameter can be left empty, in which case the provided file path is run directly; this can be useful to run system executables like "cmd.exe" on Windows. `fileTag` is optional and used by triggers in the event system. Returns a Promise that resolves when the operation has completed.

# HTML ELEMENT SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/html-element

The `IHTMLElementInstance` interface derives from IDOMInstance to add APIs specific to the HTML Element plugin.

In DOM mode the HTML Element can be accessed with standard DOM APIs using the `getElement()` method of `IDOMInstance`. However these methods can all still be used in worker mode. Most of the methods are asynchronous as in this case updates are managed by posting messages between the worker and the DOM.

## HTML Element events

See instance event for standard instance event object properties.

**"click"**

Fired when any part of the HTML Element or its content is clicked. This event fires repeatedly for the target element followed by all parent elements up to the main HTML element, simulating event bubbling. The event object has two additional properties `targetId` and `targetClass` that can be used to identify the element clicked.

**"animationend"**

Fired when any CSS animation inside the HTML element finishes (based on the DOM event of the same name, but note this event is dispatched by Construct and not the DOM). The event object has three additional properties: `targetId` and `targetClass`, which can be used to identify the element whose animation ended, and `animationName` which identifies the name of the CSS animation that ended (as defined by the `@keyframes` rule).

## HTML Element APIs

**async setContent(str, type = "html", selector = "", isAll = false)**

Replaces some content inside the HTML element with the given string `str`. The string is interpreted according to `type` which must be one of `"html"`, `"bbcode"` or `"text"`. The location to replace content is specified by a CSS selector string. This can be left blank to replace the content of the entire main HTML element. The `isAll` flag will update all elements matching the selector if set, otherwise only the first matching element is updated.

**async insertContent(str, type = "html", atEnd = true, selector = "", isAll = false)**

Insert the string of content `str` inside the HTML element. The string is interpreted according to `type` which must be one of `"html"`, `"bbcode"` or `"text"`. The location to insert content is specified by a CSS selector string. This can be left blank to insert the content to the main HTML element. The `atEnd` flag inserts content at the end if true, or the beginning if false. The `isAll` flag will update all elements matching the selector if set, otherwise only the first matching element is updated.

---

**async setContentClass(mode, classArr, selector, isAll = false)**

Adds, toggles or removes element classes according to `mode`, which must be one of `"add"`, `"toggle"` or `"remove"`. The classes to modify are given as an array of strings in `classArr`. The location to change is specified by a CSS selector string. This can be left blank to change the classes of the main HTML element. The `isAll` flag will update all elements matching the selector if set, otherwise only the first matching element is updated.

---

**async setContentAttribute(mode, attrib, value, selector, isAll = false)**

Adds or removes element attributes according to `mode`, which must be one of `"set"` or `"remove"`. The string of the attribute name to modify is specified by `attrib`, and its value as a string of `value` (ignored if removing the attribute). The location to change is specified by a CSS selector string. This can be left blank to change the attributes of the main HTML element. The `isAll` flag will update all elements matching the selector if set, otherwise only the first matching element is updated.

---

**async setContentCssStyle(propName, value, selector, isAll)**

Sets an element's style . The string of the CSS property name to modify is specified by `propName`, which may use either CSS naming (e.g. `"font-size"`) or JavaScript naming (e.g. `"fontSize"`). The value to set for this property is given by a string of `value`; set an empty string to revert the style to default. The location to change is specified by a CSS selector string. This can be left blank to change the style of the main HTML element. The `isAll` flag will update all elements matching the selector if set, otherwise only the first matching element is updated.

---

**async setScrollPosition(selector, direction, position)**

Set the horizontal or vertical scroll position of an element. The HTML element to scroll is given by the CSS selector string `selector`, which can be set to an empty string to scroll the main HTML element. This action only scrolls one element matching the selector. `direction` must be set to either `"left"` to set the scroll left (horizontal) position or `"top"` to set the scroll top (vertical) position. The *position* value is the scroll position to set in CSS pixels.

---

**htmlContent**

A string of the complete inner HTML of the main HTML element. This can also be assigned to change the full content of the HTML element.

> *Methods that change the HTML element, including assigning this property, are asynchronous. This means reading the value back won't update until the asynchronous methods have completed.*

---

### textContent

A string of the complete inner text of the main HTML element, in plain text form (with no HTML tags). This can also be assigned to change the full text content of the HTML element.

> *Methods that change the HTML element, including assigning this property, are asynchronous. This means reading the value back won't update until the asynchronous methods have completed.*

---

### async createSpriteImgElement(spriteInst, selector, insertAt, id, class_)

Creates an `<img>` element with the content of a given ISpriteInstance's current image, and inserts it to the HTML element. The location to insert is specified by a CSS selector string. The `insertAt` parameter must be a string of one of `"start"`, `"end"` or `"replace"` indicating how to insert the image element. The `id` and `class_` parameters are optional strings to set an ID or class for the inserted image element, which helps make it easy to style the inserted image with CSS.

> *This method provides a simple way to show a Sprite image on top of a HTML element, since normally HTML elements always show on top of Sprites.*

---

### async positionInstanceAtElement(worldInst, selector)

Sets the position and size of a given IWorldInstance to match the position and size of a specific HTML element given by a CSS selector string.

> *This method provides a way to use invisible HTML and CSS for complex layouts, while displaying the actual content with other objects, allowing for full use of Z order, effects and so on.*

# INTERNATIONALIZATION SCRIPT INTERFACE

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

The `IInternationalizationObjectType` interface derives from IObjectType to add APIs specific to the Internationalization plugin.

> *Internationalization is sometimes written as the shorthand i18n, referring to the fact the word starts with an I, ends with an N, and has 18 other letters in between.*

## Supported APIs

Many features of the Internationalization plugin merely access the browser-provided Intl APIs, such as for formatting dates and times, identifying plural types, and so on. When writing code, you may as well access these APIs directly rather than through the script interface, so those APIs are not duplicated here. Instead the script interface only provides access to string lookups for translations.

## Internationalization APIs

**locale**

Set or get a string representing the currently set locale as a BCP 47 language tag, e.g. `"en-US"`.

**addString(context, str)**

Add a localized string to the provided context in the localization data for the current locale.

**saveToJSONString()**

Returns a string of JSON representing the state of the Internationalization object.

**loadFromJSONString(str)**

Load the state of the Internationalization object from a string of JSON data.

**setContext(context)**

**getContext()**

Set and get a string representing the current context for string lookups.

---

**pushContext(context)**

**popContext()**

Push or pop to the context stack. When a string is pushed, it is used as the base for all relative context lookups.

> *Consider using* `createContext()` *as a more convenient alternative API, as it saves you from having to remember to call* `popContext()`.

---

**createContext(context)**

Return an `I18NLookupContext` representing the given context. This provides a convenient way to look up several strings relative to that context. See the section below for more details.

---

**lookup(context, ...args)**

**lookupPlural(context, count, ...args)**

Look up a string at the given context, and substitute placeholders with the following arguments. The context may be relative if a context has been pushed. The plural variant also makes use of the `count` argument to select the appropriate plural variant for the current locale. See the Internationalization manual entry for more details on placeholders, relative contexts and pluralization.

# I18NLookupContext

The `createContext()` method returns an `I18NLookupContext` which can be used for more conveniently looking up lots of strings from the same context. This works similarly to using `pushContext()`, but takes advantage of JavaScript's garbage collector to avoid having to need a following call to `popContext()`. This makes it more convenient and less error-prone to use than pushing and popping contexts. It's also uniquely available to the script interface, as Construct's event system does not support allow a comparable API.

The code sample below demonstrates how it can be used.

```
const intl = runtime.objects.Internationalization;

// Normal lookup having to repeat the context
let string1 = intl.lookup("forest-world.chapter-1.intro-text.title");
let string2 = intl.lookup("forest-world.chapter-1.intro-text.message");
let string3 = intl.lookup("forest-world.chapter-1.intro-text.start");

// Shorter lookups using a context
```

```
const ctx = intl.createContext("forest-world.chapter-1.intro-text");

string1 = ctx.lookup(".title");
string2 = ctx.lookup(".message");
string3 = ctx.lookup(".start");
```

## I18NLookupContext APIs

---

### lookup(context, ...args)

### lookupPlural(context, count, ...args)

These methods are equivalent to those on the Internationalization script interface, but may use contexts relative to the context this `I18NLookupContext` was created with.

---

### createContext(context)

Create another `I18NLookupContext` for the given context, which must be relative. This allows creating further `I18NLookupContext` objects for more deeply nested contexts.

# JSON SCRIPT INTERFACE

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

The `IJSONInstance` interface derives from IInstance to add APIs specific to the JSON plugin. JSON can be conveniently modified from script alone, but this interface allows exchanging data between event sheets and code.

## JSON APIs

### getJsonDataCopy()

Return a copy of the JSON data held in the object.

> *Note since this returns a copy of the data, changing the returned data will not affect the contents of the JSON object.*

### setJsonDataCopy(o)

Set the JSON data held in the object.

> *Note this takes a copy of the data, so changing the provided data after this call will not affect the contents of the JSON object.*

> *The provided data is validated and will throw an exception if it's not valid JSON.*

### setJsonString(str)

Parses a string as JSON data and stores the result in the JSON object.

> *This will throw an exception if the string is not valid JSON.*

### toCompactString()

### toBeautifiedString()

Return the contents of the JSON object converted to a string, either in compact form (which is smaller and more efficient to store and send), or "beautified" (which uses line breaks and indentation to make the result more readable).

# KEYBOARD SCRIPT INTERFACE

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

---

The `IKeyboardObjectType` interface derives from IObjectClass to add APIs specific to the Keyboard plugin.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.keyboard` instead of the named object.

## Examples

See the Simple keyboard movement example for a basic demonstration of using the `isKeyDown()` method to move a Sprite.

## Events

To detect key press events, use the `"keyup"` and `"keydown"` events fired on the Runtime script interface.

## Keyboard APIs

---

**isKeyDown(keyStringOrWhich)**

Return a boolean indicating if the specified keyboard key is currently being held down. The key can be specified either by its numeric code, corresponding to the KeyboardEvent.which property, or a string identifying the physical key, corresponding to the KeyboardEvent.code property (see also KeyboardEvent: code values).

> *Using numeric codes is now deprecated, so it's recommended to use a string for the key instead.*

# LIST SCRIPT INTERFACE

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

---

The `IListInstance` interface derives from IDOMInstance to add APIs specific to the List plugin.

## List events

See instance event for standard instance event object properties.

---

**"click"**

Fired when the list is clicked.

---

**"dblclick"**

Fired when the list is double-clicked.

---

**"selectionchange"**

Fired when there is any change to the selected item or items.

## List APIs

---

**addItem(text)**

Append a new item with the given text to the end of the list.

---

**insertItem(index, text)**

Insert a new item at a zero-based index in the list with the given text.

---

**setItemText(index, text)**

Set the item text at a zero-based index in the list.

---

**getItemText(index)**

Return a string of the current item text at a zero-based index in the list.

---

**removeItem(index)**

Delete an item at a zero-based index from the list.

**clear()**

Remove all items from the list, leaving the list empty.

**itemCount**

A read-only number representing how many items are in the list.

**selectedIndex**

Set or get a number indicating the zero-based index of the currently-selected list item.

**selectedCount**

A read-only number with the number of selected items. This is usually only useful with multi-select lists.

**getSelectedIndexAt(index)**

**getSelectedTextAt(index)**

Return the item index or the item text of a selected item by its index up to `selectedCount`. This is usually only useful with multi-select lists.

**tooltip**

A tooltip that appears if the user hovers the mouse over the list and waits. An empty string indicates no tooltip.

# MOUSE SCRIPT INTERFACE

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

---

The `IMouseObjectType` interface derives from IObjectClass to add APIs specific to the Mouse plugin.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.mouse` instead of the named object.

## Examples

The Shooting code example demonstrates pointing the player at the mouse cursor and shooting bullets towards it.

The Tracking pointers example also demonstrates how both mouse and touch input can be tracked simultaneously using pointer events.

## Events

To detect mouse button events, use the events fired on the Runtime script interface, such as `"mousedown"` or `"pointerdown"`.

## Mouse APIs

---

**getMouseX(layerNameOrIndex)**

**getMouseY(layerNameOrIndex)**

**getMousePosition(layerNameOrIndex)**

Return the current position of the mouse cursor on a layer, given by a case-insensitive string of its name or zero-based index. The parameter can be omitted for the default mouse position, which does not take in to account any specific layer's transformations. `getMousePosition()` returns both the X and Y position as `[x, y]`.

---

**isMouseButtonDown(button)**

Return a boolean indicating if the given mouse button is currently down. The button is specified the same way as the MouseEvent.button property, i.e. 0 for left, 1 for middle, and 2 for right.

---

**setCursorStyle(style)**

Set the appearance of the mouse cursor to a string of a CSS `cursor` style value, e.g. `"crosshair"`. See cursor styles on MDN for some possible values.

---

### setCursorObjectClass(objectClass)

Set the appearance of the mouse cursor to the current image of an IObjectClass. Various limitations apply: the object image is used as it appears in the image editor, not taking in to account size or rotation in the layout; the image cannot be too large (64x64 is usually the limit); the cursor may not be applied close to the edges of the browser window; and support varies depending on browser and OS. Some sample code for this method is shown below.

```
runtime.mouse.setCursorObjectClass(runtime.objects.Sprite);
```

# MULTIPLAYER SCRIPT INTERFACE

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

---

The `IMultiplayerObjectType` interface derives from IObjectClass to provide APIs specific to the Multiplayer object.

Note this class derives from the object class interface, not the instance interface. Typically it is accessed with `runtime.objects.Multiplayer`.

Designing multiplayer games is a complex topic. There is more documentation about the way the Multiplayer object works in the Multiplayer object manual entry. There is further learning material in the Online multiplayer in Construct tutorial series. This documentation covers only the scripting APIs to access the various multiplayer features Construct provides.

## API organisation

The multiplayer APIs broadly fall in to two categories:

1. Signalling APIs, which make use of the signalling server to meet other peers and establish connections to them
2. The main multiplayer APIs, which are used for peer-to-peer communication once connections have been established.

To clearly separate these usages, signalling APIs are available on a dedicated signalling interface at `runtime.objects.Multiplayer.signalling`, whereas the remaining multiplayer APIs are available on the main multiplayer interface at `runtime.objects.Multiplayer`.

There are also some statistics under `runtime.objects.Multiplayer.stats`, but usage of those is optional.

## Examples

See the Multiplayer scripting example for a demonstration of using these multiplayer APIs in JavaScript code.

## Signalling events

These events are fired on `runtime.objects.Multiplayer.signalling`.

---

**"connected"**

Triggered when a connection to the signalling server has successfully been established. The event object includes the properties:

- `myId`: the peer ID assigned to the local user
- `serverVersion`: a string with the signalling server software version
- `serverName`: a string of the signalling server name
- `serverOperator`: a string identifying the signalling server operator
- `serverMOTD`: a "message of the day" string chosen by the server operator

---

**"login"**

Fired upon successfully logging in to the signalling server. The event object `myAlias` property provides the alias assigned to the local user (which will be the alias requested, unless the alias is already in use, in which case the signalling server will have assigned a different alias derived from the requested one).

---

**"join"**

Fired upon successfully joining a room on the signalling server. The event object includes the properties:

- `isHost`: a boolean indicating whether the local user is the host of the room. The first peer to join a room is assigned the host.
- `hostId`: the peer ID of the room's host. This only needs to be referred to when `isHost` is false (since if the local user is the host, the host ID is their own peer ID).
- `hostAlias`: the alias of the room's host.
- `room`: a string of the room name that was joined. (This can only be different to the requested room name when auto-joining a room.)

---

**"leave"**

Fired upon successfully leaving a room on the signalling server.

---

**"disconnected"**

Fired when lost connection to the signalling server.

**"kicked"**

Fired when forcibly removed from the current room on the signalling server. This is similar to the `"leave"` event, but the cause of leaving the room was server-initiated rather than peer-initiated.

**"error"**

Fired if an error occurs while using the signalling server. The event object `message` property provides an error message (if any).

## Signalling APIs

These methods are available on `runtime.objects.Multiplayer.signalling`.

**addEventListener(eventName, callback)**

**removeEventListener(eventName, callback)**

Add or remove a callback function for a signalling event. See *Signalling events* above for more information.

**async connect(url = "wss://multiplayer.construct.net")**

Initiate a connection to a signalling server. The default URL is the official Scirra-hosted signalling server. The method can be awaited and resolves at the same time the signalling `"connected"` event fires, resolving with an object with the same properties as the event object.

**disconnect()**

Disconnect from the signalling server. This can be done once peer-to-peer connections are established if the signalling server is no longer necessary, but note that will prevent any new peers from joining the game late.

**isConnected**

A read-only boolean indicating whether a connection to the signalling server is currently active.

**addICEServer(url, username, credential)**

Add a custom Interactive Connectivity Establishment (ICE) server used by WebRTC to establish connections between peers. These can include STUN and TURN servers. A username and credential can also be optionally provided if the server requires them. This method should be called on startup, before any connections are made.

### async login(alias)

Attempt to log in to the signalling server and request to use the provided alias. The method can be awaited and resolves at the same time as the signalling `"login"` event, resolving with an object with the same properties as the event object.

### isLoggedIn

A boolean indicating whether the user is currently logged in on the signalling server.

### async joinRoom(game, instance, room, maxPeers = 0)

Join a specific room in the given game instance. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. `maxPeers` can be used to limit the number of peers that join. Only the host's value is used. If the room is full, subsequently joining peers will receive a "room full" error. The peer count includes the host, so 2 is the minimum value, or it can be left as 0 to allow an unlimited number of peers to join. The method can be awaited and resolves at the same time as the signalling `"join"` event, resolving with an object with the same properties as the event object.

### async autoJoinRoom(game, instance, room, maxPeers = 2, isLocking = true)

Join the first available room with the given game, instance and first room name. The player must be connected and logged in to the signalling server. The first player to join a room becomes the host. When rooms are full, the signalling server will create a new room. For example if "myroom" is full, it will try "myroom2", "myroom3", etc. This effectively arranges all joining peers in to games of a particular size. If `isLocking` is true, then the room is locked when full. In that case late-joiners are not allowed; if left unlocked and a peer leaves after the game starts, a newly joining peer may be added back to the game to top it up to `maxPeers` again. This method works similarly to `joinRoom()` in that it can be awaited and resolves when the `"join"` event fires.

### async leaveRoom()

Request to leave the current room on the signalling server. This method can be awaited and resolves at the same time the signalling `"leave"` event fires.

### async requestGameInstanceList(game)

Request a list of active game instances within the given game. A promise is returned which resolves when the response is received with an array of objects describing each game instance, with the object properties:

- `name`: the game instance name
- `peerCount`: the total number of peers in that game instance

---

**async requestRoomList(game, instance, type = "all")**

Request a list of active rooms within a given game instance. The returned rooms depends on the `type`: `"all"` includes all rooms; `"unlocked"` includes only rooms which are unlocked; and `"available"` includes only rooms which are available to join (unlocked and not full). A promise is returned which resolves when the response is received with an array of objects describing each room, with the object properties:

- `name`: the room name
- `peerCount`: the number of peers in the room
- `maxPeerCount`: the maximum number of peers allowed in the room, or 0 for unlimited
- `state`: the room state, one of `"available"`, `"locked"` or `"full"`.

## Multiplayer events

These events are fired on `runtime.objects.Multiplayer`.

---

**"peerconnect"**

Fired when a peer joins the same room. It also fires once per peer already in the room when joining an existing room, including the host. The event object includes the properties:

- `peerId`: the ID of the connected peer
- `peerAlias`: the alias of the connected peer

---

**"peerdisconnect"**

Fired when an existing peer disconnects from the room. The event object includes the properties:

- `peerId`: the ID of the disconnected peer
- `peerAlias`: the alias of the disconnected peer
- `leaveReason`: a string with an optional reason provided for the peer disconnecting

---

**"message"**

Fired when a message is received over the network. Note the order messages are received, or whether a sent message is received at all, depends on the reliability mode chosen when the message was originally sent. The event object includes the properties:

- `fromId`: the ID of the peer the message was sent by
- `fromAlias`: the alias of the peer the message was sent by
- `message`: the content of the message. This is either a string, JSON data, or an ArrayBuffer for binary content, depending on the type of the message sent.
- `transmissionMode`: the transmission mode the message was sent with.

---

**"kicked"**

Fired if kicked from the current room. This can occur if the host quits, the connection to the host could not be established, or the host otherwise decides to forcibly remove you from the room. After this fires the player is no longer in the room and must re-join a room to be able to participate in a game.

## Multiplayer APIs

These methods and properties are available on `runtime.objects.Multiplayer`.

---

**signalling**

Provides the signalling interface - see *Signalling APIs* above.

---

**stats**

Provides the statistics interface - see *Statistics APIs* below.

---

**isHost**

A read-only boolean indicating if the current peer is the room host.

---

**myId**

**myAlias**

Read-only strings with the peer ID and alias of the local user.

---

**hostId**

**hostAlias**

Read-only strings with the peer ID and alias of the room host (which will be the same as `myId` and `myAlias` if `isHost` is true).

---

**currentGame**

**currentGameInstance**

**currentRoom**

Read-only strings identifying the current game, game instance, and room.

---

**peerCount**

Read-only number of connected peers, including the local user.

---

**getAllPeers()**

Return an array of `IMultiplayerPeer` representing every peer in the room, including the local user. See *Peer APIs* below.

---

**getPeerById(peerId)**

Return a `IMultiplayerPeer` for a peer in the current room by their peer ID, or returns `null` if they don't exist. See also *Peer APIs* below.

---

**sendPeerMessage(peerId, message, transmissionMode = "o")**

Send a message over the network to a peer in the same room identified by their peer ID. The message can be a string, an object for JSON transmission (which must be convertible to a string), or an ArrayBuffer for binary content. The transmission mode can be one of `"o"` for reliable ordered, `"r"` for reliable unordered, or `"u"` for unreliable (see the Multiplayer object documentation for more details about reliability modes). When received the `"message"` event will be fired.

---

**hostBroadcastMessage(fromId, message, transmissionMode = "o")**

This is similar to `sendPeerMessage()` but can only be called by the host, and the provided message will be sent to every other peer in the room. `fromId` can optionally be set to another peer ID to make it appear that the message is from that peer, which is useful when relaying messages through the host; if left empty it will use the host ID.

---

**disconnectRoom()**

Disconnects from any peers in the current room and also leaves the room on the signalling server. If the current user is the room host, all other peers are kicked.

---

**simulateLatency(latency, pdv, loss)**

Simulate latency, PDV and packet loss on all inbound and outbound messages. This can be useful for making local testing more realistic, since unlike the Internet latency is effectively non-existent. For local testing it is only necessary to simulate latency on the host, since that guarantees every message in the game will have delay added; it is not necessary to also simulate latency on the peers. The latency for an individual message is calculated as the latency plus a random value from zero to the PDV. The packet loss indicates the chance an unreliable message is lost entirely, or in the case of reliable messages that retransmission is necessary and the latency is multiplied.

## Peer APIs

The `IMultiplayerPeer` interface represents a connected peer in the same room. It is returned by methods like `getAllPeers()` and `getPeerById()`.

---

**id**

The peer ID for this peer.

---

**alias**

The alias for this peer.

---

**isHost**

A boolean indicating if this peer is the room host.

---

**isMe**

A boolean indicating if this peer represents the local user.

---

**latency**

**pdv**

Get the measured latency and packet delay variation (PDV) on the network connection to this peer. Note peers can only use this to get the stats for the host, since that is the only connection they have, but the host can use it for any peer.

---

**send(message, transmissionMode = "o")**

This is a shorthand for calling `sendPeerMessage()` with this peer's ID.

## Statistics APIs

These properties are available under `runtime.objects.Multiplayer.stats`.

---

**inboundBandwidth**

**outboundBandwidth**

Read-only numbers with the total estimated inbound and outbound bandwidth for all data transmission over the network in bytes per second. When automatic data compression is in use, this measures the compressed size of the data actually sent over the network.

---

**inboundDecompressedBandwidth**

**outboundDecompressedBandwidth**

Read-only numbers with the total estimated decompressed inbound and outbound bandwidth for all data sent and received via the Multiplayer object in bytes per second. When automatic data compression is in use, this measures the size of the decompressed messages, which may be significantly larger than the data actually sent over the network.

---

**inboundCount**

**outboundCount**

Read-only numbers of the total number of separate inbound and outbound messages sent and received. This includes internally-used messages for things like ping and synchronisation; generally the bandwidth is the more practically useful statistic.

# PARTICLES SCRIPT INTERFACE

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

The `IParticlesInstance` interface derives from IWorldInstance to add APIs specific to the Particles plugin.

## Particles APIs

### isSpraying

A boolean indicating whether the object is actively emitting particles (when in *Continuous spray* mode).

### rate

Set or get the number of particles created per second.

> *In One-shot mode this property can be assigned immediately upon creation to adjust the number of particles created, but once particles are already created assigning the value will have no effect.*

### sprayCone

Set or get the angle in radians through which particles are fired.

### initSpeed

Set or get the initial speed of particles in pixels per second.

### initSize

Set or get the initial size of each particle in pixels.

### initOpacity

Set or get the initial opacity of each particle, as a float from 0 to 1.

### initXRandom

### initYRandom

Set or get the range of random offsets to the particle's position on each axis.

**initSpeedRandom**

Set or get the range of random offset to the particle's speed on creation.

**initSizeRandom**

Set or get the range of random offset to the particle's size on creation.

**initGrowRate**

Set or get the initial grow rate (change in size over time) for each particle, in pixels per second. Zero will keep the same size over time, a positive value will increase the size of the particle over time, and a negative value will shrink it over time.

**initGrowRandom**

Set or get the range of random offset to the particle's grow rate on creation.

**acceleration**

Set or get the acceleration of each particle, in pixels per second per second.

**gravity**

Set or get the downwards acceleration caused by gravity, in pixels per second per second.

**lifeAngleRandom**

Set or get an amount of random change to each particle's angle to apply during its lifetime, in radians.

**lifeSpeedRandom**

Set or get an amount of random change to each particle's speed to apply during its lifetime, in pixels per second.

**lifeOpacityRandom**

Set or get an amount of random change to each particle's opacity to apply during its lifetime, in the range 0-1.

**timeout**

Set the time in seconds each particle can last before being destroyed when the *Destroy mode* is *Timeout*.

**setParticleObjectClass(iObjectClass)**

Call with an IObjectClass to set the Particles object to spawn instances of that object class instead of drawing its own particles. Pass `null` to restore the default behavior of the Particles object drawing its own particles. For more information see *Advanced particle effects* in the Particles manual entry.

---

**fastForward(time)**

Skip ahead the particle effect by a time in seconds. For example fast-forwarding by 3 seconds will cause the Particles object to instantly spawn, move and destroy particles as if 3 seconds had gone by. This is useful for making sure particle effects appear ready immediately, rather than taking a few seconds to move their particles out from the spawn point.

# PROGRESS BAR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/progress-bar

The `IProgressBarInstance` interface derives from IDOMInstance to add APIs specific to the Progress Bar plugin.

## Progress Bar events

See instance event for standard instance event object properties.

### "click"

Fired when the progress bar is clicked.

## Progress Bar APIs

### progress

Set or get the current progress value, from 0 to the maximum.

### maximum

Set or get the maximum progress value, at which the progress bar is shown full indicating a completed operation.

### setIndeterminate()

Set the progress bar in to an "indeterminate" state, intended to indicate that it is working, but the progress is unknown.

> *In indeterminate mode the progress and maximum values are both set to 0.*

### tooltip

The string used as the tooltip for the progress bar.

# SHADOW LIGHT SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/shadow-light

The `IShadowLightInstance` interface derives from IWorldInstance to add APIs specific to the Shadow Light plugin.

## Shadow Light APIs

### lightX

### lightY

Set or get the position in the layout from which shadows are cast from.

> *Note that using `IWorldInstance` methods to set the object position will also update the light position. However the Shadow Light object automatically positions itself in the middle of the viewport in order to draw over the whole screen. Setting the object position to set the light position in the middle of the viewport may conflict with its automatic positioning, so setting these properties can be used as a more reliable way to guarantee the light position is placed at the given location.*

### lightHeight

Set or get the height of the light, used with the shadow caster object heights to calculate the length of shadow to cast. This property only has an effect if the light radius is 0, otherwise shadows always extend offscreen.

### shadowColor

Set or get the color of the shadows drawn by the light, as an array with 3 elements specifying the red, green and blue components as floats in the 0-1 range.

### tag

Set or get a string of the tag for this light. Combined with `castFrom` this allows using multiple lights that cast shadows of different sets of objects.

### castFrom

A string specifying which shadow caster objects to render shadows for from this object. The possible values are:

- `"all"`: every shadow caster object will get a shadow rendered for this light.
- `"same-tag"`: shadows will only be rendered for shadow casters with the same `tag`.
- `"different-tag"`: shadows will only be rendered for shadow casters with a different `tag`.

# SLIDER BAR SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/slider-bar

The `ISliderBarInstance` interface derives from IDOMInstance to add APIs specific to the Slider Bar plugin.

## Slider Bar events

See instance event for standard instance event object properties.

**"click"**

Fired when the control is clicked.

**"change"**

Fired when the user finishes changing the chosen value on the slider bar. Typically this only fires when the user releases a mouse button or touch after moving the slider.

**"input"**

Fired repeatedly as the user changes the chosen value on the slider bar. Unlike the `"change"` event this will reflect the current value of the slider as the user is still dragging it.

## Slider Bar APIs

**value**

The current value represented by the slider bar.

**minimum**

**maximum**

The minimum and maximum values, defining the range of the slider bar.

**step**

The increment of possible values. For example if the step is 10, then the slider will jump in units of 10 as it is moved, and only a multiple of 10 can be chosen as a value.

**tooltip**

A tooltip that appears if the user hovers the mouse over the text box and waits. An empty string indicates no tooltip.

---

**isEnabled**

A boolean indicating if the control is enabled or disabled.

# SPRITE SCRIPT INTERFACE

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

---

The `ISpriteInstance` interface derives from IWorldInstance to add APIs specific to the Sprite plugin.

Sprite object types also define a `ISpriteObjectType` interface which derives from IObjectClass to add Sprite APIs that affect the object type, such as dynamic animations.

## Sprite instance events

See instance event for standard instance event object properties.

---

**"framechange"**

Fired when the currently displayed animation frame changes during playback of an animation. The event object has the additional properties:

- `animationName`: a string of the name of the currently playing animation
- `animationFrame`: a zero-based index with the animation frame number of the new frame in its animation

---

**"animationend"**

Fired when playback of the current animation reaches the end. The event object has the additional properties:

- `animationName`: a string of the name of the animation that finished

## Sprite instance APIs

---

**animation**

A reference to a IAnimation script interface representing the current animation, which can be used to access additional details such as the frames in the animation.

---

**setAnimation(name, from = "beginning")**

Set the current animation by a string of its name (case-insensitive). If the animation name does not exist, an exception will be thrown. `from` can be set to either `"current-frame"` to

switch to the same frame index in the new animation, or `"beginning"` to rewind to the first frame.

> *Note that if the animation being set is already playing, this method does nothing, even if set to play from the beginning. If you intend to restart the animation, use `startAnimation("beginning")` instead.*

---

**getAnimation(name)**

Get an IAnimation for an animation in the Sprite object by a case-insensitive string of its name. Returns `null` if no animation is found.

---

**animationName**

A read-only string of the current animation name. Use the `setAnimation()` method to change the animation.

---

**startAnimation(from = "current-frame")**

Start playback of the current animation. `from` can be set to either `"current-frame"` to play from the existing frame, or `"beginning"` to play from the first frame.

---

**stopAnimation()**

Stop playback of the current animation.

---

**animationFrame**

The zero-based index of the current animation frame.

---

**animationFrameTag**

A string of the current animation frame tag (which is an empty string when not set). If assigned a new tag, and the current animation has multiple animation frames with the same tag, then it will use the first one.

---

**animationSpeed**

The current animation playback speed, in animation frames per second.

---

**animationRepeatToFrame**

The zero-based index of the animation frame to rewind to when repeating an animation.

---

**imageWidth**

**imageHeight**

**getImageSize()**

Read-only numbers indicating the size of the current animation frame's source image, in pixels. The method allows getting both values at the same time.

---

**getImagePointCount()**

Return the number of image points on the current animation frame.

---

**getImagePointX(nameOrIndex)**

**getImagePointY(nameOrIndex)**

**getImagePointZ(nameOrIndex)**

**getImagePoint(nameOrIndex)**

Return the location of an image point on the current animation frame in layout co-ordinates. The Z co-ordinate can be useful when using 3D mesh distortion. Image points are identified either by a case-insensitive string of their name, or their index. Note image point 0 is the origin, so index 1 is the first image point. If the image point is not found, this returns the origin instead. The `getImagePoint` variant returns `[x, y, z]`.

---

**getPolyPointCount()**

Return the number of collision polygon points on the current animation frame.

---

**getPolyPointX(index)**

**getPolyPointY(index)**

**getPolyPoint(index)**

Return the location of a collision polygon point on the current animation frame in layout co-ordinates, by its zero-based index. The `getPolyPoint` variant returns `[x, y]`.

> *The first poly point is repeated again at the end (at the index `getPolyPointCount()`) since it makes it easier to iterate through each edge of the collision polygon.*

---

**setSolidCollisionFilter(isInclusive, tags)**

Enable or disable collisions with the Solid behavior according to tags. Specify `tags` using a string of space-separated tag names. If `isInclusive` is true, collisions are only enabled with solids that match any of the given tags; if no tags are specified, collisions are disabled with all solids. If `isInclusive` is false, collisions are disabled with solids that match any of the given tags; if no tags are specified, collisions are enabled for all solids (the default). See also the `tags` property of ISolidBehaviorInstance.

### async replaceCurrentAnimationFrame(blob)

Replace the current animation frame image with the contents of a Blob representing an image file such as a PNG image. The blob can be locally generated or retrieved from a URL, for example:

```
// Loading an image from a URL
const response = await fetch(url);
const blob = await response.blob();
await spriteInst.replaceCurrentAnimationFrame(blob);
```

## Sprite object type APIs

These `ISpriteObjectType` APIs are available on the object type class, e.g. `runtime.objects.MySprite`, rather than instances. Note this means that any changes, such as to the animations or animation frames, will affect all instances.

### getAnimation(name)

Return an IAnimation representing the animation with the given name, or `null` if none exists.

### getAllAnimations()

Return an array of `IAnimation` representing all animations that the Sprite object type has.

### addAnimation(animName)

Add a new animation with the given name and return an `IAnimation` representing it. The name must be unique. The added animation will have a single transparent animation frame sized 100x100.

### removeAnimation(animName)

Remove an animation with the given name. An exception will be thrown if an animation with the given name does not exist, or the specified animation is the last one, as Sprite objects must have at least one animation.

### addAnimationFrame(animName, where)

Add an animation frame to the animation with the specified name. The new frame will be transparent and sized 100x100. The `where` parameter must be either a number for a zero-based index of where to insert the frame, and can be -1 to add to the end, or a string of an animation frame tag to insert relative to. When adding an animation frame not at the end, it is inserted just before the given frame. The method returns an IAnimationFrame representing the added frame.

**removeAnimationFrame(animName, where)**

Remove an animation frame from the animation with the specified name. The last frame cannot be removed, as animations must have at least one frame. The `where` parameter must be either a number for a zero-based index of where to remove a frame, and can be -1 to remove from the end, or a string of an animation frame tag to remove.

# SPRITEFONT SCRIPT INTERFACE

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

The `ISpriteFontInstance` interface derives from IWorldInstance to add APIs specific to the Sprite font plugin.

## SpriteFont APIs

**text**

The string currently displayed by the Sprite Font object.

**typewriterText(str, duration)**

Set the text over time by starting with an empty string and gradually adding characters until the full text of `str` is written out, over a `duration` specified in seconds. Note modifying the `text` property while text is being written out will cancel the effect.

**typewriterFinish()**

If text is being written out with the `typewriterText()` method, force it to finish immediately.

**characterScale**

The current text scale, defaulting to 1 for normal scale.

**characterSpacing**

The extra space in pixels to add horizontally between characters.

**lineHeight**

The extra space in pixels to add vertically between lines. 0 is the default size, negative values make lines closer together, and positive values space lines out further apart.

**horizontalAlign**

A string specifying the horizontal alignment of the text within the object bounding box, which must be one of `"left"`, `"center"` or `"right"`.

**verticalAlign**

A string specifying the vertical alignment of the text within the object bounding box, which must be one of `"top"`, `"center"` or `"bottom"`.

---

**wordWrapMode**

A string specifying the way to wrap text when it reaches the end of a line. This can be either `"word"` to wrap entire space-separated words, `"character"` to wrap at any character, or `"cjk"` to wrap at any character but with special handling for Chinese, Japanese and Korean punctuation.

---

**textWidth**

**textHeight**

**getTextSize()**

Read-only values indicating the size of the actual text content within the Sprite Font object's rectangle. The method allows getting both values at the same time.

---

**hasTagAtPosition(tag, x, y)**

Return a boolean indicating if there is text with a specific tag at the given position (case insensitive). For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then testing if the tag "mytag" is at a given position return `true` if that position is over just the part of the text that says "world", else `false`.

---

**getTagAtPosition(x, y)**

Look up the tag for a part of the text at a given position and return the tag if any, else return an empty string if no tag is specified. For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then the tag at a position over the word "world" is "mytag", and the tag at a position over the word "Hello" is "".

---

**getTagCount(tag)**

**getTagPositionAndSize(tag, index)**

Get the number of fragments, and the size and position of each fragment by its zero-based index, for a given tag. Note that a single tag may be broken in to multiple fragments - see the section *Tagged range fragmentation* in the Text object manual entry for more details (which applies equally to SpriteFonts). The `getTagPositionAndSize()` method returns the position and size as an object with the properties `{x, y, width, height}`.

# SVG PICTURE SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/svg-picture

The `ISVGPictureInstance` interface derives from IWorldInstance to add APIs specific to the SVG Picture plugin.

## SVG Picture APIs

### svgUrl

Get the path to the SVG image to display in the object.

> *This property can also be assigned, but updating the image is actually asynchronous, so it is preferable to use `setSvgUrl()` which returns a promise.*

### async setSvgUrl(url)

Set the path to the SVG image to display in the object. Note that updating the image is asynchronous, so this is an async method and can be awaited to ensure the image has been updated.

# TEXT SCRIPT INTERFACE

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

The `ITextInstance` interface derives from IWorldInstance to add APIs specific to the Text plugin.

## Text APIs

**text**

The string currently displayed by the Text object.

**typewriterText(str, duration)**

Set the text over time by starting with an empty string and gradually adding characters until the full text of `str` is written out, over a `duration` specified in seconds. Note modifying the `text` property while text is being written out will cancel the effect.

**typewriterFinish()**

If text is being written out with the `typewriterText()` method, force it to finish immediately.

**fontColor**

An array with 3 elements specifying the red, green and blue color of the text, with color values as floats in the 0-1 range.

> *Try not to confuse this with the IWorldInstance property `colorRgb`, which applies a color tint to the overall appearance of the object.*

**fontFace**

A string specifying the font used to display the text, e.g. "Arial".

**isBold**

**isItalic**

Booleans indicating whether the font face is displayed with bold or italic styles.

**sizePt**

The size of the displayed text, in points (pt).

---

**lineHeight**

The extra space in pixels to add vertically between lines. 0 is the default size, negative values make lines closer together, and positive values space lines out further apart.

---

**horizontalAlign**

A string specifying the horizontal alignment of the text within the object bounding box, which must be one of `"left"`, `"center"` or `"right"`.

---

**verticalAlign**

A string specifying the vertical alignment of the text within the object bounding box, which must be one of `"top"`, `"center"` or `"bottom"`.

---

**readAloud**

A boolean indicating whether the contents of the text object will be read aloud by screen reader software. See the *Read aloud* property of the Text object for more details.

---

**textDirection**

Set or get a string of either `"ltr"` (left-to-right) or `"rtl"` (right-to-left) specifying the direction of the text flow. See the *Text direction* property of the Text object for more details.

---

**wordWrapMode**

A string specifying the way to wrap text when it reaches the end of a line. This can be either `"word"` to wrap entire space-separated words, `"character"` to wrap at any character, or `"cjk"` to wrap at any character but with special handling for Chinese, Japanese and Korean punctuation.

---

**setFixedResolutionMode(scale)**

**setAutoResolutionMode()**

By default Text objects use *Automatic* resolution mode, which means the resolution of the text adjusts according to the 2D display scale. This produces the best quality display, but only works with 2D display, and can also cause the text to constantly re-render when being smoothly scaled, which can sometimes have a significant performance overhead. *Fixed* resolution mode causes the text to render at a fixed resolution according to a provided scale factor, and ignore the display scale. This usually results in a reduced display quality but better performance.

**textWidth**

**textHeight**

**getTextSize()**

Read-only values indicating the size of the actual text content within the text object's rectangle. The method allows getting both values at the same time.

---

**hasTagAtPosition(tag, x, y)**

Return a boolean indicating if there is text with a specific tag at the given position (case insensitive). For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then testing if the tag "mytag" is at a given position return `true` if that position is over just the part of the text that says "world", else `false`.

---

**getTagAtPosition(x, y)**

Look up the tag for a part of the text at a given position and return the tag if any, else return an empty string if no tag is specified. For example if the text has the BBcode `Hello [tag=mytag]world[/tag]`, then the tag at a position over the word "world" is "mytag", and the tag at a position over the word "Hello" is "".

---

**getTagCount(tag)**

**getTagPositionAndSize(tag, index)**

Get the number of fragments, and the size and position of each fragment by its zero-based index, for a given tag. Note that a single tag may be broken in to multiple fragments - see the section *Tagged range fragmentation* in the Text object manual entry for more details. The `getTagPositionAndSize()` method returns the position and size as an object with the properties `{x, y, width, height}`.

---

**changeIconSet(objectClass)**

Changes the *Icon set* property, replacing the Sprite used for BBcode icons to the one specified by the given IObjectClass (which must be from a Sprite object). This can be used to change the set of icons displayed by the Text object. Note if the new Sprite object does not have the same number of animation frames, or the same animation frame tags, then some icons may disappear.

---

**async getAsHtmlString()**

Converts the contents of the Text object, including any icons, in to a string of HTML. This process is asynchronous so the method must be awaited. It resolves with a string of HTML code. The result is cached (and updates the *AsHTML* expression) so repeat calls will resolve with the same HTML string, until the Text object contents is modified, after which the next call will regenerate the HTML string again.

# TEXT INPUT SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/text-input

The `ITextInputInstance` interface derives from IDOMInstance to add APIs specific to the Text Input plugin.

## Text Input events

See instance event for standard instance event object properties.

**"click"**

**"dblclick"**

Fired when the control is clicked or double-clicked.

**"change"**

Fired when user input causes the `text` property to change.

## Text Input APIs

**text**

The current string entered in the input field.

**placeholder**

A string of text that appears faintly when the field is empty. This can be used for hints for what the field is for, e.g. *Username*.

**tooltip**

A tooltip that appears if the user hovers the mouse over the text box and waits. An empty string indicates no tooltip.

**isEnabled**

A boolean indicating if the control is enabled or disabled.

**isReadOnly**

A boolean indicating if the input field is read-only, which means the text cannot be modified but can still be selected. This is different to disabling the field, where text cannot be selected.

**scrollToBottom()**

Scroll to the bottom of the control. Only has an effect when set to the *textarea* type, since it is the only multiline mode. This is useful for chat or log style textareas.

**maxLength**

Set or get the maximum number of characters allowed to be entered in the field. The value -1 indicates no limit, which is the default.

# TILED BACKGROUND SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/tiled-background

---

The `ITiledBackgroundInstance` interface derives from IWorldInstance to add APIs specific to the Tiled Background plugin.

## Tiled Background APIs

---

**imageWidth**

**imageHeight**

**getImageSize()**

The original dimensions of the Tiled Background's current image in pixels. This does not include tiling - it returns the size as shown in Construct's image editor. The method allows retrieving both values at the same time.

---

**imageOffsetX**

**imageOffsetY**

**setImageOffset(imageOffsetX, imageOffsetY)**

**getImageOffset()**

The offset of the displayed Tiled Background image in pixels. The methods allow setting or getting both values at the same time.

---

**imageScaleX**

**imageScaleY**

**setImageScale(imageScaleX, imageScaleY)**

**getImageScale()**

The scale of the displayed Tiled Background image, defaulting to 1 for original size. The methods allow setting or getting both values at the same time.

---

**imageAngle**

The angle of the displayed Tiled Background image in radians. If this is changed, `imageAngleDegrees` updates accordingly.

---

**imageAngleDegrees**

The angle of the displayed Tiled Background image in degrees. If this is changed, `imageAngle` updates accordingly.

---

**enableTileRandomization**

A boolean indicating whether tile randomization is enabled.

---

**tileXRandom**

**tileYRandom**

**setTileRandom(tileXRandom, tileYRandom)**

**getTileRandom()**

When tile randomization is enabled, the amount of random horizontal and vertical offset to use, as a percentage in the range 0-1. The methods allow setting or getting both values at the same time.

---

**tileAngleRandom**

When tile randomization is enabled, the amount of random rotation to use, as a percentage in the range 0-1.

---

**tileBlendMarginX**

**tileBlendMarginY**

**setTileBlendMargin(tileBlendMarginX, tileBlendMarginY)**

**getTileBlendMargin()**

When tile randomization is enabled, the percentage of the tile width or height which will fade in to the adjacent tile, as a percentage in the range 0-1. The methods allow setting or getting both values at the same time.

---

**async replaceImage(blob)**

Replace the current image with the contents of a Blob representing an image file such as a PNG image. The blob can be locally generated or retrieved from a URL, for example:

```
// Loading an image from a URL
const response = await fetch(url);
const blob = await response.blob();
await tbInst.replaceImage(blob);
```

# TILEMAP SCRIPT INTERFACE

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

The `ITilemapInstance` interface derives from IWorldInstance to add APIs specific to the Tilemap plugin.

## Tile numbers

Tiles in the tilemap are represented as a single 32-bit integer number and can be rotated and flipped. To support this they consist of two parts using a bitmask:

- The tile ID in the lower 29 bits - this is the number of the tile as shown in the Tilemap Bar when hovering the tile
- Tile flags in the upper 3 bits

There is also a special tile number **-1** indicating an empty tile.

### Tile flags

The Tilemap script interface exposes the following flags and masks which can be used to manipulate tile numbers:

```
ITilemapInstance.TILE_FLIPPED_HORIZONTAL = -0x80000000;
ITilemapInstance.TILE_FLIPPED_VERTICAL = 0x40000000;
ITilemapInstance.TILE_FLIPPED_DIAGONAL = 0x20000000;
ITilemapInstance.TILE_FLAGS_MASK = 0xE0000000;
ITilemapInstance.TILE_ID_MASK = 0x1FFFFFFF;
```

For example, to flip tile ID 2 horizontally, you would use bitwise OR combining the tile ID and the flag, e.g. `2 | ITilemapInstance.TILE_FLIPPED_HORIZONTAL`. Similarly you can test if the bit is set using `tile & ITilemapInstance.TILE_FLIPPED_HORIZONTAL`.

You can also use the masks to extract each component of the tile number. For example `tile & ITilemapInstance.TILE_ID_MASK` will return just the tile ID, since it removes all the flag bits.

> *Be sure to first check if the tile is the special value **-1** indicating an empty tile. This is a special value that doesn't use the bit representation so won't work when combined with flags or masks.*

## Tilemap APIs

**mapWidth**

**mapHeight**

**getMapSize()**

Read-only numbers representing the size of the tilemap in tiles. The method allows getting both values at the same time.

---

**mapDisplayWidth**

**mapDisplayHeight**

**getMapDisplaySize()**

Read-only numbers with the displayed size of the tilemap, in tiles. The method allows getting both values at the same time.

> *This can differ from* `mapWidth` *and* `mapHeight` *if the Tilemap is resized smaller at runtime - in that case the display size will be smaller, but the map size will stay the same.*

---

**tileWidth**

**tileHeight**

**getTileSize()**

Read-only numbers with the size of a tile in pixels. The method allows getting both values at the same time.

---

**getTileAt(x, y)**

Get the tile at a given position in tiles (i.e. (0, 0) is the top-left tile of the tilemap, regardless of the tilemap's position or the tile size). Returns -1 for empty tiles or tiles outside the tilemap; otherwise use bit operations to determine tile ID or flags separately.

---

**setTileAt(x, y, tile)**

Set the tile at a given position in tiles. Use -1 to set a tile empty; otherwise use bit operations to combine the tile ID and flags.

---

**async replaceImage(blob)**

Replace the current tilemap image with the contents of a Blob representing an image file such as a PNG image. The blob can be locally generated or retrieved from a URL, for example:

```
// Loading an image from a URL
const response = await fetch(url);
```

```
const blob = await response.blob();
await tmInst.replaceImage(blob);
```

# TIMELINE CONTROLLER SCRIPT INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/plugin-interfaces/timeline-controller

The `ITimelineControllerObjectType` interface derives from IObjectClass to add APIs specific to the Timeline Controller plugin.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.timelineController` instead of the named object.

## Timeline controller APIs

### play(timeline, tags)

Start playback of a timeline specified by a string of its name. The tags to associate with this playback are optional, and can be specified as a space-separated string, or an array of strings. The method returns an ITimelineState representing the playback state of the timeline.

### setInstances(instances, trackId)

Set one or more instances to be used for the next timeline playback. The `instances` parameter can be either an `IWorldInstance` or an array of `IWorldInstance` in order to set multiple instances. The instances can be different to the ones used in the editor. The instance will be set to the track with the corresponding `trackId`, specified as a string. The `trackId` is also optional: if omitted it uses the first track in the timeline. It can also be used repeatedly omitting the `trackId` to keep setting the tracks in the timeline in sequence. When the timeline is played it will then affect the specified instances instead of the ones used in the editor. This method is equivalent to the *Set instance* action; for more information see the Timeline Controller manual entry.

### *allTimelines()

Iterates all ITimelineState representing all the currently playing timelines.

### *timelinesByTags(tags)

Iterates ITimelineStates matching the specified tags, which can be specified as either a space-separated string, or an array of strings.

# TOUCH SCRIPT INTERFACE

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

The `ITouchObjectType` interface derives from IObjectClass to add APIs specific to the Touch plugin.

Note this class derives from the object class interface, not the instance interface. Typically it is used through `runtime.touch` instead of the named object.

## Examples

The Tracking pointers example demonstrates how both mouse and touch input can be tracked simultaneously using pointer events.

## Events

To detect touch input events, use the events fired on the Runtime script interface, such as `"pointerdown"`. There are also `"deviceorientation"` and `"devicemotion"` events which can be used to detect device movement.

## Touch APIs

**async requestPermission(type)**

Request permission to use device orientation or motion sensors. The `"deviceorientation"` and `"devicemotion"` events will not fire unless this method has been called and permission granted. *type* must be `"orientation"` or `"motion"`. The user may be prompted to allow permission. Note some browsers merge both types in to one permission prompt in which case only one permission request is necessary to access both orientation and motion. Returns a promise that resolves with `"granted"` if permission was allowed, else `"denied"` if the user declined.

# ADDON SDK SCRIPT INTERFACES

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces

The script interfaces in this section of the scripting reference are intended for use with Construct's Addon SDK, which allows developers to create custom plugins and behaviors. These features are not normally used when writing code in Construct projects, so this section of the documentation ought to be disregarded by anyone not using the Addon SDK.

You can learn more about the Addon SDK in the Addon SDK documentation.

## Coding conventions

As JavaScript has limited support for encapsulation, some methods which are only intended for use by the Construct runtime can still be accessed from Construct's scripting feature. With JavaScript, properties and methods are public by default, or can use private properties with the `#` prefix which can only be used by the same class. Construct needs a middle-ground status where outside callers should not use a property or method, but it must be accessible to Construct's runtime. JavaScript currently has no such language feature, and so such properties and methods have to be public. Instead Construct uses a special naming scheme: properties or methods that begin with an underscore `_` should be considered "protected", i.e. they are only for use by the Construct runtime and not should be used by any other caller. Do not use any such properties or methods other than for the purposes for which they are documented.

*Do not ignore this naming convention. If you use properties or methods beginning with an underscore in a way that is not supported, you may cause your project to become unstable, or it may break in future, and if this happens Scirra will not provide any support.*

# ILOOPINGCONDITIONCONTEXT ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/iloopingconditioncontext

---

The `ILoopingConditionContext` interface is used to implement looping conditions. It is created by the ISDKUtils method `createLoopingConditionContext()`.

## ILoopingConditionContext APIs

---

### retrigger()

Execute all subsequent conditions, actions and sub-events within this call. This essentially runs a single iteration of the looping condition.

---

### isStopped

A read-only boolean that is set to true when the loop is stopped using the system *Stop loop* action.

---

### release()

This must be called after the looping condition finishes in order to clean up resources and reset state that was applied during the looping condition.

# ISDKPLUGINBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkbehaviorbase

The `ISDKBehaviorBase` interface is used in the runtime as a base class for behaviors created with the addon SDK. It derives from IBehavior.

## ISDKBehaviorBase APIs

The `ISDKBehaviorBase` interface does not currently have any of its own APIs.

# ISDKBEHAVIORINSTANCEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkbehaviorinstancebase

---

The `ISDKBehaviorInstanceBase` interface is used as a runtime base class for behavior instances in the addon SDK. It derives from IBehaviorInstance.

## ISDKBehaviorInstanceBase APIs

---

**_postCreate()**

Optional override called after the associated object instance has finished being created. This can be useful since the behavior instance constructor is called during instance creation, so the final state is not ready yet.

---

**_release()**

Optional override for when the corresponding instance is released. Clean up any other resources that need releasing in this method. It must also call `super._release()` to release all engine resources.

---

**_getInitProperties()**

Call this in the constructor to get an array of the behavior instance's properties.

---

**_trigger(method)**

**_triggerAsync(method)**

Fire a trigger condition. The condition must be declared as a trigger in aces.json. Pass a full reference to the condition method, e.g. `this._trigger(C3.Behaviors.MyBehavior.Cnds.MyTrigger)`. The async variant returns a Promise that resolves when the trigger has finished executing, which can be used to support Construct's debugger, as it may wait if it hits a breakpoint inside the trigger.

---

**_setTicking(isTicking)**

**_setTicking2(isTicking)**

**_setPostTicking(isTicking)**

**_isTicking()**

**_isTicking2()**

### _isPostTicking()

Utility methods to start or stop the runtime calling the `_tick()`. `_tick2()` or `_postTick()` methods of your behavior instance every tick, and also to check whether ticking is active. It is recommended to stop ticking whenever the tick method is no longer needed to reduce the performance overhead of ticking. Redundant calls to start or stop ticking are ignored. The first call always takes effect (i.e. calls do not stack - if you make 3 calls to start ticking then 1 call to stop ticking, ticking is stopped).

---

### _tick()

Optional override that is called every tick just **before** events are run after `_setTicking(true)` has been called.

---

### _tick2()

Optional override that is called every tick just **after** events are run after `_setTicking2(true)` has been called.

---

### _postTick()

Optional override that is called every tick just after all other behaviors have had their `_tick()` methods called. This allows behaviors to observe the state applied by other behavior's ticking. Note you should use `_tick()` rather than `_postTick()` where possible, since it is not possible to reliably observe the state applied by other behavior's post-ticking.

---

### _getDebuggerProperties()

Override to return properties to display in the debugger. For more information see runtime scripts.

---

### _saveToJson()

Optional override to return a JSON object that represents the state of the behavior instance for savegames.

---

### _loadFromJson(o)

Optional override accepting a JSON object returned by a prior call to `_saveToJson()` that represents the state of a behavior instance to load, for savegames.

# ISDKBEHAVIORTYPEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkbehaviortypebase

The `ISDKBehaviorTypeBase` is used in the runtime as a base class for behavior types in the addon SDK (i.e. behaviors at the same level as object types). It derives from IBehaviorType.

## ISDKBehaviorTypeBase APIs

### _onCreate()

Optional override called when the runtime starts up and creates all behavior types before the project starts.

# ISDKDOMINSTANCEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkdominstancebase

The `ISDKDOMInstanceBase` interface is used as a runtime base class for DOM instances (which create a HTML element) in the addon SDK. It derives from ISDKWorldInstanceBase.

## ISDKDOMInstanceBase APIs

### _postToDOMElement(handler, data)

### _postToDOMElementAsync(handler, data)

Post a message from the runtime instance to the DOM side. The message is received using `AddDOMElementMessageHandler()` in DOMElementHandler. `handler` is a string identifying the kind of message. `data` is a JSON object that is forwarded with the message to provide additional details. The async variant returns a promise that awaits an async handler on the DOM side and forwards the return value back to the runtime, which the returned promise resolves with. The non-async variant simply posts a message and ignores the result (i.e. fire-and-forget).

### _postToDOMElementMaybeSync(handler, data)

As with `_postToDOMElement()`, but when the runtime is in DOM mode, calls the DOM handler synchronously inside the call. When the runtime is in worker mode, this still posts a message which is handled later. Usually this method is not necessary, but it can be used to work around some user input restrictions in some browsers in DOM mode only.

### _createElement(data)

Instruct the runtime to create a DOM element for this instance. It will end up calling `CreateElement()` in DOMElementHandler with `data` (an optional object with additional details to create with). The runtime associates the resulting element with this instance.

### focusElement()

### blurElement()

Helper methods to manage calling `focus()` and `blur()` on the instance's associated DOM element.

### isElementFocused()

Helper method to identify whether the associated HTML element is currently focused.

---

**setElementCSSStyle(prop, val)**

Helper method to set a CSS style on the instance's associated HTML element. For example `setElementCSSStyle("font-family", "sans-serif")` will be forwarded to `elem.style.fontFamily = "sans-serif"` on the DOM side.

---

**setElementAttribute(attribName, value)**

**removeElementAttribute(attribName)**

Helper method to set or remove an attribute on the instance's associated HTML element.

---

**setElementVisible(isVisible)**

Set whether the associated HTML element is visible or invisible. This sets the CSS style `display: none` when invisible.

---

**_getElementState()**

Override to return a JSON object representing the state of the DOM element, e.g. the text content. This is used by `CreateElement()` and `UpdateElementState()` to retrieve state to pass to the DOM side.

---

**_updateElementState()**

Send a message from the runtime to the DOM side with the element state (retrieved from `GetElementState()`). This results in a call to `UpdateState(elem, e)` on the DOM side. This is a convenient way to make sure any changes to the DOM element are applied.

---

**_getElementInDOMMode()**

When the runtime is in DOM mode, this returns the HTML element associated with this instance. When the runtime is in worker mode this method will throw an exception, as the DOM is not directly accessible from a Web Worker.

# ISDKDOMPLUGINBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkdompluginbase

The `ISDKDOMPluginBase` interface is used in the runtime as a base class for DOM plugins (that create a HTML element) in the addon SDK. It derives from ISDKPluginBase.

## ISDKDOMPluginBase APIs

**_addElementMessageHandler(handler, func)**

**_addElementMessageHandlers(arr)**

Add a message handler to receive messages posted by `PostToRuntimeElement()` in DOMElementHandler. `handler` must match the string passed to `PostToRuntimeElement()`. `func` accepts two arguments: the associated instance, and an optional object with extra details passed to `PostToRuntimeElement()`. Typically this function just forwards the handler to an instance method, e.g. `this._addElementMessageHandler("click", (inst, e) => inst._onClick(e));`. The `_addElementMessageHandlers` variant accepts an array of `[handler, callback]` which is convenient when adding multiple handlers.

# ISDKINSTANCEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkinstancebase

The `ISDKInstanceBase` interface is used as a runtime base class for instances in the addon SDK. It derives from IInstance.

## ISDKInstanceBase APIs

### _release()

Optional override for when an instance is released. Clean up any other resources that need releasing in this method. It must also call `super._release()` to release all engine resources.

### _getInitProperties()

Call this in the constructor to get an array of the instance's properties to create the instance with.

### _trigger(method)

### _triggerAsync(method)

Fire a trigger condition. The condition must be declared as a trigger in aces.json. Pass a full reference to the condition method, e.g. `this._trigger(C3.Plugins.MyPlugin.Cnds.MyTrigger)`. The async variant returns a Promise that resolves when the trigger has finished executing, which can be used to support Construct's debugger, as it may wait if it hits a breakpoint inside the trigger.

### _addDOMMessageHandler(handler, callback)

### _addDOMMessageHandlers(arr)

Add a callback to be run to handle a message posted from a DOM-side script. The handler is a string identifier. The callback receives the posted data as an argument. Note that if the caller in the DOM-side script originally used the `PostToRuntimeAsync` method, the callback may be an `async` function, and the return value is posted back to the DOM-side script. The `_addDOMMessageHandlers` variant accepts an array of `[handler, callback]` which is convenient when adding multiple handlers.

### _postToDOM(handler, data)

### _postToDOMAsync(handler, data)

Post a message to a DOM-side script. The handler is a string identifier. The `data` must be structurally clonable (since it is posted down a MessageChannel).
The async method returns a promise that resolves with the DOM-side callback's return value. The non-async method does not return a value and the DOM-side callback's return value is discarded (i.e. a "fire and forget" message).

---

**_postToDOMMaybeSync(handler, data)**

As with `_postToDOM()`, but when the runtime is in DOM mode, calls the DOM handler synchronously inside the call. When the runtime is in worker mode, this still posts a message which is handled later. Usually this method is not necessary, but it can be used to work around some user input restrictions in some browsers in DOM mode only.

---

**_setTicking(isTicking)**

**_setTicking2(isTicking)**

**_isTicking()**

**_isTicking2()**

Utility methods to start or stop the runtime calling the `_tick()` or `_tick2()` methods of your instance every tick, and also to check whether ticking is active. It is recommended to stop ticking whenever the tick method is no longer needed to reduce the performance overhead of ticking. Redundant calls to start or stop ticking are ignored. The first call always takes effect (i.e. calls do not stack - if you make 3 calls to start ticking then 1 call to stop ticking, ticking is stopped).

---

**_tick()**

Optional override that is called every tick just **before** events are run after `_setTicking(true)` has been called.

---

**_tick2()**

Optional override that is called every tick just **after** events are run after `_setTicking2(true)` has been called.

---

**_getDebuggerProperties()**

Override to return properties to display in the debugger. For more information see runtime scripts.

---

**_saveToJson()**

Optional override to return a JSON object that represents the state of the instance for savegames.

**_loadFromJson(o)**

Optional override accepting a JSON object returned by a prior call to `_saveToJson()` that represents the state of an instance to load, for savegames.

## Wrapper extension methods

These methods relate to the use of wrapper extensions. Refer to that manual section for more details; for completeness the relevant methods are also included here.

**_isWrapperExtensionAvailable()**

Returns a boolean indicating whether the corresponding wrapper extension was successfully loaded. If this returns false then no messages sent to the wrapper extension will be received, and async messages will return a promise that never resolves.

**_addWrapperExtensionMessageHandler(messageId, callback)**

**_addWrapperMessageHandlers(list)**

Add a callback to be run to handle a message posted from the corresponding wrapper extension. The callback receives the JSON data sent from the wrapper extension as an argument. The `_addWrapperMessageHandlers` variant accepts an array of `[messageId, callback]` which is convenient when adding multiple handlers.

**_sendWrapperExtensionMessage(messageId, params)**

**_sendWrapperExtensionMessageAsync(messageId, params)**

Send a message to the wrapper extension. The message ID is used to identify the kind of message. `params` is an optional array of parameters to provide. These may only be boolean, number or string type values. The async variant returns a Promise that resolves when the wrapper extension responds to the message. The promise resolves with the JSON data sent from the wrapper extension.

# ISDKOBJECTTYPEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkobjecttypebase

The `ISDKObjectTypeBase` is used in the runtime as a base class for object types in the addon SDK. It derives from IObjectClass.

## ISDKObjectTypeBase APIs

### _onCreate()

An optional override called when the runtime starts up and creates all object types before the project starts. This call may be delayed when using a loader layout.

### getImageInfo()

For plugins that use an image (by calling `this._info.SetHasImage(true)` in the editor plugin code), returns an IImageInfo representing the object's image.

### async _loadTextures(renderer)

Optional override called by the runtime when it wants your object type's textures to be created. Use the passed IRenderer to load textures. This method should return a promise that resolves when all textures have been loaded.

### _releaseTextures(renderer)

Optional override called by the runtime when it wants your object type's textures to be released. Use the passed IRenderer to release textures. This method is synchronous. Release any textures and drop the references to them.

### _onDynamicTextureLoadComplete()

Optional override called by the runtime when a dynamic texture load has completed. If an object is created at runtime that does not already have its textures loaded, the runtime will call `_loadTextures()` but continue running the project while the textures are loaded asynchronously. When they finish loading, the runtime then calls this override so you have an opportunity to update any references to the newly loaded texture.

### async _preloadTexturesWithInstances(renderer)

Optional override called just before starting a layout once all instances are created. This provides an opportunity to load textures depending on the details of the instances on the layout. Use the passed IRenderer to load textures. This may be used for purposes such as rasterizing SVG at an appropriate resolution before the layout starts.

# ISDKPLUGINBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkpluginbase

---

The `ISDKPluginBase` interface is used in the runtime as a base class for plugins created with the addon SDK. It derives from IPlugin.

## ISDKPluginBase APIs

The `ISDKPluginBase` interface does not currently have any of its own APIs.

# ISDKUTILS ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkutils

The `ISDKUtils` interface provides general APIs intended for use with the Addon SDK. It is normally accessed via `runtime.sdk`.

## ISDKUtils APIs

### addLoadPromise(promise)

Only valid while the project is still loading. Add a promise that the runtime will wait to resolve before starting the first layout. This is useful if you want to make sure your addon loads an asynchronous resource before the game starts.

> *Single-global plugins create their instance before loading, so this method can be used in single-global instance constructors. Otherwise it can only be used in the plugin or behavior constructor.*

### updateRender()

By default, Construct does not render a new frame unless something has changed. If something in your addon is changed which will affect the way it draws, call this method to ensure Construct draws a new frame to reflect the changes. Avoid calling this method unnecessarily - only call it when something has actually changed (be sure not to call it if e.g. setting something to the same value).

### sendWrapperExtensionMessage(wrapperComponentId, messageId, params)

### sendWrapperExtensionMessageAsync(wrapperComponentId, messageId, params)

These methods are equivalent to those in ISDKInstanceBase, but allow specifying an arbitrary wrapper component ID. In general it is recommended to use the instance-specific methods as cases where you need to message a different wrapper extension to your own are rare.

### getObjectClassBySid(sid)

Returns an IObjectClass with the given SID (Serialization ID), or null if none found. This can be used to look up the `IObjectClass` for plugin "object" properties, which pass a SID at runtime for the property value.

### isAutoSuspendEnabled

Set or get a boolean indicating whether the runtime automatically suspends when it detects the page or app going in to the background. This is enabled by default; by turning it off you can then use `setSuspended()` to control when the runtime suspends and resumes. Note however that even if automatic suspending is disabled, browsers may effectively suspend background pages anyway through other means, such as throttling timers and callbacks. Therefore disabling automatic suspending does not necessarily mean the runtime actually is able to continue running in the background. This API exists mainly to comply with the requirements of platforms that provide their own suspend and resume events that are required to be used.

---

### setSuspended(isSuspended)

Pass `true` to suspend the runtime, and `false` to resume it. When suspended, the runtime stops ticking and drawing anything, and remains inactive. Make sure that suspend and resume calls are paired one-to-one: for example do not suspend once but resume twice.

> *The current suspended state can be read via the property* `runtime.isSuspended`.

---

### constructVersionCode

A read-only number representing the Construct release the project has been previewed or exported with. A version code is the release number multiplied by 100, plus the patch number. For example r123.4 has the version code 12304. This can be useful for compatibility checks where feature detection (such as checking if methods exist) is not sufficient.

---

### createLoopingConditionContext(loopName)

This method is intended specifically for looping conditions (where `"isLooping": true` is specified in the condition definition). It allows "retriggering" the event, which executes all subsequent conditions, actions and sub-events in one call; a loop is implemented by repeatedly retriggering the event. The loop name is optional but allows passing the name to the system *loopindex* condition to retrieve the index in nested loops. It returns an ILoopingConditionContext. An example looping condition implementation is shown below.

```
// Sample looping condition method
TestLoop(count)
{
	const loopCtx = this.runtime.sdk.createLoopingConditionContext();

	for(let i = 0; i < count; ++i)
	{
		loopCtx.retrigger();

		if (loopCtx.isStopped)
			break;
```

```
        }

        loopCtx.release();
}
```

# ISDKWORLDINSTANCEBASE ADDON SDK INTERFACE

**View online:** https://www.construct.net/en/make-games/manuals/construct-3/scripting/scripting-reference/addon-sdk-interfaces/isdkworldinstancebase

The `ISDKWorldInstanceBase` interface is used as a runtime base class for world instances in the addon SDK. It derives from both ISDKInstanceBase and IWorldInstance.

## ISDKWorldInstanceBase APIs

### _draw(renderer)

An override for drawing plugins to draw their content using the passed IRenderer interface.

### _handleRendererContextLoss()

Utility method to enable calling of the `_onRendererContextLost()` and `_onRendererContextRestored()` methods.

### _onRendererContextLost()

### _onRendererContextRestored()

Optional overrides for handling renderer context loss. This is only necessary if your addon creates any renderer resources. These overrides are only called if your instance first calls `_handleRendererContextLoss()` (normally in the constructor to enable these callbacks for the lifetime of the instance). In a context loss event, all renderer resources have been released, so any references to them must be dropped. In a context restored event the resources may be recreated again if necessary. Alternatively they can be left released if a lazy-loading approach is used.