
Godot Engine Documentation

Release latest

Juan Linietsky, Ariel Manzur and the Godot community

Feb 03, 2018

General

Attention: This is the documentation for the unstable (master) branch. Looking for the documentation of the current **stable** branch? [Have a look here.](#)

Welcome to the official documentation of Godot Engine, the free and open source community-driven 2D and 3D game engine! If you are new to this documentation, we recommend that you read the *introduction page* to get an overview of what this documentation has to offer.

The table of contents below and in the sidebar should let you easily access the documentation for your topic of interest. You can also use the search function in the top left corner.

Note: Godot Engine is an open source project developed by a community of volunteers. It means that the documentation team can always use your feedback and help to improve the tutorials and class reference. If you do not manage to understand something, or cannot find what you are looking for in the docs, help us make the documentation better by letting us know!

Submit an issue or pull request on the [GitHub repository](#), or discuss with us on the #godotengine-devel IRC channel!

The main documentation for the site is organized into the following sections:

CHAPTER 1

About

1.1 Introduction

```
func _ready():
    $Label.text = "Hello world!"
```

Welcome to the official documentation of Godot Engine, the free and open source community-driven 2D and 3D game engine! Behind this mouthful, you will find a powerful but user-friendly tool that you can use to develop any kind of game, for any platform and with no usage restriction whatsoever.

This page aims at giving a broad presentation of the engine and of the contents of this documentation, so that you know where to start if you are a beginner or where to look if you need info on a specific feature.

1.1.1 About Godot Engine

A game engine is a complex tool, and it is therefore difficult to present Godot in a few words. Here's however our PR presentation, which you are free to reuse if you need a quick writeup about Godot Engine.

Godot Engine is a feature-packed, cross-platform game engine to create 2D and 3D games from a unified interface. It provides a comprehensive set of common tools, so that users can focus on making games without having to reinvent the wheel. Games can be exported in one click to a number of platforms, including the major desktop platforms (Linux, macOS, Windows) as well as mobile (Android, iOS) and web-based (HTML5) platforms.

Godot is completely free and open source under the very permissive MIT license. No strings attached, no royalties, nothing. Users' games are theirs, down to the last line of engine code. Godot's development is fully independent and community-driven, empowering users to help shape their engine to match their expectations. It is supported by the [Software Freedom Conservancy](#) not-for-profit.

For a more in-depth view of the engine, you are encouraged to read this documentation further, especially the *Step by step* tutorial.

1.1.2 About the documentation

This documentation is continuously written, corrected, edited and revamped by members of the Godot Engine community. It is edited via text files in the [reStructuredText](#) markup language and then compiled into a static website/offline document using the open source [Sphinx](#) and [ReadTheDocs](#) tools.

Note: You can contribute to Godot's documentation by opening issue tickets or sending patches via pull requests on its [GitHub source repository](#).

All the contents are under the permissive Creative Commons Attribution 3.0 ([CC-BY 3.0](#)) license, with attribution to “Juan Linietsky, Ariel Manzur and the Godot Engine community”.

1.1.3 Organisation of the documentation

This documentation is organised in five sections with an impressively unbalanced distribution of contents – but the way it is split up should be relatively intuitive:

- The *General* section contains this introduction as well as information about the engine, its history, its licensing, authors, etc. It also contains the *Frequently asked questions*.
- The *Getting started* section is the the main *raison d'être* of this documentation, as it contains all the necessary information on using the engine to make games. It starts with the *Step by step* tutorial which should be the entry point for all new users.
- The *Tutorials* section, on the other hand, can be read as needed, in any order. It contains many feature-specific tutorials and documentations.
- The *Development* section is intended for advanced users and contributors to the engine development, with information on compiling the engine, developing C++ modules or editor plugins.
- The *Community* gives information related to contributing to the engine development and the life of its community, e.g. how to report bugs, help with the documentation, etc. It's also the place for tutorials contributed by community members before they are properly reviewed and adapted to be integrated in the official list of the *Tutorials* section.
- Finally, the *Class reference* is the documentation of the Godot API, which is also available directly within the script editor. It is generated automatically from a file in the main repository, and the generated files of the documentation are therefore not meant to be modified. See *Contribute to the Class Reference* for details.

Have fun reading and making games with Godot Engine!

1.2 Frequently asked questions

1.2.1 What can I do with Godot? How much does it cost? What are the license terms?

Godot is Free/Libre Open Source Software available under the [OSI-approved MIT license](#).

This means it is free as in “free speech” as well as in “free beer”.

In short:

- There are no usage restrictions on Godot
- This means you can use it for any game or application, commercially or non-commercially, in any industry

- You can modify, (re)distribute and remix Godot to your heart's content

For more, see [here](#) or ask your lawyer of choice.

All the contents of the documentation are under the permissive Creative Commons Attribution 3.0 ([CC-BY 3.0](#)) license, with attribution to “Juan Linietsky, Ariel Manzur and the Godot Engine community”.

Logos and icons are generally under the same Creative Commons license. Note that some third-party libraries included with Godot’s source code may have different licenses.

For full details, look at the [COPYRIGHT.txt](#) as well as the [LICENSE.txt](#) and [LOGO_LICENSE.txt](#) files in the Godot repository.

Also see [the license page on the Godot website](#).

1.2.2 Which platforms are supported by Godot?

For the editor:

- Windows
- Mac OS X
- X11 (Linux, *BSD)

For exporting your games:

- Windows (and UWP)
- Mac OS X
- X11 (Linux, *BSD)
- Android
- iOS
- Web

Both 32 and 64 bit binaries are supported where it makes sense, with 64 being the default.

Some users also report building and using Godot successfully on ARM-based system with Linux, like the Raspberry Pi. There is also some unofficial thirdparty work being done on building for some consoles. None of this is included in the default build scripts or export templates, however.

For more on this, see the sections on *exporting* and *compiling Godot yourself*.

1.2.3 Which languages are supported in Godot?

The officially supported languages for Godot are GDScript, Visual Scripting, C# and C++. See the subcategories for each language in the [scripting](#) section.

Note that C# and Visual Scripting support is comparatively young and GDScript still has some advantages as outlined below.

Support for new languages can be added by third parties using the GDNative / NativeScript / PluginScript facilities. (See question about plugins below.)

Work is currently underway, for example, on unofficial bindings for Godot to [Python](#) and [Nim](#).

1.2.4 GDScript? Why use a custom scripting language instead of my language of choice?

The short answer is that we think the additional complexity both on your side (when first learning Godot and GDScript) as well as our side (maintenance) is worth the more integrated and seamless experience over attracting additional users with more familiar programming languages that result in a worse experience. We understand if you would rather use another language in Godot (see list of supported options above), but we strongly encourage you to try it and see the benefits for yourself.

GDScript is designed to integrate from the ground to the way Godot works, more than any other language, and is very simple and easy to learn. Takes at most a day or two to get comfortable and it's very easy to see the benefits once you do. Please do the effort to learn GDScript, you will not regret it.

Godot C++ API is also efficient and easy to use (the entire Godot editor is made with this API), and an excellent tool to optimize parts of a project, but trying to use it instead of GDScript for an entire game is, in most cases, a waste of time.

Yes, for more than a decade we tried in the past integrating several VMs (and even shipped games using them), such as Python, Squirrel and Lua (in fact we authored tolua++ in the past, one of the most popular C++ binders). None of them worked as well as GDScript does now.

More information about getting comfortable with GDScript or dynamically typed languages can be found in the *GDScript: An introduction to dynamic languages* tutorial.

For the more technically versed, proceed to the next item.

1.2.5 I don't believe you. What are the technical reasons for the item above?

The main reasons are:

1. No good thread support in most script VMs, and Godot uses threads (Lua, Python, Squirrel, JS, AS, etc.).
2. No good class extending support in most script VMs, and adapting to the way Godot works is highly inefficient (Lua, Python, JS).
3. Horrible interface for binding to C++, results in large amount of code, bugs, bottlenecks and general inefficiency (Lua, Python, Squirrel, JS, etc.)
4. No native vector types (vector3, matrix4, etc.), resulting in highly reduced performance when using custom types (Lua, Python, Squirrel, JS, AS, etc.).
5. Garbage collector results in stalls or unnecessarily large memory usage (Lua, Python, JS, AS, etc.).
6. Difficulty to integrate with the code editor for providing code completion, live editing, etc. (all of them). This is very well supported by GDScript.

GDScript was designed to solve the issues above, and performs very well in all the above scenarios. Please learn GDScript and enjoy a very smooth integration of scripting with the game engine (yes, it's a rare but very enjoyable situation when things just work). It's worth it, give it a try!

1.2.6 I want to extend Godot. What are my options for creating plugins?

For creating Godot Editor plugins look at *EditorPlugins* and tool scripts.

Additional languages could be added via PluginScript or the more low-level NativeScript.

If you want to add a certain native library, your best bet is GDNative and custom C++ modules.

Also see the official blog posts on these topics:

- A look at the GDNative architecture
- [GDNative is here!](#)

You can also take a look at the GDScript implementation, the Godot modules as well as the [unofficial Python support](#) for Godot.

1.2.7 Why is FBX not supported for import?

FBX SDK has a very [restrictive license](#), that is incompatible with the [open license](#) provided by Godot.

That said, Godot's Collada support is really good, please use the [OpenCollada](#) exporter for maximum compatibility if you are using Maya or 3DS Max. If you are using Blender, take a look at our own [Better Collada Exporter](#).

Also, glTF support was added in Godot 3.0.

FBX support could still be provided by third parties as a plugin. (See Plugins question above.)

1.2.8 Will [Insert closed SDK such as PhysX, GameWorks, etc.] be supported in Godot?

No, the aim of Godot is to create a complete open source engine licensed under MIT, so you have complete control over every single piece of it. Open versions of functionality or features from such SDKs may be eventually added though.

That said, because it is open source, and modular, nothing prevents you or anyone else interested into adding those libraries as a module and ship your game using them, as either open or closed source. Everything is allowed.

To see how support for your SDK of choice could still be provided, look at the Plugins question above.

1.2.9 How should assets be created to handle multiple resolutions and aspect ratios?

This question pops up often and it's probably thanks to the misunderstanding created by Apple when they originally doubled the resolution of their devices. It made people think that having the same assets in different resolutions was a good idea, so many continued towards that path. That originally worked to a point and only for Apple devices, but then several Android and Apple devices with different resolutions and aspect ratios were created, with a very wide range of sizes and DPIs.

The most common and proper way to achieve this is to, instead, use a single base resolution for the game and only handle different screen aspects. This is mostly needed for 2D, as in 3D it's just a matter of Camera XFov or YFov.

1. Choose a single base resolution for your game. Even if there are devices that go up to 2K and devices that go down to 400p, regular hardware scaling in your device will take care of this at little or no performance cost. Most common choices are either near 1080p (1920x1080) or 720p (1280x720). Keep in mind the higher the resolution, the larger your assets, the more memory they will take and the longer the time it will take for loading.
2. Use the stretch options in Godot, 2D stretching with keeping aspect works best. Check the *Multiple resolutions* tutorial on how to achieve this.
3. Determine a minimum resolution and then decide if you want your game to stretch vertically or horizontally for different aspect ratios, or whether there is a minimum one and you want black bars to appear instead. This is also explained in the previous step.
4. For user interfaces, use the *anchoring* to determine where controls should stay and move. If UIs are more complex, consider learning about Containers.

And that's it! Your game should work in multiple resolutions.

If there really is a desire to make your game also work on ancient devices with tiny screens (less than 300 pixels in width), you can use the export option to shrink images, and set that build to be used for certain screen sizes in the App Store or Google Play.

1.2.10 I have a great idea that will make Godot better. What do you think?

Your idea will most certainly be ignored. Examples of stuff that is ignored by the developers:

- Let's do this because it will make Godot better
- Let's do this in Godot because another game engine does it
- Let's remove this because I think it's not needed
- Let's remove clutter and bloat and make Godot look nicer
- Let's add an alternative workflow for people who prefer it

Godot developers are always willing to talk to you and listen to your feedback very openly, to an extent rarely seen in open source projects, but they will care mostly about real issues you have while using Godot, not ideas solely based on personal belief. Developers are interested in (for example):

- Your experience using the software and the problems you have (we care about this much more than ideas on how to improve it).
- The features you would like to see implemented because you need them for your project.
- The concepts that were difficult to understand in order to learn the software.
- The parts of your workflow you would like to see optimized.
- Parts where you missed clear tutorials or where the documentation wasn't up to par.

Once one of the above points is stated, we can work together on a solution and this is where your ideas and suggestions are most valuable and welcome, they need to be in context of a real issue.

As such, please don't feel that your ideas for Godot are unwelcome. Instead, try to reformulate them as a problem first, so developers and the community have a base ground to discuss first.

Examples of how NOT to state problems generally and vaguely are:

- Certain feature is ugly
- Certain workflow is slow
- Certain feature needs optimization
- Certain aspect of the UI looks cluttered

Associating something with an adjective will not get you much attention and developers will most likely not understand you. Instead, try to reformulate your problem as a story such as:

- I try to move objects around but always end up picking the wrong one
- I tried to make a game like Battlefield but I'm not managing to understand how to get lighting to look the same.
- I always forget which script I was editing, and it takes me too many steps to go back to it.

This will allow you to convey what you are thinking much better and set a common ground for discussion. Please try your best to state your problems as stories to the developers and the community, before discussing any idea. Be specific and concrete.

Bonus points for bringing screenshots, concrete numbers, test cases or example projects (if applicable).

1.2.11 How can I support Godot development or contribute?

See *Ways to contribute*.

1.2.12 Who is working on Godot? How can I contact you?

See the corresponding page on the Godot website.

CHAPTER 2

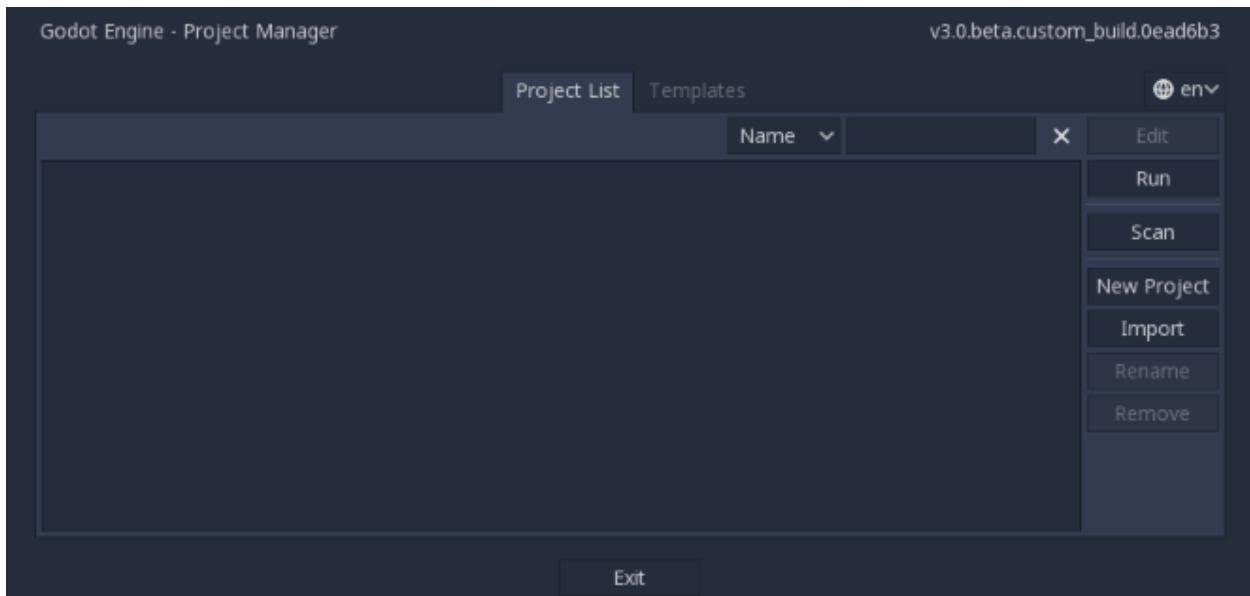
Step by step

2.1 Introduction to Godot's editor

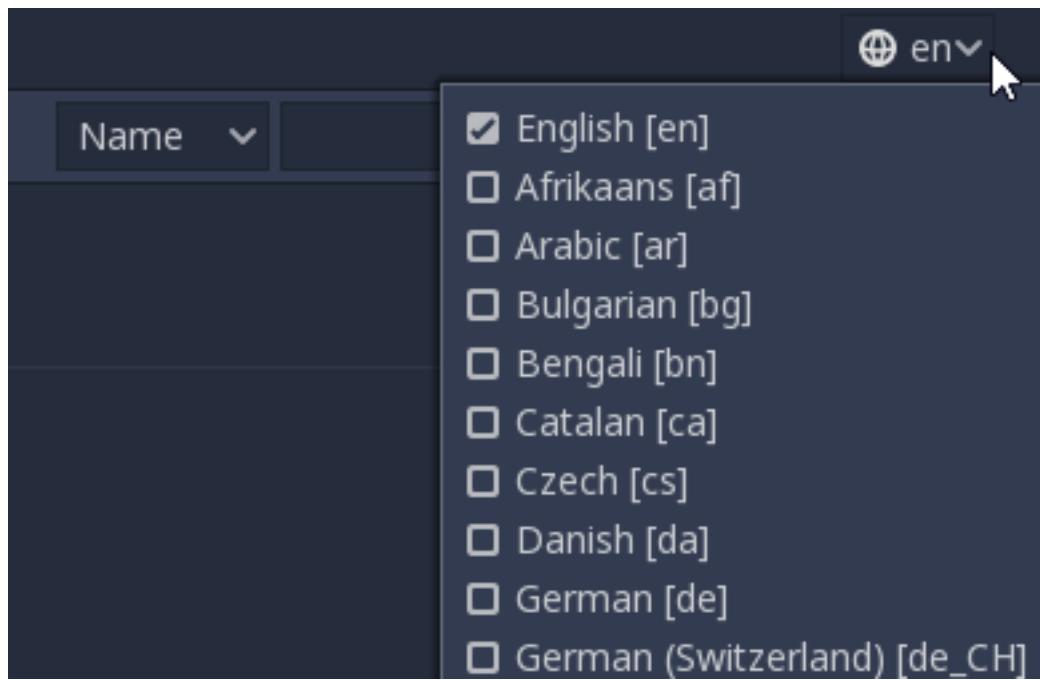
This tutorial will run you through Godot's interface. We're going to look at the **Project Manager**, **docks**, **workspaces** and everything you need to know to get started with the engine.

2.1.1 Project manager

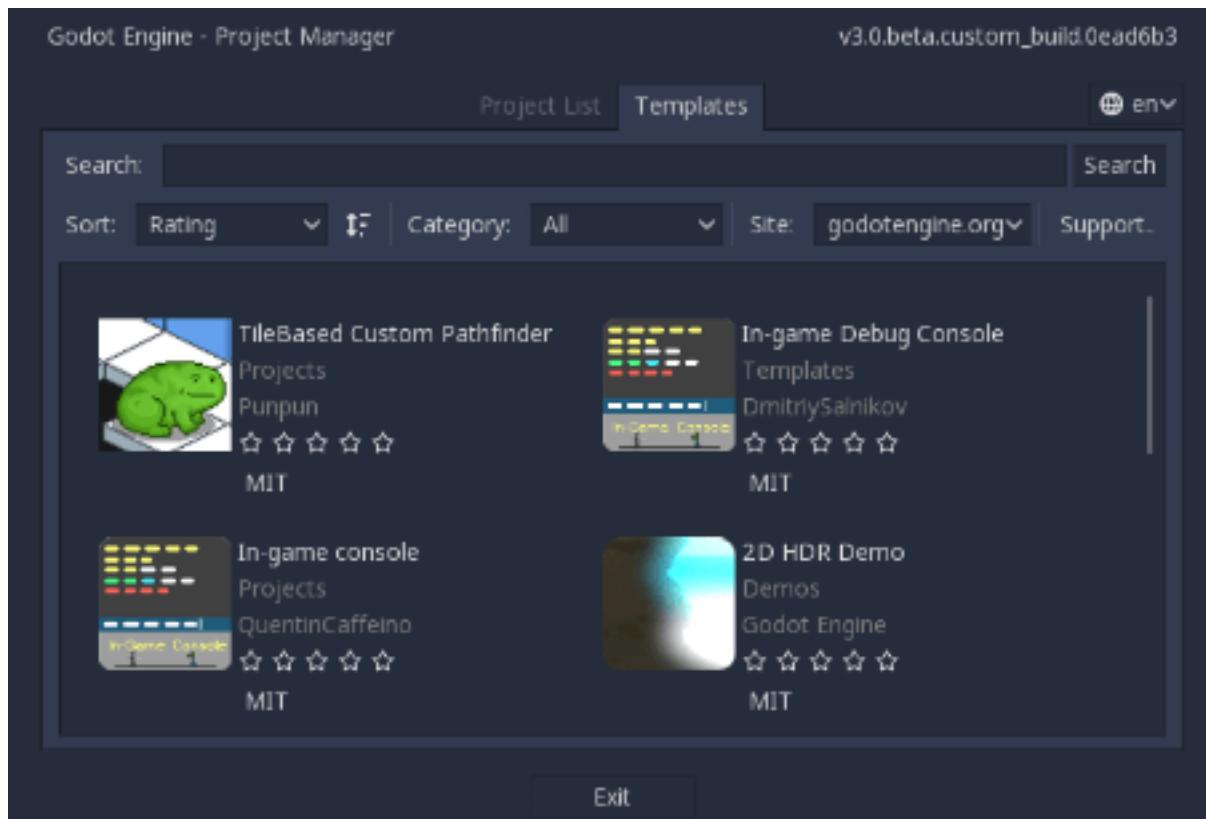
When you launch Godot, the first window you'll see is the Project Manager. It lets you create, remove, import or play game projects.



In the top-right corner you'll find a drop-down menu to change the editor's language.

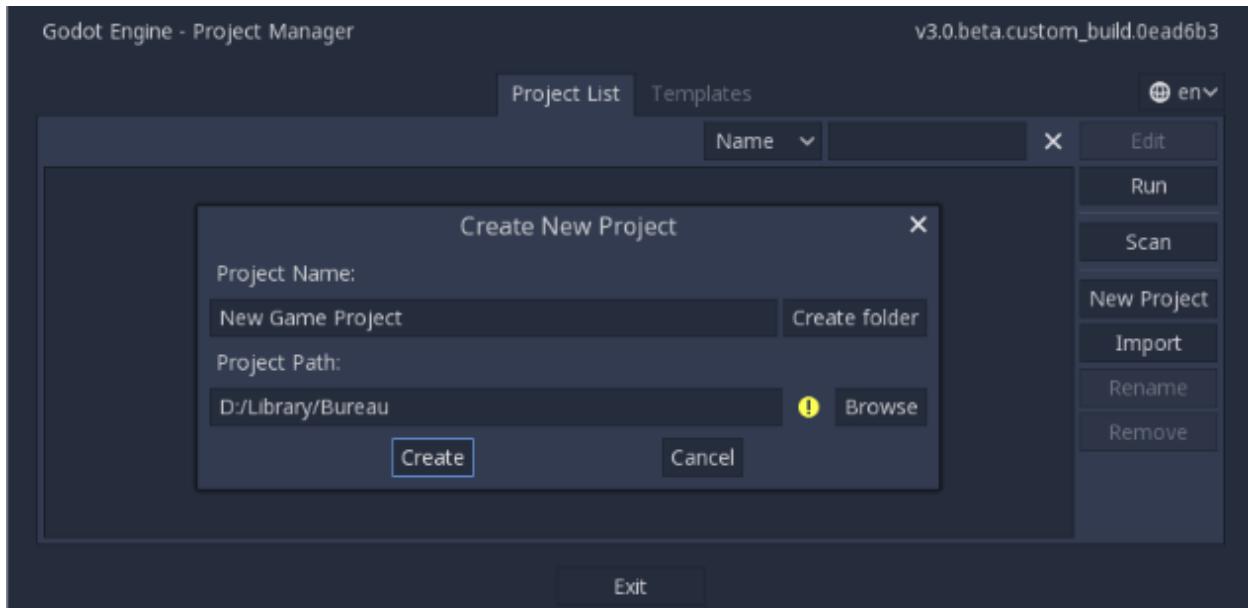


From the **Templates** tab you can download open source project templates and demos to help you get started faster.

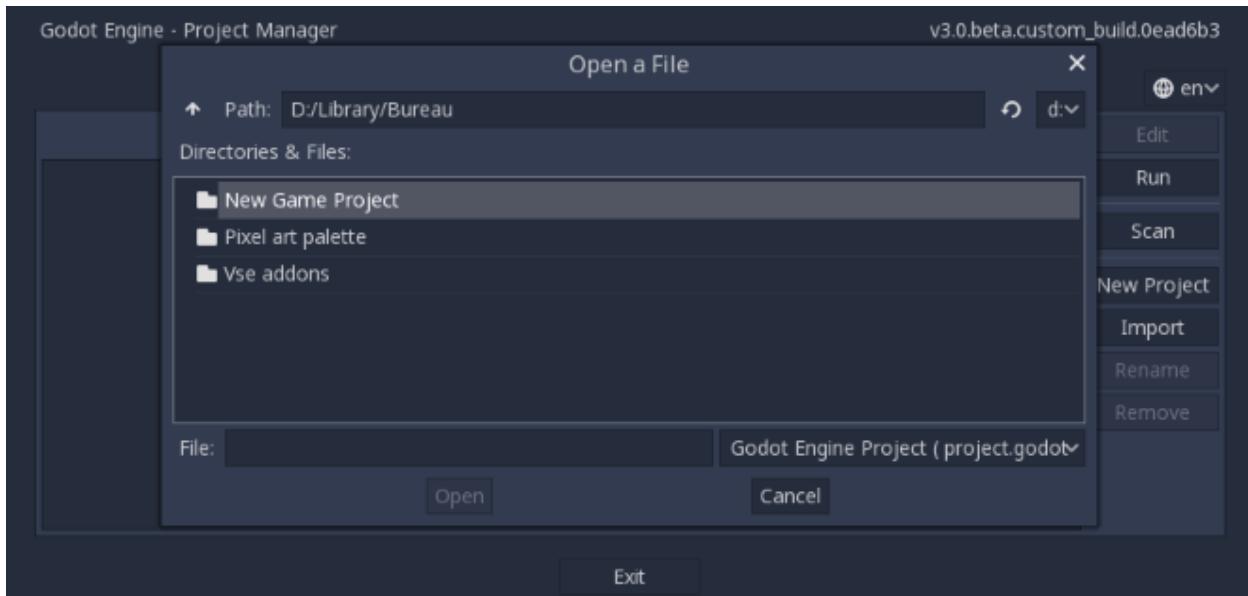


Create or import a project

To create a new project, click the New Project button on the right. Give it a name and choose an empty folder on your computer to save it.

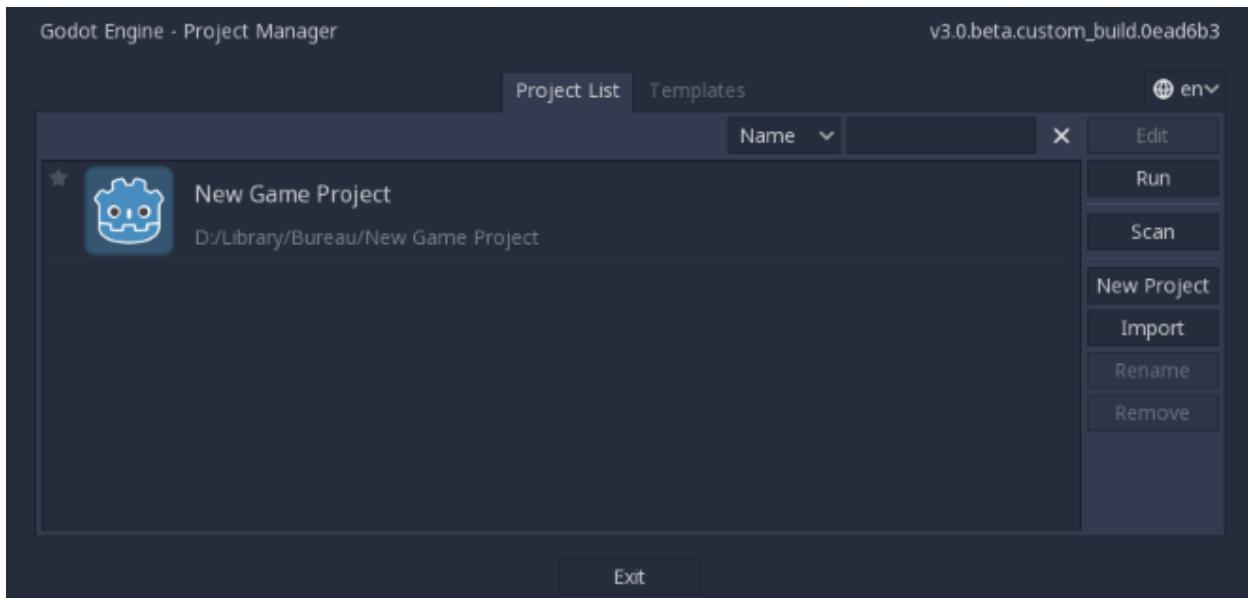


Click the Browse button to open Godot's file browser and pick a location or type the folder's path in the Project Path field.

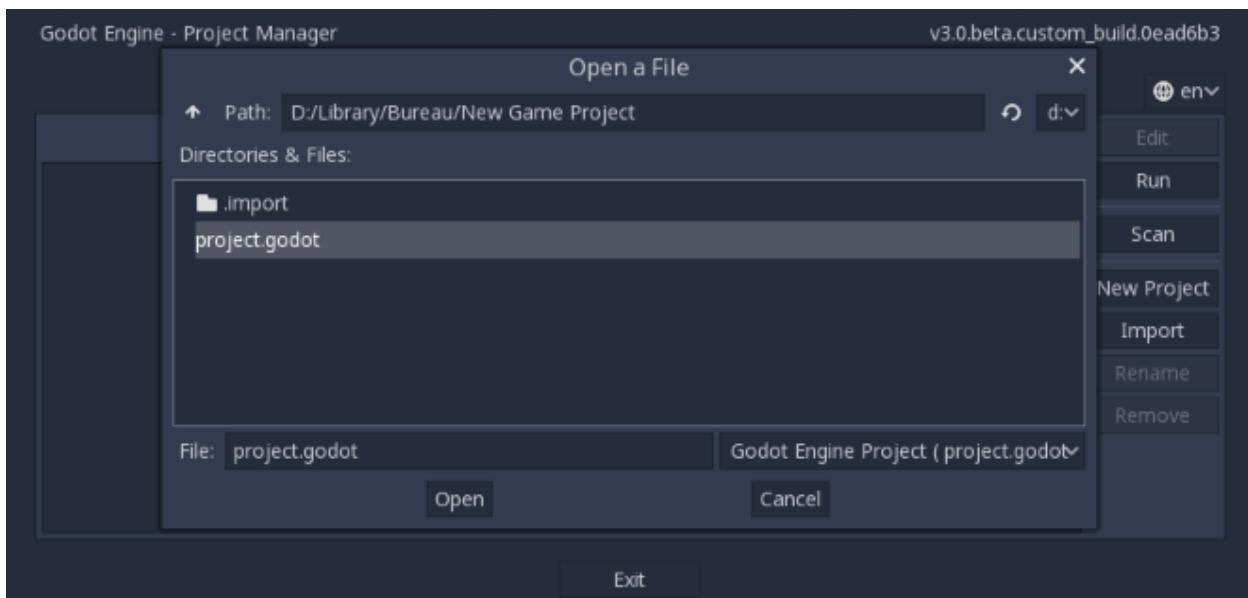


When you see the green tick on the right, it means the engine detects an empty folder and you may click Create. Godot will create the project for you and open it in the editor.

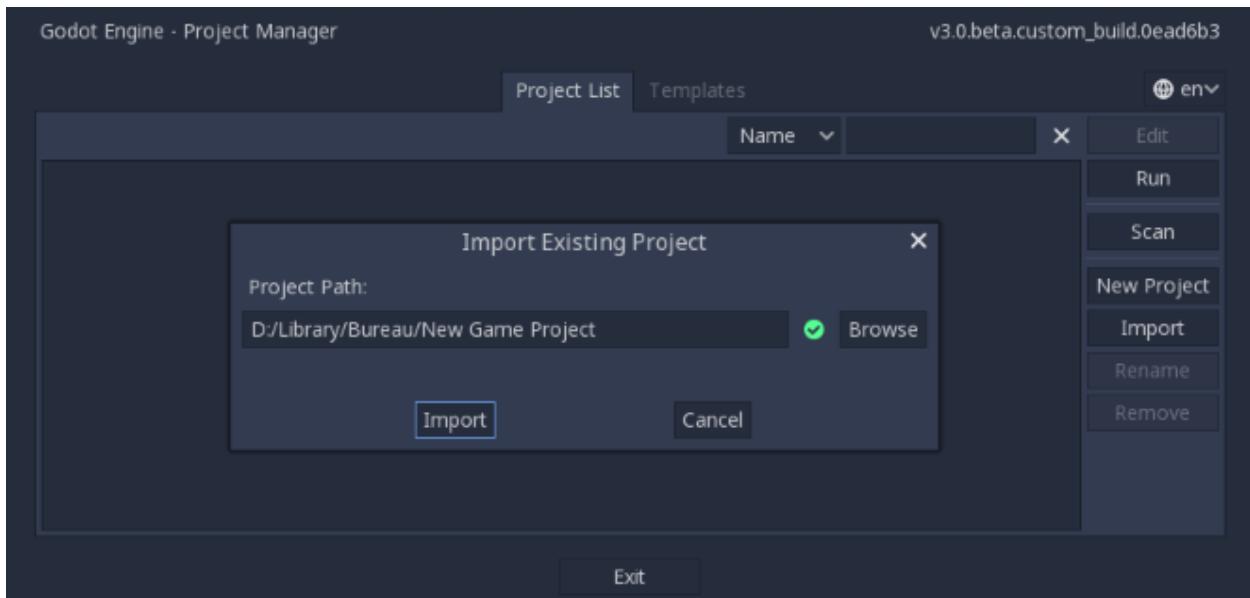
The next time you'll open Godot, you'll see your new project in the list. Double click on it to open it in the editor.



You can import existing projects in a similar way, using the Import button. Locate the folder that contains the project or the `project.godot` file to import and edit it.

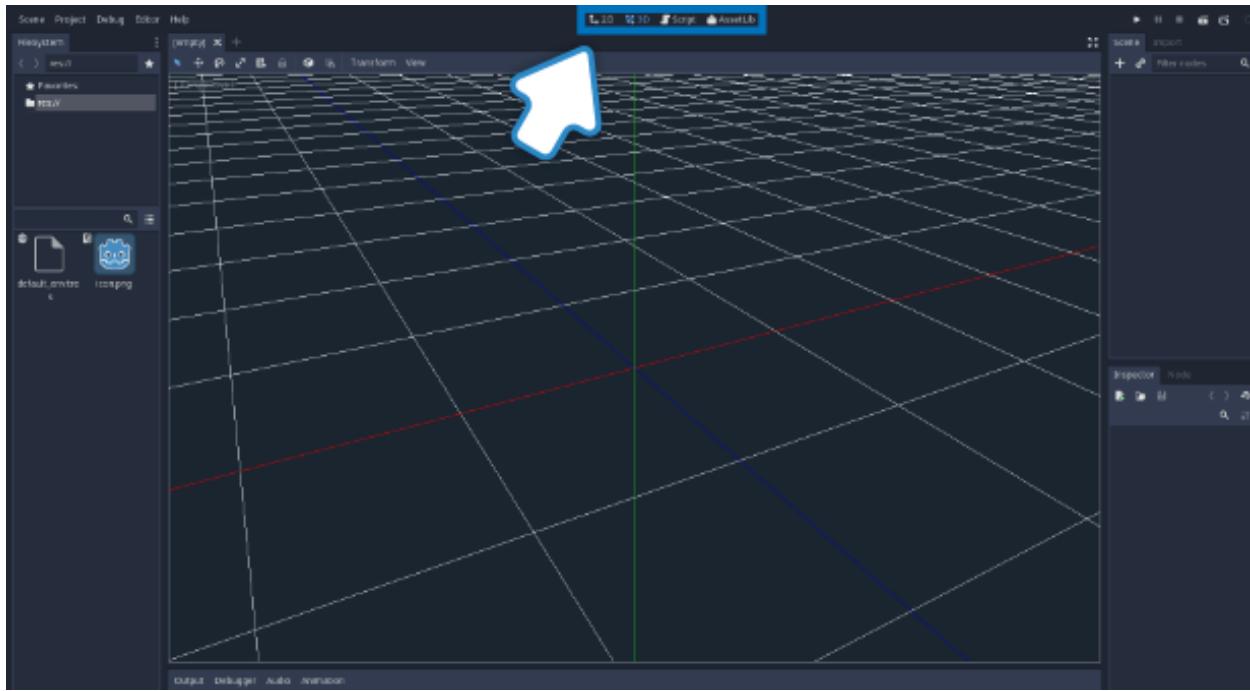


When the folder path is correct you'll see a green checkmark.

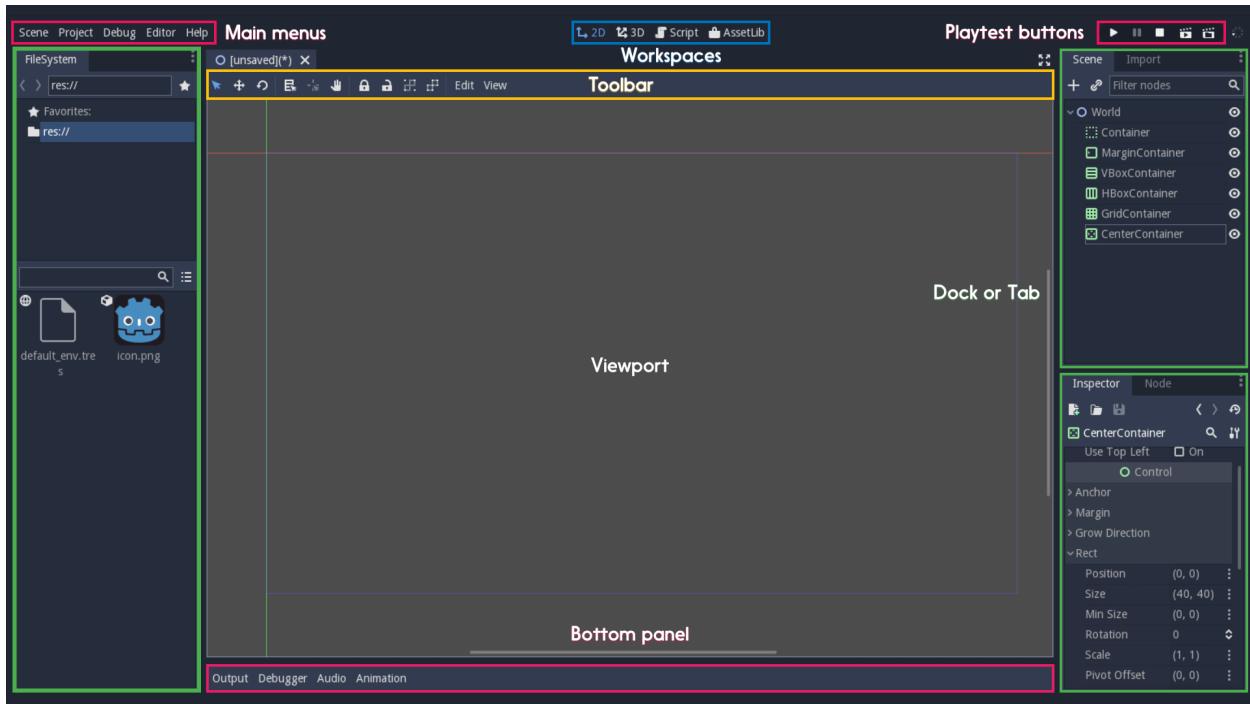


2.1.2 Your first look at Godot's editor

Welcome to Godot! With your project open you should see the editor's interface with the 3d viewport active. You can change the current workspace at the top of the interface. Click on 2d to switch to the 2d workspace.

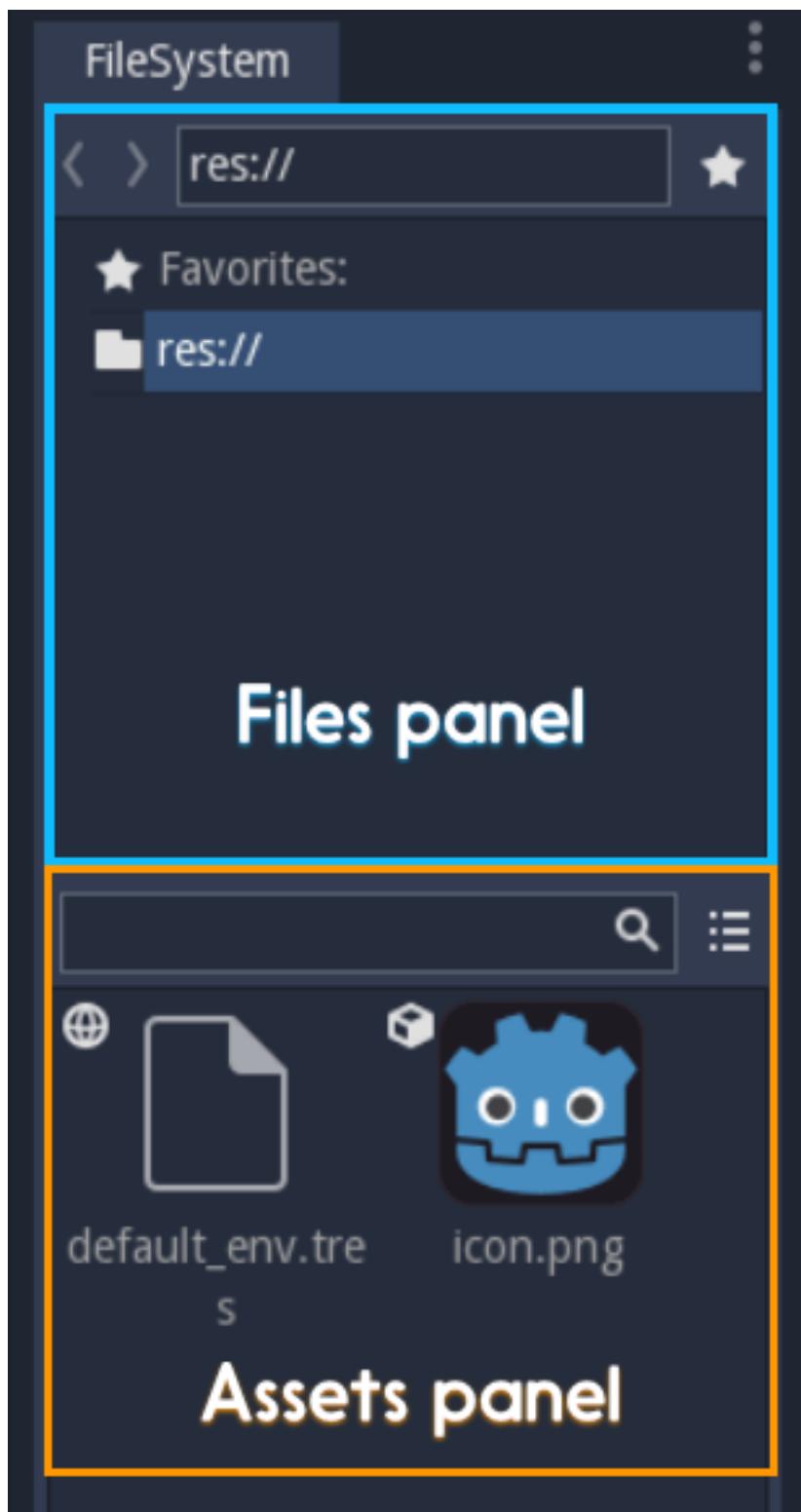


Now you should see this interface, with empty docks on the right side.

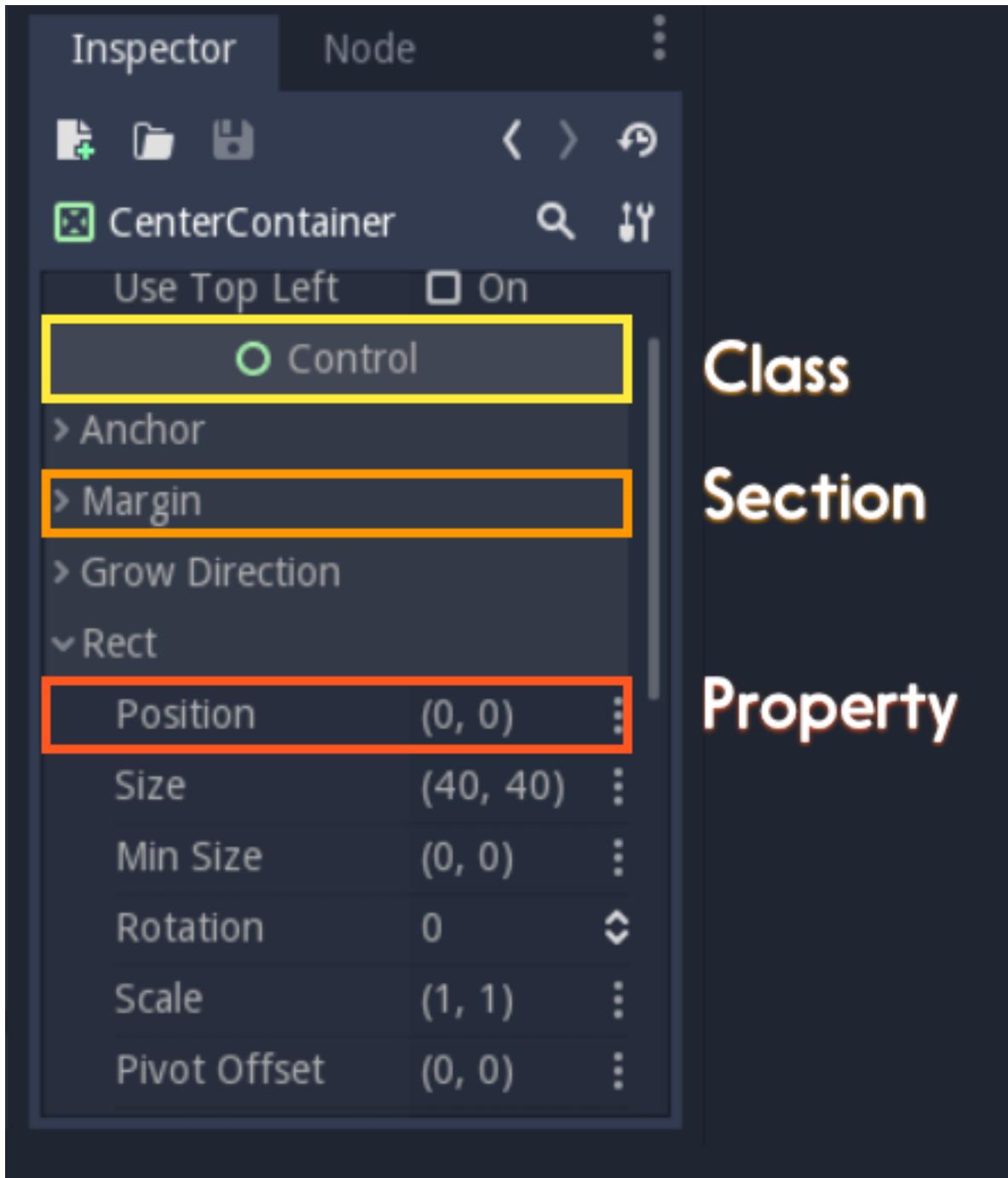


At the top, from left to right, you can see the **main menus**, the **workspaces**, and the **playtest buttons**.

On the left side you have the **FileSystem dock**, where you'll manage your project files and assets.



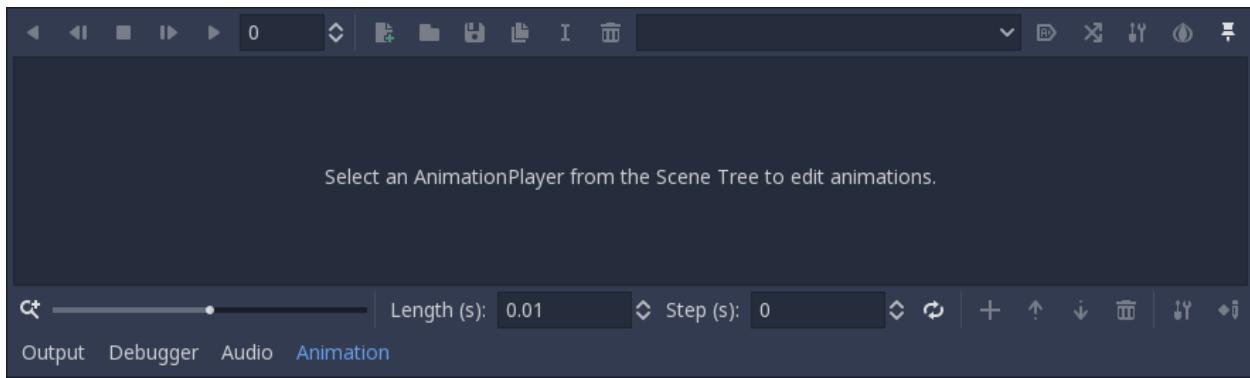
On the right side you'll find the **Scene dock** that lists the active scene's content and the **Inspector** in the bottom right corner.



In the center you have the **Toolbar** at the top, where you'll find tools to move, scale or lock your scene's objects. It changes as you jump to different workspaces.



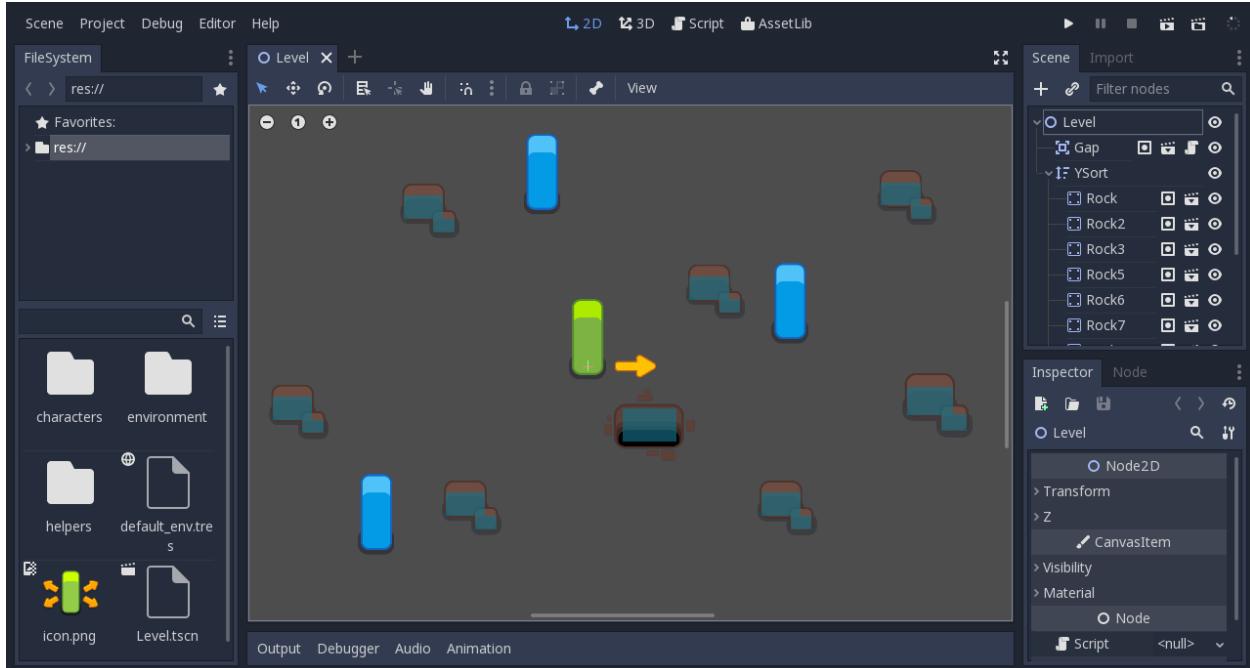
The **Bottom Panel** is the host for the debug console, the animation editor, the audio mixer... They are wide and can take precious space. That's why they're folded by default.



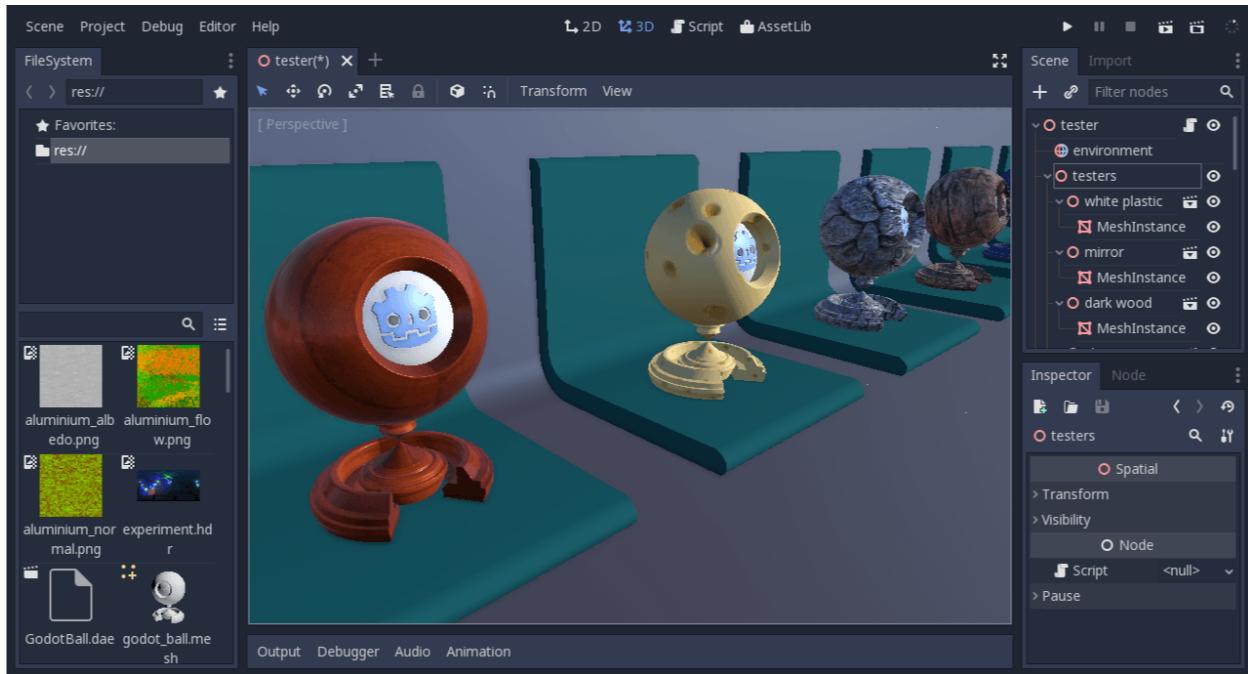
2.1.3 The workspaces

You can see four workspace buttons at the top: 2D, 3D, Script and AssetLib.

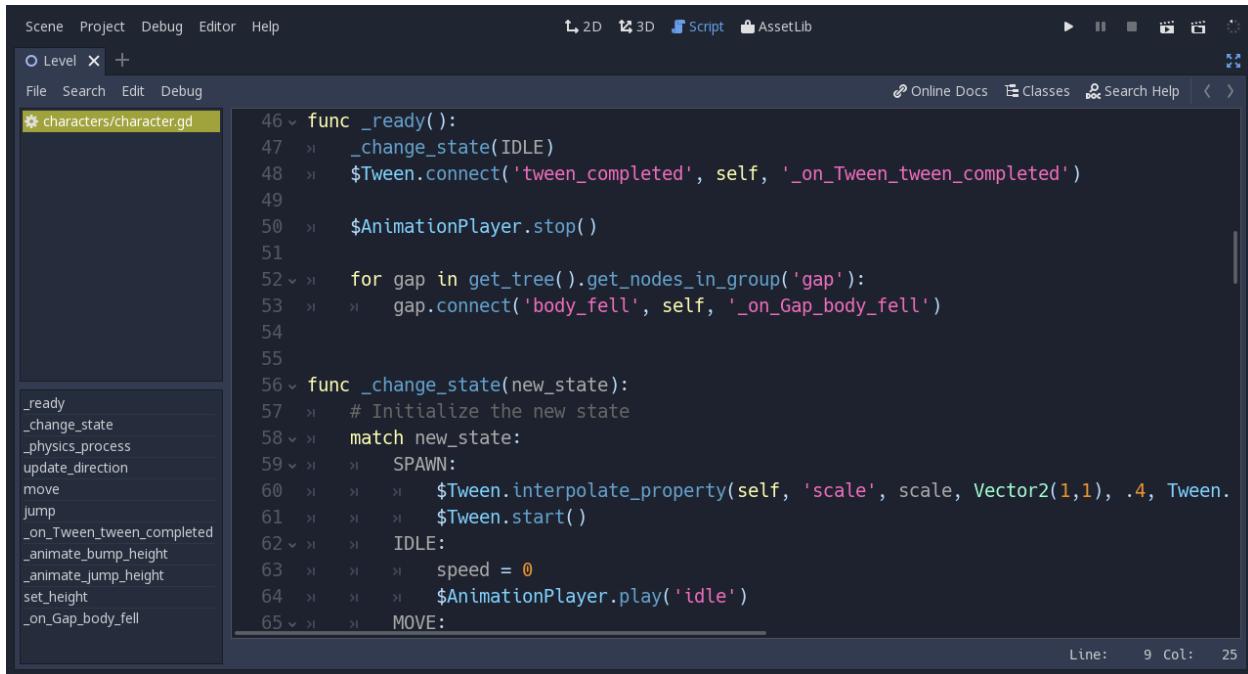
You'll use the **2D workspace** for all types of games. Press F1 to access it.



In the **3D workspace**, you can work with meshes, lights, and design levels for 3D games. Press F2 to access it.



The **Script** workspace is a complete code editor with a debugger, rich auto-completion, and built-in code reference. Press F3 to access it, and F4 to search the reference.



Finally the **AssetLib** is a library of Free add-ons, scripts and assets to use in your projects.

2.1.4 Modify the interface

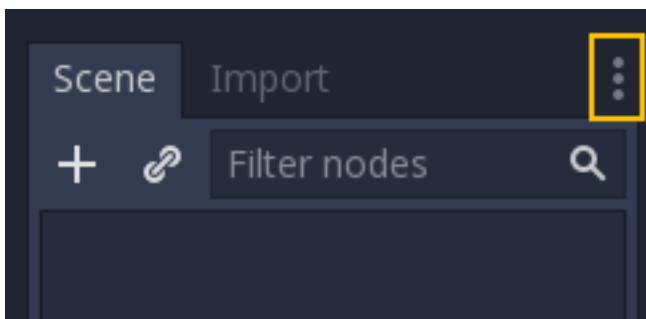
Godot's interface lives in a single window. You cannot split it across multiple screens although you can work with an external code editor like Atom or Visual Studio for instance.

Move and resize docks

Click and drag on the edge of any dock or panel to resize it horizontally or vertically.



Click the three-dotted icon at the top of any dock to change its location.



Go to the `Editor` menu and `Editor Settings` to fine-tune the look and feel of the editor.

2.2 Scenes and nodes

2.2.1 Introduction



Imagine for a second that you are not a game developer anymore. Instead, you're a chef! Change your hipster outfit for a toque and a double breasted jacket. Now, instead of making games, you create new and delicious recipes for your guests.

So, how does a chef create a recipe? Recipes are divided into two sections: the first is the ingredients and the second is the instructions to prepare it. This way, anyone can follow the recipe and savor your magnificent creation.

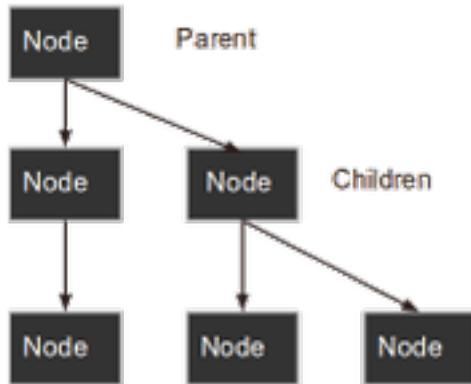
Making games in Godot feels pretty much the same way. Using the engine feels like being in a kitchen. In this kitchen, *nodes* are like a refrigerator full of fresh ingredients with which to cook.

There are many types of nodes. Some show images, others play sound, other nodes display 3D models, etc. There are dozens of them.

2.2.2 Nodes

But let's start with the basics. Nodes are fundamental building blocks for creating a game. As mentioned above, a node can perform a variety of specialized functions. However, any given node always has the following attributes:

- It has a name.
- It has editable properties.
- It can receive a callback to process every frame.
- It can be extended (to have more functions).
- It can be added to other nodes as children.

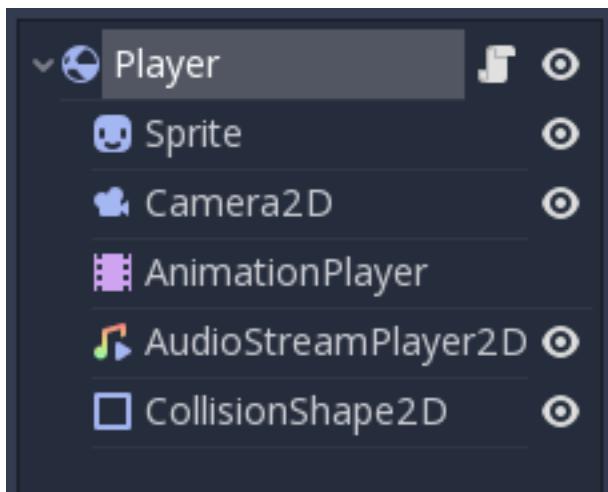


The last one is very important. Nodes can have other nodes as children. When arranged in this way, the nodes become a **tree**.

In Godot, the ability to arrange nodes in this way creates a powerful tool for organizing projects. Since different nodes have different functions, combining them allows for the creation of more complex functions.

Don't worry if this doesn't click yet. We will continue to explore this over the next few sections. The most important fact to remember for now is that nodes exist and can be arranged this way.

2.2.3 Scenes



Now that the concept of nodes has been defined, the next logical step is to explain what a Scene is.

A scene is composed of a group of nodes organized hierarchically (in tree fashion). Furthermore, a scene:

- always has only one root node.
- can be saved to disk and loaded back.
- can be *instanced* (more on that later).

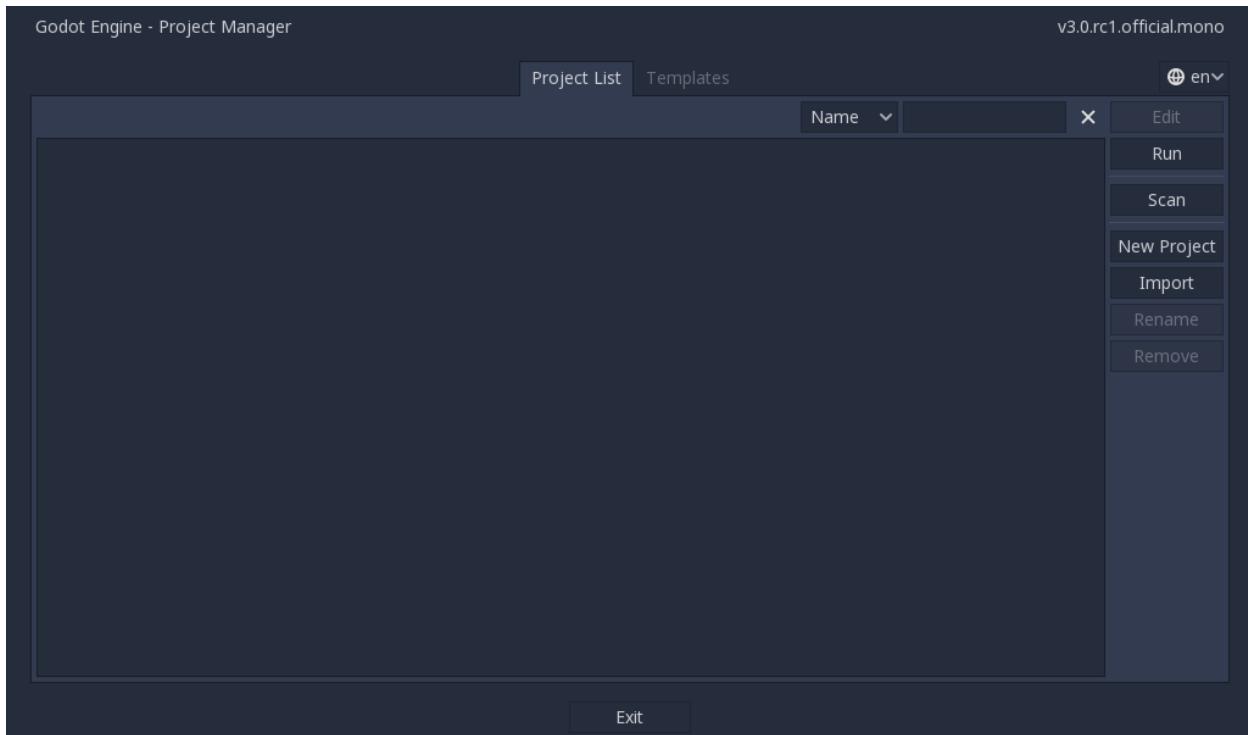
Running a game means running a scene. A project can contain several scenes, but for the game to start, one of them must be selected as the main scene.

Basically, the Godot editor is a **scene editor**. It has plenty of tools for editing 2D and 3D scenes as well as user interfaces, but the editor is based on the concept of editing a scene and the nodes that compose it.

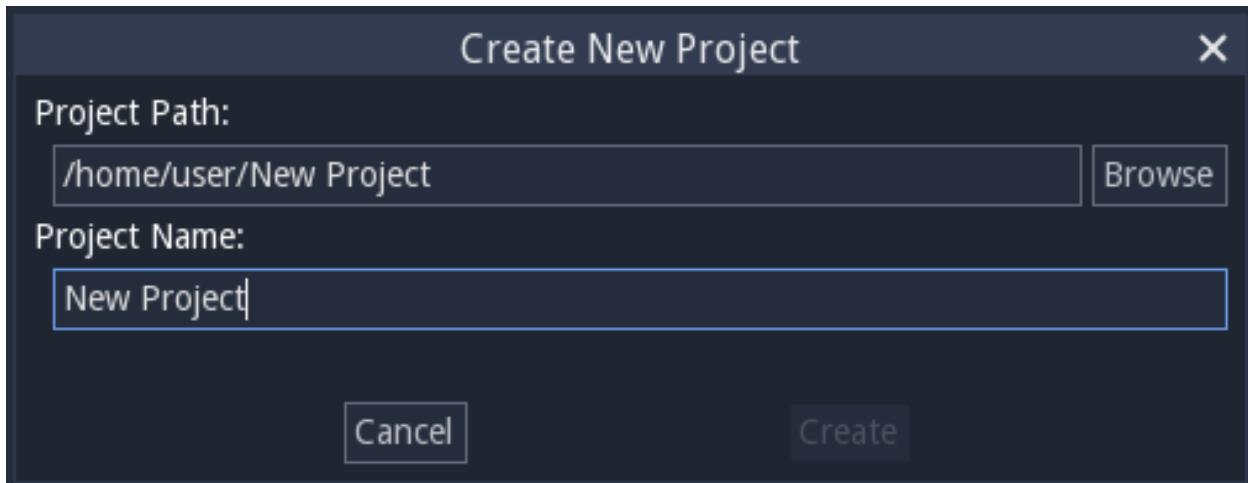
2.2.4 Creating a new project

Let's make these abstract concepts more concrete with an example. Following a long tradition in tutorials, we'll start with a "Hello World" project. This will introduce us to using the editor.

If you run the godot executable outside of a project, the Project Manager appears. This helps developers manage their projects.

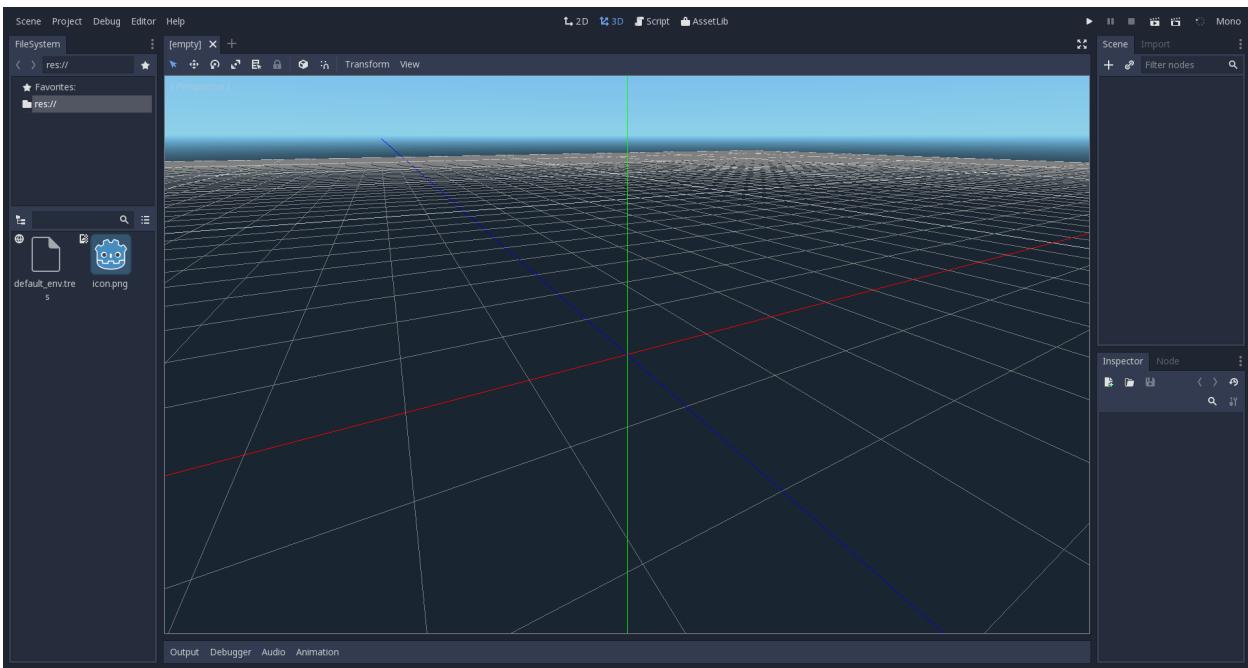


To create a new project, click the "New Project" option. Choose and create a path for the project and specify the project name "New Project":



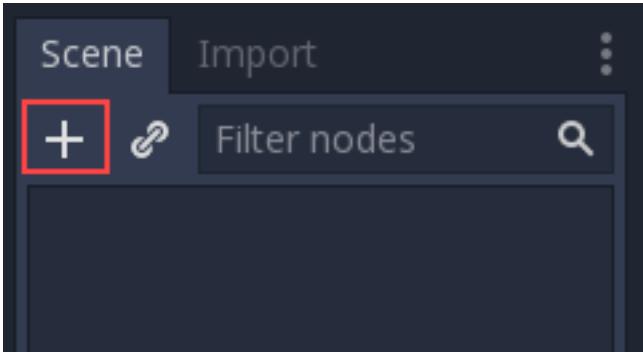
2.2.5 Editor

Once you've created the "New Project", then open it. This will open the Godot editor:

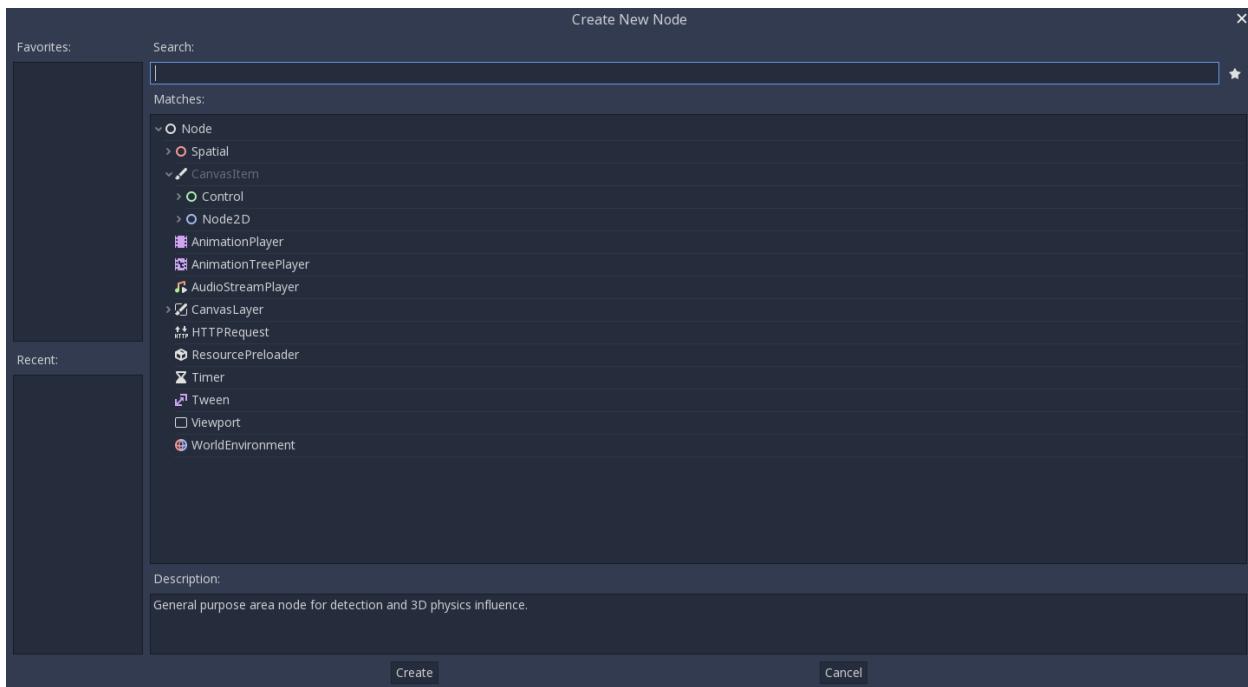


As mentioned before, making games in Godot feels like being in a kitchen, so let's open the refrigerator and add some fresh nodes to the project. We'll begin with a "Hello World!" message that we'll put on the screen.

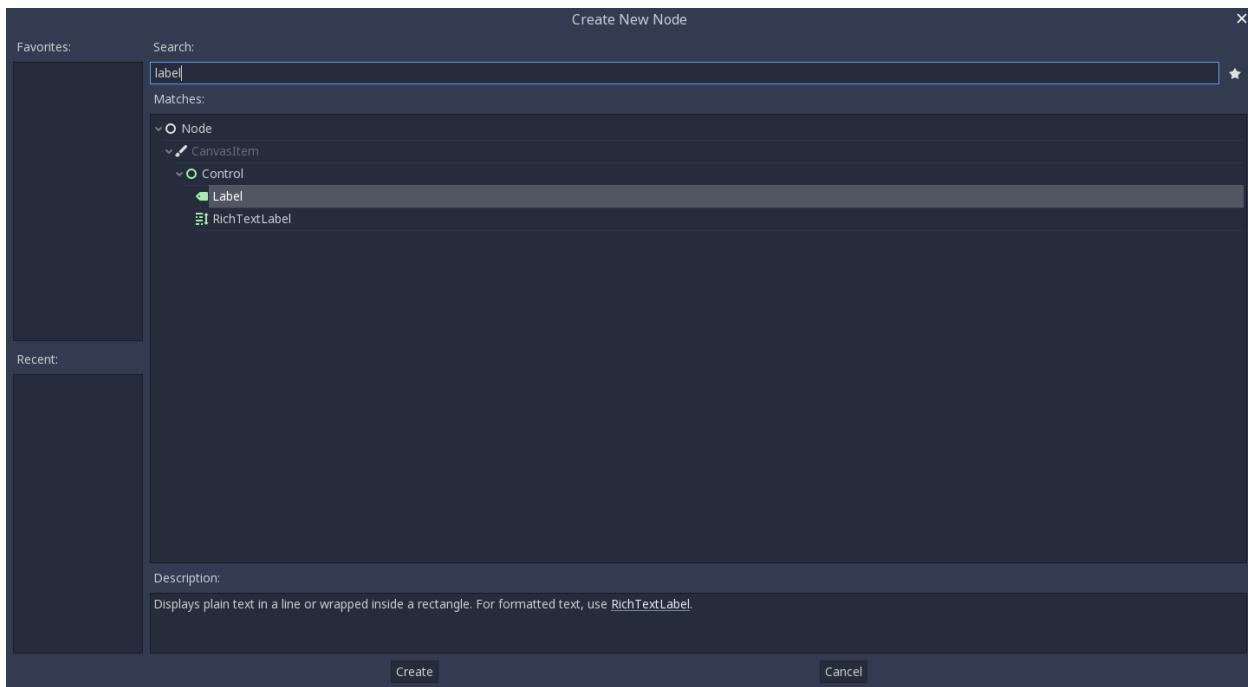
To do this, press the "New Node" button (which looks like a plus symbol):



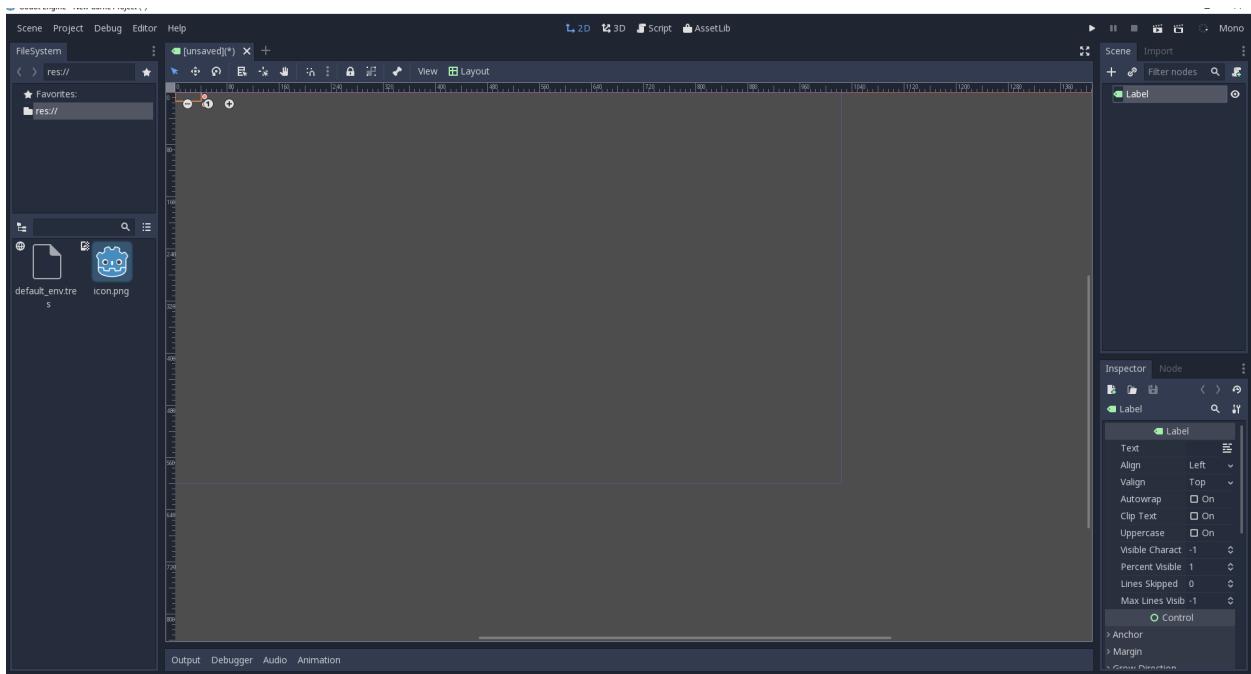
This will open the Create Node dialog, showing the long list of nodes that can be created:



From there, select the “Label” node first. Searching for it is probably the quickest way:



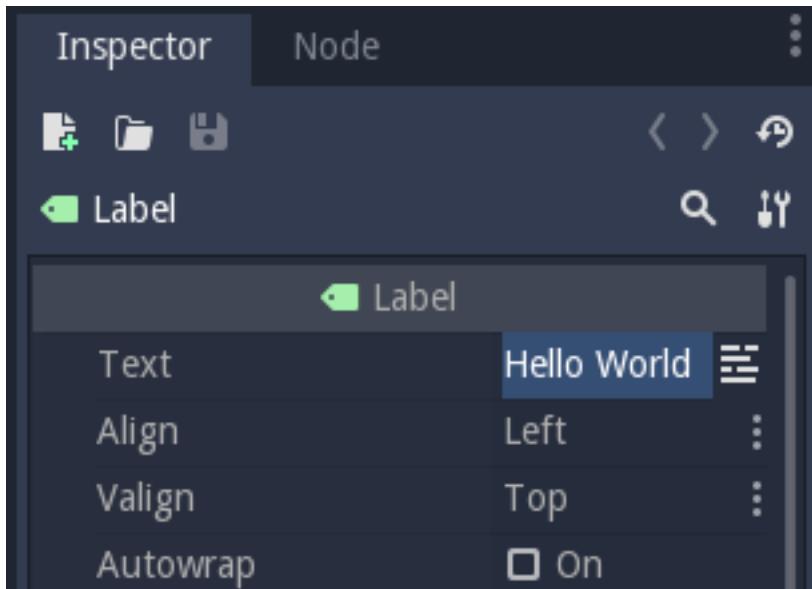
And finally, create the Label! A lot happens when Create is pressed:



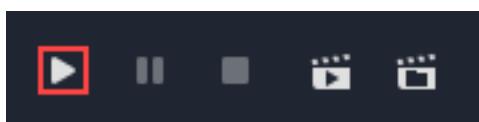
First of all, the scene changes to the 2D editor (because Label is a 2D Node type), and the Label appears, selected, at the top left corner of the viewport.

The node appears in the scene tree editor (box in the top right corner), and the label properties appear in the Inspector (box in the bottom right corner).

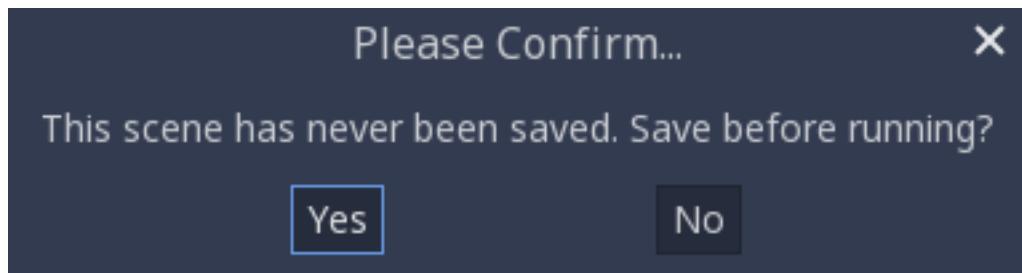
The next step will be to change the “Text” Property of the label. Let’s change it to “Hello, World!”:



Ok, everything’s ready to run the scene! Press the PLAY SCENE Button on the top bar (or hit F6):



Aaaand... Oops.

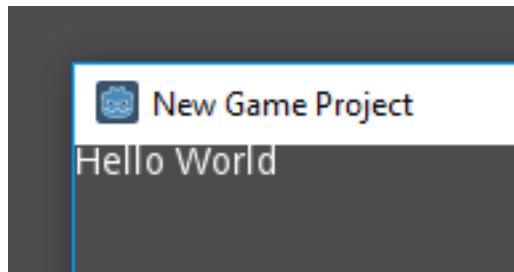


Scenes need to be saved to be run, so save the scene to something like hello.tscn in Scene -> Save:



And here's when something funny happens. The file dialog is a special file dialog, and only allows you to save inside the project. The project root is "res://" which means "resource path". This means that files can only be saved inside the project. For the future, when doing file operations in Godot, remember that "res://" is the resource path, and no matter the platform or install location, it is the way to locate where resource files are from inside the game.

After saving the scene and pressing run scene again, the "Hello, World!" demo should finally execute:



Success!

2.2.6 Configuring the project

Ok, it's time to configure the project. Right now, the only way to run something is to execute the current scene. Projects, however, may have several scenes, so one of them must be set as the main scene. This is the scene that will be loaded any time the project is run.

These settings are all stored in a project.godot file, which is a plaintext file in win.ini format (for easy editing). There are dozens of settings that you can change in this file to alter how a project executes. To simplify this process, Godot provides a project settings dialog, which acts as a sort of frontend to editing a project.godot file.

To access that dialog, select Project -> Project Settings. Try it now.

Once the window opens, let's select a main scene. Locate the *Application/Run/Main Scene* property and click on it to select 'hello.tscn'.



Now, with this change, when you press the regular Play button (or F5), this scene will run, no matter which scene is actively being edited.

The project settings dialog provides a lot of options that can be saved to a project.godot file and shows their default values. If you change a value, a tick is marked to the left of its name. This means that the property will be saved to the project.godot file and remembered.

As a side note, it is also possible to add custom configuration options and read them in at run-time using the *ProjectSettings* singleton.

2.2.7 To be continued...

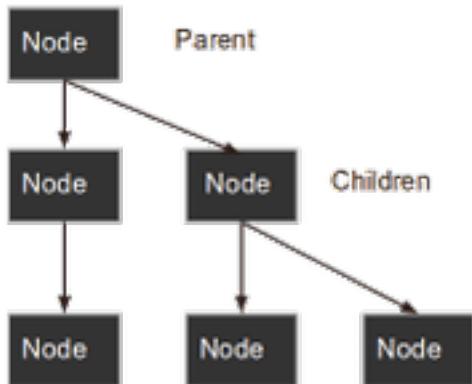
This tutorial talked about “scenes and nodes”, but so far there has been only *one* scene and *one* node! Don’t worry, the next tutorial will expand on that...

2.3 Instancing

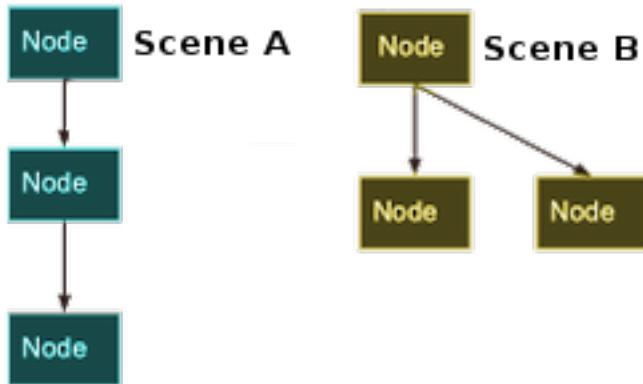
2.3.1 Introduction

Creating a single scene and adding nodes into it might work for small projects, but as a project grows in size and complexity, the number of nodes can quickly become unmanageable. To address this, Godot allows a project to be separated into any number scenes. This provides you with a powerful tool to organize the different components of your game.

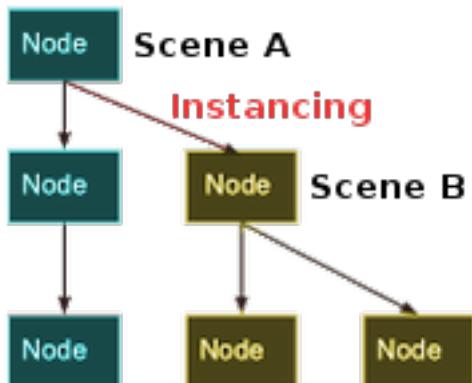
In *Scenes and nodes* you learned that a scene is a collection of nodes organized in a tree structure, with a single node as the tree root.



You can create as many scenes as you like and save them to disk. Scenes saved in this manner are called “Packed Scenes” and have a `.tscn` filename extension.



Once a scene has been saved, it can be instanced into another scene just as if it were any other node.

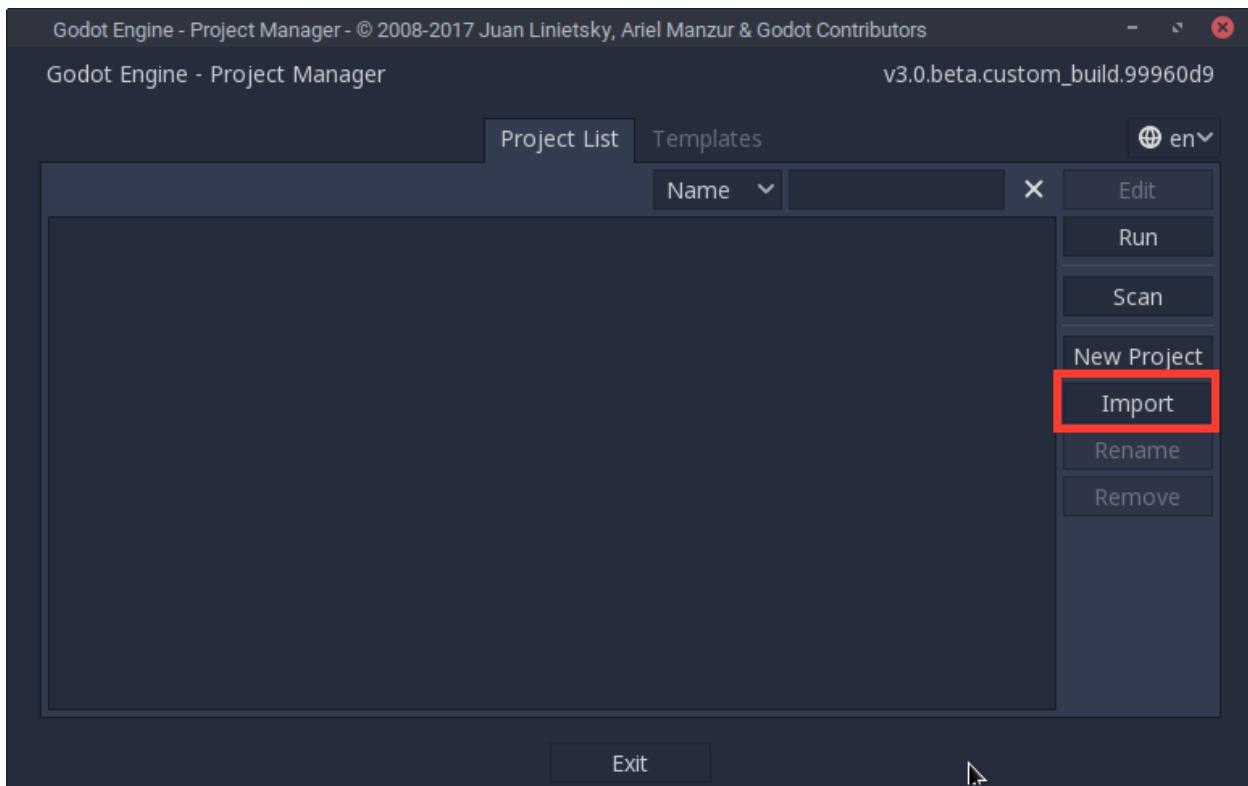


In the above picture, Scene B was added to Scene A as an instance.

2.3.2 Instancing By Example

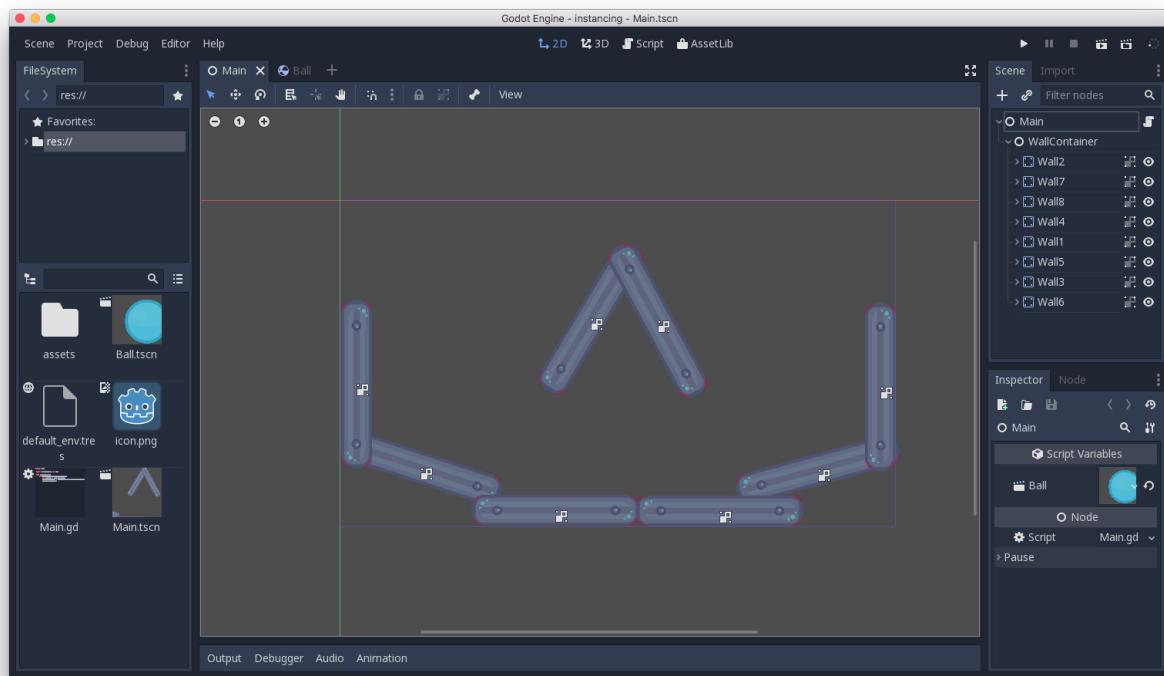
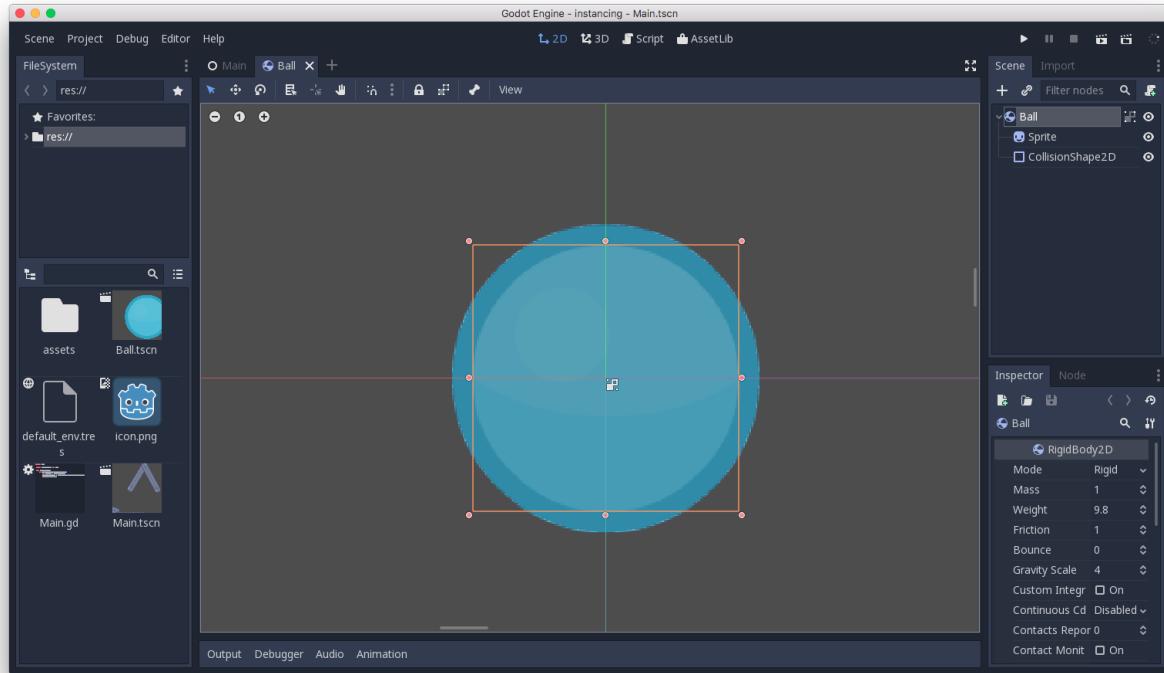
To learn how instancing works, let's start by downloading a sample project: `instancing.zip`.

Unzip this project anywhere you like. Then open Godot and add this project to the project manager using the 'Import' button:

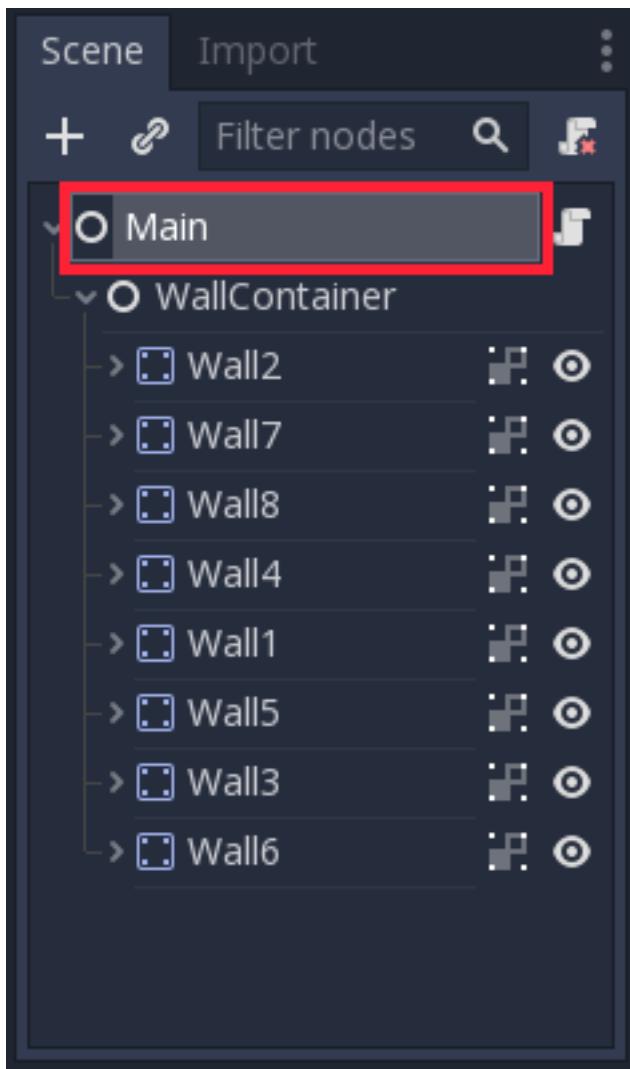


Browse to the folder you extracted and open the “`project.godot`” file you can find inside it. After doing this, the new project will appear on the list of projects. Edit the project by pressing the ‘Edit’ button.

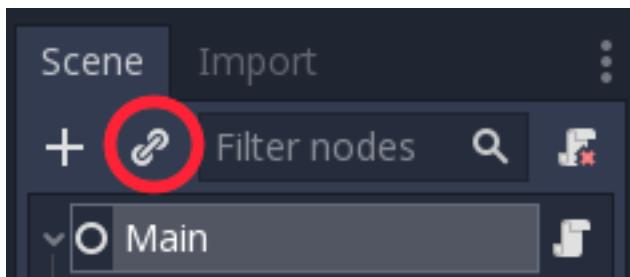
This project contains two scenes: “`Ball.tscn`” and “`Main.tscn`”. The ball scene uses a *RigidBody2D* to provide physics behavior while the main scene has a set of obstacles for the ball to collide with (using *StaticBody2D*).



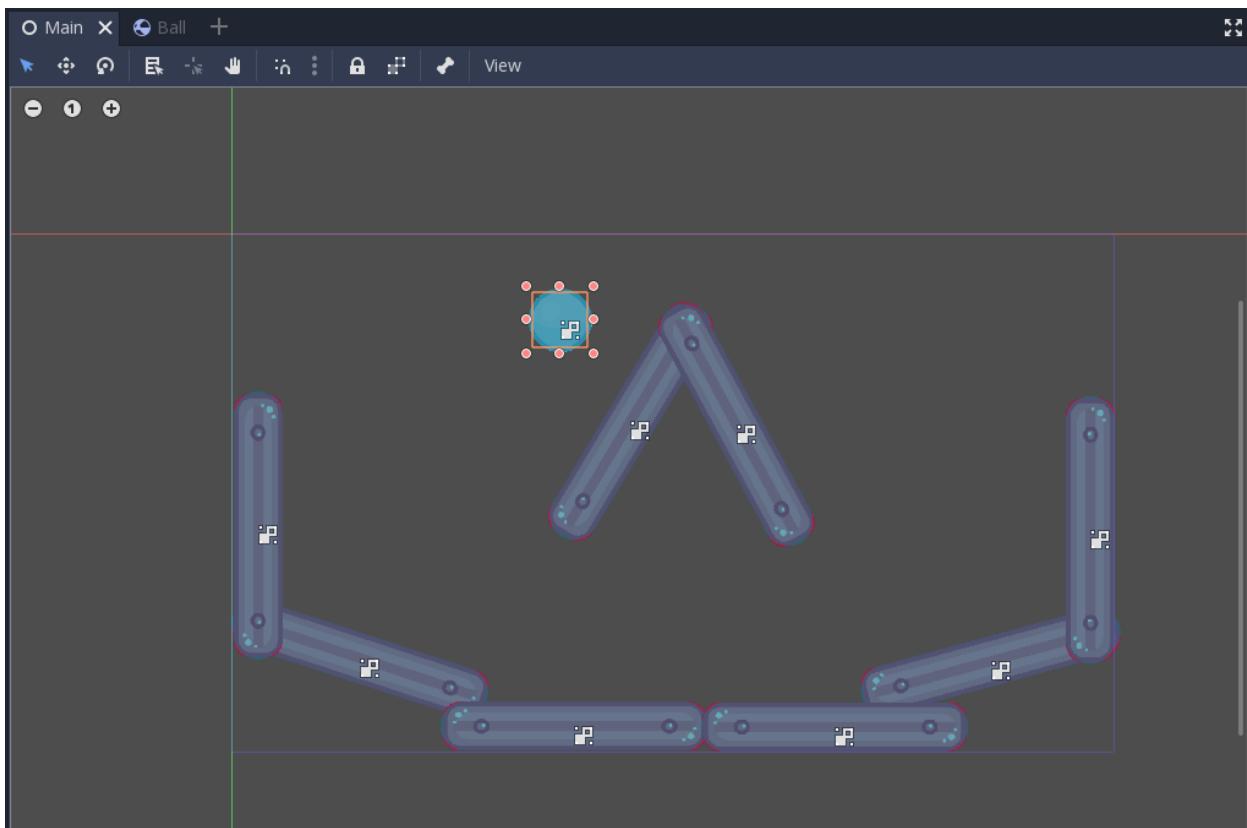
Open the `Main` scene, and then select the root node:



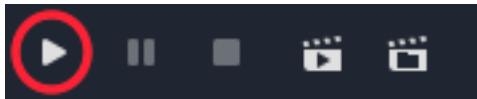
We want to add an instance of the Ball scene as a child of Main. Click the “link”-shaped button (its hover-text says “Instance a scene file as a Node.”) and select the Ball.tscn file.



The ball will be placed at the top-left corner of the screen area (this is $(0, 0)$ in screen coordinates). Click and drag the ball somewhere near the top-center of the scene:

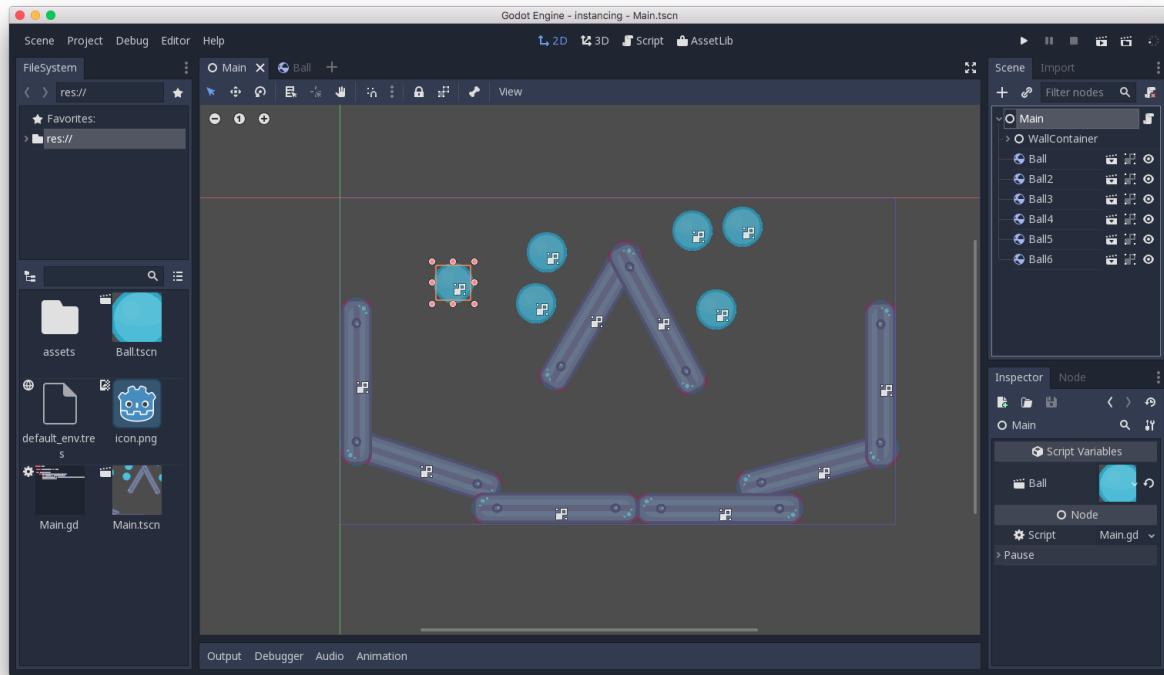


Press “Play” and watch the ball fall to the bottom of the screen:



2.3.3 Multiple Instances

You can add as many instances as you like to a scene, either by using the “Instance” button again, or by clicking on the ball instance and pressing “Duplicate” (Ctrl-D):



Run the scene again and all of the balls will fall.

2.3.4 Editing instances

Open the `Ball` scene and change the `Bounce` property in the Inspector to `1`. Press “Play” and notice that all of the instanced balls are now much more bouncy. Because the instanced balls are based on the saved scene, changes to that scene will affect all instances.

You can also adjust individual instances. Set the bounce value back to `0.5` and then in the `Main` scene, select one of the instanced balls. Set its `Bounce` to `1` and press “Play”.



Notice that a grey “revert” button appears next to the adjusted property. When this button is present, it means you modified a property in the instanced scene to override its value in the saved scene. Even if that property is modified in the original scene, the custom value will remain. Pressing the revert button will restore the property to the value in the saved scene.

2.3.5 Conclusion

Instancing can be very useful when you want to create many copies of the same object.

2.4 Instancing (continued)

2.4.1 Recap

Instancing has many handy uses. At a glance, with instancing you have:

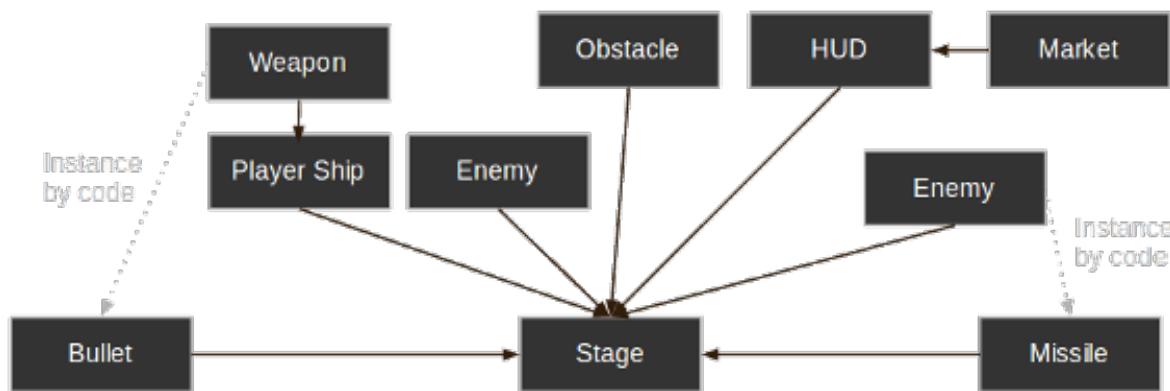
- The ability to subdivide scenes and make them easier to manage.
- A more flexible alternative to prefabs (and much more powerful given that instances can be nested).
- A way to organize and embed complex game flows or even UIs (in Godot, UI Elements are nodes, too).

2.4.2 Design language

But the greatest strength that comes with instancing scenes is that it works as an excellent design language. This is pretty much what distinguishes Godot from all the other engines out there. Godot was designed from the ground up around this concept.

When making games with Godot, the recommended approach is to dismiss most common design patterns, such as MVC or Entity-Relationship diagrams, and instead think about your scenes in a more natural way. Start by imagining the visible elements in your game, the ones that can be named not just by a programmer, but by anyone.

For example, here's how a simple shooter game could be imagined:

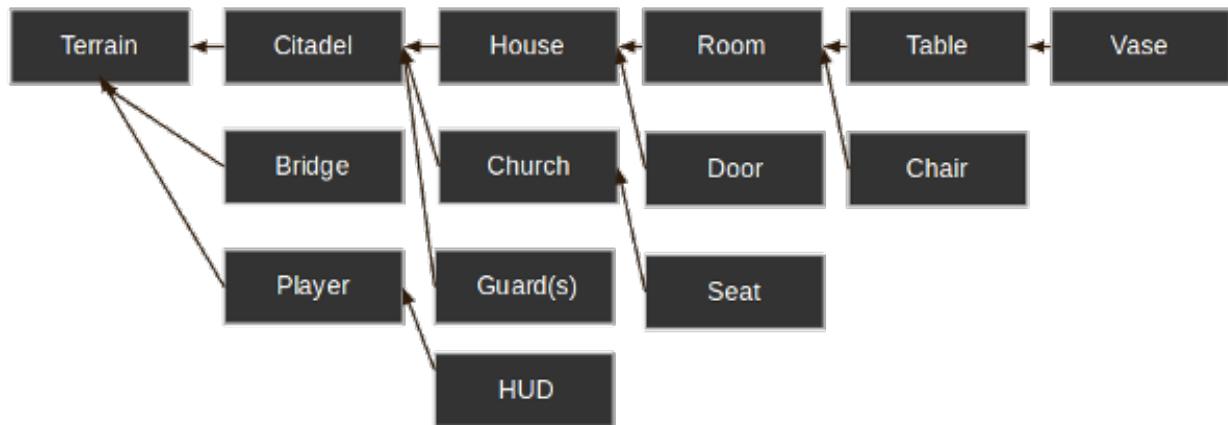


It's pretty easy to come up with a diagram like this for almost any kind of game. Just write down the parts of the game that you can visualize, and then add arrows to represent ownership of one component by another.

Once you have a diagram like this, the recommended process for making a game is to create a scene for each element listed in the diagram. You'll use instancing (either by code or directly in the editor) for the ownership relationships.

A lot of time spent in programming games (or software in general) is on designing an architecture and fitting game components to that architecture. Designing based on scenes replaces that approach and makes development much faster and more straightforward, allowing you to concentrate on the game logic itself. Because most game components map directly to a scene, using a design-based on scene instantiation means little other architectural code is needed.

Let's take a look at one more, somewhat more complex, example of an open-world type game with lots of assets and nested elements:



Take a look at the room element. Let's say we started there. We could make a couple of different room scenes, with different arrangements of furniture (also scenes) in them. Later, we could make a house scene, connecting rooms to make up its interior.

Then, we could make a citadel scene, which is made out of many instanced houses. Then, we could start working on the world map terrain, adding the citadel onto it.

Later, we could create scenes that represent guards (and other NPCs) and add them to the citadel as well. As a result, they would be indirectly added to the overall game world.

With Godot, it's easy to iterate on your game like this, as all you need to do is create and instance more scenes. Furthermore, the editor UI is designed to be user friendly for programmers and non-programmers alike. A typical team development process can involve 2D or 3D artists, level designers, game designers, and animators, all working with the editor interface.

2.4.3 Information overload!

This has been a lot of high level information dropped on you all at once. However, the important part of this tutorial was to create an awareness of how scenes and instancing are used in real projects.

Everything discussed here will become second nature to you once you start making games and putting these concepts into practice. For now, don't worry about it too much, and just go on to the next tutorial!

2.5 Scripting

2.5.1 Introduction

Before Godot 3.0, the only choice for scripting a game in Godot was to use *GDScript*. Nowadays, Godot has four (yes four!) official languages and the ability to add extra scripting languages dynamically!

This is great, mostly, due the large amount of flexibility provided, but it also makes our work of supporting languages more difficult.

The “Main” languages in Godot, though, are GDScript and VisualScript. The main reason to choose them is the level of integration with Godot, as this makes the experience smoother (both have very slick editor integration, while C# and C++ need to be edited in a separate IDE). If you are a big fan of statically typed languages, though, go with C# and C++.

GDScript

GDScript is, as mentioned above, the main language used in Godot. Using it has some really positive points compared to other languages due to the high integration with Godot:

- It's simple, elegant and designed to be friendly with users of other languages such as Lua, Python, Squirrel, etc.
- Loads and compiles blazingly fast.
- The editor integration is a pleasure to work with, with code completion expanding as far as completing nodes, signals, and many other items pertaining to the current scene being edited.
- Has vector types built-in (such as Vectors, transforms etc), making it efficient for heavy linear algebra.
- It supports multiple threads as well as statically typed languages (which is one of the limitation that avoided us going for other VMs such as Lua, Squirrel, etc).
- It uses no garbage collector, so it trades a small bit of automation (most objects are reference counted anyway), by determinism.
- The dynamic nature of it makes it easy to optimize chunks of it in C++ (via GDNative) if more performance is required at some point, without recompiling the engine.

If undecided, and have experience in programming (especially dynamically typed languages), go for GDScript!

VisualScript

Beginning 3.0, Godot offers *Visual Scripting*. This is a somewhat typical implementation of blocks and connections language, but adapted to how Godot works.

Visual scripting is a great tool for non-programmers, or even for developers with ample programming experience that want to make parts of the code more accessible to others (like game/level designers or artists).

It can also be used by programmers to build state machines or custom visual node workflows (like a dialogue system).

.NET / C#

As Microsoft's C# is a favorite amongst game developers, we have added solid and official support for it. C# is a very mature language with tons of code written for it. Support for this language was added thanks to a generous donation from Microsoft.

It has an excellent tradeoff between performance and ease to use, although the main item one must be aware of is the garbage collector.

For companies, this is usually the best choice, given the large amount of programmers that can be found in the labor market familiar with it, which reduces the learning time of the engine.

Since Godot uses the [Mono](#) .NET runtime, in theory any third-party .NET library or framework can be used for scripting in Godot, as well as any Common Language Infrastructure-compliant programming language, such as F#, Boo or ClojureCLR. In practice however, C# is the only officially-supported .NET option.

GDNative / C++

Finally, this is one of our brightest additions for the 3.0 release. GDNative allows scripting in C++ without requiring to recompile (or even restart) Godot.

Any C++ version can be used, and mixing compiler brands and versions for the generated shared libraries works perfectly, thanks to our use of an internal C API Bridge.

This language is the best choice for performance and does not need to be used for an entire game (other parts can be written in GDScript or Visual Script). Yet, the API is very clear and easy to use as it resembles, mostly, Godot's actual C++ API.

There are more languages available via the GDNative interface, but keep in mind we don't have official support for them so you will have to work with the provided in that case.

2.5.2 Scripting a scene

In the rest of this tutorial, we'll set up a simple GUI scene consisting of a button and a label, where pressing the button will update the label. This will demonstrate:

- how to write a basic script and attach it to a node
- how to hook up UI elements via *signals*
- how to write a script that can access other nodes in the scene

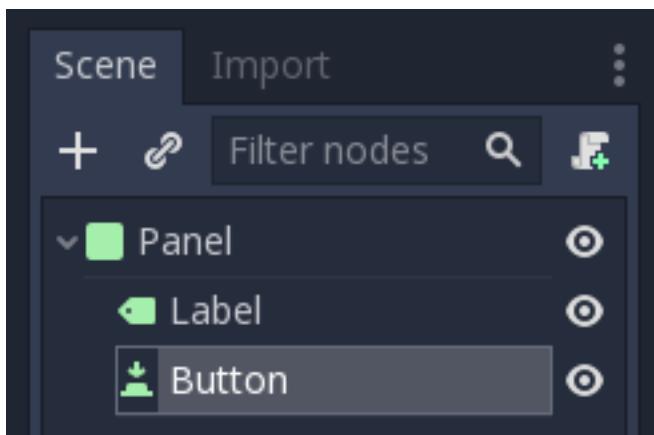
Before continuing, please make sure to read the *GDScript* reference. It's a simple language and the reference is short, so it will not take more than a few minutes to get an overview of the concepts.

Scene setup

Use the add node dialog to create the following hierarchy, with the following nodes:

- Panel
 - Label
 - Button

It should look like this in the scene tree:



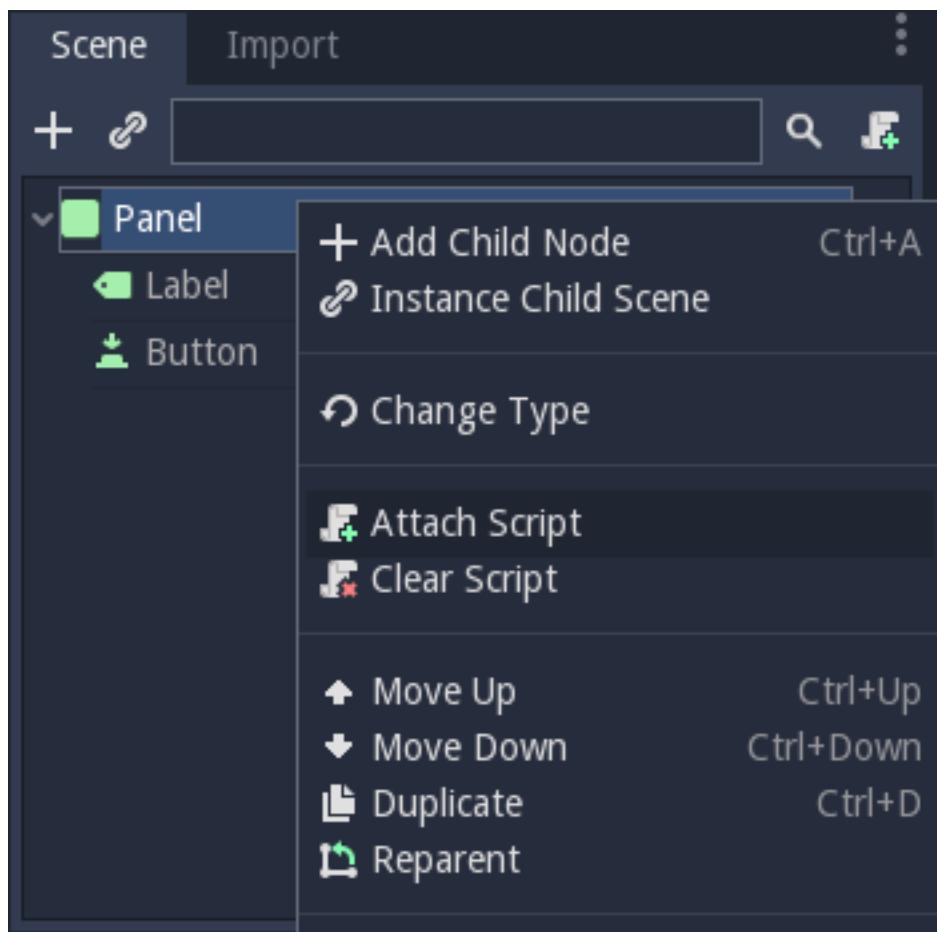
Use the 2D editor to position and resize the button and label so that they look like the image below. You can set the text in the Inspector pane.



Finally, save the scene, with a name such as “sayhello.tscn”

Adding a script

Right click on the panel node, and then select “Attach Script” in the context menu:

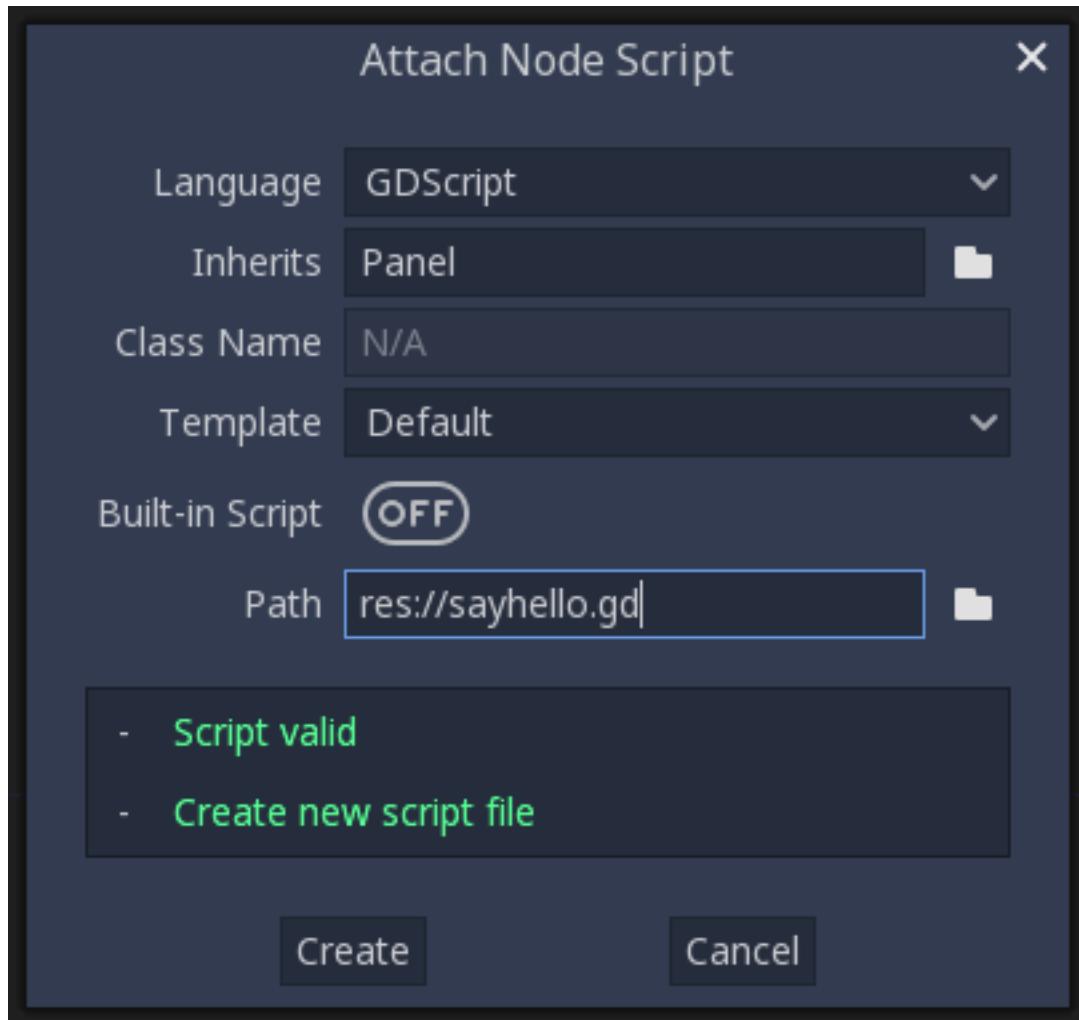


The script creation dialog will pop up. This dialog allows you to set the language, class name, and other relevant options.

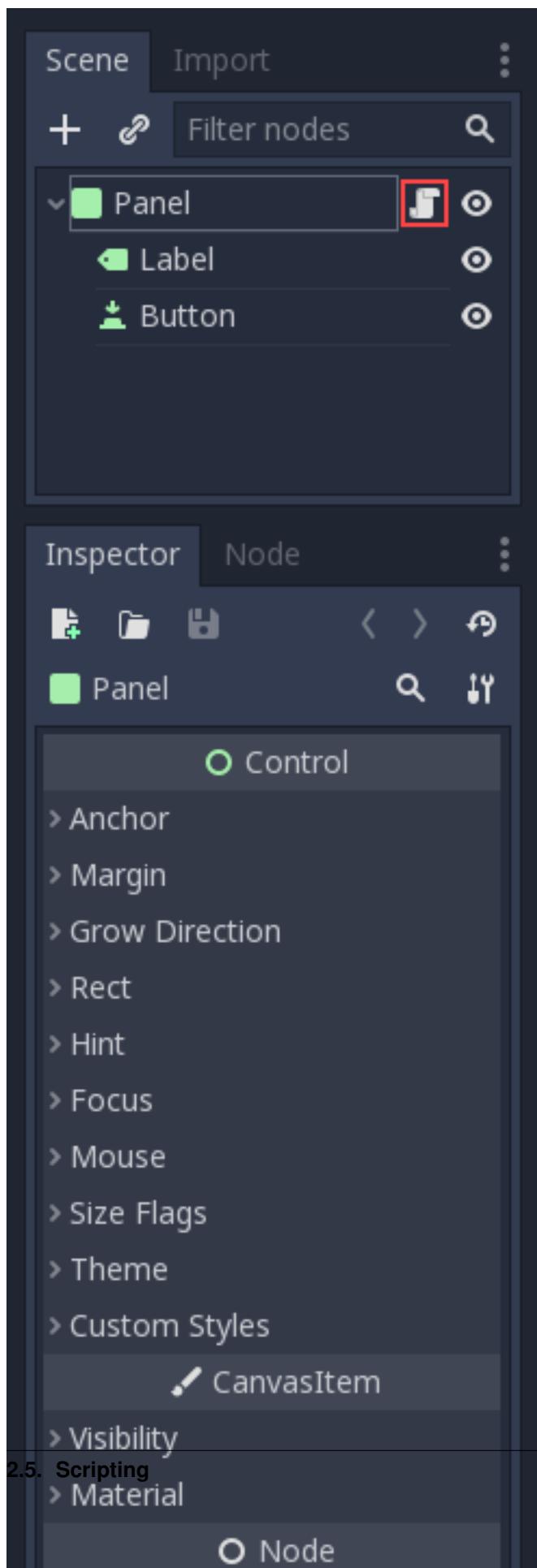
Actually, in GDScript, the file itself represents the class, so in this case, the class name field is not editable.

The node we're attaching the script to is a panel, so the "Inherits" field should automatically be filled in with "Panel". This is what we want as our script's goal is to extend this panel node's functionality.

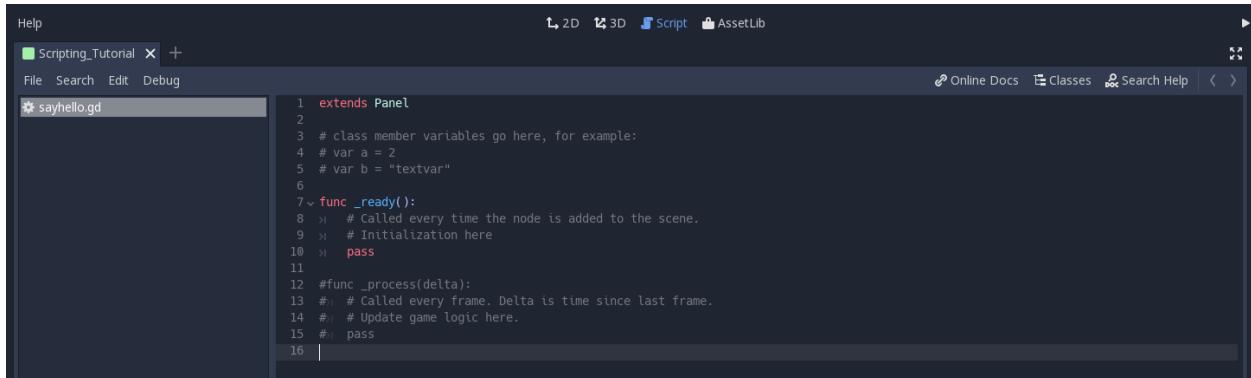
Finally, enter a path name for the script and select "Create":



Once this is done, the script will be created and added to the node. You can see this both as an extra icon in the node as well as in the script property:



To edit the script, select either of the highlighted buttons. This will bring you to the script editor where an existing template will be included by default:



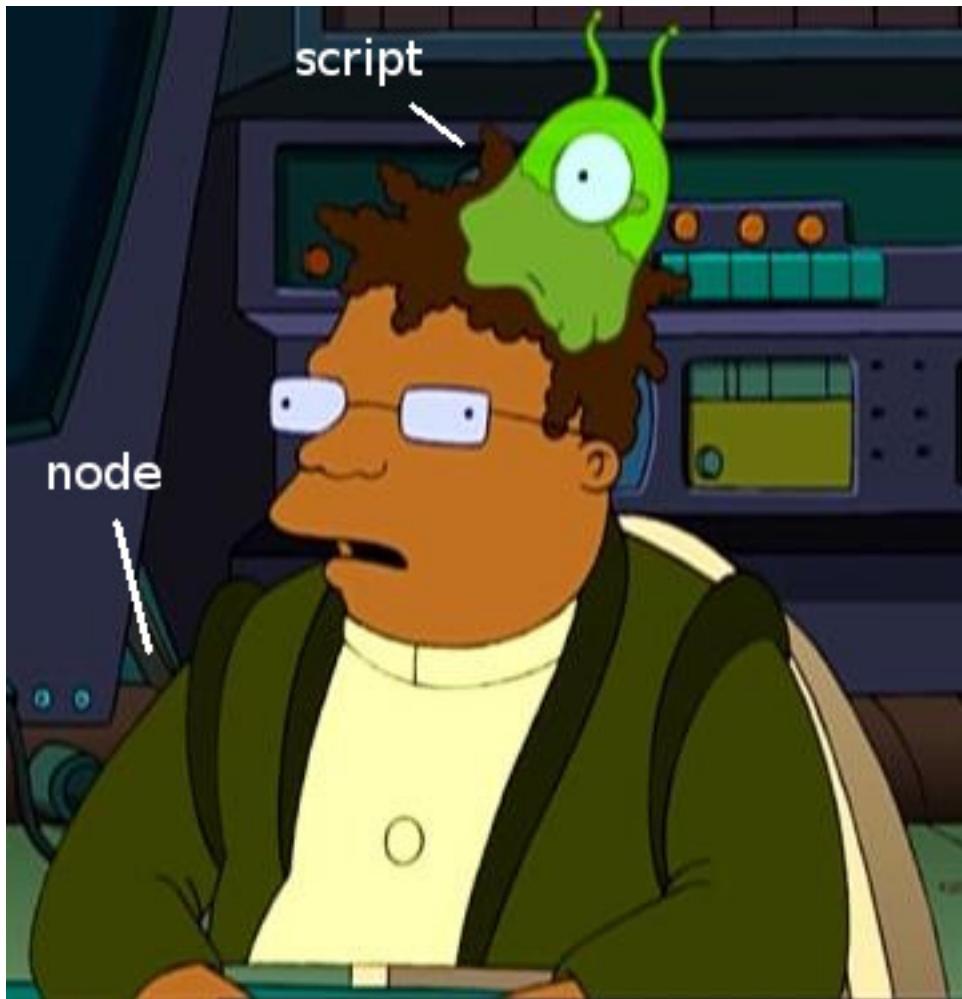
The screenshot shows the Godot Script Editor interface. The title bar says "Scripting_Tutorial". The main area displays the following GDScript code:

```
1 extends Panel
2
3 # Class member variables go here, for example:
4 # var a = 2
5 # var b = "textvar"
6
7 func _ready():
8    # Called every time the node is added to the scene.
9    # Initialization here
10   pass
11
12 #func _process(delta):
13 #    # Called every frame. Delta is time since last frame.
14 #    # Update game logic here.
15 #    pass
16 |
```

There is not much in there. The “_ready()” function is called when the node (and all its children) enter the active scene. (Note: “_ready()” is not the a constructor; the constructor is “_init()”).

The role of the script

A script adds behavior to a node. It is used to control how the node functions as well as how it interacts with other nodes (children, parent, siblings, etc.). The local scope of the script is the node. In other words, the script inherits the functions provided by that node.

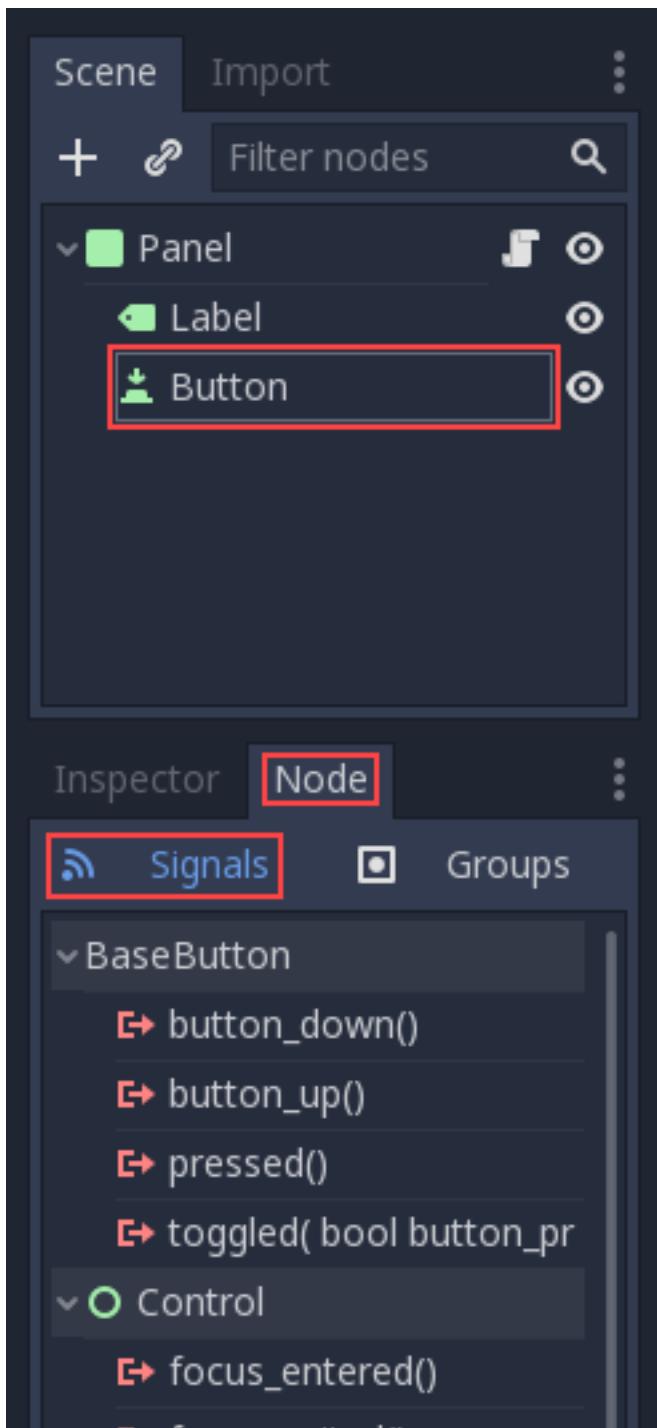


Handling a signal

Signals are “emitted” when some specific kind of action happens, and they can be connected to any function of any script instance. Signals are used mostly in GUI nodes (although other nodes have them too, and you can even define custom signals in your own scripts).

In this step, we’ll connect the “pressed” signal to a custom function.

The editor provides an interface for connecting signals to your scripts. You can access this by selecting the node in the scene tree and then selecting the “Node” tab. Next, make sure that you have “Signals” selected.



At this point, you could use the visual interface to hook up the “pressed” signal by double clicking on it and selecting a target node that already has a script attached to it. But for the sake of learning, we’re going to code up the connection manually.

To accomplish this, we will introduce a function that is probably the most used by Godot programmers, namely `Node.get_node()`. This function uses paths to fetch nodes anywhere in the scene, relative to the node that owns the script.

In our case, because the button and the label are siblings under the panel where the script is attached, you can fetch the button as follows:

GDScript

C#

```
get_node("Button")
```

```
GetNode("Button")
```

Next, write a function which will be called when the button is pressed:

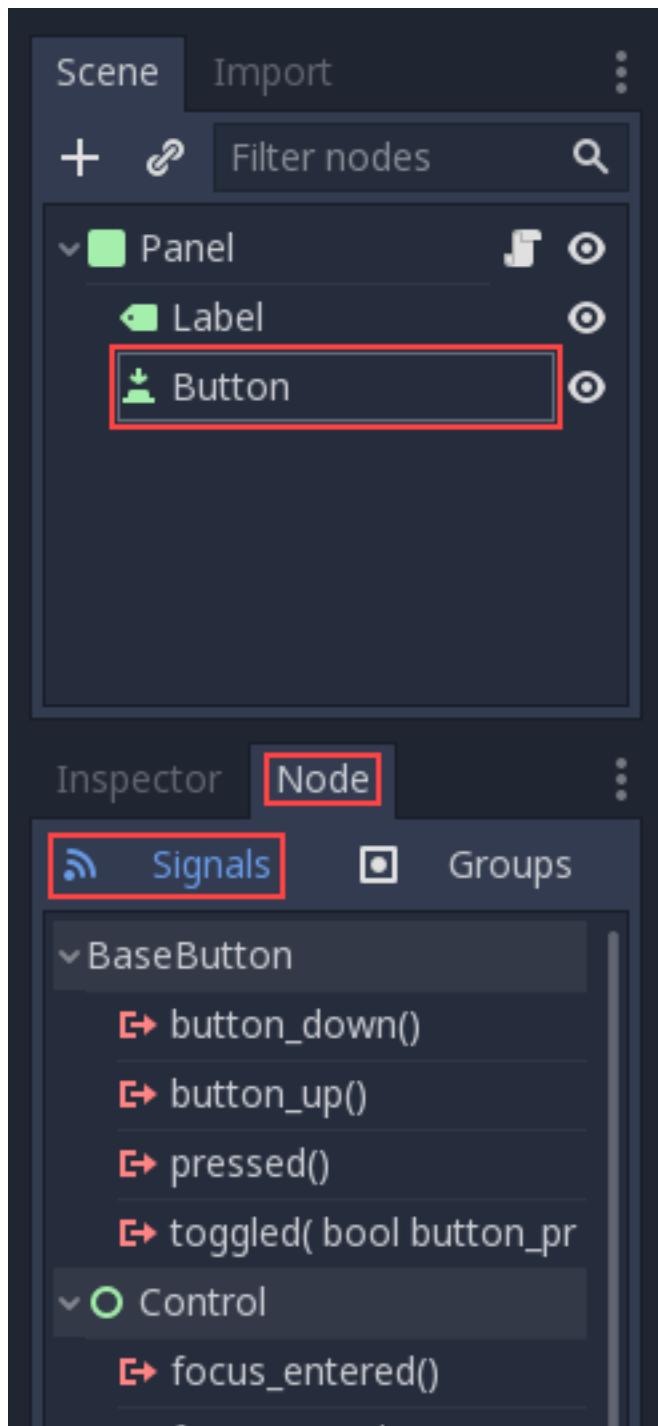
GDScript

C#

VS

```
func _on_button_pressed():
    get_node("Label").text = "HELLO!"
```

```
public void _OnButtonPressed()
{
    var label = (Label)GetNode("Label");
    label.Text = "HELLO!";
}
```



Finally, connect the button's “pressed” signal to that callback in `_ready()`, by using `Object.connect()`.

GDScript

C#

```
func _ready():
    get_node("Button").connect("pressed", self, "_on_button_pressed")
```

```
public override void _Ready()
{
    GetNode("Button").Connect("pressed", this, nameof(_OnButtonPressed));
}
```

The final script should look basically like this:

GDScript

C#

```
extends Panel

func _on_button_pressed():
    get_node("Label").text="HELLO!"

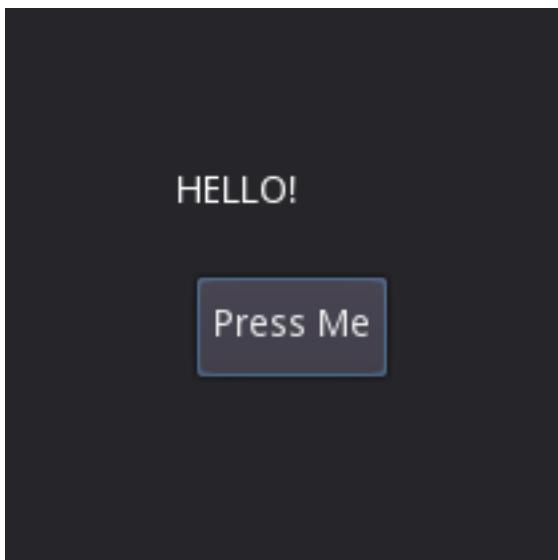
func _ready():
    get_node("Button").connect("pressed", self, "_on_button_pressed")
```

```
using Godot;

// IMPORTANT: the name of the class MUST match the filename exactly.
// this is case sensitive!
public class sayhello : Panel
{
    public void _OnButtonPressed()
    {
        var label = (Label)GetNode("Label");
        label.Text = "HELLO!";
    }

    public override void _Ready()
    {
        GetNode("Button").Connect("pressed", this, nameof(_OnButtonPressed));
    }
}
```

Run the scene and press the button. You should get the following result:



Why hello there! Congratulations on scripting your first scene.

Note: A common misunderstanding in this tutorial is how `get_node(path)` works. For some given node, `get_node(path)` searches its immediate children. In the above code, this means that *Button* must be a child of *Panel*. If *Button* were instead a child of *Label*, the code to obtain it would be:

GDScript

C#

```
# not for this case
# but just in case
get_node("Label/Button")
```

```
// not for this case
// but just in case
GetNode("Label/Button")
```

Also, remember that nodes are referenced by name, not by type.

2.6 Scripting (continued)

2.6.1 Processing

Several actions in Godot are triggered by callbacks or virtual functions, so there is no need to write code that runs all the time. Additionally, a lot can be done with animation players.

However, it is still a very common case to have a script process on every frame. There are two types of processing: idle processing and physics processing.

Idle processing is activated automatically when the method `Node._process()` is found in a script. It can be turned off (and back on) with the `Node.set_process()` function.

This method will be called every frame drawn, so it's fully depend on the frames per second (FPS) of the application:

```
func _process(delta):
    # do something...
```

The delta parameter describes the time elapsed (in seconds, as a floating point) since the previous call to `_process()`.

This delta parameter can be used to make sure things always take the same time, regardless of the hardware the game is running on.

For example, movement is often multiplied with the time delta to make the movement constant and independent from the frame rate.

Physics processing (`_physics_process()`) is similar, but it should be used for all the processes that must happen before each physics step. For example, to move a character. It always runs before a physics step and it is called at fixed time intervals, 60 times per second by default. Change the value in the Project Settings.

The function `_process()` instead is not synced with physics. Its frame rate is not constant and dependent on hardware and game optimization. Its execution is done after the physics step on single thread games.

A simple way to test this is to create a scene with a single Label node, with the following script:

```
extends Label

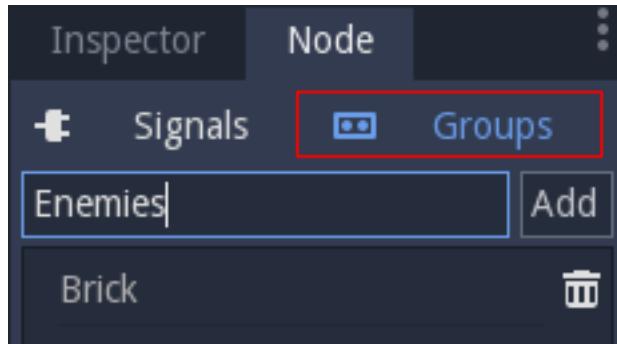
var accum=0
```

```
func _process(delta):
    accum += delta
    text = str(accum) # text is a built-in label property
```

Which will show a counter increasing each frame.

2.6.2 Groups

Nodes can be added to groups (as many as desired per node). This is a simple yet useful feature for organizing large scenes. There are two ways to do this: the first is from the UI, from the Groups button under the Node panel:



And the second from code. One useful example would be to tag scenes which are enemies.

```
func _ready():
    add_to_group("enemies")
```

This way, if the player is discovered sneaking into the secret base, all enemies can be notified about the alarm sounding, by using `SceneTree.call_group()`:

```
func _on_discovered(): # this is a fictional function
    get_tree().call_group("enemies", "player_was_discovered")
```

The above code calls the function “player_was_discovered” on every member of the group “enemies”.

Optionally, it is possible to get the full list of “enemies” nodes by calling `SceneTree.get_nodes_in_group()`:

```
var enemies = get_tree().get_nodes_in_group("enemies")
```

More will be added about `SceneTree` later.

2.6.3 Notifications

Godot has a system of notifications. This is usually not needed for scripting, as it's too low level and virtual functions are provided for most of them. It's just good to know they exist. Simply add a `Object._notification()` function in your script:

```
func _notification(what):
    if (what == NOTIFICATION_READY):
        print("This is the same as overriding _ready()...")
    elif (what == NOTIFICATION_PROCESS):
        var delta = get_process_time()
        print("This is the same as overriding _process()...")
```

The documentation of each class in the *Class Reference* shows the notifications it can receive. However, for most cases GDScript provides simpler overrideable functions.

2.6.4 Overrideable functions

Nodes provide many useful overrideable functions, which are described as follows:

```
func _enter_tree():
    # When the node enters the _Scene Tree_, it becomes active
    # and this function is called. Children nodes have not entered
    # the active scene yet. In general, it's better to use _ready()
    # for most cases.
    pass

func _ready():
    # This function is called after _enter_tree, but it ensures
    # that all children nodes have also entered the _Scene Tree_,
    # and became active.
    pass

func _exit_tree():
    # When the node exits the _Scene Tree_, this function is called.
    # Children nodes have all exited the _Scene Tree_ at this point
    # and all became inactive.
    pass

func _process(delta):
    # This function is called every frame.
    pass

func _physics_process(delta):
    # This is called every physics frame.
    pass

func _paused():
    # Called when game is paused. After this call, the node will not receive
    # any more process callbacks.
    pass

func _unpaused():
    # Called when game is unpause.
    pass
```

As mentioned before, it's best to use these functions.

2.6.5 Creating nodes

To create a node from code, call the `.new()` method, just like for any other class based datatype. Example:

```
var s
func _ready():
    s = Sprite.new() # create a new sprite!
    add_child(s) # add it as a child of this node
```

To delete a node, be it inside or outside the scene, “`free()`” must be used:

```
func _someaction():
    s.free() # immediately removes the node from the scene and frees it
```

When a node is freed, it also frees all its children nodes. Because of this, manually deleting nodes is much simpler than it appears. Just free the base node and everything else in the sub-tree goes away with it.

However, it might happen very often that we want to delete a node that is currently “blocked”, because it is emitting a signal or calling a function. This will result in crashing the game. Running Godot in the debugger often will catch this case and warn you about it.

The safest way to delete a node is by using `Node.queue_free()`. This erases the node safely during idle.

```
func _someaction():
    s.queue_free() # remove the node and delete it while nothing is happening
```

2.6.6 Instancing scenes

Instancing a scene from code is pretty easy and done in two steps. The first one is to load the scene from disk.

```
var scene = load("res://myscene.tscn") # will load when the script is instanced
```

Preloading it can be more convenient sometimes, as it happens at parse time.

```
var scene = preload("res://myscene.tscn") # will load when parsing the script
```

But ‘scene’ is not yet a node for containing subnodes. It’s packed in a special resource called *PackedScene*. To create the actual node, the function *PackedScene.instance()* must be called. This will return the tree of nodes that can be added to the active scene:

```
var node = scene.instance()
add_child(node)
```

The advantage of this two-step process is that a packed scene may be kept loaded and ready to use, so it can be used to create as many instances as desired. This is especially useful to quickly instance several enemies, bullets, etc., in the active scene.

2.7 Your First Game

2.7.1 Overview

This tutorial will guide you through making your first Godot Engine project. You will learn how the Godot Engine editor works, how to structure a project, and how to build a 2D game.

Note: This project is an introduction to the Godot Engine. It assumes that you have some programming experience already. If you’re new to programming entirely, you should start here: *Scripting*.

The game is called “*Dodge the Creeps*”. Your character must move and avoid the enemies for as long as possible. Here is a preview of the final result:

Why 2D? 3D games are much more complex than 2D ones. You should stick to 2D until you have a good understanding of the game development process.

2.7.2 Project Setup

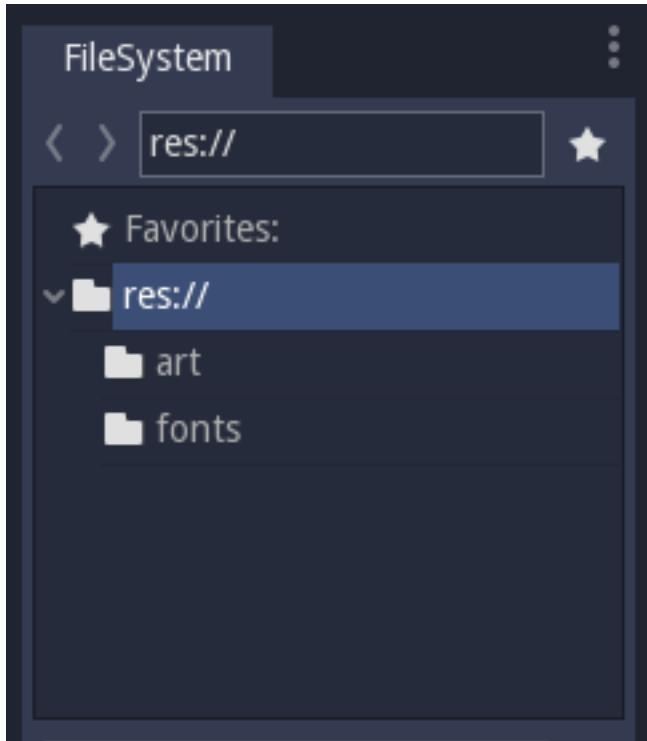
Launch Godot and create a new project. Then, download `dodge_assets.zip` - the images and sounds you'll be using to make the game. Unzip these files in your new project folder.

Note: For this tutorial, we will assume you are already familiar with the Godot Engine editor. If you haven't read *Scenes and nodes*, do so now for an explanation of setting up a project and using the editor.

This game will use "portrait" mode, so we need to adjust the size of the game window. Click on Project -> Project Settings -> Display -> Window and set Width to 480 and Height to 720.

Organizing the Project

In this project, we will make 3 independent scenes: Player, Mob, and HUD, which we will combine into the game's Main scene. In a larger project, it might be useful to make folders to hold the various scenes and their scripts, but for this relatively small game, you can save your scenes and scripts in the root folder, which is referred to as `res://`. You can see your project folders in the Filesystem Dock in the upper left corner:

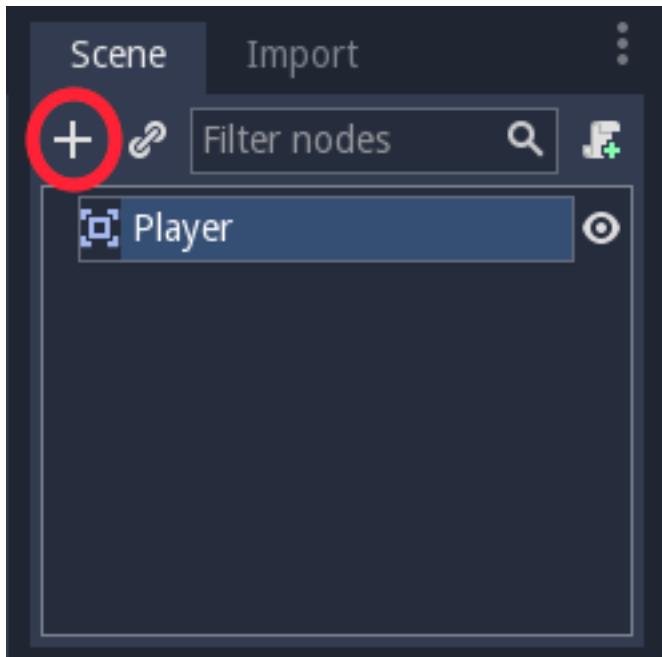


2.7.3 Player Scene

The first scene we make defines the "Player" object. One of the benefits of creating a separate Player scene is that we can test it separately, even before we've created the other parts of the game.

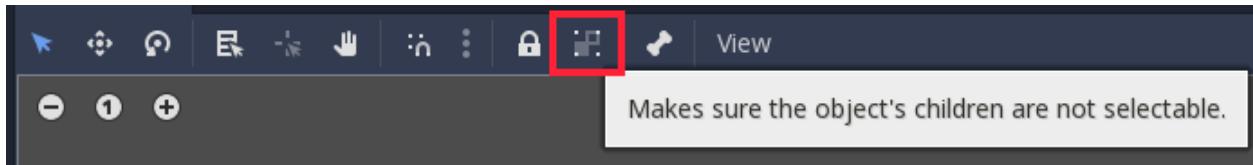
Node Structure

To begin, click the "Add/Create a New Node" button and add an *Area2D* node to the scene.



With `Area2D` we can detect other objects that overlap or run into the player. Change its name to `Player`. This is the scene’s “root” or top-level node. We can add additional nodes to the player to add functionality.

Before we add any children to the `Player` node, we want to make sure we don’t accidentally move or resize them by clicking on them. Select the player node and click the icon next to the lock - its tooltip says “Makes sure the object’s children are not selectable.”



Save the scene (click Scene -> Save, or press Control+S on Windows/Linux or Command+S on Mac).

Note: In this project, we will be following the Godot Engine naming conventions. Classes (Nodes) use CapWords, variables and functions use snake_case, and constants use ALL_CAPS.

Sprite Animation

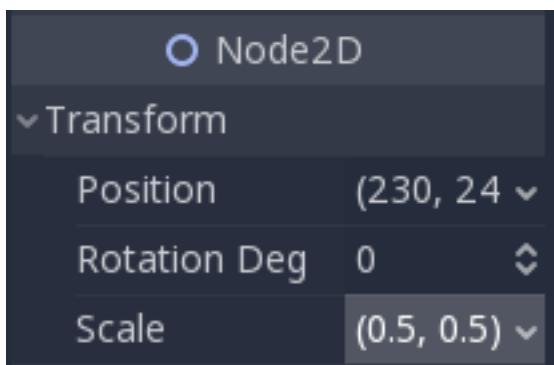
Click on the `Player` node and add an `AnimatedSprite` node as a child. The `AnimatedSprite` will handle the appearance and animations for our player. Notice that there is a warning symbol next to the node. An `AnimatedSprite` requires a `SpriteFrames` resource, which is a list of the animation(s) it can display. To create one, find the `Frames` property in the Inspector and click “`<null>`” -> “New `SpriteFrames`”. Next, in the same location, click `<SpriteFrames>` to open the “`SpriteFrames`” panel:



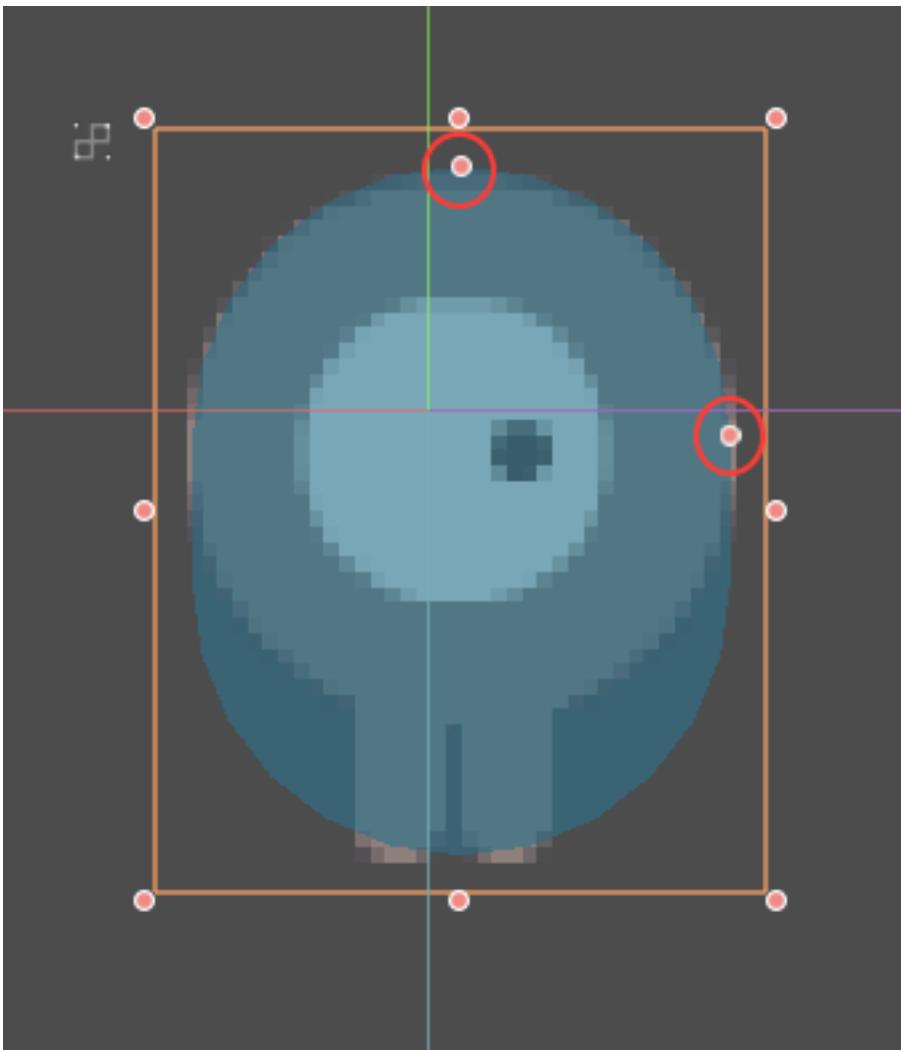
On the left is a list of animations. Click the “default” one and rename it to “right”. Then click the “Add” button to create a second animation named “up”. Drag the two images for each animation into “Animation Frames” side of the panel:



The player images are a bit too large for the game window, so we need to scale them down. Click on the AnimatedSprite node and set the Scale property to $(0.5, 0.5)$. You can find it in the Inspector under the Node2D heading.

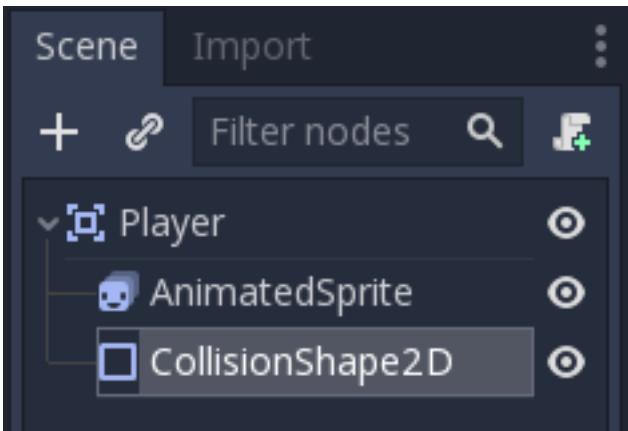


Finally, add a *CollisionShape2D* as a child of the Player. This will determine the player’s “hitbox”, or the bounds of its collision area. For this character, a CapsuleShape2D gives the best fit, so next to “Shape” in the Inspector, click “<null>” -> “New CapsuleShape2D”. Resize the shape to cover the sprite:



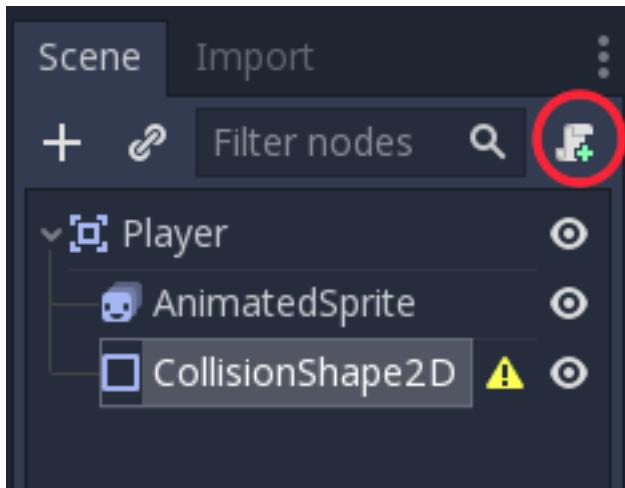
Warning: Remember not to scale the shape's outline! Only use the size handles (red) to adjust the shape!

When you're finished, your Player scene should look like this:

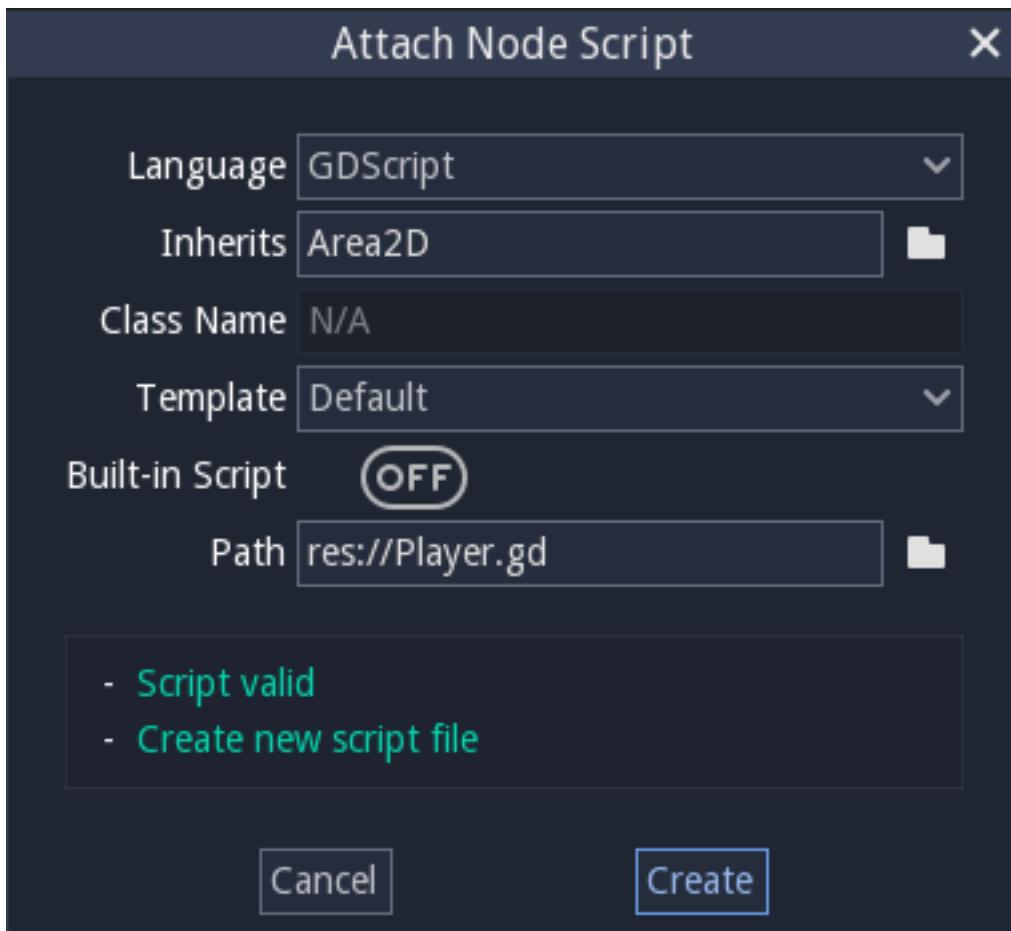


Moving the Player

Now we need to add some functionality that we can't get from a built-in node, so we'll add a script. Click the Player node and click the “Add Script” button:



In the script settings window, you can leave the default settings, just click “Create”:



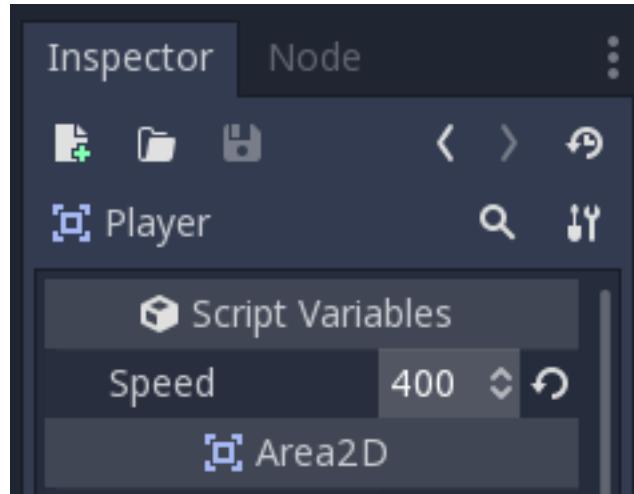
Note: If this is your first time encountering GDScript please read *Scripting* first.

Start by declaring the member variables this object will need:

```
extends Area2D

export (int) var SPEED # how fast the player will move (pixels/sec)
var velocity = Vector2() # the player's movement vector
var screensize # size of the game window
```

Using the `export` keyword on the first variable `SPEED` allows us to set its value in the Inspector. This can be very handy for values that you want to be able to adjust just like a node's built-in properties. Click on the `Player` node and set the speed property to 400.



The `_ready()` function is called when a node enters the scene tree, so that's a good time to find the size of the game window:

```
func _ready():
    screensize = get_viewport_rect().size
```

Now we can use the `_process()` function to define what the player will do. The `_process()` function is called on every frame, so we'll use it to update elements of our game which we expect to be changing often. Here we'll have it:

- check for input
- move in the given direction
- play the appropriate animation.

First, we need to check the inputs - is the player pressing a key? For this game, we have 4 direction inputs to check. Input actions are defined in the Project Settings under “Input Map”. You can define custom events and assign different keys, mouse events, or other inputs to them. For this demo, we will use the default events that are assigned to the arrow keys on the keyboard.

You can detect whether a key is pressed using `Input.is_action_pressed()`, which returns `true` if it is pressed or `false` if it isn't.

```
func _process(delta):
    velocity = Vector2()
```

```
if Input.is_action_pressed("ui_right"):
    velocity.x += 1
if Input.is_action_pressed("ui_left"):
    velocity.x -= 1
if Input.is_action_pressed("ui_down"):
    velocity.y += 1
if Input.is_action_pressed("ui_up"):
    velocity.y -= 1
if velocity.length() > 0:
    velocity = velocity.normalized() * SPEED
    $AnimatedSprite.play()
else:
    $AnimatedSprite.stop()
```

We check each input and add/subtract from the `velocity` to obtain a total direction. For example, if you hold right and down at the same time, the resulting `velocity` vector will be $(1, 1)$. In this case, since we're adding a horizontal and a vertical movement, the player would move *faster* than if it just moved horizontally.

We can prevent that if we *normalize* the velocity, which means we set its *length* to 1, and multiply by the desired speed. This means no more fast diagonal movement.

Tip: If you've never used vector math before (or just need a refresher) you can see an explanation of vector usage in Godot at *Vector math*. It's good to know but won't be necessary for the rest of this tutorial.

We also check whether the player is moving so we can start or stop the `AnimatedSprite` animation.

Tip: `$` returns the node at the relative path from this node, or returns null if the node is not found. Since `AnimatedSprite` is a child of the current node, we can just use `$AnimatedSprite`.

`$` is the short hand for `get_node()`. So in the code above, `$AnimatedSprite.play()` is the same as `get_node("AnimatedSprite").play()`.

Now that we have a movement direction, we can update the player's position and use `clamp()` to prevent it from leaving the screen:

```
position += velocity * delta
position.x = clamp(position.x, 0, screensize.x)
position.y = clamp(position.y, 0, screensize.y)
```

Tip: *Clamping* a value means restricting it to a given minimum/maximum range.

Click “Play the Edited Scene. (F6)” and confirm you can move the player around the screen in all directions.

Warning: If you get an error in the “Debugger” panel that refers to a “null instance”, this likely means you spelled the node name wrong. Node names are case sensitive and `$NodeName` or `get_node ("NodeName")` must match the name you see in the scene tree.

Choosing Animations

Now that the player can move, we need to change which animation the AnimatedSprite is playing based on direction. We have a “right” animation, which should be flipped horizontally (using the `flip_h` property) for left movement, and an “up” animation, which should be flipped vertically (`flip_v`) for downward movement. Let’s place this code at the end of our `_process()` function:

```
if velocity.x != 0:
    $AnimatedSprite.animation = "right"
    $AnimatedSprite.flip_v = false
    $AnimatedSprite.flip_h = velocity.x < 0
elif velocity.y != 0:
    $AnimatedSprite.animation = "up"
    $AnimatedSprite.flip_v = velocity.y > 0
```

Play the scene again and check that the animations are correct in each of the directions. When you’re sure that movement is working correctly, add this line to `_ready()` so the player will be hidden when the game starts:

```
hide()
```

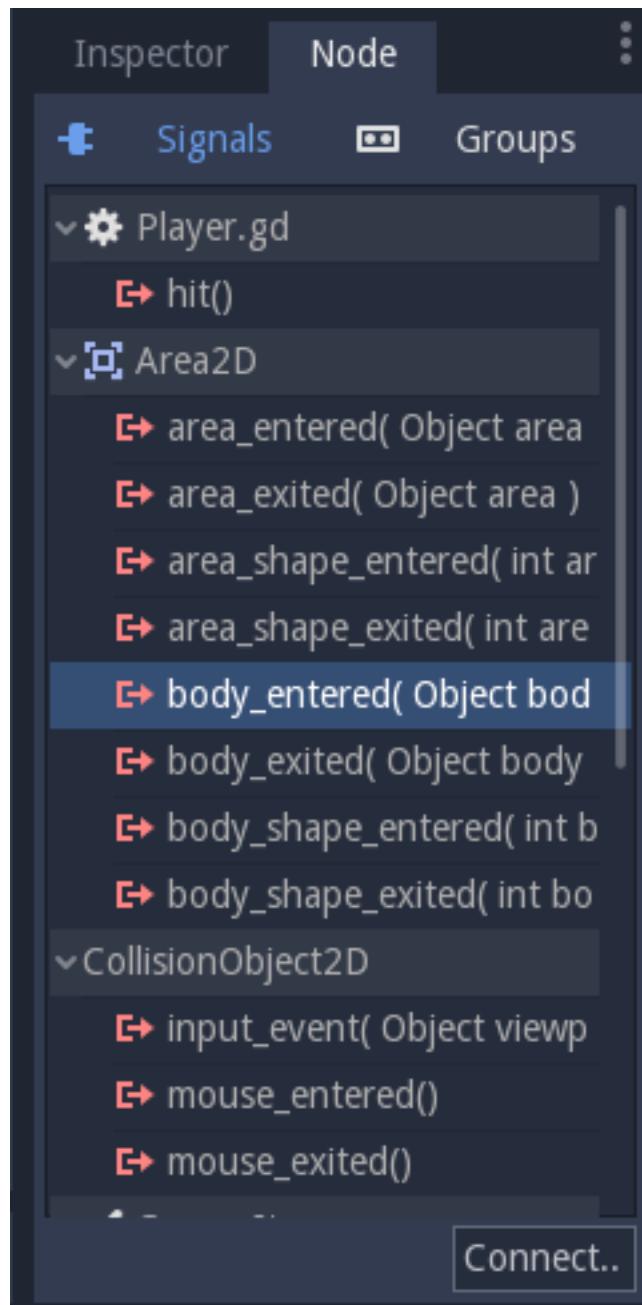
Preparing for Collisions

We want the player to detect when it is hit by an enemy, but we haven’t made any enemies yet! That’s OK because we’re going to use Godot’s *signal* functionality to make it work.

Add the following at the top of the script (after `extends Area2d`):

```
signal hit
```

This defines a custom signal called “hit” that we will have our player emit (send out) when it collides with an enemy. We will use the Area2D to detect the collision. Select the Player node and click the “Node” tab next to the Inspector to see the list of signals the player can emit:



Notice our custom “hit” signal is there as well! Since our enemies are going to be RigidBody2D nodes, we want the body_entered(Object body) signal - that will be emitted when a body contacts the player. Click “Connect..” and then “Connect” again on the “Connecting Signal” window - we don’t need to change any of those settings. Godot will automatically create a function called _on_Player_body_entered in your player’s script.

Tip: When connecting a signal, instead of having Godot create a function for you, you can also give the name of an existing function that you want to link the signal to.

Add this code to the function:

```
func _on_Player_body_entered( body ):
    hide() # Player disappears after being hit
```

```
emit_signal("hit")
$CollisionShape2D.disabled = true
```

Note: Disabling the area's collision shape means it won't detect collisions. By turning it off, we make sure we don't trigger the `hit` signal more than once.

The last piece for our player is to add a function we can call to reset the player when starting a new game.

```
func start(pos):
    position = pos
    show()
    $CollisionShape2D.disabled = false
```

2.7.4 Enemy Scene

Now it's time to make the enemies our player will have to dodge. Their behavior will not be very complex: mobs will spawn randomly at the edges of the screen and move in a straight line (in a random direction), then despawn when they go offscreen.

We will build this into a `Mob` scene, which we can then *instance* to create any number of independent mobs in the game.

Node Setup

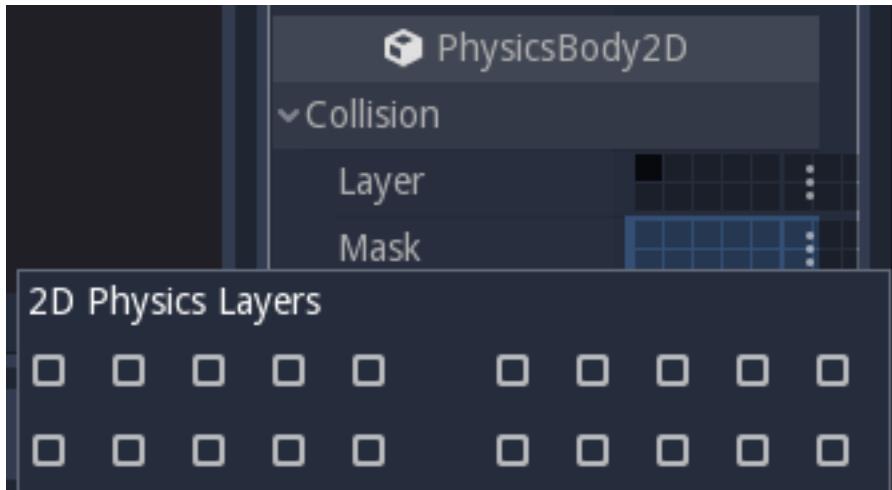
Click Scene -> New Scene and we'll create the `Mob`.

The `Mob` scene will use the following nodes:

- `RigidBody2D` (named `Mob`)
 - `AnimatedSprite`
 - `CollisionShape2D`
 - `VisibilityNotifier2D` (named `Visibility`)

Don't forget to set the children so they can't be selected, like you did with the `Player` scene.

In the `RigidBody2D` properties, set `Gravity Scale` to 0 (so that the mob will not fall downward). In addition, under the `PhysicsBody2D` section in the Inspector, click the `Mask` property and uncheck the first box. This will ensure that the mobs do not collide with each other.



Set up the *AnimatedSprite* like you did for the player. This time, we have 3 animations: “fly”, “swim”, and “walk”. Set the `Playing` property in the Inspector to “On” and adjust the “Speed (FPS)” setting as shown below. We’ll select one of these randomly so that the mobs will have some variety.

Like the player images, these mob images need to be scaled down. Set the *AnimatedSprite*’s `Scale` property to `(0.75, 0.75)`.

As in the `Player` scene, add a `CapsuleShape2D` for the collision. To align the shape with the image, you’ll need to set the `Rotation Deg` property to `90` under `Node2D`.

Enemy Script

Add a script to the Mob and add the following member variables:

```
extends RigidBody2D

export (int) var MIN_SPEED # minimum speed range
export (int) var MAX_SPEED # maximum speed range
var mob_types = ["walk", "swim", "fly"]
```

We’ll pick a random value between `MIN_SPEED` and `MAX_SPEED` for how fast each mob will move (it would be boring if they were all moving at the same speed). Set them to 150 and 250 in the Inspector. We also have an array containing the names of the three animations, which we’ll use to select a random one.

Now let’s look at the rest of the script. In `_ready()` we choose a random one of the three animation types:

```
func _ready():
    $AnimatedSprite.animation = mob_types[randi() % mob_types.size()]
```

Note: You must use `randomize()` if you want your sequence of “random” numbers to be different every time you run the scene. We’re going to use `randomize()` in our Main scene, so we won’t need it here. `randi() % n` is the standard way to get a random integer between 0 and `n-1`.

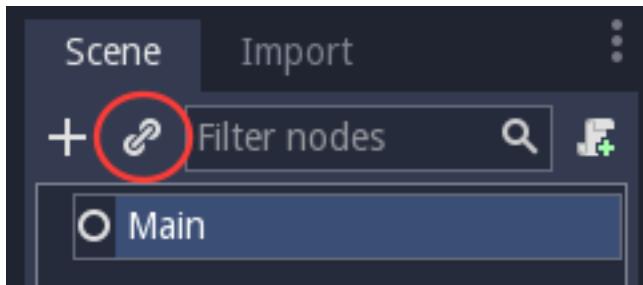
The last piece is to make the mobs delete themselves when they leave the screen. Connect the `screen_exited()` signal of the `Visibility` node and add this code:

```
func _on_Visibility_screen_exited():
    queue_free()
```

That completes the *Mob* scene.

2.7.5 Main Scene

Now it's time to bring it all together. Create a new scene and add a *Node* named `Main`. Click the “Instance” button and select your saved `Player.tscn`.



Note: See *Instancing* to learn more about instancing.

Now add the following nodes as children of `Main`, and name them as shown (values are in seconds):

- *Timer* (named `MobTimer`) - to control how often mobs spawn
- *Timer* (named `ScoreTimer`) - to increment the score every second
- *Timer* (named `StartTimer`) - to give a delay before starting
- *Position2D* (named `StartPosition`) - to indicate the player's start position

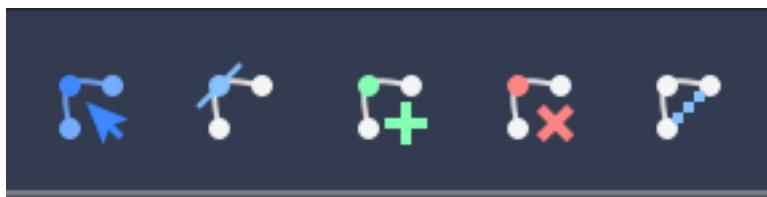
Set the `Wait Time` property of each of the *Timer* nodes as follows:

- `MobTimer`: 0.5
- `ScoreTimer`: 1
- `StartTimer`: 2

In addition, set the `One Shot` property of `StartTimer` to “On” and set `Position` of the `StartPosition` node to (240, 450).

Spawning Mobs

The `Main` node will be spawning new mobs, and we want them to appear at a random location on the edge of the screen. Add a *Path2D* named `MobPath` as a child of `Main`. When you select the *Path2D* node you will see some new buttons appear at the top of the editor:



Select the middle one (“Add Point”) and draw the path by clicking to add the points shown. To have the points snap to the grid, make sure “Snap to Grid” is checked. This option can be found under the “Snapping Options” button to the left of the “Lock” button. It appears as a series of three vertical dots.

Important: Draw the path in *clockwise* order, or your mobs will spawn pointing *outwards* instead of *inwards*!

After placing point 4 in the image, click the “Close Curve” button and your curve will be complete.

Now that the path is defined, add a *PathFollow2D* node as a child of *MobPath* and name it *MobSpawnLocation*. This node will automatically rotate and follow the path as it moves, so we can use it to select a random position and direction along the path.

Main Script

Add a script to Main. At the top of the script we use `export (PackedScene)` to allow us to choose the Mob scene we want to instance.

```
extends Node

export (PackedScene) var Mob
var score

func _ready():
    randomize()
```

Drag the *Mob.tscn* from the “FileSystem” panel and drop it in the *Mob* property.

Next, click on the Player and connect the `hit` signal. We want to make a new function named `game_over`, which will handle what needs to happen when a game ends. Type “`game_over`” in the “Method In Node” box at the bottom of the “Connecting Signal” window. Add the following code, as well as a `new_game` function to set everything up for a new game:

```
func game_over():
    $ScoreTimer.stop()
    $MobTimer.stop()

func new_game():
    score = 0
    $Player.start($StartPosition.position)
    $StartTimer.start()
```

Now connect the `timeout()` signal of each of the Timer nodes. `StartTimer` will start the other two timers. `ScoreTimer` will increment the score by 1.

```
func _on_StartTimer_timeout():
    $MobTimer.start()
    $ScoreTimer.start()

func _on_ScoreTimer_timeout():
    score += 1
```

In `_on_MobTimer_timeout()` we will create a mob instance, pick a random starting location along the `Path2D`, and set the mob in motion. The `PathFollow2D` node will automatically rotate as it follows the path, so we will use that to select the mob’s direction as well as its position.

Note that a new instance must be added to the scene using `add_child()`.

```
func _on_MobTimer_timeout():
    # choose a random location on the Path2D
    $MobPath/MobSpawnLocation.set_offset(randi())
    # create a Mob instance and add it to the scene
    var mob = Mob.instance()
    add_child(mob)
    # set the mob's direction perpendicular to the path direction
    var direction = $MobPath/MobSpawnLocation.rotation + PI/2
    # set the mob's position to the random location
    mob.position = $MobPath/MobSpawnLocation.position
    # add some randomness to the direction
    direction += rand_range(-PI/4, PI/4)
    mob.rotation = direction
    # choose the velocity
    mob.set_linear_velocity(Vector2(rand_range(mob.MIN_SPEED, mob.MAX_SPEED), 0).
    ↪rotated(direction))
```

Important: In functions requiring angles, GDScript uses *radians*, not degrees. If you’re more comfortable working with degrees, you’ll need to use the `deg2rad()` and `rad2deg()` functions to convert between the two measures.

2.7.6 HUD

The final piece our game needs is a UI: an interface to display things like score, a “game over” message, and a restart button. Create a new scene, and add a *CanvasLayer* node named **HUD** (“HUD” stands for “heads-up display”, meaning an informational display that appears as an overlay, on top of the game view).

The *CanvasLayer* node lets us draw our UI elements on the layer above the rest of the game so that the information it displays doesn’t get covered up by any game elements like the player or the mobs.

The **HUD** displays the following information:

- Score, changed by `ScoreTimer`
- A message, such as “Game Over” or “Get Ready!”
- A “Start” button to begin the game

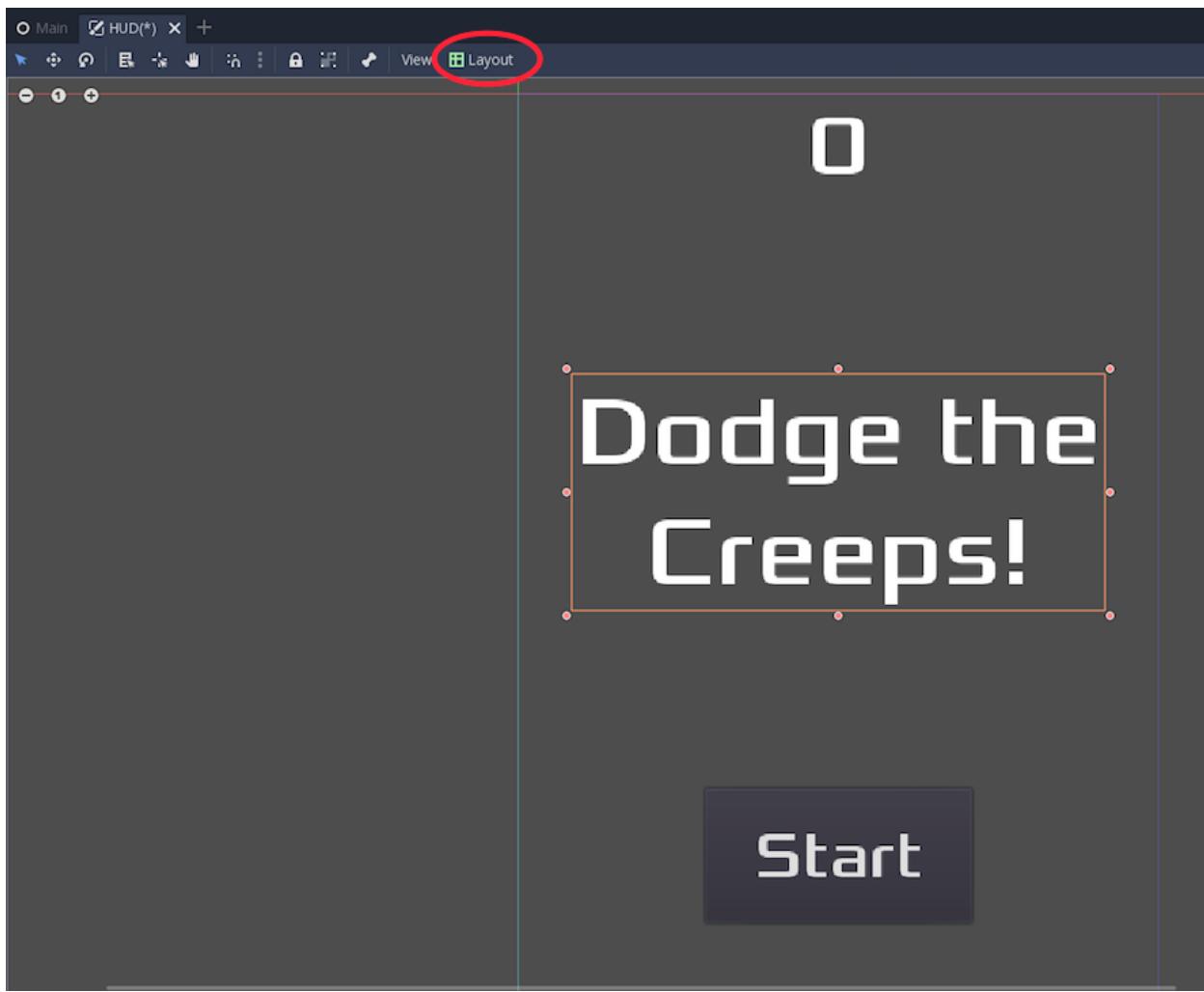
The basic node for UI elements is *Control*. To create our UI, we’ll use two types of *Control* nodes: The *Label* and the *Button*.

Create the following children of the **HUD** node:

- *Label* (named `ScoreLabel`)
- *Label* (named `MessageLabel`)
- *Button* (named `StartButton`)
- *Timer* (named `MessageTimer`)

Note: Anchors and Margins *Control* nodes have a position and size, but they also have anchors and margins. Anchors define the origin, or the reference point for the edges of the node. Margins update automatically when you move or resize a control node. They represent the distance from the control node’s edges to its anchor. See *Design interfaces with the Control nodes* for more details.

Arrange the nodes as shown below. Click the “Anchor” button to set a Control node’s anchor:



You can drag the nodes to place them manually, or for more precise placement, use the following settings:

ScoreLabel

- Layout: “Center Top”
- Margin:
 - Left: -25
 - Top: 0
 - Right: 25
 - Bottom: 100
- Text: 0

MessageLabel

- Layout: “Center Bottom”

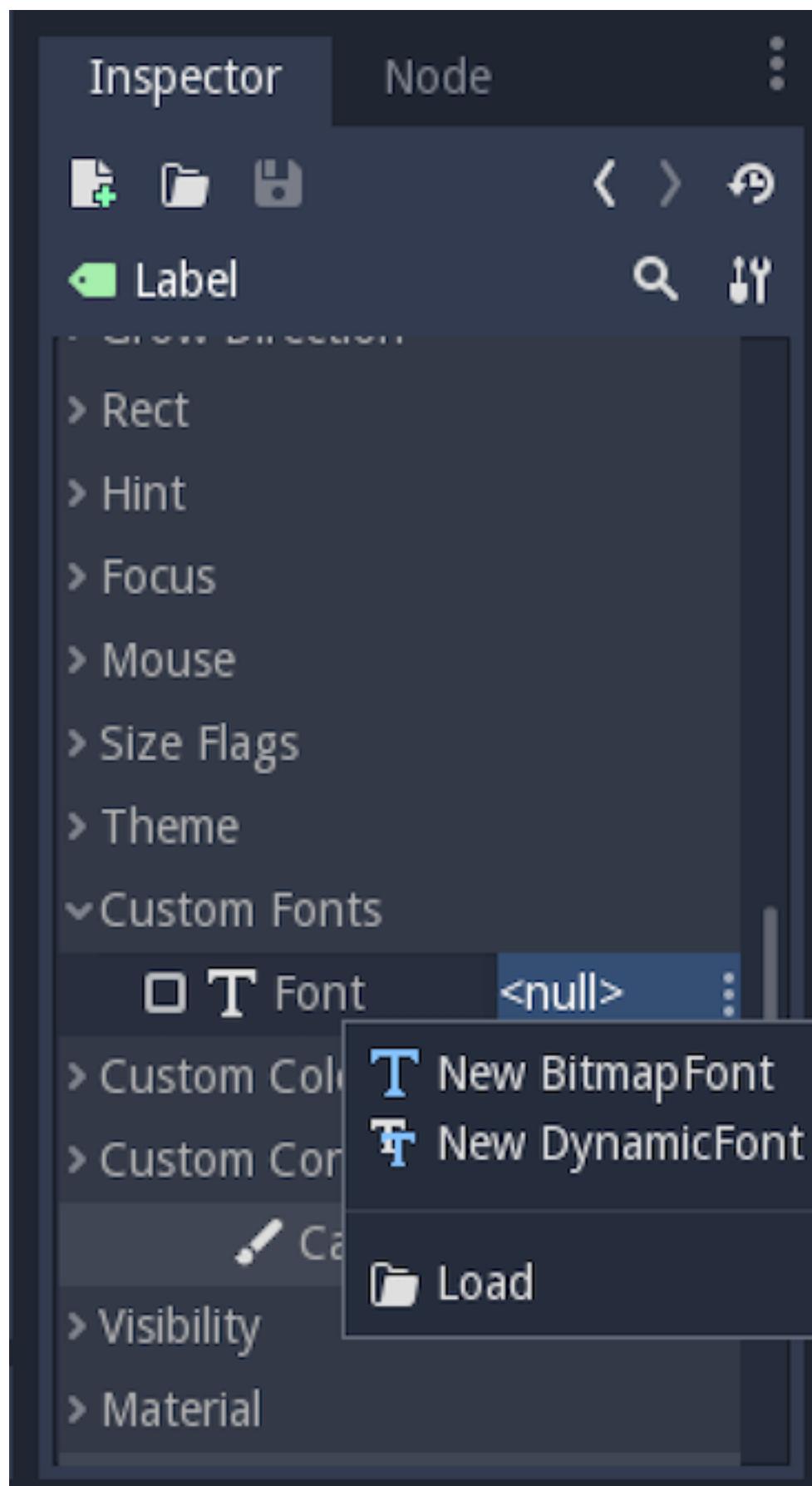
- Margin:
 - Left: -100
 - Top: -200
 - Right: 100
 - Bottom: -100
- Text: Dodge the Creeps!

StartButton

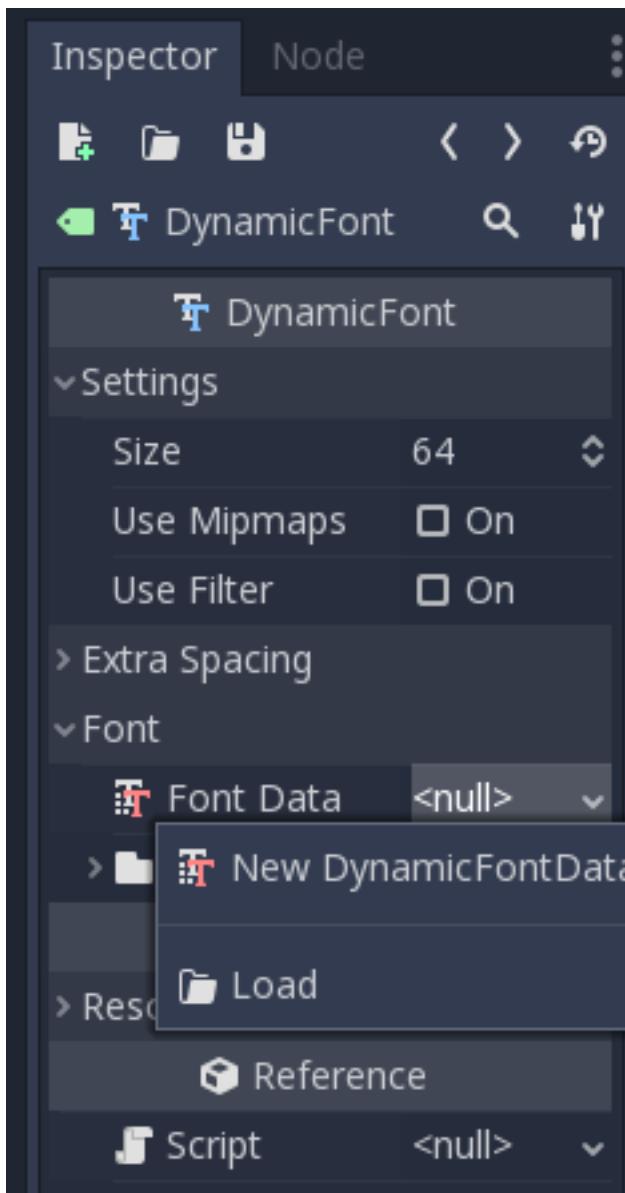
- Layout: “Center”
- Margin:
 - Left: -60
 - Top: 70
 - Right: 60
 - Bottom: 150
- Text: Start

The default font for `Control` nodes is very small and doesn't scale well. There is a font file included in the game assets called “Xolonium-Regular.ttf”. To use this font, do the following for each of the three `Control` nodes:

1. Under “Custom Fonts”, choose “New DynamicFont”



- Click on the “DynamicFont” you just added, and under “Font Data”, choose “Load” and select the “Xolonium-Regular.ttf” file. You must also set the font’s Size. A setting of 64 works well.



Now add this script to the HUD:

```
extends CanvasLayer

signal start_game
```

The `start_game` signal tells the Main node that the button has been pressed.

```
func show_message(text):
    $MessageLabel.text = text
    $MessageLabel.show()
    $MessageTimer.start()
```

This function is called when we want to display a message temporarily, such as “Get Ready”. On the

MessageTimer, set the Wait Time to 2 and set the One Shot property to “On”.

```
func show_game_over():
    show_message("Game Over")
    yield($MessageTimer, "timeout")
    $StartButton.show()
    $MessageLabel.text = "Dodge the\nCreeps!"
    $MessageLabel.show()
```

This function is called when the player loses. It will show “Game Over” for 2 seconds, and then return to the game title and show the “Start” button.

```
func update_score(score):
    $ScoreLabel.text = str(score)
```

This function is called in Main whenever the score changes.

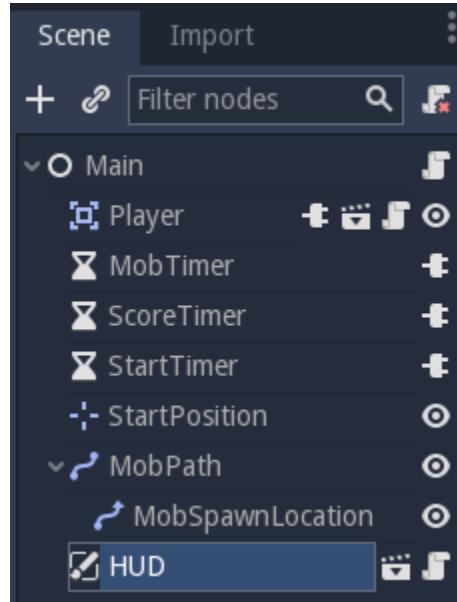
Connect the timeout () signal of MessageTimer and the pressed() signal of StartButton.

```
func _on_StartButton_pressed():
    $StartButton.hide()
    emit_signal("start_game")

func _on_MessageTimer_timeout():
    $MessageLabel.hide()
```

Connecting HUD to Main

Now that we’re done creating the HUD scene, save it and go back to Main. Instance the HUD scene in Main like you did the Player scene, and place it at the bottom of tree. The full tree should look like this, so make sure you didn’t miss anything:



Now we need to connect the HUD functionality to our Main script. This requires a few additions to the Main scene:

In the Node tab, connect the HUD’s start_game signal to the new_game () function.

In new_game (), update the score display and show the “Get Ready” message:

```
$HUD.update_score(score)
$HUD.show_message("Get Ready")
```

In `game_over()` we need to call the corresponding `HUD` function:

```
$HUD.show_game_over()
```

Finally, add this to `_on_ScoreTimer_timeout()` to keep the display in sync with the changing score:

```
$HUD.update_score(score)
```

Now you're ready to play! Click the "Play the Project" button. You will be asked to select a main scene, so choose `Main.tscn`.

2.7.7 Finishing Up

We've now completed all the functionality for our game. Below are some remaining steps to add a bit more "juice" and improve the game experience. Feel free to expand the gameplay with your own ideas.

Background

The default gray background is not very appealing, so let's change its color. One way to do this is to use a `ColorRect` node. Make it the first node under `Main` so that it will be drawn behind the other nodes. `ColorRect` only has one property: `Color`. Choose a color you like and drag the size of the `ColorRect` so that it covers the screen.

You can also add a background image, if you have one, by using a `Sprite` node.

Sound Effects

Sound and music can be the single most effective way to add appeal to the game experience. In your game assets folder, you have two sound files: "House In a Forest Loop.ogg", for background music, and "gameover.wav" for when the player loses.

Add two `AudioStreamPlayer` nodes as children of `Main`. Name one of them `Music` and the other `DeathSound`. On each one, click on the `Stream` property, select "Load" and choose the corresponding audio file.

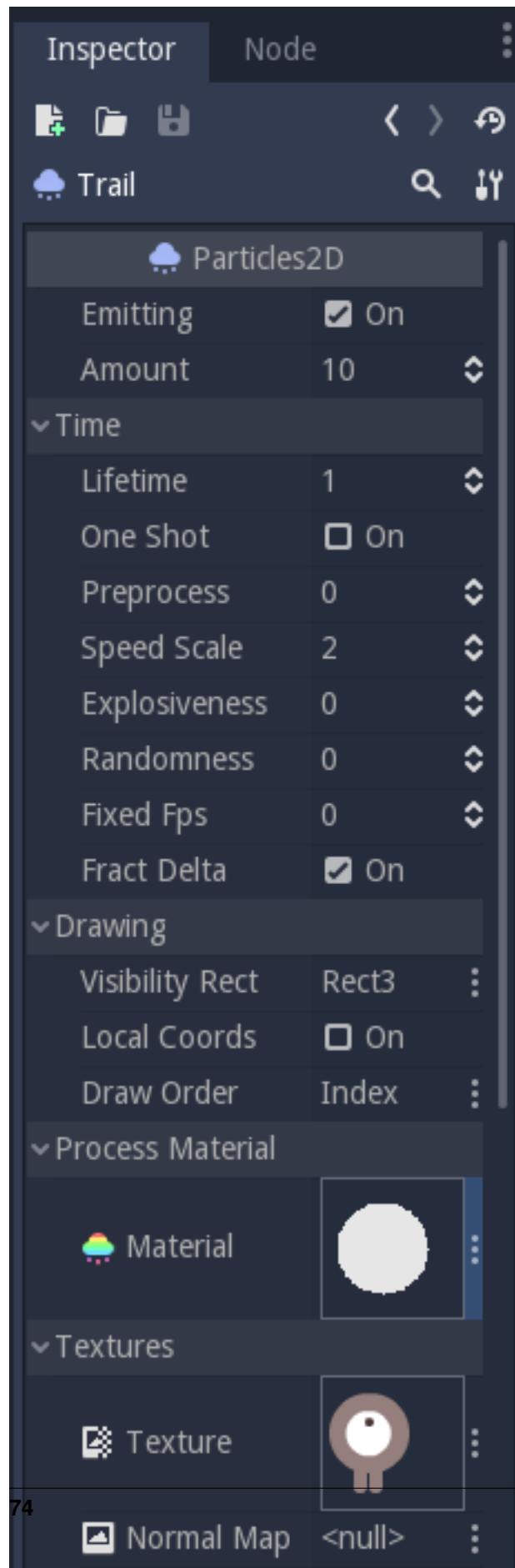
To play the music, add `$Music.play()` in the `new_game()` function and `$Music.stop()` in the `game_over()` function.

Finally, add `$DeathSound.play()` in the `game_over()` function as well.

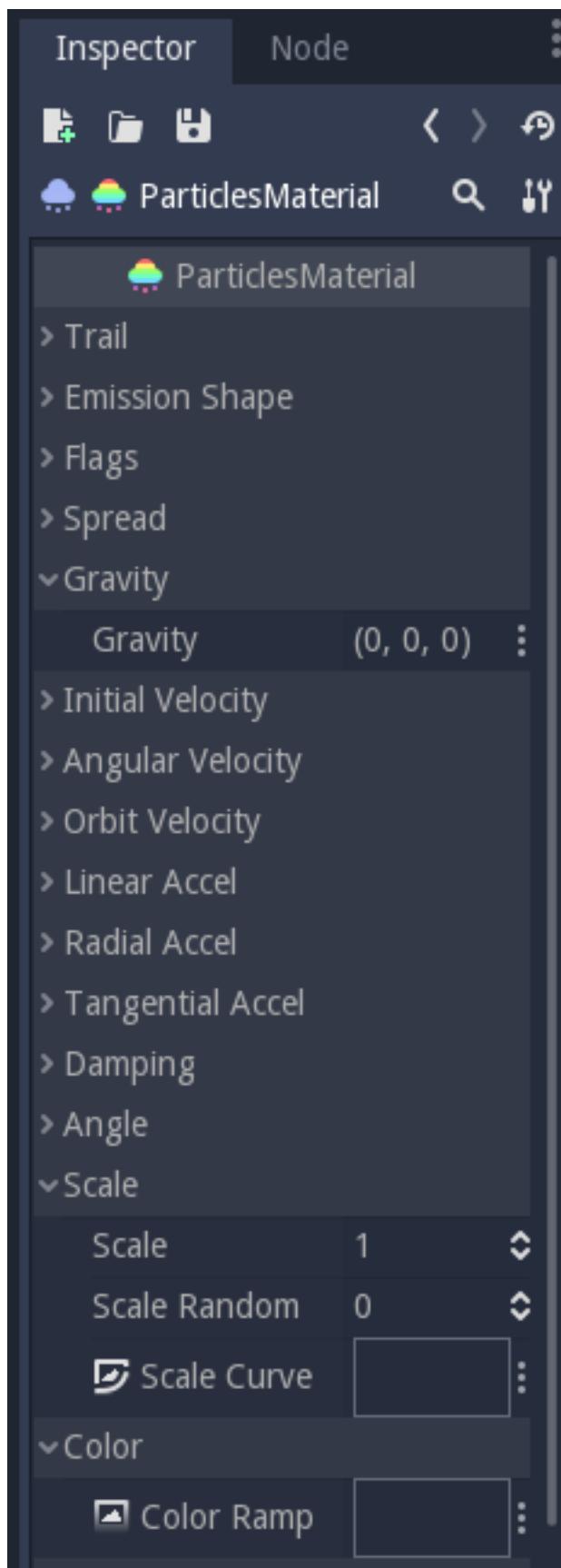
Particles

For one last bit of visual appeal, let's add a trail effect to the player's movement. Choose your `Player` scene and add a `Particles2D` node named `Trail`.

There are a very large number of properties to choose from when configuring particles. Feel free to experiment and create different effects. For the effect in the example, use the following settings:

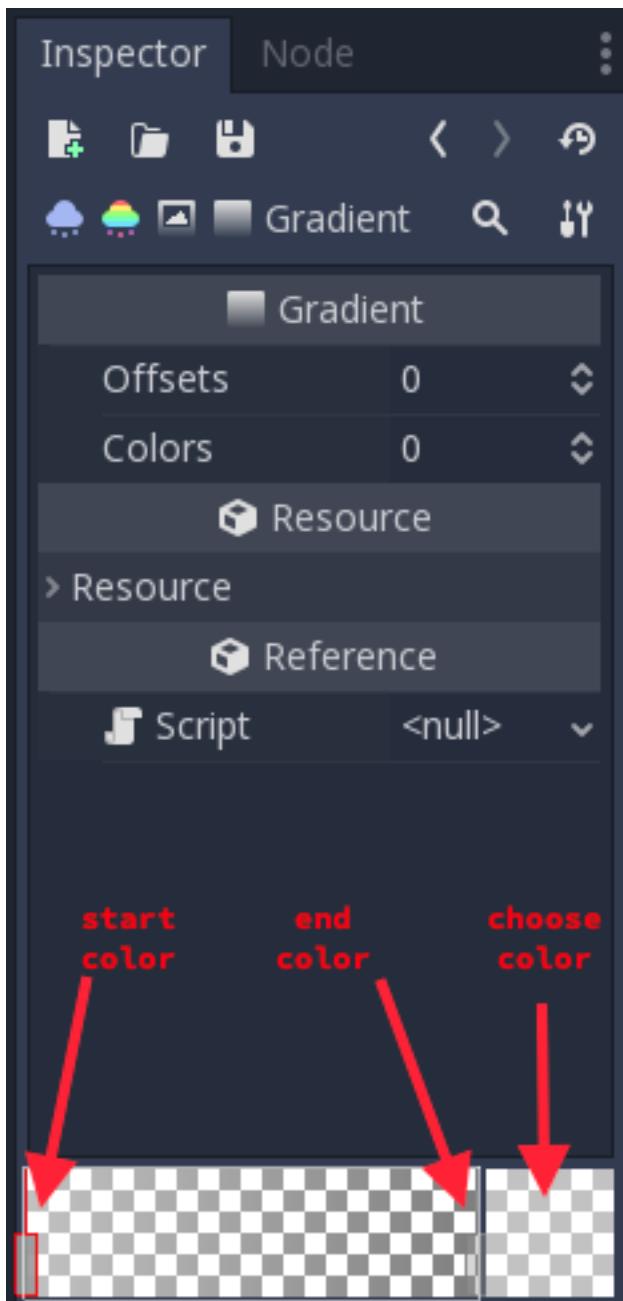


You also need to create a Material by clicking on <null> and then “New ParticlesMaterial”. The settings for that are below:



To make the gradient for the “Color Ramp” setting, we want a gradient taking the alpha (transparency) of the sprite from 0.5 (semi-transparent) to 0.0 (fully transparent).

Click “New GradientTexture”, then under “Gradient”, click “New Gradient”. You’ll see a window like this:



The left and right boxes represent the start and end colors. Click on each and then click the large square on the right to choose the color. For the first color, set the A (alpha) value to around halfway. For the second, set it all the way to 0.

See also:

See *Particles2D* for more details on using particle effects.

2.7.8 Project Files

You can find a completed version of this project here: https://github.com/kidscancode/Godot3_dodge/releases

2.8 Godot's design philosophy

Now you've gotten your hands wet, let's talk about Godot's design.

Every game engine is different and fits different needs. They do not only offer a wide range of features: each engine's design is unique. This leads to different workflows and different ways to reason about your games' structure. This all stems from their design philosophy.

This page is here to **help you understand how Godot works**, starting from some of its core pillars. It is not a list of available features or an engine comparison. To know whether any engine can be a good fit for your project or not, you need to both try it out for yourself and understand its design and limitations.

Please watch [Discover Godot 3, the Free game engine](#) if you're looking for an overview of the engine's features.

2.8.1 Object-oriented design and composition

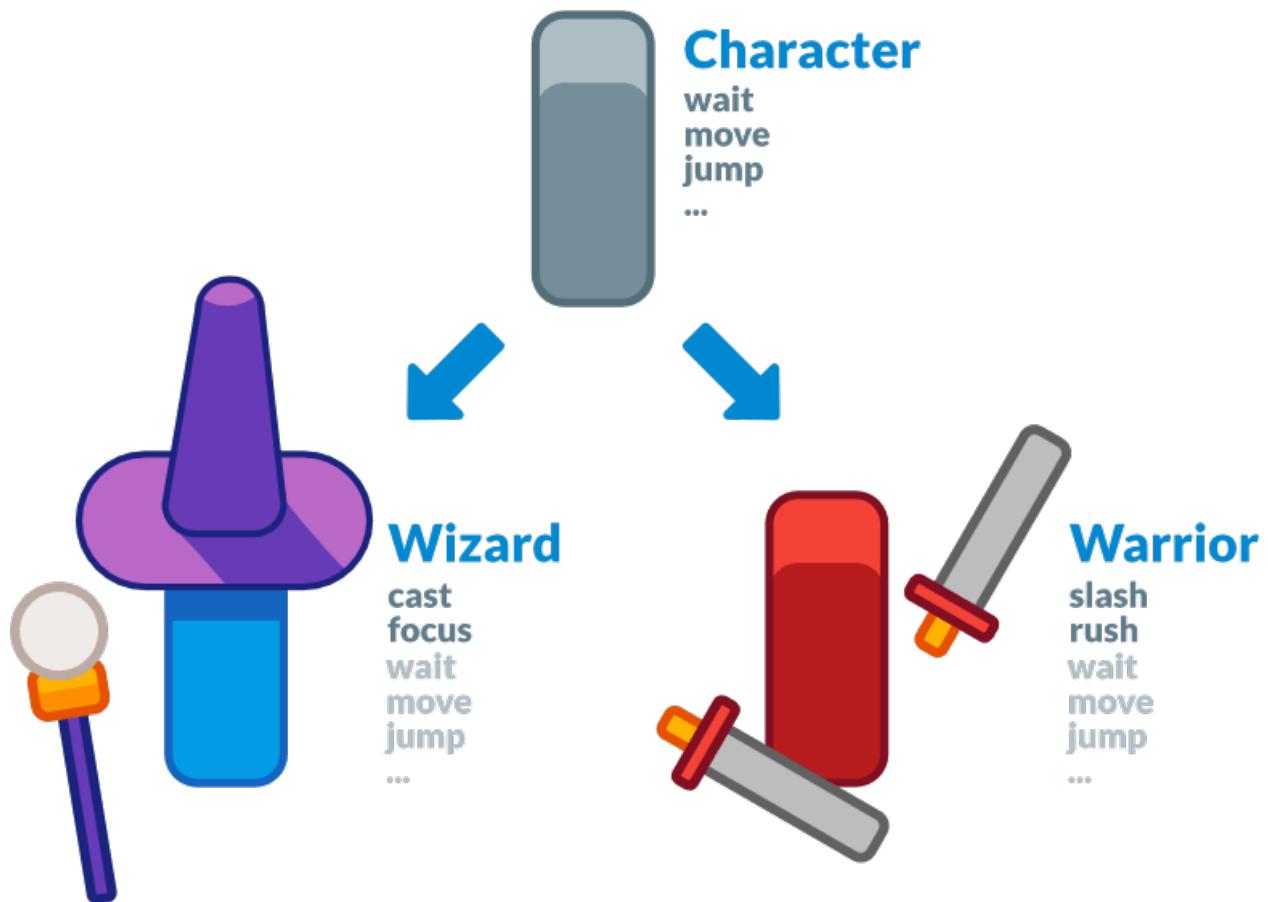
Godot embraces object-oriented design at its core with its **flexible scene system and Node hierarchy**. It tries to stay away from strict programming patterns to offer an intuitive way to structure your game.

For one, Godot lets you **compose or aggregate** scenes. It's like nested prefabs: you can create a BlinkingLight scene and a BrokenLantern scene that uses the BlinkingLight. Then, create a city filled with BrokenLanterns. Change the BlinkingLight's color, save, and all the BrokenLanterns in the city will update instantly.

On top of that, you can **inherit** from any scene.

A Godot scene could be a Weapon, a Character, an Item, a Door, a Level, part of a level... anything you'd like. It works like a class in pure code except you're free to design it using the editor, using only the code, or mixing and matching the two.

It's different from prefabs you find in several 3d engines as you can then inherit from and extend those scenes. You may create a Magician that extends your Character. Modify the Character in the editor and the Magician will update as well. It helps you build your projects so that their structure matches the game's design.



Also note that Godot offers many different types of objects called nodes, each with a specific purpose. Nodes are part of a tree and always inherit from their parents up to the Node class. Although the engine does feature some components like collision shapes, they're the exception, not the norm.

The screenshot shows the Godot Editor's Class Browser with the following tree structure:

- Node (selected)
- CanvasItem
- Node2D
 - AnimatedSprite
 - Sprite

Sprite is a Node2D, a CanvasItem and a Node. It has all the properties and features of its three parent classes, like transforms or the ability to draw custom shapes and render with a custom shader.

2.8.2 All-inclusive package

Godot tries to provide its own tools to answer all of the most common needs. It has a dedicated scripting workspace, an animation editor, a tilemap editor, a shader editor, a debugger, a profiler, hot-reload locally and on remote devices, etc.

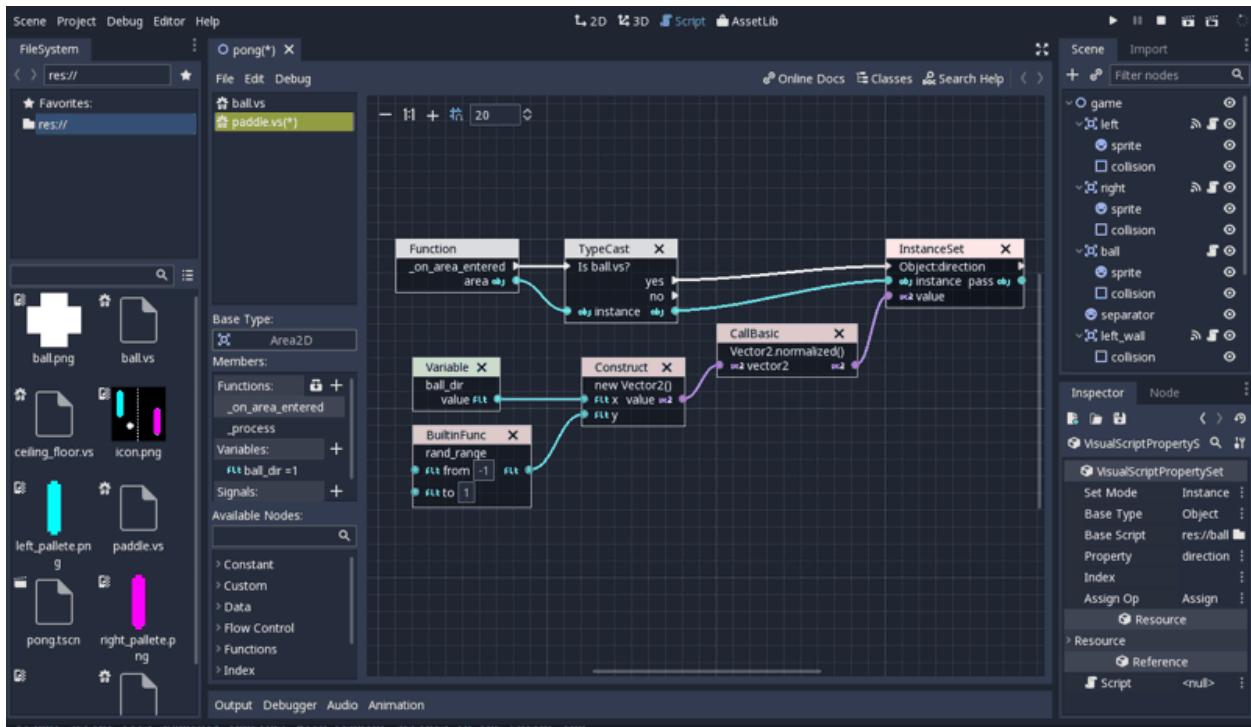


The goal is to offer a full package to create games and a continuous user experience. You can still work with external programs as long as there is an import plugin for it or you can create one, like the [Tiled map editor importer](#).

That is also partly why Godot offers its own programming languages GDscript and VisualScript, along with C#. They're designed for the needs of game developers, game designers, and they're tightly integrated in the engine and the editor.

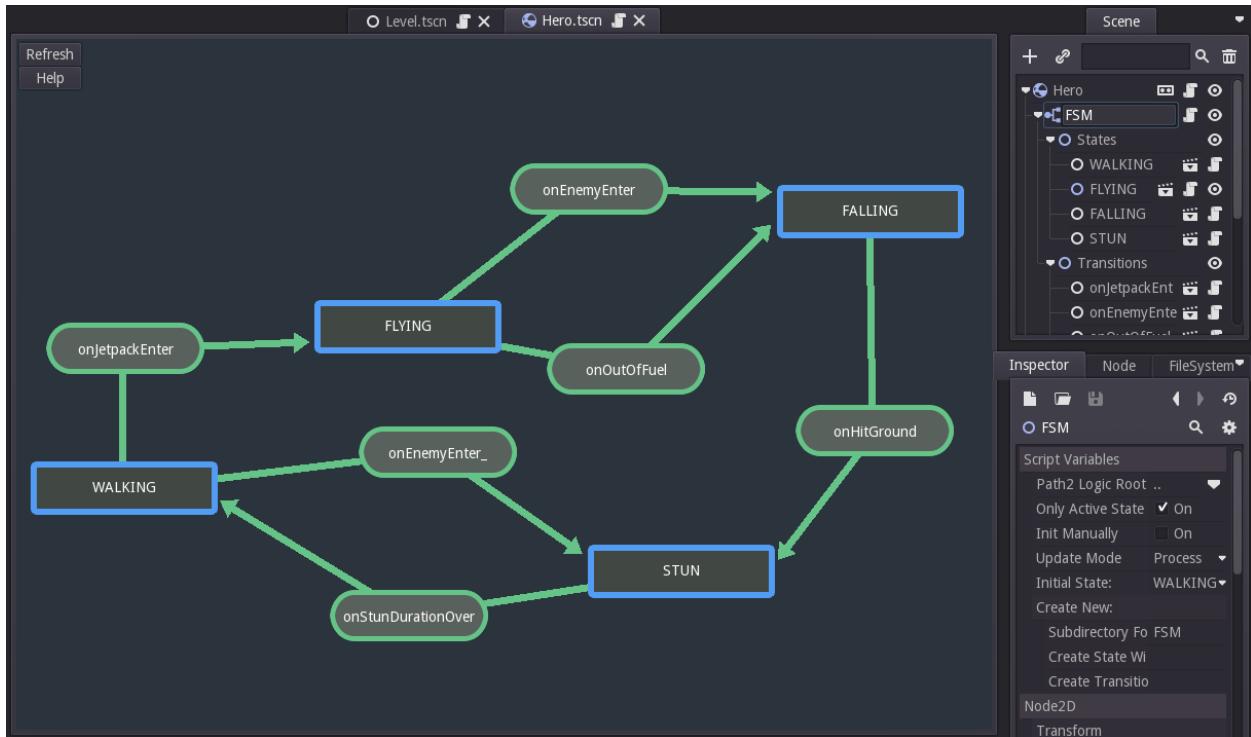
GDscript lets you write simple code using a Python-like syntax, yet it detects types and offers a static-language's quality of auto-completion. It's also optimized for gameplay code with built-in types like Vectors, Colors, etc.

Note that with GDNative, you can write high performance code using compiled languages like C, C++, Rust or Python (Cython compiler) without recompiling the engine.



VisualScript is a node-based programming language that integrates well in the editor. You can drag and drop nodes or resources into the graph to create new code blocks.

Note that if this is true for 2d at the time of writing, the 3d workspace still doesn't feature as many tools. You'll need external programs or add-ons to edit terrains, animate complex characters, etc. Godot provides a complete API to extend the editor's functionality using game code. See *The Godot editor is a Godot game* below.



A State Machine editor plugin in Godot 2 by kubecz3k. It lets you manage states and transitions visually

2.8.3 Open source

Godot offers a fully open source code-base under the **MIT license**. This means all the technologies that ship with it have to be Free as well. For the most part, they're coded from the ground up by contributors.

Anyone can plug in proprietary tools for the needs of their projects. They just won't ship with the engine. This may include NVidia PhysX, Google Admob or an FBX file importer. Any of these can come as third-party plugins instead.

On the other hand, an open codebase means you can **learn from and extend the engine** to your heart's content. You can also debug games easily as Godot will print errors with a stack trace, even if they come from the engine itself.

Note: This **does not affect the work you do with Godot** in any way: there's no strings attached to the engine or anything you make with it.

2.8.4 Community-driven

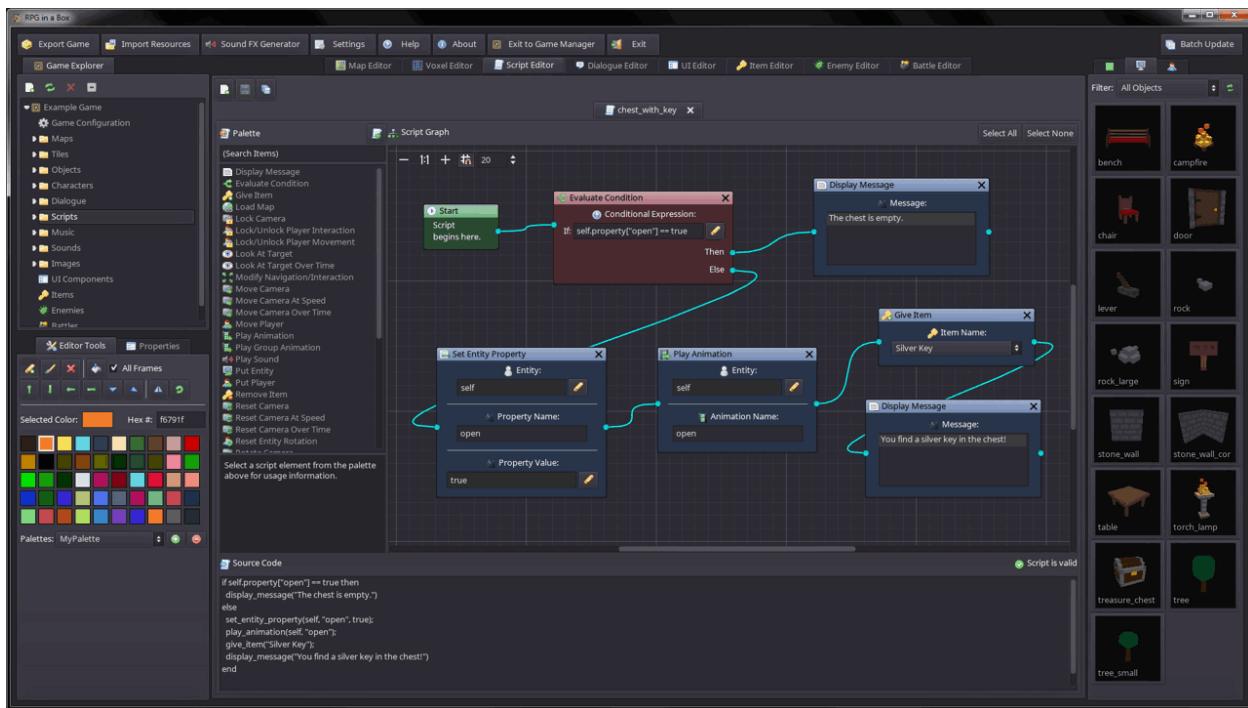
Godot is made by its community, for the community and all game creators out there. It's the needs of the users and open discussions that drive the core updates. New features from the core developers often focus on what will benefit the most users first.

That said, although a handful of core developers work on it full-time, the project has over 500 contributors at the time of writing. Benevolent programmers work on features they may need themselves so you'll see improvements in all corners of the engine at the same time in every major release.

2.8.5 The Godot editor is a Godot game

The Godot editor runs on the game engine. It uses the engine's own UI system, it can hot reload code and scenes when you test your projects, or run game code in the editor. This means you can **use the same code** and scenes for your games or **to build plugins and extend the editor**.

This leads to a reliable and flexible UI system as it powers the editor itself. With the `tool` keyword, you can run any game code in the editor.



RPG in a Box is a voxel RPG editor made in Godot 2. It uses Godot's UI tools for its node-based programming system and for the rest of the interface.

Put the `tool` keyword at the top of any GDscript file and it will run in the editor. This lets you create plugins like custom level editors, import and export plugins or scripts with the same nodes and API you use in your projects.

2.8.6 Separate 2d and 3d engines

Godot offers dedicated 2d and 3d rendering engines. As a result **the base unit for 2d scenes is pixels**. Even though the engines are separate, you can render 2d in 3d, 3d in 2d, and overlay 2d sprites and interface over your 3d world.

2.9 Design interfaces with the Control nodes

Computer displays, mobile phones, and TV screen come in all shapes and sizes. To ship a game, you'll need to support different screen ratios and resolutions. It can be hard to build responsive interfaces that adapt to all platforms. Thankfully, Godot comes with robust tools to design and manage responsive User Interface. To design your UI, you'll use the Control nodes. These are the nodes with green icons in the editor. There are dozens of them, to create anything from life bars to complex applications. Godot's entire editor and plugins use these nodes.

This guide will get you started with UI design. You will learn:

- The five most useful control nodes to build your games' interface
- How to work with the anchor of UI elements
- How to efficiently place and arrange your user interface using containers
- The five most common containers

To learn how to control the interface and connect it to other scripts, read *Build your first game UI in Godot*.

Only use Control nodes when you design your interfaces. They have unique properties that allow them to work with one another. Other nodes like Node2D, Sprite, etc. will not work. You can still use some nodes that work with others

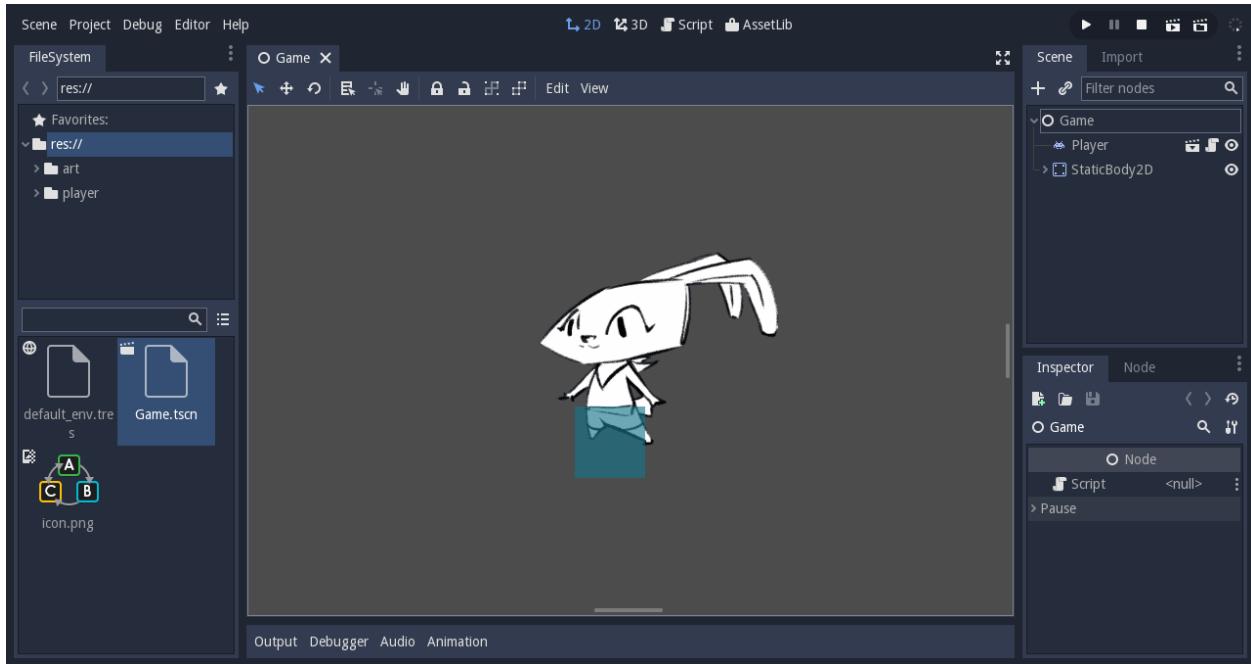


Fig. 2.1: Godot's editor is made with the engine's UI framework

like the AnimationPlayer, Tween or the StreamPlayer. Control nodes are CanvasItems like Node2D, so you can apply shaders to them.

All control nodes share the same main properties:

1. Anchor
2. Bounding rectangle
3. Focus and focus neighbour
4. Size flags
5. Margin
6. The optional UI theme

Once you understand the basics of the Control node, it will take you less time to learn all the nodes that derive from it.

2.9.1 The 5 most common UI elements

Godot ships with dozens of Control nodes. A lot of them are here to help you build editor plugins and applications. To learn more about them, check the guide about [Advanced UI nodes and Themes](#).

For most games, you'll only need five types of UI elements, and a few Containers. These five Control nodes are:

1. Label: for displaying text
2. TextureRect: used mostly for backgrounds, or everything that should be a static image
3. TextureProgress: for lifebars, loading bars, horizontal, vertical or radial
4. NinePatchRect: for scalable panels
5. TextureButton: to create buttons

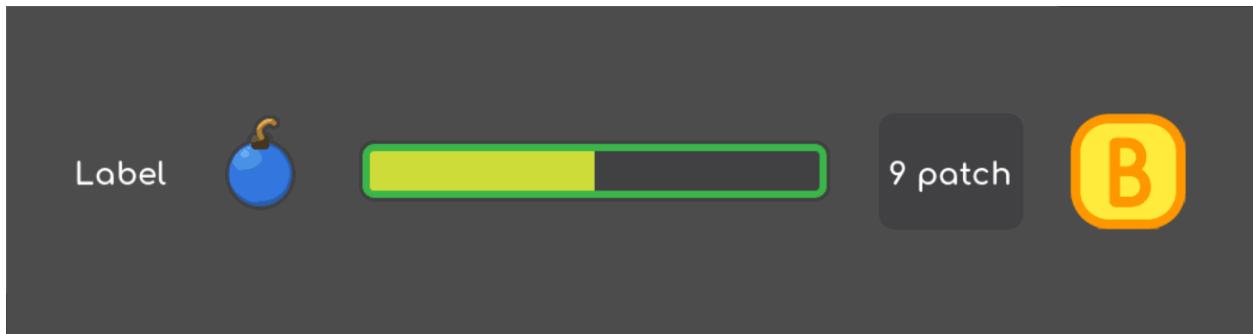


Fig. 2.2: The 5 most common Control nodes for UI design

TextureRect

TextureRect displays a texture or image inside a UI. It seems similar to the **Sprite** node but it offers multiple scaling modes. Set the **Stretch Mode** property to change its behaviour:

- **Tile** makes the texture repeat, but it won't scale
- The default, **Scale** and **Expand (compat)**, lets you scale the texture up, but it forces a minimal size
- **Keep** and **Keep Centered** force the texture to remain at its original size, in the top left corner or the center of the frame respectively

As with **Sprite** nodes, you can modulate the **TextureRect**'s colour. Click the **Modulate** property and use the color picker.



Fig. 2.3: TextureRect modulated with a red color

TextureButton

TextureButton is like **TextureRect**, except it has 5 texture slots: one for each of the button's states. Most of the time, you'll use the **Normal**, **Pressed**, and **Hover** textures. **Focused** is useful if your interface listens to the keyboard's input. The sixth image slot, the **Click Mask**, lets you define the clickable area using a 2-bit, pure black and white image.

In the **Base Button** section, you'll find a few checkboxes that change how the button behaves. **When Toggle Mode** is on, the button will toggle between active and normal states when you press it. **Disabled** makes it disabled by default, in which case it will use the **Disabled** texture. **TextureButton** shares a few properties with the **texture frame**: it has a **modulate** property, to change its color, and **Resize** and **Stretch** modes to change its scale behavior.

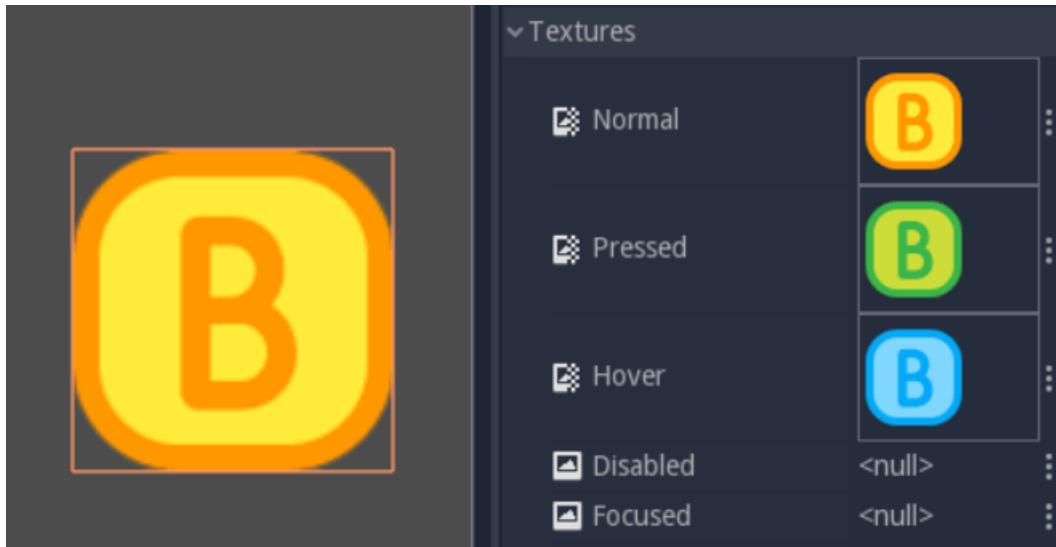


Fig. 2.4: TextureButton and its 5 texture slots

TextureProgress

TextureProgress layers up to 3 sprites to create a progress bar. The Under and Over textures sandwich the Progress one, which displays the bar's value.

The Mode property controls the direction in which the bar grows: horizontally, vertically, or radially. If you set it to radial, the Initial Angle and Fill Degrees properties let you limit the range of the gauge.

To animate the bar, you'll want to look at the Range section. Set the Min and Max properties to define the range of the gauge. For instance, to represent a character's life, you'll want to set Min to 0, and Max to the character's maximum life. Change the Value property to update the bar. If you leave the Min and Max values to the default of 1 and 100, and set the Value property to 40, 40% of the Progress texture will show up, and 60% of it will stay hidden.

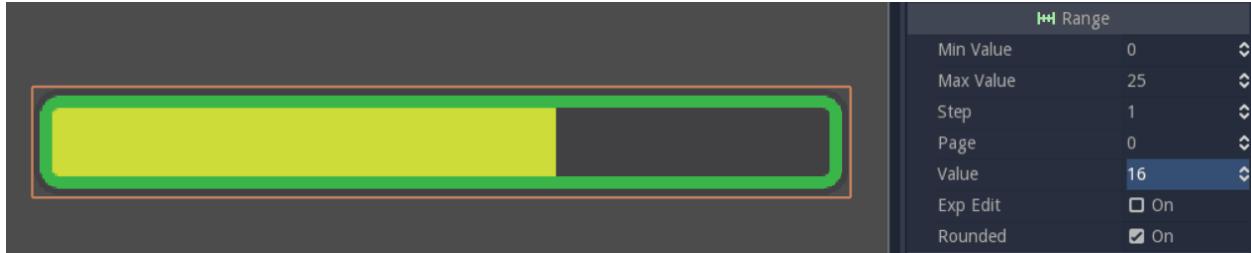


Fig. 2.5: TextureProgress bar, two thirds filled

Label

Label prints text to the screen. You'll find all its properties in the Label section, in the Inspector. Write the text in the Text property, and check Autowrap if you want it to respect the textbox's size. If Autowrap is off, you won't be able to scale the node. You can align the text horizontally and vertically with Align and Valign respectively.

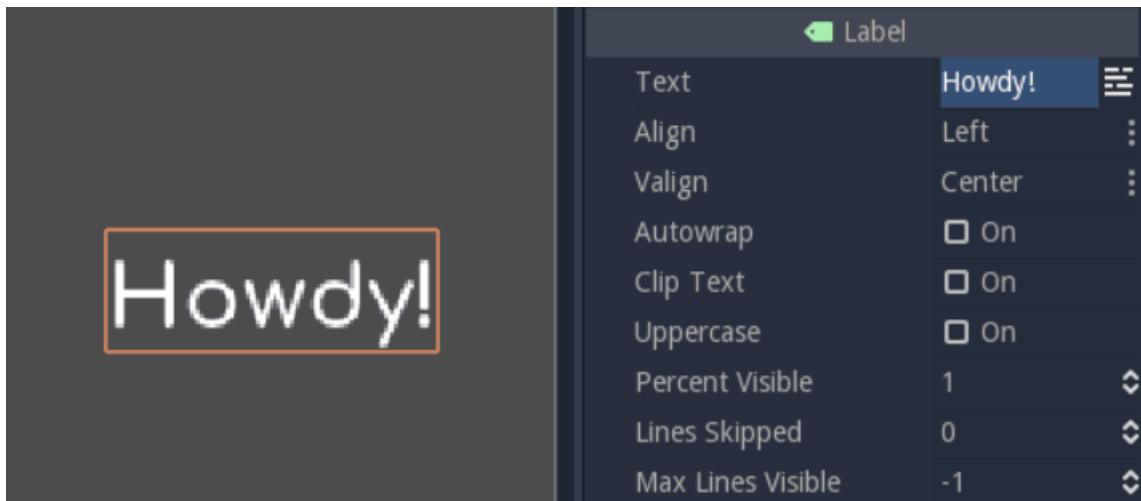


Fig. 2.6: Picture of a Label

NinePatchRect

NinePatchRect takes a texture split in 3 rows and 3 columns. The center and the sides tile when you scale the texture, but it never scales the corners. It is very useful to build panels, dialogue boxes and scalable backgrounds for your UI.



Fig. 2.7: NinePatchRect scaled with the min_size property

2.9.2 There are two workflows to build responsive UIs

There are two workflows to build scalable and flexible interfaces in Godot:

1. You have many container nodes at your disposal that scale and place UI elements for you. They take control over their children.
2. On the other side, you have the layout menu. It helps you to anchor, place and resize a UI element within its parent.

The two approaches are not always compatible. Because a container controls its children, you cannot use the layout menu on them. Each container has a specific effect so you may need to nest several of them to get a working interface. With the layout approach you work from the bottom up, on the children. As you don't insert extra containers in the scene it can make for cleaner hierarchies, but it's harder to arrange items in a row, column, grid, etc.

As you create UIs for your games and tools, you'll develop a sense for what fits best in each situation.

2.9.3 Place UI elements precisely with anchors

Control nodes have a position and size, but they also have anchors and margins. Anchors define the origin, or the reference point, for the Left, Top, Right and Bottom edges of the node. Change any of the 4 anchors to change the reference point of the margins.

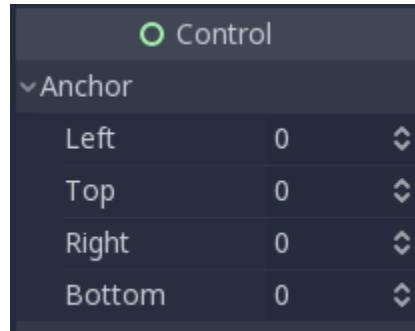


Fig. 2.8: The anchor property

How to change the anchor

Like any properties, you can edit the 4 anchor points in the Inspector, but this is not the most convenient way. When you select a control node, the layout menu appears above the viewport, in the toolbar. It gives you a list of icons to set all 4 anchors with a single click, instead of using the inspector's 4 properties. The layout menu will only show up when you select a control node.

Anchors are relative to the parent container

Each anchor is a value between 0 and 1. For the left and top anchors, a value of 0 means that without any margin, the node's edges will align with the left and top edges of its parent. For the right and bottom edges, a value of 1 means they'll align with the parent container's right and bottom edges. On the other hand, margins represent a distance to the anchor position in pixels, while anchors are relative to the parent container's size.

Margins change with the anchor

Margins update automatically when you move or resize a control node. They represent the distance from the control node's edges to its anchor, which is relative to the parent control node or container. That's why your control nodes should always be inside a container, as we'll see in a moment. If there's no parent, the margins will be relative to the node's own bounding Rectangle, set in the Rect section, in the inspector.

Try to change the anchors or nest your Control nodes inside Containers: the margins will update. You'll rarely need to edit the margins manually. Always try to find a container to help you first; Godot comes with nodes to solve all the common cases for you. Need to add space between a lifebar and the border of the screen? Use the MarginContainer. Want to build a vertical menu? Use the VBoxContainer. More on these below.

Use size tags to change how UI elements fill the available space

Every control node has Size Flags. They tell containers how the UI elements should scale. If you add the “Fill” flag to the Horizontal or Vertical property, the node's bounding box will take all the space it can, but it'll respect its siblings and retain its size. If there are 3 TextureFrame nodes in an HBoxContainer, with the “Fill” flags on both axes,

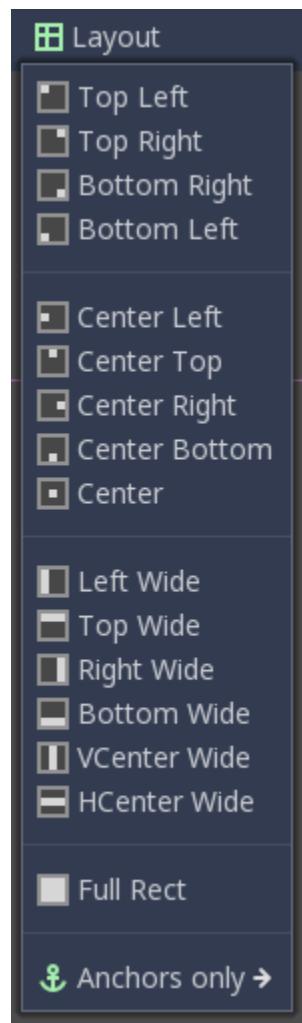


Fig. 2.9: The layout menu in the viewport

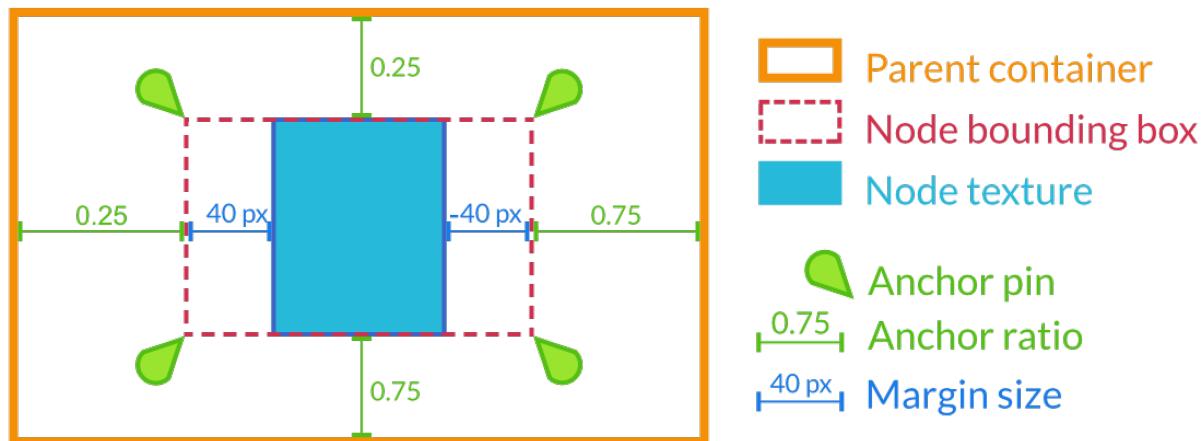


Fig. 2.10: Margins are relative to the anchor position, which is relative to the anchors. In practice, you'll often let the container update margins for you

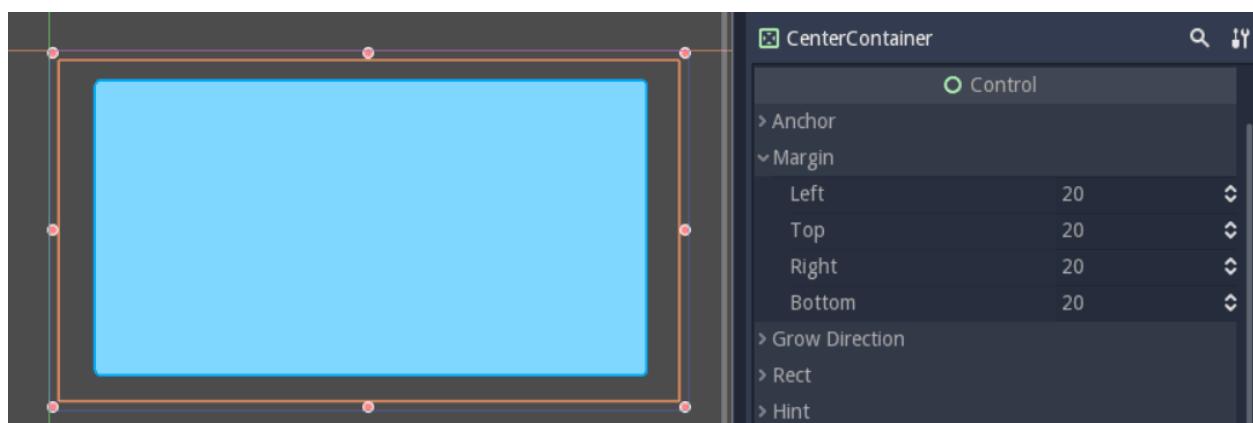


Fig. 2.11: Margins on a CenterContainer set to the “Full Rect” anchor

they'll each take up to a third of the available space, but no more. The container will take over the node and resize it automatically.

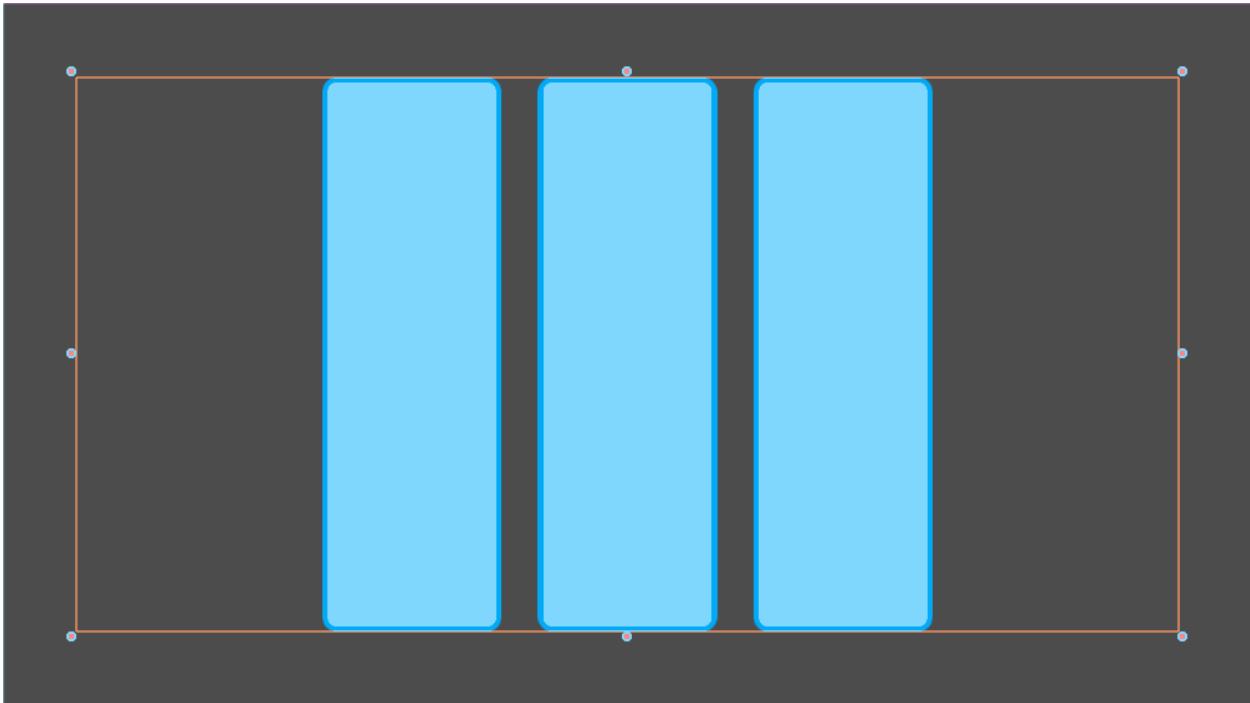


Fig. 2.12: 3 UI elements in an HBoxContainer, they align horizontally

The “Expand” flag lets the UI element take all the space it can, and push against its siblings. Its bounding rectangle will grow against the edges of its parent, or until it's blocked by another UI node.

You'll need some practice to understand the size tags, as their effect can change quite a bit depending on how you set up your interface.

2.9.4 Arrange control nodes automatically with containers

Containers automatically arrange all children Control nodes including other containers in rows, columns, and more. Use them to add padding around your interface or center nodes in their bounding rectangles. All built-in containers update in the editor so you can see the effect instantly.

Containers have a few special properties to control how they arrange UI elements. To change them, navigate down to the Custom Constants section in the Inspector.

The 5 most useful containers

If you build tools, you might need all of the containers. But for most games, a handful will be enough:

- MarginContainer, to add margins around part of the UI
- CenterContainer, to center its children in its bounding box
- VboxContainer and HboxContainer, to arrange UI elements in rows or columns
- GridContainer, to arrange Controls nodes in a grid-like pattern

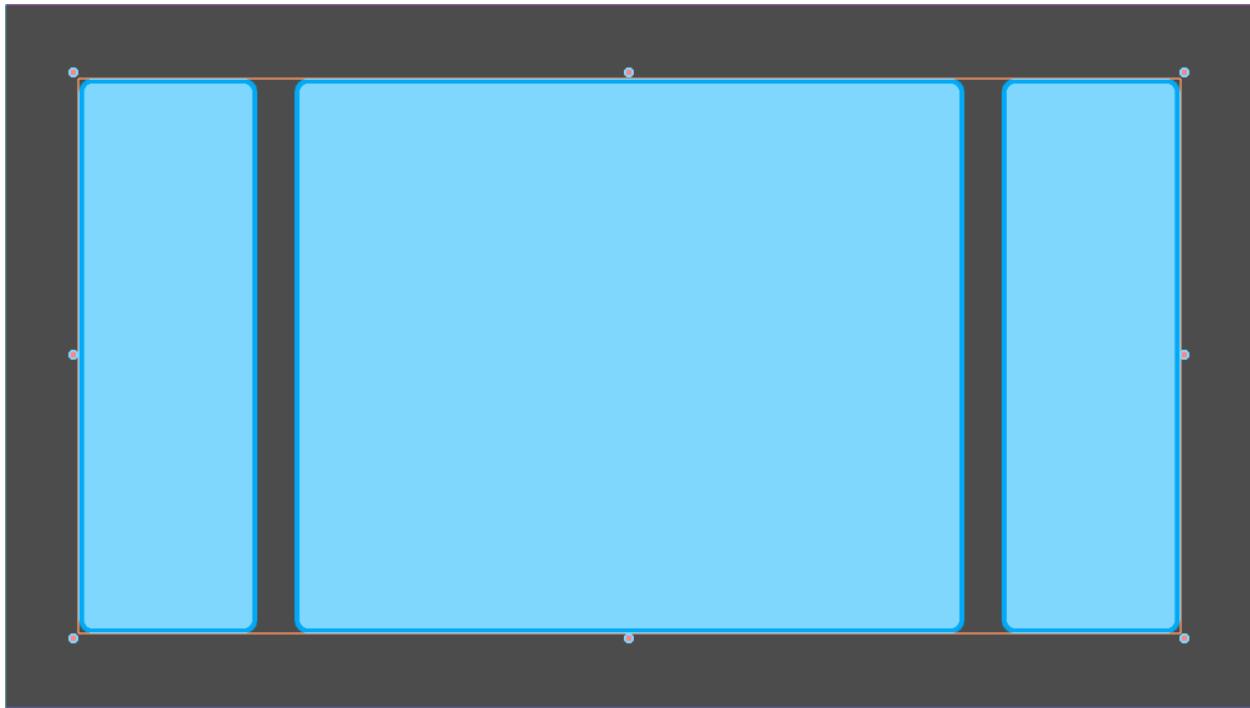


Fig. 2.13: The same example as above, but the left node has the “Expand” size flag

CenterContainer centers all its children inside of its bounding rectangle. It’s one you typically use for title screens, if you want the options to stay in the center of the viewport. As it centers everything, you’ll often want a single container nested inside it. If you use textures and buttons instead, they’ll stack up.

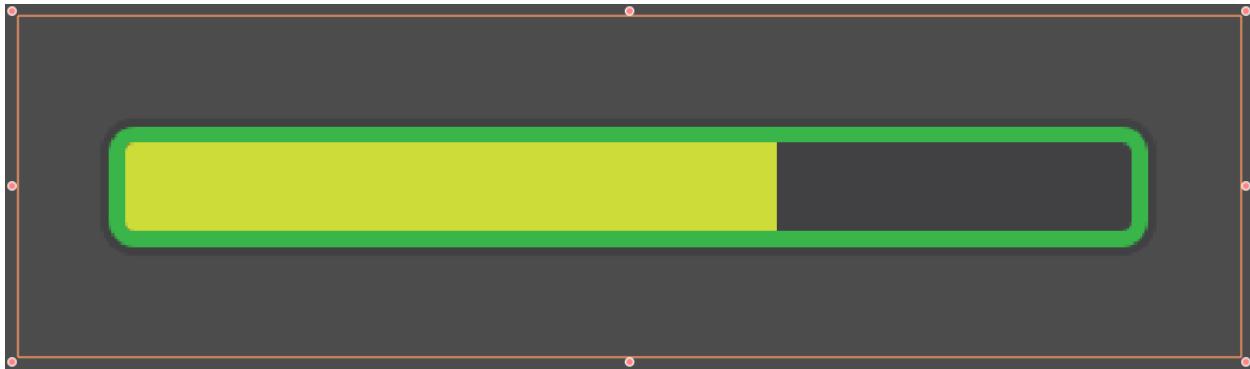


Fig. 2.14: CenterContainer in action. The life bar centers inside its parent container.

The MarginContainer adds a margin on any side of the child nodes. Add a MarginContainer that encompasses the entire viewport to add a separation between the edge of the window and the UI. You can set a margin on the top, left, right, or bottom side of the container. No need to tick the checkbox: click the corresponding value box and type any number. It will activate automatically.

There are two BoxContainers: VBoxContainer and HBoxContainer. You cannot add the BoxContainer node itself, as it is a helper class, but you can use vertical and horizontal box containers. They arrange nodes either in rows or columns. Use them to line up items in a shop, or to build complex grids with rows and columns of different sizes, as you can nest them to your heart’s content.

VBoxContainer automatically arranges its children into a column. It puts them one after the other. If you use the

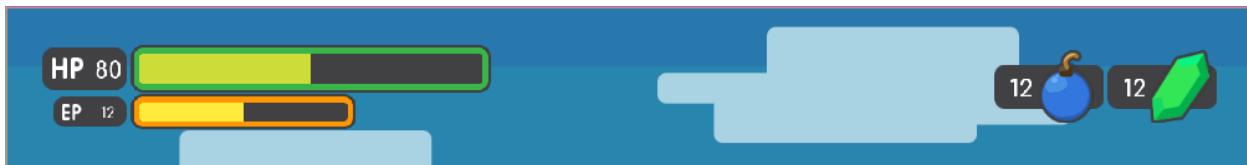


Fig. 2.15: The MarginContainer adds a 40px margin around the Game User Interface

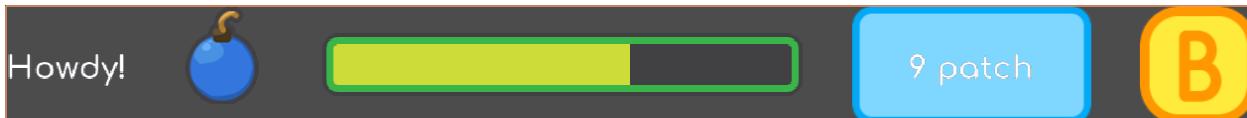


Fig. 2.16: The HBoxContainer horizontally aligns UI elements

separation parameter, it will leave a gap between its children. HBoxContainer arranges UI elements in a row. It's similar to the VBoxContainer, with an extra `add_spacer` method to add a spacer control node before its first child or after its last child, from a script.

The GridContainer lets you arrange UI elements in a grid-like pattern. You can only control the number of columns it has, and it will set the number of rows by itself, based on its children's count. If you have nine children and three columns, you will have $9 \div 3 = 3$ rows. Add three more children and you'll have four rows. In other words, it will create new rows as you add more textures and buttons. Like the box containers, it has two properties to set the vertical and horizontal separation between the rows and columns respectively.

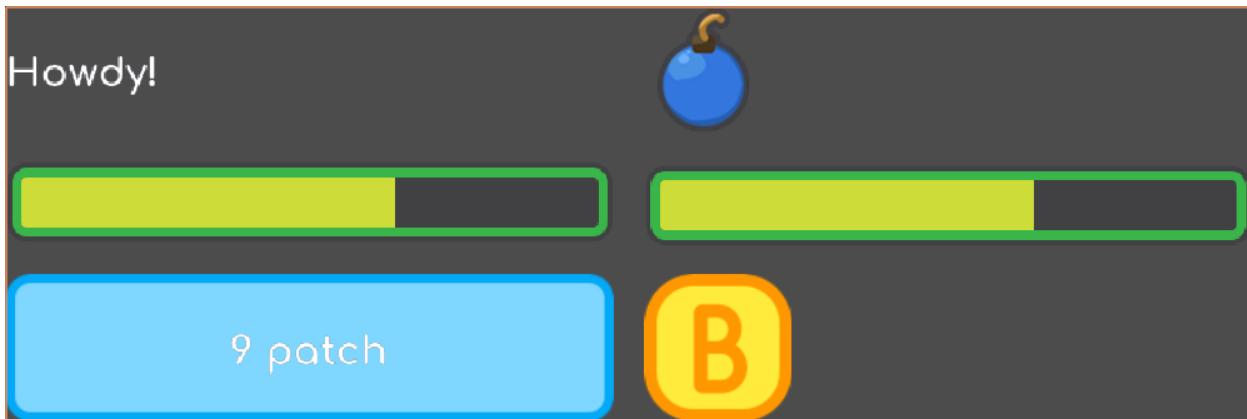


Fig. 2.17: A GridContainer with 2 columns. It sizes each column automatically.

Godot's UI system is complex, and has a lot more to offer. To learn how to design more advanced interface, read [Design advanced UI with other Control nodes](#).

2.10 Design a title screen

In the next two tutorials, you will build two responsive UI (user interface) scenes step-by-step using the engine's UI system:

1. A main menu
2. A game UI with a health bar, energy bar, bomb and money counters

You will learn how to design game UI efficiently, and how to use Godot's Control nodes. This page focuses on the visual part: everything you do from the editor. To learn how to code a life bar, read *Design the GUI*



Fig. 2.18: The GUI you're going to create

Download the project files: `ui_main_menu_design.zip` and extract the archive. Import the `start/` project in Godot to follow this tutorial. The `end/` folder contains the final result.

2.10.1 How to design your game UI

To design a good UI, you want to come up with a rough mockup first: a plain drawing version that focuses on the placement of your UI components, their size, and user interaction. Pen and paper is all you need. You shouldn't use fancy and final graphics at this stage. Then, you only need simple placeholder sprites and you're good to jump into Godot. You want to make sure the players can find their way around the interface using those placeholders.

Placeholder doesn't have to mean ugly, but you should keep the graphics simple and clean. Avoid special effects, animation, and detailed illustration before you had players playtest your UI. Otherwise:

1. The graphics might skew the players' perception of the experience and you'll miss out on valuable feedback
2. If the User Experience doesn't work, you'll have to redo some sprites

Tip: Always try to make the interface work with simple text and boxes first. It's easy to replace the textures later. Professional UX designers often work with plain outlines and boxes in greyscale. When you take colors and fancy visuals away, it's a lot easier to size and place UI elements properly. It helps you refine the design foundation you'll build upon.

There are two ways to design your UI in Godot. You can:

1. Build it all in a single scene, and eventually save some branches as reusable scenes



Fig. 2.19: The UI's rough plan or mockup

2. Build template scenes for reusable components and create specific components that inherit from your base scenes

We will use the first approach, because the first version of your UI may not work as well as you'd like. You're likely to throw parts away and redesign components as you go. When you're sure everything works, it's easy to make some parts reusable, as you'll see below.

Please download `ui_main_menu_design.zip`. It contains all the files you'll need to follow along. You'll find all the sprites in the `start/assets/main_menu` folder.

2.10.2 Design the main menu

Before we jump into the editor, we want to plan how we'll nest containers based on our mockup image.

Break down the UI mockup

Here are my three rules of thumb to find the right containers:

1. Break down the UI into nested boxes, from the largest that contains everything, to the smallest ones, that encompass one widget, like a bar with its label, a panel or a button
2. If there's some padding around an area, use a `MarginContainer`
3. If the elements are arranged in rows or columns, use an `HBoxContainer` or `VBoxContainer`

These rules are enough to get us started, and work well for simple interfaces.

For the main menu, the largest box is the entire game window. There's padding between the edges of the window and the first components: this should be a `MarginContainer`. Then, the screen is split into two columns, so we'll use

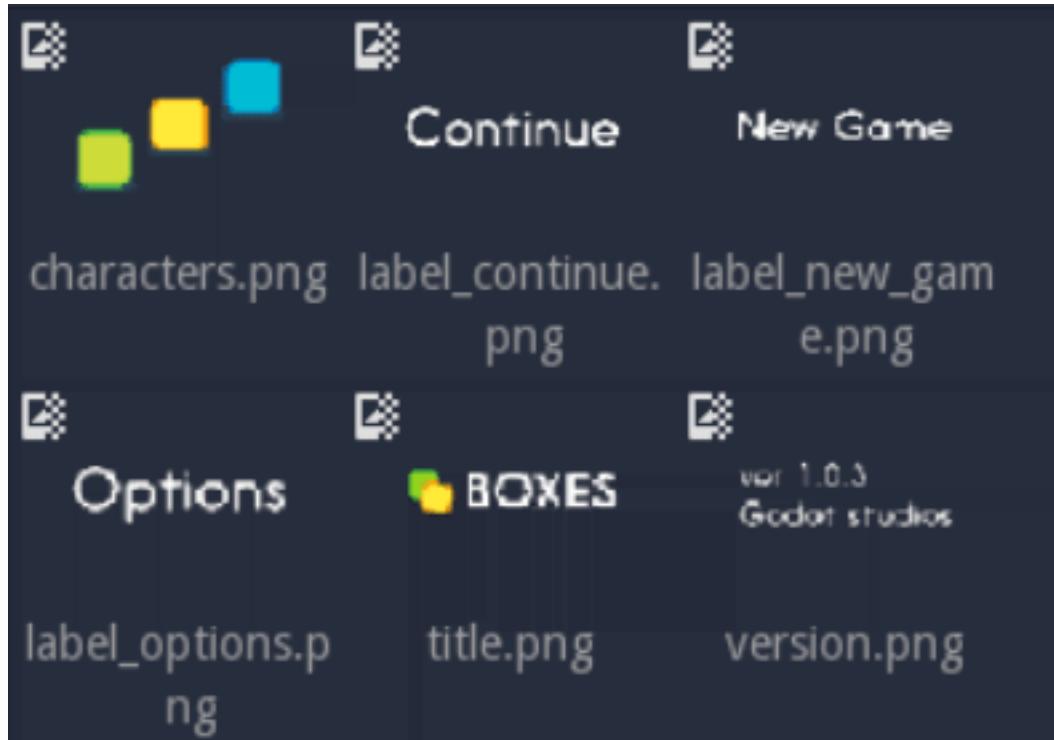


Fig. 2.20: The files you'll find in Godot. The graphics look cleaner than on the rough design, but they're still place-holders

an `HBoxContainer`. In the left column, we'll manage the rows with a `VBoxContainer`. And in the right column, we'll center the illustration with a `CenterContainer`.

Tip: Containers adapt to the window's resolution and width-to-height ratio. Although we could place UI elements by hand, containers are faster, more precise, and **responsive**.

Prepare the Main Menu scene

Let's create the main menu. We'll build it in a single scene. To create an empty scene, click on the Scene menu -> New Scene.

We have to add a root node before we can save the scene. Your UI's root should be the outermost container or element. In this case it's a `MarginContainer`. `MarginContainer` is a good starting point for most interfaces, as you often need padding around the UI. Press `Meta+S` to save the scene to the disk. Name it *MainMenu*.

Select the `MarginContainer` again, and head to the inspector to define the margins' size. Scroll down the Control class, to the Custom Constants section. Unfold it. Set the margins as such:

- Margin Right: *120*
- Margin Top: *80*
- Margin Left: *120*
- Margin Bottom: *80*

We want the container to fit the window. In the Viewport, open the Layout menu and select the last option, Full Rect.

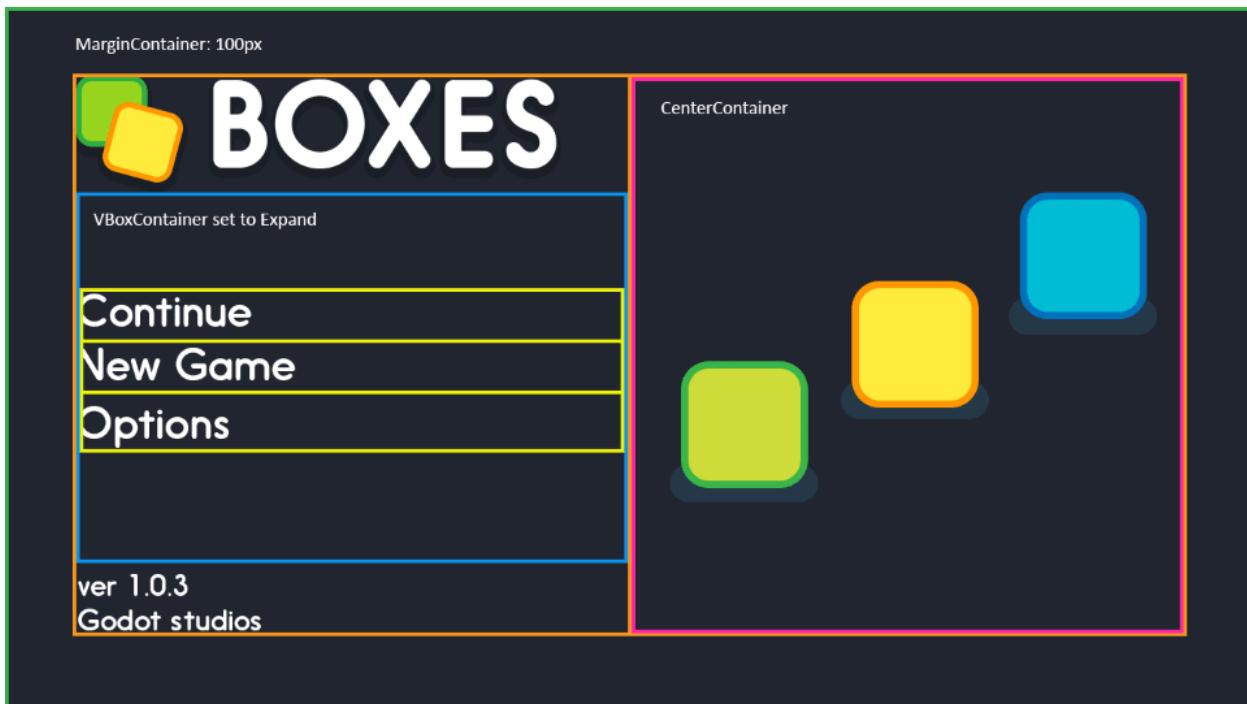


Fig. 2.21: Interface building blocks, broken down using the three rules of thumb

Add the UI sprites

Select the MarginContainer, and create the UI elements as TextureRect nodes. We need:

1. The title, or logo
2. The three text options, as individual nodes
3. The version note
4. And the main menu's illustration

Click the `Add Node` button or press `Meta+A` on your keyboard. Start to type `TextureRect` to find the corresponding node and press enter. With the new node selected, press `Meta+D` five times to create five extra `TextureRect` instances.

Click each of the nodes to select it. In the inspector, click the `...` icon to the right of the `Texture` property, and click on `Load`. A file browser opens and lets you pick a sprite to load into the texture slot.

Repeat the operation for all `TextureRect` nodes. You should have the logo, the illustration, the three menu options and the version note, each as a separate node. Then, double click on each of the nodes in the Inspector to rename them

Note: If you want to support localization in your game, use `Labels` for menu options instead of `TextureRect`.

Add containers to place UI elements automatically

Our main menu has some margin around the edges of the screen. It is split in two parts: on the left, you have the logo and the menu options. On the right, you have the characters. We can use one of two containers to achieve this: `HSplitContainer` or `HBoxContainer`. Split containers split the area into two: a left and a right side or a top

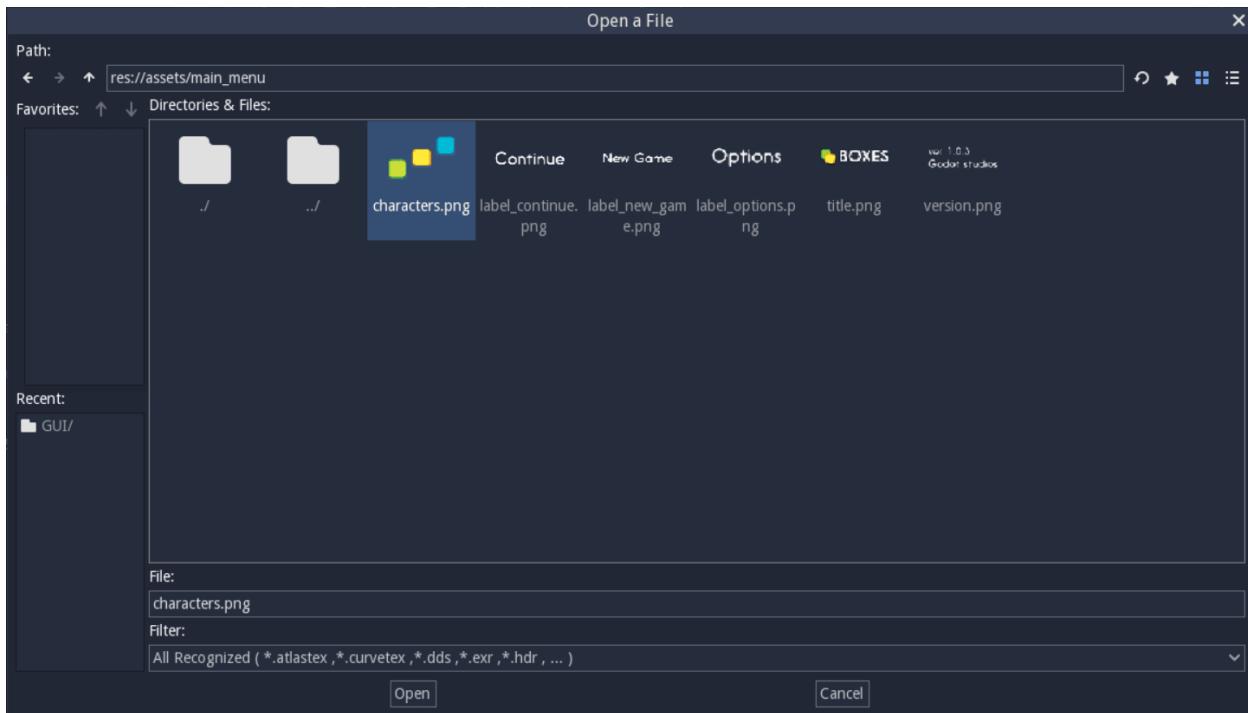


Fig. 2.22: The file browser lets you find and load textures

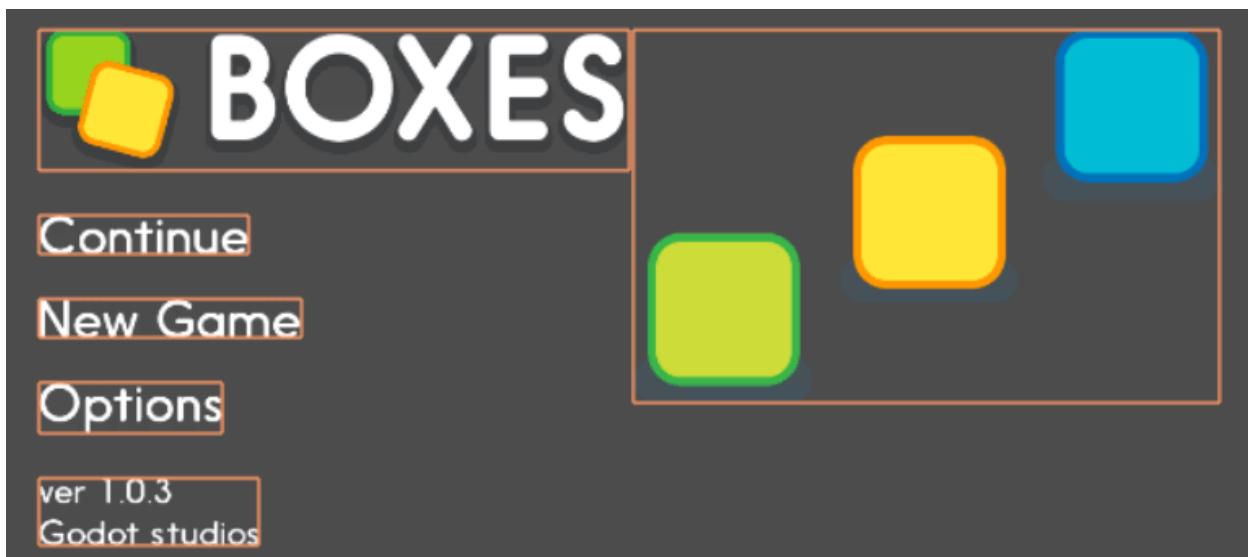


Fig. 2.23: The six nodes, with textures loaded

and a bottom side. They also allow the user to resize the left and right areas using an interactive bar. On the other hand, `HBoxContainer` just splits itself into as many columns as it has children. Although you can deactivate the split container's resize behaviour, I recommend to favour box containers.

Select the `MarginContainer` and add an `HBoxContainer`. Then, we need two containers as children of our `HBoxContainer`: a `VBoxContainer` for the menu options on the left, and a `CenterContainer` for the illustration on the right.

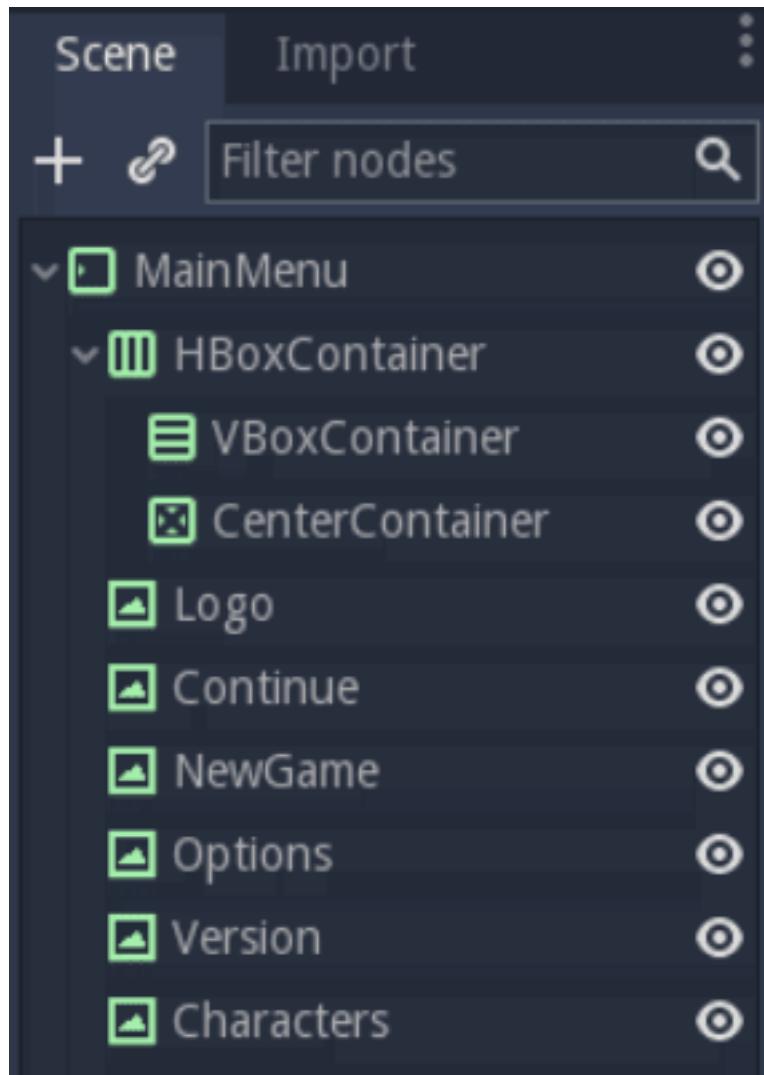


Fig. 2.24: You should have four nested containers, and the `TextureRect` nodes sitting aside from it

In the node tree, select all the `TextureRect` nodes that should go on the left side: the logo, the menu options and the version note. Drag and drop them into the `VBoxContainer`. Then, drag the illustration node into the `CenterContainer`. The nodes should position automatically.

We're left with two problems to solve:

1. The characters on the right aren't centered
2. There's no space between the logo and the other UI elements

To center the characters on the right, we'll use a `CenterContainer`. Add a `CenterContainer` node inside the `VBoxContainer`, then drag and drop the `Characters` into it. The `Characters` element will center automatically.



Fig. 2.25: Containers automatically place and resize textures

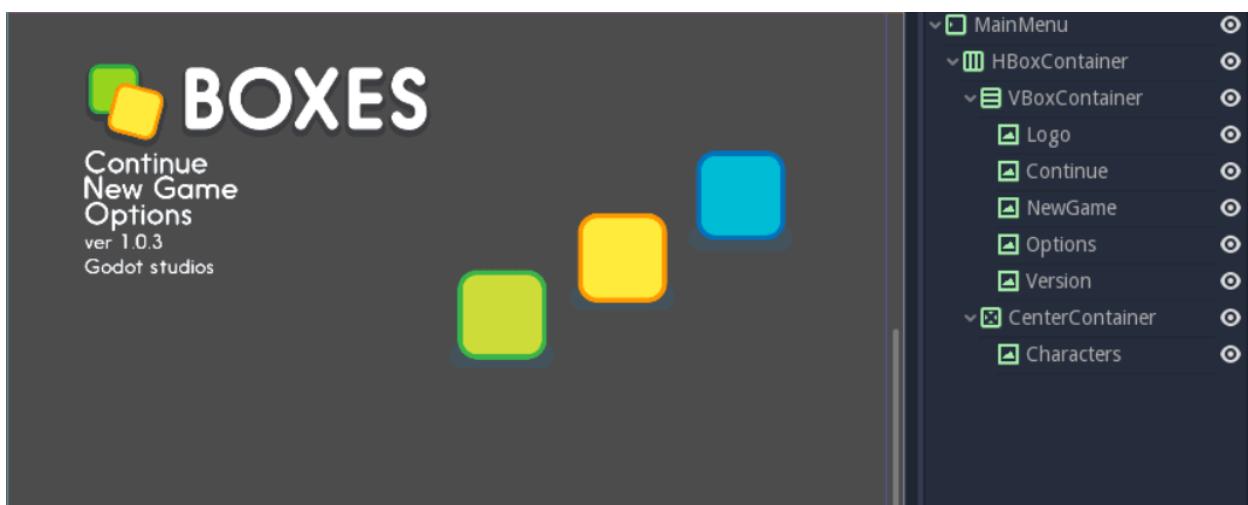


Fig. 2.26: The character node centers inside the right half of the screen as soon as you place it inside the CenterContainer

To space out the menu options and the logo on the left, we'll use one final container and its size flags. Select the VBoxContainer and press **Meta+A** to add a new node inside it. Add a second VBoxContainer and name it “MenuOptions”. Select all three menu options, Continue, NewGame and Options, and drag and drop them inside the new VBoxContainer. The UI's layout should barely change, if at all.

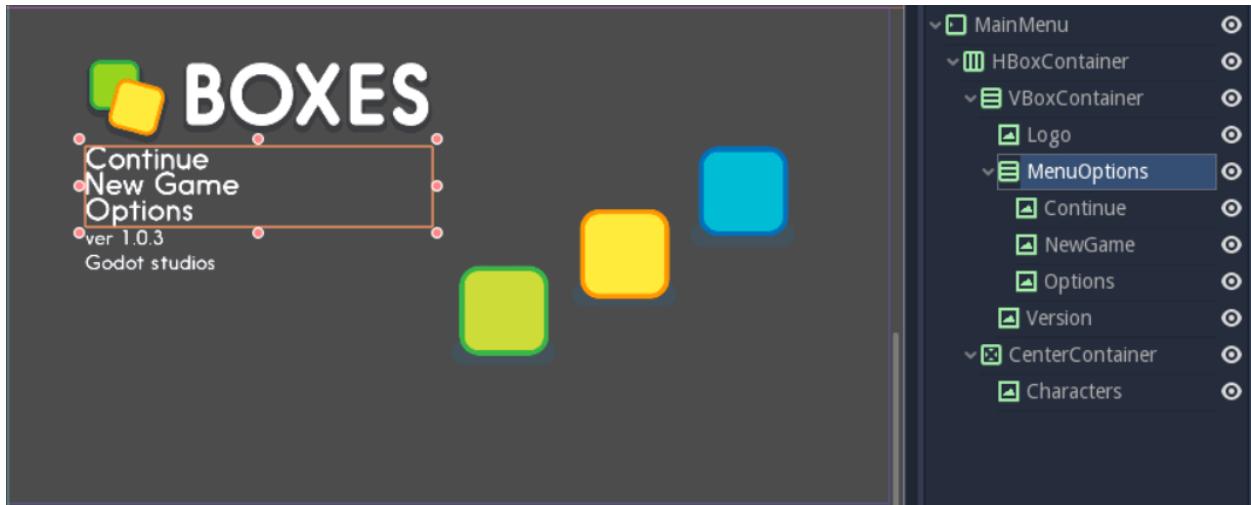


Fig. 2.27: Place the new container between the other two nodes to retain the UI's layout

Now we grouped the menu options together, we can tell their container to expand to take as much vertical space as possible. Select the **MenuOptions** node. In the Inspector, scroll down to the **Size Flags** category. Click on the field to the right of the **Vertical** property, and check **Expand**. The container expands to take all the available vertical space. But it respects its neighbors, the **Logo** and **Version** elements.

To center the nodes in the **VBoxContainer**, scroll to the top of the Inspector and change the **Alignment** property to **Center**.

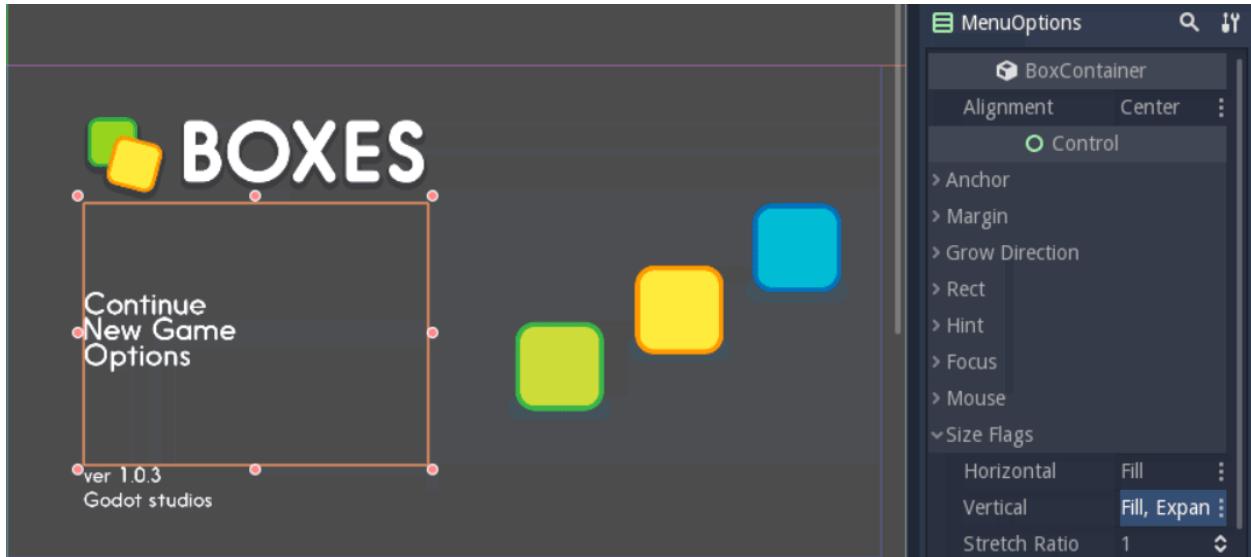


Fig. 2.28: The menu options should center vertically in the UI's left column

To wrap things up, let's add some separation between the menu options. Expand the **Custom Constants** category below **Size Flags**, and click the field next to the **Separation** parameter. Set it to 30. Once you press enter, the **Separation** property becomes active and Godot adds 30 pixels between menu options.



Fig. 2.29: The final interface

Without a single line of code, we have a precise and responsive main menu.

Congratulations for getting there! You can download the *final menu* to compare with your own. In the next tutorial, you'll create a Game User Interface with bars and item counters.

Break down the UI mockup

Responsive User Interface is all about making sure our UIs scale well on all screen types. TV screens and computer displays have different sizes and ratios. In Godot, we use containers to control the position and the size of UI elements.

The order in which you nest matters. To see if your UI adapts nicely to different screen ratios, select the root node, press the Q key to activate the Select Mode, select the container and click and drag on one of the container's corners to resize it. The UI components should flow inside of it.

You'll notice that although containers move sprites around, they don't scale them. This is normal. We want the UI system to handle different screen ratios, but we also need the entire game to adapt to different screen resolutions. To do this, Godot scales the entire window up and down.

You can change the scale mode in the project settings: click the Project menu -> Project Settings. In the window's left column, look for the Display category. Click on the Window sub-category. On the right side of the window, you'll find a Stretch section. The three settings, Mode, Aspect, and Shrink, control the screen size. For more information, see *Multiple resolutions*.

2.11 Design the GUI

Now that you've nailed the basics, we're going to see how to build a Game User Interface (GUI) with reusable UI components: a life bar, an energy bar, and bomb and rupee counters. By the end of this tutorial, you'll have a game GUI, ready to control with GDscript or VisualScript:

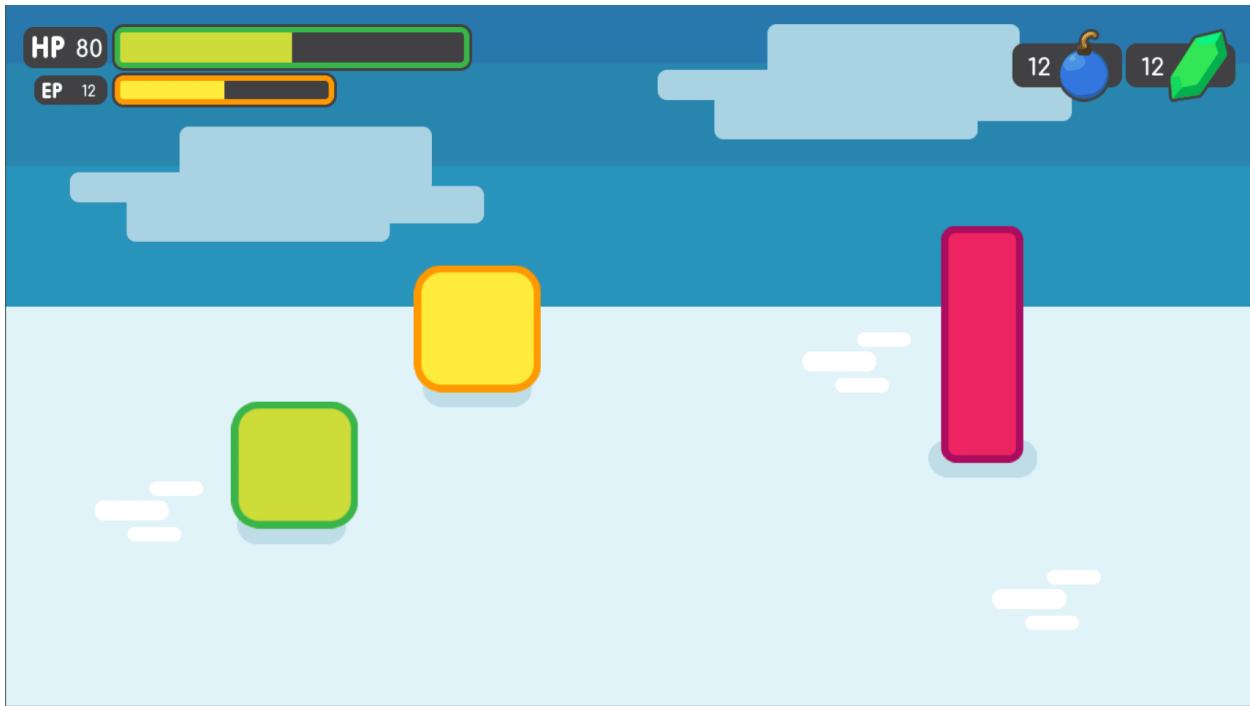


Fig. 2.30: The final result

You'll also learn to:

1. Create flexible UI components
2. Use scene inheritance
3. Build a complex UI

Download the project files: `ui_gui_design.zip` and extract the archive. Import the `start/` project in Godot to follow this tutorial. The `end/` folder contains the final result.

2.11.1 Breaking down the UI

Let's break down the final UI and plan the containers we'll use. As in the *main menu tutorial*, it starts with a `MarginContainer`. Then, we can see up to three columns:

1. The life and energy counters on the left
2. The life and energy bars
3. The bomb and rupee counters on the right

But the bar's label and the gauge are two parts of the same UI element. If we think of them this way, we're left with two columns:

1. The life and energy bars on the left
2. The bomb and rupee counters on the right

This makes it easier to nest containers: we have some margins around the border of the screen using a `MarginContainer`, followed by an `HBoxContainer` to manage our two columns. The two bars stack on top of one another inside a `VBoxContainer`. And we'll need a last `HBoxContainer` in the right column to place the bomb and rupee counters side-by-side.

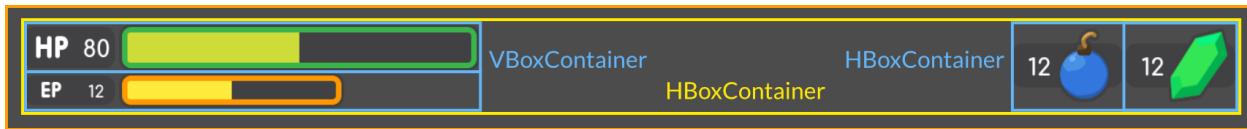


Fig. 2.31: We get a clean UI layout with only 4 containers

We will need extra containers inside the individual UI components, but this gives us the main GUI scene's structure. With this plan in place, we can jump into Godot and create our GUI.

2.11.2 Create the base GUI

There 2 possible approaches to the GUI: we can design elements in separate scenes and put them together, or prototype everything in a single scene and break it down later. I recommend working with a single scene as you can play with your UI's placement and proportions faster this way. Once it looks good, you can save entire sections of the node tree as reusable sub-scenes. We'll do just that in a moment.

For now, let's start with a few containers.

Create a new scene and add a MarginContainer. Select the node and name it `GUI`. Then save the scene as `GUI.tscn`. It will contain the entire GUI.

With the MarginContainer selected, head to the inspector and scroll down to the custom constants section. Unfold it and click the field next to each of the `Margin` properties. Set them all to 20 pixels. Next, add an HBoxContainer node. This one will contain our two bars on the left and separate them from the two counters on the right.

We want to stack the bars vertically inside the HBoxContainer. To do this, let's add a VBoxContainer. Name it `Bars`. Select the parent HBoxContainer again and this time, add another HBoxContainer. This one will hold the counters, so call it `Counters`. With these four containers, we have the base for our GUI scene.

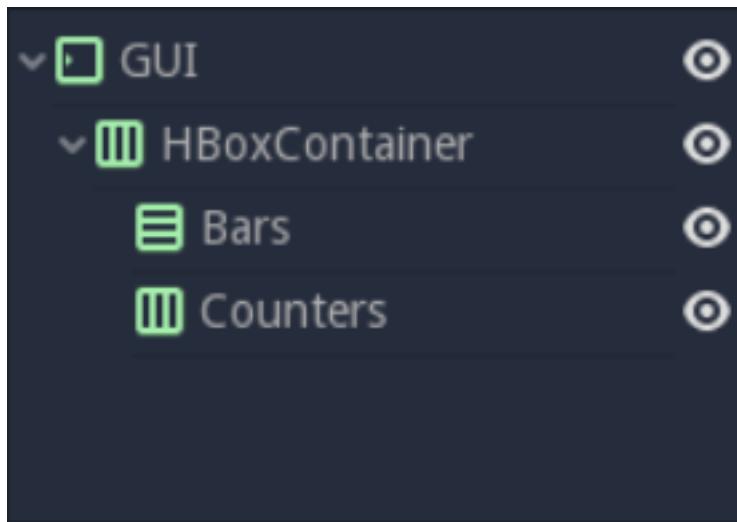


Fig. 2.32: You should have 4 containers that look like this

Note: We can work this way because we first broke down our UI design and took a few moments to think about the containers we'd use. When you follow a tutorial like this, it may seem weird. But once you're working on real games, you'll see it's an efficient workflow.

Create the bars' base

Each bar is split into two sub-elements that align horizontally: the label with the health count on the left, and the gauge on the right. Once again, the `HBoxContainer` is the perfect tool for the job. Select the `Bars` node and add a new `HBoxContainer` inside of it. Name it `Bar`.

The label itself is requires at least three nodes: a `NinePatchRect` for the background, on top of which we'll add a texture on the left, either `HP` or `EP`, and a `Label` on the right for the value. We can nest `Control` nodes however we want. We could use the `NinePatchRect` as a parent for the two other elements, as it encompasses them. In general, you want to use containers instead, as their role is to help organize UI components. We'll need a `MarginContainer` later anyway to add some space between the life count and the gauge. Select the `Bar` and add a `MarginContainer`. Name it `Count`. Inside of it, add three nodes:

1. A `NinePatchRect` named `Background`
2. A `TextureRect` named `Title`
3. And a `Label` named `Number`

To add the nodes as siblings, always select the `Count` node first.

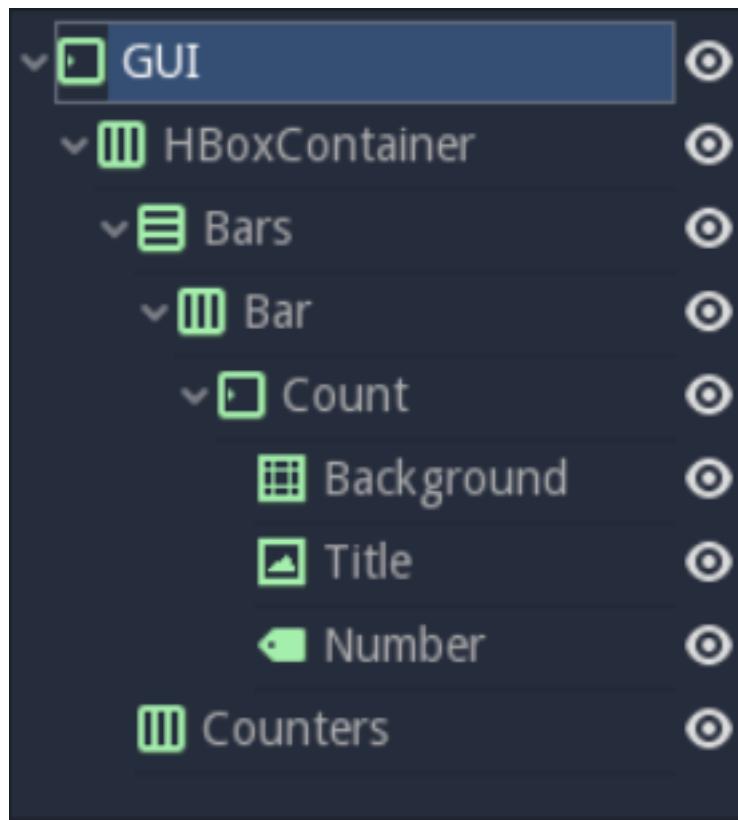


Fig. 2.33: Your scene tree should look like this. We're ready to throw in some textures

Our scene is still empty. It's time to throw in some textures. To load the textures, head to the `FileSystem` dock to the left of the viewport. Browse down to the `res://assets/GUI` folder.

Select the `Background` in the `Scene` dock. In the `Inspector`, you should see a `Texture` property. In the `FileSystem` tab, click and drag `label_HP_bg.png` onto the `Texture` slot. It stays squashed. The parent `MarginContainer` will force its size down to 0 until we force elements inside the container to have a minimum size. Select the `Background` node. In the `Inspector`, scroll down to the `Rect` section. Set `Min Size` to `(100, 40)`. You should see the `Background` resize along with its parent containers.

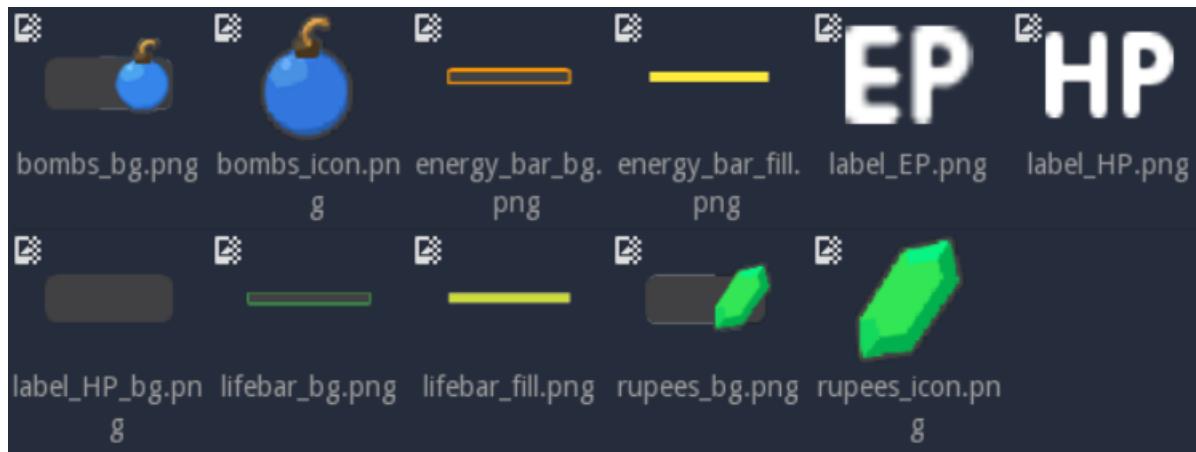


Fig. 2.34: You should see a list of textures that we'll use to skin our interface.

Next, select the Title and drag and drop `label_HP.png` into its Texture slot. Select the Number node, click the field next to the Text property and type 10. This way, we can see both nodes in the viewport. They should stack up in the top-left corner of their parent `MarginContainer`.



Fig. 2.35: If you select both nodes, you should see something like this

As they have a container as their direct parent, we cannot move them freely: the `Count` node will always reset their anchors, their size and position. Try to move and resize the nodes in the viewport. Then, select any of the three textures and press `Ctrl Up` or `Ctrl Down` to reorder them in the Scene dock. They'll snap back to their previous size and position.

Parent containers control the size, the scale, the margins, and the anchors of their direct children. To modify the nodes, you must nest them inside a regular `Control` or another UI element. We'll use the `Background` as a parent for the `Title` and `Number`. Select both the `Title` and `Number`, and drag and drop them onto `Background`.

Select the `Title` and in the Inspector, change its `Stretch Mode` property to `Keep Centered`. Resize it in the viewport with the `Select Mode` tool so it only takes the left half of the background. Next, select the `Number` node. In the viewport, click the `Layout` menu and click `Full Rect`. The node will resize to fit the `Background`. Head to the Inspector and change its `Align` property to `Right`, and the `VAlign` property to `Center`. The text should snap to the center of the `Background`'s right edge. Resize the node horizontally so it takes the right half of the `Background` and there's a bit of padding with the right edge.

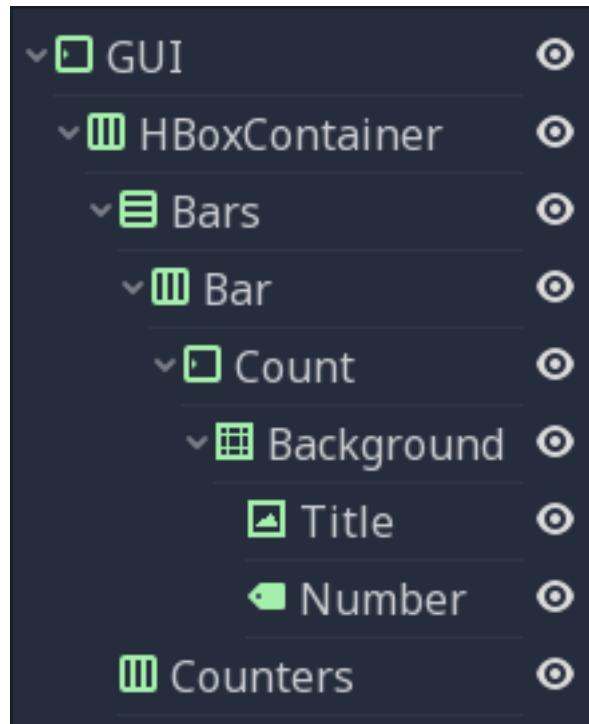


Fig. 2.36: By using the Background node as the two textures' parent, we take control away from the Count Margin-Container

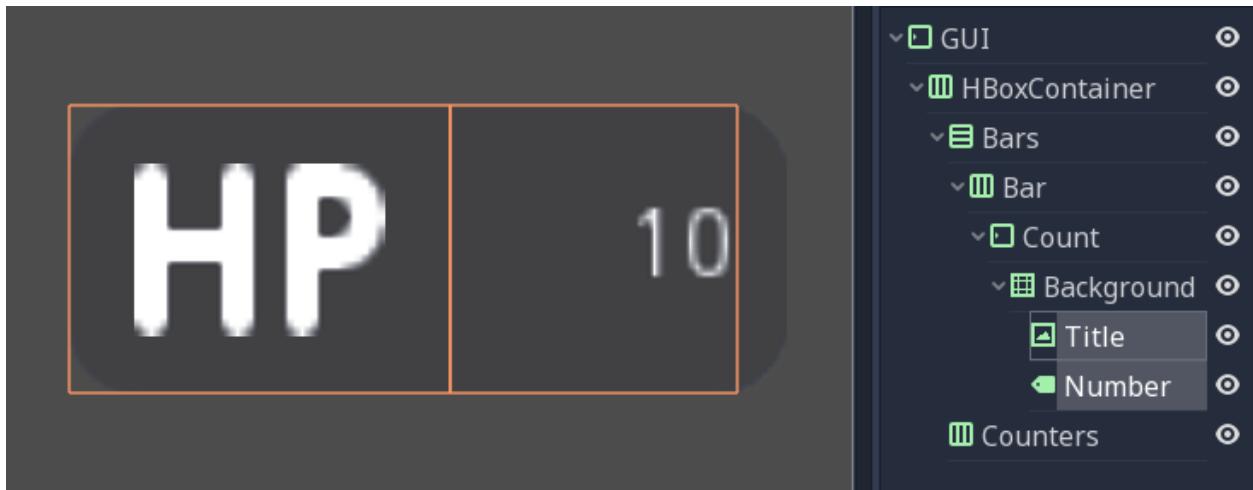


Fig. 2.37: Here's how the nodes' bounding boxes should look in the viewport. Keep it rough, you don't need to place them too precisely for now.

Replace the Label's font

The label's font is too small. We need to replace it. Select the Number node and in the Inspector, scroll down to the Control class, and find the Custom Font category. Click the field next to the Font property and click on New Dynamic Font. Click on the field again and select Edit.

You will enter the Dynamic Font resource. Unfold the Font category and click the field next to Font Data. Click the Load button. In the file browser, navigate down to the assets/font folder and double click Comfortaa-Bold.ttf to open it. You should see the font update in the viewport. Unfold the settings category to change the font size. Set the Size property to a higher value, like 24 or 28.

We now need the text's baseline, the number's lower edge, to align with the HP texture on the left. To do so, still in the DynamicFont resource, you can tweak the Bottom property under the Extra Spacing category. It adds some bottom padding to the text. Click the Number node in the Scene tab to go back to the node's properties and change the VAlign to Bottom. To adjust the text's baseline, click on the font field under the Custom Font category again and tweak the Bottom property until the text aligns with the Title node. I used a value of 2 pixels.

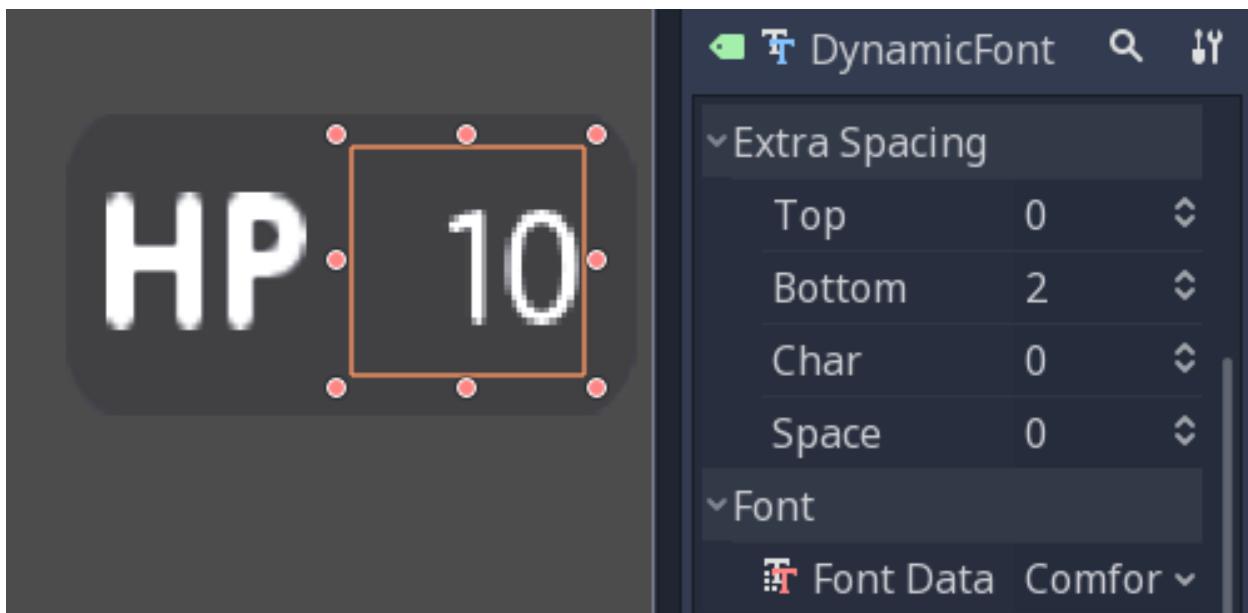


Fig. 2.38: With a Bottom value of 2 pixels, the Number aligns with the Title

With this, we just finished the hardest part of the GUI. Congratulations! Let's move on to the simpler nodes.

Add the progress bar

We need one last element to complete our life bar: the gauge itself. Godot ships with a TextureProgress node that has everything we need.

Select the Bar node and add a TextureProgress inside of it. Name it Gauge. In the inspector unfold the Textures section. Head to the FileSystem dock and drag and drop the lifebar_bg.png texture onto the Under slot. Do the same with the lifebar_fill.png image and drop it onto the Progress slot. Under the Range class in the inspector, change the Value property to 50 to see the gauge fill up.

With only five Control nodes, our first bar is ready to use.

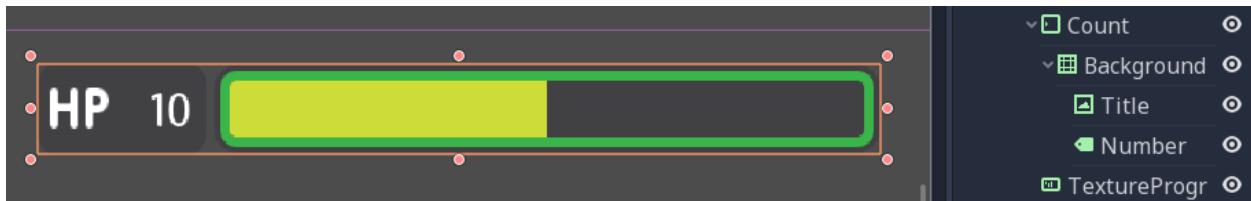


Fig. 2.39: That's it, our life bar is ready. This last part was quick, wasn't it? That's thanks to our robust container setup.

2.11.3 Design the bomb and rupee counters

The bomb and rupee counters are like the bar's `Count` node. So we'll duplicate it and use it as a template.

Under the `Bar` node, select `Count` and press `Ctrl D` to duplicate it. Drag and drop the new node under the `Counters` `HBoxContainer` at the bottom of the scene tree. You should see it resize automatically. Don't worry about this for now, we'll fix the size soon.

Rename the `Count2` node to `Counter`. Unlike the bars, we want the number to be on the left, and an icon to sit on the right. The setup is the same: we need background, a `NinePatchFrame`, the title, and the number nodes. The `Title` node is a `TextureRect`, so it's what we need to display the icon. In the scene tree, select the `Title` node, and rename it to `Icon`.

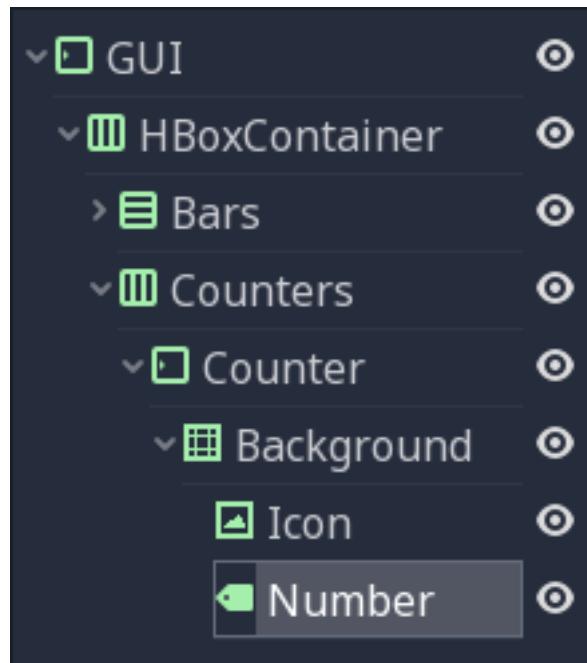


Fig. 2.40: Here's how your node tree should look so far

With the `Icon` node selected, in the inspector, scroll to the top to see the `Texture` slot. Head to the `FileSystem` dock on the left and select the `bombs_icon.png`. Drag and drop it onto the `Texture` slot. In the `Scene Tab` select both the `Icon` and the `Number` nodes. Click the `Layout` menu in the toolbar at the top of the viewport and select `Full Rect`. Both nodes will update to fit the size of the `Background`.

Let's change the `Number`'s align properties to move it to the left and center of the `Background`. Select the `Number` node, change its `Align` property to `left` and the `VAlign` property to `center`. Then resize its left edge a little bit to add some padding between the left edge of the `Background` and the text.

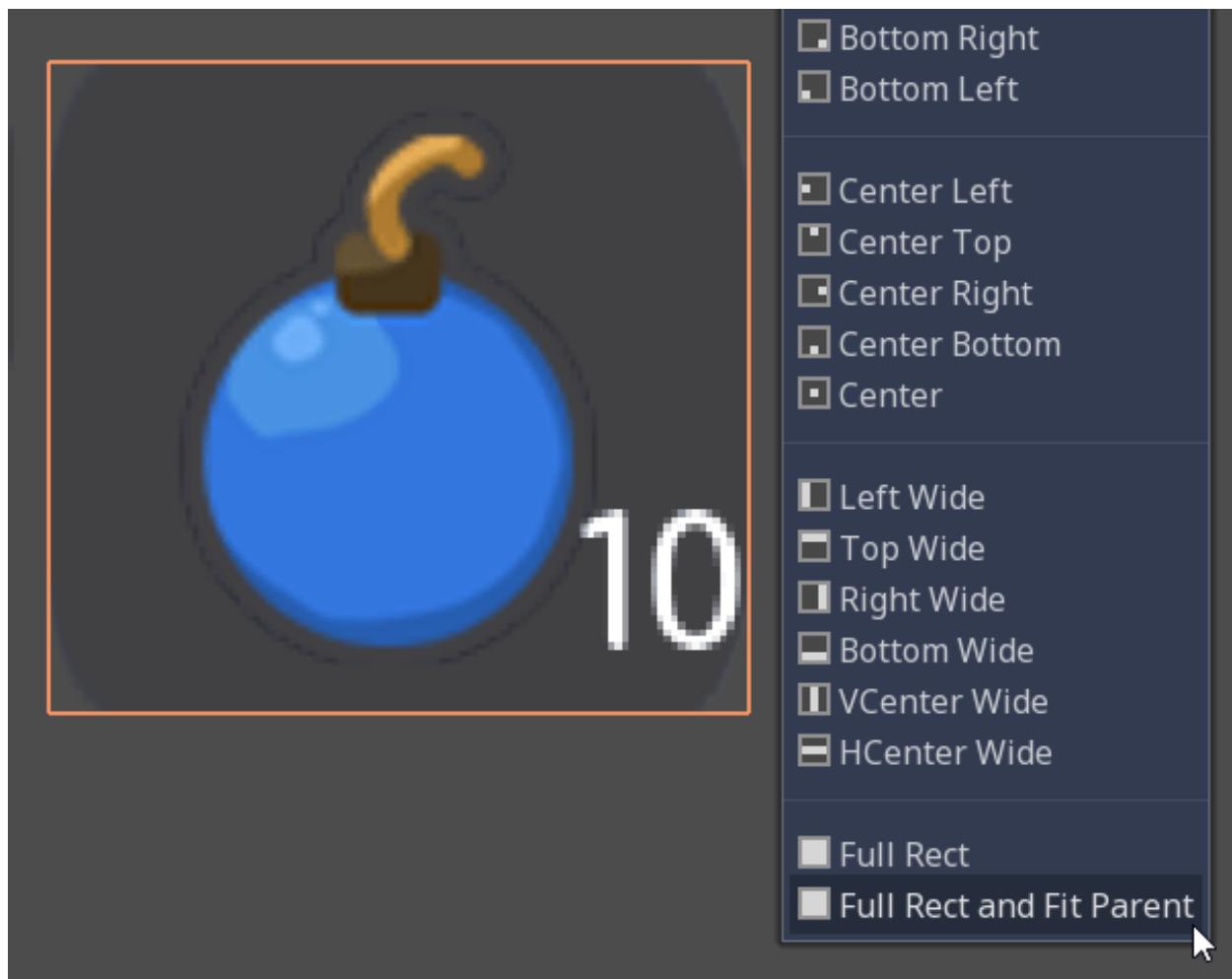


Fig. 2.41: The nodes anchor to the entire Background, but their position is off

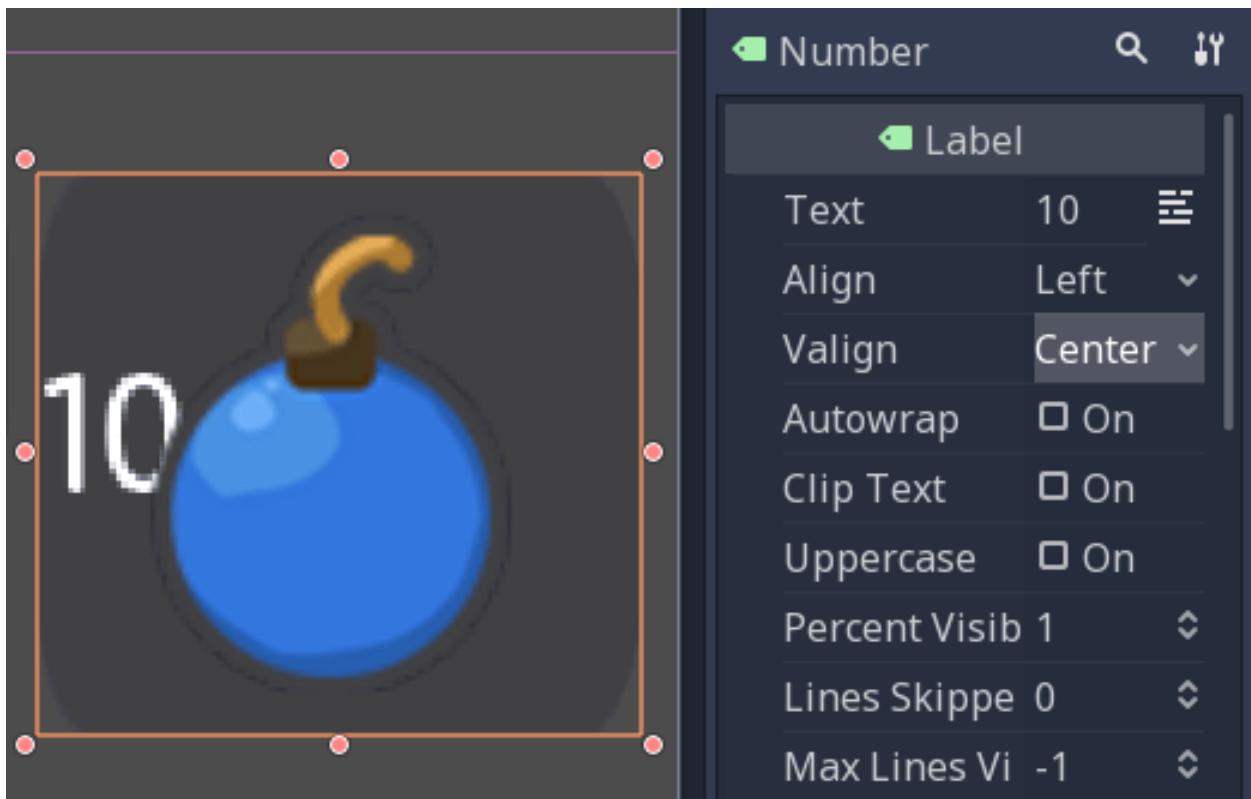


Fig. 2.42: The Number node aligned to the left and centre

To overlap the Icon and the background, we need a few tweaks. First, our background is a bit too tall. It's because it's inside a margin container that is controlled by the top-most GUI node. Select the GUI node at the top of the scene tree and downsize it vertically so that it's as thin as possible. You'll see the gauge prevents you from making it too small. A container cannot be smaller than the minimal size of its children. The container's margins also weigh in.

Select the Icon, click the Layout menu, and select `Full Rect` to re-center it. We need it to anchor to the Background's right edge. Open the Layout menu again and select `Center Right`. Move the icon up so it is centered vertically with the Background.

Because we duplicated the Counter from the bar's Count, the Number node's font is off. Select the Number node again, head to the `Font` property, and click it to access the `DynamicFont` resource. In the `Extra Spacing` section, change the `Bottom` value to `0` to reset the font's baseline. Our counter now works as expected.

While we are at it, let's make it so the Counters snap to the right edge of the viewport. To achieve this we will set the Bars container to expand and take all the horizontal space. Select the Bars node and scroll down to the `Size Flags` category. In the `Horizontal` category, check the `Expand` value. The Bars node should resize and push the counter to the rightmost of the screen.

2.11.4 Turn the bar and counter into reusable UI components

We have one bar and one counter widget. But we need two of each. We may need to change the bars' design or their functionality later on. It'd be great if we could have a single scene to store a UI element's template, and child scenes to work on variations. Godot lets us do this with Inherited Scenes.

Let's save both the Counter and the Bar branches as separate scenes that we'll reduce to create the `LifeBar`, the `EnergyBar`, the `BombCounter`, and the `RupeeCounter`. Select the Bar HBoxContainer. Right click on it and

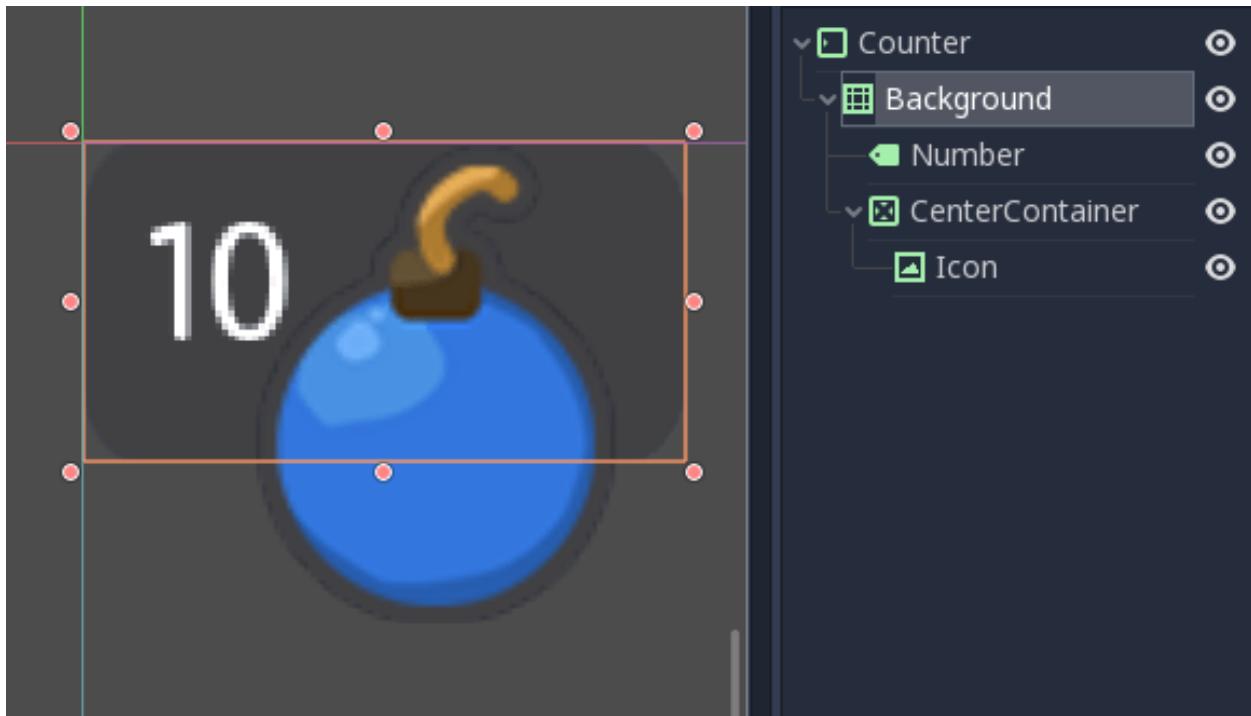


Fig. 2.43: The bomb icon anchors to the Background's right edge. Resize the Counter container to see the Icon node stick to its right side



Fig. 2.44: An expanding container eats all the space it can from its parent, pushing everything else along the way

click on Save Branch as Scene. Save the scene as `Bar.tscn`. You should see the node branch turn it to a single Bar node.

Tip: A scene is a tree of nodes. The topmost node is the tree's **root**, and the children at the bottom of the hierarchy are **leaves**. Any node other than the root along with one more children is a **branch**. We can encapsulate node branches into separate scenes, or load and merge them from other scenes into the active one. Right click on any node in the Scene dock and select Save Branch as Scene or Merge from Scene.

Then, select the Counter node and do the same. Right click, Save Branch as Scene, and save it as `Counter.tscn`. A new edit scene icon appears to the right of the nodes in the scene tree. Click on the one next to Bar to open the corresponding scene. Resize the Bar node so that its bounding box fits its content. The way we named and place the Control nodes, we're ready to inherit this template and create the life bar. It's the same for the Counter.

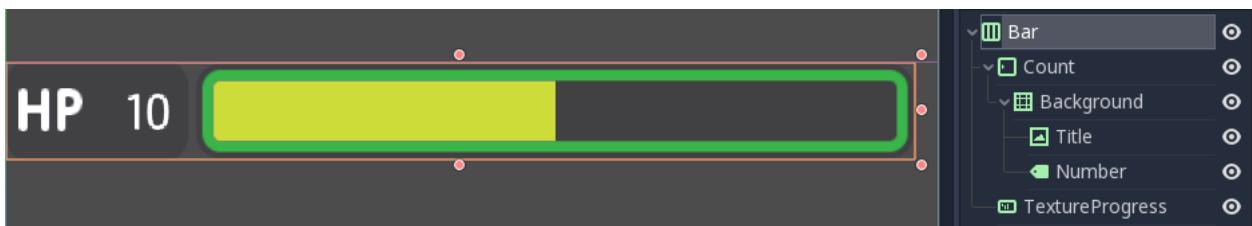


Fig. 2.45: With no extra changes, our Bar is ready to use

2.11.5 Use Scene Inheritance to create the remaining elements

We need two bars that work the same way: they should feature a label on the left, with some value, and a horizontal gauge on the right. The only difference is that one has the HP label and is green, while the other is called EP and is yellow. Godot gives us a powerful tool to create a common base to reuse for all bars in the game: **inherited scenes**.

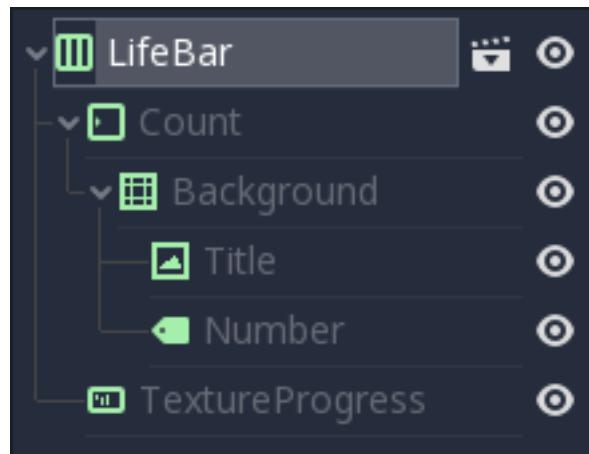


Fig. 2.46: Inherited scenes help us keep the GUI scene clean. In the end, we will only have containers and one node for each UI component.

On an inherited scene, you can change any property of every node in the inspector, aside from its name. If you modify and save the parent scene, all the inherited scenes update to reflect the changes. If you change a value in the inherited scene, it will always override the parent's property. It's useful UIs as they often require variations of the

same elements. In general, in UI design, buttons, panels etc. share a common base style and interactions. We don't want to copy it over to all variations manually.

A reload icon will appear next to the properties you override. Click it to reset the value to the parent scene's default.

Note: Think of scene inheritance like the node tree, or the `extends` keyword in GDScript. An inherited scene does everything like its parent, but you can override properties, resources and add extra nodes and scripts to extend its functionality.

Inherit the Bar Scene to build the LifeBar

Go to Scene → New Inherited Scene to create a new type of Bar. Select the Bar scene and open it. You should see a new [unsaved] tab, that's like your Bar, but with all nodes except the root in grey. Press Meta+S to save the new inherited scene and name it LifeBar.

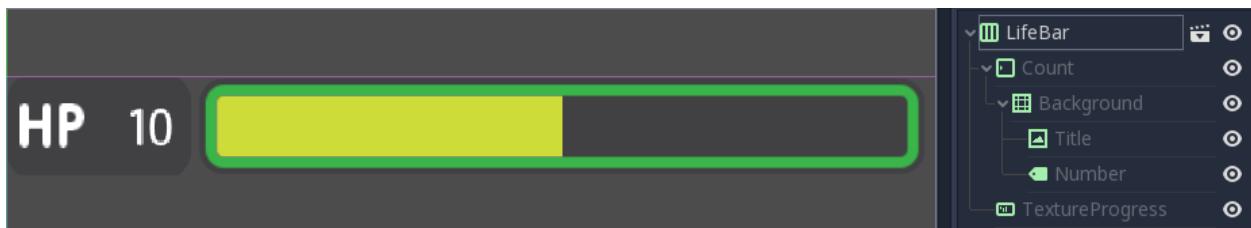


Fig. 2.47: You can't rename grey nodes. This tells you they have a parent scene

First, rename the root or top level node to LifeBar. We always want the root to describe exactly what this UI component is. The name differentiates this bar from the EnergyBar we'll create next. The other nodes inside the scene should describe the component's structure with broad terms, so it works with all inherited scenes. Like our TextureProgress and Number nodes.

Note: If you've ever done web design, it's the same spirit as working with CSS: you create a base class, and add variations with modifier classes. From a base button class, you'll have button-green and button-red variations for the user to accept and refuse prompts. The new class contains the name of the parent element and an extra keyword to explain how it modifies it. When we create an inherited scene and change the name of the top level node, we're doing the same thing

Design the EnergyBar

We already setup the LifeBar's design with the main Bar scene. Now we need the EnergyBar.

Let's create a new inherited scene, and once again select the Bar.tscn scene and open it. Double-click on the Bar root node and rename it to EnergyBar. Save the new scene as EnergyBar.tscn. We need to replace the HP texture with EP one, and to change the textures on the gauge.

Head to the FileSystem dock on the left, select the Title node in the Scene tree and drag and drop the label_EP.png file onto the texture slot. Select the Number node and change the Text property to a different value like 14.

You'll notice the EP texture is smaller than the HP one. We should update the Number's font size to better fit it. A font is a resource. All the nodes in the entire project that use this resource will be affected by any property we change. You can try to change the size to a huge value like 40 and switch back to the LifeBar or the Bar scenes. You will see the text increased in size.

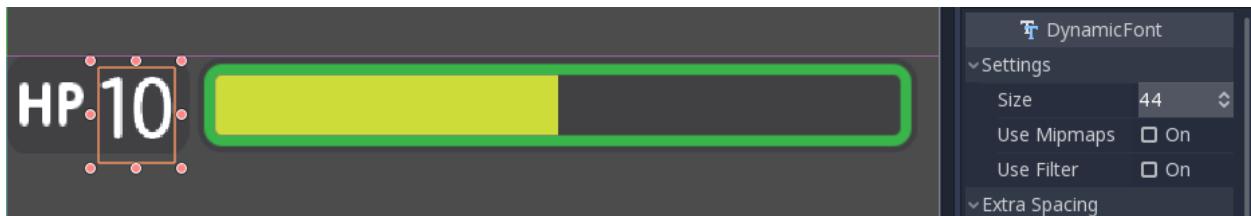


Fig. 2.48: If we change the font resource, all the nodes that use it are affected

To change the font size on this node only, we must create a copy of the font resource. Select the `Number` node again and click on the wrench and screwdriver icon on the top right of the inspector. In the drop-down menu, select the `Make Sub-Resources Unique` option. Godot will find all the resources this node uses and create unique copies for us.

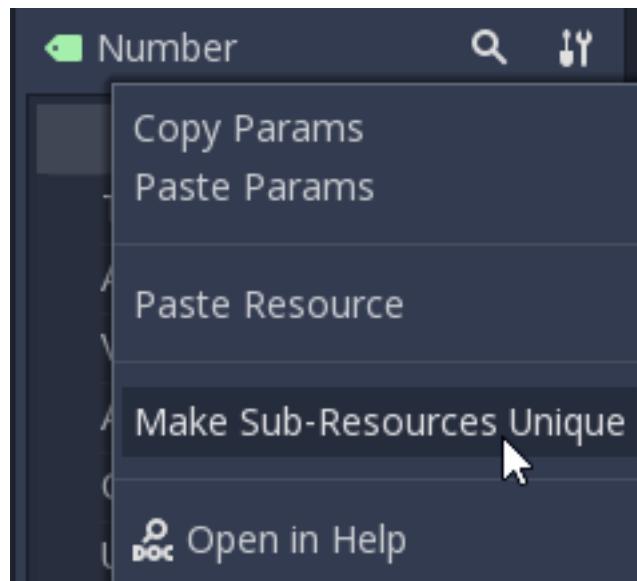


Fig. 2.49: Use this option to create unique copies of the resources for one node

Tip: When you duplicate a node from the Scene tree, with `Meta+D`, it shares its resources with the original node. You need to use `Make Sub-Resources Unique` before you can tweak the resources without affecting the source node.

Scroll down to the `Custom Font` section and open `Font`. Lower the `Size` to a smaller value like 20 or 22. You may also need to adjust the `Bottom` spacing value to align the text's baseline with the `EP` label on the left.



Fig. 2.50: The EP Count widget, with a smaller font than its HP counterpart

Now, select the `TextureProgress` node. Drag the `energy_bar_bg.png` file onto the `Under` slot and do the same for `energy_bar_fill.png` and drop it onto the `Progress` texture slot.

you can resize the node vertically so that its bounding rectangle fits the gauge. Do the same with the Count node until its size aligns with that of the bar. Because the minimal size of `TextureProgress` is set based on its textures, you won't be able to downsize the Count node below that. That is also the size the Bar container will have. You may downscale this one as well.

Last but not least, the `Background` container has a minimum size that makes it a bit large. Select it and in the `Rect` section, change the `Min Size` property down to 80 pixels. It should resize automatically and the `Title` and `Number` nodes should reposition as well.



Fig. 2.51: The Count looks better now it's a bit smaller

Tip: The Count node's size affects the position of the `TextureProgress`. As we'll align our bars vertically in a moment, we're better off using the Counter's left margin to resize our EP label. This way both the EnergyBar's Count and the LifeBar's Count nodes are one hundred pixels wide, so both gauges will align perfectly.

Prepare the bomb and rupee counters

Let us now take care of the counters. Go to `Scene -> New Inherited Scene` and select the `Counter.tscn` as a base. Rename the root node as `BombCounter` too. Save the new scene as `BombCounter.tscn`. That's all for this scene.

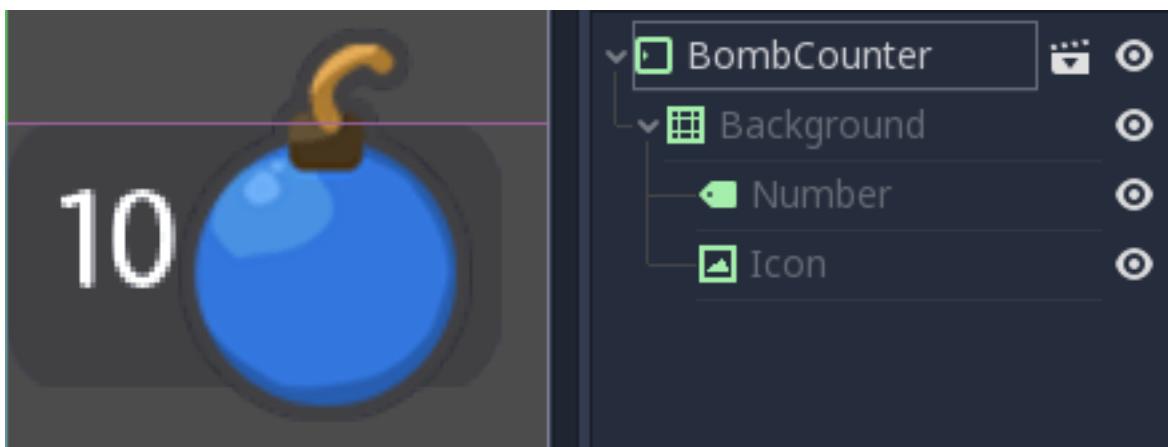


Fig. 2.52: The bomb counter is the same as the original Counter scene

Go to `Scene -> New Inherited Scene` again and select `Counter.tscn` once more. Rename the root node `RupeeCounter` and save the scene as `RupeeCounter.tscn`. For this one, we mainly need to replace the bomb icon with the rupee icon. In the `FileSystem` tab, drag the `rupees_icon.png` onto the `Icon` node's `Texture` slot. `Icon` already anchors to the right edge of the `Background` node so we can change its position and it will scale and reposition with the `RupeeCounter` container. Shift the rupee icon a little bit to the right and down. Use the Arrow Keys on the keyboard to nudge its position. Save, and we're done with all the UI elements.

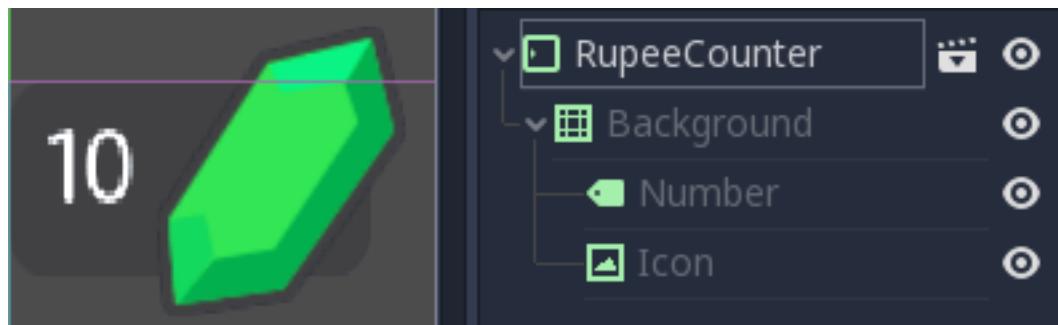


Fig. 2.53: The rupee counter should look about like this

2.11.6 Add the UI components to the final GUI

Time to add all the UI elements to the main GUI scene. Open the `GUI.tscn` scene again, and delete the `Bar` and `Counter` nodes. In the FileSystem dock, find the `LifeBar.tscn` and drag and drop it onto the `Bars` container in the scene tree. Do the same for the `EnergyBar`. You should see them align vertically.

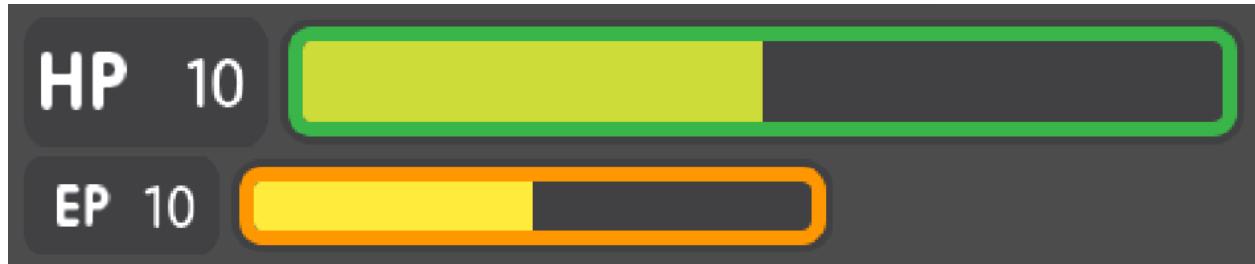


Fig. 2.54: The LifeBar and the EnergyBar align automatically

Now, drag and drop the `BombCounter` and `RupeeCounter` scenes onto the `Counters` node. They'll resize size automatically.

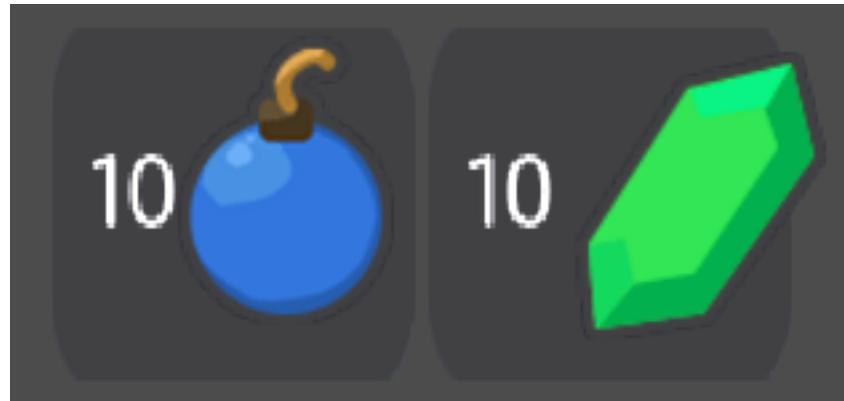


Fig. 2.55: The nodes resize to take all the available vertical space

To let the `RupeeCounter` and `BombCounter` use the size we defined in `Counter.tscn`, we need to change the `Size Flags` on the `Counters` container. Select the `Counters` node and unfold the `Size Flags` section in the Inspector. Uncheck the `Fill` tag for the `Vertical` property, and check `Shrink Center` so the container centers inside the `HBoxContainer`.

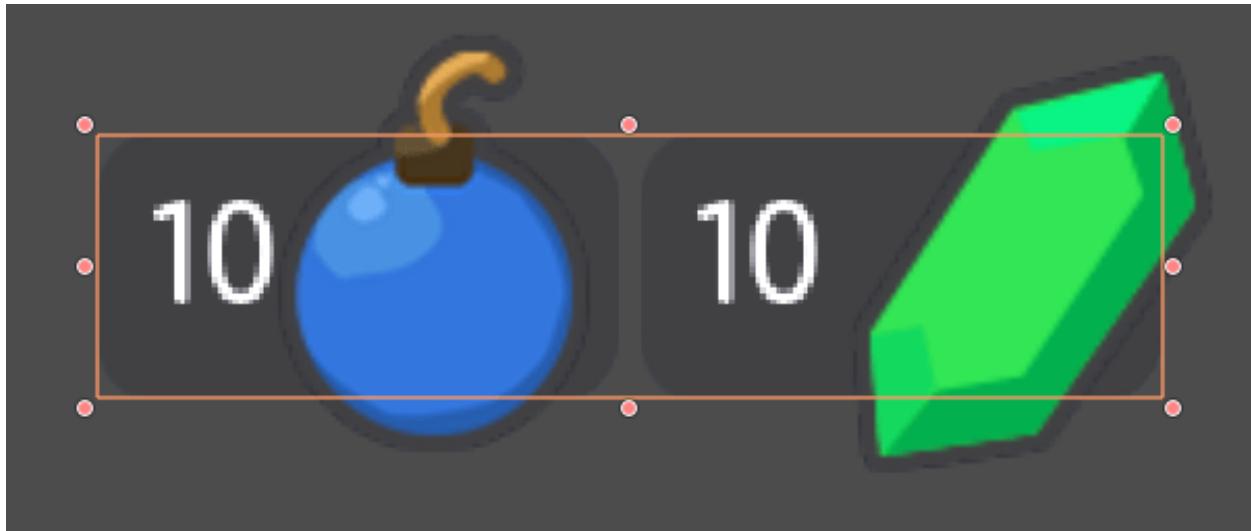


Fig. 2.56: Now both counters have a decent size

Tip: Change the `Min Size` property of the `Counters` container to control the height of the counters' background.

We have one small issue left with the EP label on the EnergyBar: the 2 bars should align vertically. Click the icon next to the EnergyBar node to open its scene. Select the Count node and scroll down to the `Custom Constant` section. As a add a `Margin Left` of 20. In the `Rect` section set the node's `Min Size` back to 100, the same value as on the LifeBar. The Count should now have some margin on the left. If you save and go back to the GUI scene, it will be aligned vertically with the Life 'Bar.



Fig. 2.57: The 2 bars align perfectly

Note: We could have setup the EnergyBar this way a few moments ago. But this shows you that you can go back to any scene anytime, tweak it, and see the changes propagate through the project!

2.11.7 Place the GUI onto the game's mockup

To wrap up the tutorial we're going to insert the GUI onto the game's mockup scene.

Head to the FileSystem dock and open `LevelMockup.tscn`.

Drag-and-drop the `GUI.tscn` scene right below the `bg` node and above the `Characters`. The GUI will scale to fit the entire viewport. Head to the Layout menu and select the `Center Top` option so it anchors to the top edge of

the game window. Then resize the GUI to make it as small as possible vertically. Now you can see how the interface looks in the context of the game.

Congratulations for getting to the end of this long tutorial. You can find final project *here*.

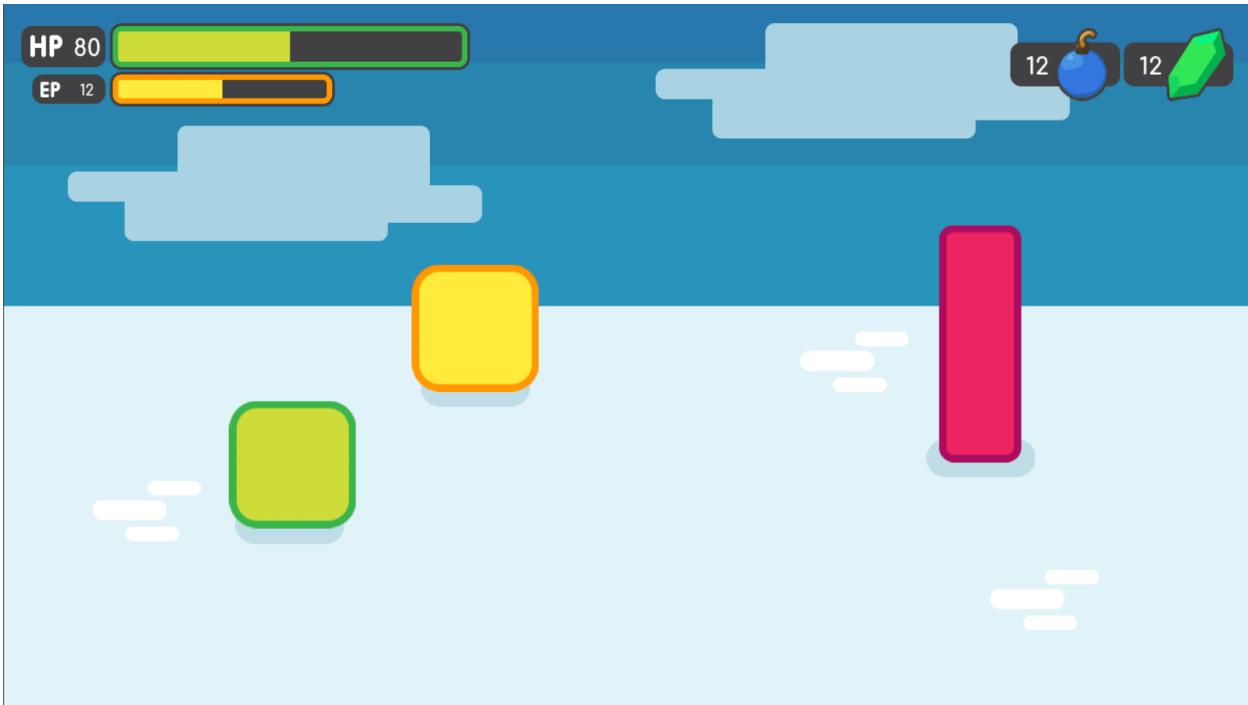


Fig. 2.58: The final result

Note: A final note about Responsive Design. If you resize the GUI, you'll see the nodes move, but the textures and text won't scale. The GUI also has a minimum size, based on the textures inside of it. In games, we don't need the interface to be as flexible as that of a website. You almost never want to support both landscape and portrait screen orientations. It's one of the other. In landscape orientation, the most common ratios range from 4:3 to 16:10. They are close to one another. That's why it's enough for the GUI elements to only move horizontally when we change the window size.

2.12 Control the game's UI with code

2.12.1 Intro

In this tutorial you will connect a character to a life bar and animate the health loss.

Fig. 2.59: Here's what you'll create: the bar and the counter animate when the character takes a hit. They fade when it dies.

You will learn:

- How to **connect** a character to a GUI with signals
- How to **control** a GUI with GDscript

- How to **animate** a life bar with the *Tween* node

If you want to learn how to set up the interface instead, check out the step-by-step UI tutorials:

- Create a main menu screen
- Create a game user interface

When you code a game, you want to build the core gameplay first: the main mechanics, player input, win and loss conditions. The UI comes a bit later. You want to keep all the elements that make up your project separate if possible. Each character should be in its own scene, with its own scripts, and so should the UI elements. This prevents bugs, keeps your project manageable, and allows different team members to work on different parts of the game.

Once the core gameplay and the UI are ready, you'll need to connect them somehow. In our example, we have the Enemy who attacks the Player at constant time intervals. We want the life bar to update when the Player takes damage.

To do this, we will use **signals**.

Note: Signals are Godot's version of the Observer pattern. They allow us to send out some message. Other nodes can connect to the object that **emits** the signal and receive the information. It's a powerful tool we use a lot for User Interface and achievement systems. You don't want to use them everywhere though. Connecting two nodes adds some coupling between them. When there's a lot of connections, they become hard to manage. For more information on check out the [signals video tutorial](#) on GDquest.

2.12.2 Download and explore the start project

Download the Godot project: `ui_code_life_bar.zip`. It contains all the assets and scripts you need to get started. Extract the .zip archive to get two folders: *start* and *end*.

Load the *start* project in Godot. In the FileSystem dock double click on `LevelMockup.tscn` to open it. It's an RPG game's mockup where 2 characters face each other. The pink enemy attacks and damages the green square at regular time intervals, until its death. Feel free to try out the game: the basic combat mechanics already work. But as the character isn't connected to the life bar the GUI doesn't do anything.

Note: This is typical of how you'd code a game: you implement the core gameplay first, handle the player's death, and only then you'll add the interface. That's because the UI listens to what's happening in the game. So it can't work if other systems aren't in place yet. If you design the UI before you prototype and test the gameplay, chances are it won't work well and you'll have to re-create it from scratch.

The scene contains a background sprite, a GUI, and two characters.

The GUI scene encapsulates all of the Game User Interface. It comes with a barebones script where we get the path to nodes that exist inside the scene:

```
onready var number_label = $Bars/LifeBar/Count/Background/Number
onready var bar = $Bars/LifeBar/TextureProgress
onready var tween = $Tween
```

- `number_label` displays a life count as a number. It's a `Label` node
- `bar` is the life bar itself. It's a `TextureProgress` node
- `tween` is a component-style node that can animate and control any value or method from any other node

Note: The project uses a simple organisation that works for game jams and tiny games.

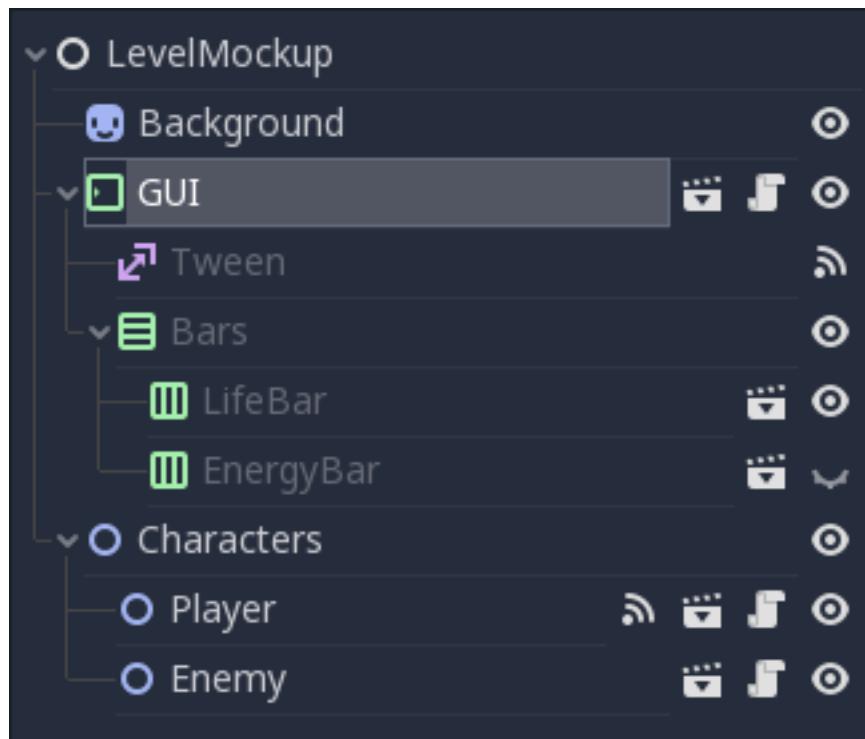


Fig. 2.60: The scene tree, with the GUI scene set to display its children

At the root of the project, in the *res://* folder, you will find the *LevelMockup*. That's the main game scene and the one we will work with. All the components that make up the game are in the *scenes/* folder. The *assets/* folder contains the game sprites and the font for the HP counter. In the *scripts/* folder you will find the enemy, the player, and the GUI controller scripts.

Click the edit scene icon to the right of the node in the scene tree to open the scene in the editor. You'll see the LifeBar and EnergyBar are sub-scenes themselves.

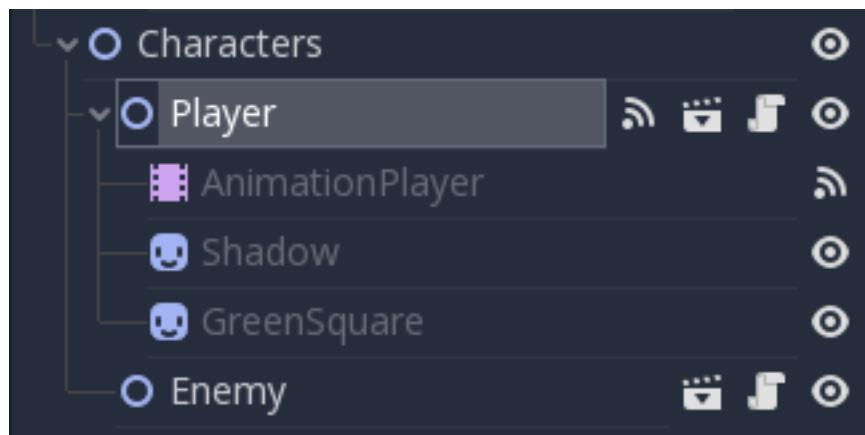


Fig. 2.61: lifebar_tutorial_Player_with_editable_children_on.png

2.12.3 Set up the Lifebar with the Player's max_health

We have to tell the GUI somehow what the player's current health is, to update the lifebar's texture, and to display the remaining health in the HP counter in the top left corner of the screen. To do this we send the player's health to the GUI every time they take damage. The GUI will then update the `Lifebar` and `Number` nodes with this value.

We could stop here to display the number, but we need to initialize the bar's `max_value` for it to update in the right proportions. The first step is thus to tell the GUI what the green character's `max_health` is.

Tip: The bar, a `TextureProgress`, has a `max_value` of `100` by default. If you don't need to display the character's health with a number, you don't need to change its `max_value` property. You send a percentage from the `Player` to the `GUI` instead: `health / max_health * 100`.



Click the script icon to the right of the `GUI` in the Scene dock to open its script. In the `_ready` function, we're going to store the `Player`'s `max_health` in a new variable and use it to set the bar's `max_value`:

```
func _ready():
    var player_max_health = $"./Characters/Player".max_health
    bar.max_value = player_max_health
```

Let's break it down. `$"./Characters/Player"` is a shorthand that goes one node up in the scene tree, and retrieves the `Characters/Player` node from there. It gives us access to the node. The second part of the statement, `.max_health`, accesses the `max_health` on the `Player` node.

The second line assigns this value to `bar.max_value`. You could combine the two lines into one, but we'll need to use `player_max_health` again later in the tutorial.

`Player.gd` sets the health to `max_health` at the start of the game, so we could work with this. Why do we still use `max_health`? There are two reasons:

We don't have the guarantee that `health` will always equal `max_health`: a future version of the game may load a level where the player already lost some health.

Note: When you open a scene in the game, Godot creates nodes one by one, following the order in your Scene dock, from top to bottom. `GUI` and `Player` are not part of the same node branch. To make sure they both exist when we access each other, we have to use the `_ready` function. Godot calls `_ready` right after it loaded all nodes, before the game starts. It's the perfect function to set everything up and prepare the game session. Learn more about `_ready`: *Scripting (continued)*

2.12.4 Update health with a signal when the player takes a hit

Our GUI is ready to receive the `health` value updates from the `Player`. To achieve this we're going to use **signals**.

Note: There are many useful built-in signals like `enter_tree` and `exit_tree`, that all nodes emit when they are respectively created and destroyed. You can also create your own using the `signal` keyword. On the `Player` node, you'll find two signals we created for you: `died` and `health_changed`.

Why don't we directly get the `Player` node in the `_process` function and look at the health value? Accessing nodes this way creates tight coupling between them. If you did it sparingly it may work. As your game grows bigger, you may have many more connections. If you get nodes from a bad it's becomes very complex very soon. Not only that: you need to listen to the changes state constantly in the `_process` function. The check happens 60 times a second and you'll likely break the game because of the order in which the code runs.

On a given frame you may look at another node's property *before* it was updated: you get a value that from the last frame. This leads to obscure bugs that are hard to fix. On the other hand, a signal is emitted right after a change happened. It **guarantees** you're getting a fresh piece of information. And you will update the state of your connected node *right after* the change happened.

Note: The Observer pattern, that signals derive from, still adds a bit of coupling between node branches. But it's generally lighter and more secure than accessing nodes directly to communicate between two separate classes. It can be okay for a parent node to get values from its children. But you'll want to favor signals if you're working with two separate branches. Read Game Programming Patterns for more information on the [Observer pattern](#). The [full book](#) is available online for free.

With this in mind let's connect the GUI to the Player. Click on the `Player` node in the scene dock to select it. Head down to the Inspector and click on the Node tab. This is the place to connect nodes to listen the one you selected.

The first section lists custom signals defined in `player.GD`:

- `died` is emitted when the character just died. We will use it in a moment to hide the UI.
- `health_changed` is emitted when the character got hit.

Select `health_changed` and click on the Connect button in the bottom right corner to open the Connect Signal window. On the left side you can pick the node that will listen to this signal. Select the GUI node. The right side of the screen lets you pack optional values with the signal. We already took care of it in `player.GD`. In general I recommend not to add too many arguments using this window as they're less convenient than doing it from the code.

Tip: You can optionally connect nodes from the code. But doing it from the editor has two advantages:

1. Godot can write new callback functions for you in the connected script 1. An emitter icon appears next to the node that emits the signal in the Scene dock
-

At the bottom of the window you will find the path to the node you selected. We're interested in the second row called "Method in Node". This is the method on the GUI node that gets called when the signal is emitted. This method receives the values sent with the signal and lets you process them. If you look to the right, there is a "Make Function" radio button that is on by default. Click the connect button at the bottom of the window. Godot creates the method inside the GUI node. The script editor opens with the cursor inside a new `_on_player_health_changed` function.

Note: When you connect nodes from the editor, Godot generates a method name with the following pattern: `_on_EmitterName_signal_name`. If you wrote the method already, the "Make Function" option will keep

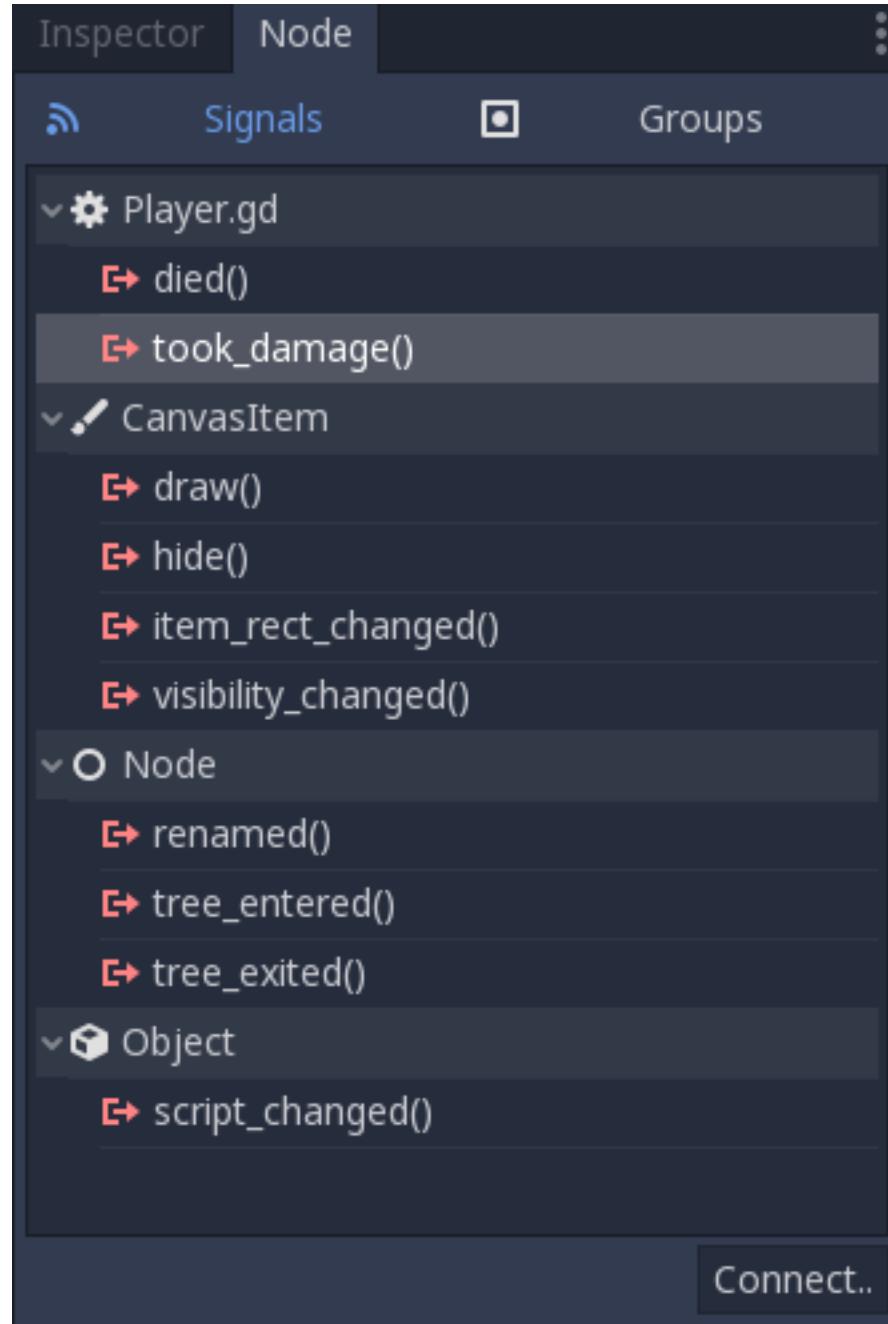


Fig. 2.62: We're connecting to the took_damage signal

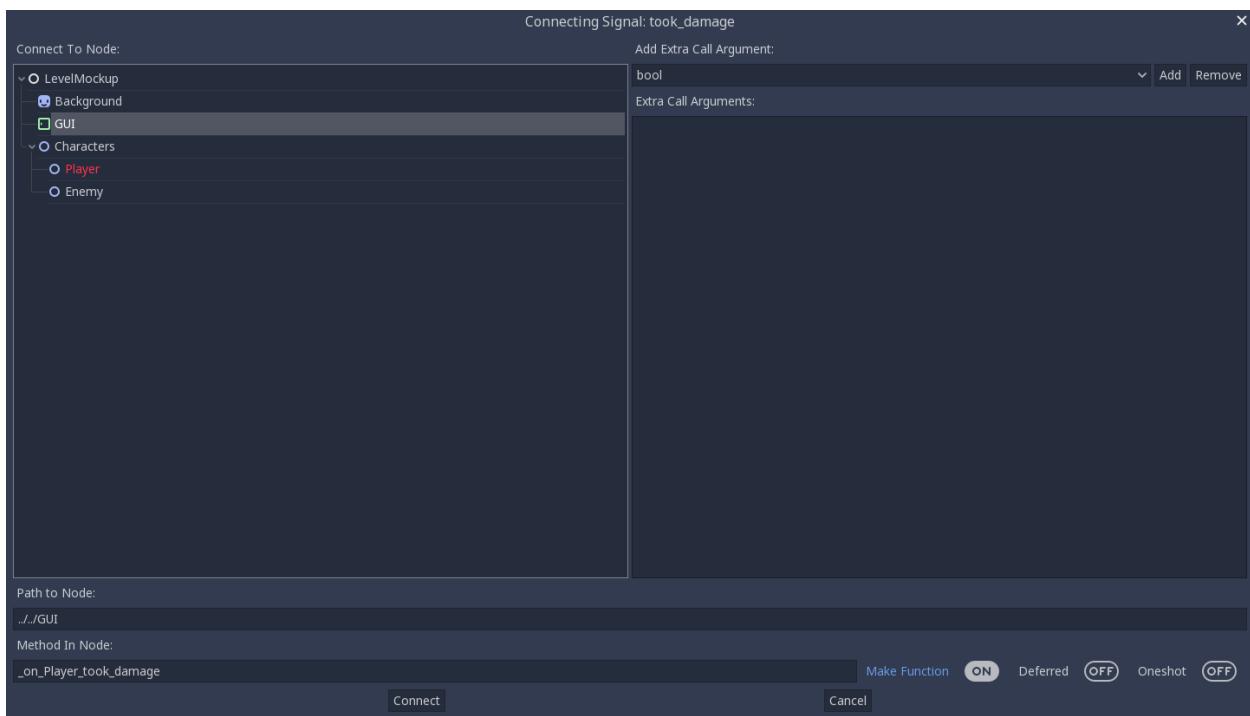


Fig. 2.63: The Connect Signal window with the GUI node selected

it. You may replace the name with anything you'd like.

```
func _on_Player_took_damage():
    pass
```

Fig. 2.64: Godot writes the callback method for you and takes you to it

Inside the parens after the function name, add a `player_health` argument. When the player emits the `health_changed` signal it will send its current `health` alongside it. Your code should look like:

```
func _on_Player_health_changed(player_health):
    pass
```

```
29 »    emit_signal("took_damage", health)
```

Fig. 2.65: In `Player.gd`, when the Player emits the `took_damage` signal, it also sends its `health` value

Inside `_on_Player_health_changed` let's call a second function called `update_health` and pass it the `player_health` variable.

Note: We could directly update the `health` value on `LifeBar` and `Number`. There are two reasons to use this method instead:

1. The name makes it very clear for our future selves and teammates that when the player took damage, we update the health count on the GUI
 2. We will reuse this method a bit later
-

Create a new `update_health` method below `_on_Player_health_changed`. It takes a `new_value` as its only argument:

```
func update_health(new_value):
```

This method needs to:

- set the Number node's text to `new_value` converted to a string
- set the TextureProgress's value to `new_value`

```
func update_health(new_value):  
    number_label.text = str(new_value)  
    bar.value = new_value
```

Tip: `str` is a built-in function that converts about any value to text. Number's `text` property requires a string so we can't assign it to `new_value` directly

Also call `update_health` at the end of the `_ready` function to initialize the Number node's `text` with the right value at the start of the game. Press F5 to test the game: the life bar updates with every attack!

Fig. 2.66: Both the Number node and the TextureProgress update when the Player takes a hit

2.12.5 Animate the loss of life with the Tween node

Our interface is functional, but it could use some animation. That's a good opportunity to introduce the `Tween` node, an essential tool to animate properties. `Tween` animates anything you'd like from a start to an end state over a certain duration. For example it can animate the health on the `TextureProgress` from its current level to the Player's new health when the character takes damage.

The GUI scene already contains a `Tween` child node stored in the `tween` variable. Let's now use it. We have to make some changes to `update_health`.

We will use the `Tween` node's `interpolate_property` method. It takes seven arguments:

1. A reference to the node who owns the property to animate
2. The property's identifier as a string
3. The starting value
4. The end value
5. The animation's duration in seconds
6. The type of the transition
7. The easing to use in combination with the equation.

The last two arguments combined correspond to an *easing equation*. This controls how the value evolves from the start to the end point.

Click the script icon next to the `GUI` node to open it again. The `Number` node needs text to update itself, and the `Bar` needs a float or an integer. We can use `interpolate_property` to animate a number, but not to animate text directly. We're going to use it to animate a new `GUI` variable named `animated_health`.

At the top of the script, define a new variable and name it `animated_health`. Navigate back to the `update_health` method and clear its content. Let's animate the `animated_health` value. Call the `Tween` node's `interpolate_property` method:

```
func update_health(new_value):
    tween.interpolate_property(self, "animated_health", animated_health, new_value, 0.
    ↪6, Tween.TRANS_LINEAR, Tween.EASE_IN)
```

Let's break down the call:

```
tween.interpolate_property(self, "animated_health", ...)
```

We target `animated_health` on `self`, that is to say the `GUI` node. `Tween`'s `interpolate_property` takes the property's name as a string. That's why we write it as "`animated_health`".

```
... _health", animated_health, new_value, 0.6 ...
```

The starting point is the current value the bar's at. We still have to code this part, but it's going to be `animated_health`. The end point of the animation is the `Player`'s health after he `health_changed`: that's `new_value`. And `0.6` is the animation's duration in seconds.

```
... 0.6, tween.TRANS_LINEAR, Tween.EASE_IN)
```

The last two arguments are constants from the `Tween` class. `TRANS_LINEAR` means the animation should be linear. `EASE_IN` doesn't do anything with a linear transition, but we must provide this last argument or we'll get an error.

The animation will not play until we activated the `Tween` node with `tween.start()`. We only have to do this once if the node is not active. Add this code after the last line:

```
if not tween.is_active():
    tween.start()
```

Note: Although we could animate the `health` property on the `Player`, we really shouldn't. Characters should lose life instantly when they get hit. It makes it a lot easier to manage their state, like to know when one died. You always want to store animations in a separate data container or node. The `tween` node is perfect for code-controlled animations. For hand-made animations, check out `AnimationPlayer`.

2.12.6 Assign the `animated_health` to the LifeBar

Now the `animated_health` variable animates but we don't update the actual `Bar` and `Number` nodes anymore. Let's fix this.

So far, the `update_health` method looks like this:

```
func update_health(new_value):
    tween.interpolate_property(self, "animated_health", animated_health, new_value, 0.
    ↪6, Tween.TRANS_LINEAR, Tween.EASE_IN)
    if not Tween.is_active():
        Tween.start()
```

In this specific case, because `number_label` takes text, we need to use the `_process` method to animate it. Let's now update the `Number` and `TextureProgress` nodes like before, inside of `_process`:

```
func _process(delta):
    number_label.text = str(animated_health)
    bar.value = animated_health
```

Note: `number_label` and `bar` are variables that store references to the `Number` and `TextureProgress` nodes.

Play the game to see the bar animate smoothly. But the text displays decimal number and looks like a mess. And considering the style of the game, it'd be nice for the life bar to animate in a choppier fashion.

Fig. 2.67: The animation is smooth but the number is broken

We can fix both problems by rounding out `animated_health`. Use a local variable named `round_value` to store the rounded `animated_health`. Then assign it to `number_label.text` and `bar.value`:

```
func _process(delta):
    var round_value = round(animated_health)
    number_label.text = str(round_value)
    bar.value = round_value
```

Try the game again to see a nice blocky animation.

Fig. 2.68: By rounding out `animated_health` we hit two birds with one stone

Every time the player takes a hit, the `GUI` calls `on_Player_health_changed`, which `in` turn calls `update_health`. This updates the animation `and` the `number_label` `and` `bar` follow `in` `process`.

The animated life bar that shows the health going down gradually `is` just a trick. It `makes` the GUI feel alive. If the `Player` takes 3 damage, it happens `in` an instant.

2.12.7 Fade the bar when the Player dies

When the green character dies, it plays a death animation and fades out. At this point, we shouldn't show the interface anymore. Let's fade the bar as well when the character died. We will reuse the same Tween node as it manages multiple animations in parallel for us.

First, the GUI needs to connect to the Player's `died` signal to know when it just died. Press :kbd:F1 to jump back to the 2D Workspace. Select the `Player` node in the Scene dock and click on the Node tab next to the Inspector.

Find the `died` signal, select it, and click the Connect button.

In the Connecting Signal window, connect to the `GUI` node again. The Path to Node should be `.../.../GUI` and the Method in Node should show `_on_Player_died`. Leave the Make Function option on and click Connect at the bottom of the window. This will take you to the `GUI.gd` file in the Script Workspace.

Note: You should see a pattern by now: every time the GUI needs a new piece of information, we emit a new signal. Use them wisely: the more connections you add, the harder they are to track.

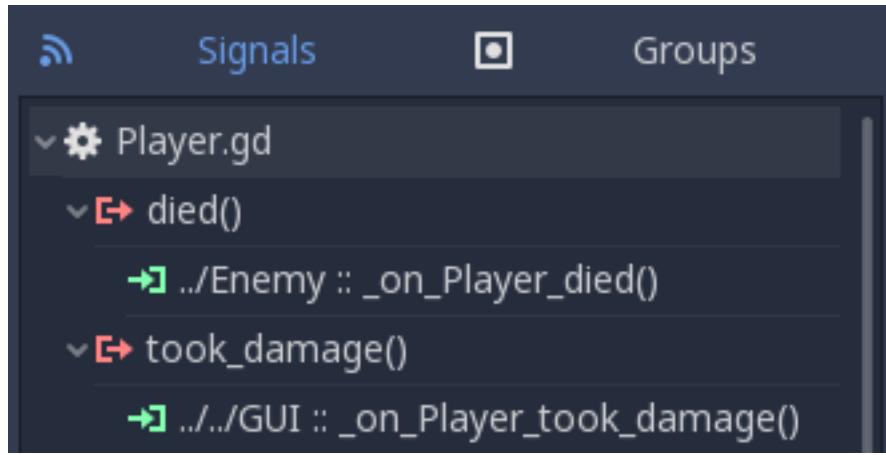


Fig. 2.69: The signal should already have the Enemy connected to it

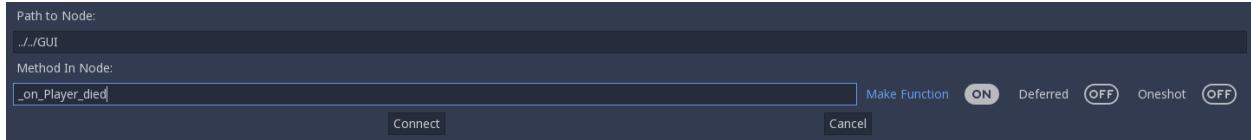


Fig. 2.70: You should get these values in the Connecting Signal window

To animate a fade on a UI element, we have to use its `modulate` property. `modulate` is a `Color` that multiplies the colors of our textures.

Note: `modulate` comes from the `CanvasItem` class, All 2D and UI nodes inherit from it. It lets you toggle the visibility of the node, assign a shader to it, and modify it using a color with `modulate`.

`modulate` takes a `Color` value with 4 channels: red, green, blue and alpha. If we darken any of the first three channels it darkens the interface. If we lower the alpha channel our interface fades out.

We're going to tween between two color values: from a white with an alpha of 1, that is to say at full opacity, to a pure white with an alpha value of 0, completely transparent. Let's add two variables at the top of the `_on_Player_died` method and name them `start_color` and `end_color`. Use the `Color()` constructor to build two `Color` values.

```
func _on_Player_died():
    var start_color = Color(1.0, 1.0, 1.0, 1.0)
    var end_color = Color(1.0, 1.0, 1.0, 0.0)
```

`Color(1.0, 1.0, 1.0)` corresponds to white. The fourth argument, respectively 1.0 and 0.0 in `start_color` and `end_color`, is the alpha channel.

We then have to call the `interpolate_property` method of the `Tween` node again:

```
Tween.interpolate_property(self, "modulate", start_color, end_color, 1.0, Tween.TRANS_
→LINEAR, Tween.EASE_IN)
```

This time we change the `modulate` property and have it animate from `start_color` to the `end_color`. The duration is of one second, with a linear transition. Here again, because the transition is linear, the easing does not matter. Here's the complete `_on_Player_died` method:

```
func _on_Player_died():
    var start_color = Color(1.0, 1.0, 1.0, 1.0)
    var end_color = Color(1.0, 1.0, 1.0, 0.0)
    tween.interpolate_property(self, "modulate", start_color, end_color, 1.0, Tween.
    ↪TRANS_LINEAR, Tween.EASE_IN)
```

And that is it. You may now play the game to see the final result!

Fig. 2.71: The final result. Congratulations for getting there!

Note: Using the exact same techniques, you can change the color of the bar when the Player gets poisoned, turn the bar red when its health drops low, shake the UI when they take a critical hit... the principle is the same: emit a signal to forward the information from the *Player* to the *GUI* and let the *GUI* process it.

2.13 Splash screen

2.13.1 Tutorial

This is a simple tutorial to establish the basic idea of how the GUI subsystem works. The goal is to create a really simple, static splash screen.

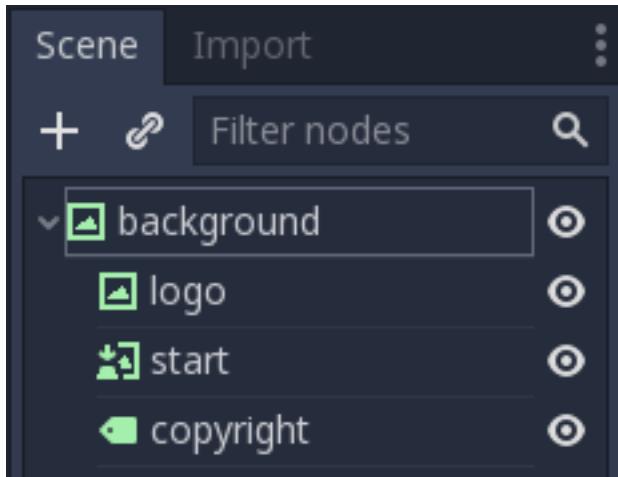


Following is a file with the assets that will be used. The extracted files can be placed directly in your project folder and Godot will import them automatically.

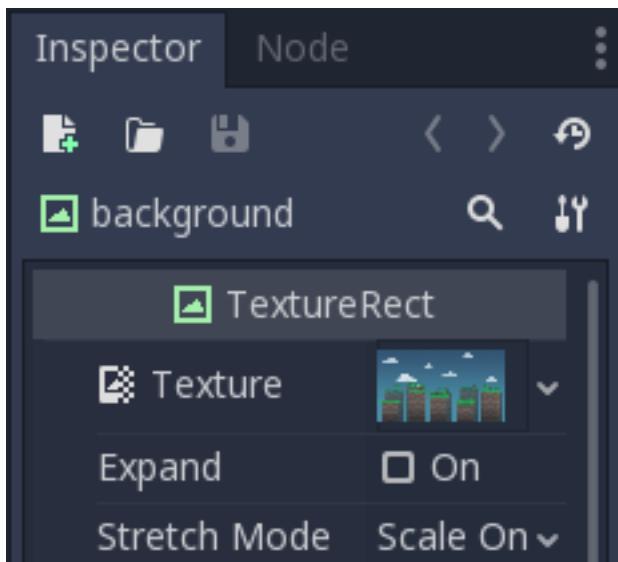
`robisplash_assets.zip`.

2.13.2 Setting up

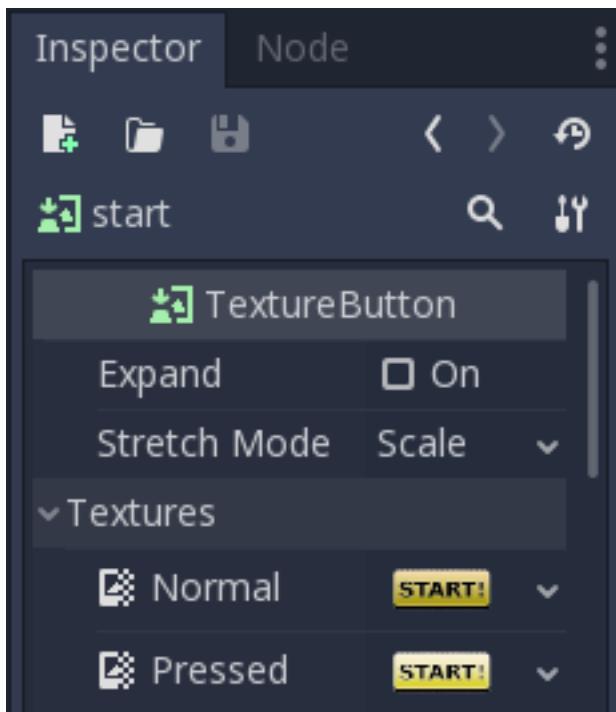
Set the display resolution to 800x450 in Project Settings, and set up a new scene like this:



The nodes “background” and “logo” are of *TextureRect* type. To display an image, just drag the corresponding asset to the texture property.

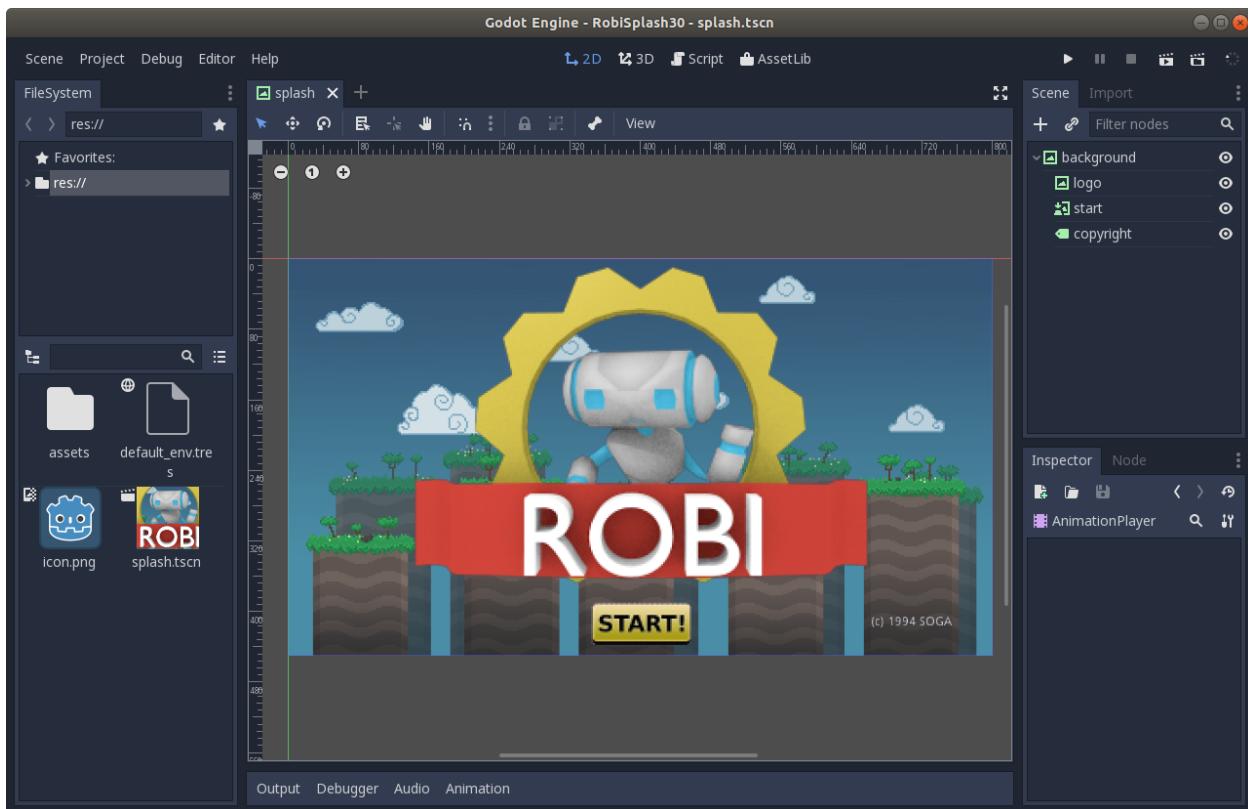


The node “start” is a *TextureButton*. It takes several images for different states, but only the normal and pressed will be supplied in this example:



Finally, the node “copyright” is a *Label*.

Your final scene should look something like this.



Go ahead and run the project. If you’re satisfied with the results, continue to the next tutorial.

2.14 Animations

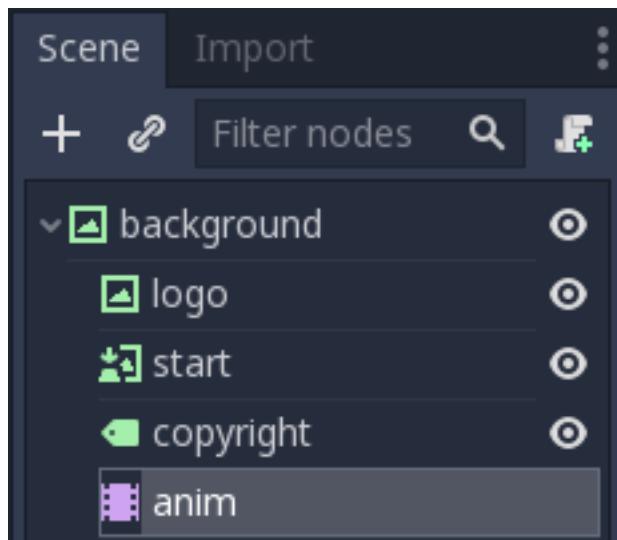
2.14.1 Introduction

Godot's animation system is extremely powerful and flexible.

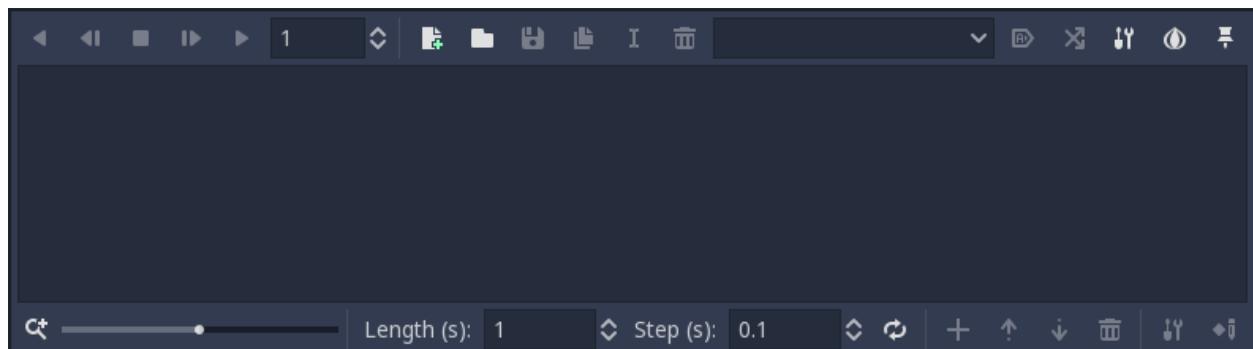
To begin, let's use the scene from the previous tutorial (*Splash screen*). The goal is to add a “fade-in” animation to the splash image. Here's a copy just in case: [robisplash.zip](#).

2.14.2 Add an animation player

First of all, add an *AnimationPlayer* node to the scene as a child of “background” (the root node):



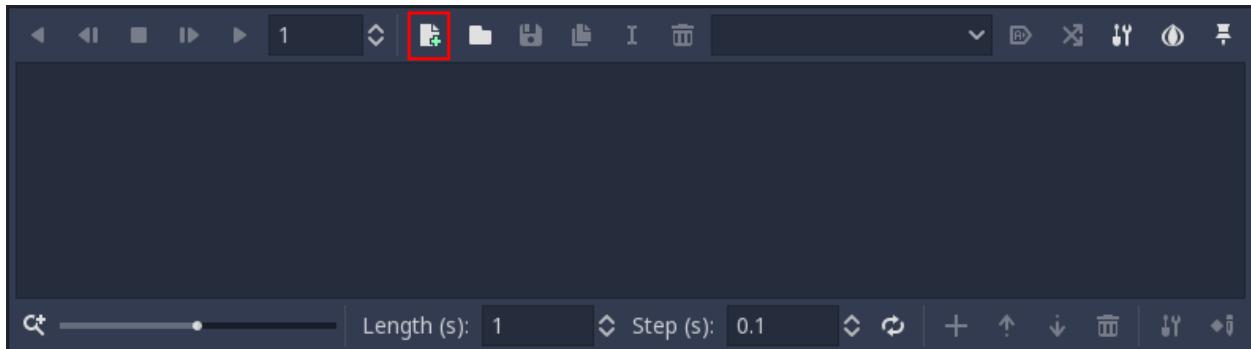
When a node of this type is selected, the animation editor panel will appear:



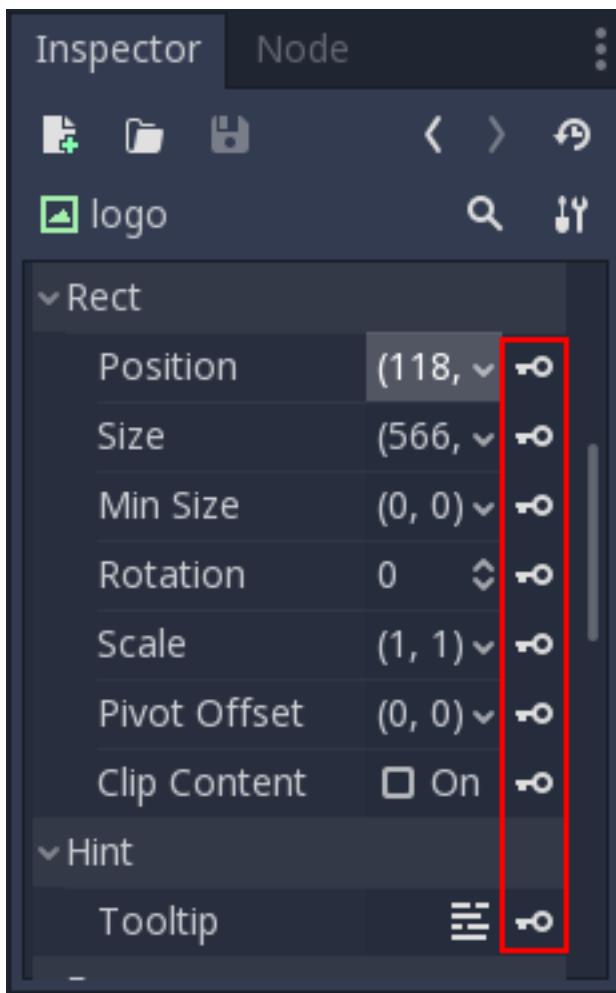
The animation editor panel stays visible until manually hidden.

2.14.3 Creating the animation

It's time to create a new animation! Press the new animation button and name the animation “intro” when the dialog appears.



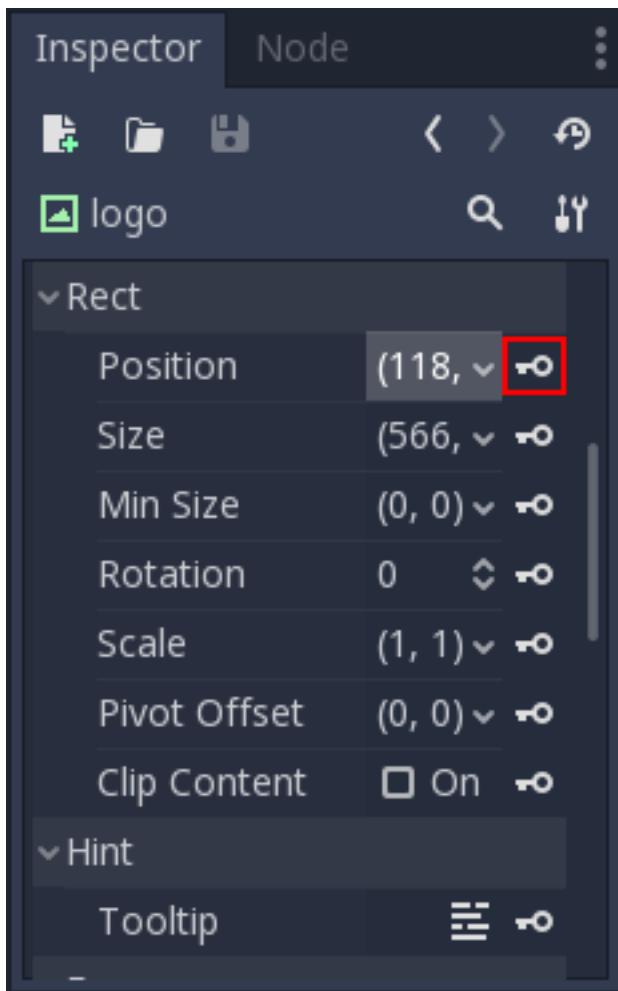
Now that we have an animation the property editor enters “animation editing” mode. In this mode, a key icon appears next to every property of the property editor. In Godot any property of an object can be animated:



2.14.4 Editing the animation

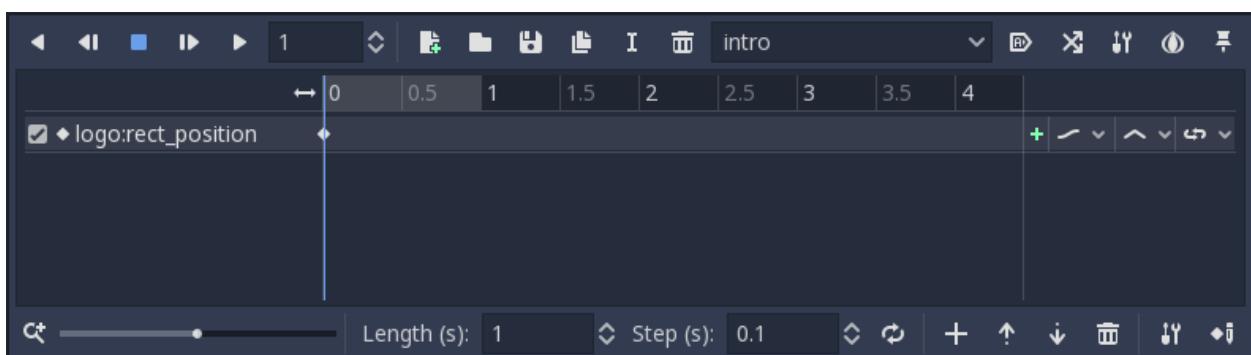
The logo will appear from the top of the screen.

With the animation editor panel open, select the “logo” node and set the “Rect / Position” property to `(118, -400)` and press the key button next to the property:

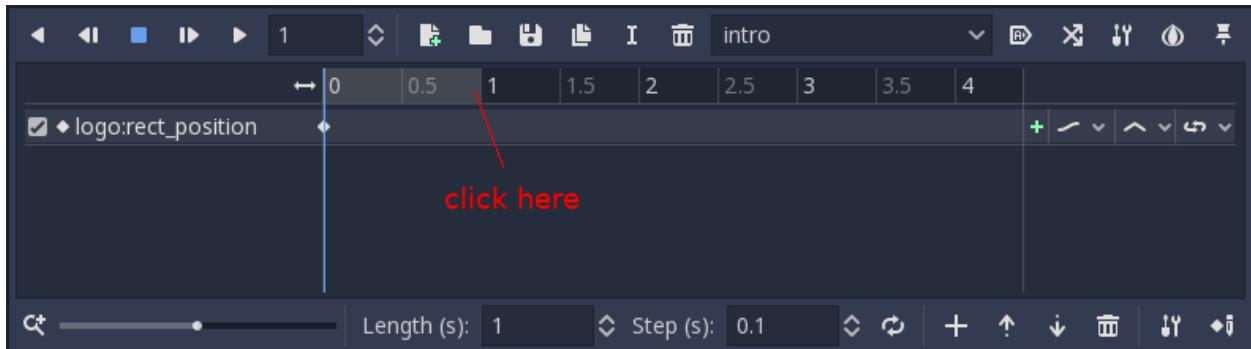


When the dialog appears, confirm that you are creating a new track.

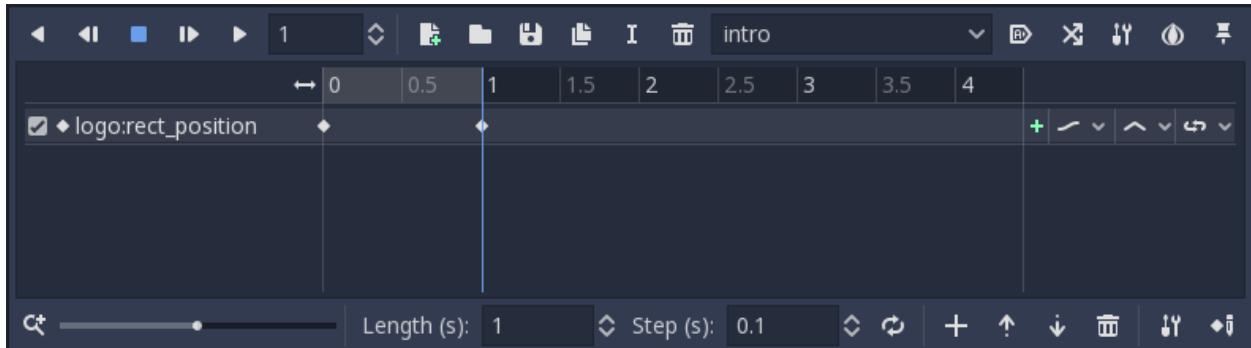
The keyframe will be added in the animation player editor:



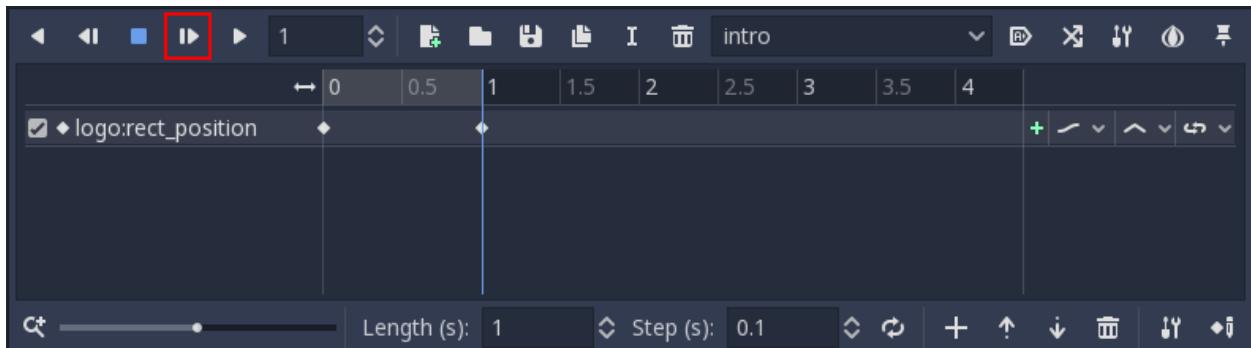
Move the editor cursor to the end, by clicking here:



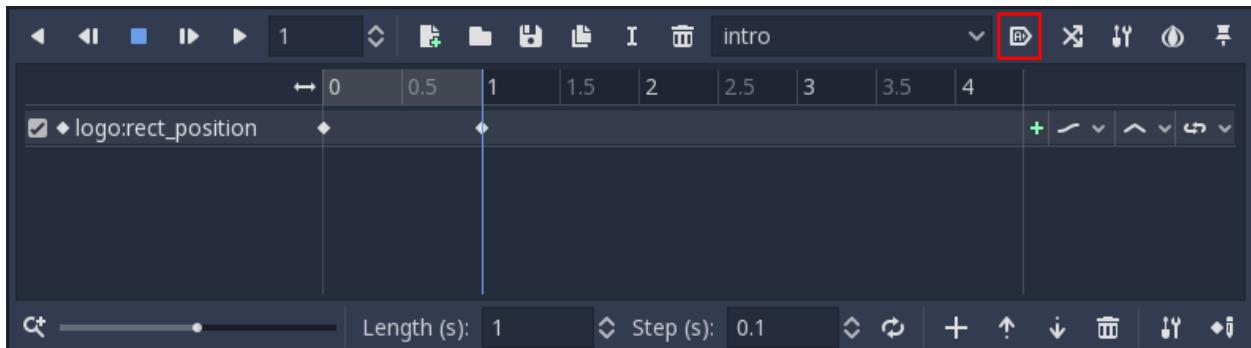
Change the logo position to `(118, 0)` and add a keyframe again. With two keyframes, the animation happens.



Pressing “Play selected animation from start. (Shift-D)” on the animation panel will make the logo descend.



Click the “Autoplay on Load” button to set the animation to start automatically when the scene starts.



And finally, when running the scene, the animation should look like this:

2.15 Resources

2.15.1 Nodes and resources

So far, *Nodes* have been the most important datatype in Godot, as most of the behaviors and features of the engine are implemented through them. There is another datatype that is equally important: *Resource*.

Where *Nodes* focus on behaviors, such as drawing a sprite, drawing a 3D model, physics, GUI controls, etc,

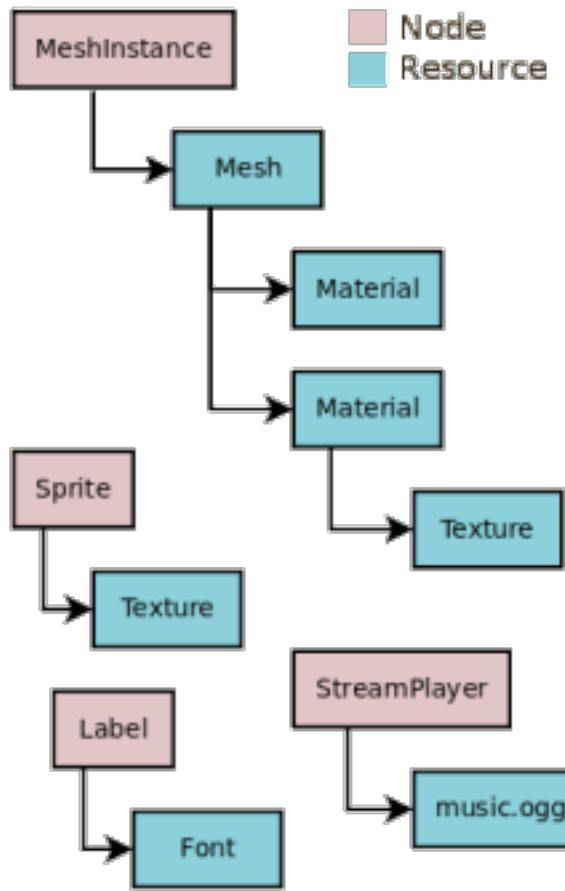
Resources are mere **data containers**. This means that they don't do any action nor process any information. Resources just contain data.

Examples of resources are *Texture*, *Script*, *Mesh*, *Animation*, *AudioStream*, *Font*, *Translation*, etc.

When Godot saves or loads (from disk) a scene (.tscn or .scn), an image (png, jpg), a script (.gd) or pretty much anything, that file is considered a resource.

When a resource is loaded from disk, **it is always loaded once**. That means, if there is a copy of that resource already loaded in memory, trying to load the resource again will just return the same copy again and again. This corresponds with the fact that resources are just data containers, so there is no need to have them duplicated.

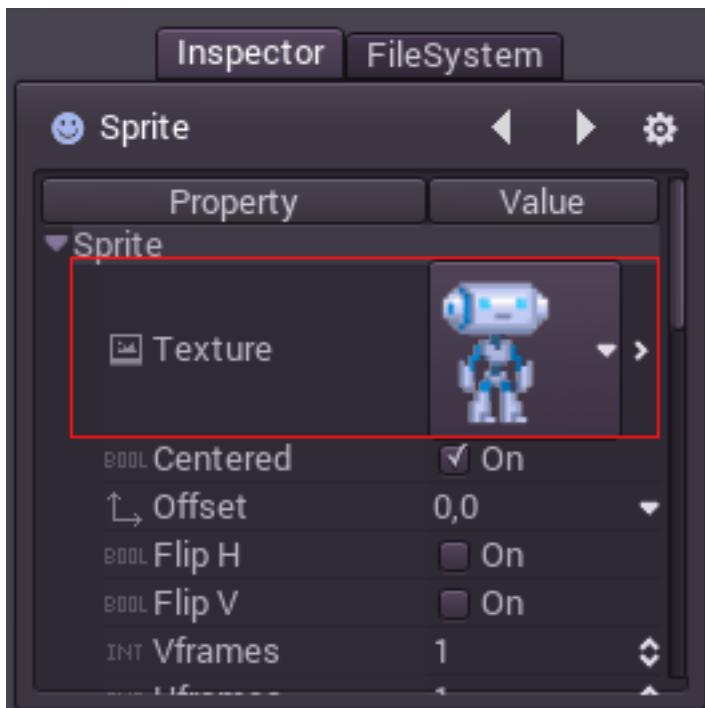
Typically, every object in Godot (Node, Resource, or anything else) can export properties, properties can be of many types (like a string, integer, Vector2, etc) and one of those types can be a resource. This means that both nodes and resources can contain resources as properties. To make it a little more visual:



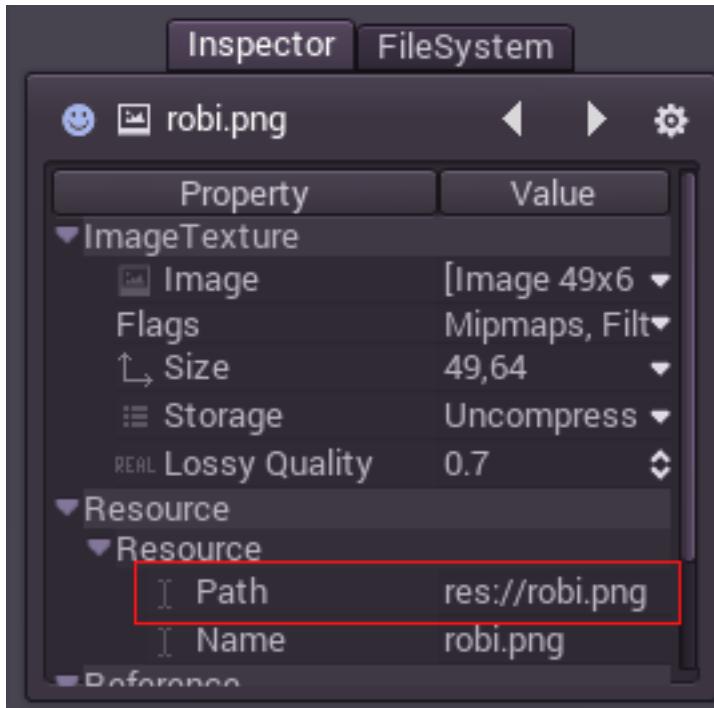
2.15.2 External vs built-in

The resource properties can reference resources in two ways, *external* (on disk) or **built-in**.

To be more specific, here's a *Texture* in a *Sprite* node:



Pressing the “>” button on the right side of the preview allows to view and edit the resources properties. One of the properties (path) shows where it comes from. In this case, it comes from a png image.



When the resource comes from a file, it is considered an *external* resource. If the path property is erased (or it never had a path to begin with), it is considered a built-in resource.

For example, if the path “`res://robi.png`” is erased from the “path” property in the above example, and then the scene is saved, the resource will be saved inside the .tscn scene file, no longer referencing the external “robi.png”. However, even if saved as built-in, and even though the scene can be instanced multiple times, the resource will always be loaded

only once. That means, different Robi robot scenes instanced at the same time will still share the same image.

2.15.3 Loading resources from code

Loading resources from code is easy. There are two ways to do it. The first is to use `load()`, like this:

```
func _ready():
    var res = load("res://robi.png") # resource is loaded when line is executed
    get_node("sprite").texture = res
```

The second way is more optimal, but only works with a string constant parameter, because it loads the resource at compile-time.

```
func _ready():
    var res = preload("res://robi.png") # resource is loaded at compile time
    get_node("sprite").texture = res
```

2.15.4 Loading scenes

Scenes are also resources, but there is a catch. Scenes saved to disk are resources of type *PackedScene*. This means that the scene is packed inside a resource.

To obtain an instance of the scene, the method *PackedScene.instance()* must be used.

```
func _on_shoot():
    var bullet = preload("res://bullet.tscn").instance()
    add_child(bullet)
```

This method creates the nodes in the scene's hierarchy, configures them (sets all the properties) and returns the root node of the scene, which can be added to any other node.

The approach has several advantages. As the *PackedScene.instance()* function is pretty fast, adding extra content to the scene can be done efficiently. New enemies, bullets, effects, etc can be added or removed quickly, without having to load them again from disk each time. It is important to remember that, as always, images, meshes, etc are all shared between the scene instances.

2.15.5 Freeing resources

Resource extends from *Reference*. As such, when a resource is no longer in use, it will automatically free itself. Since, in most cases, Resources are contained in Nodes, scripts or other resources, when a node is removed or freed, all the children resources are freed too.

2.15.6 Scripting

Like any object in Godot, not just nodes, resources can be scripted, too. However, there isn't generally much of an advantage, as resources are just data containers.

2.16 File system

2.16.1 Introduction

File systems are yet another hot topic in engine development. The file system manages how the assets are stored, and how they are accessed. A well designed file system also allows multiple developers to edit the same source files and assets while collaborating together.

Initial versions of the Godot engine (and previous iterations before it was named Godot) used a database. Assets were stored in it and assigned an ID. Other approaches were tried as well, such as local databases, files with metadata, etc. In the end the simple approach won and now Godot stores all assets as files in the file system.

2.16.2 Implementation

The file system stores resources on disk. Anything, from a script, to a scene or a PNG image is a resource to the engine. If a resource contains properties that reference other resources on disk, the paths to those resources are also included. If a resource has sub-resources that are built-in, the resource is saved in a single file together with all the bundled sub-resources. For example, a font resource is often bundled together with the font textures.

In general the Godot file system avoids using metadata files. The reason for this is simple, existing asset managers and VCSs are simply much better than anything we can implement, so Godot tries the best to play along with SVN, Git, Mercurial, Perforce, etc.

Example of a file system contents:

```
/project.godot  
/enemy/enemy.tscn  
/enemy/enemy.gd  
/enemy/enemysprite.png  
/player/player.gd
```

2.16.3 project.godot

The project.godot file is the project description file, and it is always found at the root of the project. In fact its location defines where the root is. This is the first file that Godot looks for when opening a project.

This file contains the project configuration in plain text, using the win.ini format. Even an empty project.godot can function as a basic definition of a blank project.

2.16.4 Path delimiter

Godot only supports / as a path delimiter. This is done for portability reasons. All operating systems support this, even Windows, so a path such as c:\project\project.godot needs to be typed as c:/project/project.godot.

2.16.5 Resource path

When accessing resources, using the host OS file system layout can be cumbersome and non-portable. To solve this problem, the special path res:// was created.

The path res:// will always point at the project root (where project.godot is located, so in fact res://project.godot is always valid).

This file system is read-write only when running the project locally from the editor. When exported or when running on different devices (such as phones or consoles, or running from DVD), the file system will become read-only and writing will no longer be permitted.

2.16.6 User path

Writing to disk is often still needed for various tasks such as saving game state or downloading content packs. To this end, the engine ensures that there is a special path `user://` that is always writable.

2.16.7 Host file system

Alternatively host file system paths can also be used, but this is not recommended for a released product as these paths are not guaranteed to work on all platforms. However, using host file system paths can be very useful when writing development tools in Godot!

2.16.8 Drawbacks

There are some drawbacks to this simple file system design. The first issue is that moving assets around (renaming them or moving them from one path to another inside the project) will break existing references to these assets. These references will have to be re-defined to point at the new asset location.

To avoid this, do all your move, delete and rename operations from within Godot, on the FileSystem dock. Never move assets from outside Godot, or dependencies will have to be fixed manually (Godot detects this and helps you fix them anyway, but why going the hardest route?).

The second is that under Windows and macOS file and path names are case insensitive. If a developer working in a case insensitive host file system saves an asset as “myfile.PNG”, but then references it as “myfile.png”, it will work just fine on their platform, but not on other platforms, such as Linux, Android, etc. This may also apply to exported binaries, which use a compressed package to store all files.

It is recommended that your team clearly defines a naming convention for files when working with Godot! One simple fool-proof convention is to only allow lowercase file and path names.

2.17 SceneTree

2.17.1 Introduction

This is where things start getting abstract, but don’t panic. There’s not really much more depth than this.

In previous tutorials, everything revolved around the concept of nodes. Scenes are simply a collection of nodes. They become active once they enter the *scene tree*.

This concept deserves going into a little more detail. In fact, the scene system is not even a core component of Godot, as it is possible to skip it and write a script (or C++ code) that talks directly to the servers. But making a game that way would be a lot of work.

2.17.2 MainLoop

The way Godot works internally is as follows. There is the *OS* class, which is the only instance that runs at the beginning. Afterwards, all drivers, servers, scripting languages, scene system, etc are loaded.

When initialization is complete, *OS* needs to be supplied a *MainLoop* to run. Up to this point, all this is internals working (you can check `main/main.cpp` file in the source code if you are ever interested to see how this works internally).

The user program, or game, starts in the *MainLoop*. This class has a few methods, for initialization, idle (frame-synchronized callback), fixed (physics-synchronized callback), and input. Again, this is really low level and when making games in Godot, writing your own *MainLoop* does not even make sense.

2.17.3 SceneTree

One of the ways to explain how Godot works, is that it's a high level game engine over a low level middleware.

The scene system is the game engine, while the *OS* and servers are the low level API.

In any case, the scene system provides its own main loop to *OS*, *SceneTree*. This is automatically instanced and set when running a scene, no need to do any extra work.

It's important to know that this class exists because it has a few important uses:

- It contains the root *Viewport*, to which a scene is added as a child when it's first opened, to become part of the *Scene Tree* (more on that next)
- It contains information about the groups, and has means to call all nodes in a group, or get a list of them.
- It contains some global state functionality, such as setting pause mode, or quitting the process.

When a node is part of the Scene Tree, the *SceneTree* singleton can be obtained by simply calling `Node.get_tree()`.

2.17.4 Root viewport

The root *Viewport* is always at the top of the scene. From a node, it can be obtained in two different ways:

```
get_tree().get_root() # access via scenemainloop  
get_node("/root") # access via absolute path
```

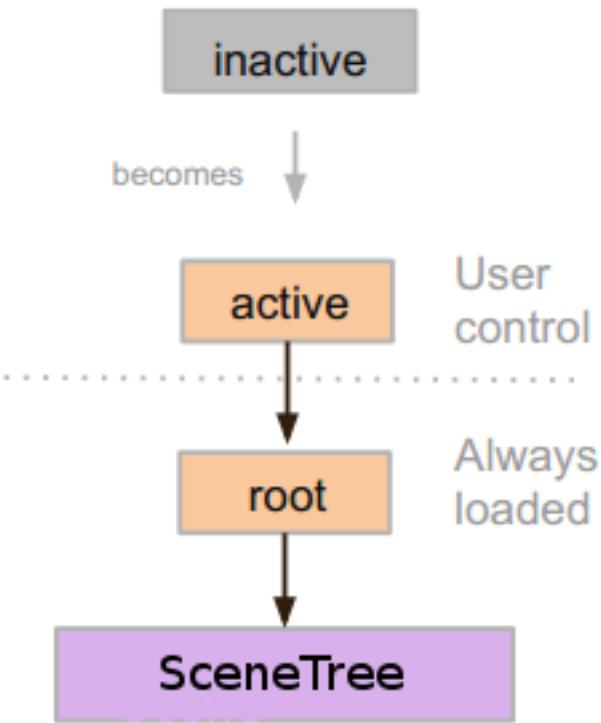
This node contains the main viewport, anything that is a child of a *Viewport* is drawn inside of it by default, so it makes sense that the top of all nodes is always a node of this type, otherwise nothing would be seen!

While other viewports can be created in the scene (for split-screen effects and such), this one is the only one that is never created by the user. It's created automatically inside *SceneTree*.

2.17.5 Scene tree

When a node is connected, directly or indirectly, to the root viewport, it becomes part of the *scene tree*.

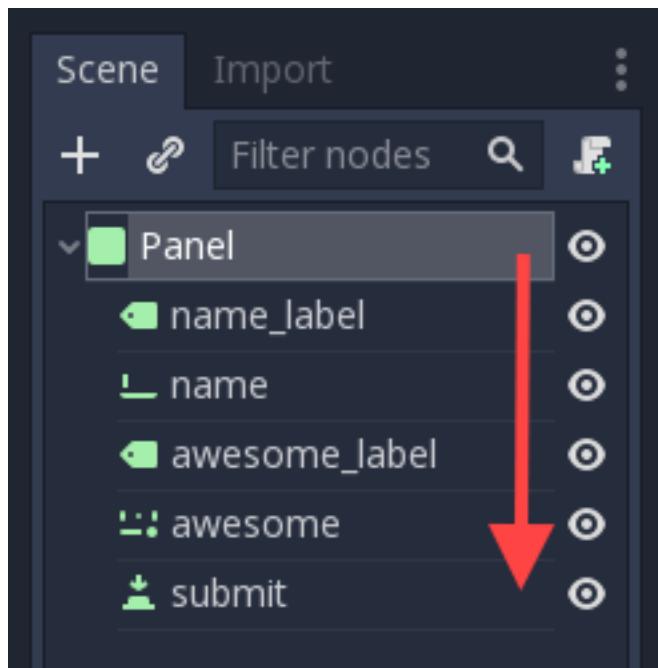
This means that, as explained in previous tutorials, it will get the `_enter_tree()` and `_ready()` callbacks (as well as `_exit_tree()`).



When nodes enter the *Scene Tree*, they become active. They get access to everything they need to process, get input, display 2D and 3D, notifications, play sound, groups, etc. When they are removed from the *scene tree*, they lose access.

2.17.6 Tree order

Most node operations in Godot, such as drawing 2D, processing or getting notifications are done in tree order. This means that parents and siblings with a smaller rank in the tree order will get notified before the current node.



2.17.7 “Becoming active” by entering the *Scene Tree*

1. A scene is loaded from disk or created by scripting.
2. The root node of that scene (only one root, remember?) is added as either a child of the “root” Viewport (from `SceneTree`), or to any child or grand-child of it.
3. Every node of the newly added scene, will receive the “enter_tree” notification (`_enter_tree()` callback in GDScript) in top-to-bottom order.
4. An extra notification, “ready” (`_ready()` callback in GDScript) is provided for convenience, when a node and all its children are inside the active scene.
5. When a scene (or part of it) is removed, they receive the “exit scene” notification (`_exit_tree()` callback in GDScript) in bottom-to-top order

2.17.8 Changing current scene

After a scene is loaded, it is often desired to change this scene for another one. The simple way to do this is to use the `SceneTree.change_scene()` function:

```
func _my_level_was_completed():
    get_tree().change_scene("res://levels/level2.tscn")
```

This is a quick and useful way to switch scenes, but has the drawback that the game will stall until the new scene is loaded and running. At some point in your game, it may be desired to create proper loading screens with progress bar, animated indicators or thread (background) loading. This must be done manually using autoloads (see next chapter!) and *Background loading*.

2.18 Singletons (AutoLoad)

2.18.1 Introduction

Scene singletons are very useful, catering to a common use case where you need to store persistent information between scenes.

Albeit very powerful, the scene system by itself has a few drawbacks:

- There is no common place to store information (e.g. a player's items etc.) required by more than one scene.
- While it is possible for a scene that loads and unloads other scenes as its children to store information common to these child scenes, it is no longer possible to run these scenes by themselves and expect them to work correctly.
- While information can be stored to disk in ‘user://’ and this information can be loaded by scenes that require it, continuously saving and loading this data when changing scenes is cumbersome and may be slow.

However there is still a need in Godot to create parts of a scene that:

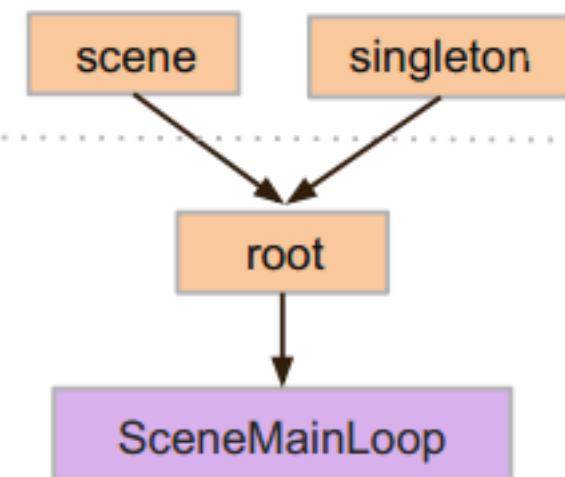
- Are always loaded, no matter which scene is opened from the editor
- Can store global variables, such as player information, items, money etc. and share information between scenes
- Can handle switching scenes and transitions
- Acts like a singleton, since GDScript does not support global variables by design.

Auto-loading nodes and scripts caters to this need.

2.18.2 AutoLoad

You can use AutoLoad to load a scene, or a script that inherits from Node (a node will be created and the script will be set to it).

To autoload a scene or script, select Project > Project Settings from the menu and switch to the AutoLoad tab. Each entry in the list requires a name, which is used as the name of the node, and the node is always added to the root viewport before any other scenes are loaded.



This means that any node can access a singleton named “playervariables” with:

```
var player_vars = get_node("/root/playervariables")
player_vars.health
```

Or even simpler using the name directly:

```
playervariables.health
```

2.18.3 Custom scene switcher

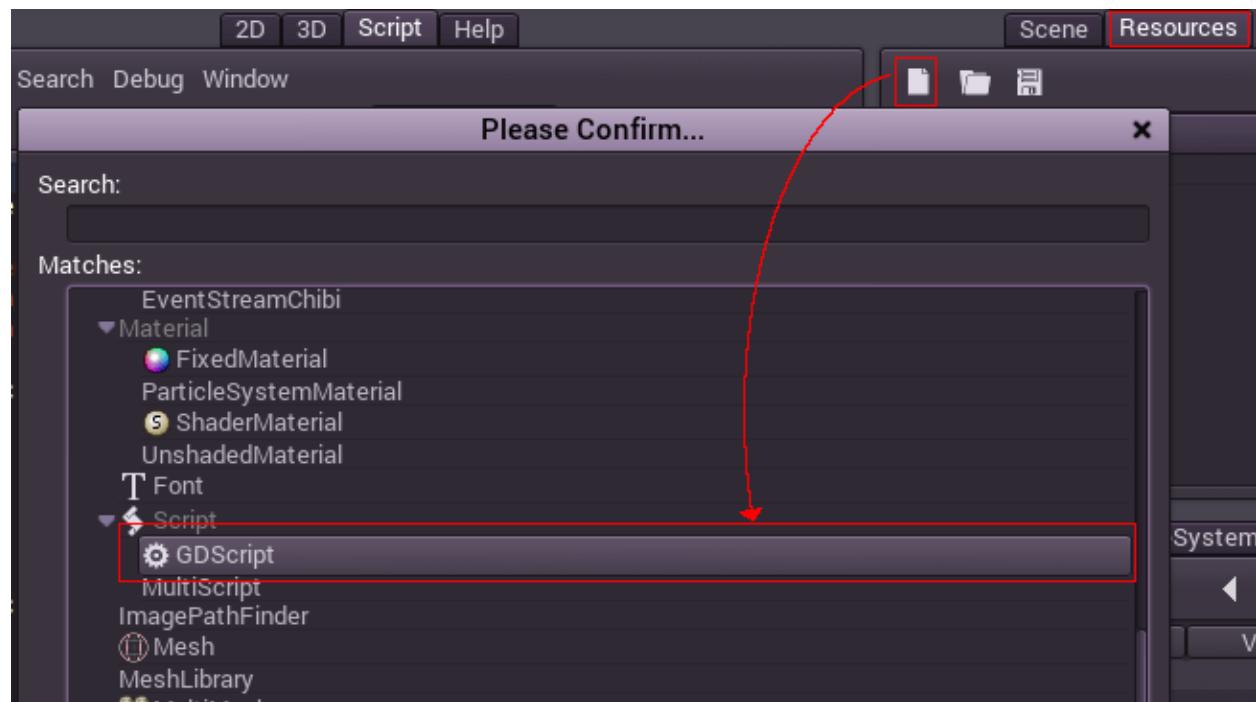
This short tutorial will explain how to make a scene switcher using autoload. For simple scene switching, the *SceneTree.change_scene()* method suffices (described in *SceneTree*), so this method is for more complex behavior when switching between scenes.

First download the template from here: [autoload.zip](#), then open it.

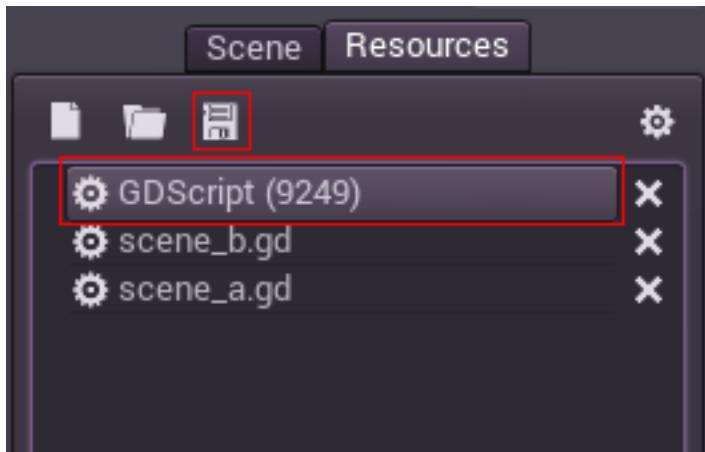
Two scenes are present, *scene_a.tscn* and *scene_b.tscn* on an otherwise empty project. Each are identical and contain a button connected to a callback for switching to the other scene. When the project runs, it starts in *scene_a.tscn*. However, this currently does nothing and pressing the button does not work.

2.18.4 global.gd

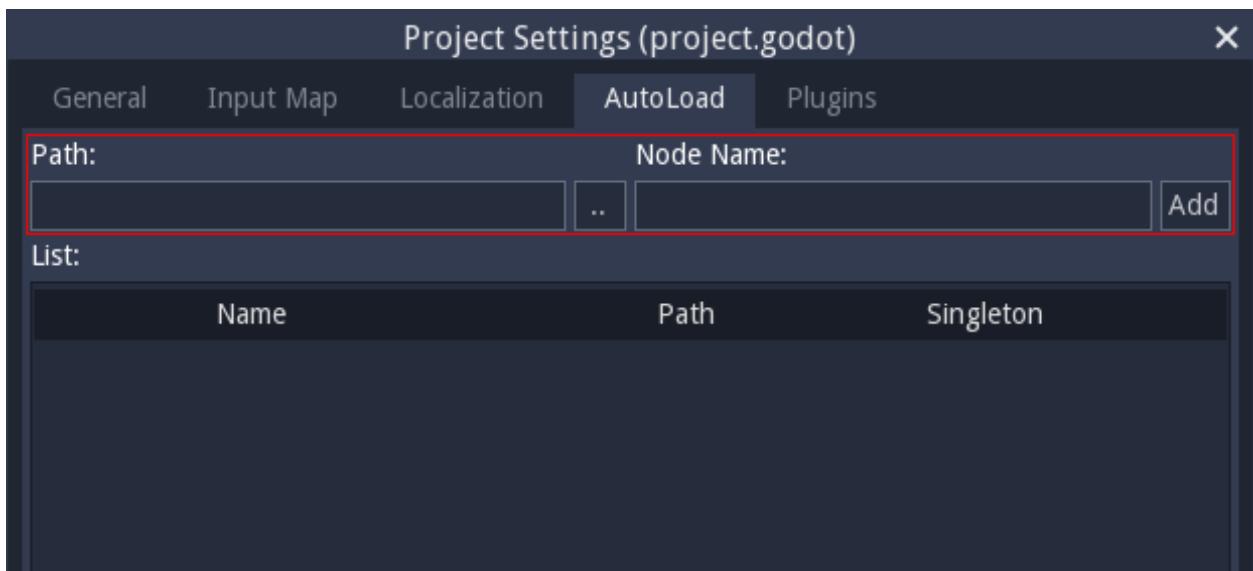
First of all, create a *global.gd* script. The easy way to create a resource from scratch is from the new resource button in the inspector tab:



Save the script as *global.gd*:



The script should open in the script editor. The next step is to add it to AutoLoad list. Select Project > Project Settings from the menu, switch to the AutoLoad tab and add a new entry with name “global” that points to this file:



Now, whenever you run any of your scenes, the script is always loaded.

Returning to our script, the current scene needs to be fetched in the `_ready()` function. Both the current scene and `global.gd` are children of root, but the autoloaded nodes are always first. This means that the last child of root is always the loaded scene.

Note: Make sure that `global.gd` extends `Node`, otherwise it won't be loaded!

```
extends Node

var current_scene = null

func _ready():
    var root = get_tree().get_root()
    current_scene = root.get_child( root.get_child_count() -1 )
```

Next up is the function for changing the scene. This function frees the current scene and replaces it with the requested one.

```

func goto_scene(path):

    # This function will usually be called from a signal callback,
    # or some other function from the running scene.
    # Deleting the current scene at this point might be
    # a bad idea, because it may be inside of a callback or function of it.
    # The worst case will be a crash or unexpected behavior.

    # The way around this is deferring the load to a later time, when
    # it is ensured that no code from the current scene is running:

    call_deferred("_deferred_goto_scene", path)

func _deferred_goto_scene(path):

    # Immediately free the current scene,
    # there is no risk here.
    current_scene.free()

    # Load new scene
    var s = ResourceLoader.load(path)

    # Instance the new scene
    current_scene = s.instance()

    # Add it to the active scene, as child of root
    get_tree().get_root().add_child(current_scene)

    # optional, to make it compatible with the SceneTree.change_scene() API
    get_tree().set_current_scene( current_scene )

```

As mentioned in the comments above, we really want to avoid the situation of having the current scene being deleted while being used (code from functions of it being run), so using *Object.call_deferred()* is desired at this point. The result is that execution of the commands in the second function will happen at a later time when no code from the current scene is running.

Finally, all that is left is to fill the empty functions in scene_a.gd and scene_b.gd:

```

#add to scene_a.gd

func _on_goto_scene_pressed():
    get_node("/root/global").goto_scene("res://scene_b.tscn")

```

and

```

#add to scene_b.gd

func _on_goto_scene_pressed():
    get_node("/root/global").goto_scene("res://scene_a.tscn")

```

Now if you run the project, you can switch between both scenes by pressing the button!

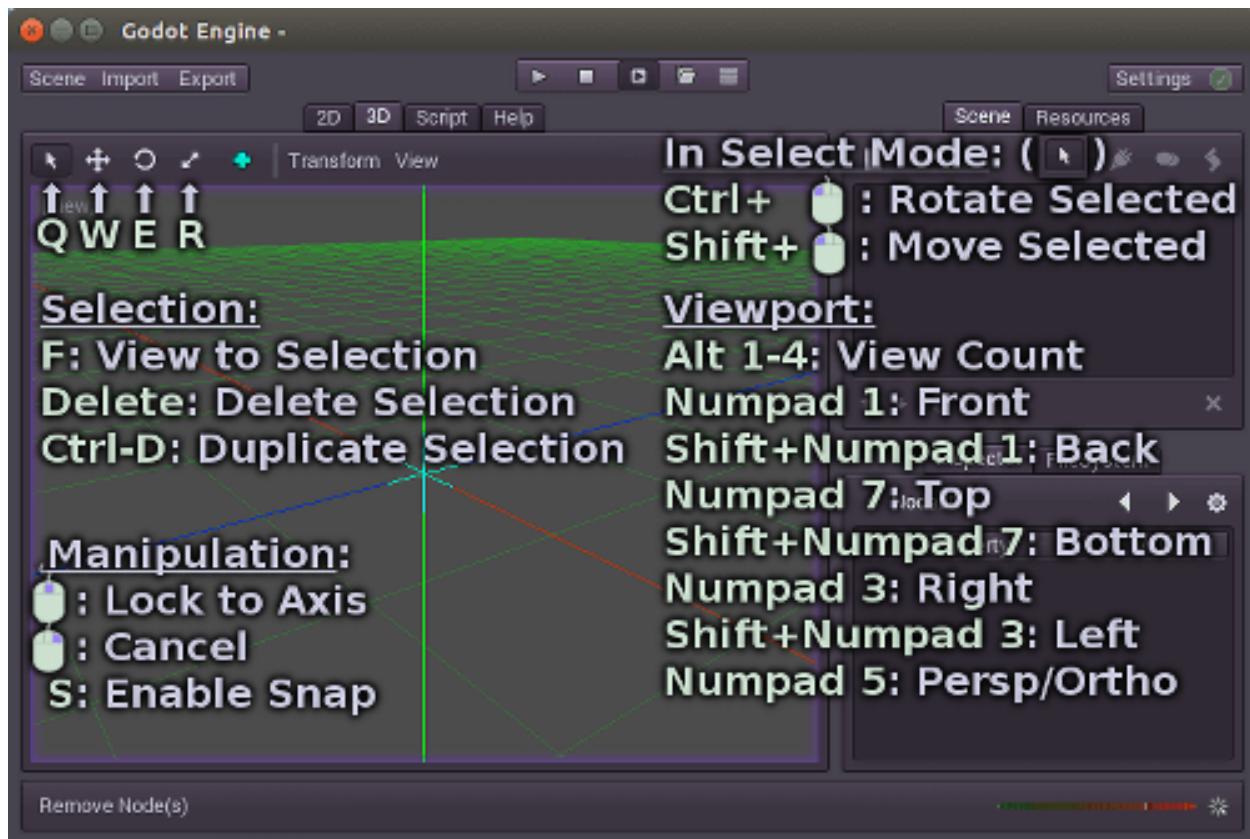
To load scenes with a progress bar, check out the next tutorial, *Background loading*

3.1 2D and 3D keybindings

3.1.1 2D viewport



3.1.2 3D viewport



3.2 From Unity3D to Godot Engine

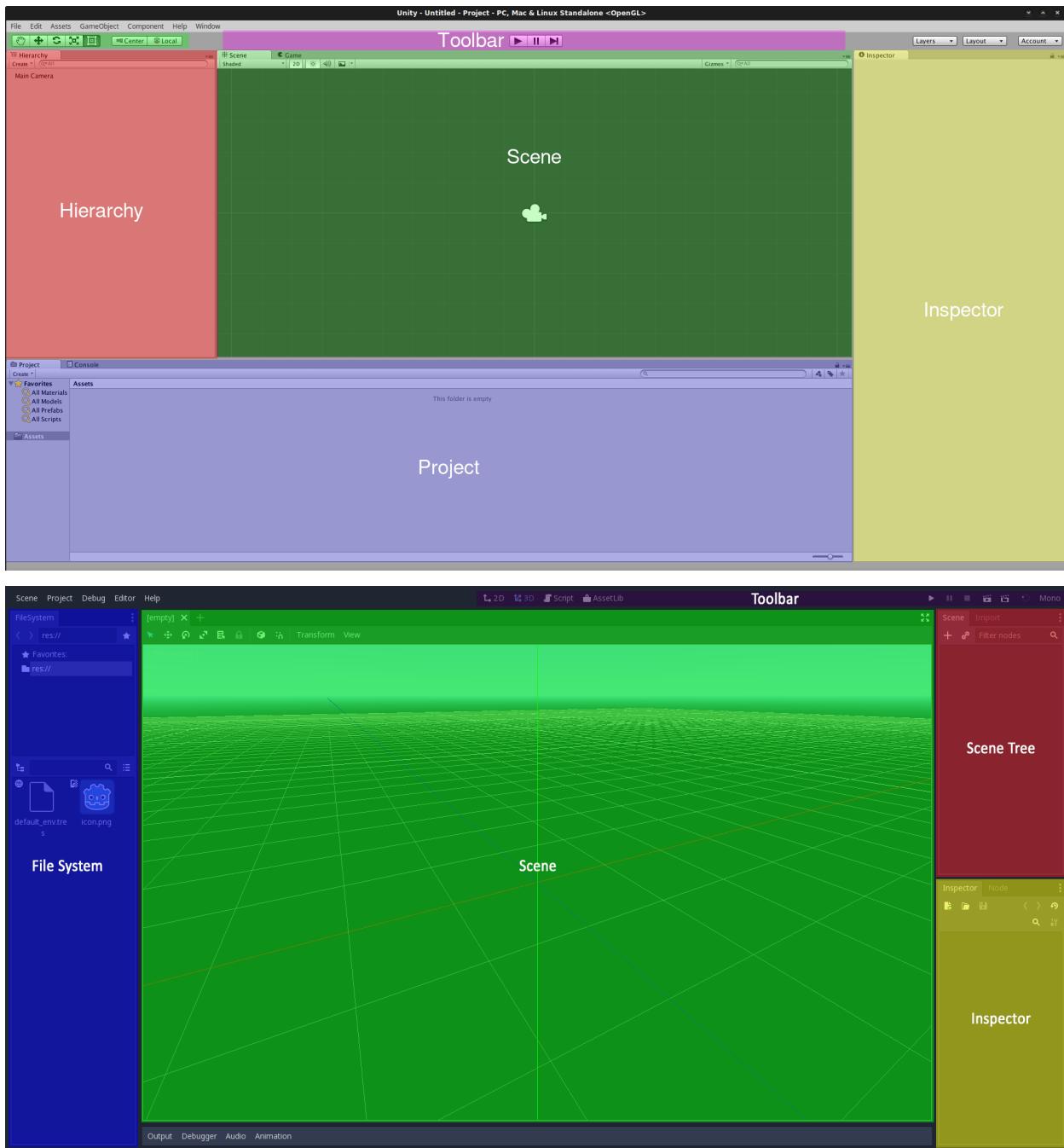
This guide provides an overview of Godot Engine from the viewpoint of a Unity user, and aims to help you migrate your existing Unity experience into the world of Godot.

3.2.1 Differences

	Unity	Godot
License	Proprietary, closed, free license with revenue caps and usage restrictions	MIT license, free and fully open source without any restriction
OS (editor)	Windows, macOS, Linux (unofficial and unsupported)	Windows, X11 (Linux, *BSD), Haiku, macOS
OS (export)	Desktop: Windows, Linux/SteamOS, macOS Mobile: Android, iOS, Windows Phone, Tizen, Web: WebAssembly or asm.js Consoles: PS4, PS Vita, Xbox One, Xbox 360, Wii U, Nintendo 3DS VR: Oculus Rift, SteamVR, Google Cardboard, Playstation VR, Gear VR, HoloLens TV: Android TV, Samsung SMART TV, tvOS	Desktop: Windows, X11, macOS Mobile: Android, iOS, Web: WebAssembly
Scene system	Component/Scene (GameObject > Component) Prefabs	Scene tree and nodes, allowing scenes to be nested and/or inherit other scenes
Third-party tools	Visual Studio or SharpEditor	Android SDK for Android export External editors are possible
Killer features	Huge community Large assets store	Scene System Animation Pipeline Easy to write Shaders Debug on Device

3.2.2 The editor

Godot Engine provides a rich-featured editor that allows you to build your games. The pictures below display both editors with colored blocks to indicate common functionalities.



Note that Godot editor allows you to dock each panel at the side of the scene editor you wish.

While both editors may seem similar, there are many differences below the surface. Both let you organize the project using the filesystem, but Godot approach is simpler, with a single configuration file, minimalist text format, and no metadata. All this contributes to Godot being much friendlier to VCS systems such as Git, Subversion or Mercurial.

Godot's Scene panel is similar to Unity's Hierarchy panel but, as each node has a specific function, the approach used by Godot is more visually descriptive. In other words, it's easier to understand what a specific scene does at a glance.

The Inspector in Godot is more minimalist and designed to only show properties. Thanks to this, objects can export a much larger amount of useful parameters to the user, without having to hide functionality in language APIs. As a plus, Godot allows animating any of those properties visually, so changing colors, textures, enumerations or even links to resources in real-time is possible without involving code.

Finally, the Toolbar at the top of the screen is similar in the sense that it allows controlling the project playback, but projects in Godot run in a separate window, as they don't execute inside the editor (but the tree and objects can still be explored in the debugger window).

This approach has the disadvantage that the running game can't be explored from different angles (though this may be supported in the future, and displaying collision gizmos in the running game is already possible), but in exchange has several advantages:

- Running the project and closing it is very fast (Unity has to save, run the project, close the project and then reload the previous state).
- Live editing is a lot more useful, because changes done to the editor take effect immediately in the game, and are not lost (nor have to be synced) when the game is closed. This allows fantastic workflows, like creating levels while you play them.
- The editor is more stable, because the game runs in a separate process.

Finally, the top toolbar includes a menu for remote debugging. These options make it simple to deploy to a device (connected phone, tablet or browser via HTML5), and debug/live edit on it after the game was exported.

3.2.3 The scene system

This is the most important difference between Unity and Godot, and actually the favourite feature of most Godot users.

Unity's scene system consist in embedding all the required assets in a scene, and link them together by setting components and scripts to them.

Godot's scene system is different: it actually consists in a tree made of nodes. Each node serves a purpose: Sprite, Mesh, Light... Basically, this is similar to Unity scene system. However, each node can have multiple children, which make each a subscene of the main scene. This means you can compose a whole scene with different scenes, stored in different files.

For example, think of a platformer level. You would compose it with multiple elements:

- Bricks
- Coins
- The player
- The enemies

In Unity, you would put all the GameObjects in the scene: the player, multiple instances of enemies, bricks everywhere to form the ground of the level, and multiple instances of coins all over the level. You would then add various components to each element to link them and add logic in the level: for example, you'd add a BoxCollider2D to all the elements of the scene so that they can collide. This principle is different in Godot.

In Godot, you would split your whole scene into 3 separate, smaller scenes, which you would then instance in the main scene.

1. First, a scene for the Player alone.

Consider the player as a reusable element in other levels. It is composed of one node in particular: an AnimatedSprite node, which contains the sprite textures to form various animations (for example, walking animation)

2. Second, a scene for the Enemy.

There again, an enemy is a reusable element in other levels. It is almost the same as the Player node - the only differences are the script (that manages AI, mostly) and sprite textures used by the AnimatedSprite.

3. Lastly, the Level scene.

It is composed of Bricks (for platforms), Coins (for the player to grab) and a certain number of instances of the previous Enemy scene. These will be different, separate enemies, whose behaviour and appearance will be the same as defined in the Enemy scene. Each instance is then considered as a node in the Level scene tree. Of course, you can set different properties for each enemy node (to change its color for example).

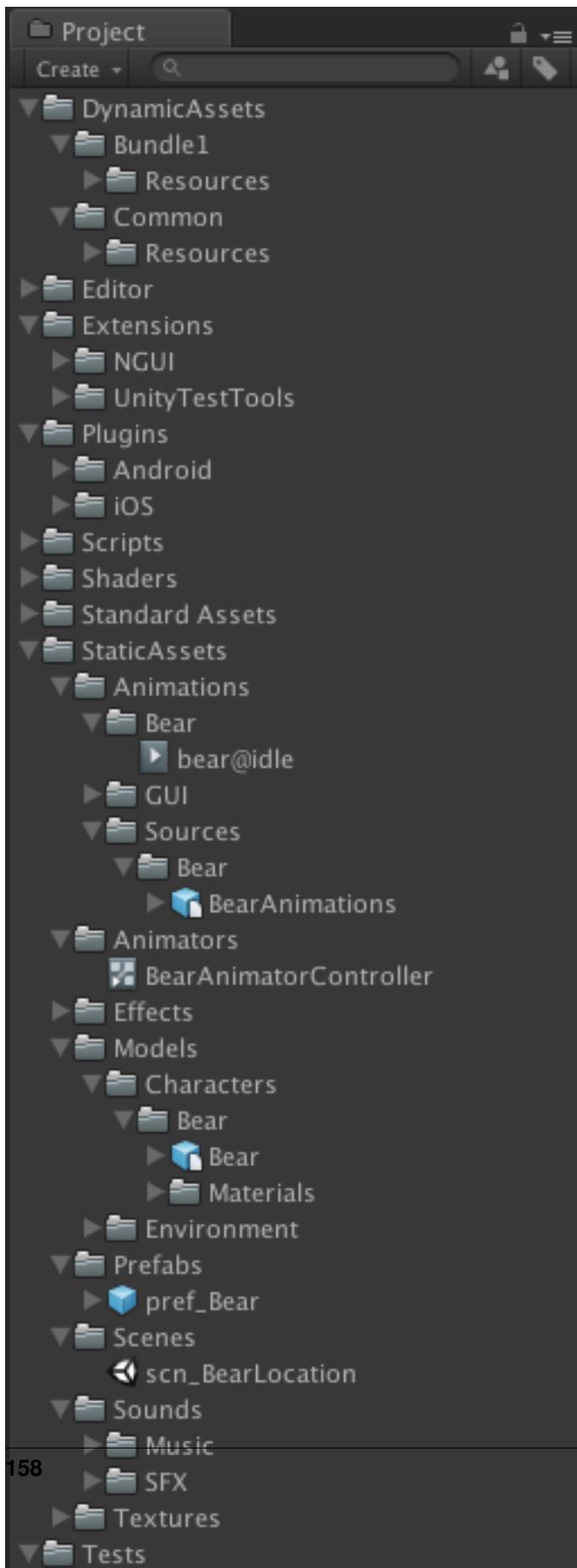
Finally, the main scene would then be composed of one root node with 2 children: a Player instance node, and a Level instance node. The root node can be anything, generally a “root” type such as “Node” which is the most global type, or “Node2D” (root type of all 2D-related nodes), “Spatial” (root type of all 3D-related nodes) or “Control” (root type of all GUI-related nodes).

As you can see, every scene is organized as a tree. The same goes for nodes’ properties: you don’t *add* a collision component to a node to make it collidable like Unity does. Instead, you make this node a *child* of a new specific node that has collision properties. Godot features various collision types nodes, depending of the use (see the *Physics introduction*).

- Question: What are the advantages of this system? Wouldn’t this system potentially increase the depth of the scene tree? Besides, Unity allows organizing GameObjects by putting them in empty GameObjects.
 - First, this system is closer to the well-known Object-Oriented paradigm: Godot provides a number of nodes which are not clearly “Game Objects”, but they provide their children with their own capabilities: this is inheritance.
 - Second, it allows the extraction a subtree of scene to make it a scene of its own, which answers to the second and third questions: even if a scene tree gets too deep, it can be split into smaller subtrees. This also allows a better solution for reusability, as you can include any subtree as a child of any node. Putting multiple nodes in an empty GameObject in Unity does not provide the same possibility, apart from a visual organization.

These are the most important concepts you need to remember: “node”, “parent node” and “child node”.

3.2.4 Project organization



We previously observed that there is no perfect solution to set a project architecture. Any solution will work for Unity and Godot, so this point has a lesser importance.

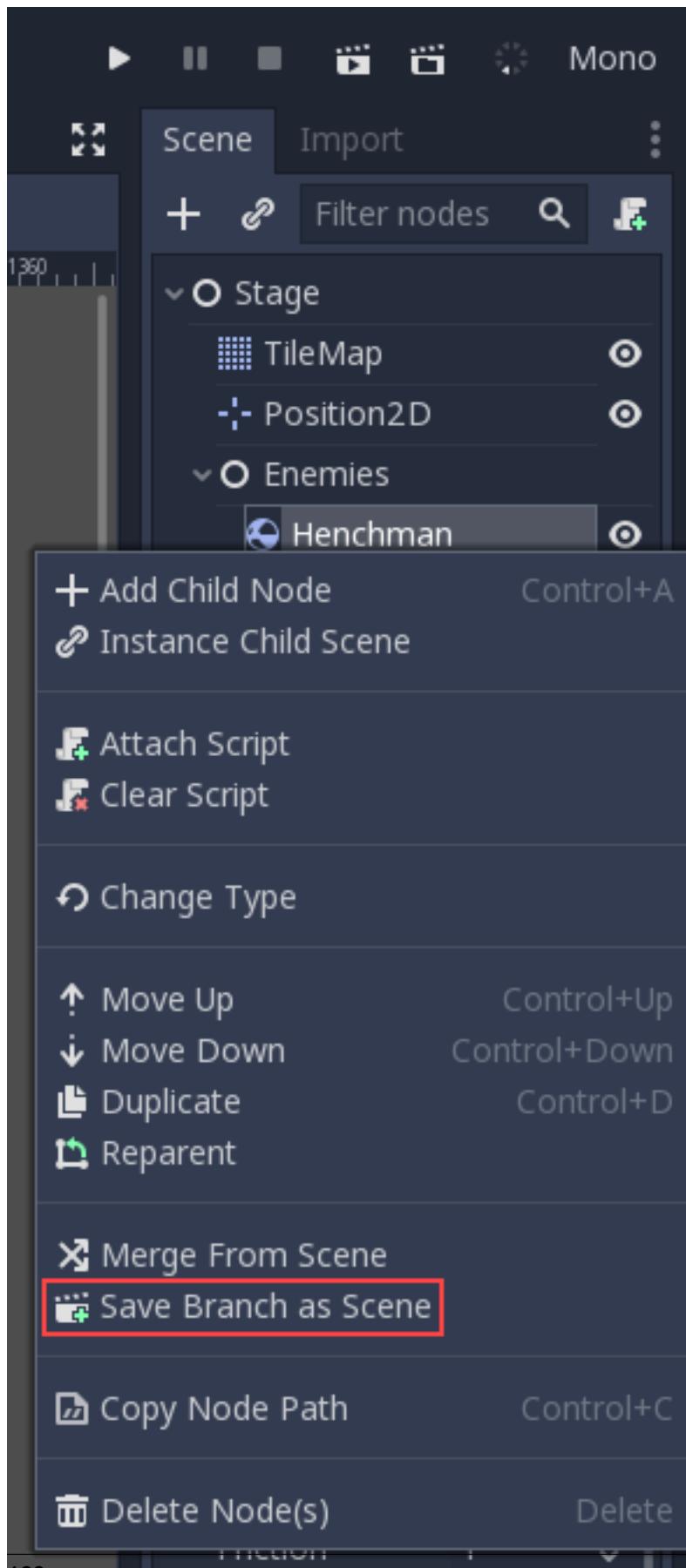
However, we often observe a common architecture for Unity projects, which consists in having one Assets folder in the root directory, that contains various folders, one per type of asset: Audio, Graphics, Models, Materials, Scripts, Scenes, etc.

As described before, Godot scene system allows splitting scenes in smaller scenes. Since each scene and subscene is actually one scene file in the project, we recommend organizing your project a bit differently. This wiki provides a page for this: *Project organization*.

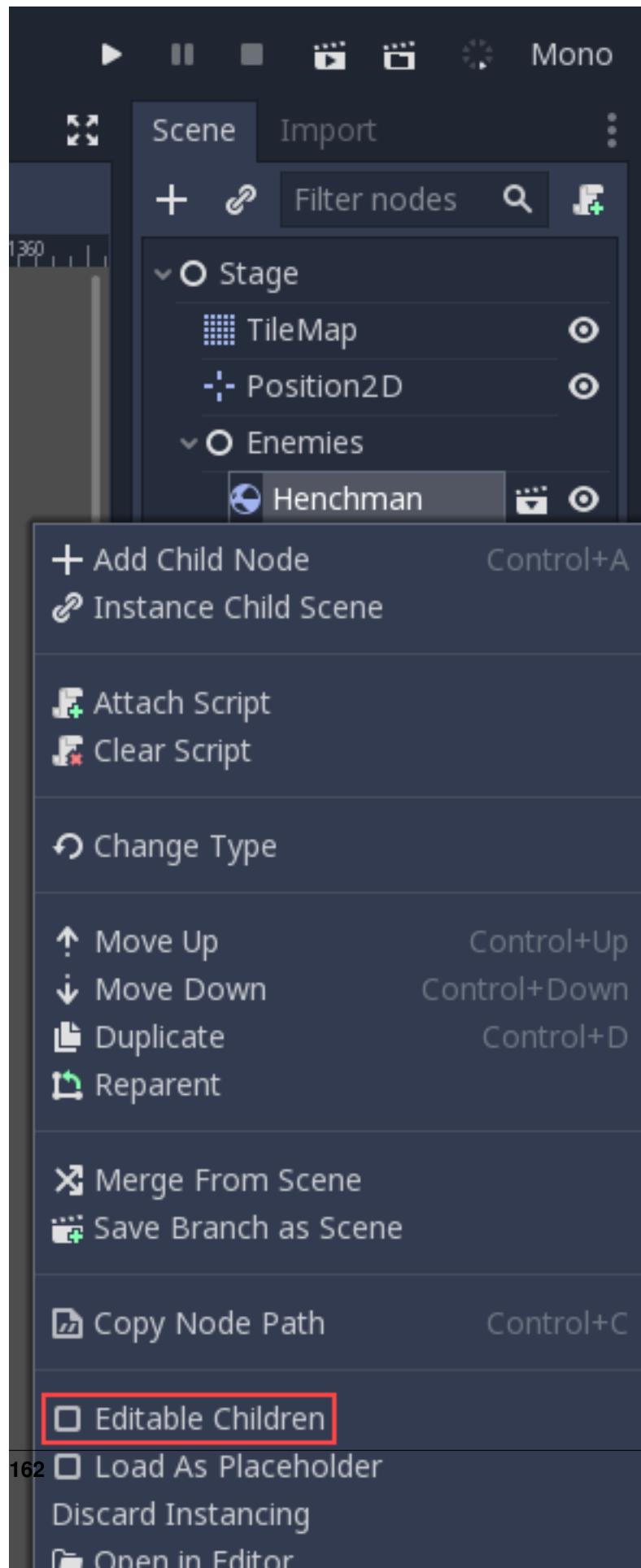
3.2.5 Where are my prefabs?

The concept of prefabs as provided by Unity is a ‘template’ element of the scene. It is reusable, and each instance of the prefab that exists in the scene has an existence of its own, but all of them have the same properties as defined by the prefab.

Godot does not provide prefabs as such, but this functionality is here again filled thanks to its scene system: as we saw the scene system is organized as a tree. Godot allows you to save a subtree of a scene as its own scene, thus saved in its own file. This new scene can then be instanced as many times as you want. Any change you make to this new, separate scene will be applied to its instances. However, any change you make to the instance will not have any impact on the ‘template’ scene.



To be precise, you can modify the parameters of the instance in the Inspector panel. However, the nodes that compose this instance are locked and you can unlock them if you need to by right clicking the instance in the Scene tree, and selecting “Editable children” in the menu. You don’t need to do this to add new children nodes to this node, but remember that these new children will belong to the instance, not the ‘template’ scene. If you want to add new children to all the instances of your ‘template’ scene, then you need to add it once in the ‘template’ scene.



3.2.6 Glossary correspondence

GameObject -> Node Add a component -> Inheriting Prefab -> Externalized branch

3.2.7 Scripting: GDScript, C# and Visual Script

Design

As you may know already, Unity supports C#. C# benefits from its integration with Visual Studio and other features, such as static typing.

Godot provides its own scripting language, *GDScript* as well as support for *Visual Script* and *Introduction*. GDScript borrows its syntax from Python, but is not related to it. If you wonder about the reasoning for a custom scripting language, please read *GDScript* and [FAQ](#) pages. GDScript is strongly attached to the Godot API, but it is really easy to learn: between one evening for an experienced programmer and a week for a complete beginner.

Unity allows you to attach as many scripts as you want to a GameObject. Each script adds a behaviour to the GameObject: for example, you can attach a script so that it reacts to the player's controls, and another that controls its specific game logic.

In Godot, you can only attach one script per node. You can use either an external GDScript file, or include it directly in the node. If you need to attach more scripts to one node, then you may consider two solutions, depending on your scene and on what you want to achieve:

- either add a new node between your target node and its current parent, then add a script to this new node.
- or, you can split your target node into multiple children and attach one script to each of them.

As you can see, it can be easy to turn a scene tree to a mess. This is why it is important to have a real reflection, and consider splitting a complicated scene into multiple, smaller branches.

Connections : groups and signals

You can control nodes by accessing them using a script, and call functions (built-in or user-defined) on them. But there's more: you can also place them in a group and call a function on all nodes contained in this group! This is explained in [this page](#).

But there's more! Certain nodes throw signals when certain actions happen. You can connect these signals to call a specific function when they happen. Note that you can define your own signals and send them whenever you want. This feature is documented [here](#).

3.2.8 Using Godot in C++

Just for your information, Godot also allows you to develop your project directly in C++ by using its API, which is not possible with Unity at the moment. As an example, you can consider Godot Engine's editor as a "game" written in C++ using Godot API.

If you are interested in using Godot in C++, you may want to start reading the *Developing in C++* page.

3.3 Command line tutorial

Some developers like using the command line extensively. Godot is designed to be friendly to them, so here are the steps for working entirely from the command line. Given the engine relies on little to no external libraries, initialization times are pretty fast, making it suitable for this workflow.

3.3.1 Path

It is recommended that your godot binary is in your PATH environment variable, so it can be executed easily from any place by typing `godot`. You can do so on Linux by placing the Godot binary in `/usr/local/bin` and making sure it is called `godot`.

3.3.2 Creating a project

Creating a project from the command line is simple, just navigate the shell to the desired place and just make `project.godot` file exist, even if empty.

```
user@host:~$ mkdir newgame
user@host:~$ cd newgame
user@host:~/newgame$ touch project.godot
```

That alone makes for an empty Godot project.

3.3.3 Running the editor

Running the editor is done by executing `godot` with the `-e` flag. This must be done from within the project directory, or a subdirectory, otherwise the command is ignored and the project manager appears.

```
user@host:~/newgame$ godot -e
```

If a scene has been created and saved, it can be edited later by running the same code with that scene as argument.

```
user@host:~/newgame$ godot -e scene.tscn
```

3.3.4 Erasing a scene

Godot is friends with your filesystem, and will not create extra metadata files, simply use `rm` to erase a file. Make sure nothing references that scene, or else an error will be thrown upon opening.

```
user@host:~/newgame$ rm scene.tscn
```

3.3.5 Running the game

To run the game, simply execute Godot within the project directory or subdirectory.

```
user@host:~/newgame$ godot
```

When a specific scene needs to be tested, pass that scene to the command line.

```
user@host:~/newgame$ godot scene.tscn
```

3.3.6 Debugging

Catching errors in the command line can be a difficult task because they just fly by. For this, a command line debugger is provided by adding `-d`. It works for both running the game or a simple scene.

```
user@host:~/newgame$ godot -d
```

```
user@host:~/newgame$ godot -d scene.tscn
```

3.3.7 Exporting

Exporting the project from the command line is also supported. This is especially useful for continuous integration setups. The version of Godot that is headless (server build, no video) is ideal for this.

```
user@host:~/newgame$ godot -export "Linux X11" /var/builds/project
user@host:~/newgame$ godot -export Android /var/builds/project.apk
```

The platform names recognized by the `-export` switch are the same as displayed in the export wizard of the editor. To get a list of supported platforms from the command line, just try exporting to a non-recognized platform and the full listing of platforms your configuration supports will be shown.

To export a debug version of the game, use the `-export_debug` switch instead of `-export`. Their parameters and usage are the same.

3.3.8 Running a script

It is possible to run a simple .gd script from the command line. This feature is especially useful in very large projects, for batch conversion of assets or custom import/export.

The script must inherit from SceneTree or MainLoop.

Here is a simple example of how it works:

```
#sayhello.gd
extends SceneTree

func _init():
    print("Hello!")
    quit()
```

And how to run it:

```
user@host:~/newgame$ godot -s sayhello.gd
Hello!
```

If no engine.cfg exists at the path, current path is assumed to be the current working directory (unless `-path` is specified).

CHAPTER 4

Scripting

4.1 GDScript

4.1.1 GDScript

Introduction

GDScript is a high level, dynamically typed programming language used to create content. It uses a syntax similar to [Python](#) (blocks are indent-based and many keywords are similar). Its goal is to be optimized for and tightly integrated with Godot Engine, allowing great flexibility for content creation and integration.

History

In the early days, the engine used the [Lua](#) scripting language. Lua is fast, but creating bindings to an object oriented system (by using fallbacks) was complex and slow and took an enormous amount of code. After some experiments with [Python](#), it also proved difficult to embed.

The last third party scripting language that was used for shipped games was [Squirrel](#), but it was dropped as well. At that point, it became evident that a custom scripting language could more optimally make use of Godot's particular architecture:

- Godot embeds scripts in nodes. Most languages are not designed with this in mind.
- Godot uses several built-in data types for 2D and 3D math. Script languages do not provide this, and binding them is inefficient.
- Godot uses threads heavily for lifting and initializing data from the net or disk. Script interpreters for common languages are not friendly to this.
- Godot already has a memory management model for resources, most script languages provide their own, which results in duplicate effort and bugs.
- Binding code is always messy and results in several failure points, unexpected bugs and generally low maintainability.

The result of these considerations is *GDScript*. The language and interpreter for GDScript ended up being smaller than the binding code itself for Lua and Squirrel, while having equal functionality. With time, having a built-in language has proven to be a huge advantage.

Example of GDScript

Some people can learn better by just taking a look at the syntax, so here's a simple example of how GDScript looks.

```
# a file is a class!

# inheritance

extends BaseClass

# member variables

var a = 5
var s = "Hello"
var arr = [1, 2, 3]
var dict = {"key": "value", 2:3}

# constants

const answer = 42
const thename = "Charly"

# enums

enum {UNIT_NEUTRAL, UNIT_ENEMY, UNIT_ALLY}
enum Named {THING_1, THING_2, ANOTHER_THING = -1}

# built-in vector types

var v2 = Vector2(1, 2)
var v3 = Vector3(1, 2, 3)

# function

func some_function(param1, param2):
    var local_var = 5

    if param1 < local_var:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("fail!")

    for i in range(20):
        print(i)

    while(param2 != 0):
        param2 -= 1

    var local_var2 = param1+3
    return local_var2
```

```
# inner class

class Something:
    var a = 10

# constructor

func _init():
    print("constructed!")
    var lv = Something.new()
    print(lv.a)
```

If you have previous experience with statically typed languages such as C, C++, or C# but never used a dynamically typed one before, it is advised you read this tutorial: *GDScript: An introduction to dynamic languages*.

Language

In the following, an overview is given to GDScript. Details, such as which methods are available to arrays or other objects, should be looked up in the linked class descriptions.

Identifiers

Any string that restricts itself to alphabetic characters (a to z and A to Z), digits (0 to 9) and _ qualifies as an identifier. Additionally, identifiers must not begin with a digit. Identifiers are case-sensitive (foo is different from FOO).

Keywords

The following is the list of keywords supported by the language. Since keywords are reserved words (tokens), they can't be used as identifiers.

Keyword	Description
if	See if/else/elif .
elif	See if/else/elif .
else	See if/else/elif .
for	See for .
do	Reserved for future implementation of do...while loops.
while	See while .
match	See match .
switch	Reserved for future implementation.
case	Reserved for future implementation.
break	Exits the execution of the current <code>for</code> or <code>while</code> loop.
continue	Immediately skips to the next iteration of the <code>for</code> or <code>while</code> loop.
pass	Used where a statement is required syntactically but execution of code is undesired, e.g. in empty functions.
return	Returns a value from a function.
class	Defines a class.
extends	Defines what class to extend with the current class.
is	Tests whether a variable extends a given class.
tool	Executes the script in the editor.
signal	Defines a signal.
func	Defines a function.
static	Defines a static function. Static member variables are not allowed.
const	Defines a constant.
enum	Defines an enum.
var	Defines a variable.
onready	Initializes a variable once the Node the script is attached to and its children are part of the scene tree.
export	Saves a variable along with the resource it's attached to and makes it visible and modifiable in the editor.
setget	Defines setter and getter functions for a variable.
break-point	Editor helper for debugger breakpoints.

Operators

The following is the list of supported operators and their precedence (TODO, change since this was made to reflect python operators)

Operator	Description
x [index]	Subscription, Highest Priority
x . attribute	Attribute Reference
is	Instance Type Checker
~	Bitwise NOT
-x	Negative
* / %	Multiplication / Division / Remainder NOTE: The result of these operations depends on the operands types. If both are Integers, then the result will be an Integer. That means 1/10 returns 0 instead of 0.1. If at least one of the operands is a float, then the result is a float: float(1)/10 or 1.0/10 return both 0.1.
+ -	Addition / Subtraction
<< >>	Bit Shifting
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
< > == !	Comparisons
= >= <=	
in	Content Test
! not	Boolean NOT
and &&	Boolean AND
or	Boolean OR
if x else	Ternary if/else
= += -= *= /= %= &= =	Assignment, Lowest Priority

Literals

Literal	Type
45	Base 10 integer
0x8F51	Base 16 (hex) integer
3.14, 58.1e-10	Floating point number (real)
"Hello", "Hi"	Strings
"""Hello, Dude"""	Multiline string
@ "Node/Label"	NodePath or StringName

Comments

Anything from a # to the end of the line is ignored and is considered a comment.

```
# This is a comment
```

Built-in types

Basic built-in types

A variable in GDScript can be assigned to several built-in types.

null

`null` is an empty data type that contains no information and can not be assigned any other value.

bool

The Boolean data type can only contain `true` or `false`.

int

The integer data type can only contain integer numbers, (both negative and positive).

float

Used to contain a floating point value (real numbers).

String

A sequence of characters in [Unicode](#) format. Strings can contain the [standard C escape sequences](#). GDScript supports *format strings aka printf functionality*.

Vector built-in types

Vector2

2D vector type containing `x` and `y` fields. Can also be accessed as array.

Rect2

2D Rectangle type containing two vectors fields: `position` and `size`. Alternatively contains an `end` field which is `position+size`.

Vector3

3D vector type containing `x`, `y` and `z` fields. This can also be accessed as an array.

Transform2D

3x2 matrix used for 2D transforms.

Plane

3D Plane type in normalized form that contains a `normal` vector field and a `d` scalar distance.

Quat

Quaternion is a datatype used for representing a 3D rotation. It's useful for interpolating rotations.

AABB

Axis-aligned bounding box (or 3D box) contains 2 vectors fields: `position` and `size`. Alternatively contains an `end` field which is `position+size`.

Basis

3x3 matrix used for 3D rotation and scale. It contains 3 vector fields (`x`, `y` and `z`) and can also be accessed as an array of 3D vectors.

Transform

3D Transform contains a `Basis` field `basis` and a `Vector3` field `origin`.

Engine built-in types

Color

Color data type contains `r`, `g`, `b`, and `a` fields. It can also be accessed as `h`, `s`, and `v` for hue/saturation/value.

NodePath

Compiled path to a node used mainly in the scene system. It can be easily assigned to, and from, a String.

RID

Resource ID (RID). Servers use generic RIDs to reference opaque data.

Object

Base class for anything that is not a built-in type.

Container built-in types

Array

Generic sequence of arbitrary object types, including other arrays or dictionaries (see below). The array can resize dynamically. Arrays are indexed starting from index 0. Starting with Godot 2.1, indices may be negative like in Python, to count from the end.

```
var arr=[]
arr=[1, 2, 3]
var b = arr[1]           # this is 2
var c = arr[arr.size()-1] # this is 3
var d = arr[-1]          # same as the previous line, but shorter
arr[0] = "Hi!"           # replacing value 1 with "Hi"
arr.append(4)             # array is now ["Hi", 2, 3, 4]
```

GDScript arrays are allocated linearly in memory for speed. Very large arrays (more than tens of thousands of elements) may however cause memory fragmentation. If this is a concern special types of arrays are available. These only accept a single data type. They avoid memory fragmentation and also use less memory but are atomic and tend to run slower than generic arrays. They are therefore only recommended to use for very large data sets:

- *PoolByteArray*: An array of bytes (integers from 0 to 255).
- *PoolIntArray*: An array of integers.
- *PoolRealArray*: An array of floats.
- *PoolStringArray*: An array of strings.
- *PoolVector2Array*: An array of *Vector2* objects.
- *PoolVector3Array*: An array of *Vector3* objects.
- *PoolColorArray*: An array of *Color* objects.

Dictionary

Associative container which contains values referenced by unique keys.

```
var d={4:5, "a key":"a value", 28:[1,2,3]}
d["Hi!"] = 0
d = {
    22           : "Value",
    "somekey"    : 2,
    "otherkey"   : [2,3,4],
    "morekey"    : "Hello"
}
```

Lua-style table syntax is also supported. Lua-style uses = instead of : and doesn't use quotes to mark string keys (making for slightly less to write). Note however that like any GDScript identifier, keys written in this form cannot start with a digit.

```
var d = {
    test22 = "Value",
    somekey = 2,
    otherkey = [2,3,4],
    morekey = "Hello"
}
```

To add a key to an existing dictionary, access it like an existing key and assign to it:

```
var d = {} # create an empty Dictionary
d.Waiting = 14 # add String "Waiting" as a key and assign the value 14 to it
d[4] = "hello" # add integer `4` as a key and assign the String "hello" as its value
d["Godot"] = 3.01 # add String "Godot" as a key and assign the value 3.01 to it
```

Data

Variables

Variables can exist as class members or local to functions. They are created with the `var` keyword and may, optionally, be assigned a value upon initialization.

```
var a # data type is null by default
var b = 5
var c = 3.8
var d = b + c # variables are always initialized in order
```

Constants

Constants are similar to variables, but must be constants or constant expressions and must be assigned on initialization.

```
const a = 5
const b = Vector2(20, 20)
const c = 10 + 20 # constant expression
const d = Vector2(20, 30).x # constant expression: 20
const e = [1, 2, 3, 4][0] # constant expression: 1
const f = sin(20) # sin() can be used in constant expressions
const g = x + 20 # invalid; this is not a constant expression!
```

Enums

Enums are basically a shorthand for constants, and are pretty useful if you want to assign consecutive integers to some constant.

If you pass a name to the enum, it would also put all the values inside a constant dictionary of that name.

```
enum {TILE_BRICK, TILE_FLOOR, TILE_SPIKE, TILE_TELEPORT}
# Is the same as:
const TILE_BRICK = 0
const TILE_FLOOR = 1
const TILE_SPIKE = 2
const TILE_TELEPORT = 3

enum State {STATE_IDLE, STATE_JUMP = 5, STATE_SHOOT}
# Is the same as:
const STATE_IDLE = 0
const STATE_JUMP = 5
const STATE_SHOOT = 6
const State = {STATE_IDLE = 0, STATE_JUMP = 5, STATE_SHOOT = 6}
```

Functions

Functions always belong to a *class*. The scope priority for variable look-up is: local → class member → global. The `self` variable is always available and is provided as an option for accessing class members, but is not always required (and should *not* be sent as the function's first argument, unlike Python).

```
func myfunction(a, b):
    print(a)
    print(b)
    return a + b # return is optional; without it null is returned
```

A function can `return` at any point. The default return value is `null`.

Referencing Functions

To call a function in a *base class* (i.e. one `extend`-ed in your current class), prepend `.` to the function name:

```
.basefunc(args)
```

Contrary to Python, functions are *not* first class objects in GDScript. This means they cannot be stored in variables, passed as an argument to another function or be returned from other functions. This is for performance reasons.

To reference a function by name at runtime, (e.g. to store it in a variable, or pass it to another function as an argument) one must use the `call` or `funcref` helpers:

```
# Call a function by name in one step
mynode.call("myfunction", args)

# Store a function reference
var myfunc = funcref(mynode, "myfunction")
# Call stored function reference
myfunc.call_func(args)
```

Remember that default functions like `_init`, and most notifications such as `_enter_tree`, `_exit_tree`, `_process`, `_physics_process`, etc. are called in all base classes automatically. So there is only a need to call the function explicitly when overloading them in some way.

Static functions

A function can be declared static. When a function is static it has no access to the instance member variables or `self`. This is mainly useful to make libraries of helper functions:

```
static func sum2(a, b):
    return a + b
```

Statements and control flow

Statements are standard and can be assignments, function calls, control flow structures, etc (see below). `;` as a statement separator is entirely optional.

if/else/elif

Simple conditions are created by using the `if/else/elif` syntax. Parenthesis around conditions are allowed, but not required. Given the nature of the tab-based indentation, `elif` can be used instead of `else/if` to maintain a level of indentation.

```
if [expression]:
    statement(s)
elif [expression]:
    statement(s)
else:
    statement(s)
```

Short statements can be written on the same line as the condition:

```
if (1 + 1 == 2): return 2 + 2
else:
    var x = 3 + 3
    return x
```

Sometimes you might want to assign a different initial value based on a boolean expression. In this case ternary-if expressions come in handy:

```
var x = [true-value] if [expression] else [false-value]
y += 3 if y < 10 else -1
```

while

Simple loops are created by using `while` syntax. Loops can be broken using `break` or continued using `continue`:

```
while [expression]:
    statement(s)
```

for

To iterate through a range, such as an array or table, a `for` loop is used. When iterating over an array, the current array element is stored in the loop variable. When iterating over a dictionary, the `index` is stored in the loop variable.

```
for x in [5, 7, 11]:
    statement # loop iterates 3 times with x as 5, then 7 and finally 11

var dict = {"a":0, "b":1, "c":2}
for i in dict:
    print(dict[i]) # loop provides the keys in an arbitrary order; may print 0, 1, 2,
    ↵ or 2, 0, 1, etc...

for i in range(3):
    statement # similar to [0, 1, 2] but does not allocate an array

for i in range(1,3):
    statement # similar to [1, 2] but does not allocate an array

for i in range(2,8,2):
    statement # similar to [2, 4, 6] but does not allocate an array
```

```
for c in "Hello":
    print(c)      # iterate through all characters in a String, print every letter on
    ↪new line
```

match

A `match` statement is used to branch execution of a program. It's the equivalent of the `switch` statement found in many other languages but offers some additional features.

Basic syntax:

```
match [expression]:
    [pattern](s): [block]
    [pattern](s): [block]
    [pattern](s): [block]
```

Crash-course for people who are familiar to switch statements:

1. Replace `switch` with `match`
2. Remove `case`
3. Remove any `break`'s. If you don't want to `break` by default you can use `continue` for a fallthrough.
4. Change `default` to a single underscore.

Control flow:

The patterns are matched from top to bottom. If a pattern matches, the corresponding block will be executed. After that, the execution continues below the `match` statement. If you want to have a fallthrough you can use `continue` to stop execution in the current block and check the ones below it.

There are 6 pattern types:

- **constant pattern** constant primitives, like numbers and strings

```
match x:
    1:      print("We are number one!")
    2:      print("Two are better than one!")
    "test": print("Oh snap! It's a string!")
```

- **variable pattern** matches the contents of a variable/enum

```
match typeof(x):
    TYPE_FLOAT:  print("float")
    TYPE_STRING: print("text")
    TYPE_ARRAY:  print("array")
```

- **wildcard pattern** This pattern matches everything. It's written as a single underscore.

It can be used as the equivalent of the `default` in a `switch` statement in other languages.

```
match x:
    1: print("it's one!")
    2: print("it's one times two!")
    _: print("it's not 1 or 2. I don't care tbh.")
```

- **binding pattern** A binding pattern introduces a new variable. Like the wildcard pattern, it matches everything - and also gives that value a name. It's especially useful in array and dictionary patterns.

```
match x:
    1:           print("it's one!")
    2:           print("it's one times two!")
    var new_var: print("it's not 1 or 2, it's ", new_var)
```

- **array pattern** matches an array. Every single element of the array pattern is a pattern itself so you can nest them.

The length of the array is tested first, it has to be the same size as the pattern, otherwise the pattern don't match.

Open-ended array: An array can be bigger than the pattern by making the last subpattern ..

Every subpattern has to be comma separated.

```
match x:
    []:
        print("empty array")
    [1, 3, "test", null]:
        print("very specific array")
    [var start, _, "test"]:
        print("first element is ", start, ", and the last is \"test\"")
    [42, ...]:
        print("open ended array")
```

- **dictionary pattern** Works in the same way as the array pattern. Every key has to be a constant pattern.

The size of the dictionary is tested first, it has to be the same size as the pattern, otherwise the pattern don't match.

Open-ended dictionary: A dictionary can be bigger than the pattern by making the last subpattern ..

Every subpattern has to be comma separated.

If you don't specify a value, then only the existence of the key is checked.

A value pattern is separated from the key pattern with a :

```
match x:
    {}:
        print("empty dict")
    {"name": "dennis"}:
        print("the name is dennis")
    {"name": "dennis", "age": var age}:
        print("dennis is ", age, " years old.")
    {"name", "age"}:
        print("has a name and an age, but it's not dennis :(")
    {"key": "godotisawesome", ...}:
        print("I only checked for one entry and ignored the rest")
```

Multipatterns: You can also specify multiple patterns separated by a comma. These patterns aren't allowed to have any bindings in them.

```
match x:
    1, 2, 3:
        print("it's 1 - 3")
    "sword", "splash potion", "fist":
        print("yep, you've taken damage")
```

Classes

By default, the body of a script file is an unnamed class and it can only be referenced externally as a resource or file. Class syntax is meant to be very compact and can only contain member variables or functions. Static functions are allowed, but not static members (this is in the spirit of thread safety, since scripts can be initialized in separate threads without the user knowing). In the same way, member variables (including arrays and dictionaries) are initialized every time an instance is created.

Below is an example of a class file.

```
# saved as a file named myclass.gd

var a = 5

func print_value_of_a():
    print(a)
```

Inheritance

A class (stored as a file) can inherit from

- A global class
- Another class file
- An inner class inside another class file.

Multiple inheritance is not allowed.

Inheritance uses the `extends` keyword:

```
# Inherit/extend a globally available class
extends SomeClass

# Inherit/extend a named class file
extends "somefile.gd"

# Inherit/extend an inner class in another file
extends "somefile.gd".SomeInnerClass
```

To check if a given instance inherits from a given class the `is` keyword can be used:

```
# Cache the enemy class
const enemy_class = preload("enemy.gd")

# [...]

# use 'is' to check inheritance
if (entity is enemy_class):
    entity.apply_damage()
```

Class Constructor

The class constructor, called on class instantiation, is named `_init`. As mentioned earlier, the constructors of parent classes are called automatically when inheriting a class. So there is usually no need to call `._init()` explicitly.

If a parent constructor takes arguments, they are passed like this:

```
func _init(args). (parent_args):
    pass
```

Inner classes

A class file can contain inner classes. Inner classes are defined using the `class` keyword. They are instanced using the `ClassName.new()` function.

```
# inside a class file

# An inner class in this class file
class SomeInnerClass:
    var a = 5
    func print_value_of_a():
        print(a)

# This is the constructor of the class file's main class
func _init():
    var c = SomeInnerClass.new()
    c.print_value_of_a()
```

Classes as resources

Classes stored as files are treated as *resources*. They must be loaded from disk to access them in other classes. This is done using either the `load` or `preload` functions (see below). Instancing of a loaded class resource is done by calling the `new` function on the class object:

```
# Load the class resource when calling load()
var MyClass = load("myclass.gd")

# Preload the class only once at compile time
var MyClass2 = preload("myclass.gd")

func _init():
    var a = MyClass.new()
    a.somefunction()
```

Exports

Class members can be exported. This means their value gets saved along with the resource (e.g. the *scene*) they're attached to. They will also be available for editing in the property editor. Exporting is done by using the `export` keyword:

```
extends Button

export var number = 5 # value will be saved and visible in the property editor
```

An exported variable must be initialized to a constant expression or have an export hint in the form of an argument to the `export` keyword (see below).

One of the fundamental benefits of exporting member variables is to have them visible and editable in the editor. This way artists and game designers can modify values that later influence how the program runs. For this, a special export

syntax is provided.

```
# If the exported value assigns a constant or constant expression,
# the type will be inferred and used in the editor

export var number = 5

# Export can take a basic data type as an argument which will be
# used in the editor

export(int) var number

# Export can also take a resource type to use as a hint

export(Texture) var character_face
export(PackedScene) var scene_file

# Integers and strings hint enumerated values

# Editor will enumerate as 0, 1 and 2
export(int, "Warrior", "Magician", "Thief") var character_class
# Editor will enumerate with string names
export(String, "Rebecca", "Mary", "Leah") var character_name

# Strings as paths

# String is a path to a file
export(String, FILE) var f
# String is a path to a directory
export(String, DIR) var f
# String is a path to a file, custom filter provided as hint
export(String, FILE, "*.txt") var f

# Using paths in the global filesystem is also possible,
# but only in tool scripts (see further below)

# String is a path to a PNG file in the global filesystem
export(String, FILE, GLOBAL, "*.png") var tool_image
# String is a path to a directory in the global filesystem
export(String, DIR, GLOBAL) var tool_dir

# The MULTILINE setting tells the editor to show a large input
# field for editing over multiple lines
export(String, MULTILINE) var text

# Limiting editor input ranges

# Allow integer values from 0 to 20
export(int, 20) var i
# Allow integer values from -10 to 20
export(int, -10, 20) var j
# Allow floats from -10 to 20, with a step of 0.2
export(float, -10, 20, 0.2) var k
# Allow values y = exp(x) where y varies between 100 and 1000
# while snapping to steps of 20. The editor will present a
# slider for easily editing the value.
export(float, EXP, 100, 1000, 20) var l

# Floats with easing hint
```

```
# Display a visual representation of the ease() function
# when editing
export(float, EASE) var transition_speed

# Colors

# Color given as Red-Green-Blue value
export(Color, RGB) var col # Color is RGB
# Color given as Red-Green-Blue-Alpha value
export(Color, RGBA) var col # Color is RGBA

# another node in the scene can be exported too

export(NodePath) var node
```

It must be noted that even if the script is not being run while at the editor, the exported properties are still editable (see below for “tool”).

Exporting bit flags

Integers used as bit flags can store multiple true/false (boolean) values in one property. By using the export hint int, FLAGS, they can be set from the editor:

```
# Individually edit the bits of an integer
export(int, FLAGS) var spell_elements = ELEMENT_WIND | ELEMENT_WATER
```

Restricting the flags to a certain number of named flags is also possible. The syntax is very similar to the enumeration syntax:

```
# Set any of the given flags from the editor
export(int, FLAGS, "Fire", "Water", "Earth", "Wind") var spell_elements = 0
```

In this example, Fire has value 1, Water has value 2, Earth has value 4 and Wind corresponds to value 8. Usually, constants should be defined accordingly (e.g. const ELEMENT_WIND = 8 and so on).

Using bit flags requires some understanding of bitwise operations. If in doubt, boolean variables should be exported instead.

Exporting arrays

Exporting arrays works but with an important caveat: While regular arrays are created local to every class instance, exported arrays are *shared* between all instances. This means that editing them in one instance will cause them to change in all other instances. Exported arrays can have initializers, but they must be constant expressions.

```
# Exported array, shared between all instances.
# Default value must be a constant expression.

export var a=[1,2,3]

# Typed arrays also work, only initialized empty:

export var vector3s = PoolVector3Array()
export var strings = PoolStringArray()
```

```
# Regular array, created local for every instance.  
# Default value can include run-time values, but can't  
# be exported.  
  
var b = [a, 2, 3]
```

Setters/getters

It is often useful to know when a class' member variable changes for whatever reason. It may also be desired to encapsulate its access in some way.

For this, GDScript provides a *setter/getter* syntax using the `setget` keyword. It is used directly after a variable definition:

```
var variable = value setget setterfunc, getterfunc
```

Whenever the value of `variable` is modified by an *external* source (i.e. not from local usage in the class), the *setter* function (`setterfunc` above) will be called. This happens *before* the value is changed. The *setter* must decide what to do with the new value. Vice-versa, when `variable` is accessed, the *getter* function (`getterfunc` above) must return the desired value. Below is an example:

```
var myvar setget myvar_set,myvar_get  
  
func myvar_set(newvalue):  
    myvar=newvalue  
  
func myvar_get():  
    return myvar # getter must return a value
```

Either of the *setter* or *getter* functions can be omitted:

```
# Only a setter  
var myvar = 5 setget myvar_set  
# Only a getter (note the comma)  
var myvar = 5 setget ,myvar_get
```

Get/Setters are especially useful when exporting variables to editor in tool scripts or plugins, for validating input.

As said *local* access will *not* trigger the setter and getter. Here is an illustration of this:

```
func _init():  
    # Does not trigger setter/getter  
    myinteger=5  
    print(myinteger)  
  
    # Does trigger setter/getter  
    self.myinteger=5  
    print(self.myinteger)
```

Tool mode

Scripts, by default, don't run inside the editor and only the exported properties can be changed. In some cases it is desired that they do run inside the editor (as long as they don't execute game code or manually avoid doing so). For this, the `tool` keyword exists and must be placed at the top of the file:

```
tool
extends Button

func _ready():
    print("Hello")
```

Memory management

If a class inherits from `Reference`, then instances will be freed when no longer in use. No garbage collector exists, just simple reference counting. By default, all classes that don't define inheritance extend `Reference`. If this is not desired, then a class must inherit `Object` manually and must call `instance.free()`. To avoid reference cycles that can't be freed, a `weakref` function is provided for creating weak references.

Signals

It is often desired to send a notification that something happened in an instance. GDScript supports creation of built-in Godot signals. Declaring a signal in GDScript is easy using the `signal` keyword.

```
# No arguments
signal your_signal_name
# With arguments
signal your_signal_name_with_args(a,b)
```

These signals, just like regular signals, can be connected in the editor or from code. Just take the instance of a class where the signal was declared and connect it to the method of another instance:

```
func _callback_no_args():
    print("Got callback!")

func _callback_args(a,b):
    print("Got callback with args! a: ",a," and b: ",b)

func _at_some_func():
    instance.connect("your_signal_name", self, "_callback_no_args")
    instance.connect("your_signal_name_with_args", self, "_callback_args")
```

It is also possible to bind arguments to a signal that lacks them with your custom values:

```
func _at_some_func():
    instance.connect("your_signal_name", self, "_callback_args", [22, "hello"])
```

This is very useful when a signal from many objects is connected to a single callback and the sender must be identified:

```
func _button_pressed(which):
    print("Button was pressed: ", which.get_name())

func _ready():
    for b in get_node("buttons").get_children():
        b.connect("pressed", self, "_button_pressed", [b])
```

Finally, emitting a custom signal is done by using the `Object.emit_signal` method:

```
func _at_some_func():
    emit_signal("your_signal_name")
```

```
emit_signal("your_signal_name_with_args", 55, 128)
someinstance.emit_signal("somesignal")
```

Coroutines

GDScript offers support for [coroutines](#) via the `yield` built-in function. Calling `yield()` will immediately return from the current function, with the current frozen state of the same function as the return value. Calling `resume` on this resulting object will continue execution and return whatever the function returns. Once resumed the state object becomes invalid. Here is an example:

```
func myfunc():
    print("hello")
    yield()
    print("world")

func _ready():
    var y = myfunc()
    # Function state saved in 'y'
    print("my dear")
    y.resume()
    # 'y' resumed and is now an invalid state
```

Will print:

```
hello
my dear
world
```

It is also possible to pass values between `yield()` and `resume()`, for example:

```
func myfunc():
    print("hello")
    print( yield() )
    return "cheers!"

func _ready():
    var y = myfunc()
    # Function state saved in 'y'
    print( y.resume("world") )
    # 'y' resumed and is now an invalid state
```

Will print:

```
hello
world
cheers!
```

Coroutines & signals

The real strength of using `yield` is when combined with signals. `yield` can accept two parameters, an object and a signal. When the signal is received, execution will recommence. Here are some examples:

```
# Resume execution the next frame
yield(get_tree(), "idle_frame")

# Resume execution when animation is done playing:
yield(get_node("AnimationPlayer"), "finished")

# Wait 5 seconds, then resume execution
yield(get_tree().create_timer(5.0), "timeout")
```

Onready keyword

When using nodes, it's very common to desire to keep references to parts of the scene in a variable. As scenes are only warranted to be configured when entering the active scene tree, the sub-nodes can only be obtained when a call to `Node._ready()` is made.

```
var mylabel

func _ready():
    mylabel = get_node("MyLabel")
```

This can get a little cumbersome, especially when nodes and external references pile up. For this, GDScript has the `onready` keyword, that defers initialization of a member variable until `_ready` is called. It can replace the above code with a single line:

```
onready var mylabel = get_node("MyLabel")
```

4.1.2 GDScript: An introduction to dynamic languages

About

This tutorial aims to be a quick reference for how to use GDScript more efficiently. It focuses on common cases specific to the language, but also covers a lot of information on dynamically typed languages.

It's meant to be especially useful for programmers with little or no previous experience with dynamically typed languages.

Dynamic nature

Pros & cons of dynamic typing

GDScript is a Dynamically Typed language. As such, its main advantages are that:

- The language is very simple to learn.
- Most code can be written and changed quickly and without hassle.
- Less code written means less errors & mistakes to fix.
- Easier to read the code (less clutter).

- No compilation is required to test.
- Runtime is tiny.
- Duck-typing and polymorphism by nature.

While the main disadvantages are:

- Less performance than statically typed languages.
- More difficult to refactor (symbols can't be traced)
- Some errors that would typically be detected at compile time in statically typed languages only appear while running the code (because expression parsing is more strict).
- Less flexibility for code-completion (some variable types are only known at run-time).

This, translated to reality, means that Godot+GDScript are a combination designed to create games very quickly and efficiently. For games that are very computationally intensive and can't benefit from the engine built-in tools (such as the Vector types, Physics Engine, Math library, etc), the possibility of using C++ is present too. This allows to still create the entire game in GDScript and add small bits of C++ in the areas that need a performance boost.

Variables & assignment

All variables in a dynamically typed language are “variant”-like. This means that their type is not fixed, and is only modified through assignment. Example:

Static:

```
int a; // value uninitialized
a = 5; // this is valid
a = "Hi!"; // this is invalid
```

Dynamic:

```
var a # null by default
a = 5 # valid, 'a' becomes an integer
a = "Hi!" # valid, 'a' changed to a string
```

As function arguments:

Functions are of dynamic nature too, which means they can be called with different arguments, for example:

Static:

```
void print_value(int value)
{
    printf("value is %i\n", value);
}

[...]

print_value(55); // valid
print_value("Hello"); // invalid
```

Dynamic:

```
func print_value(value):
    print(value)
[...]

print_value(55) # valid
print_value("Hello") # valid
```

Pointers & referencing:

In static languages such as C or C++ (and to some extent Java and C#), there is a distinction between a variable and a pointer/reference to a variable. The later allows the object to be modified by other functions by passing a reference to the original one.

In C# or Java, everything not a built-in type (int, float, sometimes String) is always a pointer or a reference. References are also garbage-collected automatically, which means they are erased when no longer used. Dynamically typed languages tend to use this memory model too. Some Examples:

- C++:

```
void use_class(SomeClass *instance) {

    instance->use();
}

void do_something() {

    SomeClass *instance = new SomeClass; // created as pointer
    use_class(instance); // passed as pointer
    delete instance; // otherwise it will leak memory
}
```

- Java:

```
@Override
public final void use_class(SomeClass instance) {

    instance.use();
}

public final void do_something() {

    SomeClass instance = new SomeClass(); // created as reference
    use_class(instance); // passed as reference
    // garbage collector will get rid of it when not in
    // use and freeze your game randomly for a second
}
```

- GDScript:

```
func use_class(instance); # does not care about class type
    instance.use() # will work with any class that has a ".use()" method.

func do_something():
    var instance = SomeClass.new() # created as reference
    use_class(instance) # passed as reference
    # will be unreferenced and deleted
```

In GDScript, only base types (int, float, string and the vector types) are passed by value to functions (value is copied). Everything else (instances, arrays, dictionaries, etc) is passed as reference. Classes that inherit *Reference* (the default if nothing is specified) will be freed when not used, but manual memory management is allowed too if inheriting manually from *Object*.

Arrays

Arrays in dynamically typed languages can contain many different mixed datatypes inside and are always dynamic (can be resized at any time). Compare for example arrays in statically typed languages:

```
int *array = new int[4]; // create array
array[0] = 10; // initialize manually
array[1] = 20; // can't mix types
array[2] = 40;
array[3] = 60;
// can't resize
use_array(array); // passed as pointer
delete[] array; // must be freed

// or

std::vector<int> array;
array.resize(4);
array[0] = 10; // initialize manually
array[1] = 20; // can't mix types
array[2] = 40;
array[3] = 60;
array.resize(3); // can be resized
use_array(array); // passed reference or value
// freed when stack ends
```

And in GDScript:

```
var array = [10, "hello", 40, 60] # simple, and can mix types
array.resize(3) # can be resized
use_array(array) # passed as reference
# freed when no longer in use
```

In dynamically typed languages, arrays can also double as other datatypes, such as lists:

```
var array = []
array.append(4)
array.append(5)
array.pop_front()
```

Or unordered sets:

```
var a = 20
if a in [10, 20, 30]:
    print("We have a winner!")
```

Dictionaries

Dictionaries are a very powerful tool in dynamically typed languages. Most programmers that come from statically typed languages (such as C++ or C#) ignore their existence and make their life unnecessarily more difficult. This datatype is generally not present in such languages (or only on limited form).

Dictionaries can map any value to any other value with complete disregard for the datatype used as either key or value. Contrary to popular belief, they are very efficient because they can be implemented with hash tables. They are, in fact, so efficient that some languages will go as far as implementing arrays as dictionaries.

Example of Dictionary:

```
var d = { "name": "john", "age": 22 } # simple syntax
print("Name: ", d["name"], " Age: ", d["age"])
```

Dictionaries are also dynamic, keys can be added or removed at any point at little cost:

```
d["mother"] = "Rebecca" # addition
d["age"] = 11 # modification
d.erase("name") # removal
```

In most cases, two-dimensional arrays can often be implemented more easily with dictionaries. Here's a simple battleship game example:

```
# battleship game

const SHIP = 0
const SHIP_HIT = 1
const WATER_HIT = 2

var board = {}

func initialize():
    board[Vector(1,1)] = SHIP
    board[Vector(1,2)] = SHIP
    board[Vector(1,3)] = SHIP

func missile(pos):
    if pos in board: # something at that pos
        if board[pos] == SHIP: # there was a ship! hit it
            board[pos] = SHIP_HIT
        else:
            print("already hit here!") # hey dude you already hit here
        else: # nothing, mark as water
            board[pos] = WATER_HIT

func game():
    initialize()
    missile(Vector2(1,1))
    missile(Vector2(5,8))
    missile(Vector2(2,3))
```

Dictionaries can also be used as data markup or quick structures. While GDScript dictionaries resemble python dictionaries, it also supports Lua style syntax an indexing, which makes it very useful for writing initial states and quick structs:

```
# same example, lua-style support
# this syntax is a lot more readable and usable

var d =
    name = "john",
    age = 22
}
```

```

print("Name: ", d.name, " Age: ", d.age) # used "." based indexing

# indexing

d.mother = "rebecca"
d["mother"] = "rebecca"
d.name = "caroline" # if key exists, assignment does work, this is why it's like a
→quick struct.

```

For & while

Iterating in some statically typed languages can be quite complex:

```

const char* strings = new const char*[50];

[...]

for(int i=0; i<50; i++)
{
    printf("value: %s\n", i, strings[i]);
}

// even in STL:

for(std::list<std::string>::const_iterator it = strings.begin(); it != strings.end(); →it++)
{
    std::cout << *it << std::endl;
}

```

This is usually greatly simplified in dynamically typed languages:

```

for s in strings:
    print(s)

```

Container datatypes (arrays and dictionaries) are iterable. Dictionaries allow iterating the keys:

```

for key in dict:
    print(key, " -> ", dict[key])

```

Iterating with indices is also possible:

```

for i in range(strings.size()):
    print(strings[i])

```

The range() function can take 3 arguments:

```

range(n) (will go from 0 to n-1)
range(b, n) (will go from b to n-1)
range(b, n, s) (will go from b to n-1, in steps of s)

```

Some examples:

```
for(int i=0; i<10; i++) {}

for(int i=5; i<10; i++) {}

for(int i=5; i<10; i+=2) {}
```

Translate to:

```
for i in range(10):

for i in range(5, 10):

for i in range(5, 10, 2):
```

And backwards looping is done through a negative counter:

```
for(int i=10; i>0; i--) {}
```

becomes

```
for i in range(10, 0, -1):
```

While

while() loops are the same everywhere:

```
var i = 0

while(i < strings.size()):
    print(strings[i])
    i += 1
```

Duck typing

One of the most difficult concepts to grasp when moving from a statically typed language to a dynamic one is duck typing. Duck typing makes overall code design much simpler and straightforward to write, but it's not obvious how it works.

As an example, imagine a situation where a big rock is falling down a tunnel, smashing everything on its way. The code for the rock, in a statically typed language would be something like:

```
void BigRollingRock::on_object_hit(Smashable *entity)
{
    entity->smash();
}
```

This way, everything that can be smashed by a rock would have to inherit Smashable. If a character, enemy, piece of furniture, small rock were all smashable, they would need to inherit from the class Smashable, possibly requiring multiple inheritance. If multiple inheritance was undesired, then they would have to inherit a common class like Entity. Yet, it would not be very elegant to add a virtual method `smash()` to Entity only if a few of them can be smashed.

With dynamically typed languages, this is not a problem. Duck typing makes sure you only have to define a `smash()` function where required and that's it. No need to consider inheritance, base classes, etc.

```
func _on_object_hit(object):
    object.smash()
```

And that's it. If the object that hit the big rock has a smash() method, it will be called. No need for inheritance or polymorphism. Dynamically typed languages only care about the instance having the desired method or member, not what it inherits or the class type. The definition of Duck Typing should make this clearer:

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck"

In this case, it translates to:

"If the object can be smashed, don't care what it is, just smash it."

Yes, we should call it Hulk typing instead.

It's possible that the object being hit doesn't have a smash() function. Some dynamically typed languages simply ignore a method call when it doesn't exist (like Objective C), but GDScript is more strict, so checking if the function exists is desirable:

```
func _on_object_hit(object):
    if (object.has_method("smash")):
        object.smash()
```

Then, simply define that method and anything the rock touches can be smashed.

4.1.3 GDScript Style Guide

Description

This styleguide lists conventions to write elegant GDScript. The goal is to encourage writing clean, readable code and promote consistency across projects, discussions, and tutorials. Hopefully, this will also encourage development of auto-formatting tools.

Since GDScript is close to Python, this guide is inspired by Python's [PEP 8](#) programming styleguide.

Note: Godot's built-in script editor uses a lot of these conventions by default. Let it help you.

Code Structure

Indentation

Indent type: Tabs (*editor default*)

Indent size: 4 (*editor default*)

Each indent level should be one greater than the block containing it.

Good

```
for i in range(10):
    print("hello")
```

Bad

```
for i in range(10):
    print("hello")

for i in range(10):
    print("hello")
```

Use 2 indent levels to distinguish continuation lines from regular code blocks.

Good

```
effect.interpolate_property(sprite, 'transform/scale',
                           sprite.get_scale(), Vector2(2.0, 2.0), 0.3,
                           Tween.TRANS_QUAD, Tween.EASE_OUT)
```

Bad

```
effect.interpolate_property(sprite, 'transform/scale',
                           sprite.get_scale(), Vector2(2.0, 2.0), 0.3,
                           Tween.TRANS_QUAD, Tween.EASE_OUT)
```

Blank lines

Surround functions and class definitions by a blank line.

Use one blank line inside functions to separate logical sections.

One Statement per Line

Never combine multiple statements on a single line. No, C programmers, not with a single line conditional statement (except with the ternary operator)!

Good

```
if position.x > width:
    position.x = 0

if flag:
    print("flagged")
```

Bad

```
if position.x > width: position.x = 0

if flag: print("flagged")
```

Avoid Unnecessary Parentheses

Avoid parentheses in expressions and conditional statements. Unless necessary for order of operations, they only reduce readability.

Good

```
if is_colliding():
    queue_free()
```

Bad

```
if (is_colliding()):  
    queue_free()
```

Whitespace

Always use one space around operators and after commas. Avoid extra spaces in dictionary references and function calls, or to create “columns.”

Good

```
position.x = 5  
position.y = mpos.y + 10  
dict['key'] = 5  
myarray = [4, 5, 6]  
print('foo')
```

Bad

```
position.x=5  
position.y = mpos.y+10  
dict ['key'] = 5  
myarray = [4,5,6]  
print ('foo')
```

Never!

```
x      = 100  
y      = 100  
velocity = 500
```

Naming Conventions

These naming conventions follow the Godot Engine style. Breaking these will make your code clash with the built-in naming conventions, which is ugly.

Classes and Nodes

Use PascalCase: extends KinematicBody

Also when loading a class into a constant or variable:

```
const MyCoolNode = preload('res://my_cool_node.gd')
```

Functions and Variables

Use snake_case: get_node()

Prepend a single underscore (_) to virtual methods (functions the user must override), private functions, and private variables: func _ready()

Signals

Use past tense:

```
signal door_opened
signal score_changed
```

Constants

Use CONSTANT_CASE, all caps, with an underscore (_) to separate words: const MAX_SPEED = 200

4.1.4 GDScript format strings

GDScript offers a feature called *format strings* which allows reusing text templates to succinctly create different but similar strings.

Format strings are just like normal strings, except they contain certain placeholder character-sequences. These placeholders can then easily be replaced by parameters handed to the format string.

As an example, with %s as a placeholder, the format string "Hello %s, how are you?" can easily be changed to "Hello World, how are you?". Notice the placeholder is in the middle of the string; modifying it without format strings could be cumbersome.

Usage in GDScript

Examine this concrete GDScript example:

```
# Define a format string with placeholder '%s'
var format_string = "We're waiting for %s."

# Using the '%' operator, the placeholder is replaced with the desired value
var actual_string = format_string % "Godot"

print(actual_string)
# output: "We're waiting for Godot."
```

Placeholders always start with a %, but the next character or characters, the *format specifier*, determines how the given value is converted to a string.

The %s seen in the example above is the simplest placeholder and works for most use cases: it converts the value by the same method by which an implicit String conversion or str() would convert it. Strings remain unchanged, Booleans turn into either "True" or "False", an integral or real number becomes a decimal, other types usually return their data in a human-readable string.

There are other *format specifiers*.

Multiple placeholders

Format strings may contain multiple placeholders. In such a case, the values are handed in the form of an array, one value per placeholder (unless using a format specifier with *, see *dynamic padding*):

```
var format_string = "%s was reluctant to learn %s, but now he enjoys it."
var actual_string = format_string % ["Estragon", "GDScript"]

print(actual_string)
# output: "Estragon was reluctant to learn GDScript, but now he enjoys it."
```

Note the values are inserted in order. Remember all placeholders must be replaced at once, so there must be an appropriate number of values.

Format specifiers

There are format specifiers other than `s` that can be used in placeholders. They consist of one or more characters. Some of them work by themselves like `s`, some appear before other characters, some only work with certain values or characters.

Placeholder types

One and only one of these must always appear as the last character in a format specifier. Apart from `s`, these require certain types of parameters.

<code>s</code>	Simple conversion to String by the same method as implicit String conversion.
<code>c</code>	A single Unicode character . Expects an unsigned 8-bit integer (0-255) for a code point or a single-character string.
<code>d</code>	A decimal integral number. Expects an integral or real number (will be floored).
<code>o</code>	An octal integral number. Expects an integral or real number (will be floored).
<code>x</code>	A hexadecimal integral number with lower-case letters. Expects an integral or real number (will be floored).
<code>X</code>	A hexadecimal integral number with upper-case letters. Expects an integral or real number (will be floored).
<code>f</code>	A decimal real number. Expects an integral or real number.

Placeholder modifiers

These characters appear before the above. Some of them work only under certain conditions.

<code>+</code>	In number specifiers, show + sign if positive.
<code>Inte-</code> <code>ger</code>	Set padding . Padded with spaces or with zeroes if integer starts with 0 in an integer placeholder. When used after <code>.</code> , see <code>..</code> .
<code>.</code>	Before <code>f</code> , set precision to 0 decimal places. Can be followed up with numbers to change. Padded with zeroes.
<code>-</code>	Pad to the right rather than the left.
<code>*</code>	Dynamic padding , expect additional integral parameter to set padding or precision after <code>..</code> , see <i>dynamic padding</i> .

Padding

The `.` (dot), `*` (asterisk), `-` (minus sign) and digit (0-9) characters are used for padding. This allows printing several values aligned vertically as if in a column, provided a fixed-width font is used.

To pad a string to a minimum length, add an integer to the specifier:

```
print("%10d" % 12345)
# output: "      12345"
# 5 leading spaces for a total length of 10
```

If the integer starts with 0, integral values are padded with zeroes instead of white space:

```
print("%010d" % 12345)
# output: "0000012345"
```

Precision can be specified for real numbers by adding a . (*dot*) with an integer following it. With no integer after ., a precision of 0 is used, rounding to integral value. The integer to use for padding must appear before the dot.

```
# pad to minimum length of 10, round to 3 decimal places
print("%10.3f" % 10000.5555)
# output: " 10000.556"
# 1 leading space
```

The – character will cause padding to the right rather than the left, useful for right text alignment:

```
print("%-10d" % 12345678)
# output: "12345678"
# 2 trailing spaces
```

Dynamic padding

By using the * (*asterisk*) character, the padding or precision can be set without modifying the format string. It is used in place of an integer in the format specifier. The values for padding and precision are then passed when formatting:

```
var format_string = "%.*.*f"
# pad to length of 7, round to 3 decimal places:
print(format_string % [7, 3, 8.8888])
# output: " 8.889"
# 2 leading spaces
```

It is still possible to pad with zeroes in integer placeholders by adding 0 before *:

```
print("%0*d" % [2, 3])
#output: "03"
```

Escape sequence

To insert a literal % character into a format string, it must be escaped to avoid reading it as a placeholder. This is done by doubling the character:

```
var health = 56
print("Remaining health: %d%%" % health)
# output: "Remaining health: 56%"
```

4.2 VisualScript

4.2.1 What is Visual Scripting

Visual Scripting is a tool designed to make the entry barrier to programming much lower. As code is more visual, it needs less abstract thinking to be understood. Any artist, animator, game designer, etc. can look at it and quickly grasp the flow of logic.

The reason it does not make existing programming obsolete is, simply, that it does not scale as well. It takes considerably more time to create code with it, and it's often more difficult to modify than just writing a few characters.

With the misunderstanding cleared up, the question that remains is what are the practical uses for Visual Scripting.

The most common use cases are as follows:

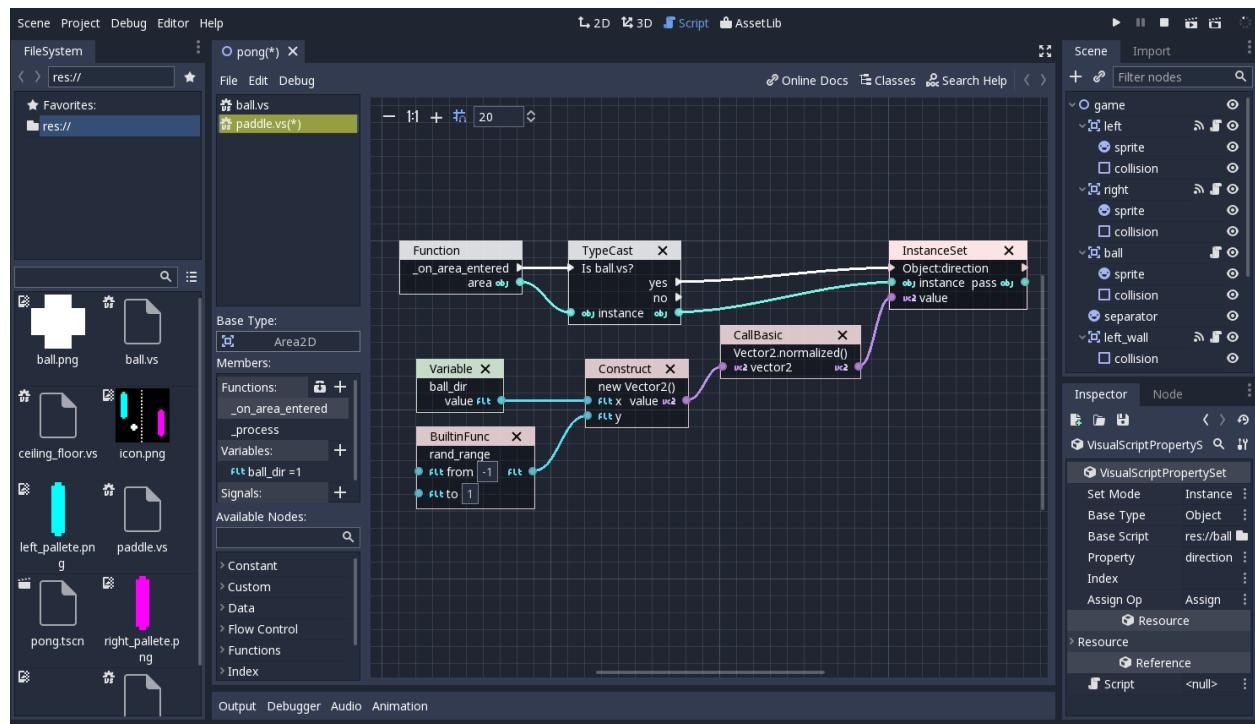
- Game development beginners who want to learn an engine but have no programming experience yet.
- Artists and Game Designers who have no experience in programming and want to create quick prototypes or simple games.
- Programmers working in a team that want to make part of the game logic available to Artists or Game Designers in order to offload some of their work.

These scenarios are far more common than one might think, so this is why Godot has added this feature.

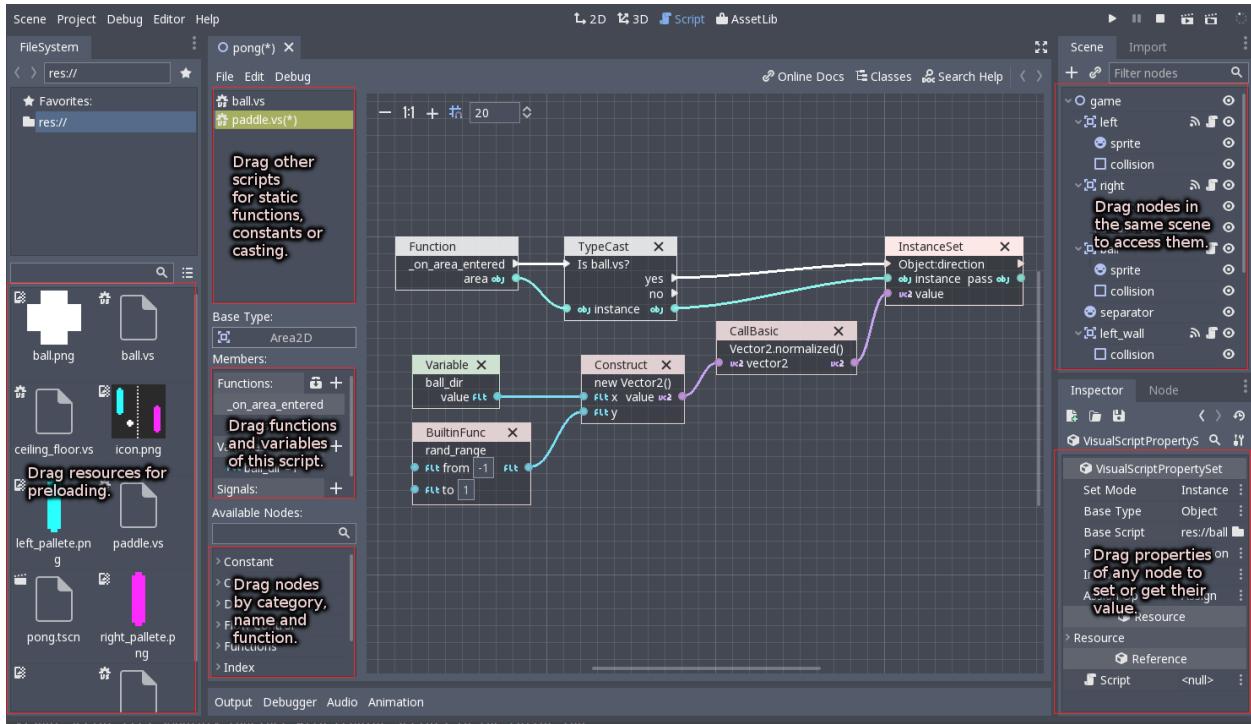
4.2.2 Getting started with Visual Scripting

As with everything in Godot, we prioritize a good experience over copying or integrating third party solutions which might not fit nicely in the current workflow. This led us to write our own version of how we believe this feature would work best with the engine.

In Godot, a Visual Script fits smoothly together with regular scripts in the Editor tab

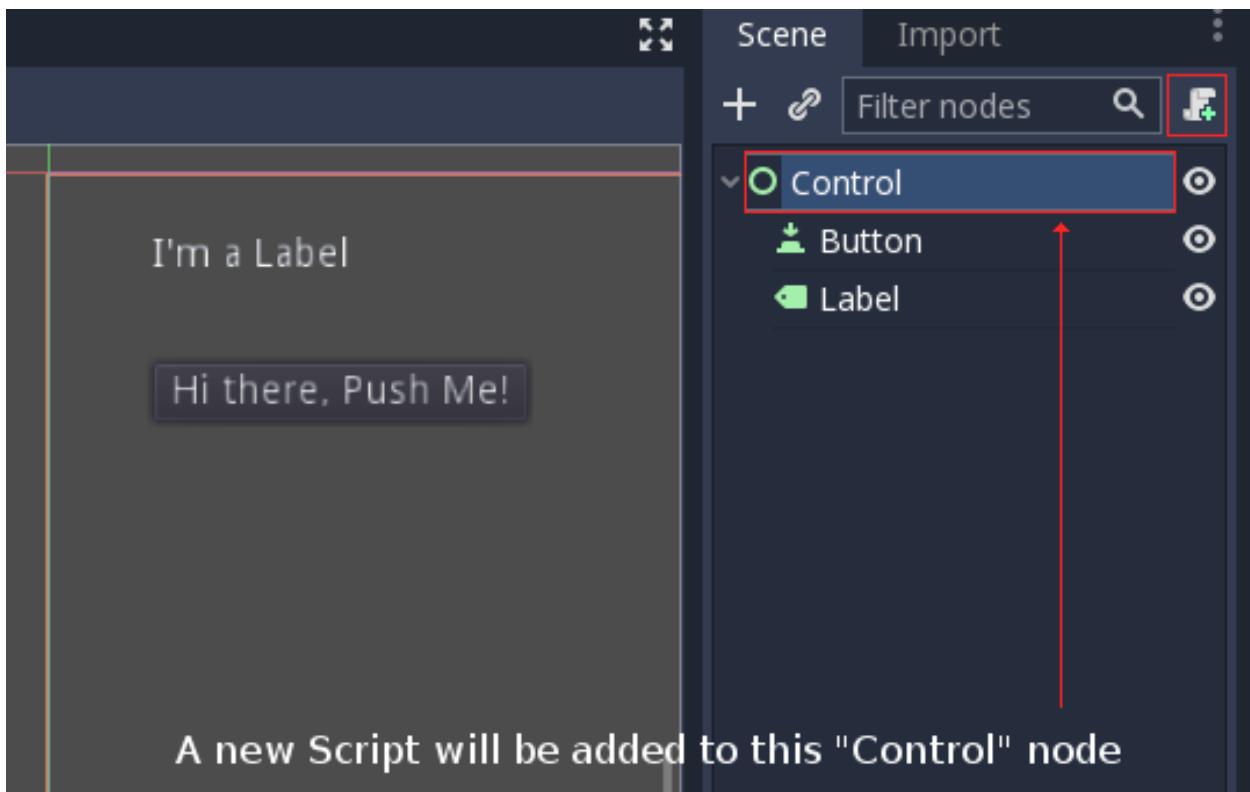


In fact, Visual Scripting integrates so well to Godot that it's hard to believe it was added only in version 3.0. This is because, when editing, the rest of Godot panels and docks act like a palette from where you can drag and drop all sorts of information to the script canvas:

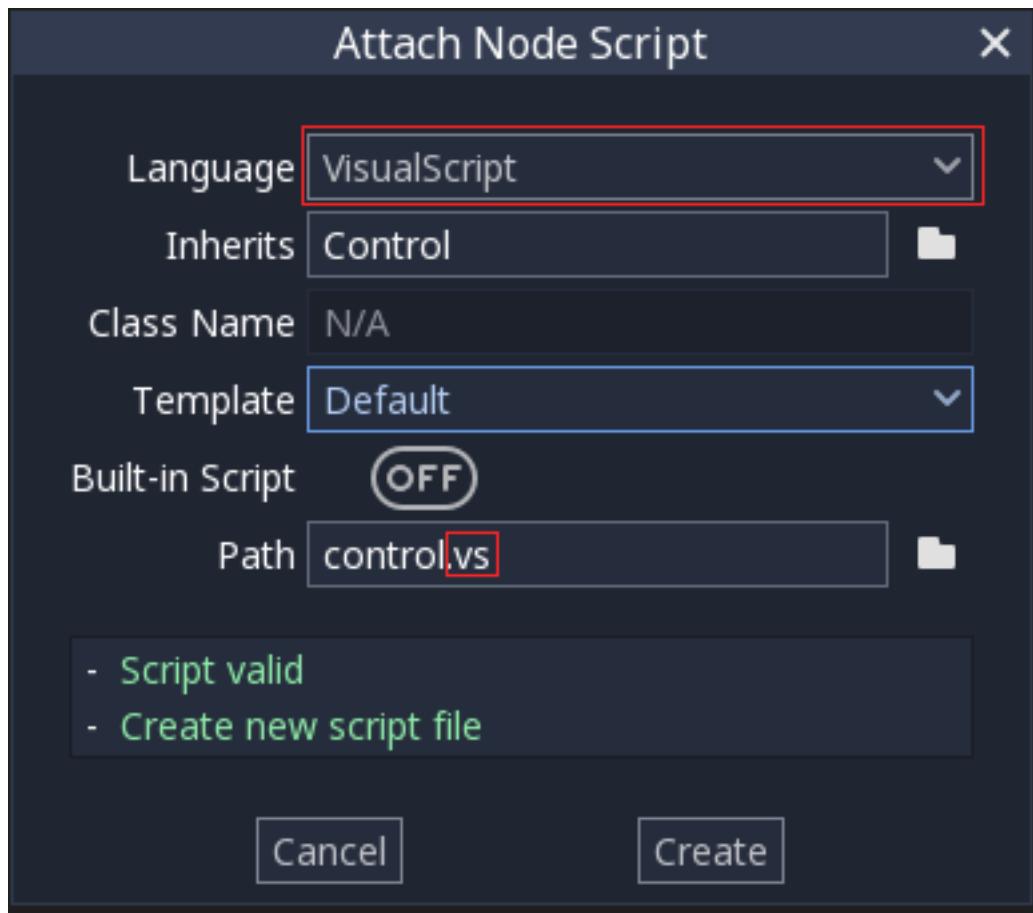


Creating a Script

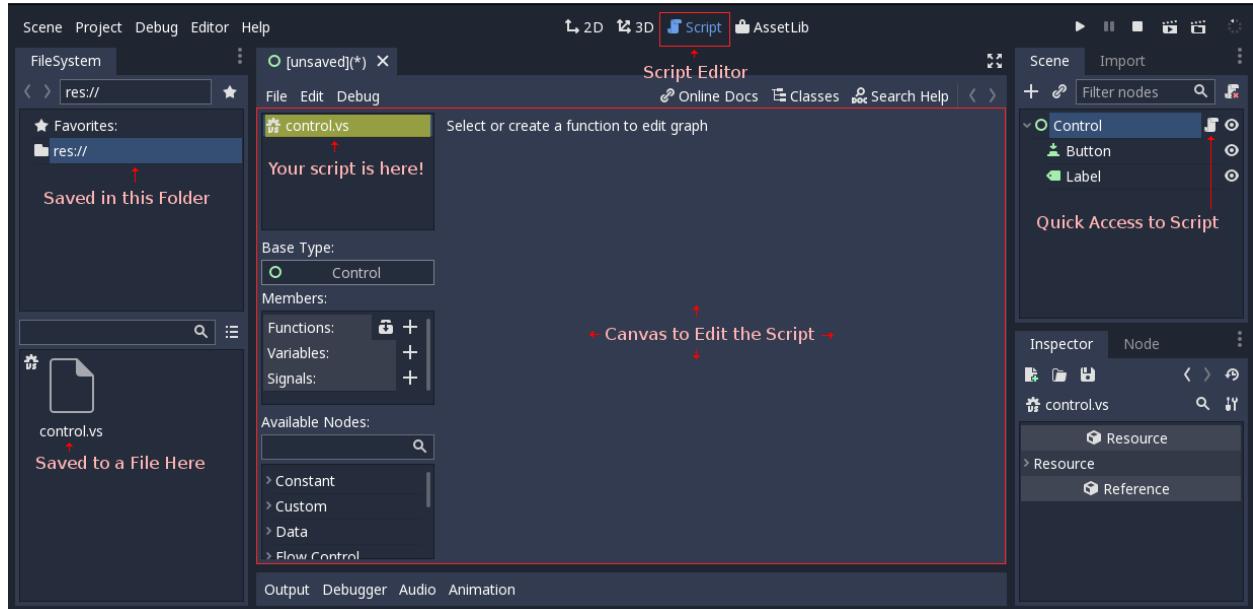
Creating scripts works the same as with other scripting languages: Just select any node in the scene and push the “New Script” button at the top right corner of the Scene Tree dock:



Once it opens, the script type “Visual Script” must be selected from the drop down list. The script extension must be “.vs” (for Visual Script!).



Finally, the Script Editor will open, allowing to start the editing of the visual script:



Adding a Function

Unlike other visual scripting implementations, Visual Scripting in Godot is heavily based on functions. This happens because it uses the same interface to communicate with the engine as other scripting engines. In Godot, the scripting interface is universal and all implementations conform to it.

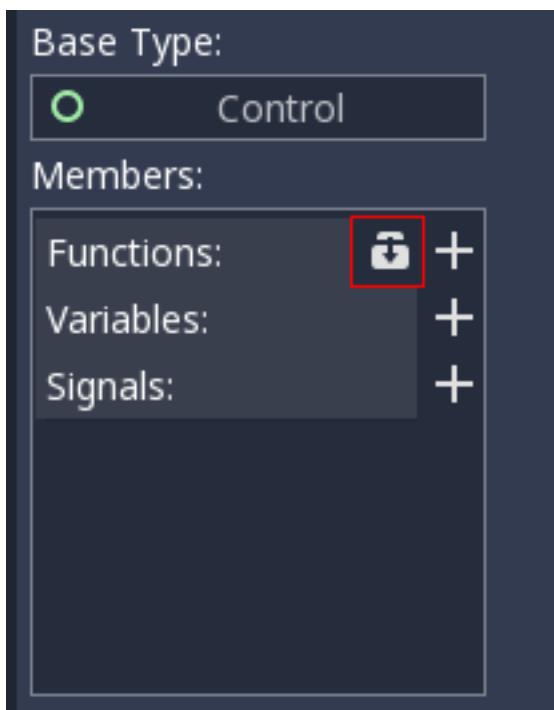
A function is an individual canvas with nodes connected.

A single script can contain many functions, each of which will have a canvas of its own, allowing for more organization.

There are three main ways to add functions in a script:

Overriding a Virtual Function

Most types of nodes and other types of objects in Godot contain virtual functions. These are functions that will be called (run your code) when something happens and can be looked up in the reference. Virtual functions are listed when pressing the “Override” icon in the member panel:

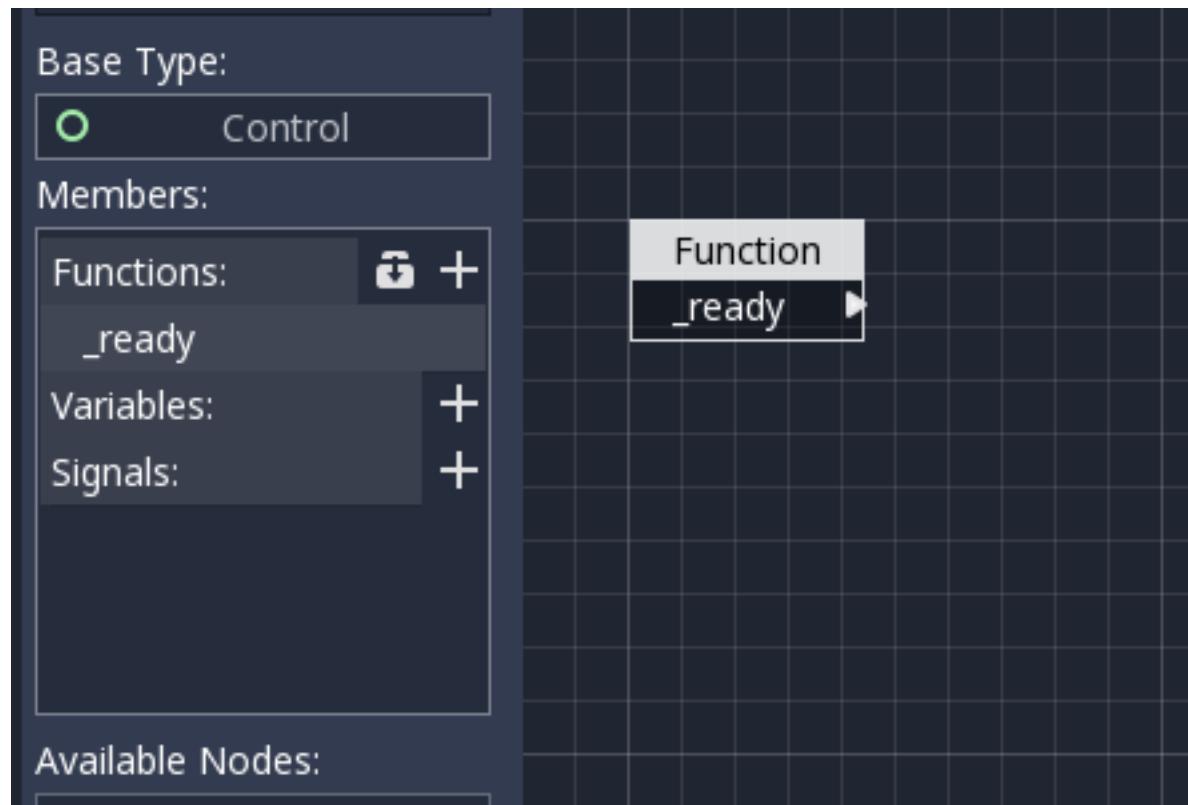


In the following example, a function will be executed when the node is loaded and added to the running scene. For this, the `_ready()` virtual method will be overridden:

Node

```
void _process ( float delta ) virtual  
void _fixed_process ( float delta ) virtual  
void _enter_tree ( ) virtual  
void _exit_tree ( ) virtual  
void _ready ( ) virtual  
void _input ( Object event ) virtual  
void _unhandled_input ( Object event ) virtual  
void _unhandled_key_input ( Object event ) virtual
```

Finally, a canvas appears for this function, showing the override:



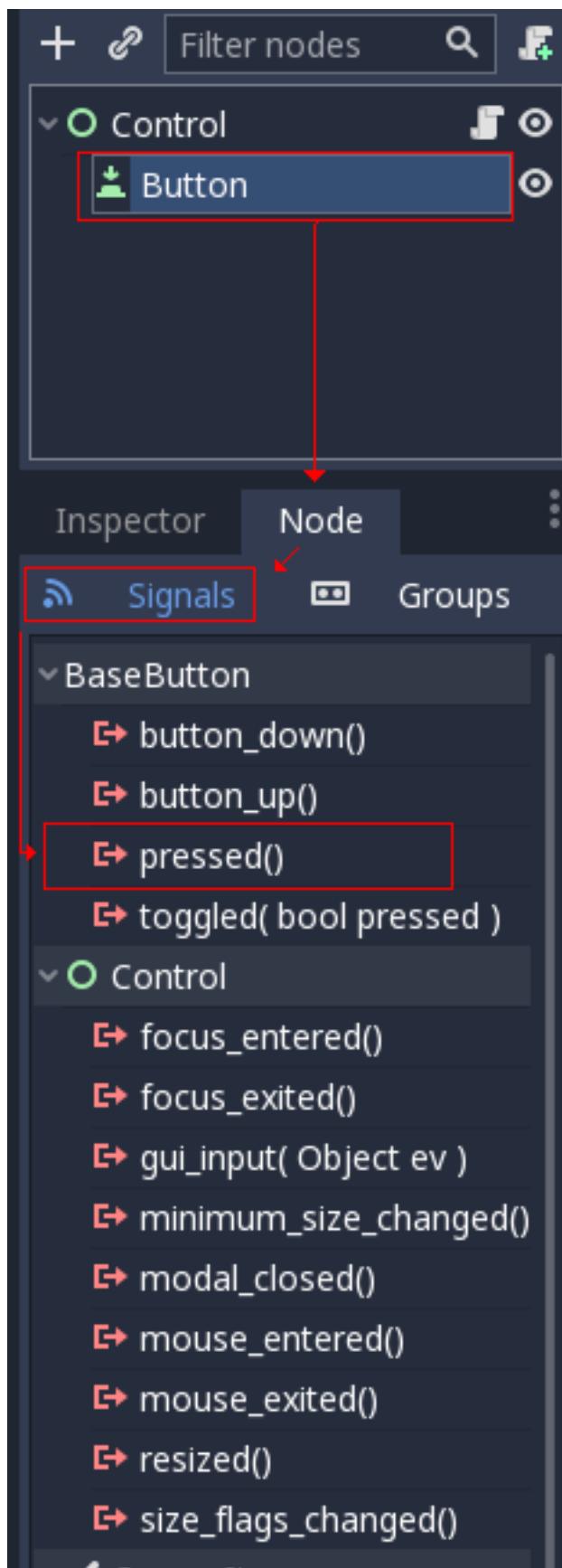
As some functions expect you to return a value, they will also add a return node where such value is supposed to be provided:



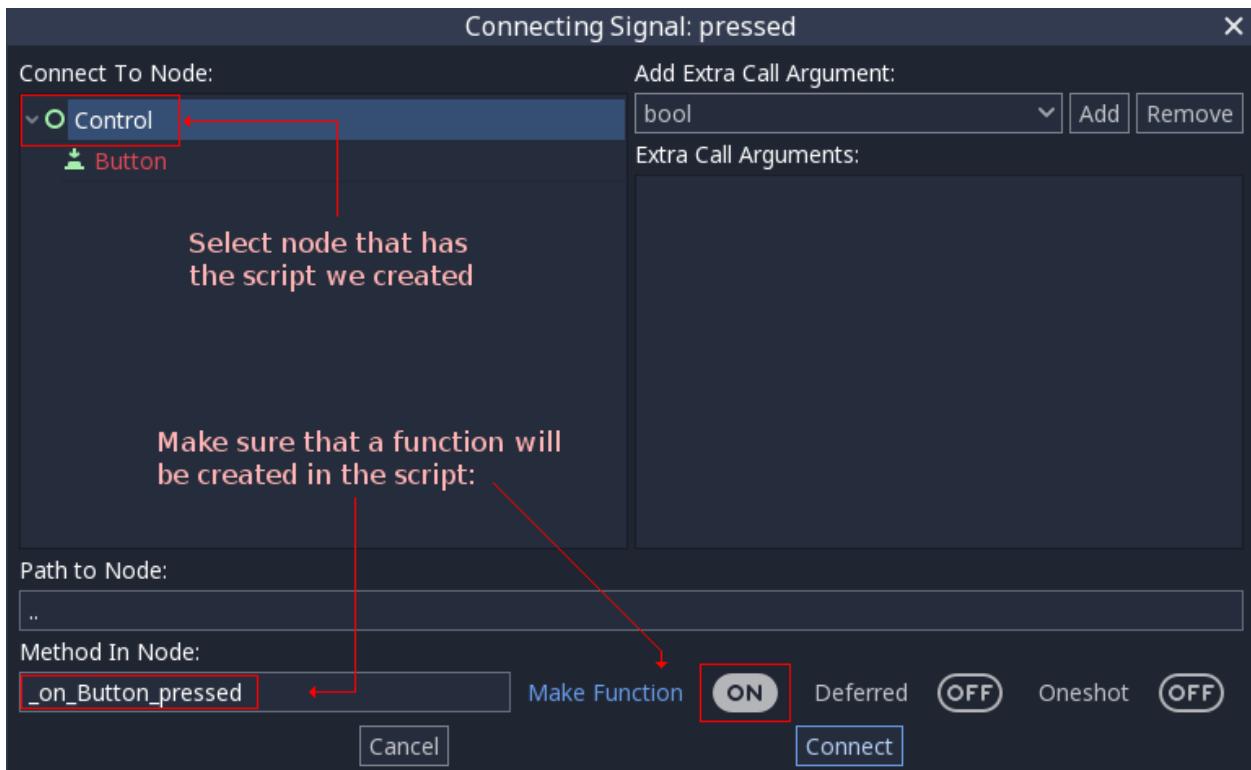
Connecting a Signal to a Function

Nodes in a tree emit signals when something happens. Godot uses signals for all sorts of things. A typical example would be a button that emits a “pressed” signal when actually pressed.

For this, a node must be selected and the Node tab opened. This will allow inspecting the signals. Once they are displayed, connect the “pressed” signal:



This will open the connection dialog. In this dialog, you must select the node where the signal will be connected to, and the function that will receive the signal:



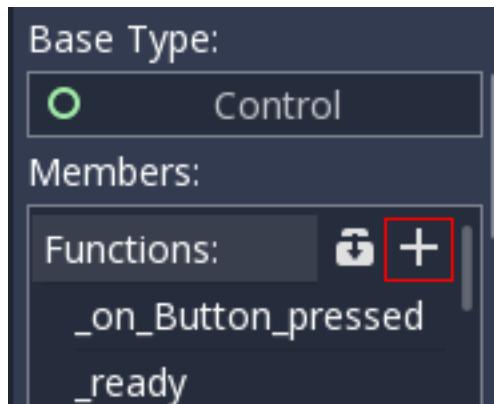
If this is done right, a new function will be created in our script and a signal will automatically be connected to it:



Creating a Function Manually

The last way to create functions is to do it manually. In general this is not as common unless you really need it. Custom functions work when another (or the same) script calls them manually. The main use case for this is to separate a function into more, or reusing your visual code.

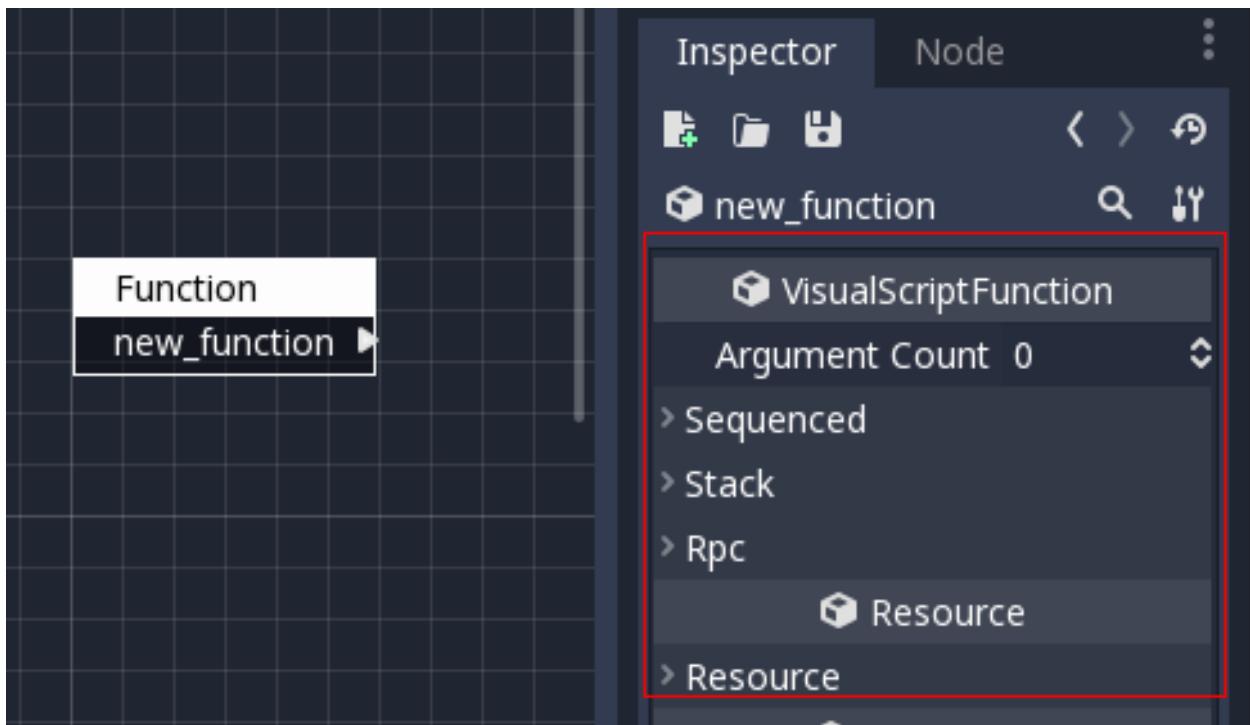
To create a function manually, push the big “Plus” button, and a new function will be added with a default name:



This will add a new function, which can be renamed by simply double clicking its name:



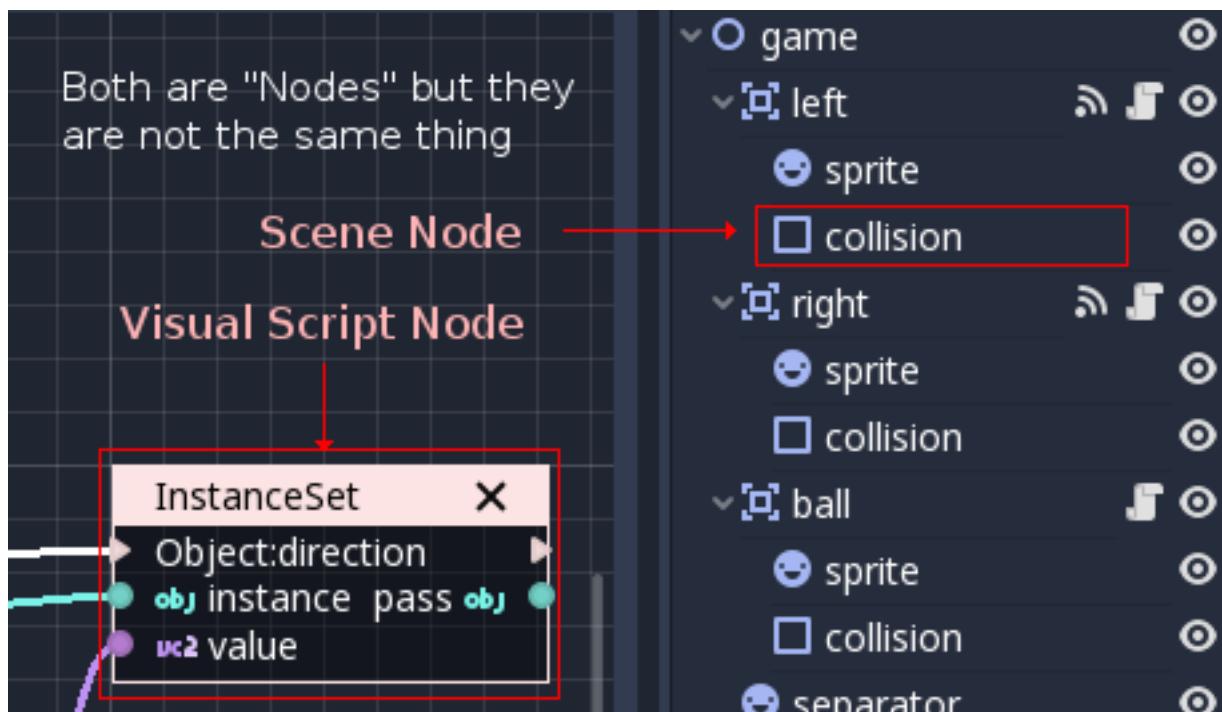
To edit the “arguments” this function can get (the values you pass to it when you call this function), simply click the Function node and check the inspector:



More on that will be explained later in this document.

4.2.3 Nodes and Terminology

Before continuing, it must be noted that the *Node* terminology needs to be used with care. When referring to *Visual Script Nodes* (or generally *Nodes*) this text will refer to the little boxes you connect with lines, which are part of a graph. When referring to *Scene Nodes*, it is implied that the elements that make up a Scene are being referred, which are part of a tree. Their naming is similar but their function is different. When referring to *Node* here, it will be implied that a *Visual Script Node* is referred to unless indicated otherwise.



Node Properties

Like in most visual scripting implementations, each node has editable properties. In Godot, though, we try to avoid bloating the nodes with editable controls for the sake of readability.

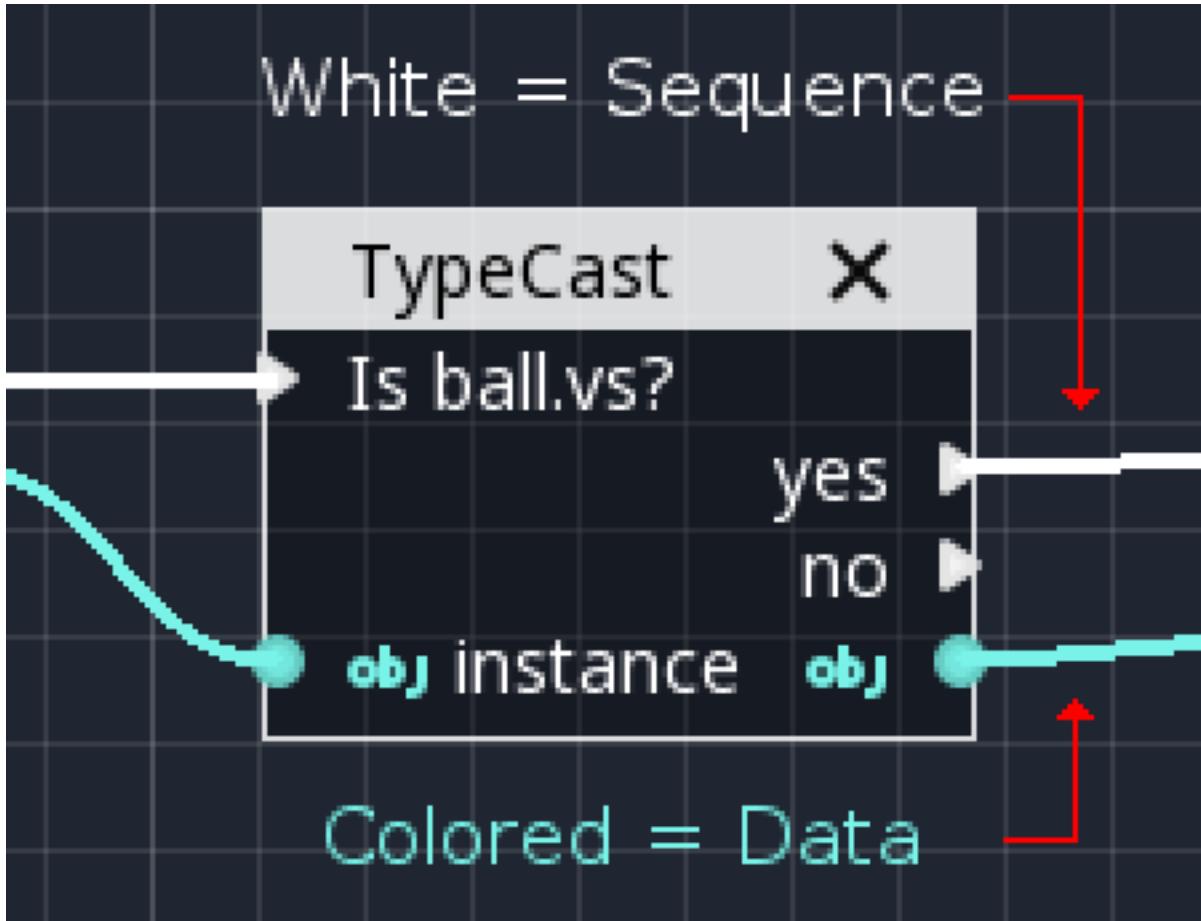
Nodes still display the required information as text, but editing is done via the *Inspector*. To edit them, just select any node and edit its properties in the *Inspector*.

Ports and Connections

Programming in Godot Visual Scripting is done via *Nodes* and *Port Connections* inside each function.

Ports

Nodes in Godot Visual Scripting have *Ports*. These are endpoints that appear to the left and right of nodes and which can be used to make *Connections*: There are two types of *Ports*: *Sequence* and *Data*.



Sequence Ports indicate the order in which operations are executed. Typically when a *Node* is done processing, it will go to the next node from one of the ports at the right. If nothing is connected the function may end, or another output *Sequence Port* might be tried (this depends on the node). Thanks to this, it's easy to understand the logic within a function by just following the white lines. Not every *Node* has *Sequence Ports*. In fact, most do not.

Data Ports ports contain typed values. Types can be any regular Godot types, such as a boolean, an integer, a string, a Vector3, an array, any Object or Scene Node, etc. A *Data Port* on the right side of a node is considered an output, while, a port on the left side is an input. Connecting them allows information to flow to the next node.

Not all *Data Port* types are compatible and will allow connections, though. Pay special attention to colors and icons, as each type has a different representation:



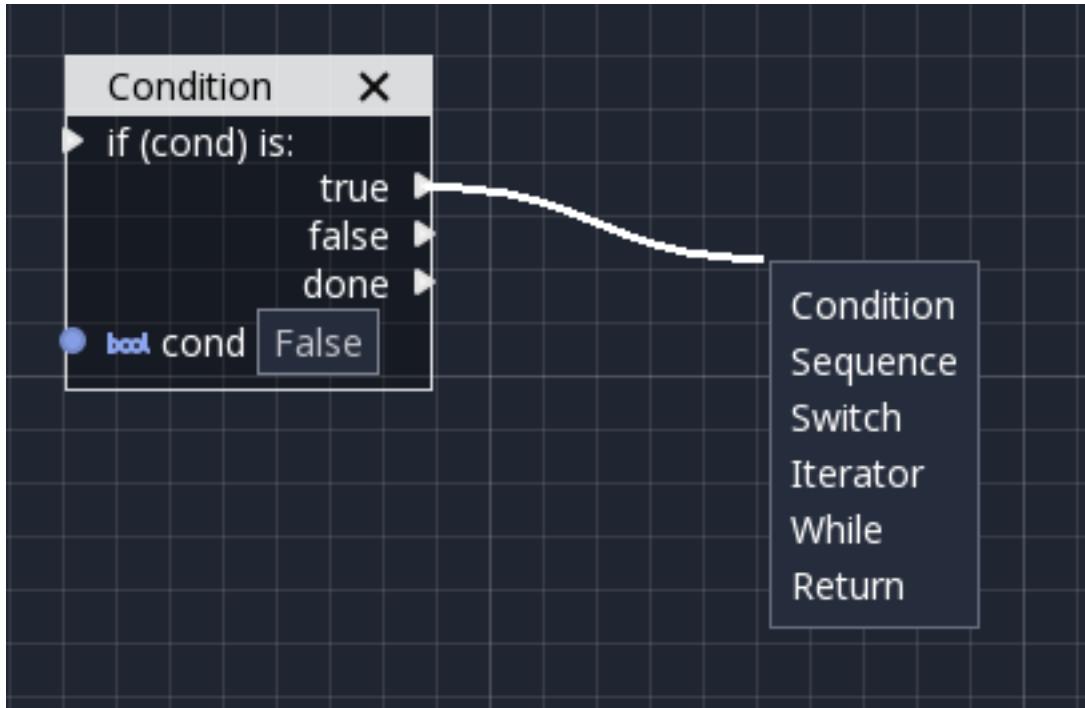
Connections

Connecting is a relatively simple process. Just drag an *Output Port* towards an *Input Port*.

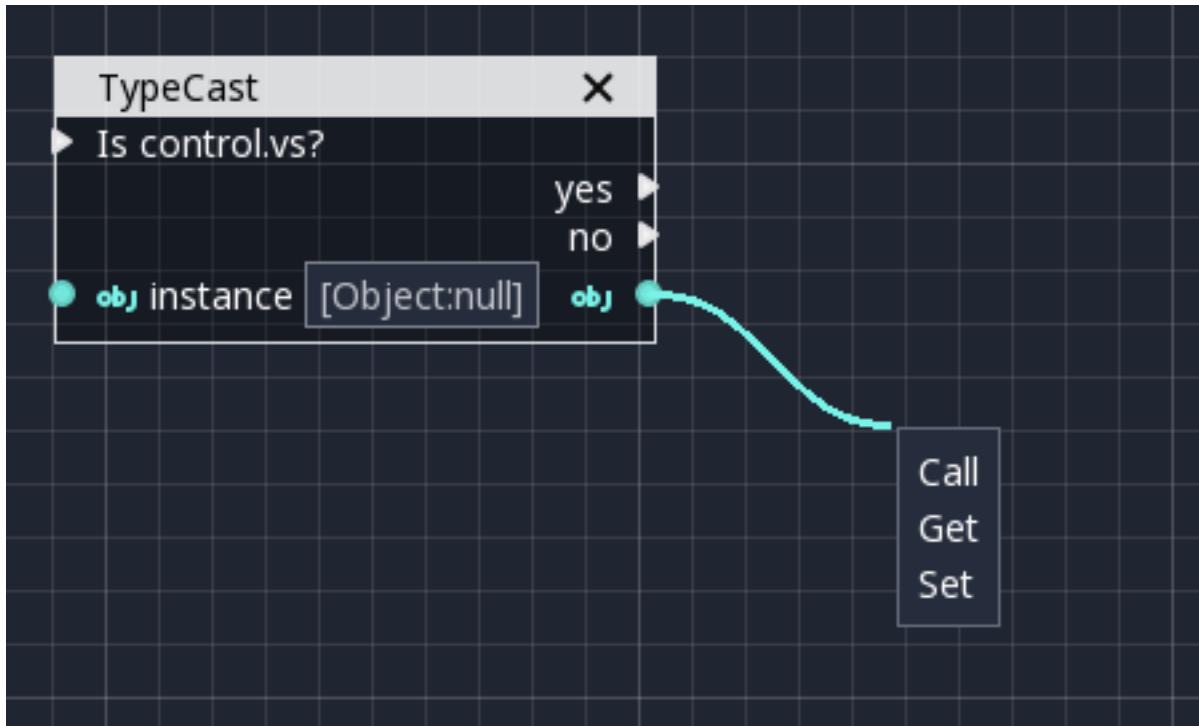
Disconnecting takes a bit more practice. Disconnecting in *Data Ports* happens by dragging the *Input* away, while for *Sequence Ports*, this happens by dragging the *Output* away.

This may seem strange at the beginning, but it happens because *Data Ports* are 1:N (A single output port can connect to many inputs), while *Sequence Ports* are N:1 (Many sequence outputs can be connected to a single input).

Connecting to empty space (drag to connect but unpress over empty space) is also context sensitive, it will supply a list of most common operations. For sequences, it will be conditional nodes:



While, for data, a contextual set/get/call menu will open:

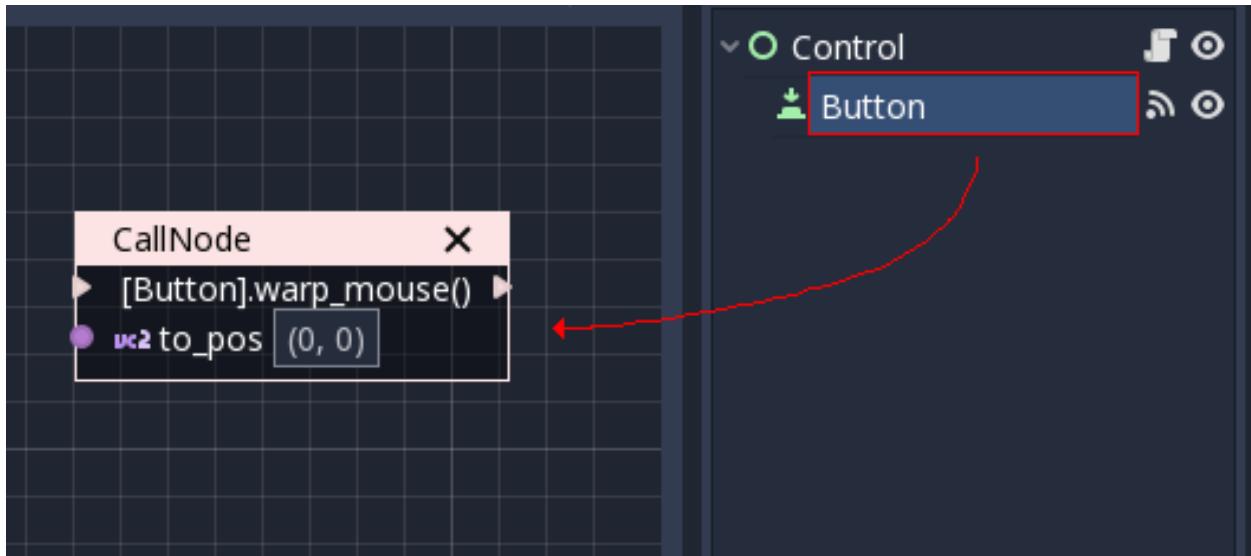


Adding Nodes

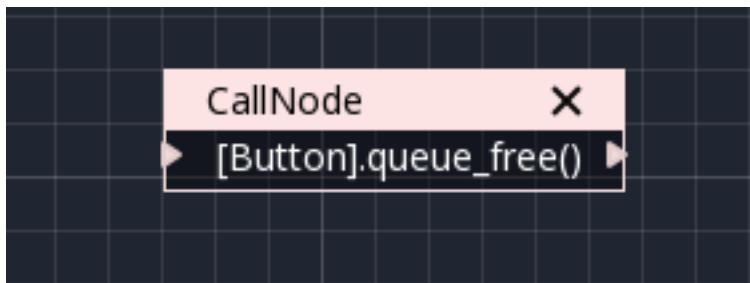
Finally! We got to the fun part! But, before explaining in more detail what each type of node does, let's take a short look at how nodes are most commonly added and dealt with.

Accessing Scene Nodes

One of the most common tasks is accessing Scene Tree Nodes (again, not to mistake with *Visual Script Nodes*). Dragging from the Scene Tree and dropping into the canvas will ask you to *call a method* (sometimes referred to as *member function*) on this node.



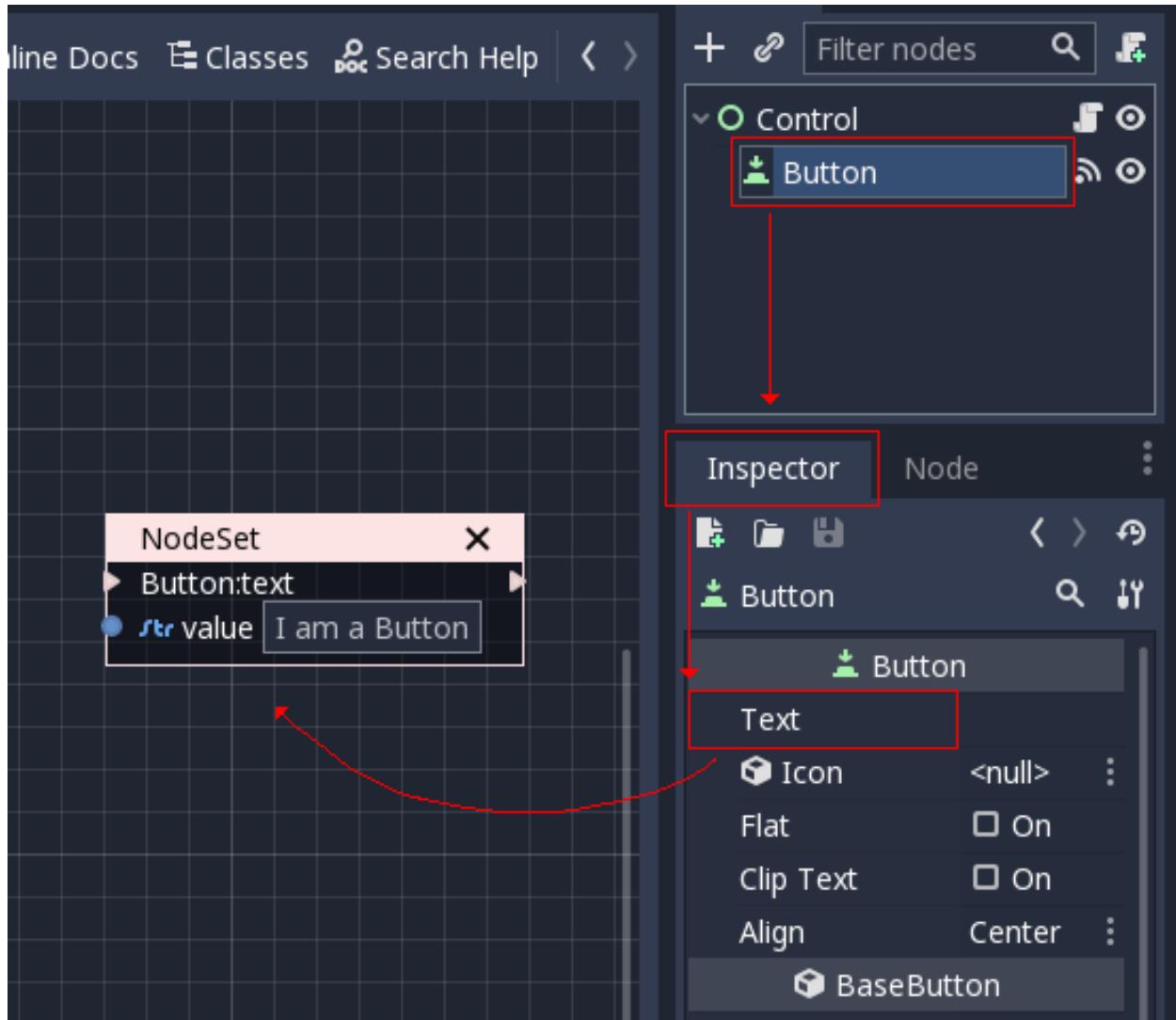
While accessing properties is desired in most cases (more on that below), sometimes *calling methods* can be useful too. Methods execute specific actions on objects. In the above case, the mouse pointer can be warped to a position in local coordinates to the control. Another common use case is queueing a node for deletion, which is done with the *queue_free* method.



Care must be taken that this only works if the scene being edited contains your *Visual Script* in one of the nodes! Otherwise, a warning will be shown.

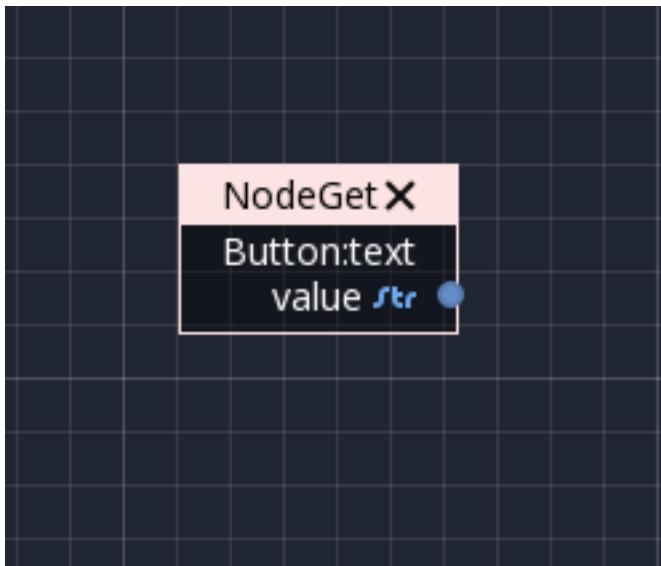
Accessing Scene Node Properties

This is the most common way to edit *Scene Nodes* in Visual Scripting. Select a *Scene Node* from the *Scene Tree*, go to the Inspector, find *the Name* of the property you want to edit (hint, *not* the value!) and drag it to the canvas:



The result is that this value can be changed from your script by writing to a *Data Port*.

If instead reading this value is desired, just drag the node again but hold the *Control* key (or Command on Mac). This will create a getter:

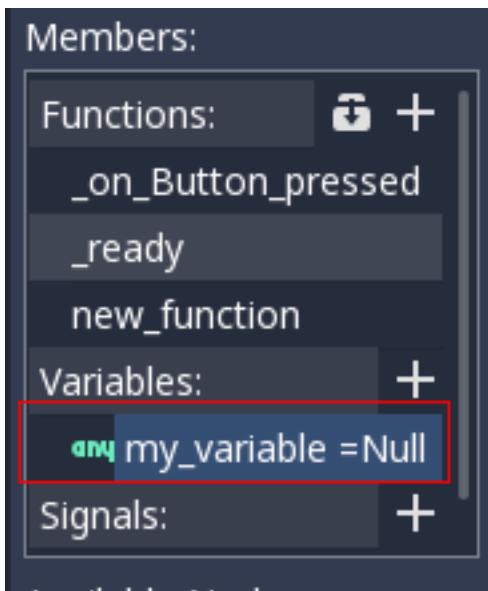


In this case, the value can be read from a *Data Port*.

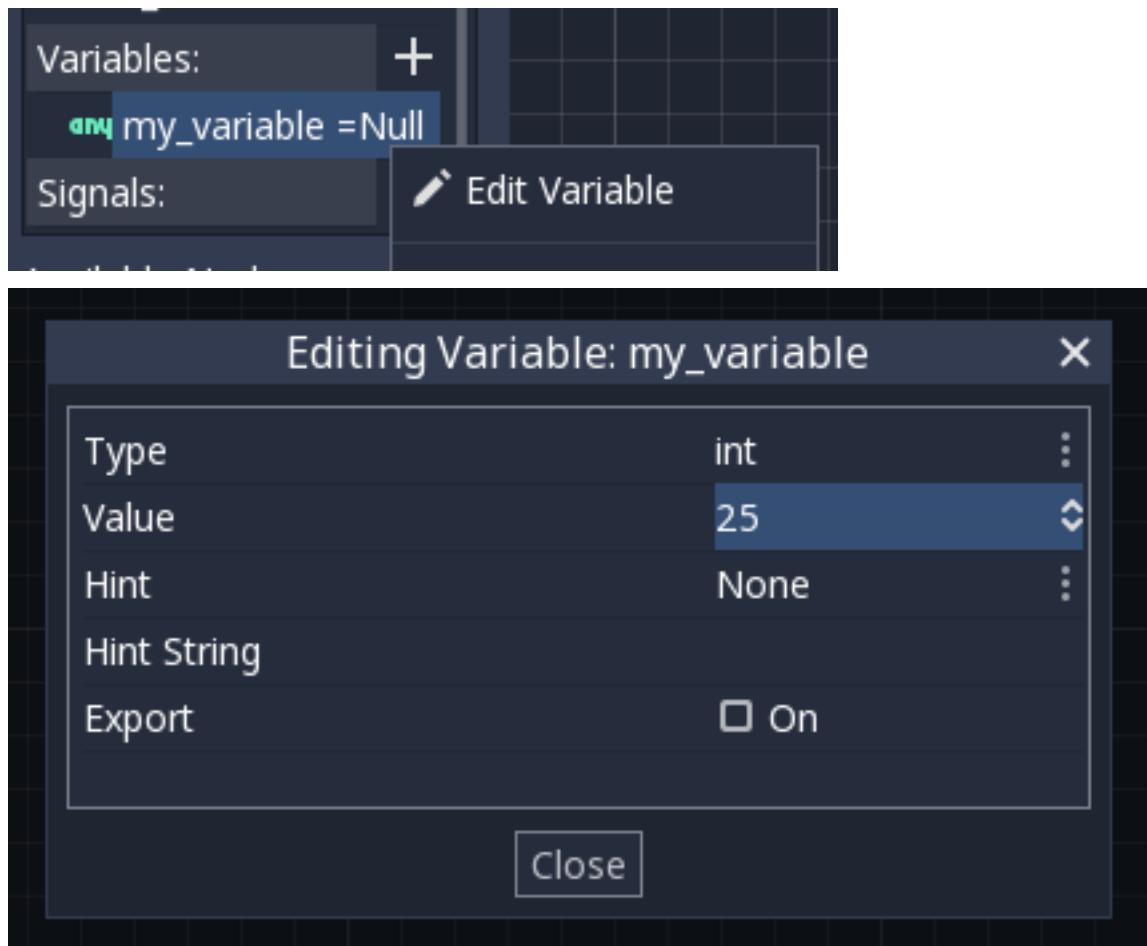
Variables

Variables are memory containers local to the script which can hold a value. This value can be read from any of the functions of the script or from other scripts via the method described in the previous step.

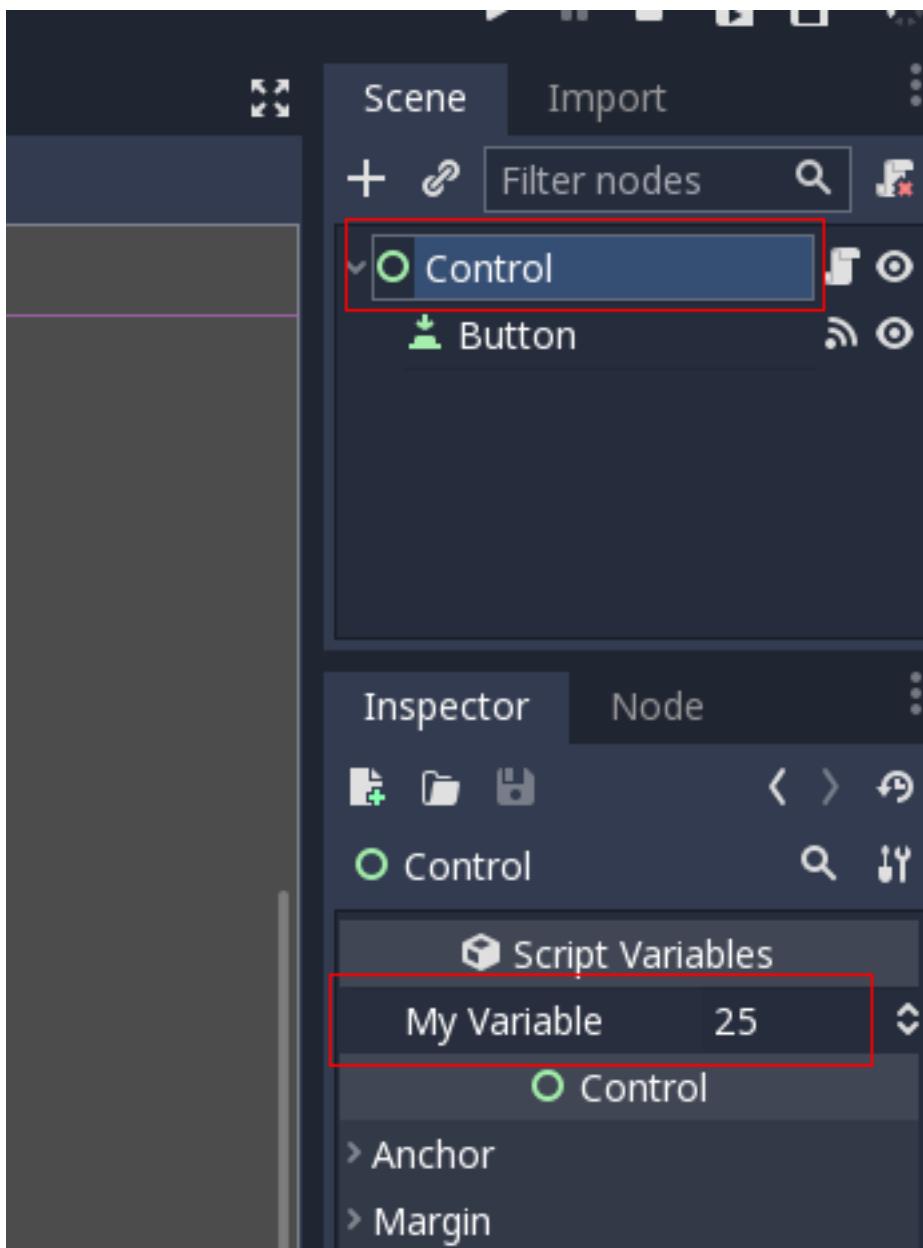
To add a Variable, push the “+” button on the *Variables* section of the Members panel. Double-click the new variable to rename it:



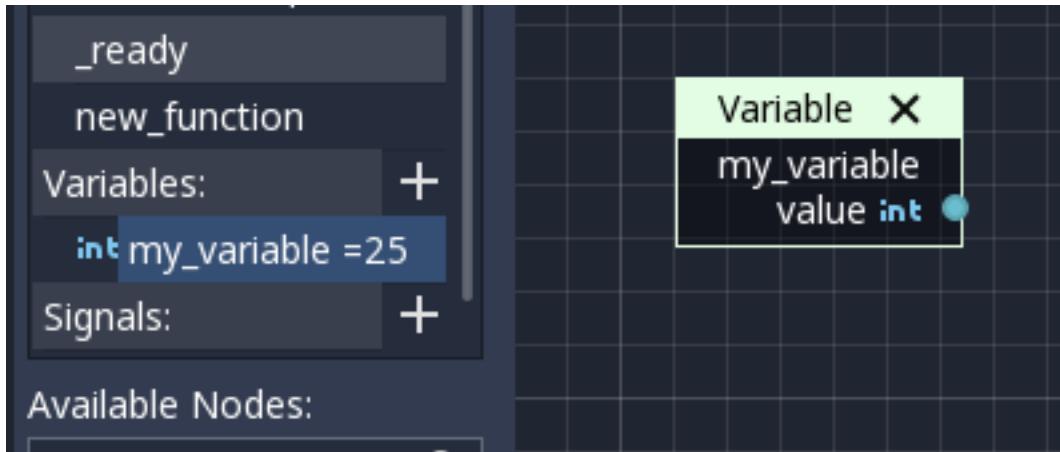
Right-clicking the variable allows you to configure its properties:



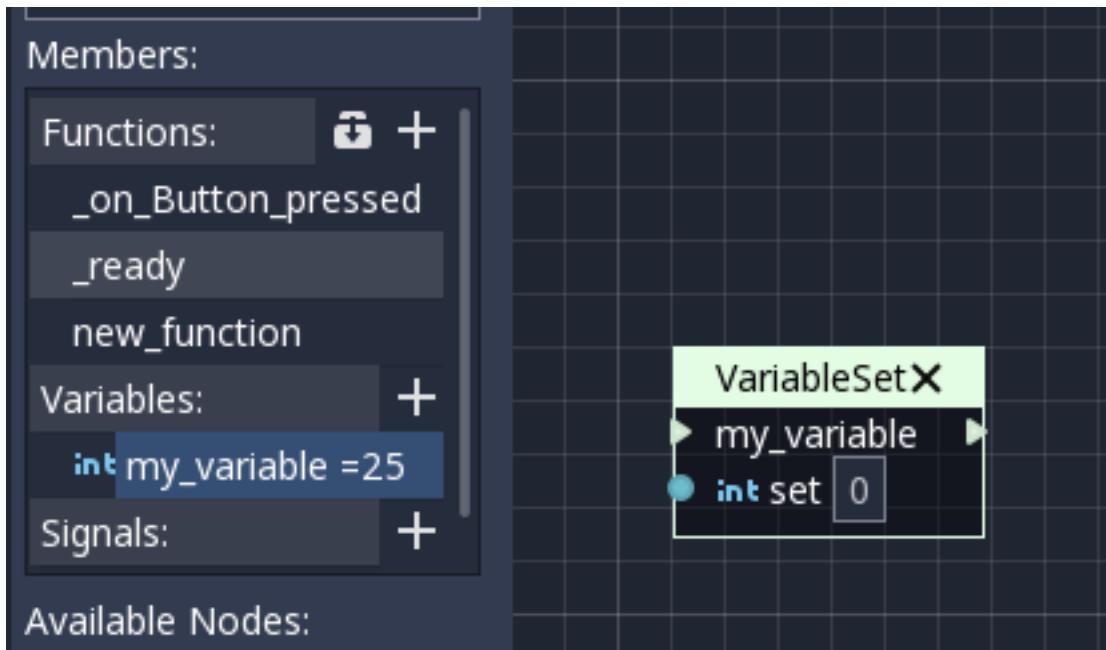
As it can be seen above, the type and initial value of the variable can be changed, as well as some property hints (@TODO, document this). Ticking the “Export” options makes the variable visible in the Inspector when selecting the node. This also makes it available to other scripts via the method described in the previous step.



To use the variable in the script, simply drag it to the canvas to create a getter:

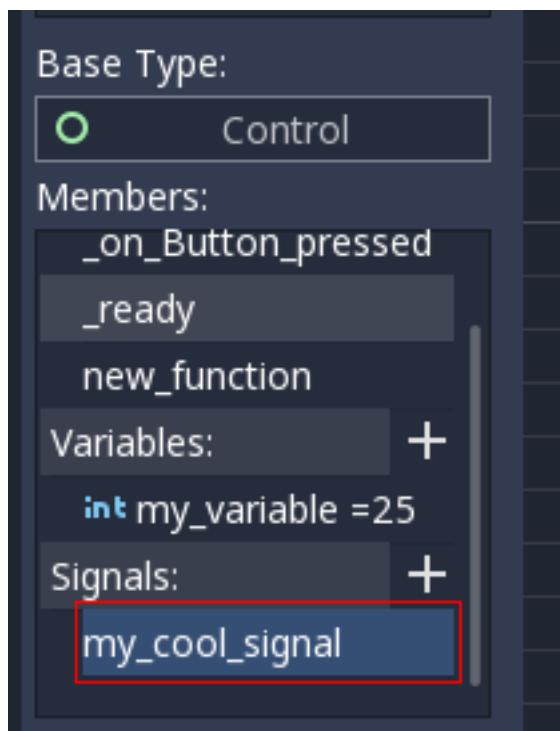


Likewise, hold *Control* (*Command* on Mac) to drop a setter:



Signals

It is also possible to create your own signals in a script and use them. For this, do the same steps you did for variables in the previous step, except for *Signals*:

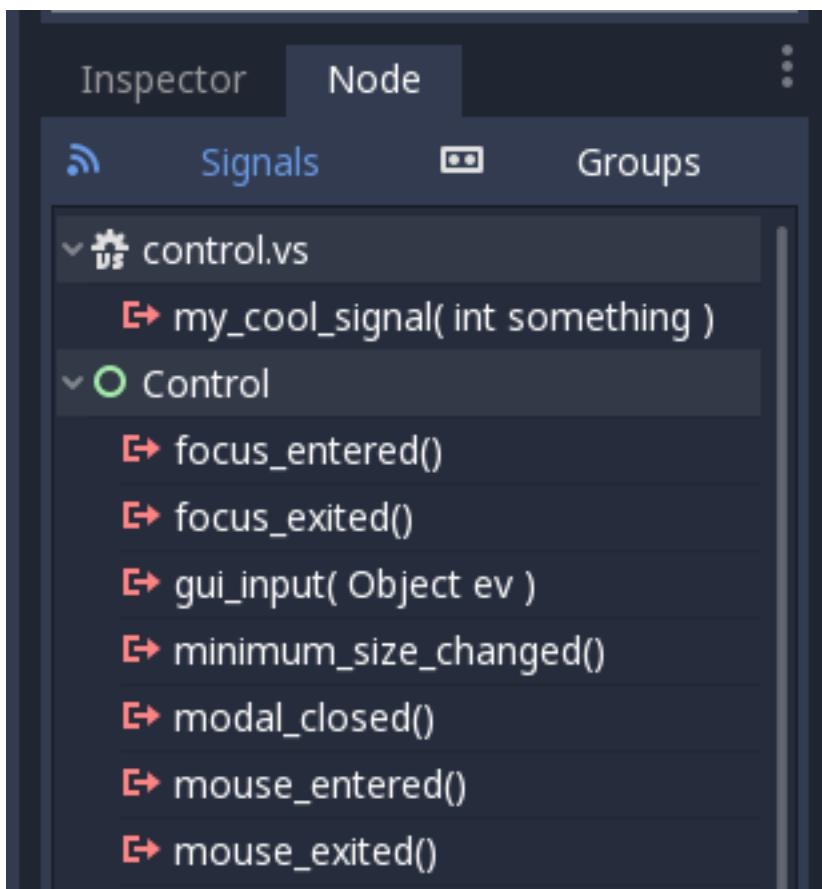


A signal can also be edited via right-click menu to customize its arguments:

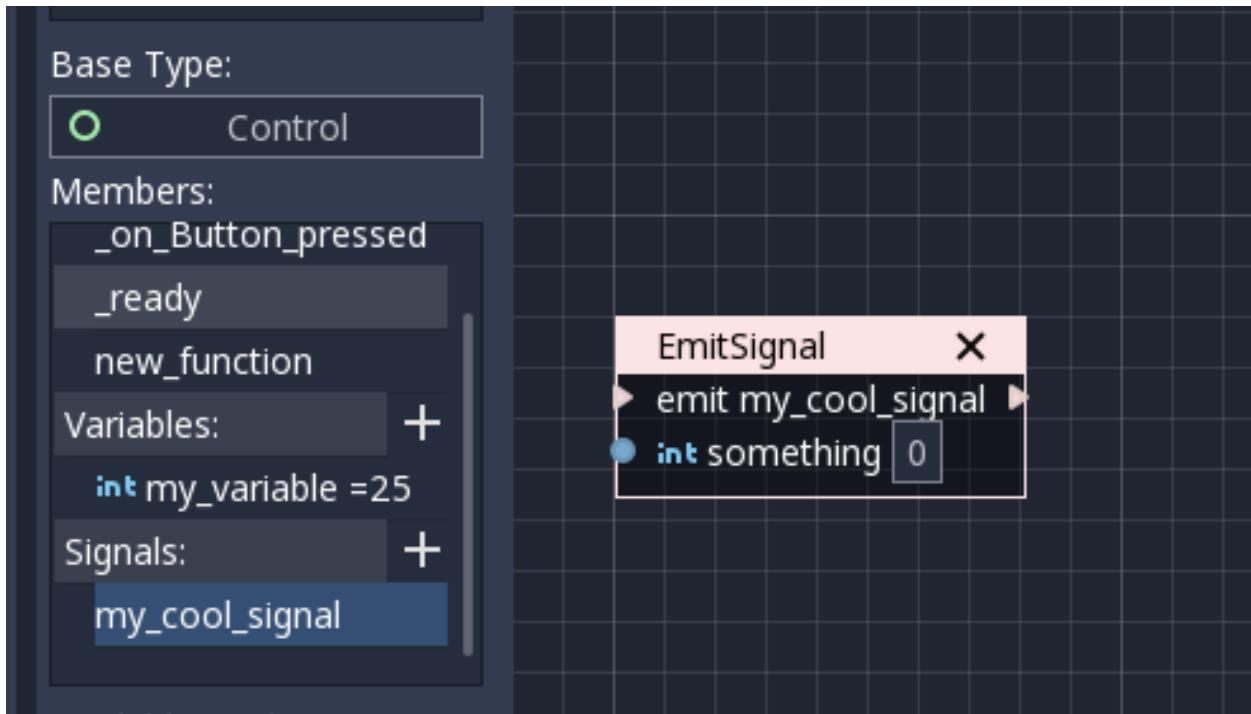


The signal you have just created will appear in the Inspector along with the built-in node signals. This allows you to

connect it from another script from another *Scene Node*:



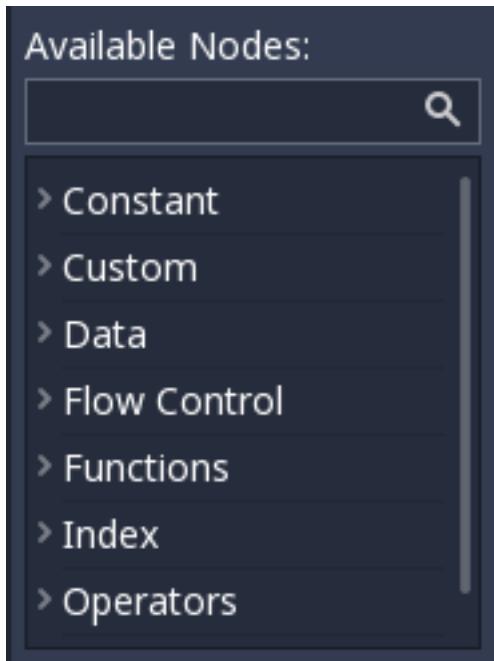
Finally, to emit the signal, simply drag it to the canvas:



Remember that emitting a signal is a sequenced operation, so it must come from a Sequence port.

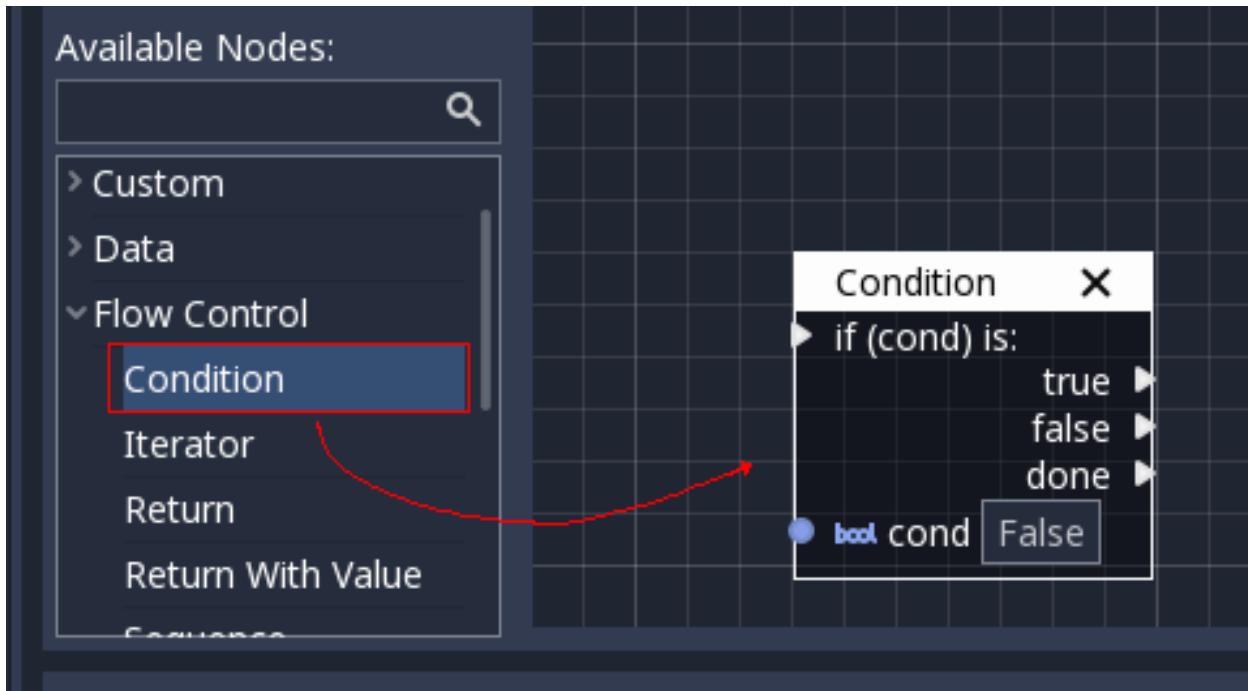
Adding More Nodes

Now that the basics are covered, let's discuss the large amount of utility nodes available for your canvas! Below the member panel, exists the list of all available node types:



Ctrl-F (Command-F on Mac) allows you to search the list.

Any of them can be dragged to the scene. Unlike nodes (e.g. dragging a property from the Inspector sets the context to the node being edited automatically), these are added without any “contextual” information, so this has to be done manually.



Remember that you can check the class reference for what each node does, as they are documented there. That mentioned, a brief overview of node types follows:

Constants

Constant nodes are nodes that provide values that, while not changing over time, can be useful as reference values. Most of the time they are integer or float.



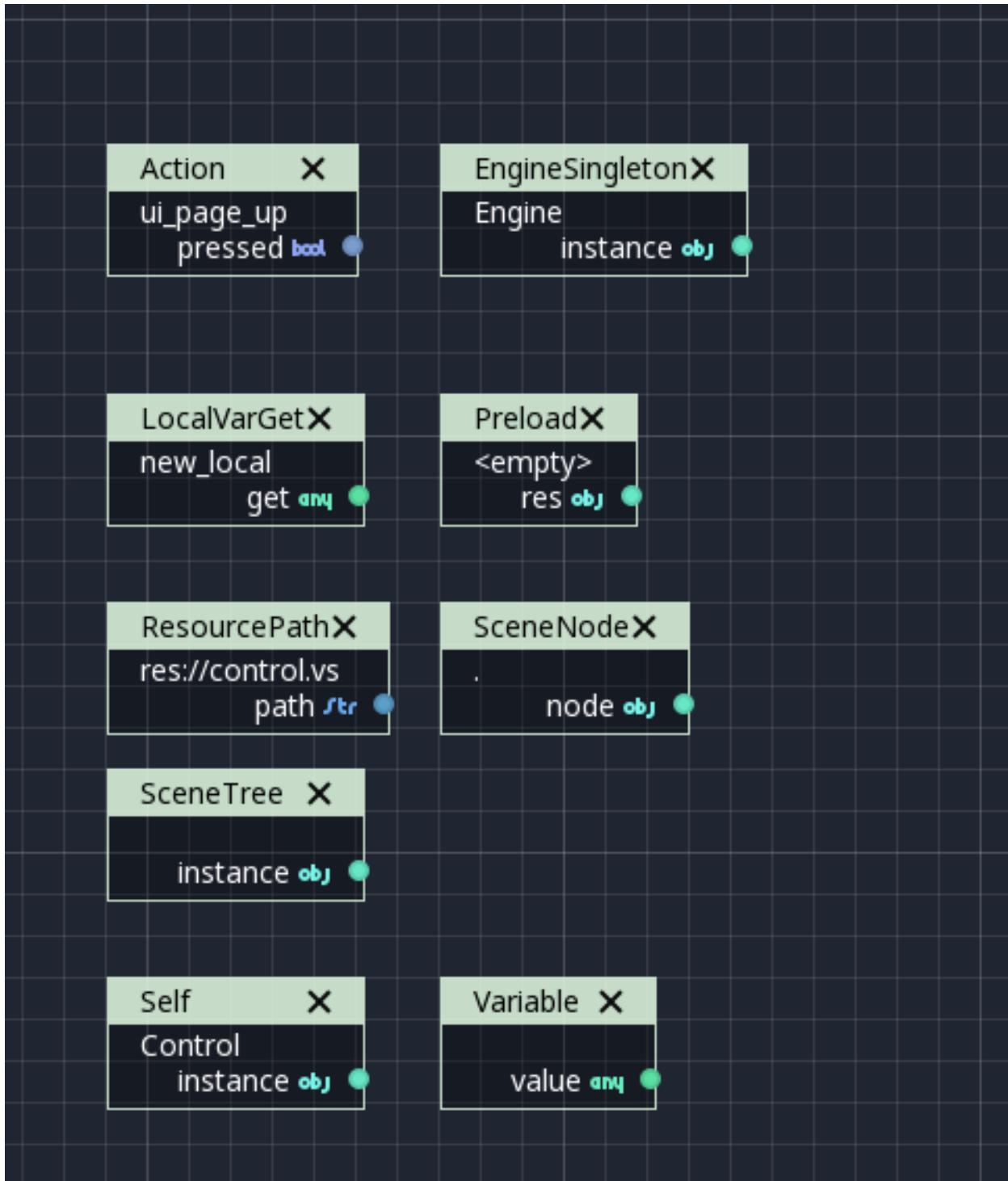
The first one is “Constant” which allows you to select any value of any type as constant, from an integer (42) to a String (“Hello!”). In general this node is not used that often because of default input values in *Data Ports*, but it’s good to know it exists.

The second is the GlobalConstant node, which contains a long list of constants for global types in Godot. In there you can find some useful constants to refer to key names, joystick or mouse buttons, etc.

The third one is MathConstant, which provides typical mathematical constants such as PI, E, etc.

Data

Data nodes deal with all sorts of access to information. Any information in Godot is accessed via these nodes, so they are some of the most important ones to use and pretty diverse.

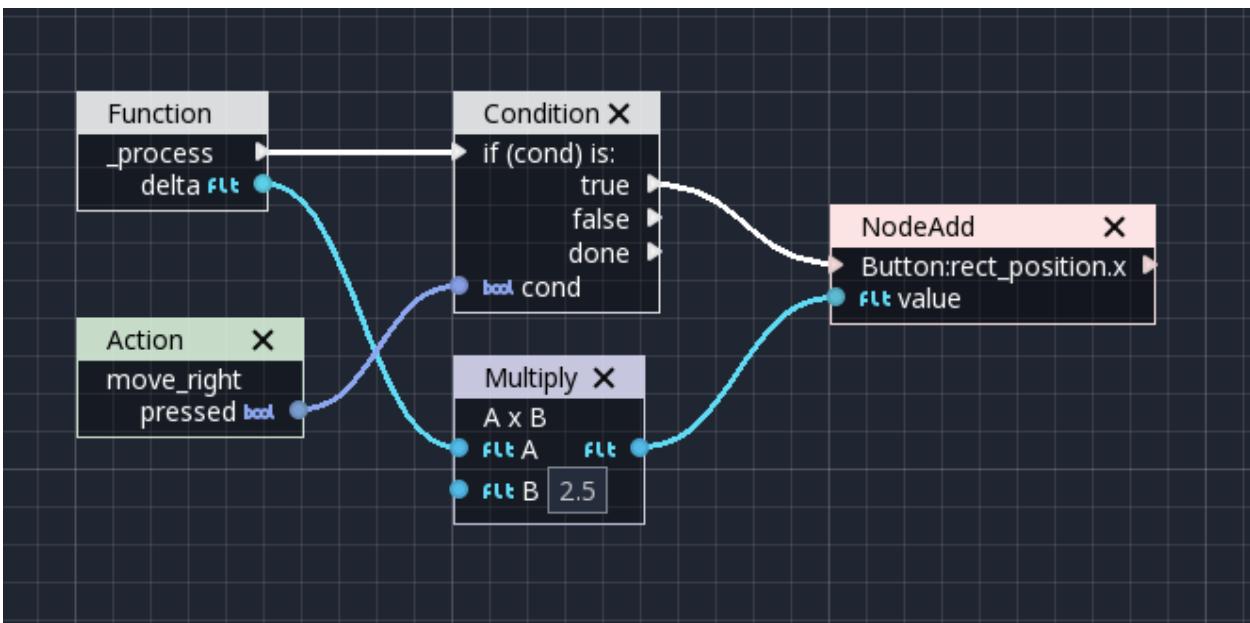


There are many types of nodes of interest here, so a short attempt to describe them will follow:

Action

Action nodes are vital when dealing with input from a device. You can read more about actions in the (@TODO ACTION TUTE LINK). In the following example below, the control is moved to the right when the “move_right”

action is pressed.

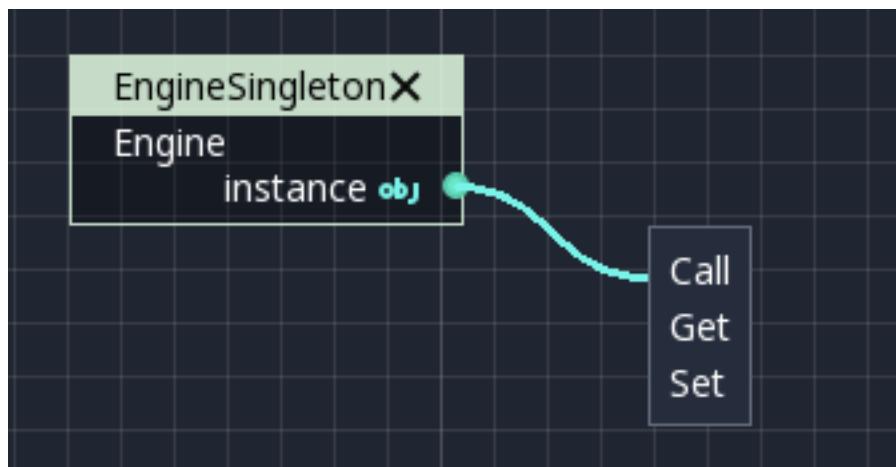


Engine Singleton

Engine singletons are global interfaces (meaning they can be accessed without a reference, unlike Scene Nodes, they are always available). They have several purposes, but in general they are useful for low level access or OS-related access.

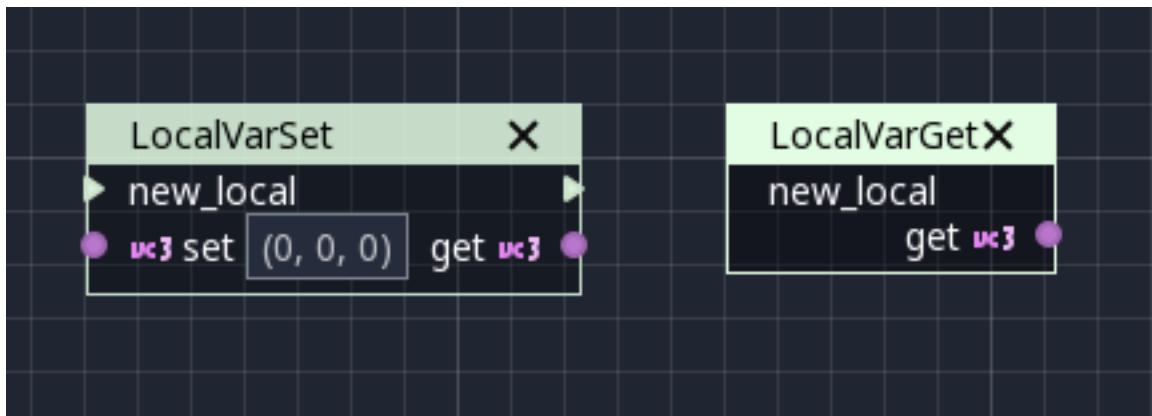
Performance
ProjectSettings
IP
Geometry
ResourceLoader
ResourceSaver
OS
Engine
ClassDB
Marshalls
TranslationServer
Input
InputMap
VisualServer
AudioServer
PhysicsServer
Physics2DServer
ARVRServer

Remember that dragging a connection to empty space will help you call functions or set/get properties on these:



Local Variables

These are nodes you can use as temporary storage for your graphs. Just make sure they all have the same name and type when using them and they will reference the same piece of memory.



As it can be seen above, there are two nodes available: A simple getter, and a sequenced getter (setting requires a sequence port).

Scene Node

This is just a reference to a node in the tree, but it's easier to use this node by just dragging the actual node from the scene tree to the canvas (this will create it and configure it).

Self

In some rare occasions, it may be desired to pass this Scene Node as argument. It can be used to call functions and set/get properties, or just drag nodes (or even the node itself that has the script) from the Scene Tree to the canvas for this.

SceneTree

This node is similar to the Singleton node because it references the SceneTree, which contains the active scene. SceneTree, however, only works when the node is sitting in the scene and active, otherwise accessing it will return as an error.

SceneTree allows for many low level things, like setting stretch options, calling groups, make timers, or even load another scene. It's a good class to get familiar with.

Preload

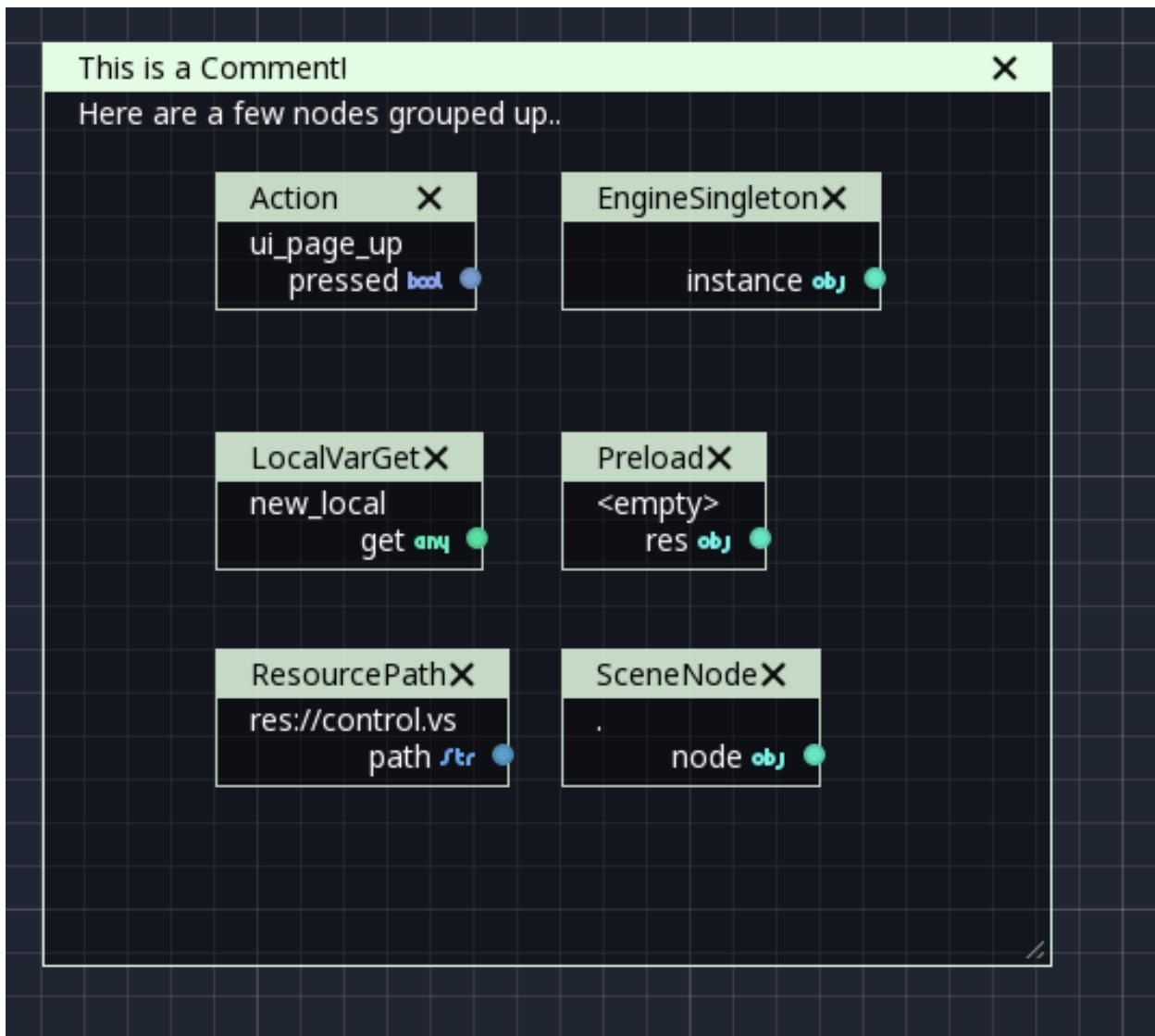
This does the same function as preload() in GDScript. It maintains this resource loaded and ready to use. Rather than instancing the node, it's simpler to just drag the desired resource from the filesystem dock to the canvas.

Resource Path

This node is a simple helper to get a string with a path to a resource you can pick. It's useful in functions that load things from disk.

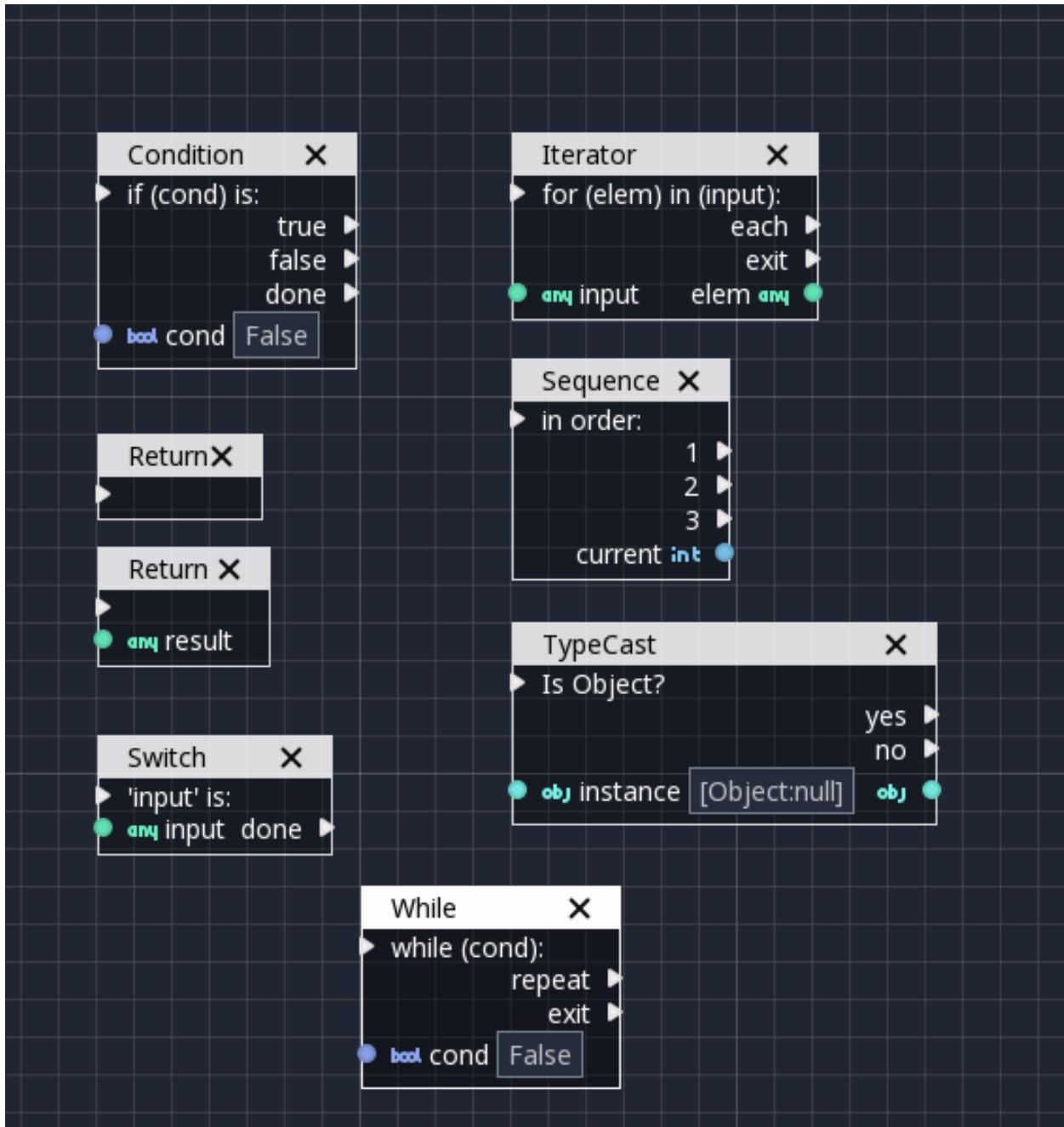
Comment

A Comment node works as a node you can resize to put around other nodes. It will not try to get focus or be brought to top when selecting it. It can also be used to write text on it.



Flow Control

Flow control nodes allow the execution to take different branches, usually depending on a given condition.



Condition

This is a simple node that checks a bool port. If true, it will go via the “true” sequence port. If false, the second. After going for either of them, it goes via the “done” port. Leaving sequence ports disconnected is fine if not all of them are used.

Iterator

Some data types in Godot (ie, arrays, dictionaries) are iterable. This means that a bit of code can run for each element that it has.

The Iterator node goes through all elements and, for each of them, it goes via the “each” sequence port, making the element available in the “elem” data port.

When done, it goes via the “exit” sequence port.

Return

Some functions can return values. In general for virtual ones, Godot will add the Return node for you. A return node forces the function to end.

Sequence

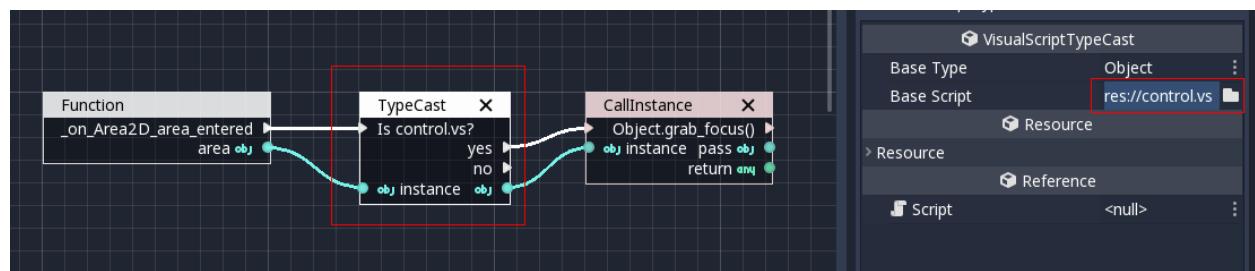
This node is useful mostly for organizing your graph. It calls its sequence ports in order.

TypeCast

This is a very useful and commonly used node. You can use it to cast arguments or other objects to the type you desire. Afterwards, you can even drag the object output to get full completion.



It is also possible to cast to a script, which will allow complete script properties and functions:



Switch

The Switch node is similar to the Condition node, but it matches many values at the same time.

While

This is a more primitive form of iteration. “repeat” sequence output will be called as long as the condition in the “cond” data port is met.

Functions

Functions are simple helpers, most of the time deterministic. They take some arguments as input and return an output. They are almost never sequenced.

Built-In

There is a list of built in helpers. The list is almost identical to the one from GDScript (@TODO, link to gdscript methods?). Most of them are mathematical functions, but others can be very useful helpers. Just make sure to take a look at the list at some point.

By Type

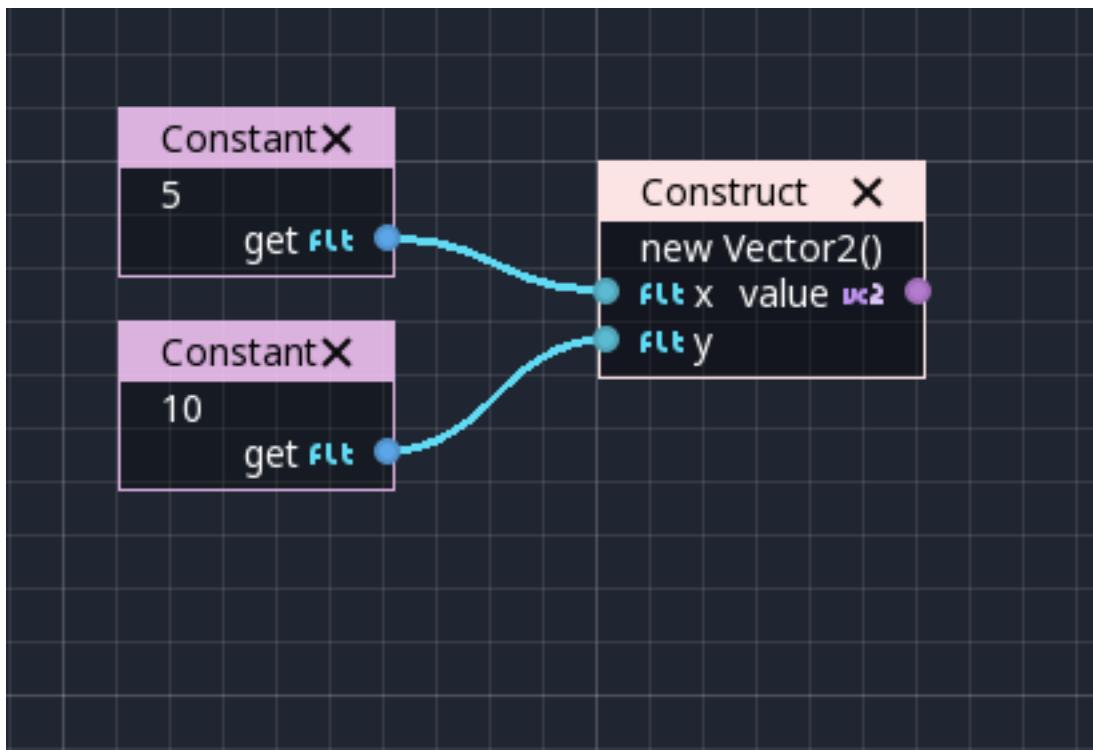
Those are the methods available to basic types. For example, if you want a dot-product, you can search for “dot” instead of the Vector3 category. In most cases just search the list of nodes, it should be faster.

Call

This is the generic calling node. It is rarely used directly but by dragging to empty space on an already configured node.

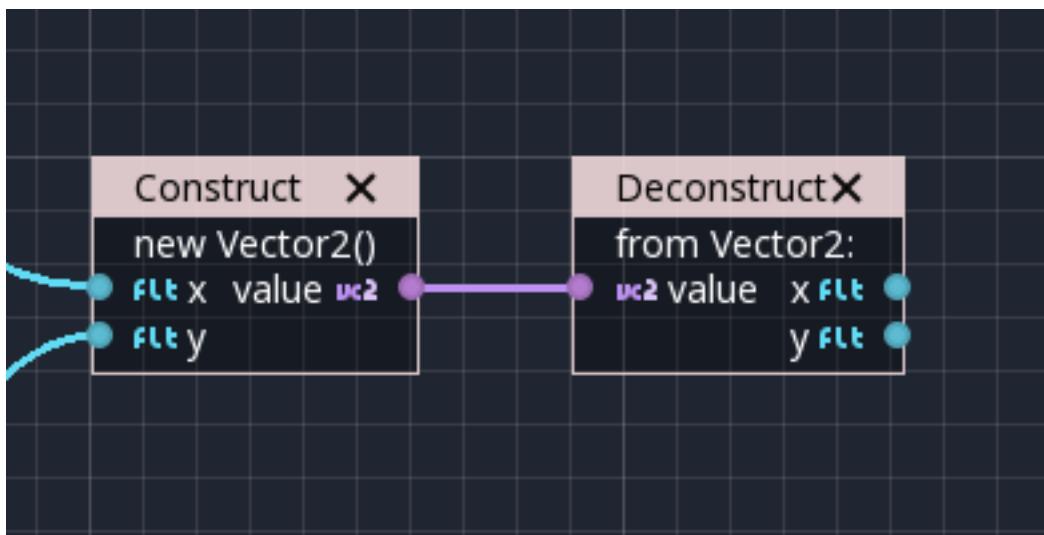
Constructors

These are all the functions needed to create Godot basic datatypes. For example, If you need to create a Vector3 out of 3 floats, a constructor must be used.



Destructor

This is the opposite to Constructor, it allows to separate any basic type (ie, Vector3) into its sub-elements.



Emit Signal

Emits signals from any object. In general it's not very useful, as dragging a signal to the canvas works better.

Get/Set

Generic Getter/Setter node. Dragging properties from the Inspector works better, as they appear properly configured on drop.

Wait

The Wait nodes will suspend execution of the function until something happens (many frames can pass until resuming, in fact). Default nodes allow you to wait for a frame to pass, a fixed frame or a given amount of time until execution is resumed.

Yield

This node completely suspends the execution of the script, and it will make the function return a value that can be used to resume execution.

Yield Signal

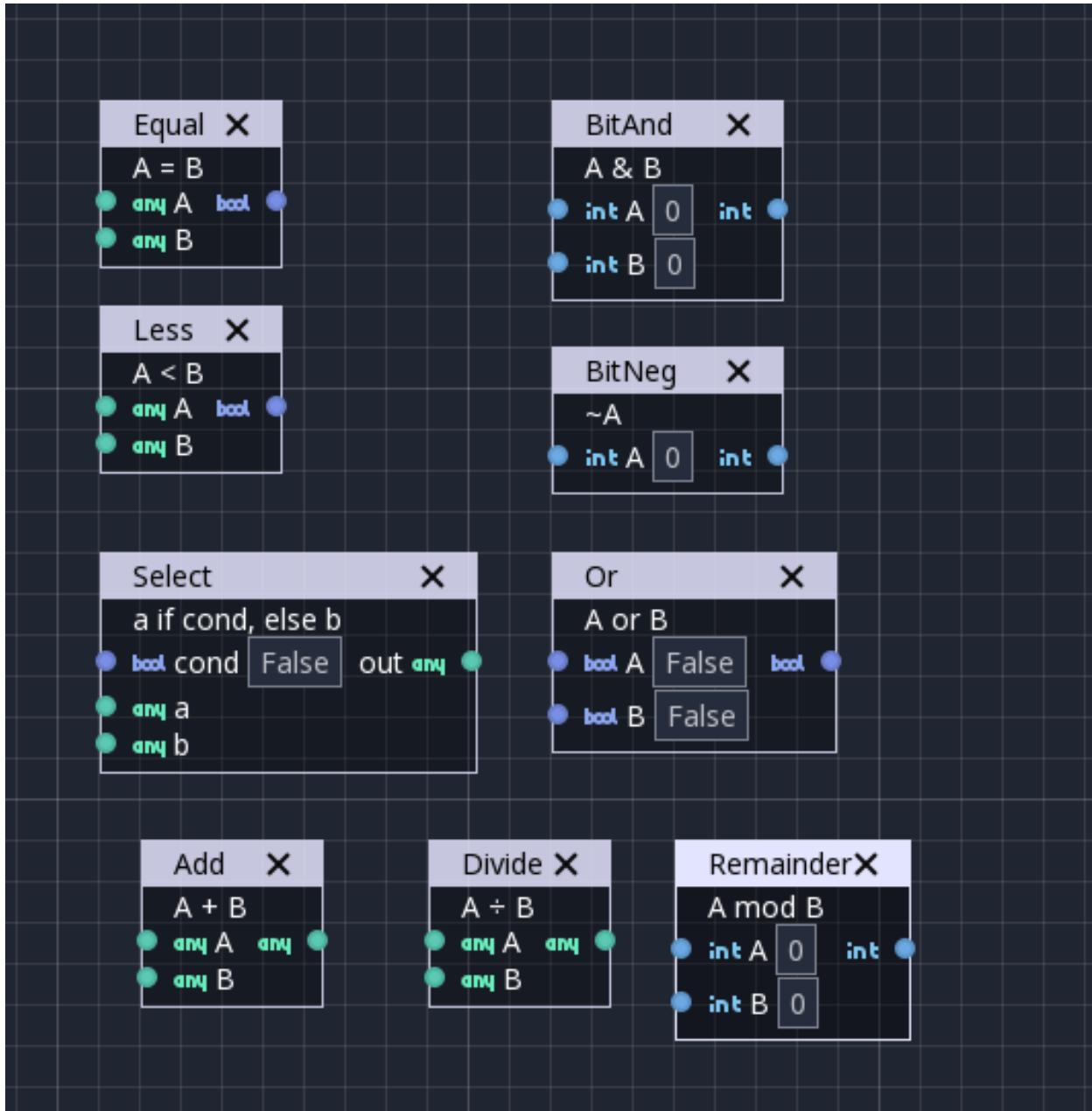
Same as Yield, but will wait until a given signal is emitted.

Index

Generic indexing operator, not often used but it's good that it exists just in case.

Operators

These are mostly generic operators such as addition, multiplication, comparison, etc. By default, these mostly accept any datatype (and will error in run-time if the types feeded do not match for the operator). It is always recommended to set the right type for operators to catch errors faster and make the graph easier to read.

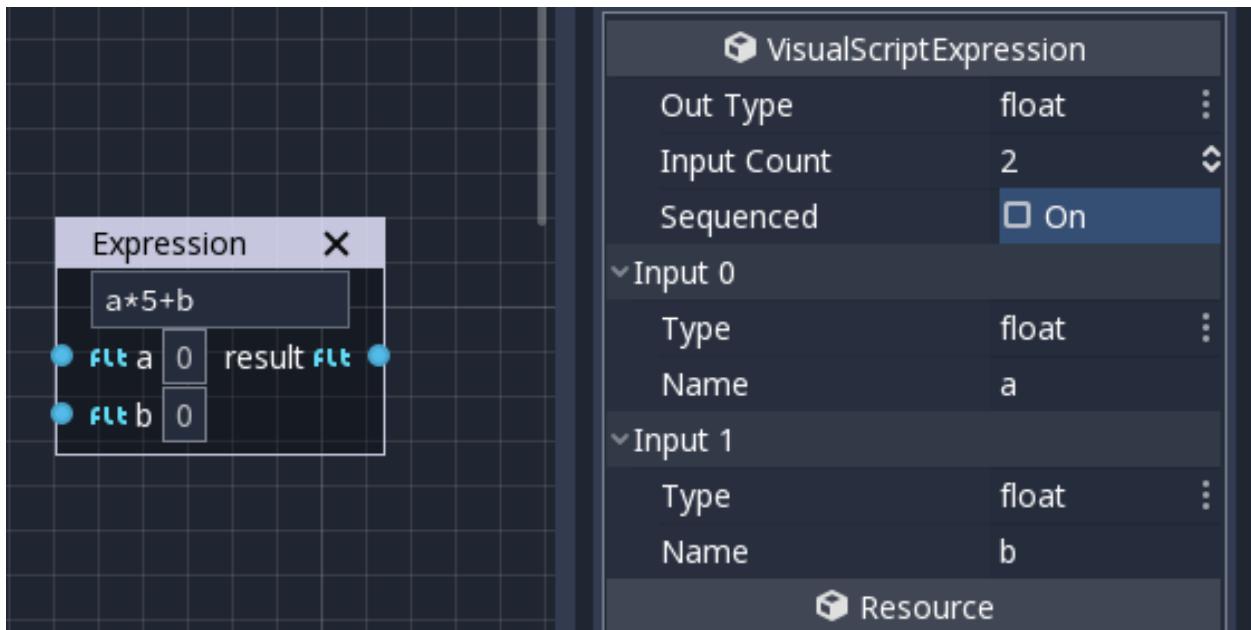


Expression Node

Among the operators, the *Expression* node is the most powerful. If well used, it allows you to enormously simplify visual scripts that are math or logic heavy. Just type any expression on it and it will be executed in real-time.

Expression nodes can:

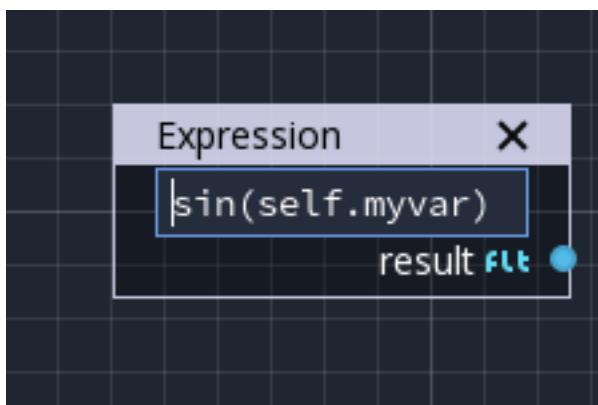
- Perform math and logic expressions based on custom inputs (eg: “ $a*5+b$ ”, where a and b are custom inputs):



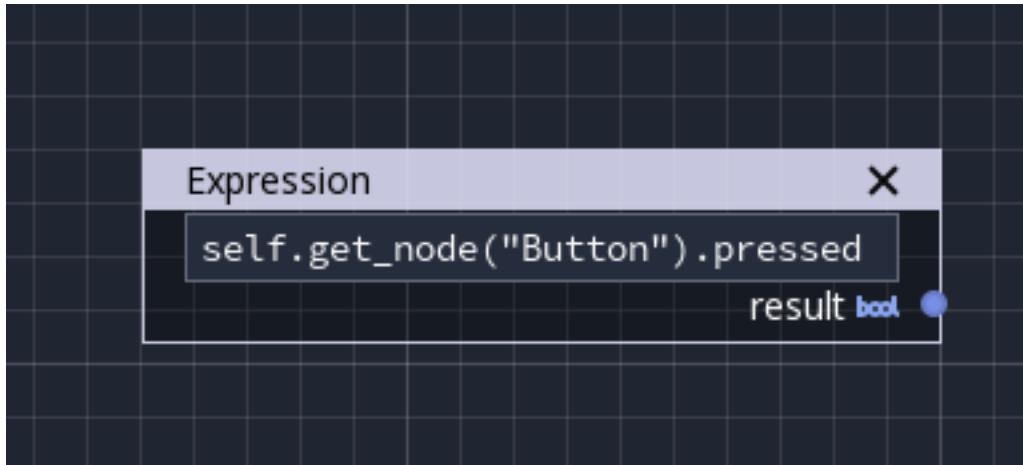
- Access local variables or properties:



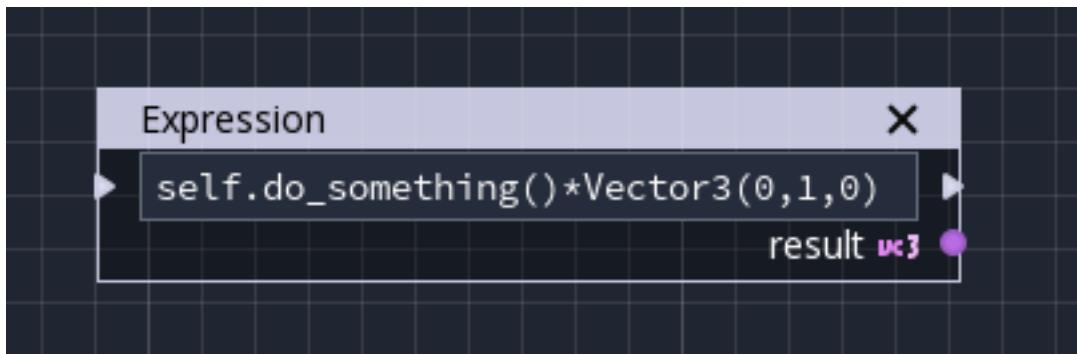
- Use most of the existing built-in functions that are available to GDScript, such as sin(),cos(),print(), as well as constructors, such as Vector3(x,y,z),Rect2(..), etc.:



- Call API functions:



- Use sequenced mode, which makes more sense in case of respecting the processing order:



4.3 C#

4.3.1 Introduction

Warning: C# support is a new feature in Godot 3.0. As such, you may still run into some issues, or find spots where the documentation could be improved. Please report issues with C# in Godot on the [engine Github page](#). And any documentation issues on the [documentation Github Page](#).

This page provides a brief intro to C#, both what it is and how to use it in Godot. Afterwards, you may want to look at *how to use specific features*, read about the *differences between the C# and the GDScript API* and (re)visit the *Scripting section* of the step-by-step tutorial.

C# is a high-level programming language developed by Microsoft. In Godot it is implemented with the Mono 5.2 .NET framework including full support for C# 7.0. Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C# and the Common Language Runtime. A good starting point for checking its capabilities is the [Compatibility](#) page in the Mono documentation.

Note: This is **not** a full-scale tutorial on the C# language as a whole. If you aren't already familiar with its syntax or features, see the [Microsoft C# guide](#) or look for a suitable introduction elsewhere.

Setup C# for Godot

To use C# in Godot you must have [Mono](#) installed (at least version 5.2), as well as MSBuild (at least version 15.0) which should come with the Mono installation.

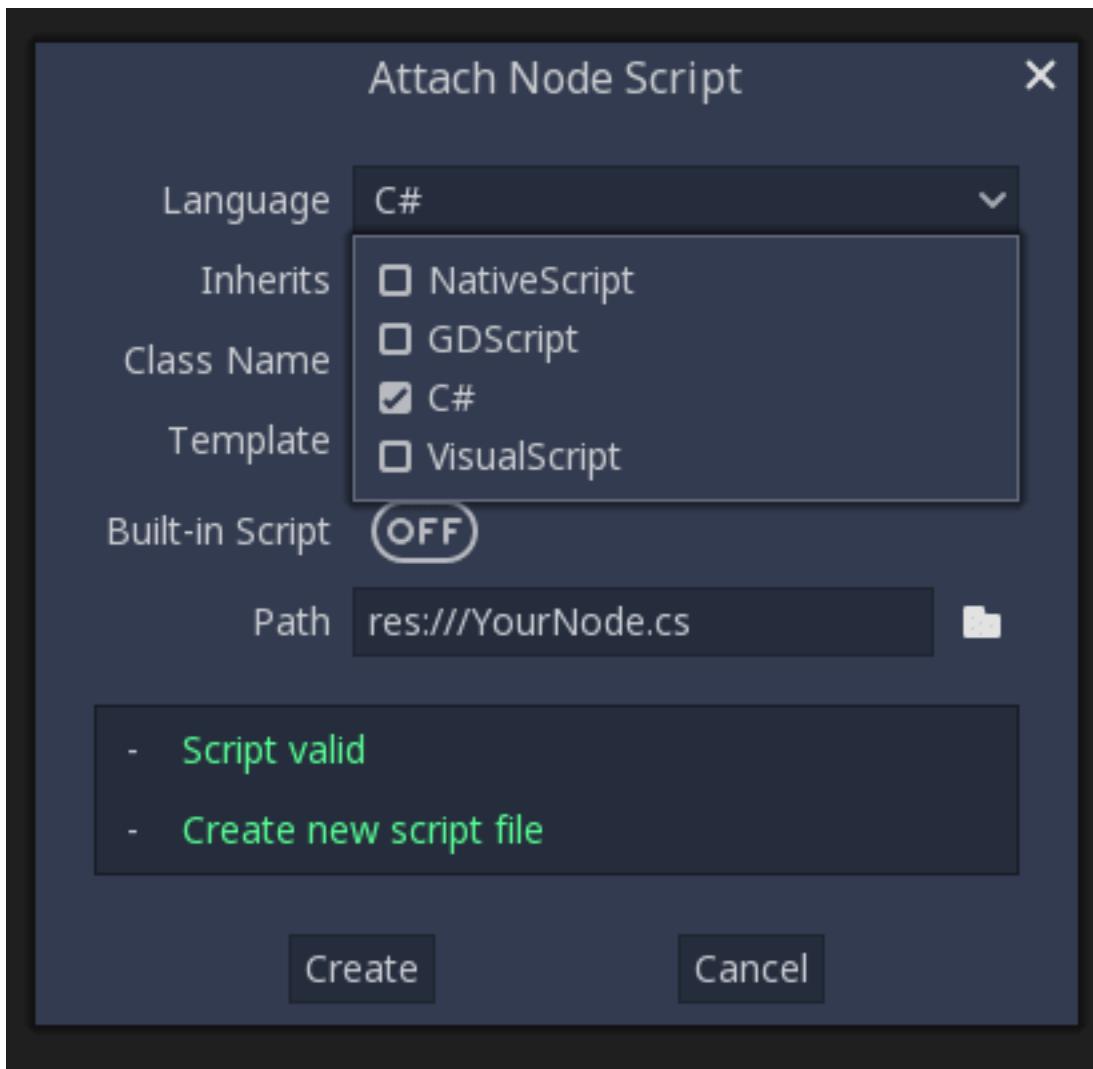
Additionally, your Godot version must have Mono support enabled, so take care to download the [Mono version](#) of Godot. If you are building Godot from source, make sure to follow the steps to include Mono support in your build outlined on the [Compiling with Mono](#) page.

Configuring an external editor

While Godot does have its own scripting editor, its support for C# is kept minimal, and it's recommended that you use an external IDE or editor, such as Microsoft Visual Studio Code, or MonoDevelop, which provide auto-completion, debugging and other features useful when working with C#. To set it up, in Godot click on [Editor](#), then [Editor Settings](#). Scroll down to the bottom, to the [Mono](#) settings. Under Mono click on [Editor](#), and on that page choose your external editor of choice.

Creating a C# script

After you successfully setup C# for Godot, you should see the following option when selecting `Attach script` in the context menu of a node in your scene:



Note that while some specifics change, most of the things work the same when using C# for scripting. If you're new to Godot, you may want to peruse the tutorials on *Scripting* at this point. While some places in the documentation still lack C# examples, most things can be transferred easily from GDScript.

Project setup and workflow

When you create the first C# script, Godot initializes the C# project files for your Godot project. This includes generating a C# solution (.sln) and project (.csproj) as well as some utility files and folders (.mono, sometimes Properties). All of these but .mono are important and should be kept in your version control system. .mono can be safely added to the ignore list of your VCS. When troubleshooting, it sometimes can help to delete the .mono folder and let it regenerate.

Note that currently there are some issues where the Godot and the C# project don't stay in sync; if you delete, rename or move things like scripts or nodes, they may no longer match up. In this case, it can help to edit the solution files manually.

Example: If you created a script (e.g. Test.cs) and delete it in Godot, compilation will fail because the now missing file is still expected to be there by the CS project. You can for now simply open up the .csproj and look for the ItemGroup, there should be a line included like the following:

```
<ItemGroup>
    <Compile Include="Test.cs" />` 
    <Compile Include="AnotherTest.cs" />` 
</ItemGroup>
```

Simply remove that line and your project should now again build fine. Same for renaming and moving things, simply rename and move them in the project file if needed.

Example

Here's a blank C# script with some comments to demonstrate how it works.

```
using Godot;
using System;

public class YourCustomClass : Node
{
    // Member variables here, example:
    private int a = 2;
    private string b = "textvar";

    public override void _Ready()
    {
        // Called every time the node is added to the scene.
        // Initialization here
        GD.Print("Hello from C# to Godot :)");
    }

    public override void _Process(float delta)
    {
        // Called every frame. Delta is time since last frame.
        // Update game logic here.
    }
}
```

As you can see, the things normally in global scope in GDScript like Godot's print function are available in the GD namespace. For a list of those, see the class reference pages for [@GDScript](#) and [@GlobalScope](#).

General differences between C# and GDScript

The C# API uses PascalCase instead of snake_case in GDScript/C++. Where possible, fields and getters/setters have been converted to properties. In general, the C# Godot API strives to be as idiomatic as is reasonably possible.

For more, see the *API differences to GDScript* page.

Current gotchas and known issues

As C# support is quite new to Godot, there are some growing pains and things that still need to be ironed out. Below is a list of the most important issues you should be aware of when diving into C# in Godot, but if in doubt also take a look over the official [issue tracker for Mono issues](#).

- As explained above, the C# project isn't always kept in sync automatically when things are deleted, renamed or moved in Godot ([#12917](#))
- Signals can only be added with `add_user_signal()` and don't show up in UI ([#11956](#))

- Debug output like stack traces, file path and line numbers is missing on Windows (#14589)
- Writing editor plugins and tool scripts in C# is not yet supported
- Exporting a project may not yet work

Performance of C# in Godot

According to some preliminary [benchmarks](#), performance of C# in Godot - while generally in the same order of magnitude - is roughly ~4x that of GDScript in some naive cases. For full performance, C++ is still a little faster; the specifics are going to vary according to your use case. GDScript is likely fast enough for most general scripting workloads. C# is faster, but requires some expensive marshalling when talking to Godot.

4.3.2 Features

This page provided an overview over the commonly used features of both C# and Godot and how they are used together.

Type Conversion and Casting

C# is a statically typed language. Therefore you can't do the following:

```
var mySprite = GetNode("MySprite")
mySprite.SetFrame(0)
```

The method `GetNode()` returns a `Node` instance. You must explicitly convert it to the desired derived type, `Sprite` in this case.

For this, you have various options in C#.

Casting and Type Checking

Throws `InvalidOperationException` if the returned node cannot be casted to `Sprite`. You would use it instead of the `as` operator if you are pretty sure it won't fail.

```
Sprite mySprite = (Sprite)GetNode("MySprite");
mySprite.SetFrame(0);
```

Using the AS operator

The `as` operator returns `null` if the node cannot be casted to `Sprite`, and for this reason it cannot be used with value types.

```
Sprite mySprite = GetNode("MySprite") as Sprite;
// Only call SetFrame() if mySprite is not null
mySprite?.SetFrame(0);;
```

Type checking using the IS operator

To check if the node can be casted to `Sprite`, you can use the `is` operator. The `is` operator returns false if the node cannot be casted to `Sprite`, otherwise it returns true.

```
if (GetNode("MySprite") is Sprite)
{
    // Yup, it's a sprite!
}
```

For more advanced type checking, you can look into [Pattern Matching](#).

Signals

For a complete C# example, see the **Handling a signal** section in the step by step *Scripting* tutorial.

You can use `Connect("SomeSignal", someObject, "SomeMethod")` to connect to signals. `AddUserSignal("SignalName")` is used to define custom signals. Emitting signals is done with `EmitSignal("SignalName")`. Params can be given, like `EmitSignal("SignalName", arg1, arg2, ...)`.

Custom signals

```

1  public class ExampleScript : Node
2  {
3      public override void _Ready()
4      {
5          AddUserSignal("YourSignal");
6      }
7
8      public override void _Process(float delta)
9      {
10         EmitSignal("YourSignal");
11     }
12 }
```

Above in line 5, `AddUserSignal()` is used to define the new custom signal `YourSignal`. In line 10 `EmitSignal()` is used to emit that custom signal on every frame in `_Process()`.

Make sure that `AddUserSignal()` is always executed before any calls using that signal (`EmitSignal()` and `Connect()`). If you are using both `AddUserSignal()` and `Connect()` or `EmitSignal()` in `_Ready()`, this is especially important as load order of your node may change, and thus the order in which your various `_Ready()` functions are called.

4.3.3 API differences to GDScript

This is a (incomplete) list of API differences between C# and GDScript.

General Differences

As explained in the *Introduction*, C# generally uses `PascalCase` instead of the `snake_case` in GDScript and C++.

Global Scope

Available under `Godot.GD`. Some things were moved to their own classes, like `Math` and `Random`. See below.

Global functions like `print`, `var2str` and `weakref` are located under `GD` in C#.

`ERR_*` constants were moved to `Godot.Error`.

Math

Math functions like `abs`, `acos`, `asin`, `atan` and `atan2` are located under `Mathf` instead of in global scope. `PI` is `Mathf.PI`

Random

Random functions like `rand_range` and `rand_seed` are located under Random, so use `Random.Range` instead of `rand_range`.

Export keyword

Use the `[Export]` attribute instead of the GDScript `export` keyword.

Singletons

Singletons provide static methods rather than using the singleton pattern in C#. This is to make code less verbose and similar to GDScript. Example:

```
Input.IsActionPressed("ui_down")
```

String

Use `System.String(string)`. All the Godot String methods are provided by the `StringExtensions` class as extension methods. Example:

```
string upper = "I LIKE SALAD FORKS";
string lower = upper.ToLower();
```

There are a few differences though:

- `erase`: Strings are immutable in C#, so we cannot modify the string passed to the extension method. For this reason `Erase` was added as an extension method of `StringBuilder` instead of `string`. Alternatively you can use `string.Remove`.
- `IsSubsequenceOf/IsSubsequenceOfi`: An additional method is provided which is an overload of `IsSubsequenceOf` allowing to explicitly specify case sensitivity:

```
str.IsSubsequenceOf("ok"); // Case sensitive
str.IsSubsequenceOf("ok", true); // Case sensitive
str.IsSubsequenceOfi("ok"); // Case insensitive
str.IsSubsequenceOf("ok", false); // Case insensitive
```

- `Match/Matchn/ExprMatch`: An additional method is provided besides `Match` and `Matchn`, which allows to explicitly specify case sensitivity:

```
str.Match("*.txt"); // Case sensitive
str.ExprMatch("*.txt", true); // Case sensitive
str.Matchn("*.txt"); // Case insensitive
str.ExprMatch("*.txt", false); // Case insensitive
```

Basis

Structs cannot have parameterless constructors in C#, therefore `new Basis()` initializes all primitive members to their default value. Use `Basis.Identity` for the equivalent to `Basis()` in GDScript and C++.

The following methods were converted to properties with their respective names changed:

GDScript	C#
get_scale()	Scale

Transform2D

Structs cannot have parameterless constructors in C#, therefore new Transform2D() initializes all primitive members to their default value. Please use Transform2D.Identity for the equivalent to Transform2D() in GDScript and C++.

The following methods were converted to properties with their respective names changed:

GDScript	C#
get_origin()	Origin
get_rotation()	Rotation
get_scale()	Scale

Plane

The following methods were converted to properties with their respective names changed:

GDScript	C#
center()	Center

Rect2

The following fields were converted to properties with their respective names changed:

GDScript	C#
end	End

The following methods were converted to properties with their respective names changed:

GDScript	C#
get_area()	Area

Quat

Structs cannot have parameterless constructors in C#, therefore new Quat() initializes all primitive members to their default value. Please use Quat.Identity for the equivalent to Quat() in GDScript and C++.

Array

This is temporary. Array is ref-counted, so it will need its own type that wraps the native side. PoolArrays will also need their own type to be used the way they are meant to.

GDScript	C#
Array	object[]
PoolIntArray	int[]
PoolByteArray	byte[]
PoolFloatArray	float[]
PoolStringArray	String[]
PoolColorArray	Color[]
PoolVector2Array	Vector2[]
PoolVector3Array	Vector3[]

In some exceptional cases a raw array (`type []`) may be required instead of a `List`.

Dictionary

This is temporary. Array is ref-counted, so it will need its own type that wraps the native side.

Use `Dictionary<object, object>`.

Variant

`System.Object (object)` is used in place of `Variant`.

Communicating with other scripting languages

The methods `object Object.call(string method, params object[] args)`, `object Object.get(string field)` and `object Object.set(string field, object value)` are provided to communicate with instances of other scripting languages via the `Variant` API.

Other differences

`preload`, `assert` and `yield` as they work in GDScript are currently not available in C#.

Other differences:

GDScript	C#
Color8	Color.Color8
is_inf	float.IsInfinity
is_nan	float.IsNaN
dict2inst	? TODO
inst2dict	? TODO
load	GD.load which is the same as ResourceLoader.load

CHAPTER 5

Project workflow

5.1 Project setup

5.1.1 Project organization

Introduction

This tutorial is aimed to propose a simple workflow on how to organize projects. Since Godot allows the programmer to use the file-system as they please, figuring out a way to organize the projects when starting to use the engine can be a little challenging. Because of this, a simple workflow will be described, which can be used or not, but should work as a starting point.

Additionally, using version control can be challenging so this proposition will include that too.

Organization

Godot is scene based in nature, and uses the filesystem as-is, without metadata or an asset database.

Unlike other engines, a lot of resource are contained within the scene itself, so the amount of files in the filesystem is considerably lower.

Considering that, the most common approach is to group assets close to scenes as, when a project grows, it makes it more maintainable.

As example, base sprite images, 3D model scenes or meshes, materials, etc. can usually be organized in a place, while a separate folder is used to store built levels that use them.

```
/project.godot
/docs/.gdignore
/docs/learning.html
/models/town/house/house.dae
/models/town/house/window.png
/models/town/house/door.png
/characters/player/cubio.dae
```

```
/characters/player/cubio.png  
/characters/enemies/goblin/goblin.dae  
/characters/enemies/goblin/goblin.png  
/characters/npcs/suzanne/suzanne.dae  
/characters/npcs/suzanne/suzanne.png  
/levels/riverdale/riverdale.scn
```

Importing

Godot version previous to 3.0 did the import process from files outside the project. While this can be useful in very large projects, it resulted in an organization hassle for most developers.

Because of this, assets are now imported from within the project folder transparently.

If a folder shouldn't be imported into Godot an exception can be made with a `.gdignore` file.

5.2 Assets workflow

5.2.1 Import process

Importing assets in Godot 3.0+

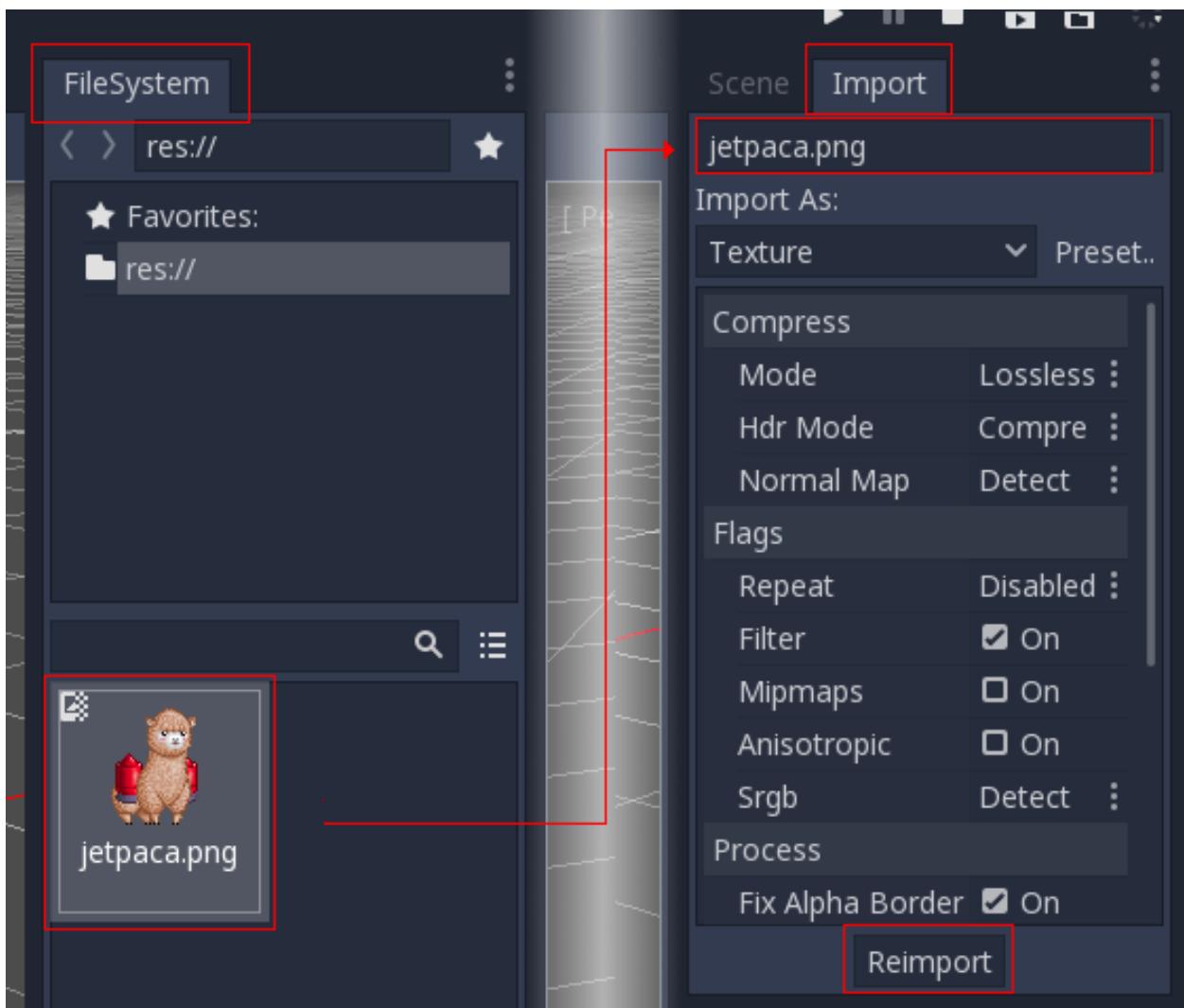
Previously, importing assets in Godot 2.x required manual maintenance of a separate directory with source assets. Without doing this, it was impossible to specify how to convert and change import flags for textures, audios, scenes, etc.

In Godot 3.0, we use a more modern approach to importing: Simply drop your assets (image files, scenes, audios, fonts, etc) directly in the project folder (copy them manually with your OS file explorer). Godot will automatically import these files internally and keep the imported resources hidden in a `res://.import` folder.

This allows changing all the import parameters transparently.

Changing import parameters

Changing the import parameters of an asset in Godot (again, keep in mind import parameters are only present in non-native Godot resource types) is easy. Just select in the filesystem dock the relevant resource:



And, after adjusting the parameters, just press “Reimport”. The parameters used will be only for this asset and will be used on future reimports.

Changing import parameters of several assets at the same time is also possible. Simply select all of them together in the resources dock and the exposed parameters will apply to all of them when reimporting.

Automatic reimport

When the MD5 checksum of the source asset changes, Godot will perform an automatic reimport of it, applying the preset configured for that specific asset.

Files generated

Importing will add an extra `<asset>.import` file, containing the import configuration. Make sure to commit these to your version control system!

```
$ ls
jetpaca.png
jetpaca.png.import
project.godot
```

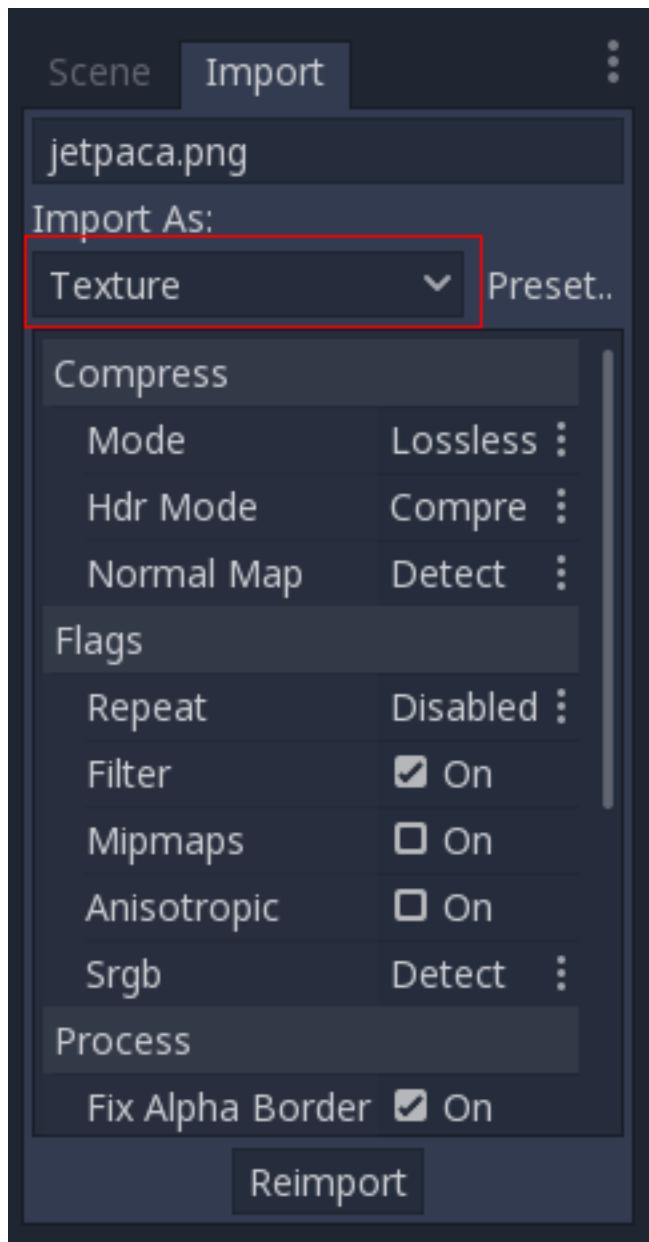
Additionally, extra assets will be present in the hidden `res://.import` folder:

```
$ ls .import/
jetpaca.png-132feef2cf69a8b25d46e5d16825ba89.stex
```

If any of the files present in this folder is erased (or the whole folder), the asset or assets will be reimported automatically. As such, committing this folder to the version control system is optional. It can save time on reimporting time when checking out in another computer, but it takes considerably more space and transfer time. Pick your poison!

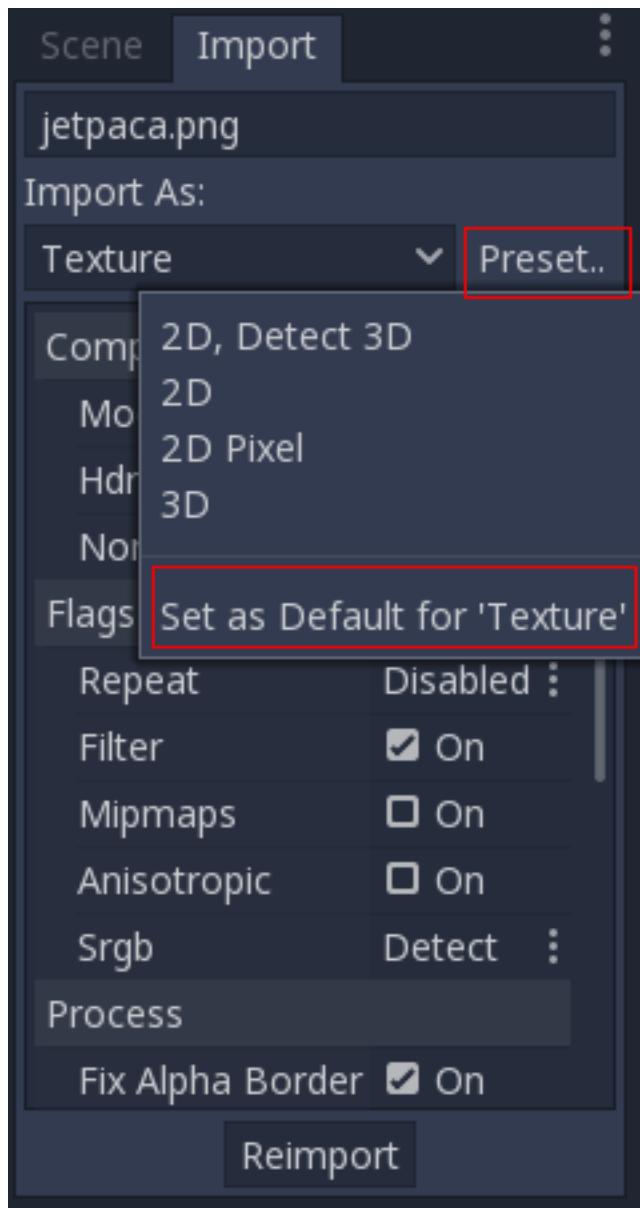
Changing import resource type

Some source assets can be imported as different types of resources. For this, just select the relevant type of resource desired and press “Reimport”:



Changing default import parameters

Different types of games might require different defaults. Changing the defaults per project can be achieved by using the “Preset..” Menu. Besides some resource types offering presets, the default setting can be saved and cleared too:



Simplicity is key!

This is a very simple workflow which should take very little time to get used to. It also enforces a more correct way to deal with resources.

There are many types of assets available for import, so please continue reading to understand how to work with all of them!

5.2.2 Importing Images

Why importing them?

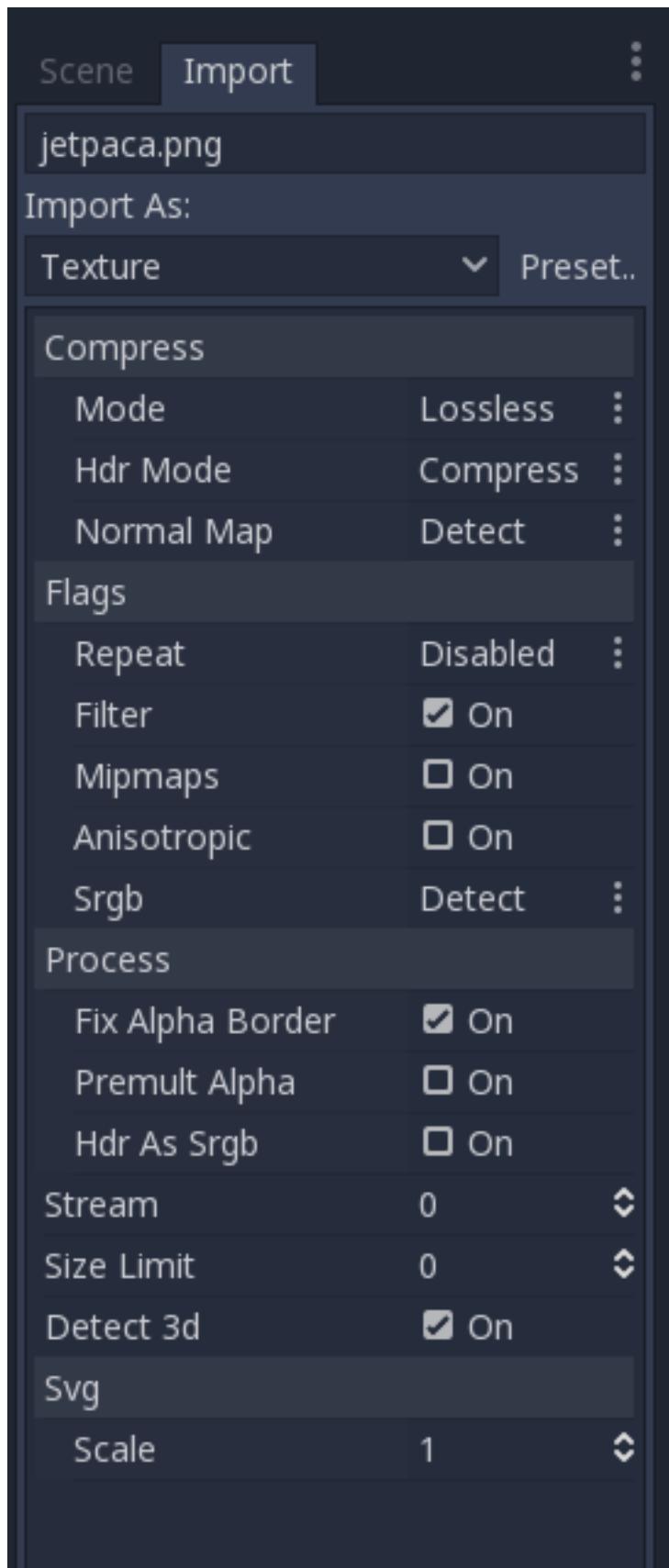
In Godot 3+, image files are no longer native resources and they must be imported. The reason behind this is the large amount of configuration parameters that image files can be imported with.

This small tutorial will explain what these parameters are and how to best make use of them.

Importing Textures

The default action in Godot is to import images as textures. Textures are stored in video memory and can't be accessed directly. This is what makes drawing them efficient.

Import options are vast:



Compression:

Images are one of the largest assets in a game. To handle them efficiently, they need to be compressed. Godot offers several compression methods, depending on the use case.

Compress Mode

- **VRAM Compression:** This is the most common compression mode for 3D assets. File on disk is reduced and video memory usage is also reduced considerably. For 3D, it may present unwanted artifacts, though.
- **Lossless Compression:** This is the most common compression for 2D assets. It shows assets without any kind of artifacting, and disk compression is decent. It will use considerably more amount of video memory than VRAM, though.
- **Lossy Compression:** For games with lots of large 2D assets, lossy compression can be a great choice. It has some artifacting, but less than VRAM and the file size is almost a tenth of Lossless.
- **Uncompressed:** Only useful for formats that can't be compressed (like, raw float).

In this table, each of the four options are described together with their advantages and disadvantages ( = Best,  = Worst):

	Uncom-pressed	Compress Loss-less (PNG)	Compress Lossy (WebP)	Compress VRAM
Description	Stored as raw pixels	Stored as PNG	Stored as WebP	Stored as S3TC/BC,PVRTC/ETC, depending on platform
Size on Disk	 Large	 Small	 Very Small	 Small
Memory Usage	 Large	 Large	 Large	 Small
Performance	 Normal	 Normal	 Normal	 Fast
Quality Loss	 None	 None	 Slight	 Moderate
Load Time	 Normal	 Slow	 Slow	 Fast

HDR Mode

Godot supports high dynamic range textures (as .HDR or .EXR). These are mostly useful as high dynamic range equirectangular panorama skys (the internet has plenty of if you look for them), which replace Cubemaps in Godot 2.x. Modern PCs support the BC6H VRAM format, but there are still plenty that do not.

If you want Godot to ensure full compatibility in for kind of textures, enable the “Force RGBE” option.

Normal Map

When using a texture as normal map, only the red and green channels are required. Given regular texture compression algorithms produce artifacts that don't look that nice in normal maps, the RGTC compression format is the best fit

for this data. Forcing this option to “Enabled” will make Godot import the image as RGTC compressed. By default, it’s set to “Detect” which means that if the texture is ever used as a normal map, it will be changed to “Enabled” and reimported automatically.

Flags

There are plenty of settings that can be toggled when importing an image as a texture, depending on the use case.

Repeat

This setting is mostly commonly used in 3D than 2D (thus it’s generally disabled in 2D). It makes UV coordinates going beyond the 0,0 - 1,1 range to “loop”. Repeating can optionally be set to mirrored mode.

Filter

When pixels become larger than the screen pixels, this options enable linear interpolation for them. The result is a smoother (less blocky) texture. This setting can be commonly used in 2D and 3D, but it’s usually disabled when making pixel perfect games.

Mipmaps

When pixels become smaller than the screen, mipmaps kick in. This helps reduce the grainy effect when shrinking the textures. Keep in mind that, in older hardware (GLES2, mainly mobile), there are some requirements to use mipmaps:

- Texture width and height must be powers of 2
- Repeat must be enabled

Keep in mind the above when making phone games and applications, want to aim for full compatibility, and need mipmaps.

When doing 3D, mipmap should be turned on as this also improves performance (smaller versions of the texture are used for objects further away).

Anisotropic

When textures are near parallel to the view (like floors), this option makes them have more detail by reducing blurriness.

SRGB

Godot uses Linear colorspace when rendering 3D. Textures mapped to albedo or detail channels need to have this option turned on in order for colors to look correct. When set to “Detect” mode, the texture will be marked as SRGB when used in albedo channels.

Process

Some special processes can be applied to images when imported as texture.

Fix Alpha Border

This puts pixels of the same surrounding color in transition from transparency to non transparency. It helps mitigate the outline effect when exporting images from Photoshop and the likes.



It's a good idea to leave it on by default, unless specific values are needed.

Premultiplied Alpha

An alternative to fix darkened borders is to use premultiplied alpha. By enabling this option, the texture will be converted to this format. Keep in mind that a material will need to be created that uses the PREMULT ALPHA blend mode on canvas items that need it.

HDR as SRGB

Some few HDR files are broken and contain SRGB color data. It is advised to not use them but, in the worst case, toggling this option on will make them look right.

Detect 3D

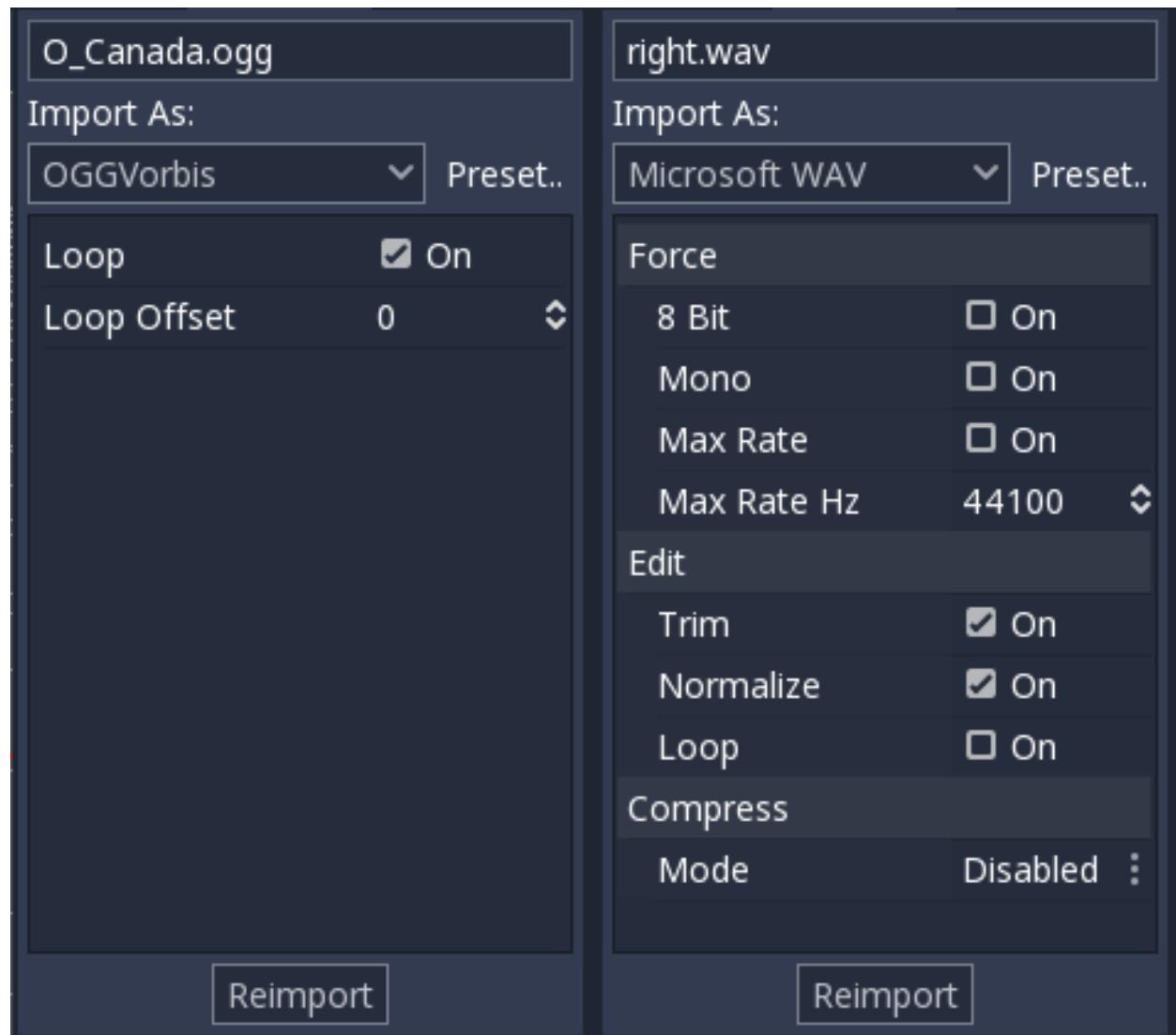
This option makes Godot be aware of when a texture (which is imported for 2D as default) is used in 3D. If this happens, setting are changed so the texture flags are friendlier to 3D (mipmaps, filter and repeat become enabled and compression is changed to VRAM). Texture is also reimported automatically.

5.2.3 Importing audio samples

Why importing?

Raw audio data in general is large and undesired. Godot provides two main options to import your audio data: WAV and OGG Vorbis.

Each has different advantages. * Wav files use raw data or light compression, require small amount of CPU to play back (hundreds of simultaneous voices in this format are fine), but take up significant space. * Ogg Vorbis files use a stronger compression that results in much smaller file size, but uses significantly more processor to play back.



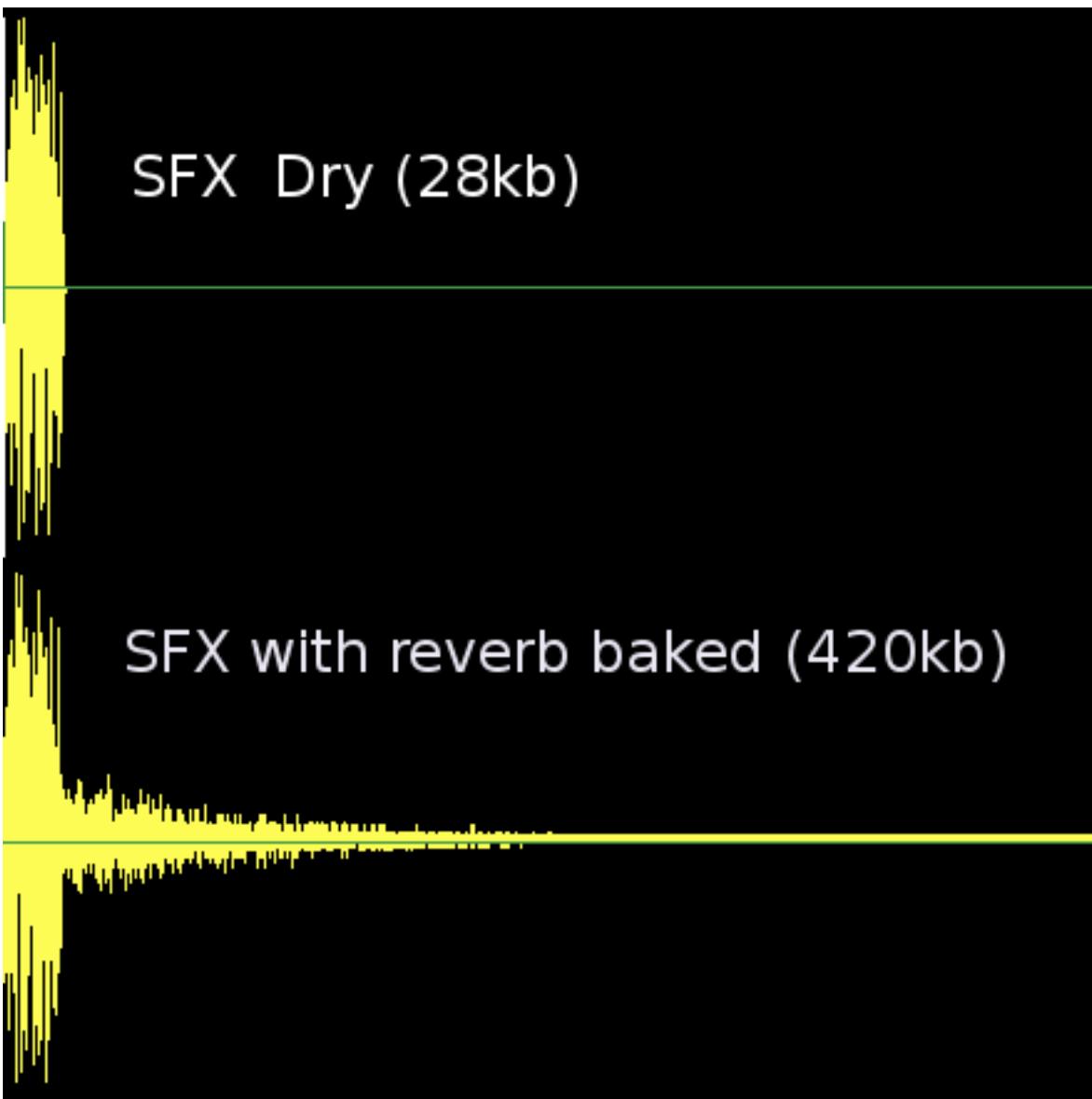
Here is a comparative chart.

Format	1 Second of Audio
WAV 24 bits, 96 kHz, Stereo	576kb
WAV 16 bits, 44 kHz, Mono	88kb
WAV 16 bits, IMA-ADPCM, Mono	22kb
OGG 128kbps, Stereo	16kb
OGG Vorbis 96kbps, Stereo	12kb

In general, what is recommended, is to use WAV for most sound effects, especially those that are short and repetitive, and OGG for music, voice and long sound effects.

Best Practices

Godot 3+ has an amazing bus system with built in effects. This saves SFX artists the need to add reverb to the sound effects, reducing their size greatly and ensuring correct trimming. Say no to SFX with baked reverb!



As you can see above, sound effects become huge with reverb added.

Trimming

One issue that happens often is that the waveform are exported with long silences at the beginning and at the end. These are inserted by DAWs when saving to a waveform, increase their size unnecessarily and add latency to the moment they are played back.

Importing as WAV with the Trimming option enabled solves this.

Looping

Godot supports looping in the samples (Tools such as Sound Forge or Audition can add loop points to wav files). This is useful for sound effects such as engines, machine guns, etc. Ping-pong looping is also supported.

As an alternative, the import screen has a “loop” option that enables looping for the entire sample when importing.

5.2.4 Importing translations

Games and internationalization

The world is full of different markets and cultures and, to maximize profits™, nowadays games are released in several languages. To solve this, internationalized text must be supported in any modern game engine.

In regular desktop or mobile applications, internationalized text is usually located in resource files (or .po files for GNU stuff). Games, however, can use several orders of magnitude more text than applications, so they must support efficient methods for dealing with loads of multilingual text.

There are two approaches to generate multilingual language games and applications. Both are based on a key:value system. The first is to use one of the languages as the key (usually English), the second is to use a specific identifier. The first approach is probably easier for development if a game is released first in English, later in other languages, but a complete nightmare if working with many languages at the same time.

In general, games use the second approach and a unique ID is used for each string. This allows you to revise the text while it is being translated to other languages. The unique ID can be a number, a string, or a string with a number (it's just a unique string anyway).

Translators also usually prefer to work with spreadsheets.

Translation format

To complete the picture and allow efficient support for translations, Godot has a special importer that can read CSV files. All spreadsheet editors (be it Libreoffice, Microsoft Office, Google Docs, etc.) can export to this format, so the only requirement is that the files have a special arrangement. The CSV files must be saved in UTF-8 encoding and be formatted as follows:

	<lang1>	<lang2>	<langN>
KEY1	string	string	string
KEY2	string	string	string
KEYN	string	string	string

The “lang” tags must represent a language, which must be one of the *valid locales* supported by the engine. The “KEY” tags must be unique and represent a string universally (they are usually in uppercase, to differentiate from other strings). Here’s an example:

id	en	es	ja
GREET	Hello, friend!	Hola, Amigo!	
ASK	How are you?	Cómo está?	
BYE	Good Bye	Adiós	

CSV Importer

Godot will treat CSV files as translations by default. It will import them and generate one or more compressed translation resource files next to it.

Importing will also add the translation to the list of translations to load when the game runs, specified in project.godot (or the project settings). Godot allows loading and removing translations at runtime as well.

5.2.5 Importing 3D Scenes

Godot Scene Importer

When dealing with 3D assets, Godot has a very flexible and configurable importer.

Godot works with *scenes*. This means that the entire scene being worked on in your favorite 3D DCC will be transferred as close as possible.

Godot supports the following 3D *scene file formats*:

- DAE (Collada), which is currently the most mature workflow.
- GLTF 2.0. Both text and binary formats are supported. Godot has full support for it, but the format is new and gaining traction.
- OBJ (Wavefront) formats. It is also fully supported, but pretty limited (no support for pivots, skeletons, etc).

Just copy the scene file together with the texture to the project repository, and Godot will do a full import.

Why not FBX?

Most game engines use the FBX format for importing 3D scenes, which is definitely one of the most standardized in the industry. However, this format requires the use of a closed library from Autodesk which is distributed with a more restrictive licensing terms than Godot.

The plan is, sometime in the future, to offer a binary plug-in using GDNative.

Exporting DAE files from Maya and 3DS Max

Autodesk added built-in collada support to Maya and 3DS Max, but it's broken by default and should not be used. The best way to export this format is by using the [OpenCollada](#) plugins. They work really well, although they are not always up-to date with the latest version of the software.

Exporting DAE files from Blender

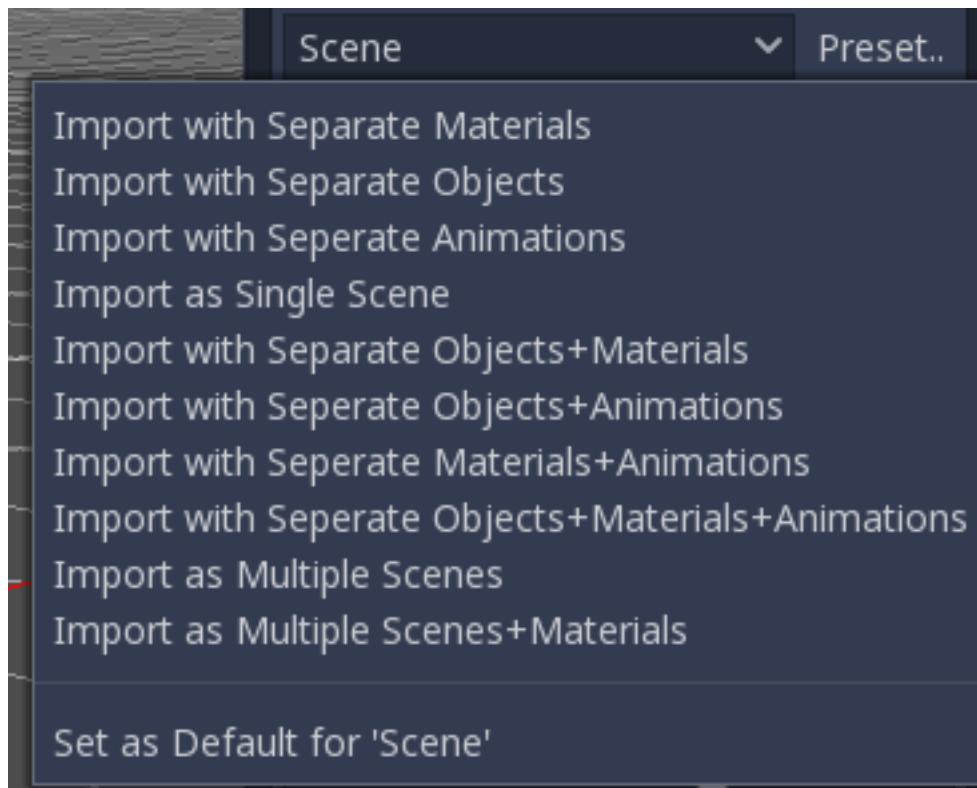
Blender has built-in collada support too, but it's also broken and should not be used.

Godot provides a [Python Plugin](#) that will do a much better job of exporting the scenes.

Import workflows

Godot scene importer allows different workflows regarding how data is imported. Depending on many options, it is possible to import a scene with:

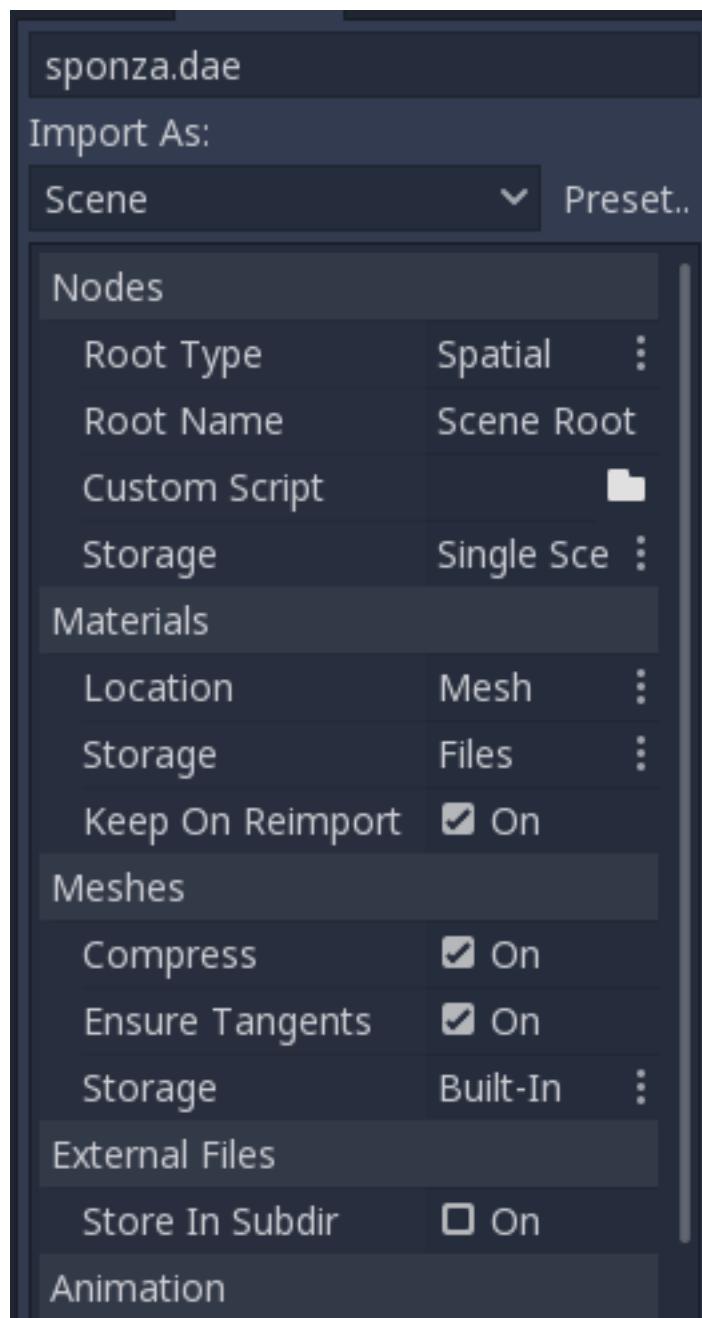
- External materials (default): Where each material is saved to a file resource. Modifications to them are kept.
- External meshes: Where each mesh is saved to a different file. Many users prefer to deal with meshes directly.
- External animations: Allowing saved animations to be modified and merged when sources change.
- External scenes: Save the root nodes of the imported scenes each as a separate scene.
- Single Scene: A single scene file with everything built in.



As different developers have different needs, this import process is highly customizable.

Import Options

The importer has several options, which will be discussed below:



Nodes:

Root Type

By default, the type of the root node in imported scenes is “Spatial”, but this can be modified.

Root Name

Allows setting a specific name to the generated root node.

Custom Script

A special script to process the whole scene after import can be provided. This is great for post processing, changing materials, doing funny stuff with the geometry etc.

Create a script that basically looks like this:

```
tool # needed so it runs in editor
extends EditorScenePostImport

func post_import(scene):
    # do your stuff here
    return scene # remember to return the imported scene
```

The post-import function takes the imported scene as argument (the parameter is actually the root node of the scene). The scene that will finally be used must be returned. It can be a different one.

Storage

By default, Godot imports a single scene. This option allows specifying that nodes below the root will each be a separate scene and instanced into the imported one.

Of course, instancing such imported scenes in other places manually works too.

Materials

Location

Godot supports materials in meshes or nodes. By default, materials will be put on each node.

Storage

Materials can be stored within the scene or in external files. By default, they are stored in external files so editing them is possible. This is because most 3D DCCs don't have the same material options as those present in Godot.

When materials are built-in, they will be lost each time the source scene is modified and re-imported.

Keep on Reimport

Once materials are edited to use Godot features, the importer will keep the edited ones and ignore the ones coming from the source scene. This option is only present if materials are saved as files.

Compress

Makes meshes use less precise numbers for multiple aspects of the mesh in order to save space.

These are:

- Transform Matrix (Location, rotation, and scale) : 32-bit float to 16-bit signed integer.
- Vertices : 32-bit float to 16-bit signed integer.
- Normals : 32-bit float to 32-bit unsigned integer.

- Tangents : 32-bit float to 32-bit unsigned integer.
- Vertex Colors : 32-bit float to 32-bit unsigned integer.
- UV : 32-bit float to 32-bit unsigned integer.
- UV2 : 32-bit float to 32-bit unsigned integer.
- Vertex weights : 32-bit float to 16-bit unsigned integer.
- Armature bones : 32-bit float to 16-bit unsigned integer.
- Array index : 32-bit or 16-bit unsigned integer based on how many elements there are.

Additional info:

- UV2 = The second UV channel for detail textures and baked lightmap textures.
- Array index = An array of numbers that number each element of the arrays above; i.e. they number the vertexes and normals.

In some cases, this might lead to loss of precision so disabling this option may be needed. For instance, if a mesh is very big or there are multiple meshes being imported that cover a large area, compressing the import of this mesh(s) may lead to gaps in geometry or vertices not being exactly where they should be.

Meshes

Ensure Tangents

If textures with normalmapping are to be used, meshes need to have tangent arrays. This option ensures that these are generated if not present in the source scene. Godot uses Mikktspace for this, but it's always better to have them generated in the exporter.

Storage

Meshes can be stored in separate files (resources) instead of built-in. This does not have much practical use unless one wants to build objects with them directly.

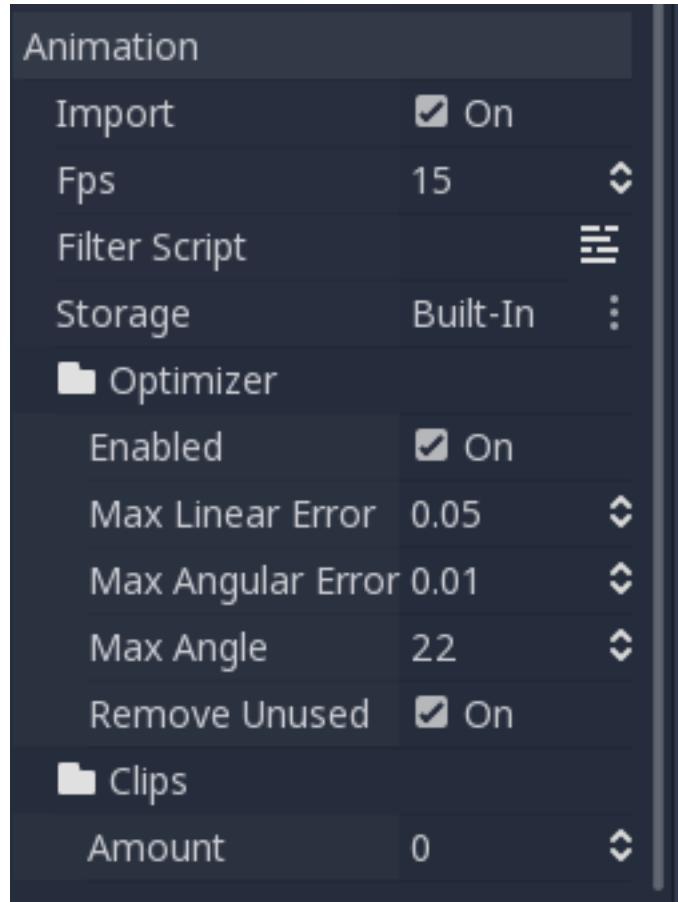
This option is provided to help those who prefer working directly with meshes instead of scenes.

External Files

Generated meshes and materials can be optionally stored in a subdirectory with the name of the scene.

Animation Options

Godot provides many options regarding how animation data is dealt with. Some exporters (such as Blender), can generate many animations in a single file. Others, such as 3DS Max or Maya, need many animations put into the same timeline or, at worst, put each animation in a separate file.



Import of animations is enabled by default.

Animation : FPS

Most 3D export formats store animation timeline in seconds instead of frames. To ensure animations are imported as faithfully as possible, please specify the frames per second used to edit them. Failing to do this may result in minimal jitter.

Animation : Filter Script

It is possible to specify a filter script in a special syntax to decide which tracks from which animations should be kept. (@TODO this needs documentation)

Animation : Storage

By default, animations are saved as built-in. It is possible to save them to a file instead. This allows adding custom tracks to the animations and keeping them after a reimport.

Animation : Optimizer

When animations are imported, an optimizer is run which reduces the size of the animation considerably. In general, this should always be turned on unless you suspect that an animation might be broken due to it being enabled.

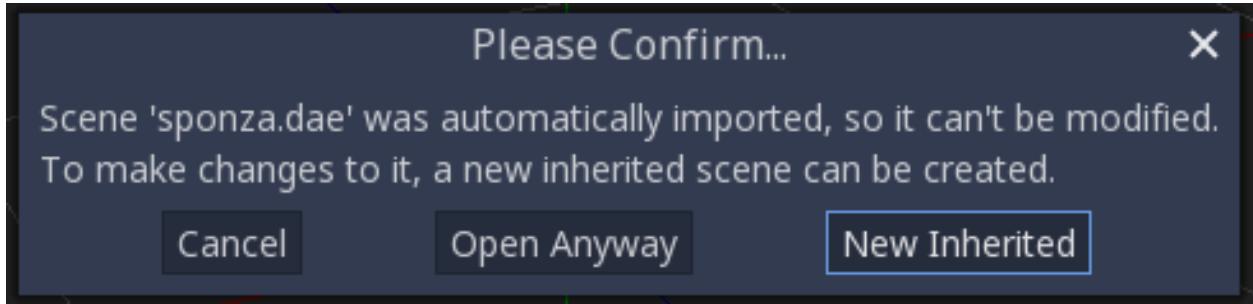
Animation : Clips

It is possible to specify multiple animations from a single timeline as clips. Just specify from which frame to which frame each clip must be taken (and, of course, don't forget to specify the FPS option above).

Scene Inheritance

In many cases, it may be desired to do modifications to the imported scene. By default, this is not really possible because if the source asset changes (source .dae,.gltf,.obj file re-exported from 3D modelling app), Godot will re-import the whole scene.

It is possible, however, to do local modifications by using *Scene Inheritance*. Just try to open the imported scene and the following dialog will appear:



In inherited scenes, the only limitations for modifications are:

- * Nodes can't be removed (but can be added anywhere).
- * Sub-Resources can't be edited (save them externally as described above for this)

Other than that, everything is allowed!

Import Hints

Many times, when editing a scene, there are common tasks that need to be done after exporting:

- * Adding collision detection to objects:
- * Setting objects as navigation meshes
- * Deleting nodes that are not used in the game engine (like specific lights used for modelling)

To simplify this workflow, Godot offers a few suffixes that can be added to the names of the objects in your 3D modelling software. When imported, Godot will detect them and perform actions automatically:

Remove nodes (-noimp)

Node names that have this suffix will be removed at import time, no matter what their type is. They will not appear in the imported scene.

Create collisions (-col, -colonly, -convcolonly)

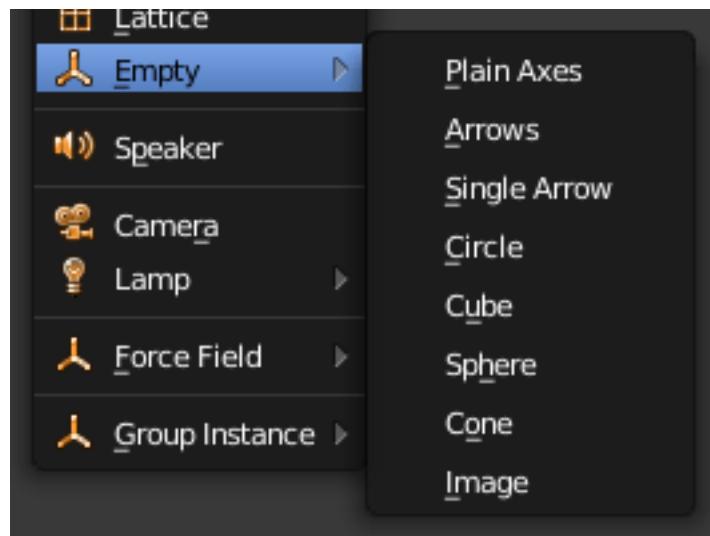
Option “-col” will work only for Mesh nodes. If it is detected, a child static collision node will be added, using the same geometry as the mesh.

However, it is often the case that the visual geometry is too complex or too un-smooth for collisions, which ends up not working well.

To solve this, the “-colonly” modifier exists, which will remove the mesh upon import and create a *StaticBody* collision instead. This helps the visual mesh and actual collision to be separated.

Option “-convcolonly” will create *ConvexPolygonShape* instead of *ConcavePolygonShape*.

Option “-colonly” can be also used with Blender’s empty objects. On import it will create a *StaticBody* with collision node as a child. Collision node will have one of predefined shapes, depending on the Blender’s empty draw type:



- Single arrow will create *RayShape*
- Cube will create *BoxShape*
- Image will create *PlaneShape*
- Sphere (and other non-listed) will create *SphereShape*

For better visibility in Blender’s editor user can set “X-Ray” option on collision empties and set some distinct color for them in User Preferences / Themes / 3D View / Empty.

Create navatomp (-navmesh)

A mesh node with this suffix will be converted to a navigation mesh. Original Mesh node will be removed.

Rigid Body (-rigid)

Creates a rigid body from this mesh

5.3 Export

5.3.1 Exporting projects

Why exporting?

Originally, Godot did not have any means to export projects. The developers would compile the proper binaries and build the packages for each platform manually.

When more developers (and even non-programmers) started using it, and when our company started taking more projects at the same time, it became evident that this was a bottleneck.

On PC

Distributing a game project on PC with Godot is rather easy. Just drop the godot.exe (or godot) binary together in the same place as the engine.cfg file, zip it and you are done. This can be taken advantage of to make custom installers.

It sounds simple, but there are probably a few reasons why the developer may not want to do this. The first one is that it may not be desirable to distribute loads of files. Some developers may not like curious users peeking at how the game was made, others may just find it inelegant, etc.

Another reason is that, for distribution, the developer might prefer a specially compiled binary, which is smaller in size, more optimized and does not include tools inside (like the editor, debugger, etc.).

Finally, Godot has a simple but efficient system for creating DLCs as extra package files.

On mobile

The same scenario in mobile is a little worse. To distribute a project in those devices, a binary for each of those platforms is built, then added to a native project together with the game data.

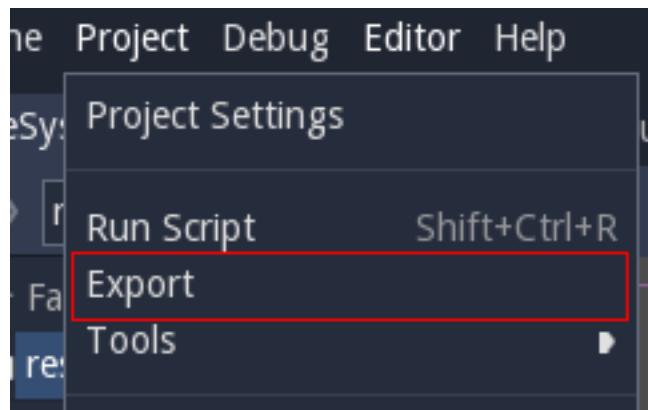
This can be troublesome because it means that the developer must be familiarized with the SDK of each platform before even being able to export. While learning each SDK is always encouraged, it can be frustrating to be forced to do it at an undesired time.

There is also another problem with this approach. Different devices prefer some data in different formats to run. The main example of this is texture compression. All PC hardware uses S3TC (BC) compression and that has been standardized for more than a decade, but mobile devices use different formats for texture compression, such as PVRCT (iOS) or ETC (Android).

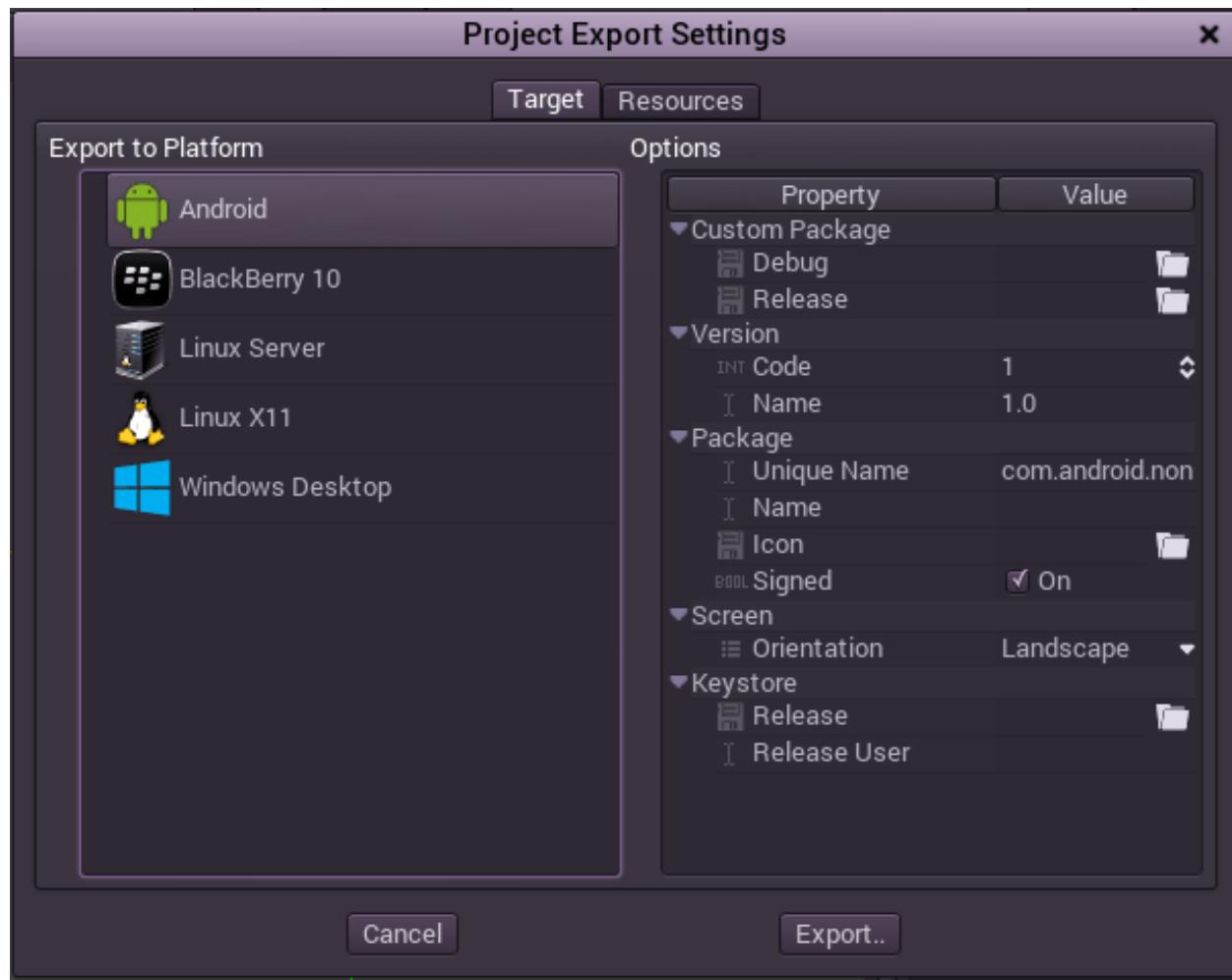
Export dialog

After many attempts at different export workflows, the current one has proven to work the best. At the time of this writing, not all platforms are supported yet, but the supported platforms continue to grow.

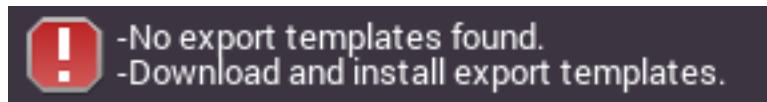
To open the export dialog, just click the “Export” button:



The dialog will open, showing all the supported export platforms:



The default options are often enough to export, so tweaking them is not necessary, but provide extra control. However, many platforms require additional tools (SDKs) to be installed to be able to export. Additionally, Godot needs export templates installed to create packages. The export dialog will complain when something is missing and will not allow the user to export for that platform until they resolve it:

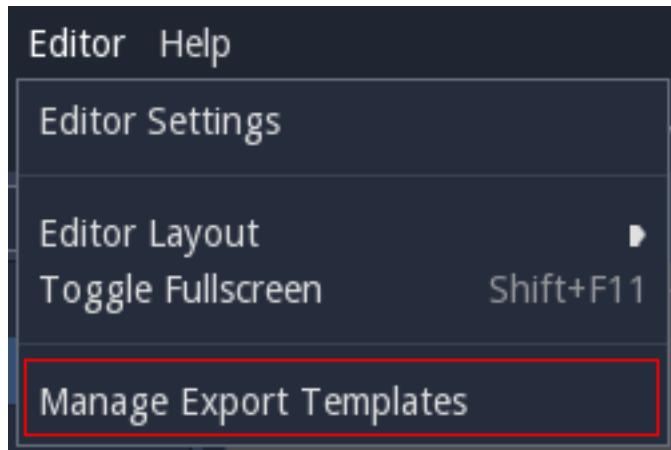


At that time, the user is expected to come back to the documentation and follow instructions on how to properly set up that platform.

Export templates

Apart from setting up the platform, the export templates must be installed to be able to export projects. They can be obtained as a .tpz (a renamed .zip) file from the [download page of the website](#).

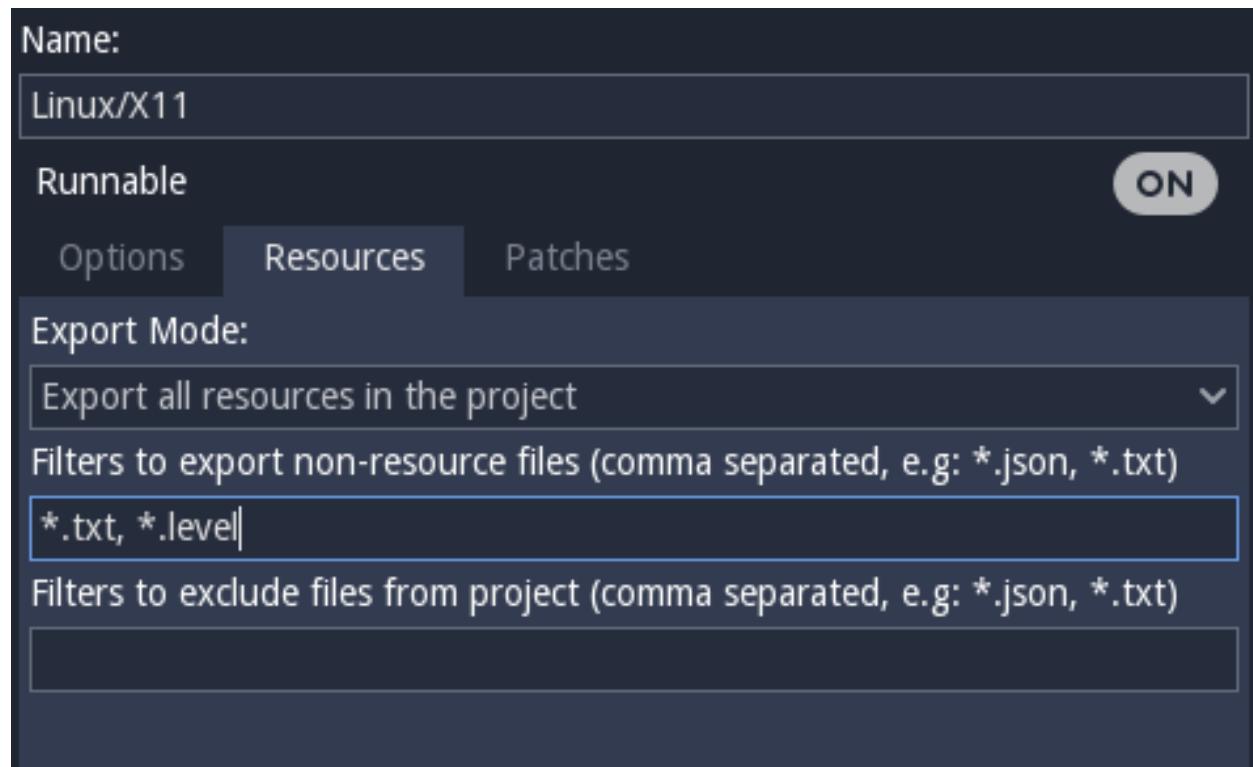
Once downloaded, they can be installed using the “Install Export Templates” option in the editor:



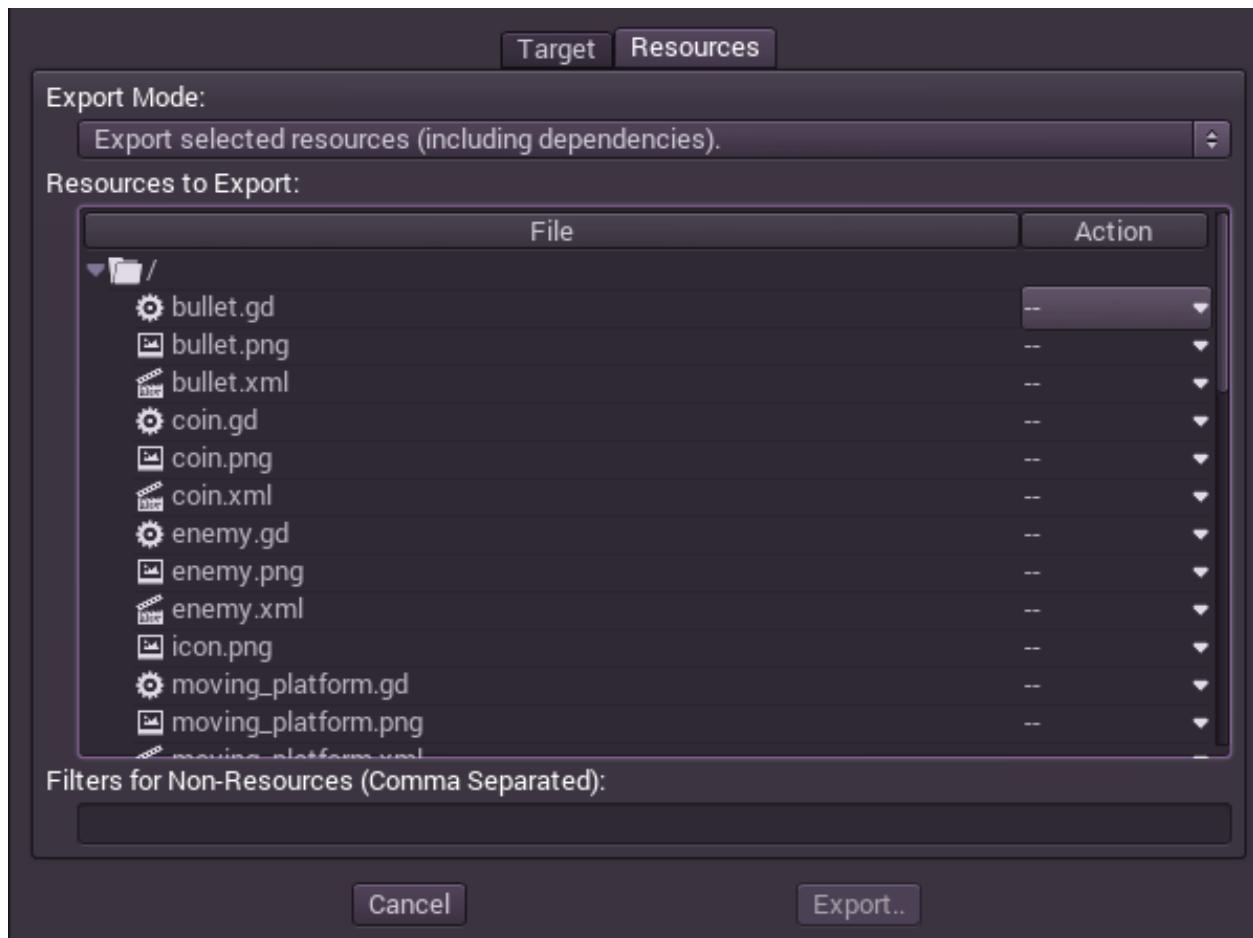
Export mode

When exporting, Godot makes a list of all the files to export and then creates the package. There are 3 different modes for exporting:

- Export every single file in the project
- Export only resources (+custom filter), this is default.
- Export only selected resources (+custom filter)



- **Export every single file** - This mode exports every single file in the project. This is good to test if something is being forgotten, but developers often have a lot of unrelated stuff around in the dev directory, which makes it a bad idea.
- **Export only resources** - Only resources are exported. For most projects, this is enough. However many developers like to use custom datafiles in their games. To compensate for this, filters can be added for extra extensions (like, .txt,.csv, etc.).
- **Export only selected resources** - Only select resources from a list are exported. This is probably overkill for most projects, but in some cases it is justified (usually huge projects). This mode offers total control of what is exported. Individual resources can be selected and dependency detection is performed to ensure that everything needed is added. As a plus, this mode allows to “Bundle” scenes and dependencies into a single file, which is *really* useful for games distributed on optical media.



5.3.2 Feature Tags

Introduction

Godot has a special system to tag availability of features. Each *feature* is represented as a string, and it can refer to many of the following:

- Platform name.
- Platform bits (64/32).
- Platform type (desktop/mobile).
- Supported texture compression in platform.
- Whether a build is debug or release.
- Many more things.

Features can be queried in run-time to the singleton API by calling:

```
OS.has_feature(name)
```

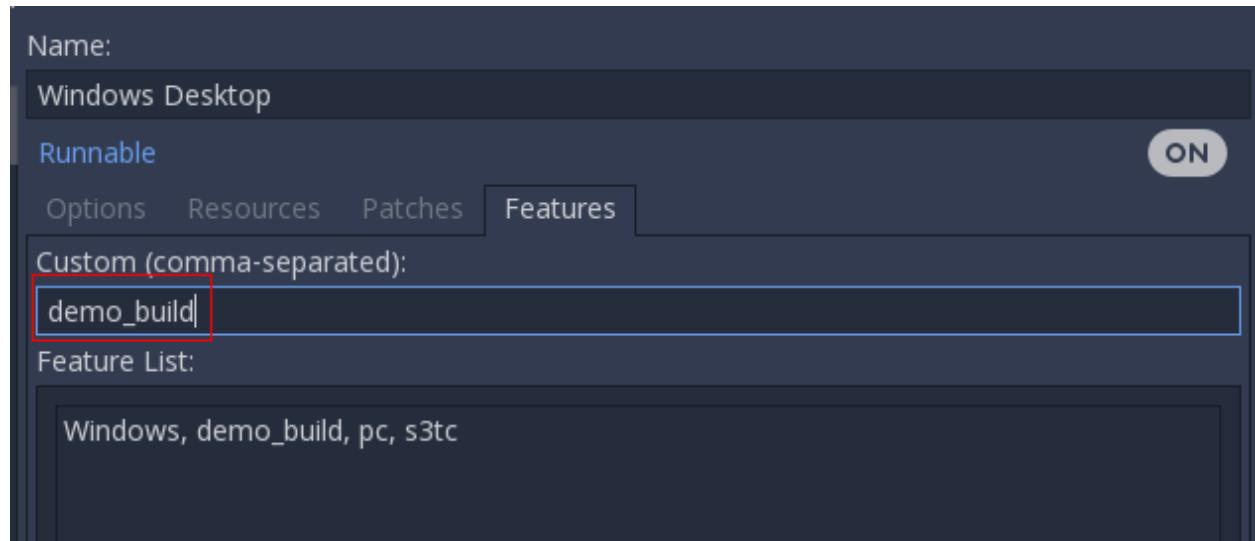
Default features

Here is a list of most feature tags in Godot. Keep in mind they are *case sensitive*:

Feature Tag	Description
Android	Running on Android
JavaScript	Running on JavaScript (HTML5)
OSX	Running on macOS
iOS	Running on iOS
UWP	Running on UWP
Windows	Running on Windows
X11	Running on X11
debug	Running on debug build
release	Running on release build
32	Running on 32-bit build
64	Running on 64-bit build
mobile	Host OS is a mobile platform
pc	Host OS is a PC platform (desktop/laptop)
web	Host OS is a Web browser
etc	Textures using ETC1 compression are supported
etc2	Textures using ETC2 compression are supported
s3tc	Textures using S3TC (DXT/BC) compression are supported
pvrbc	Textures using PVRTC compression are supported

Custom features

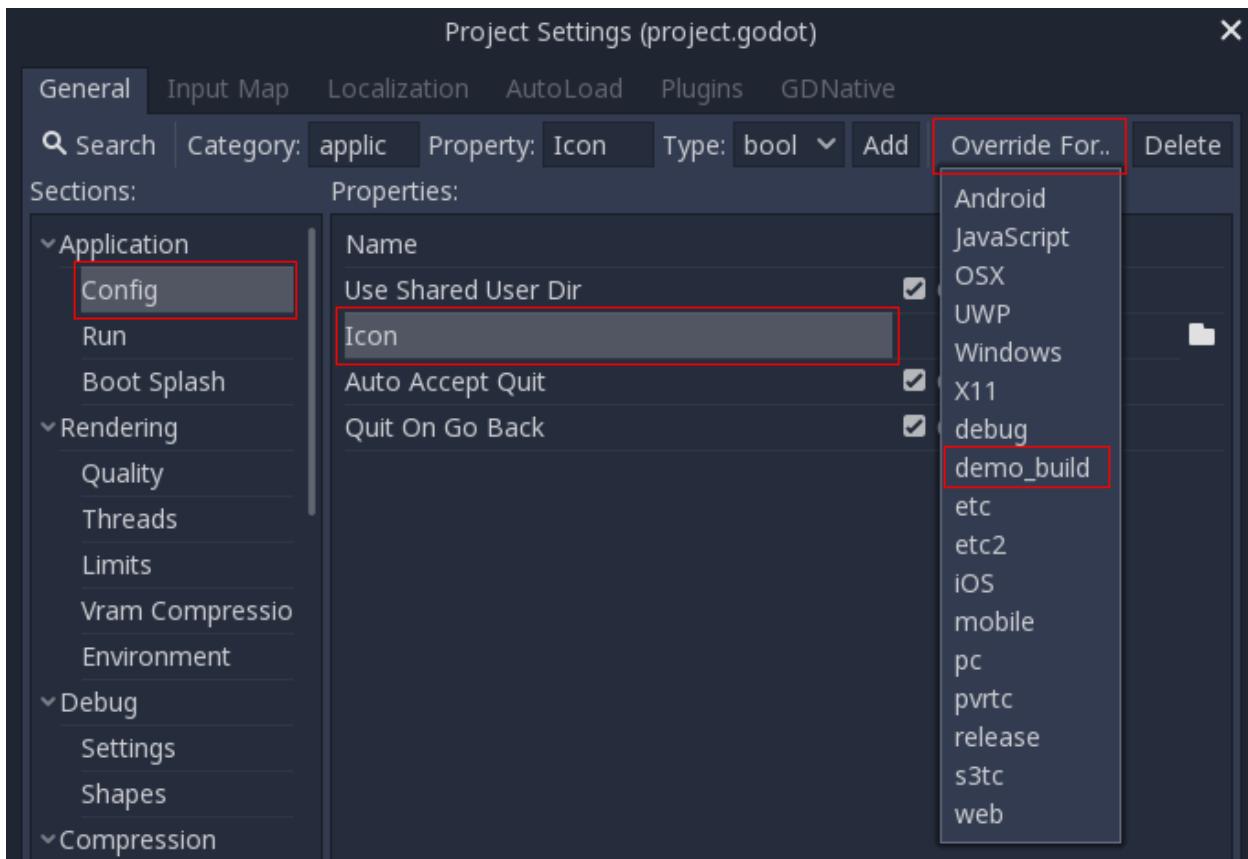
It is possible to add custom features to a build, just use the relevant field in the *export preset* used to generate it:



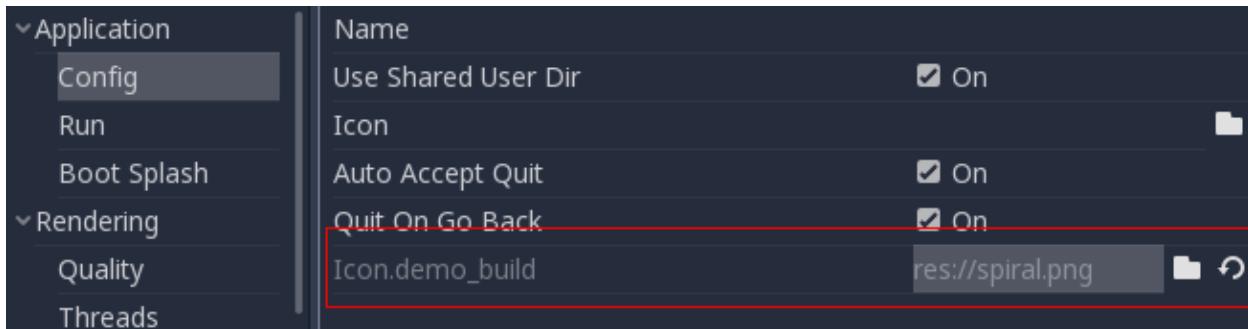
Overriding project settings

Features can be used to override specific configuration values in the *Project Settings*. This allows to better customize any configuration when doing a build.

In the following example, a different icon is added for the demo build of the game (which was customized in a special export preset which, in turn, includes only demo levels).

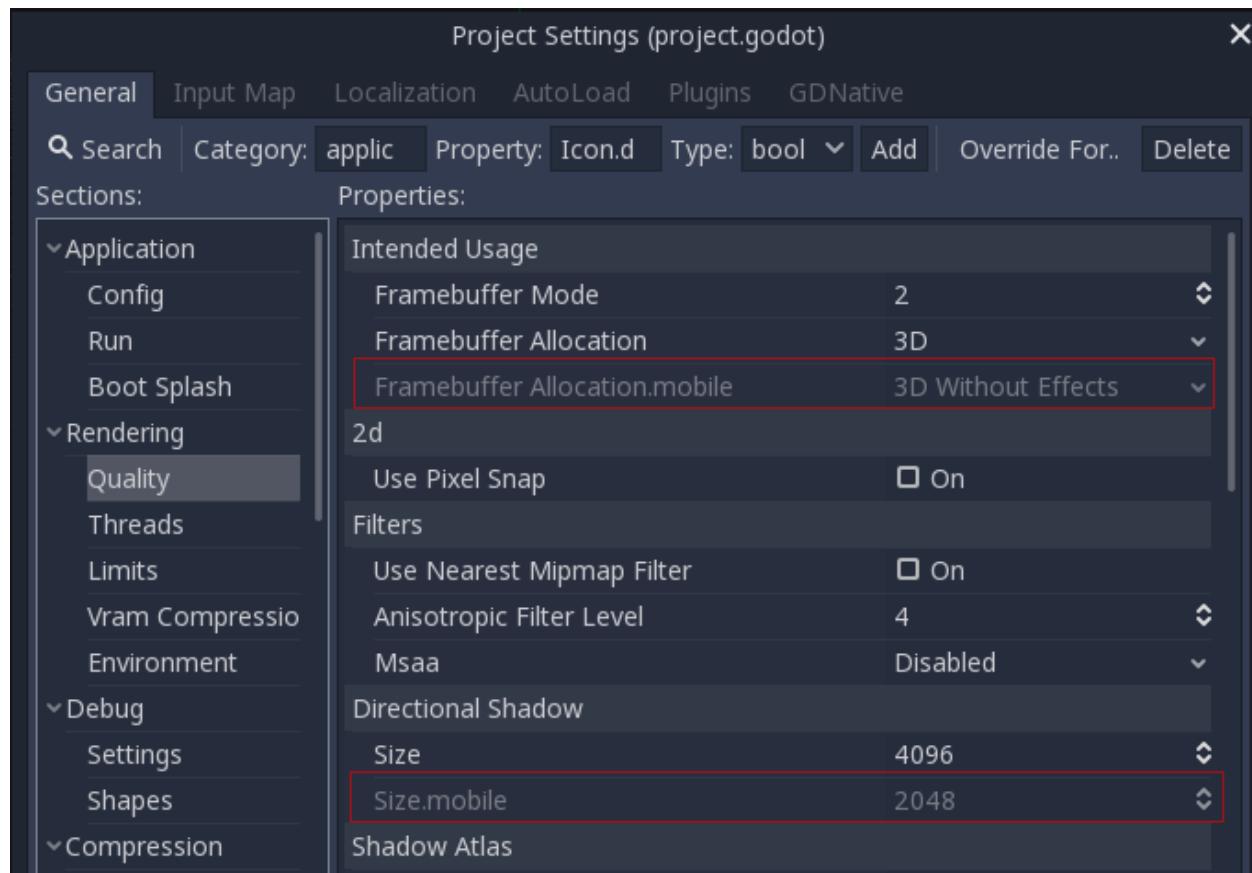


After overriding, a new field is added for this specific configuration:



Default overrides

There are already a lot of settings that come with overrides by default, they can be found in many sections of the project settings.



Customizing Build

Feature tags can be used to customize a build process too, by writing a custom **ExportPlugin**. They also are used to specify which shared library is loaded and exported in **GDNative**.

5.3.3 Exporting for PC

The simplest way to distribute a game for PC is to copy the executables (godot.exe on windows, godot on the rest), zip the folder and send it to someone else. However, this is often not desired.

Godot offers a more elegant approach for PC distribution when using the export system. When exporting for PC (Linux, Windows, Mac), the exporter takes all the project files and creates a “data.pck” file. This file is bundled with a specially optimized binary that is smaller, faster and lacks tools and debugger.

Optionally, the files can be bundled inside the executable, though this does not always work properly.

5.3.4 Exporting for iOS

Exporting for iOS is done manually at the moment. These are the steps to load your game in an XCode project, where you can deploy to a device, publish, etc.

Requirements

- Download XCode for iOS

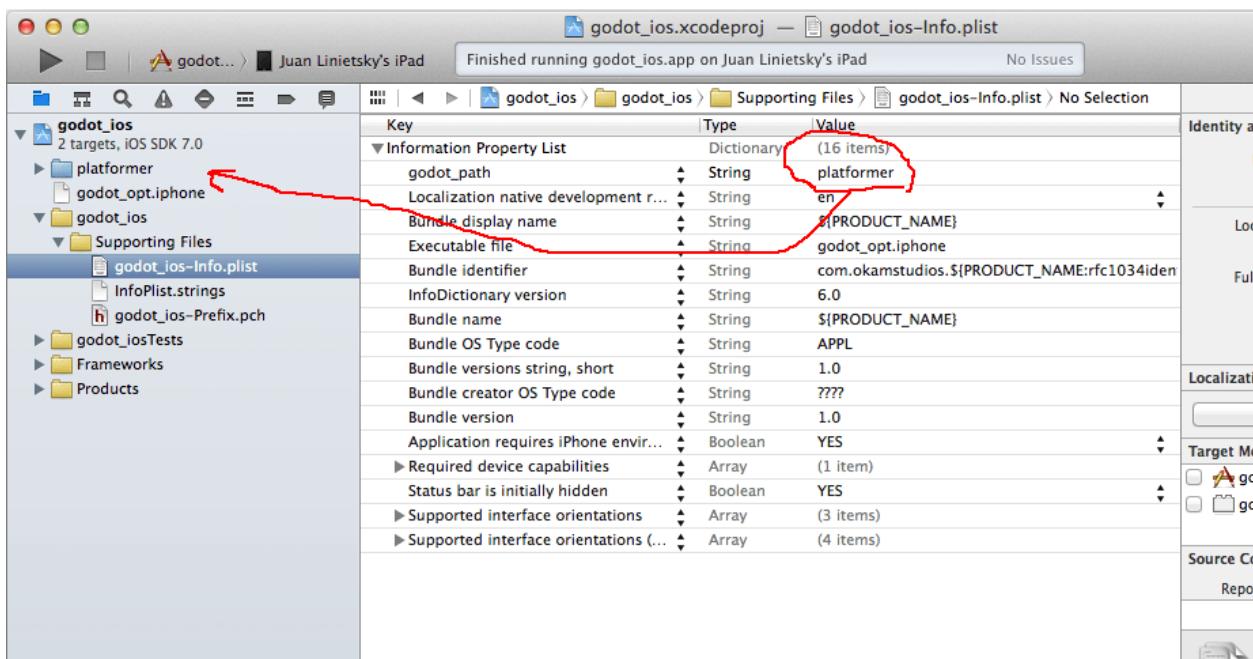
- Download the export templates: <https://godotengine.org/download>
- Since there is no automatic deployer yet, unzip export_templates.tpz manually and extract GodotiOSCode.zip from it.

The zip contains an XCode project, godot_ios.xcodeproj, an empty data.pck file and the engine executable. Open the project, and modify the game name, icon, organization, provisioning signing certificate identities (??), etc.

Add your project data

Using the Godot editor, *Exporting for PC*, to obtain the data.pck file. Replace the empty data.pck in the XCode project with the new one, and run/archive.

If you want to test your scenes on the iOS device as you edit them, you can add your game directory to the project (instead of data.pck), and add a property “godot_path” to Info.plist, with the name of your directory as its value.



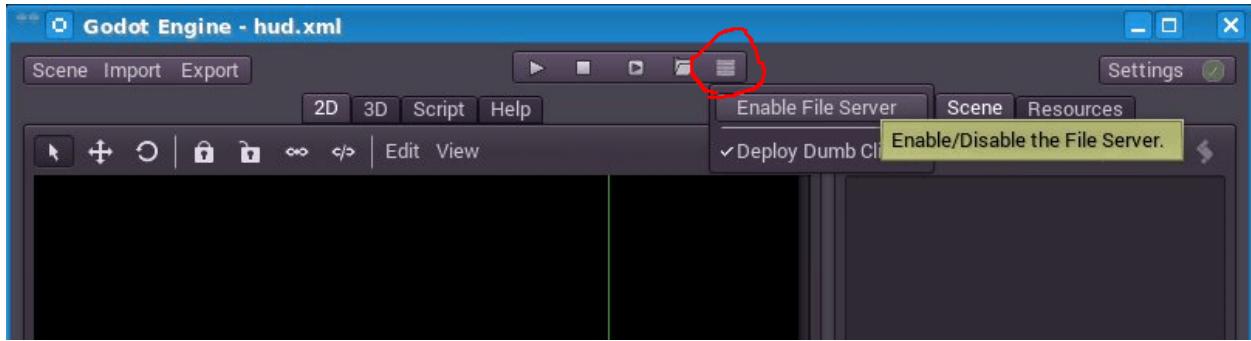
Alternatively you can add all the files from your game directly, with “engine.cfg” at the root.

Loading files from a host

Sometimes your game becomes too big and deploying to the device takes too long every time you run. In that case you can deploy only the engine executable, and serve the game files from your computer.

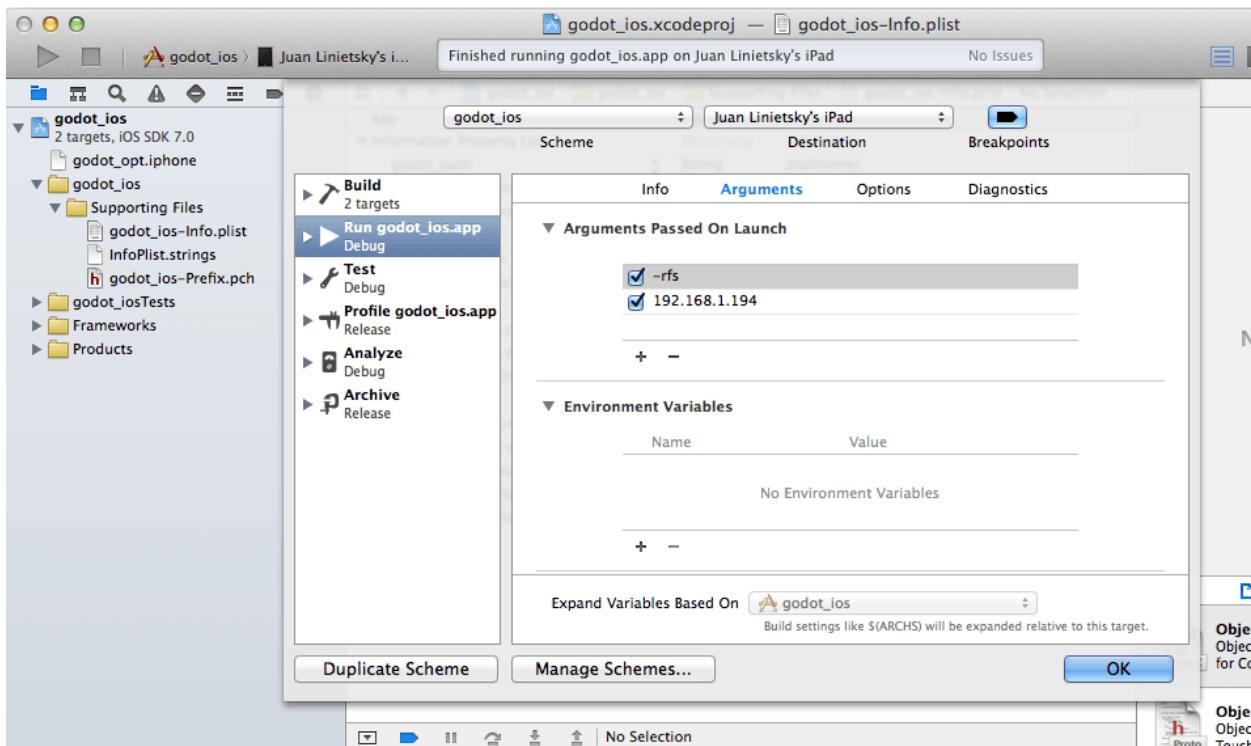
Setting up the file host

On your PC, open the editor, and click the right-most icon on the top-center group of icons, and select “Enable File Server”. The icon turns red. Your PC will open a port and accept connections to serve files from your project’s directory (so enable your local firewall accordingly).



Setting up the game

On XCode, click on your app name (top left, next to the “Stop” button), and select “Edit Scheme”. Go to the “Arguments” tab, and add 2 arguments, “-rfs” and the IP of your PC.



When you run, your device will connect to the host and open the files remotely. Note that the directory with the game data (“platformer”) is no longer added to the project, only the engine executable.

Services for iOS

Special iOS services can be used in Godot. Check out the *Services for iOS* page.

5.3.5 Exporting for Universal Windows Platform

There’s no extra requirement to export an .appx package that can be installed as a Windows App or submitted to the Windows Store. Exporting packages also works from any platform, not only on Windows.

However, if you want to install and run the app, you need to sign it with a trusted signature. Currently, Godot supports no signing of packages and you need to use externals to tools to do so.

Also, make sure the Publisher name you set when export the package matches the name on the certificate.

Limitations on Xbox One

As described in [UWP documentation](#):

- available RAM is 1GB (after exceeding it, application will encounter memory allocation failures and will crash)
- share of 2-4 CPU cores
- share of 45% of GPU power

Creating a signing certificate

This requires the tools `MakeCert.exe` and `Pvk2Pfx.exe` which comes with the Windows SDK. If you use Visual Studio, open one of its Developer Prompts since they come with those tools available and in the path.

You can get more detailed instructions from [Microsoft documentation](#).

First, run `MakeCert` to create a private key:

```
MakeCert /n publisherName /r 0 /eku "1.3.6.1.5.5.7.3.3,1.3.6.1.4.1.311.10.3.13" /e
expirationDate /sv MyKey.pvk MyKey.cer
```

Where `publisherName` matches the Publisher Name of your package and `expirationDate` is in the mm/dd/yyyy format.

Next, create a Personal Information Exchange (.pfx) file using `Pvk2Pfx.exe`:

```
Pvk2Pfx /pvk MyKey.pvk /pi pvkPassword /spc MyKey.cer /pfx MyKey.pfx [/po pfxPassword]
```

If you don't specify a password with `/po` argument, the PFX will have the same password as the private key.

You also need to trust this certificate to be able to actually install the apps. Open the Command Prompt as Administrator and run the following command:

```
Certutil -addStore TrustedPeople MyKey.cer
```

Signing the package

Using the `SignTool.exe` this requires a single command:

```
SignTool sign /fd SHA256 /a /f MyKey.pfx /p pfxPassword package.appx
```

Installing the package

After Windows 10 Anniversary Update you can install packages by just double clicking the .appx file from the Windows Explorer.

It's also possible to install using the `Add-AppxPackage` PowerShell cmdlet.

Note that if you don't update the version number, you'll have to uninstall the previous installed package before reinstalling it.

5.3.6 Exporting for Android

Exporting for Android has fewer requirements than compiling Godot for it. The following steps detail what is needed to setup the SDK and the engine.

Download the Android SDK

Download and install the Android SDK from <http://developer.android.com/sdk/index.html>

Install OpenJDK or Oracle JDK

Download and install OpenJDK or Oracle JDK. Version 6 and 8 are known to work, some users have reported issues with the jarsigner (used to sign the APKs) in JDK 7.

Create a debug.keystore

Android needs a debug keystore file to install to devices and distribute non-release APKs. If you have used the SDK before and have built projects, ant or eclipse probably generated one for you (on Linux and macOS, you can find it in the `~/ .android` directory).

If you can't find it or need to generate one, the keytool command from the JDK can be used for this purpose:

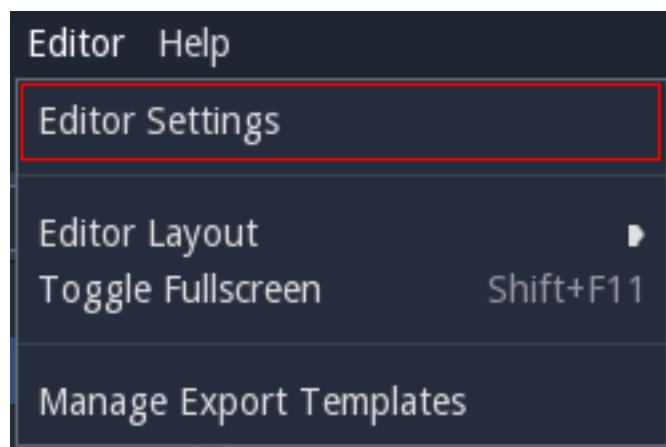
```
keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android -keystore
↳debug.keystore -storepass android -dname "CN=Android Debug,O=Android,C=US" -
↳validity 9999
```

Make sure you have adb

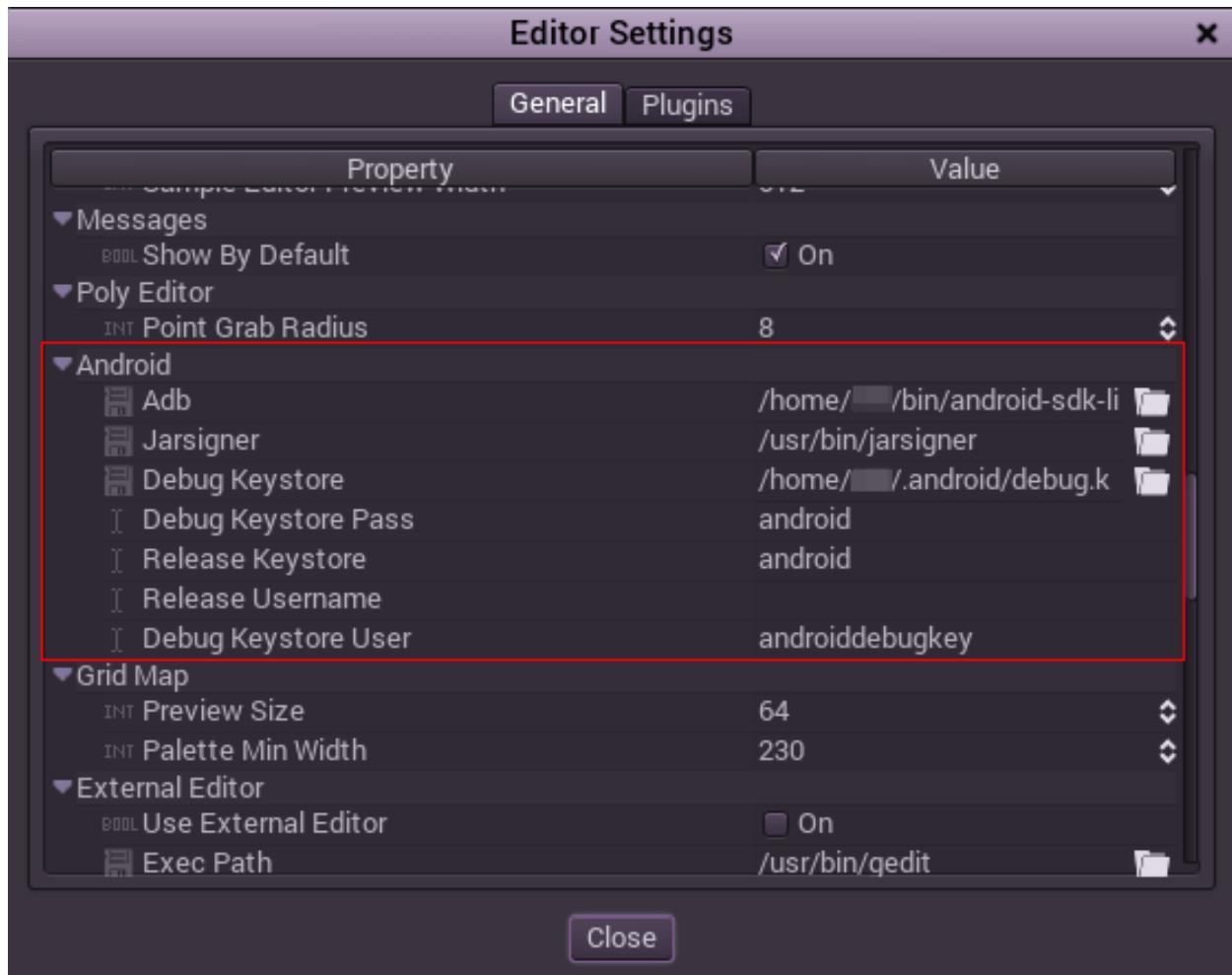
Android Debug Bridge (adb) is the command line tool used to communicate with Android devices. It's installed with the SDK, but you may need to install one (any) of the Android API levels for it to be installed in the SDK directory.

Setting it up in Godot

Enter the Editor Settings screen. This screens contains the editor settings for the user account in the computer (It's independent from the project).



Scroll down to the section where the Android settings are located:



In that screen, the path to 3 files needs to be set:

- The *adb* executable (adb.exe on Windows)
- The *jarsigner* executable (from JDK 6 or 8)
- The debug *keystore*

Once that is configured, everything is ready to export to Android!

5.3.7 Exporting for the Web

HTML5 export allows publishing games made in Godot Engine to the browser. This requires support for the recent technologies [WebAssembly](#) and [WebGL 2.0](#) in the user's browser. **Firefox** and **Chromium** (Chrome, Opera) are the most popular supported browsers, **Safari** and **Edge** do not work yet.

Limitations

For security and privacy reasons, many features that work effortlessly on native platforms are more complicated on the web platform. Following is a list of limitations you should be aware of when porting a Godot game to the web.

Exported .html file must not be reused

On export, several text placeholders are replaced in the **generated HTML file** specifically for the given export options. It must not be reused in further exports.

Using cookies for data persistence

Users must **allow cookies** (specifically IndexedDB) if persistence of the `user://` file system is desired. When playing a game presented in an `iframe`, **third-party** cookies must also be enabled. Incognito/private mode also prevents persistence.

The method `OS.is_userfs_persistent()` can be used to check if the `user://` file system is persistent, but can give false positives in some cases.

Full screen and mouse capture

Browsers do not allow arbitrarily **entering full screen** at any time. The same goes for **capturing the cursor**. Instead, these actions have to occur as a response to a JavaScript input event. In Godot, this is most easily done by entering full screen from within an input callback such as `_input` or `_unhandled_input`.

For the same reason, the full screen project setting is ignored.

HTTPClient

The `HTTPClient` implementation for the HTML5 platform has several restrictions:

- Accessing or changing the `StreamPeer` is not possible
- Blocking mode is not available
- No chunked responses
- Host verification cannot be disabled
- Subject to [same-origin policy](#)

Unimplemented functionality

The following functionality is currently unavailable on the HTML5 platform:

- Threads
- GDNative
- Clipboard synchronisation between engine and operating system
- Networking other than `HTTPClient`

Check the [list of open HTML5 issues on Github](#) to see if functionality you're interested in has an issue yet. If not, open one to communicate your interest.

Starting exported games from the local file system

Many browsers will not load exported projects when **opened locally** per `file://` protocol. To get around this, use a local server.

Python offers an easy method for this, using `python -m SimpleHTTPServer` with Python 2 or `python -m http.server` with Python 3 will serve the current working directory on `http://localhost:8000`.

Serving the files

Exporting for the web generates several files to be served from a web server, including a default HTML page for presentation. A custom HTML file can be used, see *Customizing the Web export HTML page*.

The generated `.html` file can be used as `DirectoryIndex` in Apache servers and can be renamed to e.g. `index.html` at any time, its name is never depended on by default.

The HTML page is designed to fit the game perfectly without cutting off parts of the canvas when the browser window is scaled to the game's dimensions. This way it can be inserted into an `<iframe>` with the game's size, as is common on most web game hosting sites.

The other exported files are served as they are, next to the `.html` file, names unchanged. The `.wasm` file is a binary WebAssembly module implementing the engine. The `.pck` file is the Godot main pack containing your game. The `.js` file contains start-up code and is used by the `.html` file to access the engine. The `.png` file contains the boot splash image. It is not used in the default HTML page, but is included for *custom HTML pages*.

The `.pck` file is binary, usually delivered with the MIME-type `application/octet-stream`. The `.wasm` file is delivered as `application/wasm`.

Delivering the files with server-side compression is recommended especially for the `.pck` and `.wasm` files, which are usually large in size. The WebAssembly module compresses particularly well, down to around a quarter of its original size with GZip.

Export options

If a runnable web export template is available, a button appears between the *Stop scene* and *Play edited Scene* buttons in the editor to quickly open the game in the default browser for testing.

If a path to a **Custom HTML shell** file is given, it will be used instead of the default HTML page. See *Customizing the Web export HTML page*.

Head Include is appended into the `<head>` element of the generated HTML page. This allows to, for example, load webfonts and third-party JavaScript APIs, include CSS, or run JavaScript code.

Turning on **Export with Debug** when exporting will, in addition to enabling various debug features of the engine, display a debug output below the canvas when using the default HTML page, displaying JavaScript and engine errors. You can also use the browser-integrated developer console, usually opened with the F12 key, which often shows more information, including WebGL errors.

Calling JavaScript from script

In web builds, the `JavaScript` singleton is implemented. It offers a single method called `eval` that works similarly to the `eval` function of the same name. It takes a string as an argument and executes it as JavaScript code. This allows interacting with the browser in ways not possible with script languages integrated into Godot.

```
func my_func():
    JavaScript.eval("alert('Calling JavaScript per GDScript!');")
```

The value of the last JavaScript statement is converted to a GDScript value and returned by `eval()` under certain circumstances:

- JavaScript `number` is returned as GDScript `float`
- JavaScript `boolean` is returned as GDScript `bool`
- JavaScript `string` is returned as GDScript `String`
- JavaScript `ArrayBuffer`, `TypedArray` and `DataView` are returned as GDScript `PoolByteArray`

```
func my_func2():
    var js_return = JavaScript.eval("var myNumber = 1; myNumber + 2;")
    print(js_return) # prints '3.0'
```

Any other JavaScript value is returned as `null`.

Calling `JavaScript.eval` on platforms other than HTML5 will also return `null`.

The `eval` method also accepts a second, optional Boolean argument, which specifies whether to execute the code in the global execution context, defaulting to `false` to prevent polluting the global namespace:

```
func my_func3():
    # execute in global execution context,
    # thus adding a new JavaScript global variable `MyGlobal`
    JavaScript.eval("var SomeGlobal = {};", true)
```

5.3.8 Customizing the Web export HTML page

Rather than the default HTML page that comes with the export templates, it is also possible to use a custom HTML page. This allows drastic customization of the final web presentation and behavior. The path to custom HTML page is specified in the export options as `Html/Custom Html Shell`.

The default HTML page is available in the Godot Engine repository at [/mist/dist/html/default.html](#).

Placeholder substitution

When exporting the game, several placeholders in the HTML page are substituted by values depending on the export:

Placeholder	substituted by
<code>\$GODOT_BASENAME</code>	Basename of exported files without suffixes, e.g. <code>game</code> when exporting <code>game.html</code>
<code>\$GODOT_DEBUG_ENABLED</code>	<code>true</code> if debugging, <code>false</code> otherwise
<code>\$GODOT_HEAD_INCLUDE</code>	Custom string to include just before the end of the HTML <code><head></code> element

The HTML file must evaluate the JavaScript file `$GODOT_BASENAME.js`. This file defines a global `Engine` object used to start the engine, *see below* for details.

The boot splash image is exported as `$GODOT_BASENAME.png` and can be used e.g. in `` elements.

`$GODOT_DEBUG_ENABLED` can be useful to optionally display e.g. an output console or other debug tools.

`$GODOT_HEAD_INCLUDE` is substituted with the string specified by the export option `Html/Head Include`.

The Engine object

The JavaScript global object `Engine` is defined by `$GODOT_BASENAME.js` and serves as an interface to the engine start-up process.

The object itself has only two methods, `load()` and `unload()`.

`Engine.load(basePath)`

Loads the engine from the passed base path.

Returns a promise that resolves once the engine is loaded.

`Engine.unload()`

Unloads the module to free memory. This is called automatically once the module is instantiated unless explicitly disabled.

Starting an Engine instance

The more interesting interface is accessed by instantiating `Engine` using the `new` operator:

```
var engine = new Engine();
```

This `Engine` instance, referred to as `engine` with a lower-case `e` from here, is a startable instance of the engine, usually a game. To start such an instance, the global `Engine` object must be loaded, then the `engine` instance must be initialized and started.

`engine.init(basePath)`

Initializes the instance. If the engine wasn't loaded yet, a base path must be passed from which the engine will be loaded.

Returns a promise that resolves once the engine is loaded and initialized. It can then be started with `engine.startGame()`

`engine.preloadFile(file, fileName)`

This loads a file so it is available in the file system once the instance is started. This must be called **before** starting the instance.

If `file` is a string, the file will be loaded from that URL and the file name will be retained. If `file` is an `ArrayBuffer` or a view on one, the buffer will be available as a file under the name given by `fileName`.

Returns a promise that resolves once the file is preloaded.

`engine.start(arg1, arg2, ...)`

Starts the instance of the engine, handing the passed strings as arguments to the `main()` function. This allows great control over how the engine is used, but usually the other methods whose names start with `engine.start` are simpler to use.

Returns a promise that resolves once the engine started.

`engine.startGame(mainPack)`

Starts the game with the main pack loaded from the passed URL string and starts the engine with it.

If the engine isn't loaded yet, the base path of the passed URL will be used to load the engine.

Returns a promise that resolves once the game started.

Configuring start-up behaviour

Beside starting the engine, other methods of the engine instance allow configuring the behavior:

`engine.setUnloadAfterInit(enabled)`

Sets whether the Engine will be unloaded automatically after the instance is initialized. This frees browser memory by unloading files that are no longer needed once the instance is initialized. However, if more instances of the engine will be started, the Engine will have to be loaded again.

Defaults to `true`.

`engine.setCanvas(canvasElem)`

By default, the first canvas element on the page is used for rendering. By calling this method, another canvas can be specified.

`engine.setCanvasResized.onStart(enabled)`

Sets whether the canvas will be resized to the width and height specified in the project settings on start. Defaults to `true`.

`engine.setLocale(locale)`

By default, the engine will try to guess the locale to use from the JavaScript environment. It is usually preferable to use a server-side user-specified locale, or at least use the locale requested in the HTTP `Accept-Language` header. This method allows specifying such a custom locale string.

`engine.setExecutableName(execName)`

By default, the base name of the loaded engine files is used for the executable name. This method allows specifying another name.

Customizing the presentation

The following methods are used to implement the presentation:

engine.setProgressFunc(func)

This method is used to display download progress. The passed callback function is called with two number arguments, the first argument specifies bytes loaded so far, the second argument specifies the total number of bytes to load.

```
function printProgress(current, total) {
    console.log("Loaded " + current + " of " + total + " bytes");
}
engine.setProgressFunc(printProgress);
```

If the total is 0, it couldn't be calculated. Possible reasons include:

- Files are delivered with server-side chunked compression
- Files are delivered with server-side compression on Chromium
- Not all file downloads have started yet (usually on servers without multi-threading)

engine.setStdoutFunc(func), engine.set.StderrFunc(func)

These methods allow implementing custom behavior for the `stdout` and `stderr` streams. The functions passed in will be called with one string argument specifying the string to print.

```
function print.Stderr(text) {
    console.warn("Error: " + text);
}
engine.set.StderrFunc(print.Stderr);
```

These methods should usually only be used in debug pages. The `$GODOT_DEBUG_ENABLED` placeholder can be used to check for this.

By default, `console.log()` and `console.warn()` are used respectively.

Accessing the Emscripten Module

If you know what you're doing, you can access the runtime environment (Emscripten's Module) as `engine.rtenv`. Check the official Emscripten documentation for information on how to use it: https://kripken.github.io/emscripten-site/docs/api_reference/module.html

5.3.9 One-click deploy

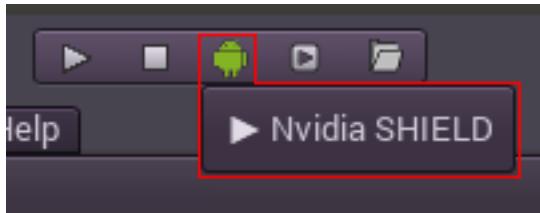
Sounds good, what is it?

This feature will pop up automatically once a platform is properly configured and a supported device is connected to the computer. Since things can go wrong at many levels (platform may not be configured correctly, SDK may be incorrectly installed, device may be improperly configured, kitty ate the USB cable, etc.), it's good to let the user know that it exists.

Some platforms (at the time of this writing, only Android and Blackberry 10) can detect when a USB device is connected to the computer, and offer the user to automatically export, install and run the project (in debug mode) on the device. This feature is called, in industry buzz-words, “One Click Deploy” (though, it’s technically two clicks...).

Steps for one-click deploy

1. Configure target platform.
2. Configure device (make sure it's in developer mode, like the computer, usb is recognized, usb cable is plugged, etc.).
3. Connect the device..
4. And voila!



Click once.. and deploy!

6.1 Viewport and canvas transforms

6.1.1 Introduction

This tutorial is created after a topic that is a little dark for most users, and explains all the 2D transforms going on for nodes from the moment they draw their content locally to the time they are drawn into the screen.

6.1.2 Canvas transform

As mentioned in the previous tutorial, *Canvas layers*, every `CanvasItem` node (remember that `Node2D` and `Control` based nodes use `CanvasItem` as their common root) will reside in a *Canvas Layer*. Every canvas layer has a transform (translation, rotation, scale, etc.) that can be accessed as a *Transform2D*.

Also covered in the previous tutorial, nodes are drawn by default in Layer 0, in the built-in canvas. To put nodes in a different layer, a `CanvasLayer` node can be used.

6.1.3 Global canvas transform

Viewports also have a Global Canvas transform (also a *Transform2D*). This is the master transform and affects all individual *Canvas Layer* transforms. Generally this transform is not of much use, but is used in the `CanvasItem` Editor in Godot's editor.

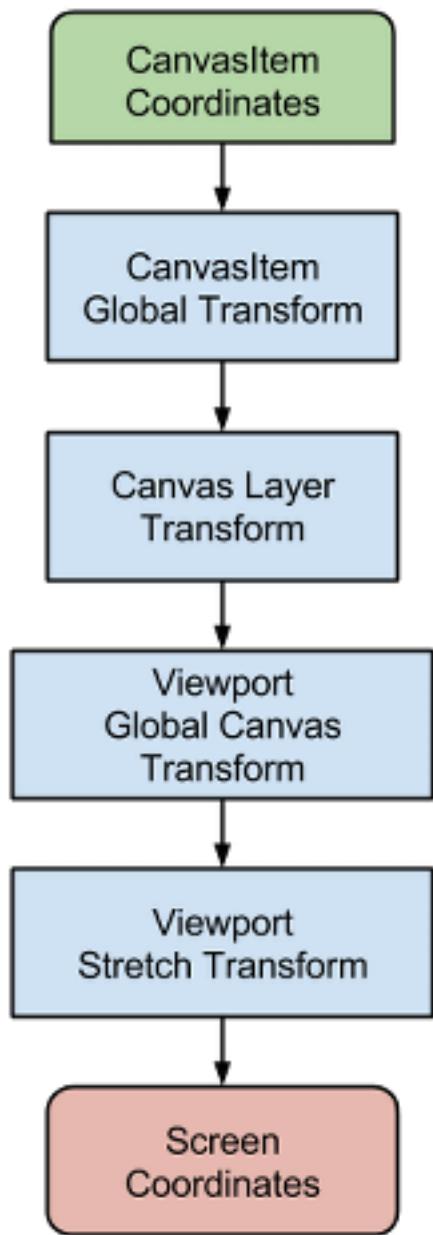
6.1.4 Stretch transform

Finally, viewports have a *Stretch Transform*, which is used when resizing or stretching the screen. This transform is used internally (as described in *Multiple resolutions*), but can also be manually set on each viewport.

Input events received in the `MainLoop._input_event()` callback are multiplied by this transform, but lack the ones above. To convert `InputEvent` coordinates to local `CanvasItem` coordinates, the `CanvasItem.make_input_local()` function was added for convenience.

6.1.5 Transform order

For a coordinate in CanvasItem local properties to become an actual screen coordinate, the following chain of transforms must be applied:



6.1.6 Transform functions

Obtaining each transform can be achieved with the following functions:

Type	Transform
CanvasItem	<code>CanvasItem.get_global_transform()</code>
CanvasLayer	<code>CanvasItem.get_canvas_transform()</code>
CanvasLayer+GlobalCanvas+Stretch	<code>CanvasItem.get_viewport_transform()</code>

Finally then, to convert a `CanvasItem` local coordinates to screen coordinates, just multiply in the following order:

```
var screen_coord = get_viewport_transform() * (get_global_transform() * local_pos)
```

Keep in mind, however, that it is generally not desired to work with screen coordinates. The recommended approach is to simply work in Canvas coordinates (`CanvasItem.get_global_transform()`), to allow automatic screen resolution resizing to work properly.

6.1.7 Feeding custom input events

It is often desired to feed custom input events to the scene tree. With the above knowledge, to correctly do this, it must be done the following way:

```
var local_pos = Vector2(10,20) # local to Control/Node2D
var ie = InputEventMouseButton.new()
ie.button_index = BUTTON_LEFT
ie.position = get_viewport_transform() * (get_global_transform() * local_pos)
get_tree().input_event(ie)
```

6.2 Canvas layers

6.2.1 Viewport and Canvas items

Regular 2D nodes, such as `Node2D` or `Control` both inherit from `CanvasItem`, which is the base for all 2D nodes. `CanvasItems` can be arranged in trees and they will inherit their transform. This means that when moving the parent, the children will be moved too.

These nodes are placed as direct or indirect children to a `Viewport`, and will be displayed through it.

`Viewport` has the property `Viewport.canvas_transform`, which allows to transform all the `CanvasItem` hierarchy by a custom `Transform2D` transform. Nodes such as `Camera2D`, work by changing that transform.

Changing the canvas transform is useful because it is a lot more efficient than moving the root canvas item (and hence the whole scene). Canvas transform is a simple matrix that offsets the whole 2D drawing, so it's the most efficient way to do scrolling.

6.2.2 Not enough...

But this is not enough. There are often situations where the game or application may not want *everything* transformed by the canvas transform. Examples of this are:

- **Parallax Backgrounds:** Backgrounds that move slower than the rest of the stage.
- **HUD:** Head's up display, or user interface. If the world moves, the life counter, score, etc. must stay static.
- **Transitions:** Effects used for transitions (fades, blends) may also want it to remain at a fixed location.

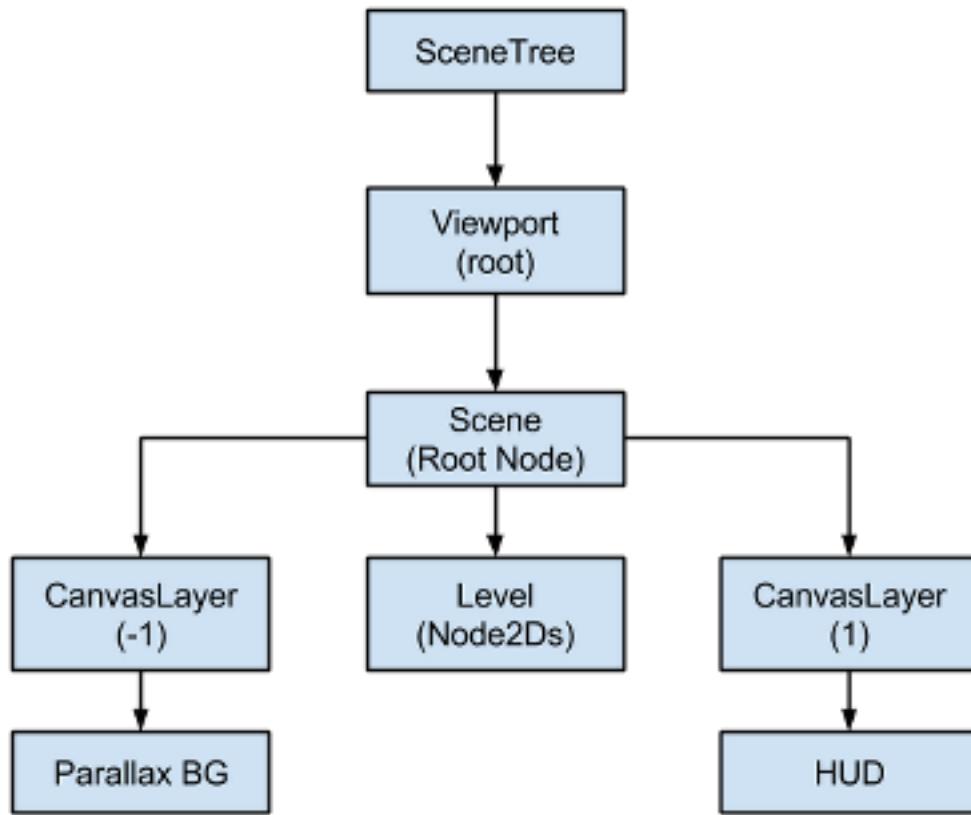
How can these problems be solved in a single scene tree?

6.2.3 CanvasLayers

The answer is *CanvasLayer*, which is a node that adds a separate 2D rendering layer for all its children and grandchildren. Viewport children will draw by default at layer “0”, while a *CanvasLayer* will draw at any numeric layer. Layers with a greater number will be drawn above those with a smaller number. *CanvasLayers* also have their own transform, and do not depend on the transform of other layers. This allows the UI to be fixed in-place, while the world moves.

An example of this is creating a parallax background. This can be done with a *CanvasLayer* at layer “-1”. The screen with the points, life counter and pause button can also be created at layer “1”.

Here’s a diagram of how it looks:



CanvasLayers are independent of tree order, and they only depend on their layer number, so they can be instantiated when needed.

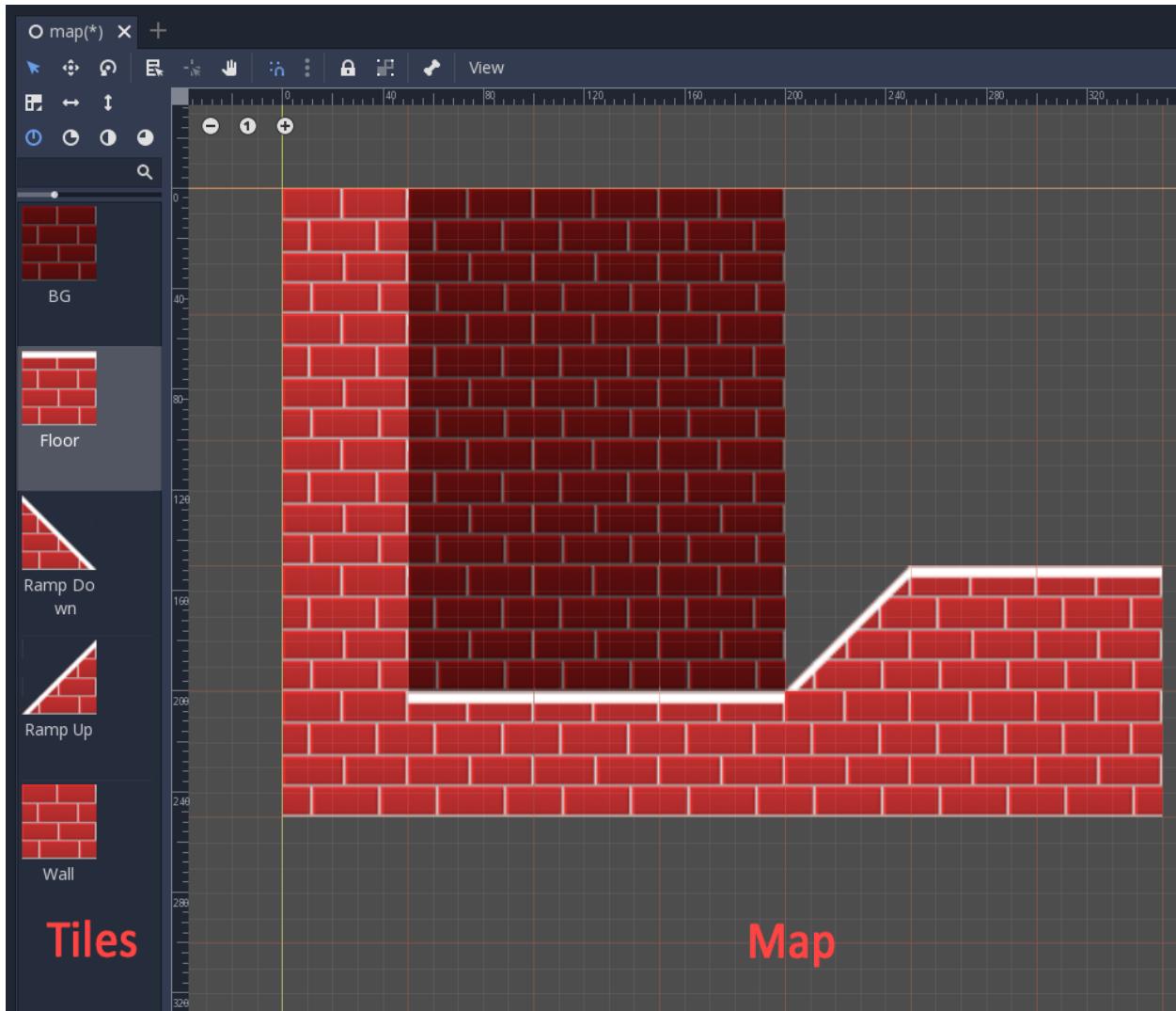
6.2.4 Performance

Even though there shouldn’t be any performance limitation, it is not advised to use excessive amount of layers to arrange drawing order of nodes. The most optimal way will always be arranging them by tree order. 2d nodes also have a property for controlling their drawing order (see *Node2D.z_index*).

6.3 Using tilemaps

6.3.1 Introduction

Tilemaps are a simple and quick way to make 2D game levels. Basically, you start with bunch of reference tiles (or pieces) that can be put in a grid, as many times each as desired:



Collisions can also be added to the tiles, allowing for both 2D side scrolling and top down games.

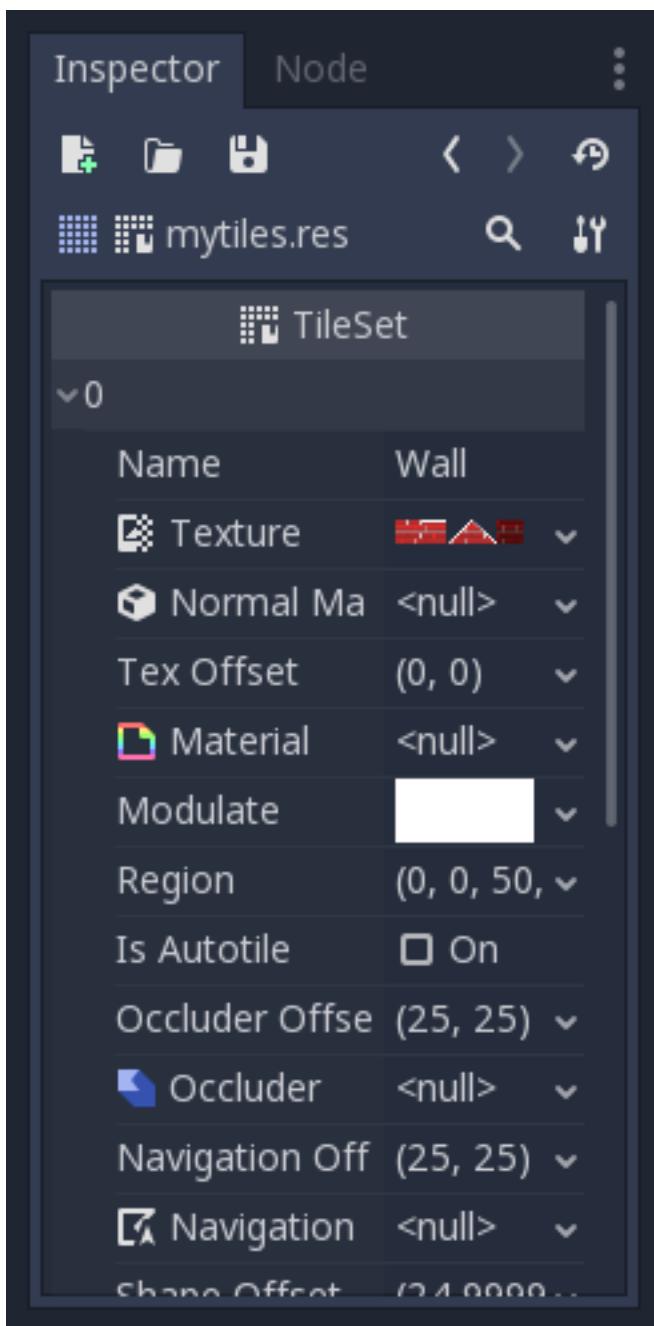
6.3.2 Making a tileset

To begin, a tileset needs to be made. Here are some tiles for it. They are all in the same image because artists will often prefer this. Having them as separate images also works.



Create a new project and move the above png image into the directory. Next go into the image settings and turn off *Filter*, keeping it on will cause issues later.

We will be creating a *TileSet* resource. While this resource exports properties, it's pretty difficult to get complex data into it and maintain it. Here is what it would look like to manually edit the resource:



There's enough properties to get by. With some effort, editing this way can work. But the easiest way to edit and maintain a tileset is exporting it from a specially-crafted scene!

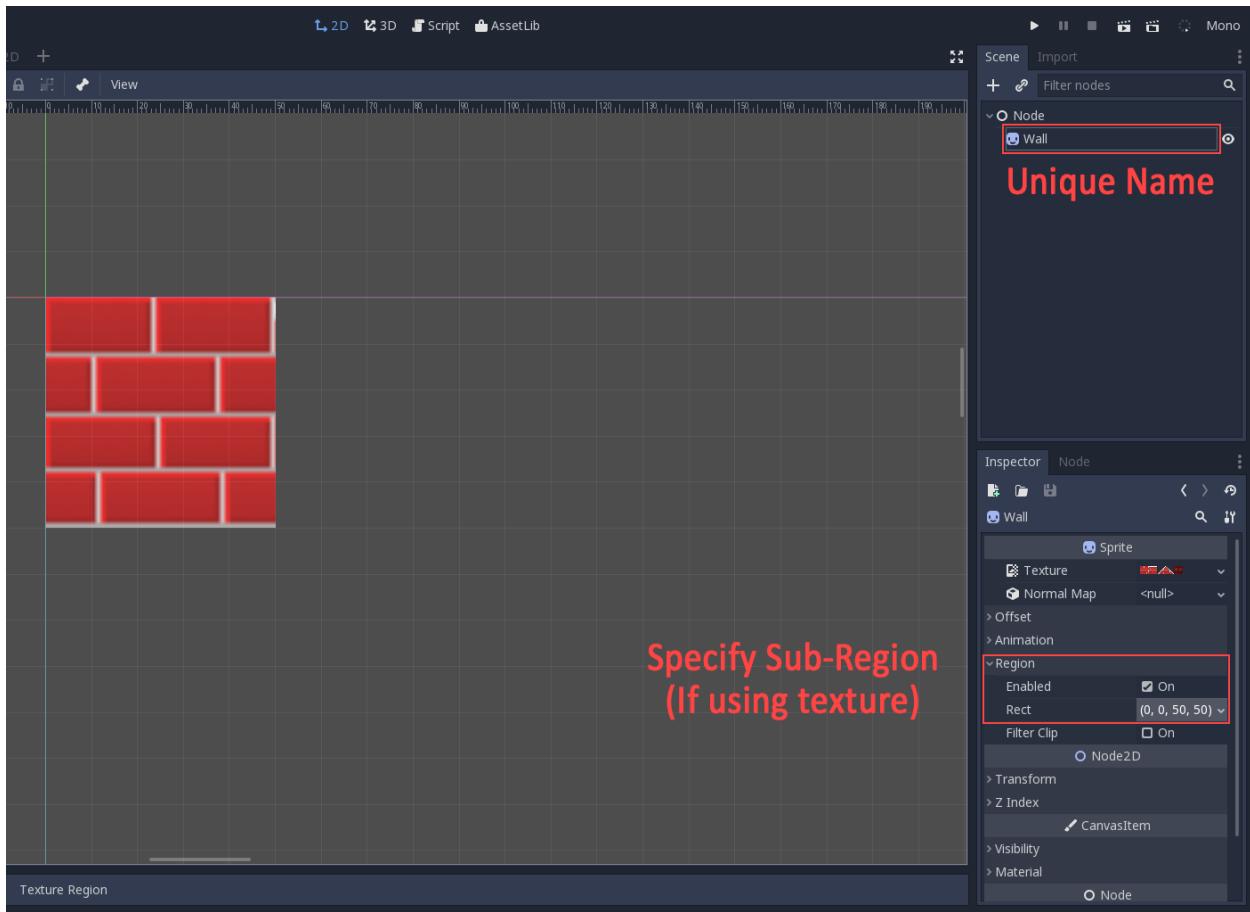
6.3.3 TileSet scene

Create a new scene with a regular node or node2d as root. For each tile, add a sprite as a child. Since tiles here are 50x50, you should turn on the grid (G key) and enable snap. Moving tiles with the mouse might still be inaccurate so use your arrow keys as well.

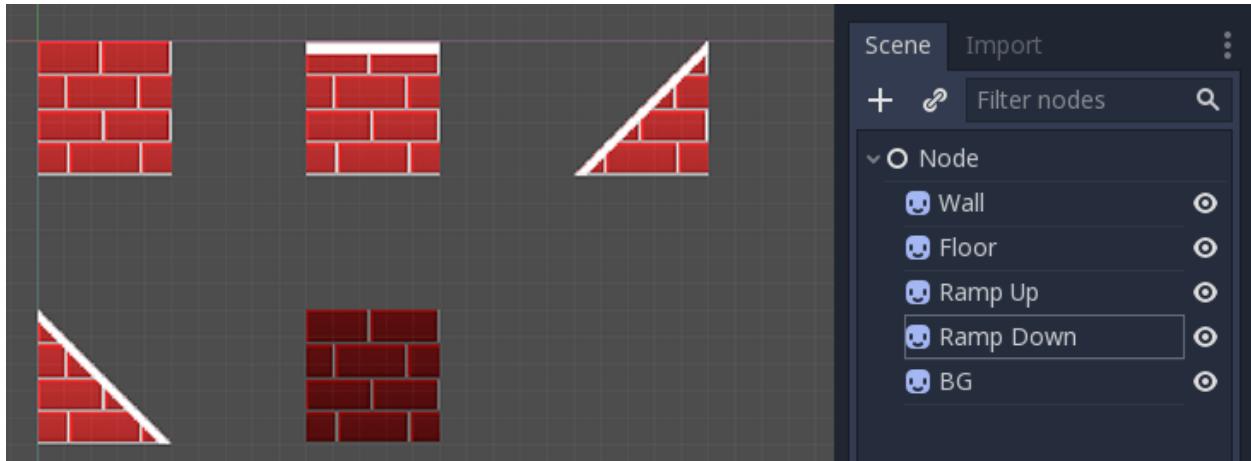
If more than one tile is present in the source image, make sure to use the region property of the sprite to adjust which part of the texture is being used.

Finally, make sure to name your sprite node correctly. This will ensure that, in subsequent edits to the tileset (for example, if added collision, changed the region, etc), the tile will still be **identified correctly and updated**. This name should be unique.

Sounds like a lot of requirements, so here's a screenshot that shows where everything of relevance is:

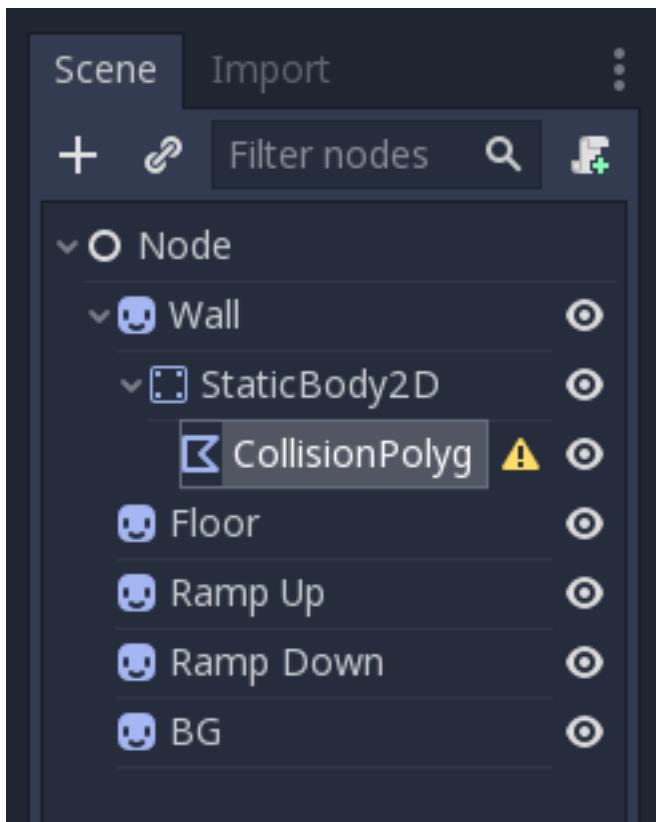


Continue adding all the tiles, adjusting the offsets if needed (that is, if you have multiple tiles in a single source image). Again, *remember that their names must be unique*.

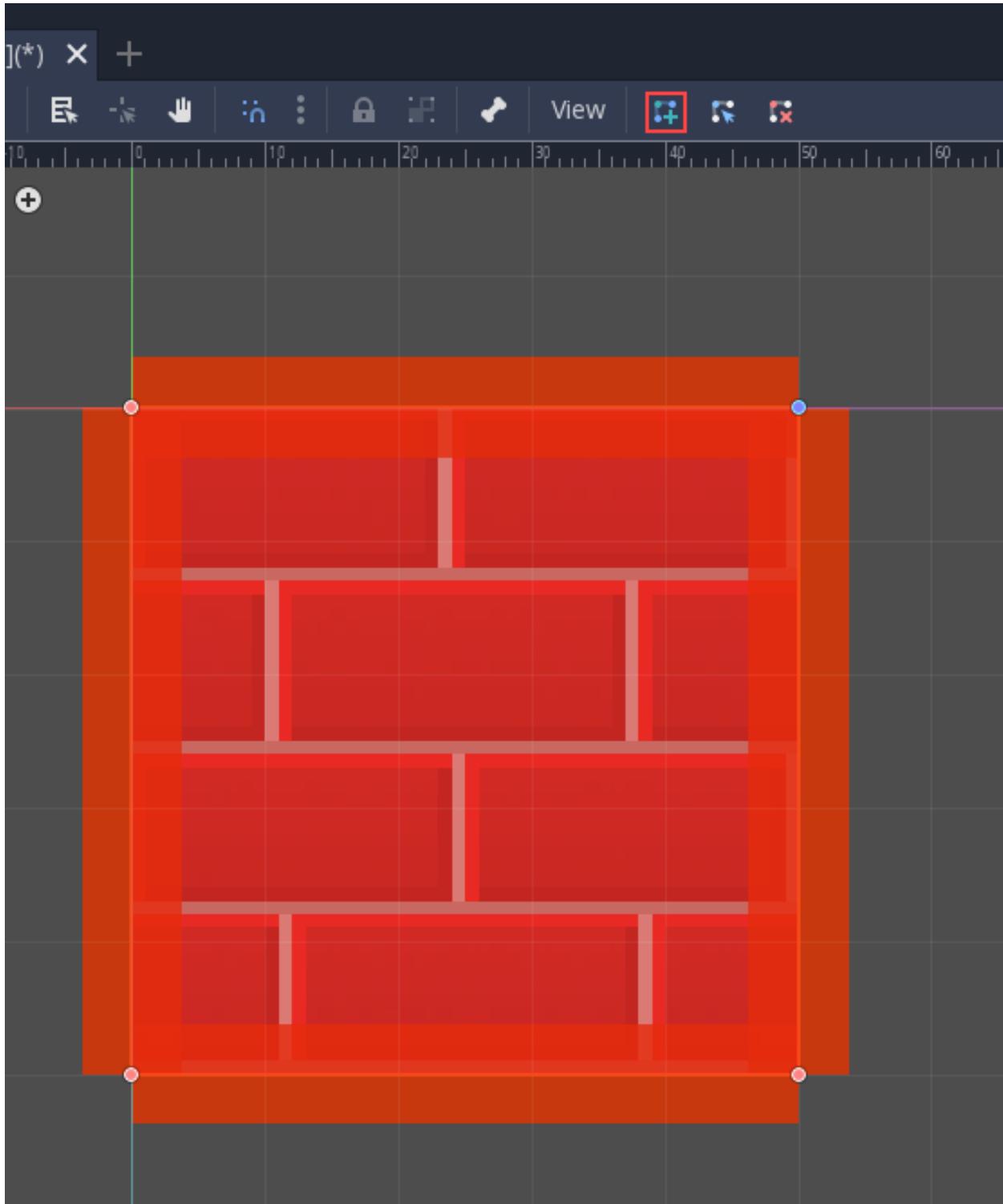


6.3.4 Collision

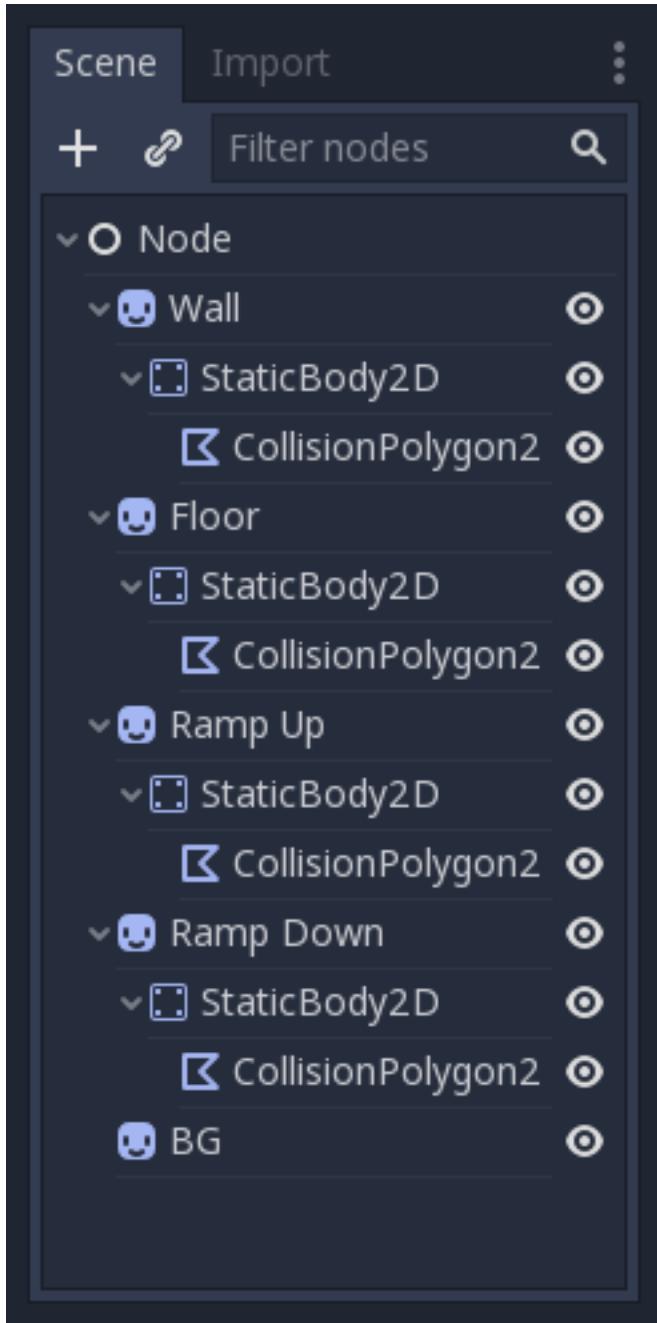
To add collision to a tile, create a `StaticBody2D` child for each sprite. This is a static collision node. Then create a `CollisionShape2D` or `CollisionPolygon` as a child of the `StaticBody2D`. The `CollisionPolygon` is recommended because it is easier to edit.



Finally, edit the polygon, this will give the tile a collision, and fix the warning icon next to the `CollisionPolygon` node. **Remember to use snap!** Using snap will make sure collision polygons are aligned properly, allowing a character to walk seamlessly from tile to tile. Also **do not scale or move** the collision and/or collision polygon nodes. Leave them at offset 0,0, with scale 1,1 and rotation 0 with respect to the parent sprite.



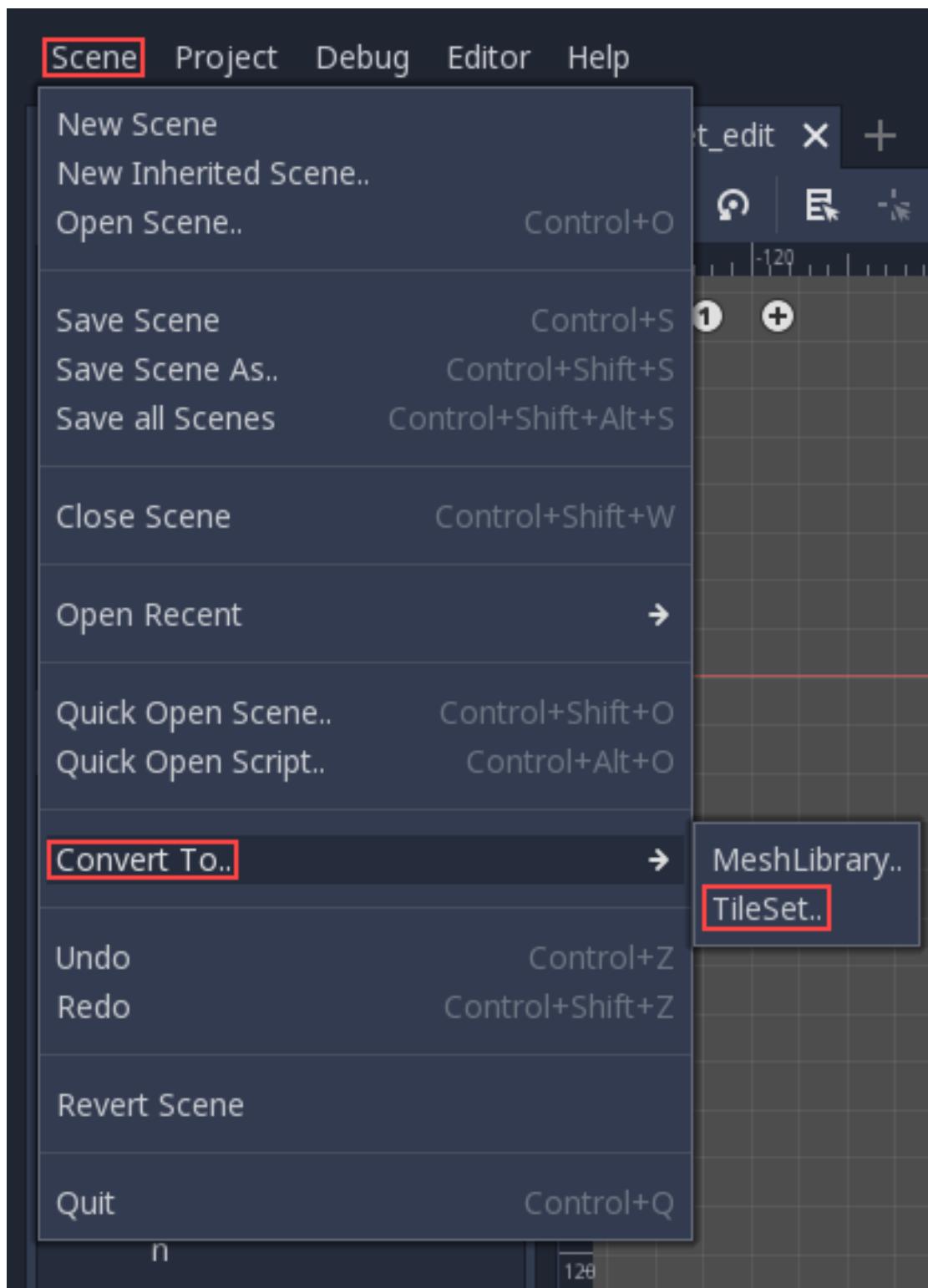
Keep adding collisions to the tiles until we are done. Note that BG is just a background, so it should not have a collision.



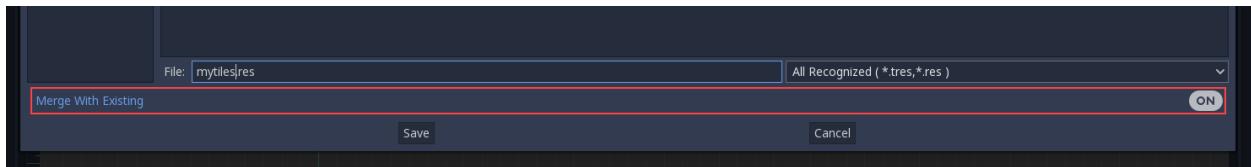
OK! We're done! Remember to save this scene for future edit. Name it "tileset_edit.scn" or something like that.

6.3.5 Exporting a TileSet

With the scene created and opened in the editor, the next step will be to create a tileset. Use Scene > Convert To > Tile Set from the Scene Menu:

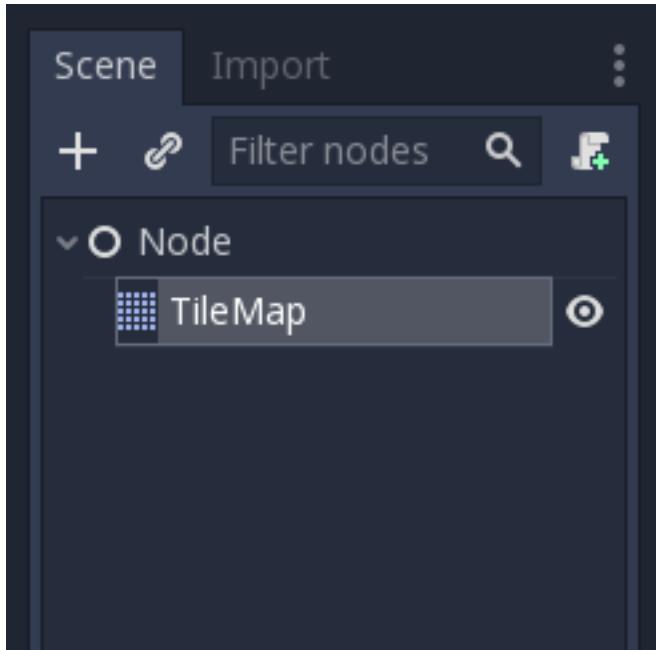


Then choose a filename, like “mytiles.tres”. Make sure the “Merge With Existing” option is toggled on. This way, every time the tileset resource file is overwritten, existing tiles are merged and updated (they are referenced by their unique name, so again, **name your tiles properly**).

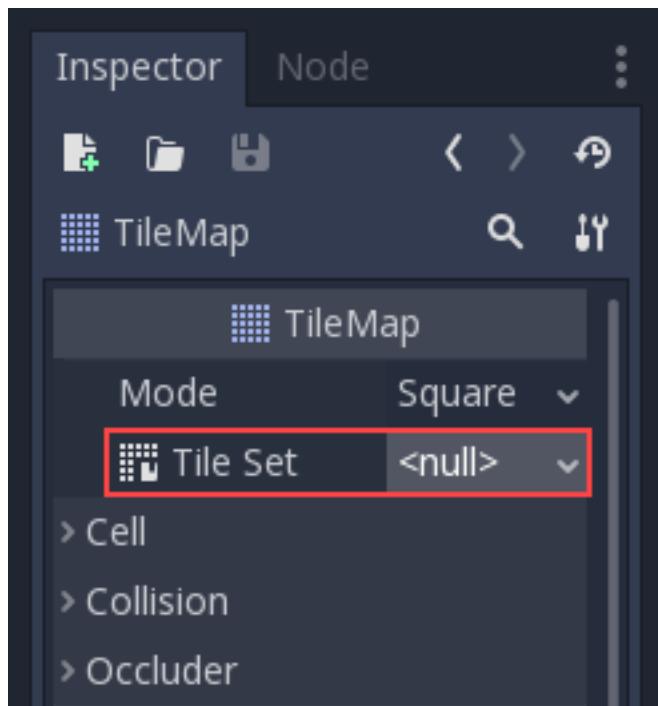


6.3.6 Using the TileSet in a TileMap

Create a new scene, using any node or node2d as root, and then create a *TileMap* as a child.



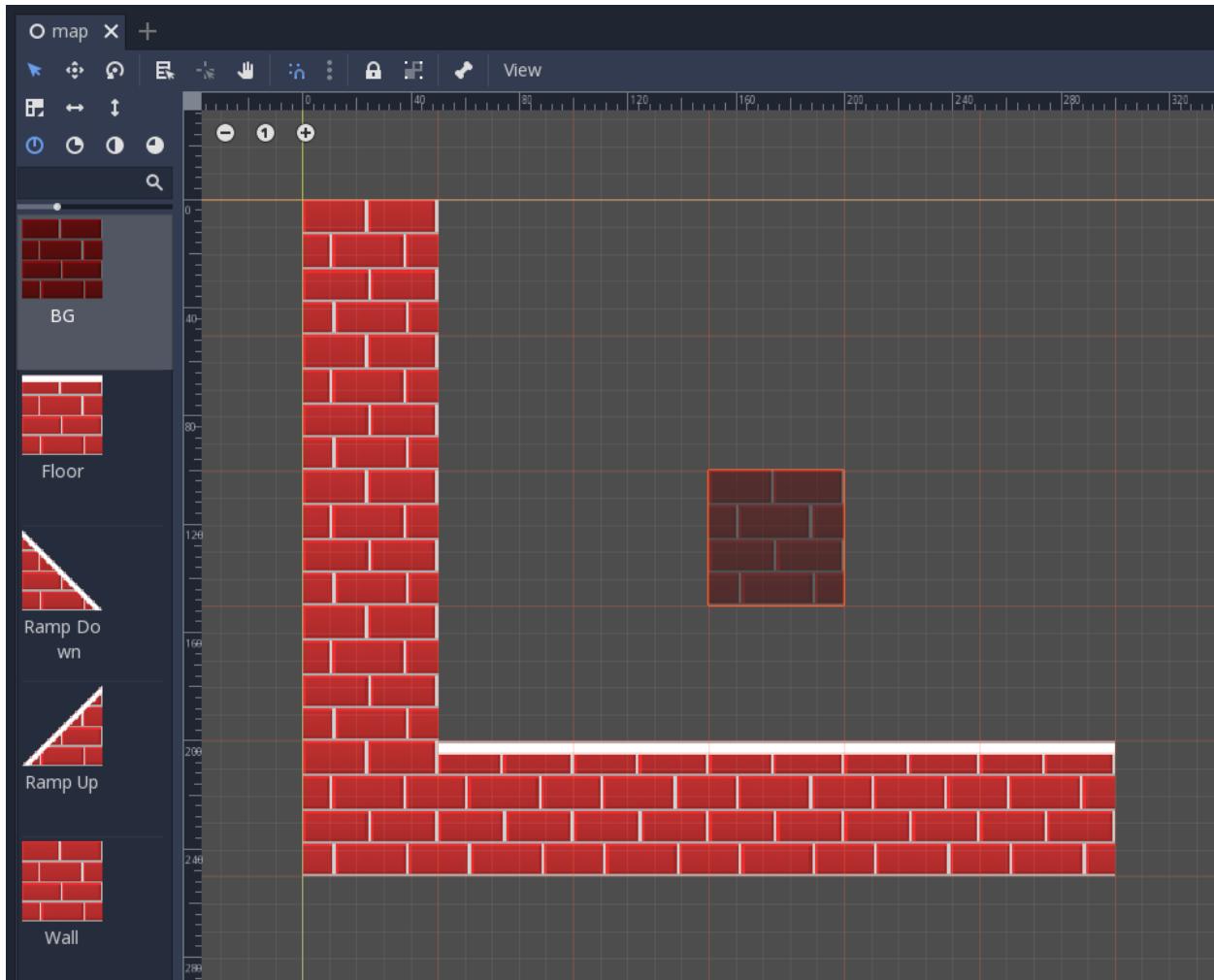
Go to the TileSet property of this node and assign the one created in previous steps:



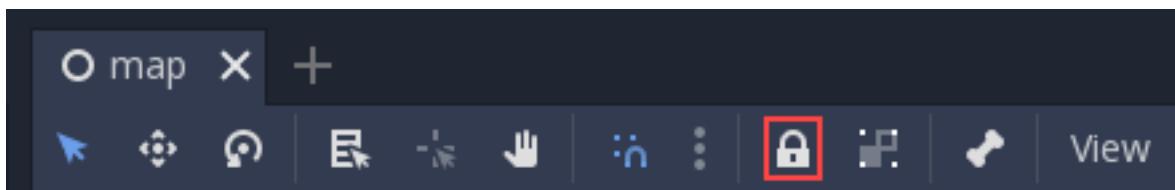
Also set the cell size to '50', since that is the size used by the tiles. Quadrant size is a tuning value, which means that the engine will draw and cull the tilemap in blocks of 16x16 tiles. This value is usually fine and does not need to be changed, but can be used to fine tune performance in specific cases (if you know what you are doing).

6.3.7 Painting your world

With all set, make sure the TileMap node is selected. A red grid will appear on screen, allowing to paint on it with the selected tile on the left palette.



To avoid accidentally moving and selecting the tilemap node (something common, given it's a huge node), it is recommended that you lock it, using the lock button:

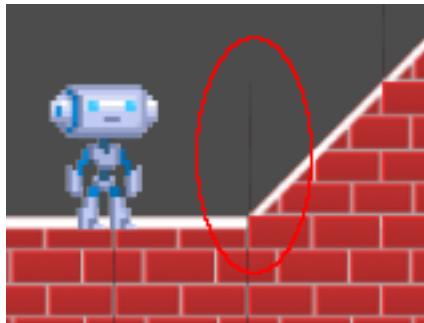


If you accidentally place a tile somewhere you don't want it to be, you can delete it with RMB while in the tilemap editor.

You can also flip and rotate sprites in the TileMap editor (note: flipping the sprite in the TileSet will have no effect). Icons at the top right of the editor allow flipping and rotating of the currently selected sprite - you can also use the A and S keys to flip the sprite horizontally and vertically. With a brick pattern like this tutorial uses, flipping the sprites would create unpleasant discontinuities unless you're flipping an entire region of bricks. But for some kinds of tiles, flipping can be a convenient and space-saving feature.

6.3.8 Offset and scaling artifacts

When using a single texture for all the tiles, scaling the tileset (or even moving to a non pixel-aligned location) will most likely result in filtering artifacts like so:



This is unavoidable, as it is the way the hardware bilinear filter works. So, to avoid this situation, there are a few workarounds. Try the one that looks better for you:

- Disable filtering for either the tileset texture or the entire image loader (see the *Importing Images* asset pipeline tutorial).
- Enable pixel snap (set: “Scene > Project Settings > Display/use_2d_pixel_snap” to true).
- Viewport Scaling can often help with shrinking the map (see the *Viewports* tutorial).
- Use a single image for each tile. This will remove all artifacts, but can be more cumbersome to implement.

6.4 Custom drawing in 2D

6.4.1 Why?

Godot has nodes to draw sprites, polygons, particles, and all sorts of stuff. For most cases this is enough, but not always. Before crying in fear, angst, and rage because a node to draw that specific *something* does not exist... it would be good to know that it is possible to easily make any 2D node (be it *Control* or *Node2D* based) draw custom commands. It is *really* easy to do it too.

6.4.2 But...

Custom drawing manually in a node is *really* useful. Here are some examples why:

- Drawing shapes or logic that is not handled by nodes (example: making a node that draws a circle, an image with trails, a special kind of animated polygon, etc).
- Visualizations that are not that compatible with nodes: (example: a tetris board). The tetris example uses a custom draw function to draw the blocks.
- Managing drawing logic of a large amount of simple objects (in the hundreds of thousands). Using a thousand nodes is probably not nearly as efficient as drawing, but a thousand of draw calls are cheap. Check the “Shower of Bullets” demo as example.
- Making a custom UI control. There are plenty of controls available, but it’s easy to run into the need to make a new, custom one.

6.4.3 OK, how?

Add a script to any *CanvasItem* derived node, like *Control* or *Node2D*. Then override the `_draw()` function.

```
extends Node2D

func _draw():
    #your draw commands here
    pass
```

Draw commands are described in the *CanvasItem* class reference. There are plenty of them.

6.4.4 Updating

The `_draw()` function is only called once, and then the draw commands are cached and remembered, so further calls are unnecessary.

If re-drawing is required because a state or something else changed, simply call `CanvasItem.update()` in that same node and a new `_draw()` call will happen.

Here is a little more complex example. A texture variable that will be redrawn if modified:

```
extends Node2D

export var texture setget _set_texture

func _set_texture(value):
    #if the texture variable is modified externally,
    #this callback is called.
    texture=value #texture was changed
    update() #update the node

func _draw():
    draw_texture(texture,Vector2())
```

In some cases, it may be desired to draw every frame. For this, just call `update()` from the `_process()` callback, like this:

```
extends Node2D

func _draw():
    #your draw commands here
    pass

func _process(delta):
    update()

func _ready():
    set_process(true)
```

6.4.5 An example: drawing circular arcs

We will now use the custom drawing functionality of the Godot Engine to draw something that Godot doesn't provide functions for. As an example, Godot provides a `draw_circle()` function that draws a whole circle. However, what about drawing a portion of a circle? You will have to code a function to perform this, and draw it yourself.

Arc function

An arc is defined by its support circle parameters. That is: the center position, and the radius. And the arc itself is then defined by the angle it starts from, and the angle at which it stops. These are the 4 parameters we have to provide to our drawing. We'll also provide the color value so we can draw the arc in different colors if we wish.

Basically, drawing a shape on screen requires it to be decomposed into a certain number of points, linked from one to the following one. As you can imagine, the more points your shape is made of, the smoother it will appear, but the heavier it will be, in terms of processing cost. In general, if your shape is huge (or in 3D, close to the camera), it will require more points to be drawn without it being angular-looking. On the contrary, if your shape is small (or in 3D, far from the camera), you may reduce its number of points to save processing costs. This is called *Level of Detail (LoD)*. In our example, we will simply use a fixed number of points, no matter the radius.

```
func draw_circle_arc( center, radius, angle_from, angle_to, color ) :
    var nb_points = 32
    var points_arc = Vector2Array()

    for i in range(nb_points+1):
        var angle_point = angle_from + i*(angle_to-angle_from)/nb_points - 90
        var point = center + Vector2( cos(deg2rad(angle_point)), sin(deg2rad(angle_
→point)) ) * radius
        points_arc.push_back( point )

    for indexPoint in range(nb_points):
        draw_line(points_arc[indexPoint], points_arc[indexPoint+1], color)
```

Remember the number of points our shape has to be decomposed into? We fixed this number in the `nb_points` variable to a value of 32. Then, we initialize an empty `Vector2Array`, which is simply an array of `Vector2`.

The next step consists of computing the actual positions of these 32 points that compose an arc. This is done in the first for-loop: we iterate over the number of points for which we want to compute the positions, plus one to include the last point. We first determine the angle of each point, between the starting and ending angles.

The reason why each angle is reduced by 90° is that we will compute 2D positions out of each angle using trigonometry (you know, cosine and sine stuff...). However, to be simple, `cos()` and `sin()` use radians, not degrees. The angle of 0° (0 radian) starts at 3 o'clock, although we want to start counting at 0 o'clock. So, we just reduce each angle by 90° in order to start counting from 0 o'clock.

The actual position of a point located on a circle at angle ‘angle’ (in radians) is given by `Vector2(cos(angle), sin(angle))`. Since `cos()` and `sin()` return values between -1 and 1, the position is located on a circle of radius 1. To have this position on our support circle, which has a radius of ‘radius’, we simply need to multiply the position by ‘radius’. Finally, we need to position our support circle at the ‘center’ position, which is performed by adding it to our `Vector2` value. Finally, we insert the point in the `Vector2Array` which was previously defined.

Now, we need to actually draw our points. As you can imagine, we will not simply draw our 32 points: we need to draw everything that is between each of them. We could have computed every point ourselves using the previous method, and drew it one by one. But this is too complicated and inefficient (except if explicitly needed). So, we simply draw lines between each pair of points. Unless the radius of our support circle is very big, the length of each line between a pair of points will never be long enough to see them. If this happens, we simply would need to increase the number of points.

Draw the arc on screen

We now have a function that draws stuff on the screen: it is time to call in the `_draw()` function.

```
func _draw():
    var center = Vector2(200,200)
```

```
var radius = 80
var angle_from = 75
var angle_to = 195
var color = Color(1.0, 0.0, 0.0)
draw_circle_arc( center, radius, angle_from, angle_to, color )
```

Result:



Arc polygon function

We can take this a step further and not only write a function that draws the plain portion of the disc defined by the arc, but also its shape. The method is exactly the same as previously, except that we draw a polygon instead of lines:

```
func draw_circle_arc_poly( center, radius, angle_from, angle_to, color ):
    var nb_points = 32
    var points_arc = Vector2Array()
    points_arc.push_back(center)
    var colors = ColorArray([color])

    for i in range(nb_points+1):
        var angle_point = angle_from + i*(angle_to-angle_from)/nb_points - 90
        points_arc.push_back(center + Vector2( cos(deg2rad(angle_point)), sin(_
        &deg2rad(angle_point)) ) * radius)
    draw_polygon(points_arc, colors)
```



Dynamic custom drawing

Alright, we are now able to draw custom stuff on screen. However, it is very static: let's make this shape turn around the center. The solution to do this is simply to change the angle_from and angle_to values over time. For our example, we will simply increment them by 50. This increment value has to remain constant, else the rotation speed will change accordingly.

First, we have to make both angle_from and angle_to variables global at the top of our script. Also note that you can store them in other nodes and access them using get_node().

```
extends Node2D

var rotation_ang = 50
var angle_from = 75
var angle_to = 195
```

We make these values change in the `_process(delta)` function. To activate this function, we need to call `set_process(true)` in the `_ready()` function.

We also increment our angle_from and angle_to values here. However, we must not forget to wrap() the resulting values between 0 and 360°! That is, if the angle is 361°, then it is actually 1°. If you don't wrap these values, the script will work correctly. But angle values will grow bigger and bigger over time, until they reach the maximum integer value Godot can manage ($2^{31} - 1$). When this happens, Godot may crash or produce unexpected behavior. Since Godot doesn't provide a wrap() function, we'll create it here, as it is relatively simple.

Finally, we must not forget to call the `update()` function, which automatically calls `_draw()`. This way, you can control when you want to refresh the frame.

```
func _ready():
    set_process(true)
```

```
func wrap(value, min_val, max_val):
    var f1 = value - min_val
    var f2 = max_val - min_val
    return fmod(f1, f2) + min_val

func _process(delta):
    angle_from += rotation_ang
    angle_to += rotation_ang

    # we only wrap angles if both of them are bigger than 360
    if (angle_from > 360 && angle_to > 360):
        angle_from = wrap(angle_from, 0, 360)
        angle_to = wrap(angle_to, 0, 360)
    update()
```

Also, don't forget to modify the `_draw()` function to make use of these variables:

```
func _draw():
    var center = Vector2(200,200)
    var radius = 80
    var color = Color(1.0, 0.0, 0.0)

    draw_circle_arc( center, radius, angle_from, angle_to, color )
```

Let's run! It works, but the arc is rotating insanely fast! What's wrong?

The reason is that your GPU is actually displaying the frames as fast as it can. We need to “normalize” the drawing by this speed. To achieve, we have to make use of the ‘delta’ parameter of the `_process()` function. ‘delta’ contains the time elapsed between the two last rendered frames. It is generally small (about 0.0003 seconds, but this depends on your hardware). So, using ‘delta’ to control your drawing ensures that your program runs at the same speed on everybody’s hardware.

In our case, we simply need to multiply our ‘`rotation_ang`’ variable by ‘`delta`’ in the `_process()` function. This way, our 2 angles will be increased by a much smaller value, which directly depends on the rendering speed.

```
func _process(delta):
    angle_from += rotation_ang * delta
    angle_to += rotation_ang * delta

    # we only wrap angles if both of them are bigger than 360
    if (angle_from > 360 && angle_to > 360):
        angle_from = wrap(angle_from, 0, 360)
        angle_to = wrap(angle_to, 0, 360)
    update()
```

Let's run again! This time, the rotation displays fine!

6.4.6 Tools

Drawing your own nodes might also be desired while running them in the editor, to use as preview or visualization of some feature or behavior.

Remember, to just use the “tool” keyword at the top of the script (check the *GDScript* reference if you forgot what this does).

6.5 Particle Systems (2D)

6.5.1 Intro

A simple (but flexible enough for most uses) particle system is provided. Particle systems are used to simulate complex physical effects such as sparks, fire, magic particles, smoke, mist, magic, etc.

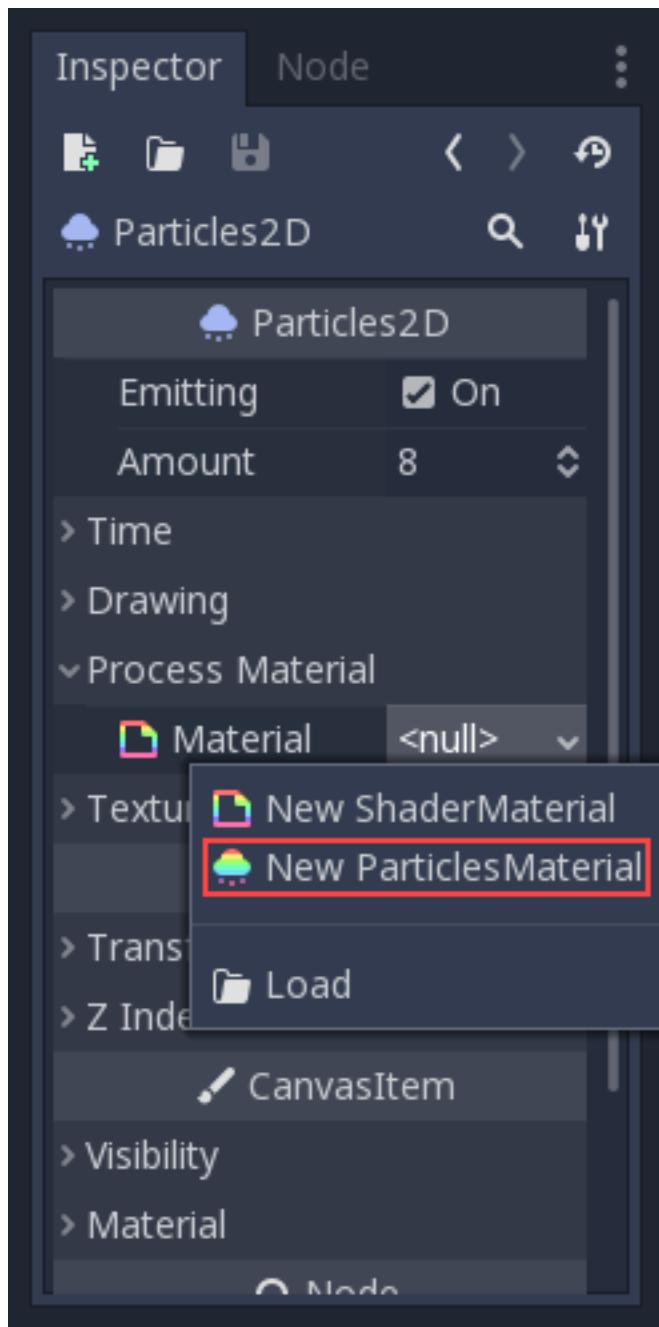
The idea is that a “particle” is emitted at a fixed interval and with a fixed lifetime. During its lifetime, every particle will have the same base behavior. What makes every particle different and provides a more organic look is the “randomness” associated to each parameter. In essence, creating a particle system means setting base physics parameters and then adding randomness to them.

Particles2D

Particle systems are added to the scene via the *Particles2D* node. However, after creating that node you will notice that only a white dot was created, and that there is a warning icon next to your Particles2D node in the inspector. This is because the node needs a ParticlesMaterial to function.

ParticlesMaterial

To add a process material to your particles node, go to Process Material in your inspector panel. Click on the box next to material, and from the dropdown menu select New Particles Material.

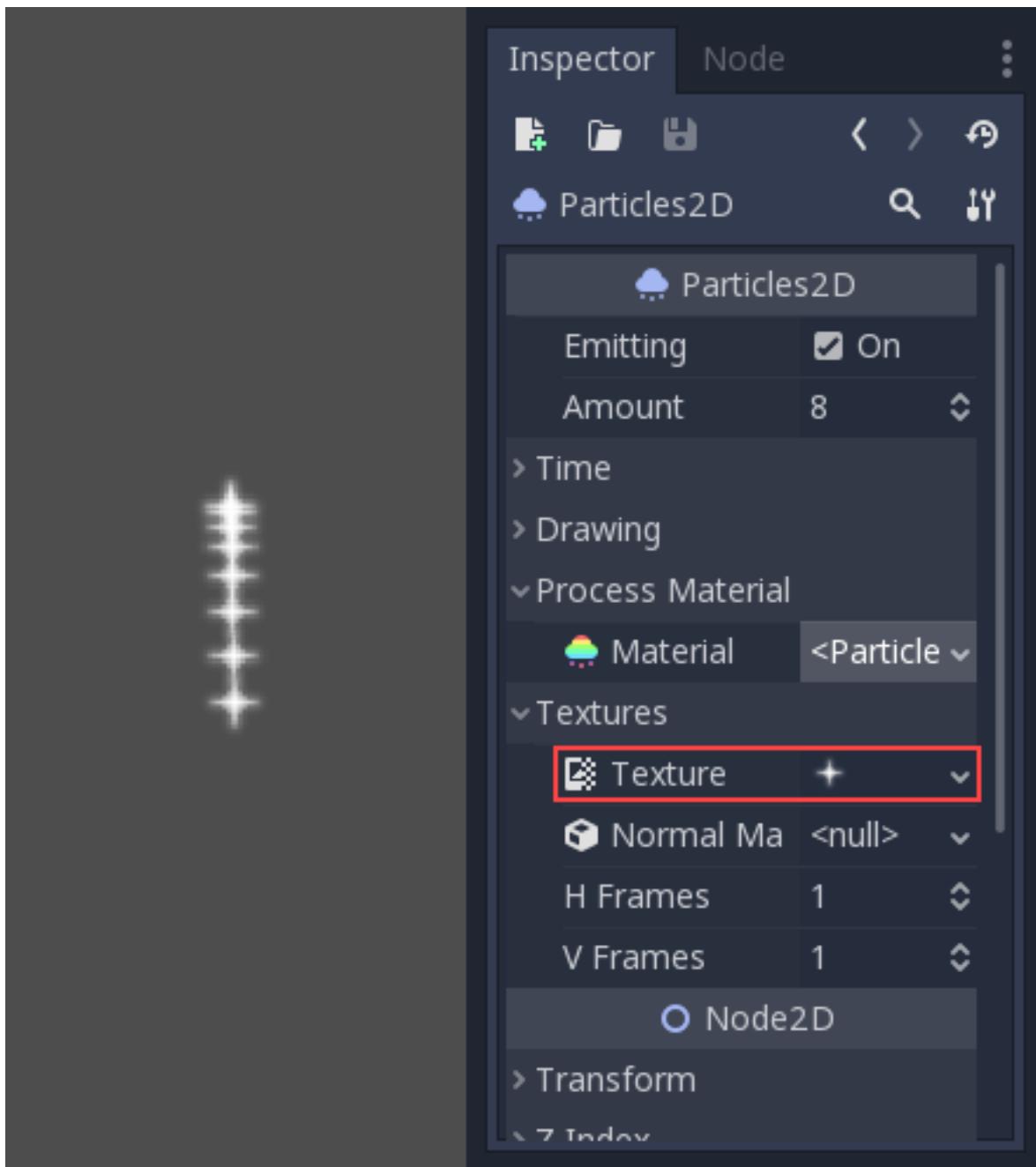


Your Particles2D node should now be emitting white points downward.



Texture

A particle system uses a single texture (in the future this might be extended to animated textures via spritesheet). The texture is set via the relevant texture property:



6.5.2 Time Parameters

Lifetime

The time in seconds that every particle will stay alive. When lifetime ends, a new particle is created to replace it.

Lifetime: 0.5

Lifetime: 4.0

One Shot

When enabled, a Particles2D node will emit all of its particles once and then never again.

Preprocess

Particle systems begin with zero particles emitted, then start emitting. This can be an inconvenience when just loading a scene and systems like a torch, mist, etc. begin emitting the moment you enter. Preprocess is used to let the system process a given number of seconds before it is actually drawn the first time.

Speed Scale

The speed scale has a default value of 1, and is used to adjust the speed of a particle system. Lowering the value will make the particles slower, increasing the value will make the particles much faster.

Explosiveness

If lifetime is 1 and there are ten particles, it means a particle will be emitted every 0.1 seconds. The explosiveness parameter changes this, and forces particles to be emitted all together. Ranges are:

- 0: Emit particles at regular intervals (default value).
- 1: Emit all particles simultaneously.

Values in the middle are also allowed. This feature is useful for creating explosions or sudden bursts of particles:

Randomness

All physics parameters can be randomized. Random values range from 0 to 1. The formula to randomize a parameter is:

```
initial_value = param_value + param_value * randomness
```

Fixed FPS

This setting can be used to set the particle system to render at a fixed FPS. For instance, changing the value to 2 will make the particles render at 2 frames per second. Note this does not slow down the particle system itself.

Fract Delta

This can be used to turn Fract Delta on or off.

6.5.3 Drawing Parameters

Visibility Rect

The `W` and `H` values control width and height of the visibility rectangle. The `X` and `Y` values control the position of the upper-left corner of the visibility rectangle relative to the particle emitter.

Local Coords

By default this option is on, and it means that the space that particles are emitted to is relative to the node. If the node is moved, all particles are moved with it:

If disabled, particles will emit to global space, meaning that if the node is moved, already emitted particles are not affected:

Draw Order

This controls the order in which individual particles are drawn. `Index` means particles are drawn according to their emission order (default). `Lifetime` means they are drawn in order of remaining lifetime.

6.5.4 ParticlesMaterial settings

Direction

This is the base angle at which particles emit. Default is 0 (down):

Changing it will change the emissor direction, but gravity will still affect them:

This parameter is useful because, by rotating the node, gravity will also be rotated. Changing direction allows them to be separated.

Spread

This parameter is the angle in degrees which will be randomly added in either direction to the base `Direction`. A spread of 180 will emit in all directions (+/- 180).

Gravity

The gravity applied to every particle.

Initial Velocity

Linear velocity is the speed at which particles will be emitted (in pixels/sec). Speed might later be modified by gravity or other accelerations (as described further below).

Angular Velocity

Angular velocity is the initial angular velocity applied to particles.

Spin Velocity

Spin velocity is the speed at which particles turn around their center (in degrees/sec).

Orbit Velocity

Orbit velocity is used to make particles turn around their center.

Linear Acceleration

The linear acceleration applied to each particle.

Radial Acceleration

If this acceleration is positive, particles are accelerated away from the center. If negative, they are absorbed towards it.

Tangential Acceleration

This acceleration will use the tangent vector to the center. Combining with radial acceleration can do nice effects.

Damping

Damping applies friction to the particles, forcing them to stop. It is especially useful for sparks or explosions, which usually begin with a high linear velocity and then stop as they fade.

Angle

Determines the initial angle of the particle (in degrees). This parameter is mostly useful randomized.

Scale

Determines the initial scale of the particles.

Color

Used to change the color of the particles being emitted.

Hue variation

The variation value sets the initial hue variation applied to each particle. The Variation rand value controls the hue variation randomness ratio.

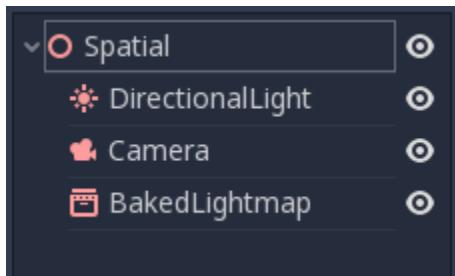
7.1 Introduction to 3D

Creating a 3D game can be challenging. That extra Z coordinate makes many of the common techniques that helped to make 2D games simple no longer work. To aid in this transition, it is worth mentioning that Godot uses very similar APIs for 2D and 3D. Most nodes are the same and are present in both 2D and 3D versions. In fact, it is worth checking the 3D platformer tutorial, or the 3D kinematic character tutorials, which are almost identical to their 2D counterparts.

In 3D, math is a little more complex than in 2D, so also checking the *Vector math* in the wiki (which were especially created for game developers, not mathematicians or engineers) will help pave the way into efficiently developing 3D games.

7.1.1 Spatial node

Node2D is the base node for 2D. *Control* is the base node for everything GUI. Following this reasoning, the 3D engine uses the *Spatial* node for everything 3D.



Spatial nodes have a local transform, which is relative to the parent node (as long as the parent node is also **or inherits** of type Spatial). This transform can be accessed as a 4x3 *Transform*, or as 3 *Vector3* members representing location, Euler rotation (x,y and z angles) and scale.



7.1.2 3D content

Unlike 2D, where loading image content and drawing is straightforward, 3D is a little more difficult. The content needs to be created with special 3D tool (usually referred to as DCCs) and exported to an exchange file format in order to be imported in Godot (3D formats are not as standardized as images).

DCC-created models

There are two pipelines to import 3D models in Godot. The first and most common one is through the *Importing 3D Scenes* importer, which allows to import entire scenes (just as they look in the DCC), including animation, skeletal rigs, blend shapes, etc.

The second pipeline is through the `doc_importing_3d_meshes` importer. This second method allows importing simple .OBJ files as mesh resources, which can be then put inside a *MeshInstance* node for display.

Generated geometry

It is possible to create custom geometry by using the *Mesh* resource directly, simply create your arrays and use the `Mesh.add_surface()` function. A helper class is also available, *SurfaceTool*, which provides a more straightforward API and helpers for indexing, generating normals, tangents, etc.

In any case, this method is meant for generating static geometry (models that will not be updated often), as creating vertex arrays and submitting them to the 3D API has a significant performance cost.

Immediate geometry

If, instead, there is a requirement to generate simple geometry that will be updated often, Godot provides a special node, *ImmediateGeometry* which provides an OpenGL 1.x style immediate-mode API to create points, lines, triangles, etc.

2D in 3D

While Godot packs a powerful 2D engine, many types of games use 2D in a 3D environment. By using a fixed camera (either orthogonal or perspective) that does not rotate, nodes such as *Sprite3D* and *AnimatedSprite3D* can be used to

create 2D games that take advantage of mixing with 3D backgrounds, more realistic parallax, lighting/shadow effects, etc.

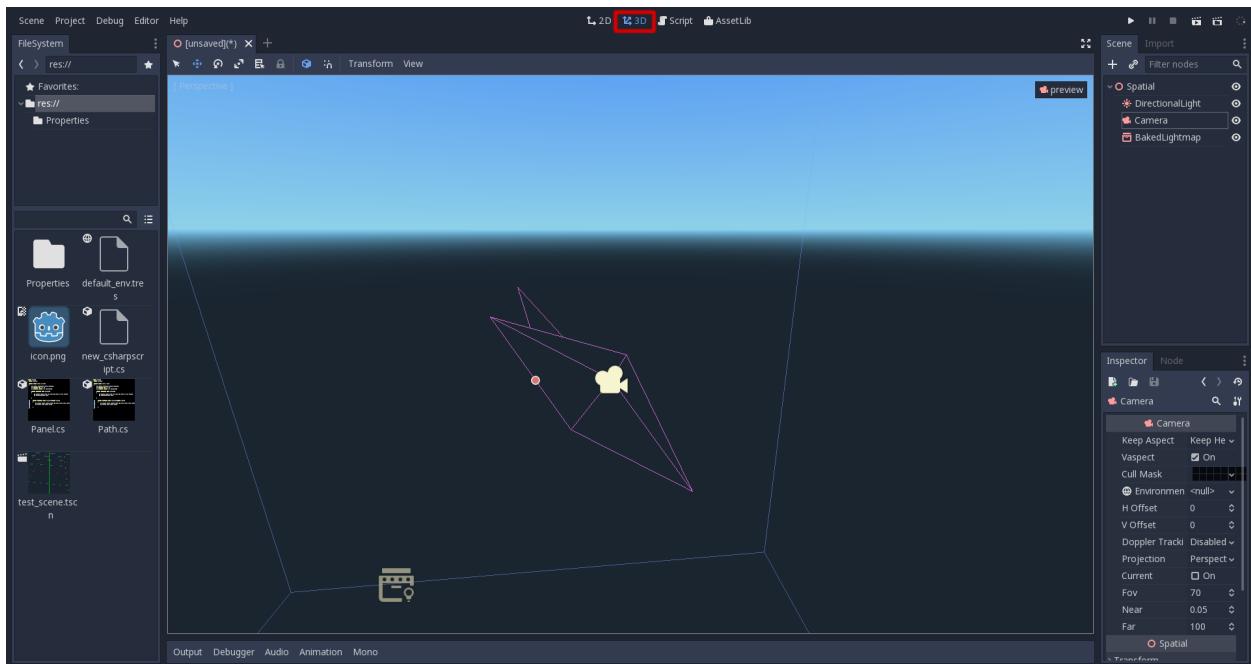
The disadvantage is, of course, that added complexity and reduced performance in comparison to plain 2D, as well as the lack of reference of working in pixels.

7.1.3 Environment

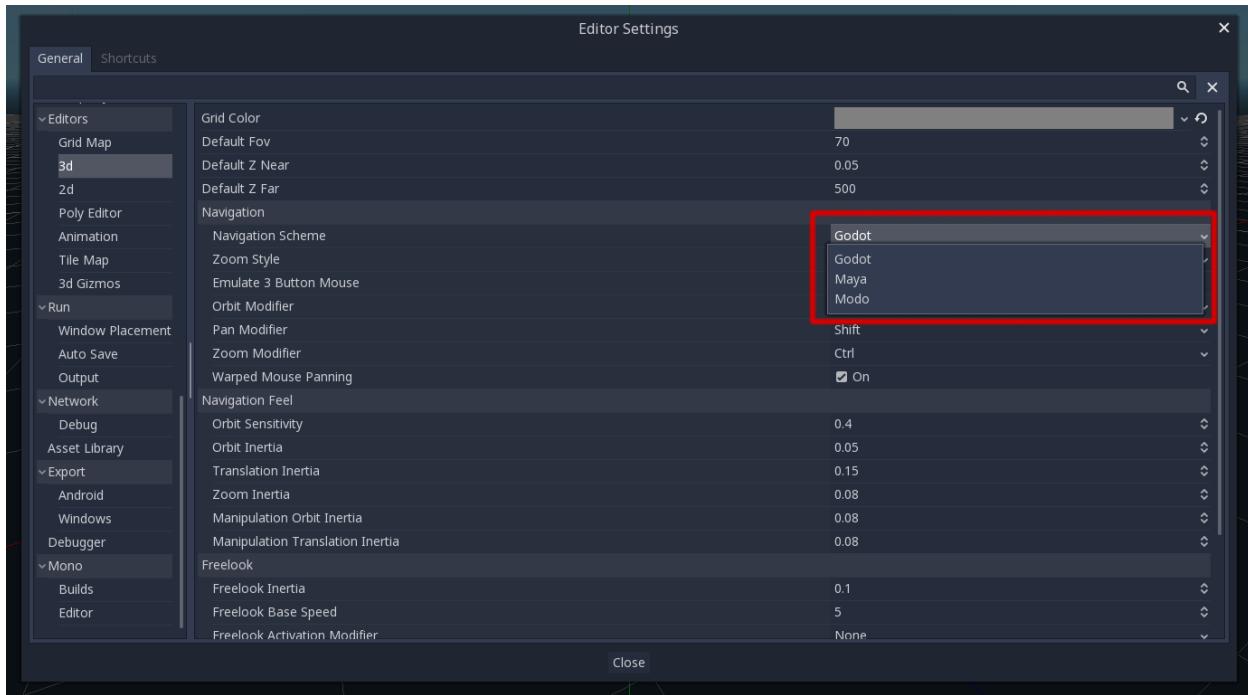
Besides editing a scene, it is often common to edit the environment. Godot provides a *WorldEnvironment* node that allows changing the background color, mode (as in, put a skybox), and applying several types of built-in post-processing effects. Environments can also be overridden in the Camera.

7.1.4 3D viewport

Editing 3D scenes is done in the 3D tab. This tab can be selected manually, but it will be automatically enabled when a Spatial node is selected.



Default 3D scene navigation controls are similar to Blender (aiming to have some sort of consistency in the free software pipeline..), but options are included to customize mouse buttons and behavior to be similar to other tools in Editor Settings:



Coordinate system

Godot uses the [metric](#) system for everything. 3D Physics and other areas are tuned for this, so attempting to use a different scale is usually a bad idea (unless you know what you are doing).

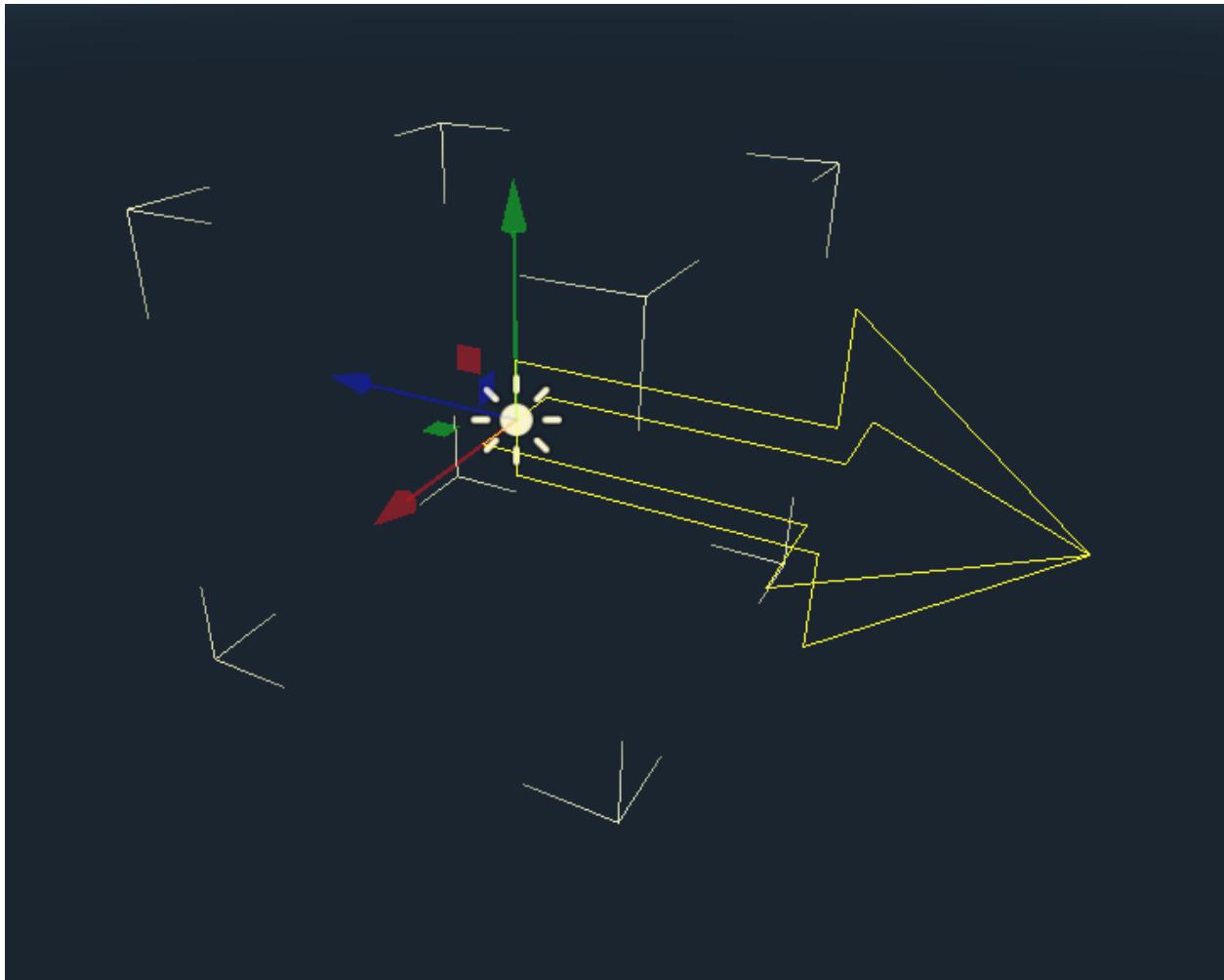
When working with 3D assets, it's always best to work in the correct scale (set your DCC to metric). Godot allows scaling post-import and, while this works in most cases, in rare situations it may introduce floating point precision issues (and thus, glitches or artifacts) in delicate areas such as rendering or physics. So, make sure your artists always work in the right scale!

The Y coordinate is used for “up”, though for most objects that need alignment (like lights, cameras, capsule collider, vehicle, etc.), the Z axis is used as a “pointing towards” direction. This convention roughly means that:

- **X** is sides
- **Y** is up/down
- **Z** is front/back

Space and manipulation gizmos

Moving objects in the 3D view is done through the manipulator gizmos. Each axis is represented by a color: Red, Green, Blue represent X,Y,Z respectively. This convention applies to the grid and other gizmos too (and also to the shader language, ordering of components for Vector3,Color,etc.).

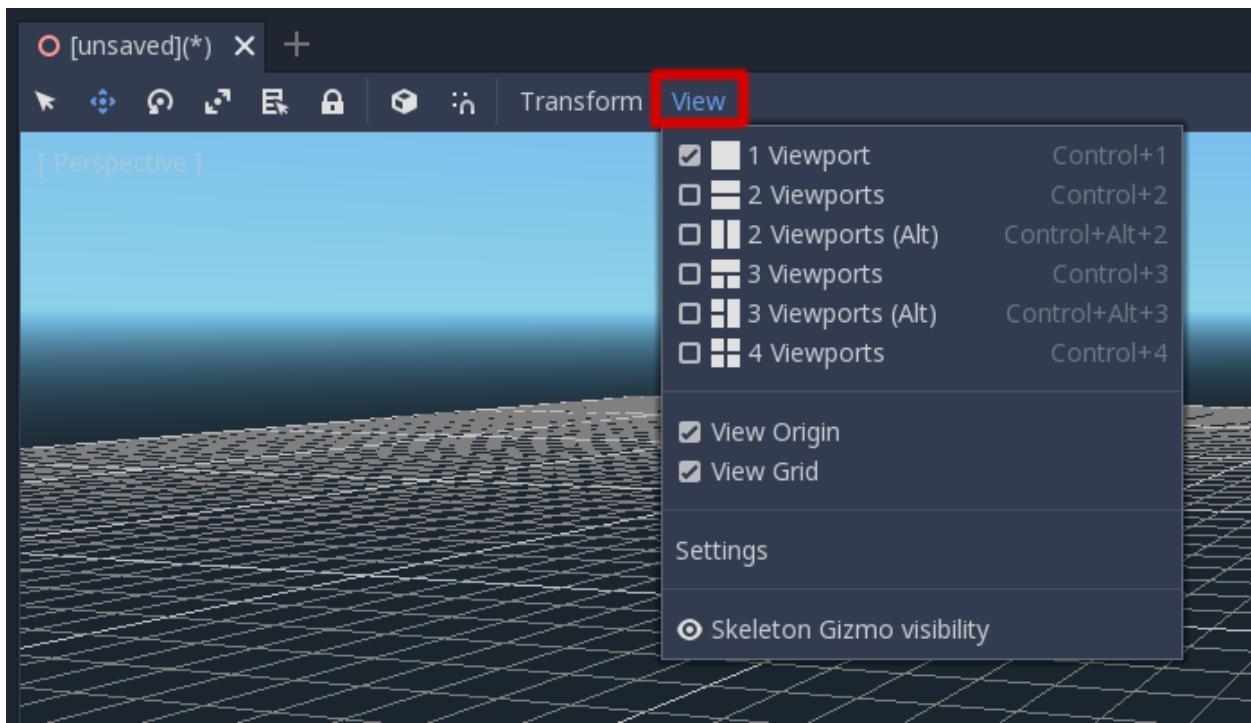


Some useful keybindings:

- To snap motion or rotation, press the “s” key while moving, scaling or rotating.
- To center the view on the selected object, press the “f” key.

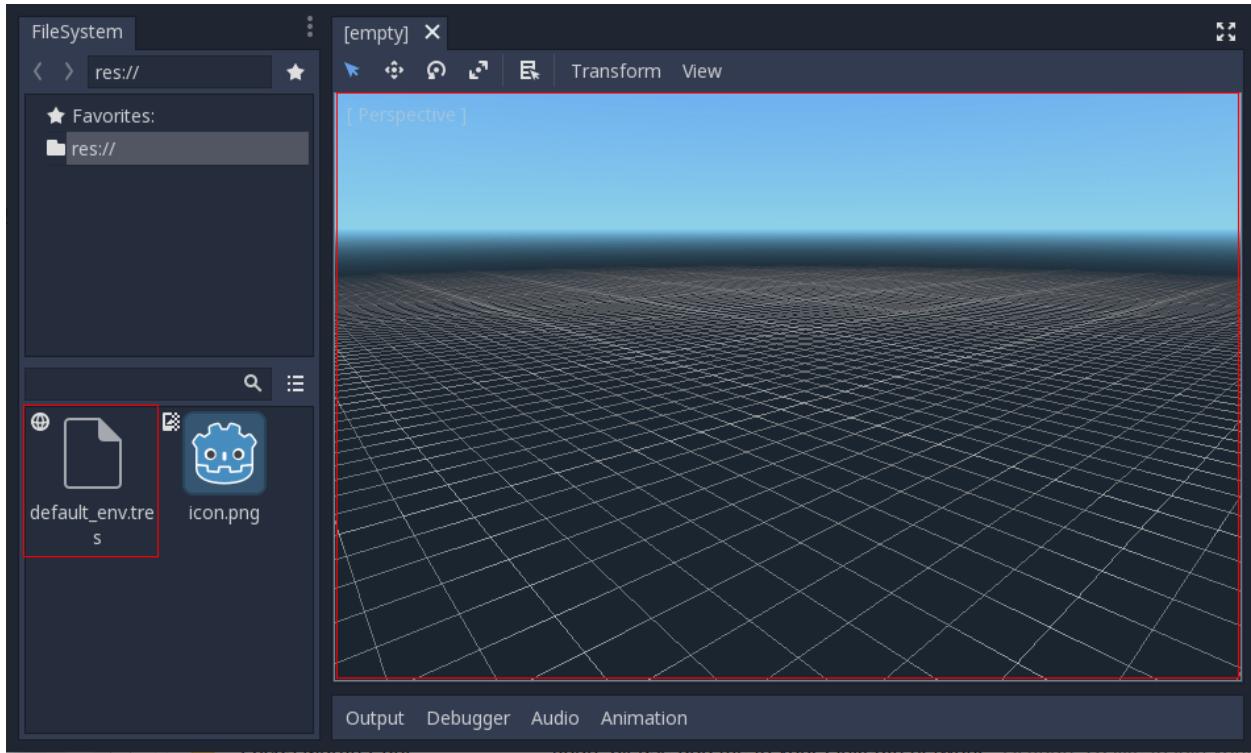
View menu

The view options are controlled by the “[view]” menu. Pay attention to this little menu inside the window because it is often overlooked!



Default environment

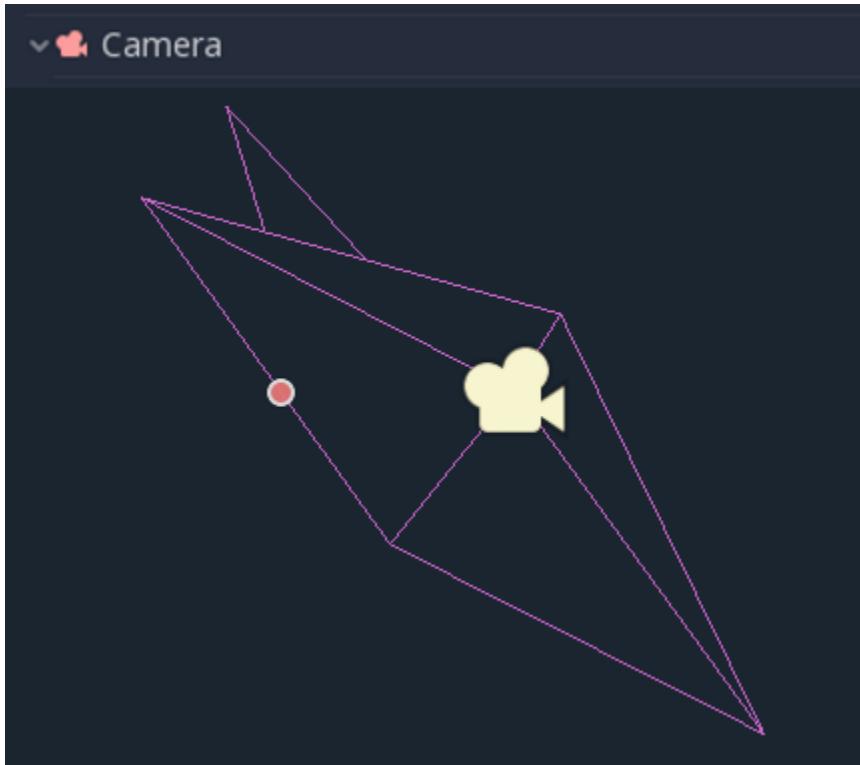
When created from the Project Manager, the 3D environment has a default sky.



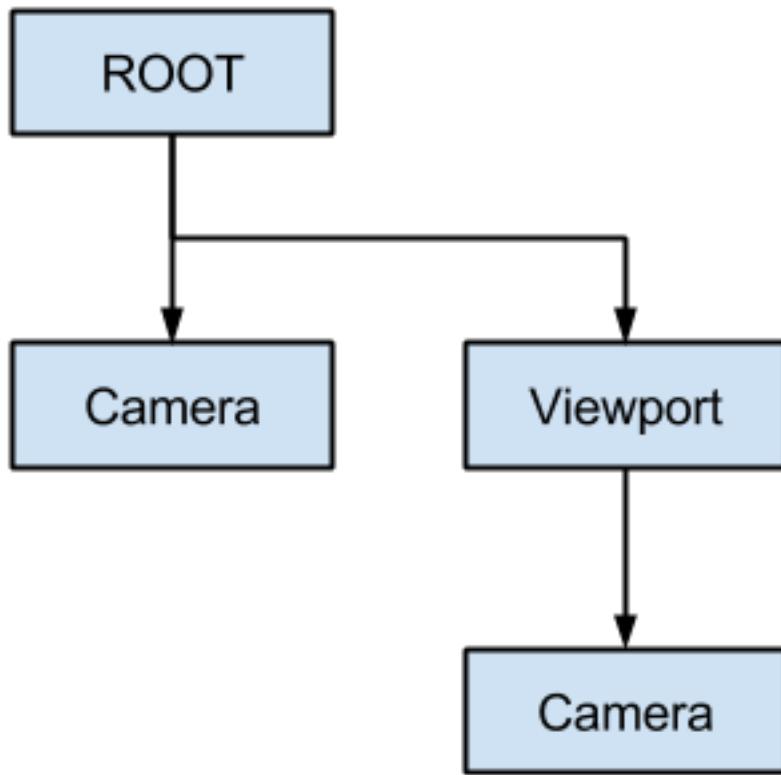
Given how physically based rendering works, it is advised to always try to work with a default environment in order to provide indirect and reflected light to your objects.

Cameras

No matter how many objects are placed in 3D space, nothing will be displayed unless a *Camera* is also added to the scene. Cameras can either work in orthogonal or perspective projections:



Cameras are associated and only display to a parent or grand-parent viewport. Since the root of the scene tree is a viewport, cameras will display on it by default, but if sub-viewports (either as render target or picture-in-picture) are desired, they need their own children cameras to display.



When dealing with multiple cameras, the following rules are followed for each viewport:

- If no cameras are present in the scene tree, the first one that enters it will become the active camera. Further cameras entering the scene will be ignored (unless they are set as *current*).
- If a camera has the “*current*” property set, it will be used regardless of any other camera in the scene. If the property is set, it will become active, replacing the previous camera.
- If an active camera leaves the scene tree, the first camera in tree-order will take its place.

Lights

There is no limitation on the number of lights nor of types of lights in Godot. As many as desired can be added (as long as performance allows).

7.2 3D performance and limitations

7.2.1 Introduction

Godot follows a balanced performance philosophy. In performance world, there are always trade-offs, which consist in trading speed for usability and flexibility. Some practical examples of this are:

- Rendering objects efficiently in high amounts is easy, but when a large scene must be rendered it can become inefficient. To solve this, visibility computation must be added to the rendering, which makes rendering less efficient, but at the same time less objects are rendered, so efficiency overall improves.

- Configuring the properties of every material for every object that needs to be renderer is also slow. To solve this, objects are sorted by material to reduce the costs, but at the same time sorting has a cost.
- In 3D physics a similar situation happens. The best algorithms to handle large amounts of physics objects (such as SAP) are very slow at insertion/removal of objects and ray-casting. Algorithms that allow faster insertion and removal, as well as ray-casting will not be able to handle as many active objects.

And there are many more examples of this! Game engines strive to be general purpose in nature, so balanced algorithms are always favored over algorithms that might be the fast in some situations and slow in others.. or algorithms that are fast but make usability more difficult.

Godot is not an exception and, while it is designed to have backends swappable for different algorithms, the default ones (or more like, the only ones that are there for now) prioritize balance and flexibility over performance.

With this clear, the aim of this tutorial is to explain how to get the maximum performance out of Godot.

7.2.2 Rendering

3D rendering is one of the most difficult areas to get performance from, so this section will have a list of tips.

Reuse shaders and materials

Godot renderer is a little different to what is out there. It's designed to minimize GPU state changes as much as possible. *SpatialMaterial* does a good job at reusing materials that need similar shaders but, if custom shaders are used, make sure to reuse them as much as possible. Godot's priorities will be like this:

- **Reusing Materials:** The less amount of different materials in the scene, the faster the rendering will be. If a scene has a huge amount of objects (in the hundreds or thousands) try reusing the materials or in the worst case use atlases.
- **Reusing Shaders:** If materials can't be reused, at least try to re-use shaders (or SpatialMaterials with different parameters but same configuration).

If a scene has, for example, 20.000 objects with 20.000 different materials each, rendering will be really slow. If the same scene has 20.000 objects, but only uses 100 materials, rendering will be blazing fast.

Pixels cost vs vertex cost

It is a common thought that the lower the polygons in a model, the faster it will be rendered. This is *really* relative and depends on many factors.

On a modern PC and consoles, vertex cost is low. Very low. GPUs originally only rendered triangles, so all the vertices:

1. Had to be transformed by the CPU (including clipping).
2. Had to be sent to the GPU memory from the main RAM.

Nowadays, all this is handled inside the GPU, so the performance is extremely high. 3D artists usually have the wrong feeling about polycount performance because 3D DCCs (such as Blender, Max, etc.) need to keep geometry in CPU memory in order for it to be edited, reducing actual performance. Truth is, a model rendered by a 3D engine is much more optimal than how 3D DCCs display them.

On mobile devices, the story is different. PC and Console GPUs are brute-force monsters that can pull as much electricity as they need from the power grid. Mobile GPUs are limited to a tiny battery, so they need to be a lot more power efficient.

To be more efficient, mobile GPUs attempt to avoid *overdraw*. This means, the same pixel on the screen being rendered (as in, with lighting calculation, etc.) more than once. Imagine a town with several buildings, GPUs don't really know what is visible and what is hidden until they draw it. A house might be drawn and then another house in front of it (rendering happened twice for the same pixel!). PC GPUs normally don't care much about this and just throw more pixel processors to the hardware to increase performance (but this also increases power consumption).

On mobile, pulling more power is not an option, so a technique called “Tile Based Rendering” is used (almost every mobile hardware uses a variant of it), which divide the screen into a grid. Each cell keeps the list of triangles drawn to it and sorts them by depth to minimize *overdraw*. This technique improves performance and reduces power consumption, but takes a toll on vertex performance. As a result, less vertices and triangles can be processed for drawing.

Generally, this is not so bad, but there is a corner case on mobile that must be avoided, which is to have small objects with a lot of geometry within a small portion of the screen. This forces mobile GPUs to put a lot of strain on a single screen cell, considerably decreasing performance (as all the other cells must wait for it to complete in order to display the frame).

To make it short, do not worry about vertex count so much on mobile, but avoid concentration of vertices in small parts of the screen. If, for example, a character, NPC, vehicle, etc. is far away (so it looks tiny), use a smaller level of detail (LOD) model instead.

An extra situation where vertex cost must be considered is objects that have extra processing per vertex, such as:

- Skinning (skeletal animation)
- Morphs (shape keys)
- Vertex Lit Objects (common on mobile)

Texture compression

Godot offers to compress textures of 3D models when imported (VRAM compression). Video RAM compression is not as efficient in size as PNG or JPG when stored, but increase performance enormously when drawing.

This is because the main goal of texture compression is bandwidth reduction between memory and the GPU.

In 3D, the shapes of objects depend more on the geometry than the texture, so compression is generally not noticeable. In 2D, compression depends more on shapes inside the textures, so the artifacting resulting from the compression is more noticeable.

As a warning, most Android devices do not support texture compression of textures with transparency (only opaque), so keep this in mind.

Transparent objects

As mentioned before, Godot sorts objects by material and shader to improve performance. This, however, can not be done on transparent objects. Transparent objects are rendered from back to front to make blending with what is behind work. As a result, please try to keep transparent objects to a minimum! If an object has a small section with transparency, try to make that section a separate material.

Level of detail (LOD)

As also mentioned before, using objects with less vertices can improve performance in some cases. Godot has a very simple system to use level of detail, *GeometryInstance* based objects have a visibility range that can be defined. Having several *GeometryInstance* objects in different ranges works as LOD.

Use instancing (*MultiMesh*)

If several identical objects have to be drawn in the same place or nearby, try using *MultiMesh* instead. *MultiMesh* allows drawing of dozens of thousands of objects at very little performance cost, making it ideal for flocks, grass, particles, etc.

Bake lighting

Small lights are usually not a performance issue. Shadows a little more. In general, if several lights need to affect a scene, it's ideal to bake it (*Baked Lightmaps*). Baking can also improve the scene quality by adding indirect light bounces.

If working on mobile, baking to texture is recommended, since this method is even faster.

7.3 Spatial Material

7.3.1 Introduction

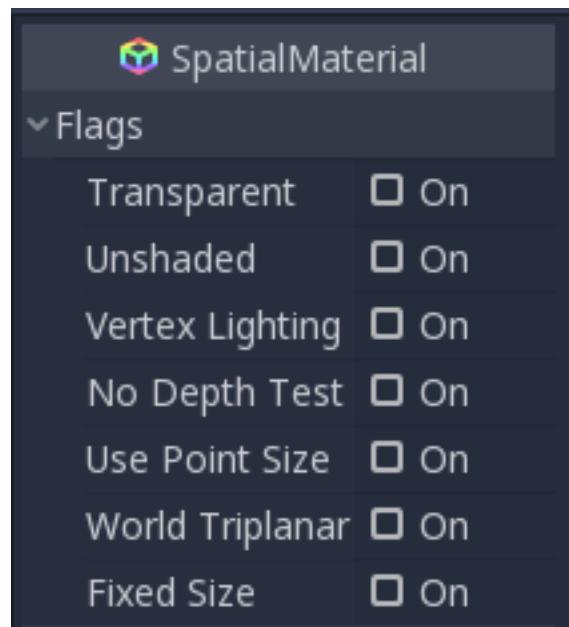
For Godot 3, instead of following the trend and focusing on shader graphs, we put in most of the work offering a default material that covers far more use cases. This replaces the old “FixedMaterial” in Godot 2.x

SpatialMaterial is a 3D material and aims to have most features artists look for in a material. Additionally, it can be converted to shader code and be further modified if desired.

This tutorial will attempt to cover most parameters present in SpatialMaterial.

7.3.2 Flags

Spatial materials have many flags determining the general usage of a material.



Transparent

In Godot, materials are not transparent unless specifically toggled as such. The main reason behind this is that transparent materials are rendered using a different technique (sorted from back to front and rendered in order).

This technique is less efficient (many state changes happen) and makes the materials unusable with many mid and post processing effects (such as SSAO, SSR, etc) that require perfectly opaque geometry.

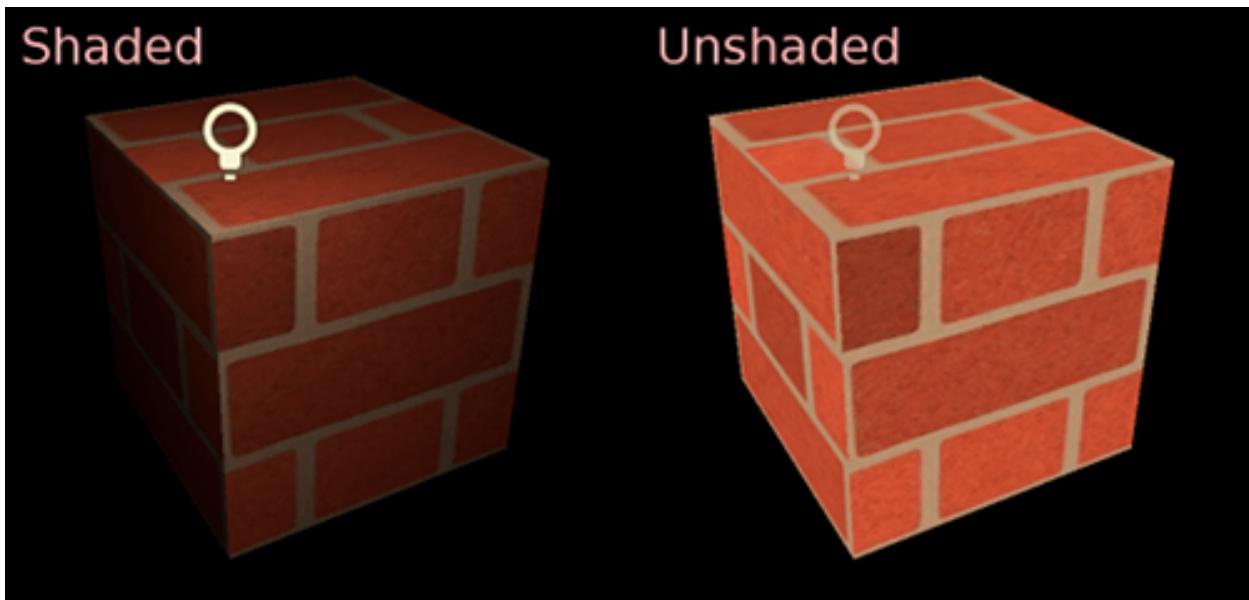
For this reason, materials in Godot are assumed opaque unless specified otherwise. The main settings that enable transparency are:

- Transparent flag (this one)
- Blend mode set to other than Mix
- Enabling distance or proximity fade

Unshaded

In most cases, it is common for materials to be affected by lighting (shaded).

Sometimes, however, one might want to show just the albedo (color) and ignore the rest. Toggling this flag on will remove all shading and show pure, unlit, color.

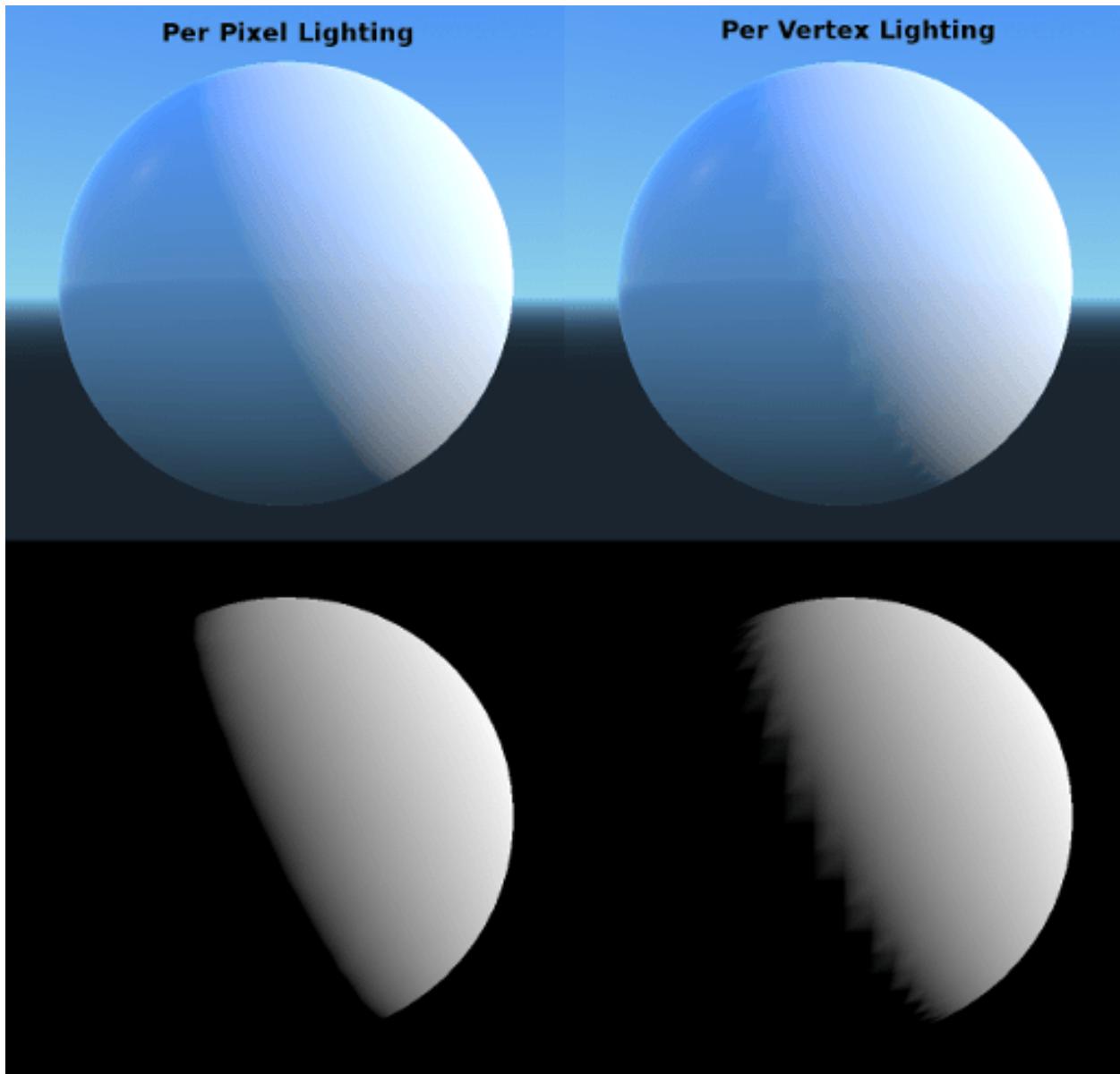


Vertex Lighting

Godot has a more or less uniform cost per pixel (thanks to depth pre pass), all lighting calculations are made by running the lighting shader on every pixel.

As these calculations are costly, performance can be brought down considerably in some corner cases such as drawing several layers of transparency (common in particle systems). Switching to per vertex lighting may help these cases.

Additionally, on very low end devices such as mobile, switching to vertex lighting can considerably increase rendering performance.

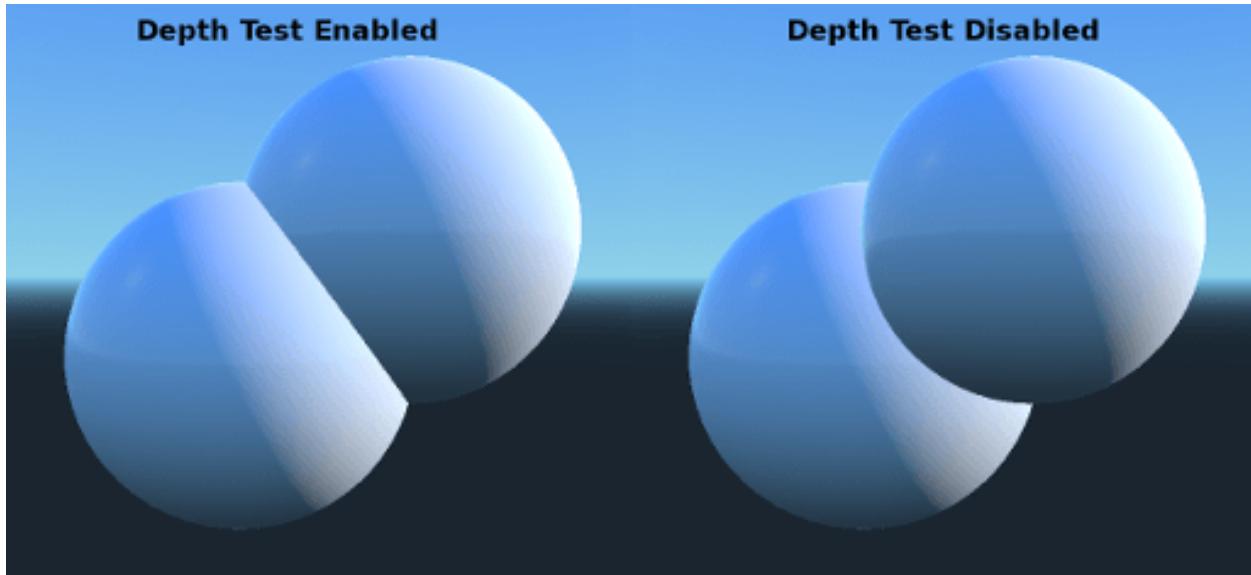


Keep in mind that, when vertex lighting is enabled, only directional lighting can produce shadows (for performance reasons).

No Depth Test

In order for close objects to appear over far away objects, depth testing is performed. Disabling it has the result of objects appearing over (or under) everything else.

Disabling this makes the most sense for drawing indicators in world space, and works very well with the “render priority” property of Material (see bottom).



Use Point Size

This option is only active when the geometry rendered is made of points (it generally is just made of triangles when imported from 3D DCCs). If so, then points can be sized (see below).

World Triplanar

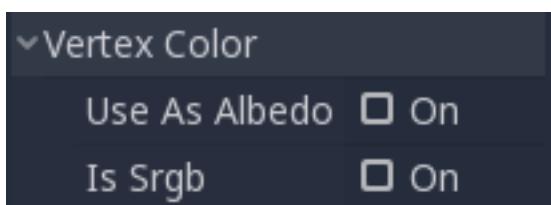
When using triplanar mapping (see below, in the UV1 and UV2 settings) triplanar is computed in object local space. This option makes triplanar work in world space.

Fixed Size

Makes the object rendered at the same size no matter the distance. This is, again, useful mostly for indicators (no depth test and high render priority) and some types of billboards.

7.3.3 Vertex Color

This menu allows choosing what is done by default to vertex colors that come from your 3D modelling application. By default, they are ignored.



Use as Albedo

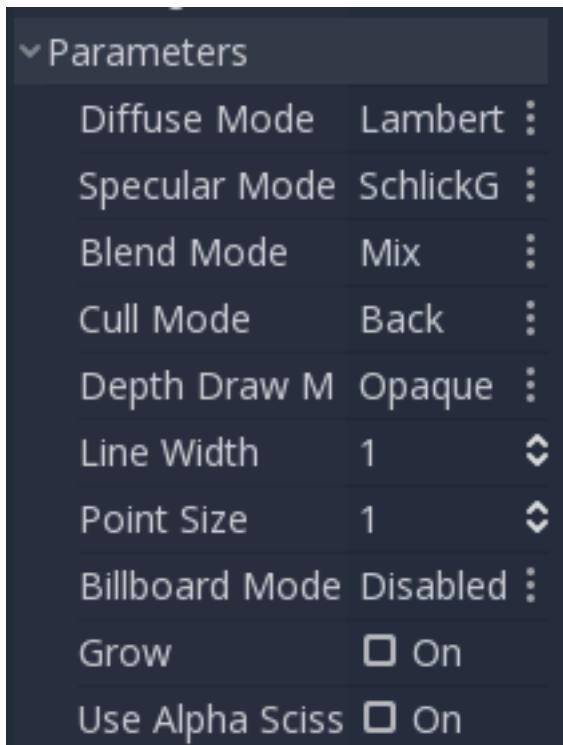
Vertex color is used as albedo color.

Is SRGB

Most 3D DCCs will likely export vertex colors as SRGB, so toggling this option on will help them look correct.

7.3.4 Parameters

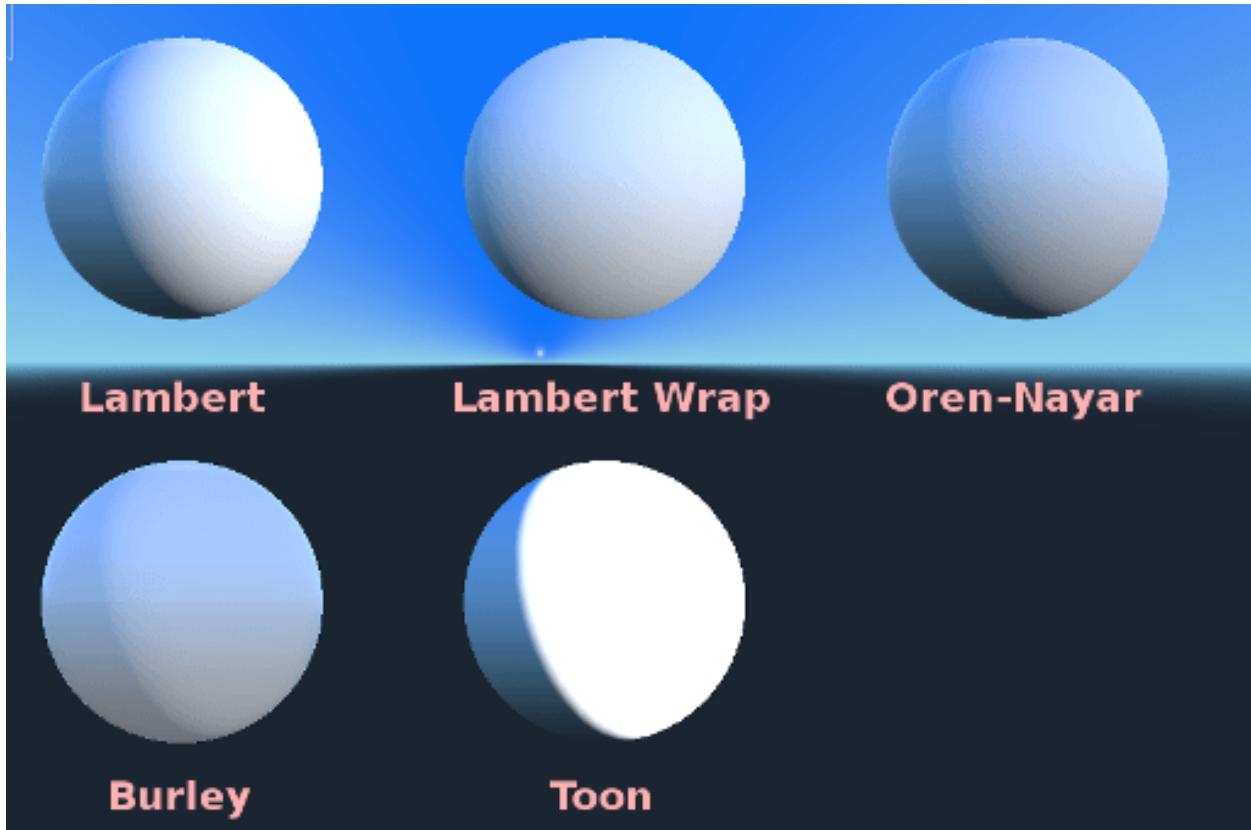
SpatialMaterial also has several configurable parameters to tweak many aspects of the rendering:



Diffuse Mode

Specifies the algorithm used by diffuse scattering of light when hitting the object. The default one is Burley. Other modes are also available:

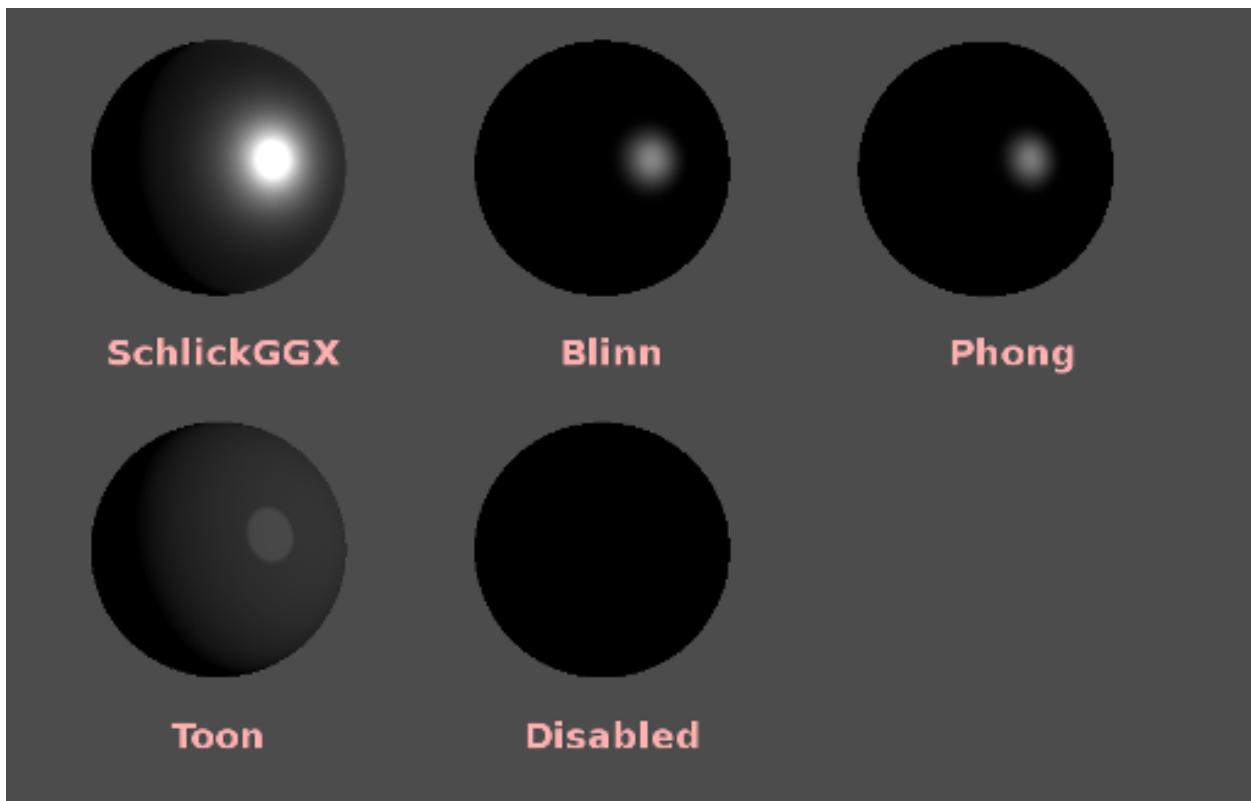
- **Burley:** Default mode, the original Disney Principled PBS diffuse algorithm.
- **Lambert:** Is not affected by roughness.
- **Lambert Wrap:** Extends Lambert to cover more than 90 degrees when roughness increases. Works great for hair and simulating cheap subsurface scattering. This implementation is energy conserving.
- **Oren Nayar:** This implementation aims to take microsurfacing into account (via roughness). Works really well for clay-like materials and some types of cloth.
- **Toon:** Provides a hard cut for lighting, with smoothing affected by roughness.



Specular Mode

Specifies how the specular blob will be rendered. The specular blob represents the shape of a light source reflected in the object.

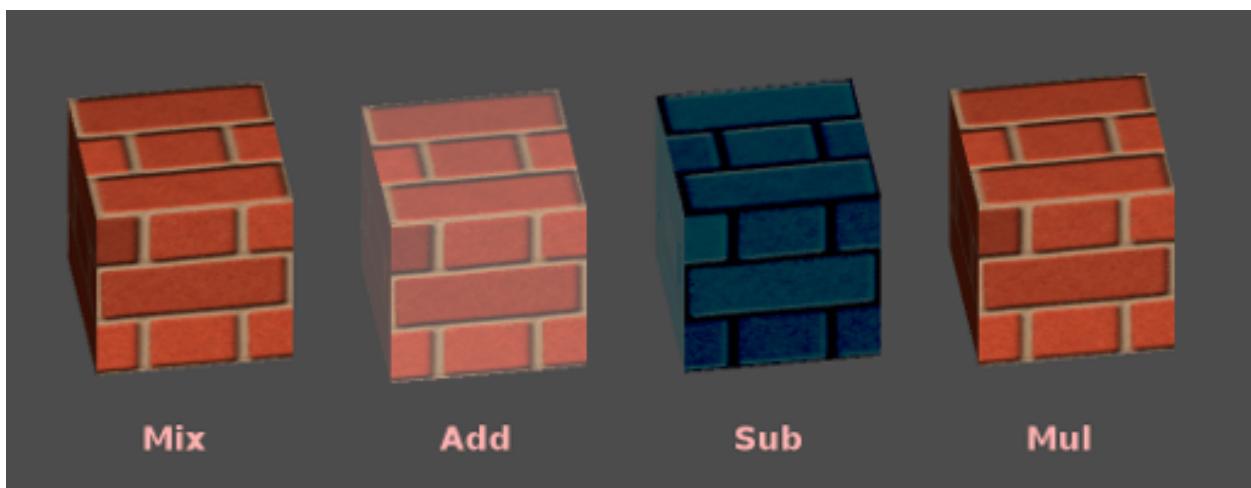
- **ShlickGGX:** The most common blob used by PBR 3D engines nowadays.
- **Blinn:** Common in previous-generation engines. Not worth using nowadays, but left here for the sake of compatibility.
- **Phong:** Same as above.
- **Toon:** Creates a toon blob, which changes size depending on roughness.
- **Disabled:** Sometimes, that blob gets in the way. Be gone!



Blend Mode

Controls the blend mode for the material. Keep in mind that any mode other than Mix forces the object to go through transparent pipeline.

- Mix: Default blend mode, alpha controls how much the object is visible.
- Add: Object is blended additively, nice for flares or some fire-like effects.
- Sub: Object is subtracted.
- Mul: Object is multiplied.



Cull Mode

Determines which side of the object is not drawn when back-faces are rendered:

- Back: Back of the object is culled when not visible (default)
- Front: Front of the object is culled when not visible
- Disabled: Used for objects that are double sided (no culling is performed)

Depth Draw Mode

Specifies when depth rendering must take place.

- Opaque Only (default): Depth is only drawn for opaque objects
- Always: Depth draw is drawn for both opaque and transparent objects
- Never: No depth draw takes place (note: do not confuse with depth test option above)
- Depth Pre-Pass: For transparent objects, an opaque pass is made first with the opaque parts, then transparency is drawn above. Use this option with transparent grass or tree foliage.

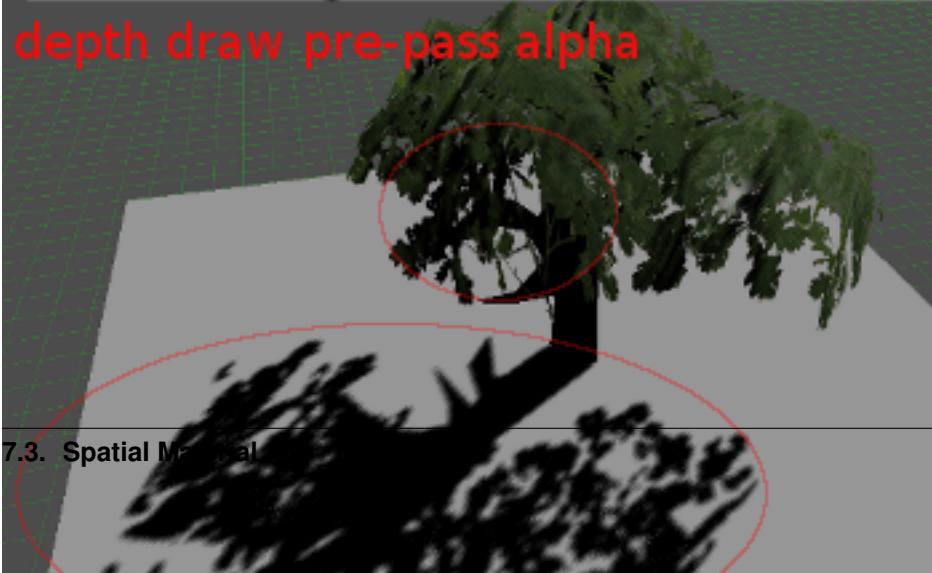
depth draw always



depth draw opaque only



depth draw pre-pass alpha



Line Width

When drawing lines, specify the width of the lines being drawn. This option is not available in most modern hardware.

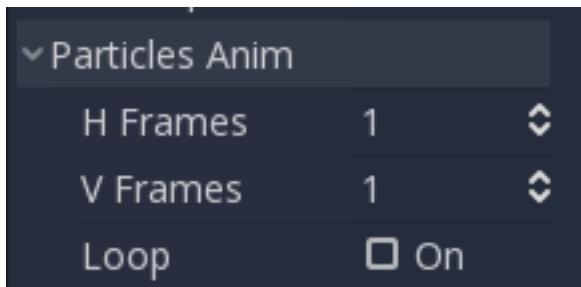
Point Size

When drawing points, specify the point size in pixels.

Billboard Mode

Enables billboard mode for drawing materials. This control how the object faces the camera:

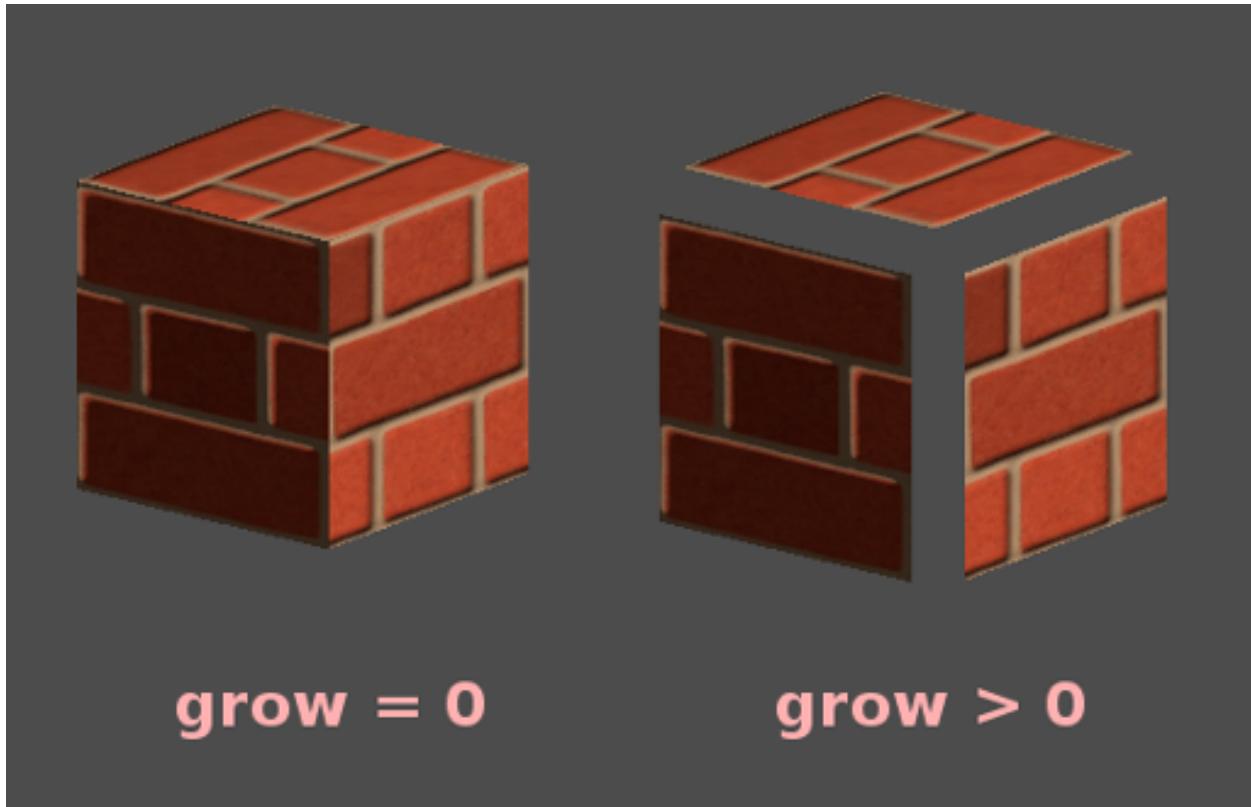
- Disabled: Billboard mode is disabled
- Enabled: Billboard mode is enabled, object -Z axis will always face the camera.
- Y-Billboard: Object X axis will always be aligned with the camera
- Particles: When using particle systems, this type of billboard is best, because it allows specifying animation options.



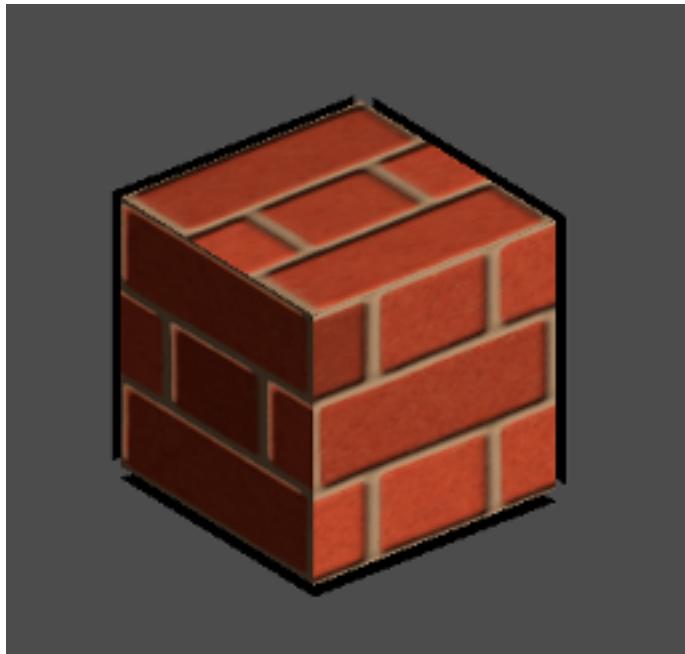
Above options are only enabled for Particle Billboard.

Grow

Grows the object vertices in the direction pointed by their normals:

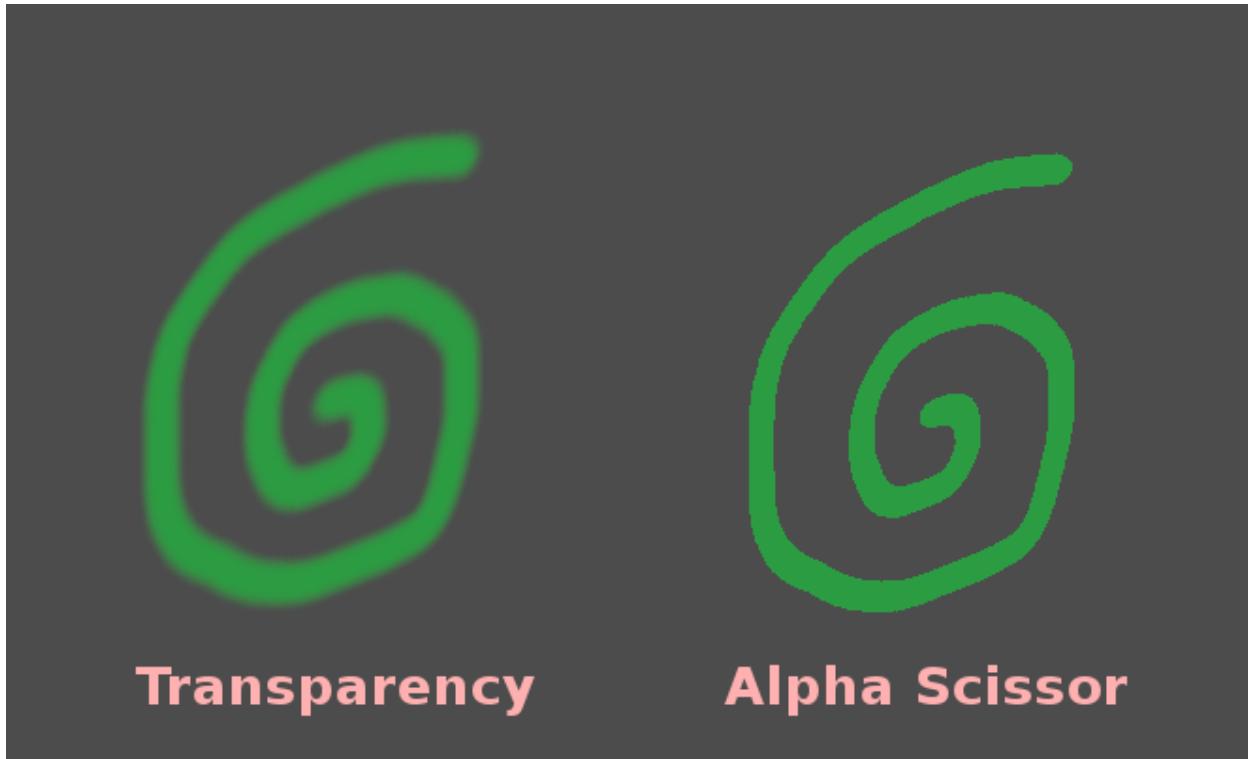


This is commonly used to create cheap outlines. Add a second material pass, make it black and unshaded, reverse culling (Cull Front), and add some grow:



Use Alpha Scissor

When transparency other than 0 or 1 is not needed, it's possible to set a threshold to avoid the object from rendering these pixels.



This renders the object via the opaque pipeline, which is faster and allows it to do mid and post process effects such as SSAO, SSR, etc.

7.3.5 Material colors, maps and channels

Besides the parameters, what defines materials themselves are the colors, textures and channels. Godot supports a very extensive list of them (arguably far more than any of the other prominent game engines). They will be described in detail below:

Albedo

Albedo is the base color for the material. Everything else works based on it. When set to *unshaded* this is the only color that is visible as-is. In previous versions of Godot, this channel was named *diffuse*. The change of name mainly happens because, in PBR rendering, this color affects many more calculations than just the diffuse lighting path.

Albedo color and texture can be used together, as they are multiplied.

Alpha channel in albedo color and texture is also used for the object transparency. If you use a color or texture with *alpha channel*, make sure to either enable transparency or *alpha scissoring* for it to work.

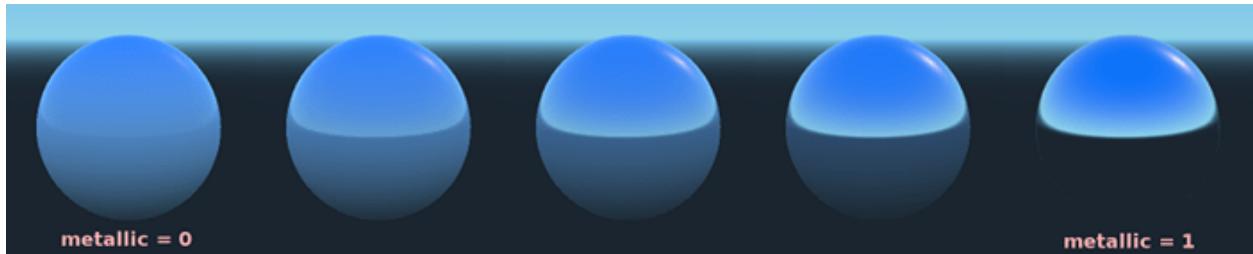
Metallic

Godot uses a Metallic model over competing models due to its simplicity. This parameter pretty much defines how reflective the materials is. The more reflective it is, the least diffuse/ambient light and the more reflected light. This

model is called “energy conserving”.

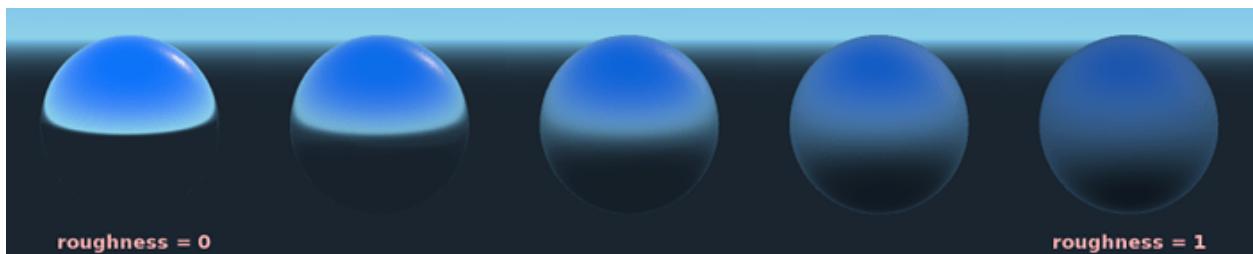
The “specular” parameter here is just a general amount of for the reflectivity (unlike *metallic*, this one is not energy conserving, so simply leave it as 0.5 and don’t touch it unless you need to).

The minimum internal reflectivity is 0.04, so (just like in real life) it’s impossible to make a material completely unreflective.



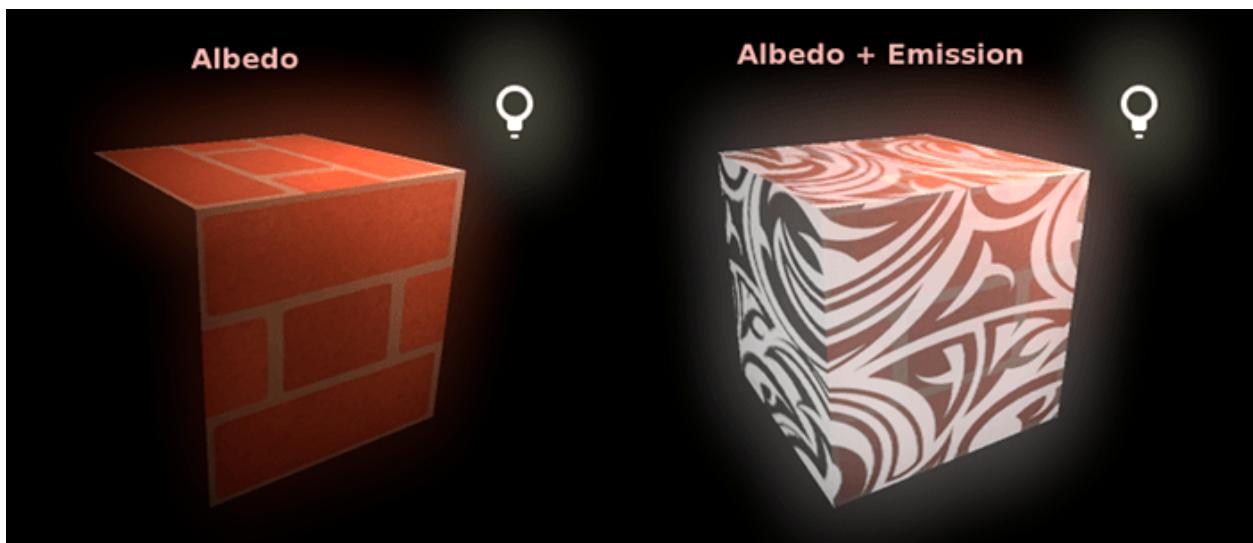
Roughness

Roughness affects mainly the way reflection happens. A value of 0 makes it a perfect mirror, while a value of 1 completely blurs the reflection (simulating the natural microsurfacing). Most common types of materials can be achieved from the right combination of *Metallic* and *Roughness*.



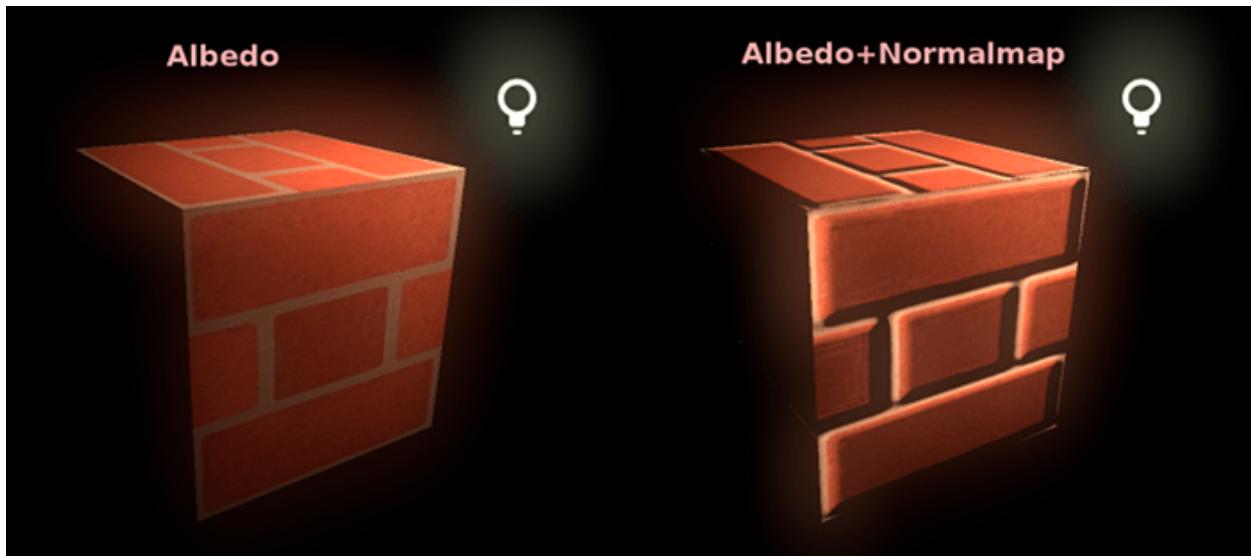
Emission

Emission specifies how much light is emitted by the material (keep in mind this does not do lighting on surrounding geometry unless GI Probe is used). This value is just added to the resulting final image, and is not affected by other lighting in the scene.



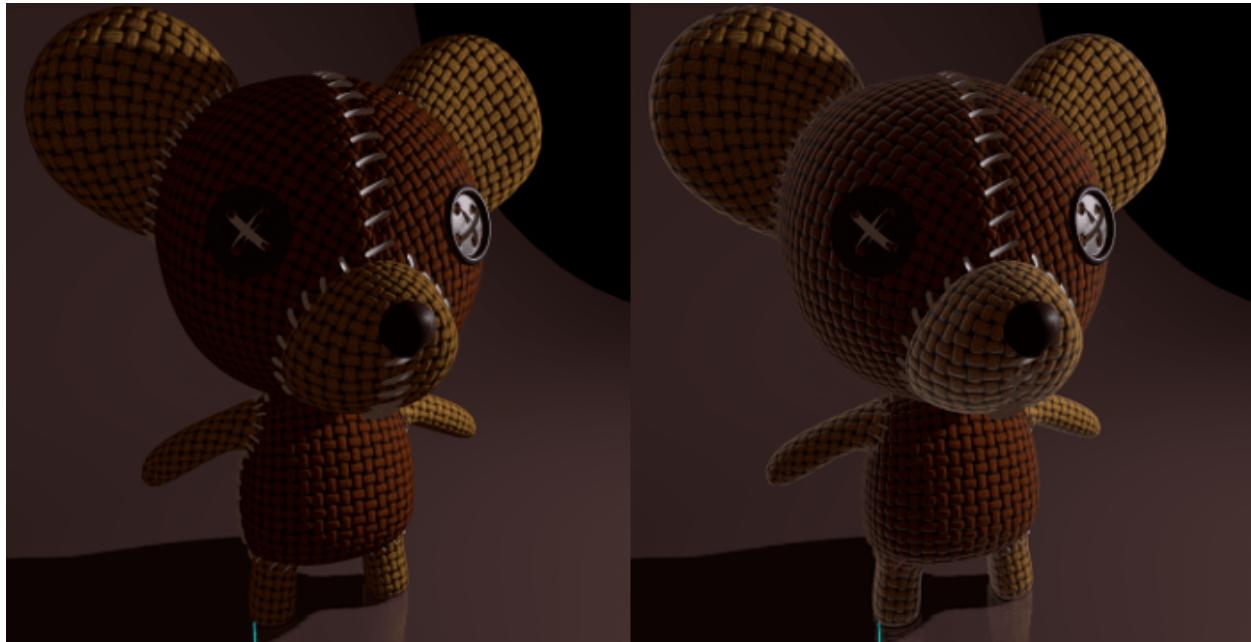
Normalmap

Normal mapping allows to set a texture that represents finer shape detail. This does not modify geometry, just the incident angle for light. In Godot, only R and G are used for normalmaps, in order to attain better compatibility.



Rim

Some fabrics have small micro fur that causes light to scatter around it. Godot emulates this with the *rim* parameter. Unlike other rim lighting implementations which just use the emission channel, this one actually takes light into account (no light means no rim). This makes the effect considerably more believable.



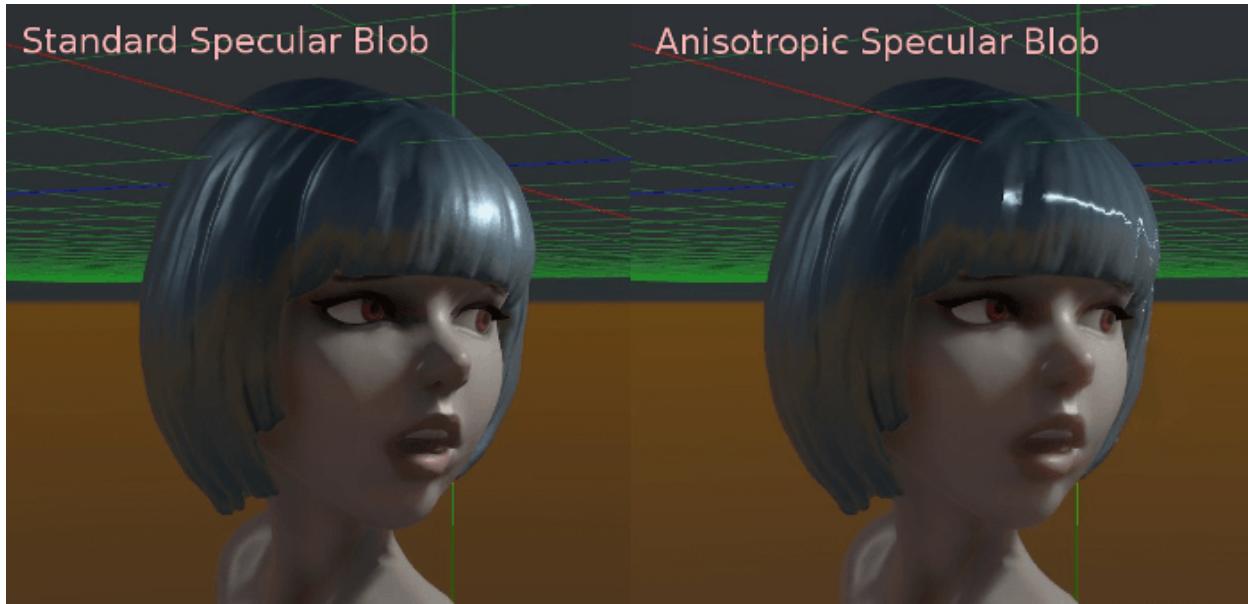
Rim size depends on roughness and there is a special parameter to specify how it must be colored. If *tint* is 0, the color of the light is used for the rim. If *tint* is 1, then the albedo of the material is used. Using intermediate values generally works best.

Clearcoat

The *clearcoat* parameter is used mostly to add a *secondary* pass of transparent coat to the material. This is very common in car paint and toys. In practice, it's a smaller specular blob added on top of the existing material.

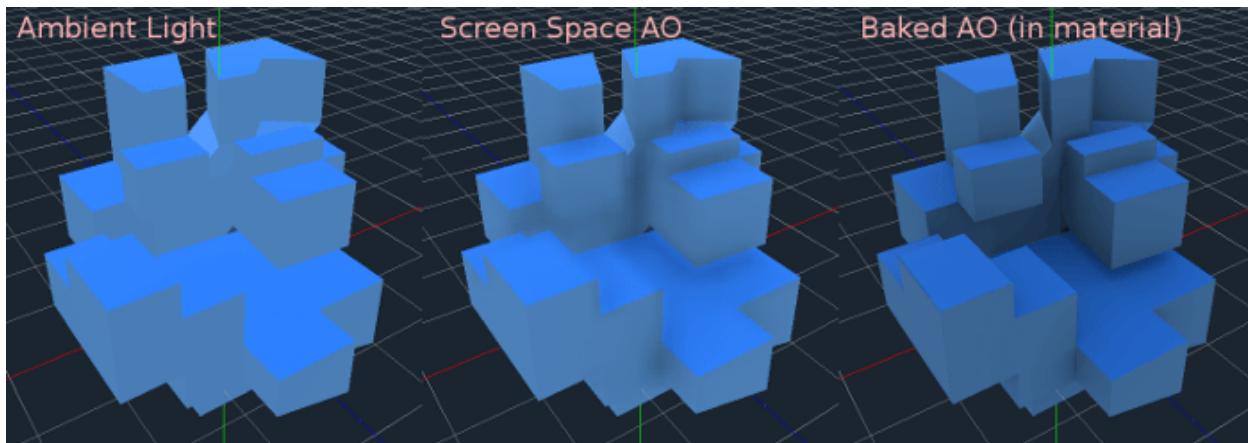
Anisotropy

Changes the shape of the specular blow and aligns it to tangent space. Anisotropy is commonly used with hair, or to make materials such as brushed alluminium more realistic. It works especially well when combined with flowmaps.



Ambient Occlusion

In Godot's new PBR workflow, it is possible to specify a pre-baked ambient occlusion map. This map affects how much ambient light reaches each surface of the object (it does not affect direct light). While it is possible to use Screen Space Ambient Occlusion (SSAO) to generate AO, nothing will beat the quality of a nicely baked AO map. It is recommended to pre-bake AO whenever possible.



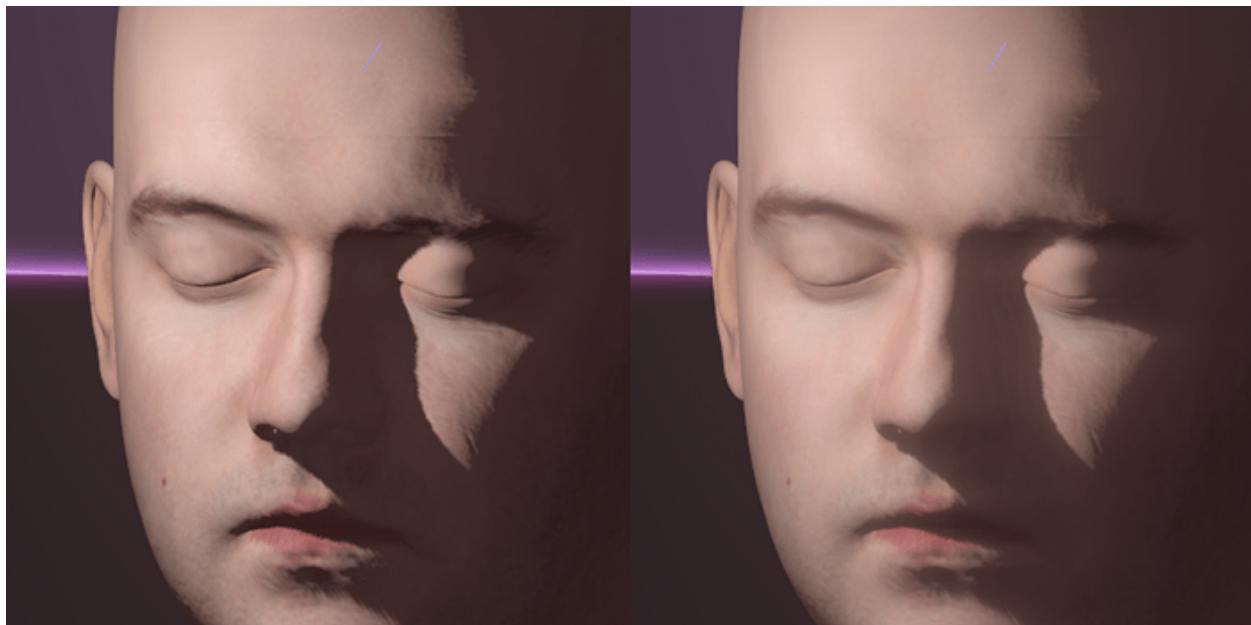
Depth

Setting a depth map to a material produces a ray-marched search to emulate the proper displacement of cavities along the view direction. This is not real added geometry, but an illusion of depth. It may not work for complex objects, but it produces a realistic depth effect for textures. For best results, *Depth* should be used together with normal mapping.



Subsurface Scattering

This effect emulates light that goes beneath an object's surface, is scattered, and then comes out. It's very useful to make realistic skin, marble, colored liquids, etc.



Transmission

Controls how much light from the lit side (visible to light) is transferred to the dark side (opposite side to light). This works very well for thin objects such as tree/plant leaves, grass, human ears, etc.



Refraction

When refraction is enabled, it supersedes alpha blending and Godot attempts to fetch information from behind the object being rendered instead. This allows distorting the transparency in a way very similar to refraction.



Detail

Godot allows using secondary albedo and normal maps to generate a detail texture, which can be blended in many ways. Combining with secondary UV or triplanar modes, many interesting textures can be achieved.



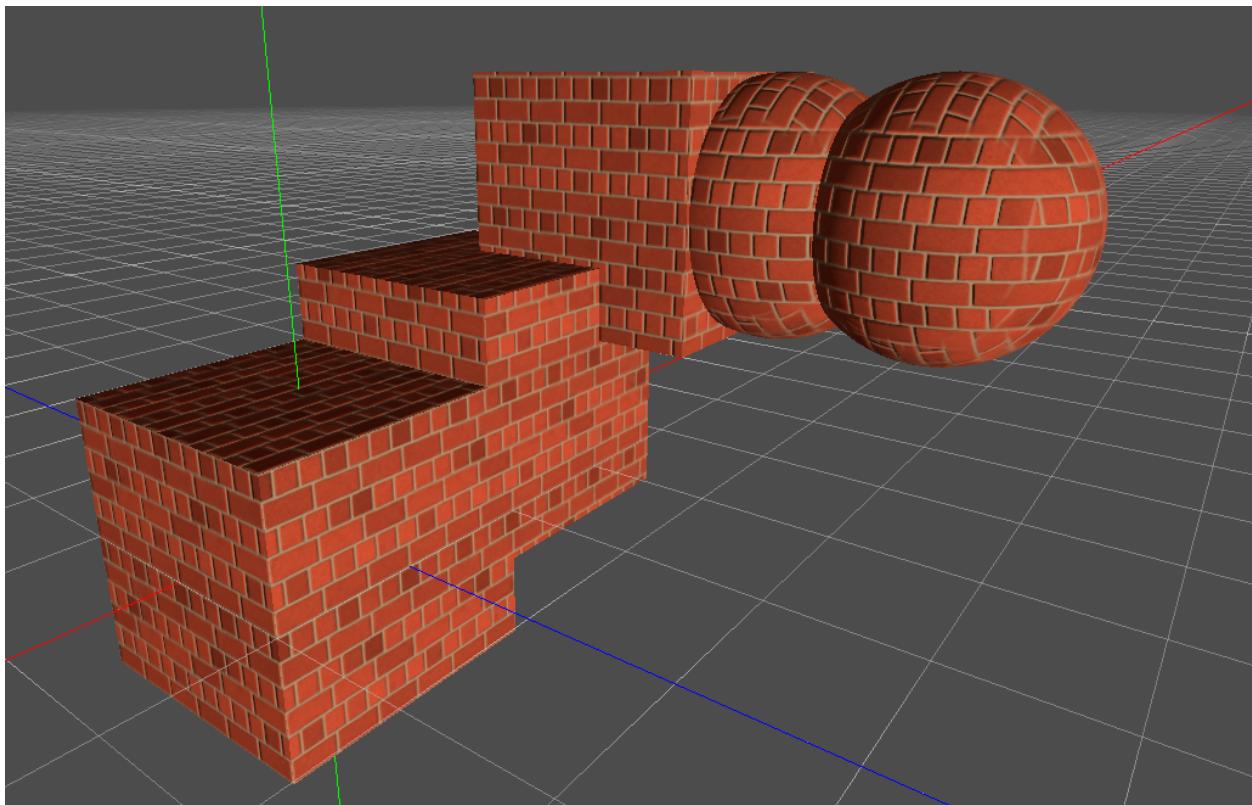
UV1 and UV2

Godot supports 2 UV channels per material. Secondary UV is often useful for AO or Emission (baked light). UVs can be scaled and offseted, which is useful in textures with repeat.

Triplanar Mapping

Trilplanar mapping is supported for both UV1 and UV2. This is an alternative way to obtain texture coordinates, often called “Autotexture”. Textures are sampled in X,Y and Z and blended by the normal. Triplanar can be either worldspace or object space.

In the image below, you can see how all primitives share the same material with world triplanar, so bricks continue smoothly between them.



7.3.6 Proximity and Distance Fade

Godot allows materials to fade by proximity to another, as well as depending on the distance to the viewer. Proximity fade is very useful for effects such as soft particles, or a mass of water with a smooth blending to the shores. Distance fade is useful for light shafts or indicators that are only present after a given distance.

Keep in mind enabling these enables alpha blending, so abusing them for a whole scene is not generally a good idea.

7.3.7 Render Priority

Rendering order can be changed for objects, although this is mostly useful for transparent objects (or opaque objects that do depth draw but no color draw, useful for cracks on the floor).

7.4 Lights And Shadows

7.4.1 Introduction

Lights emit light that mix with the materials and produces a visible result. Light can come from several types of sources in a scene:

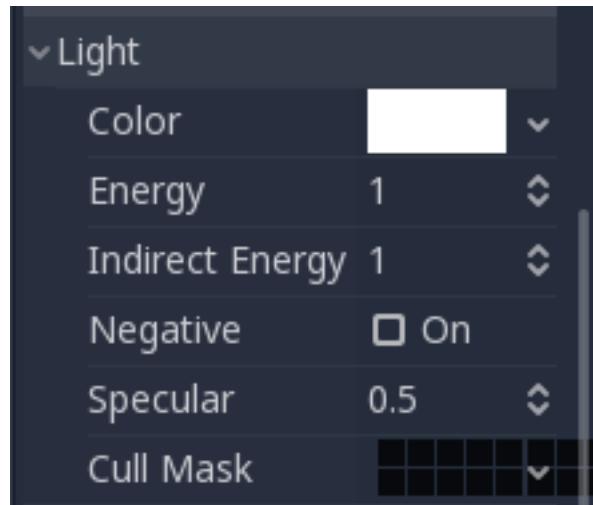
- From the Material itself, in the form of the emission color (though it does not affect nearby objects unless baked).
- Light Nodes: Directional, Omni and Spot.

- Ambient Light in the *Environment*.
- Baked Light (read *Baked Lightmaps*).

The emission color is a material property. You can read more about it in the *Spatial Material* tutorial.

7.4.2 Light nodes

As mentioned before, there are three types of light nodes: Directional, Omni and Spot. Each has different uses and will be described in detail below, but first let's take a look at the common parameters for lights:



Each one has a specific function:

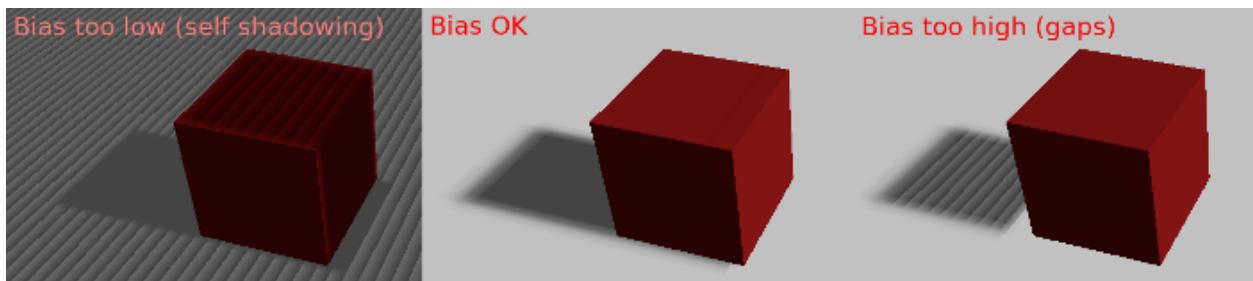
- **Color:** Base color for emitted light.
- **Energy:** Energy multiplier. This is useful to saturate lights or working with *High dynamic range*.
- **Indirect Energy:** Secondary multiplier used with indirect light (light bounces). This works in baked light or GIProbe.
- **Negative:** Light becomes subtractive instead of additive. It's sometimes useful to manually compensate some dark corners.
- **Specular:** Affects the intensity of the specular blob in objects affected by this light. At zero, this light becomes a pure diffuse light.
- **Cull Mask:** Objects that are in the selected layers below will be affected by this light.

Shadow Mapping

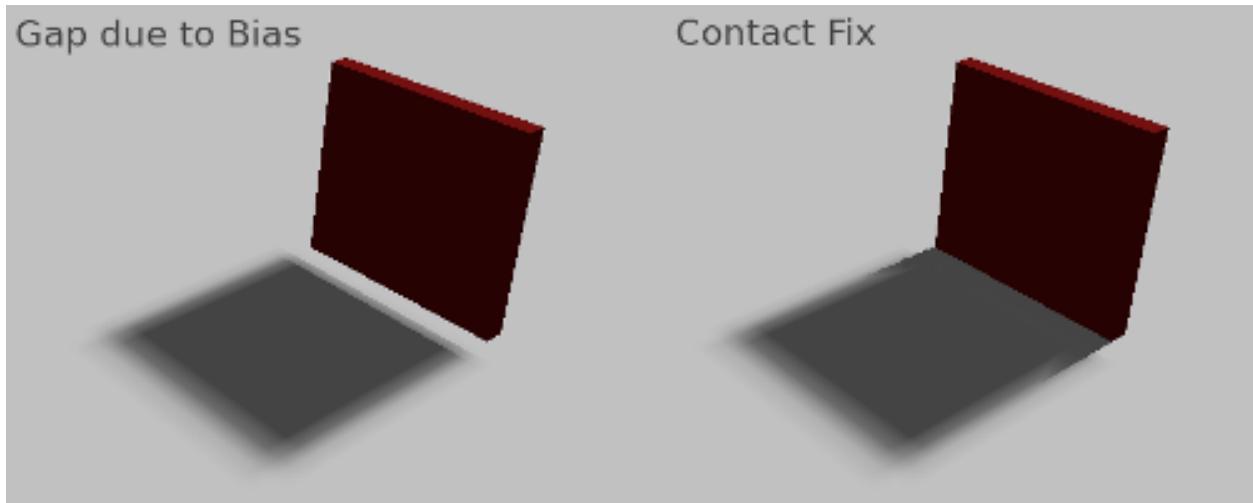
Lights can optionally cast shadows. This gives them greater realism (light does not reach occluded areas), but it can incur a bigger performance cost. There is a list of generic shadow parameters, each also has a specific function:

- **Enabled:** Check to enable shadow mapping in this light.
- **Color:** Areas occluded are multiplied by this color. It is black by default, but it can be changed to tint shadows.
- **Bias:** When this parameter is too small, self shadowing occurs. When too large, shadows separate from the casters. Tweak to what works best for you.
- **Contact:** Performs a short screen-space raycast to reduce the gap generated by the bias.
- **Reverse Cull Faces:** Some scenes work better when shadow mapping is rendered with face-culling inverted.

Below is an image of how tweaking bias looks like. Default values work for most cases, but in general it depends on the size and complexity of geometry.



Finally, if gaps can't be solved, the **Contact** option can help:

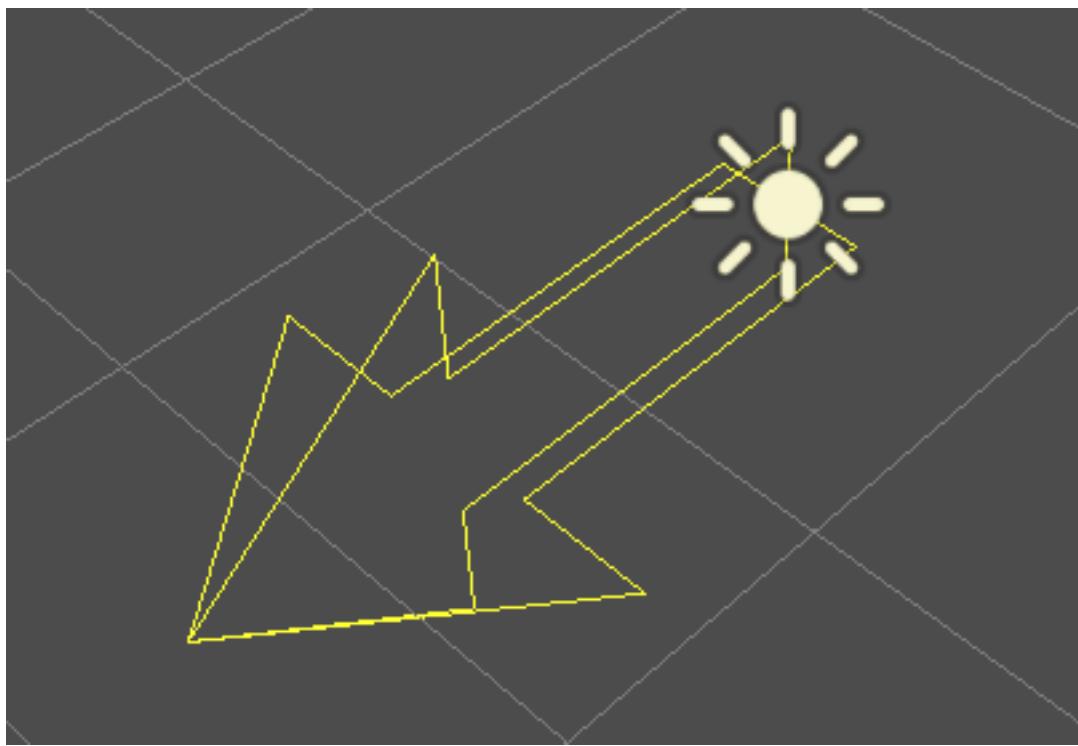


Any sort of bias issues can always be fixed by increasing the shadow map resolution, although that may lead to decreased performance on low-end hardware.

Directional light

This is the most common type of light and represents a light source very far away (such as the sun). It is also the cheapest light to compute and should be used whenever possible (although it's not the cheapest shadow-map to compute, but more on that later).

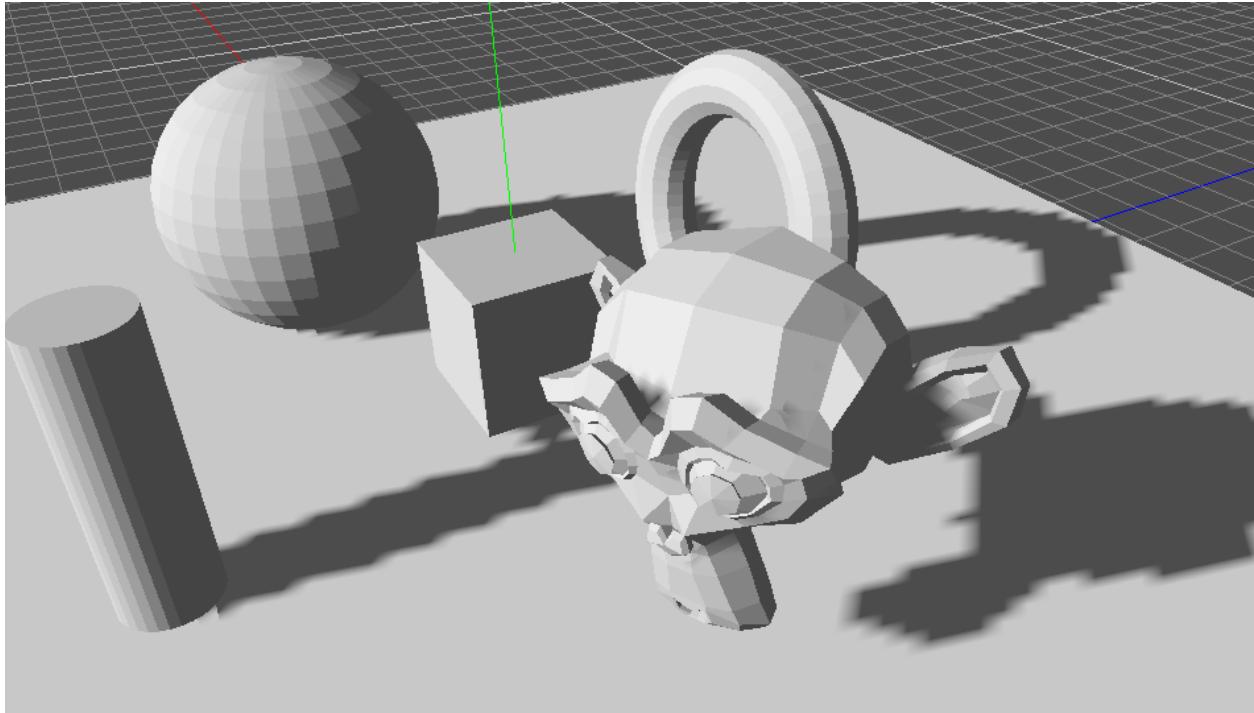
Directional light models an infinite number of parallel light rays covering the whole scene. The directional light node is represented by a big arrow, which indicates the direction of the light rays. However, the position of the node does not affect the lighting at all, and can be anywhere.



Every face whose front-side is hit by the light rays is lit, the others stay dark. Most light types have specific parameters but directional lights are pretty simple in nature so they don't.

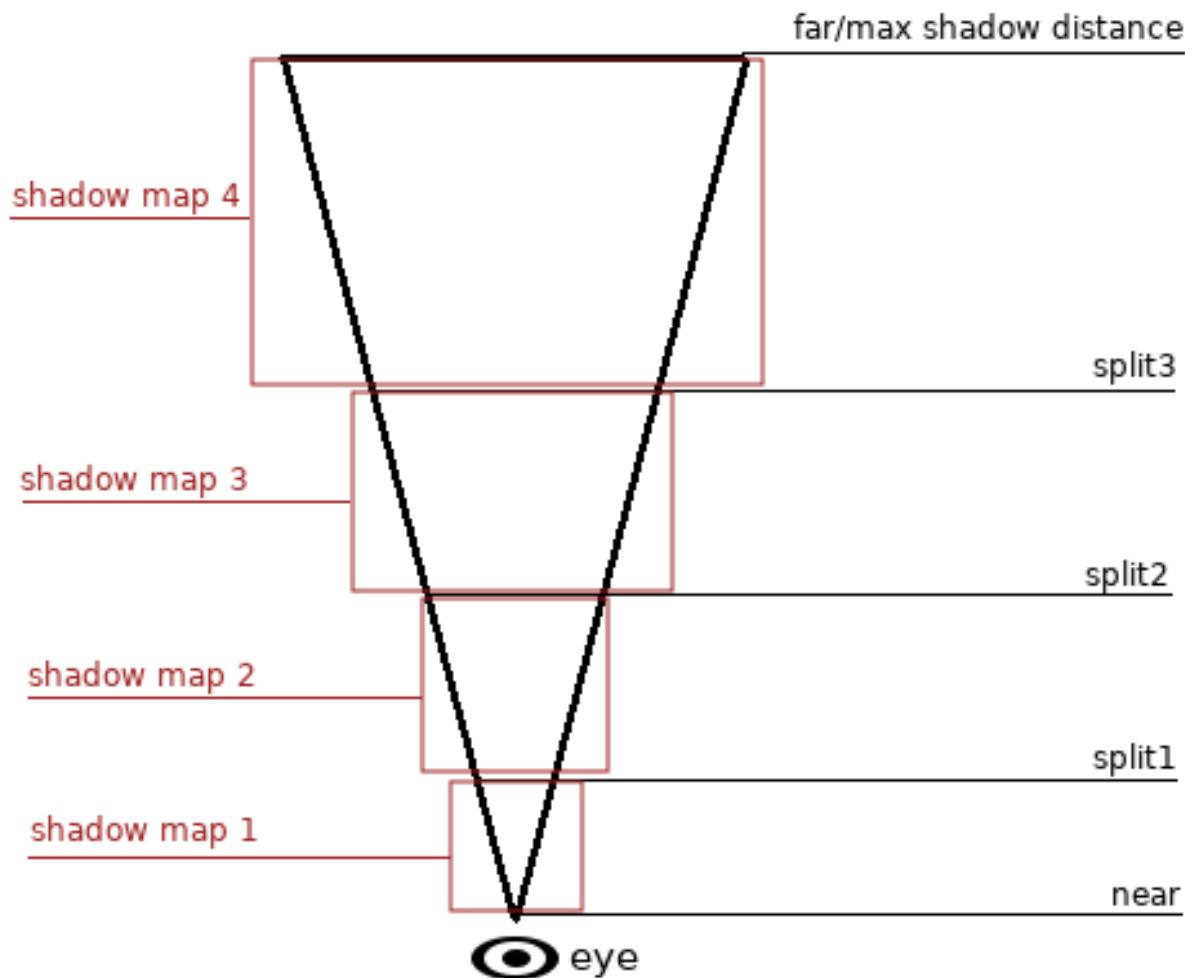
Directional Shadow Mapping

To compute shadow maps, the scene is rendered (only depth) from an orthogonal point of view that covers the whole scene (or up to the max distance). There is, however, a problem with this approach because objects closer to the camera receive blocky shadows.

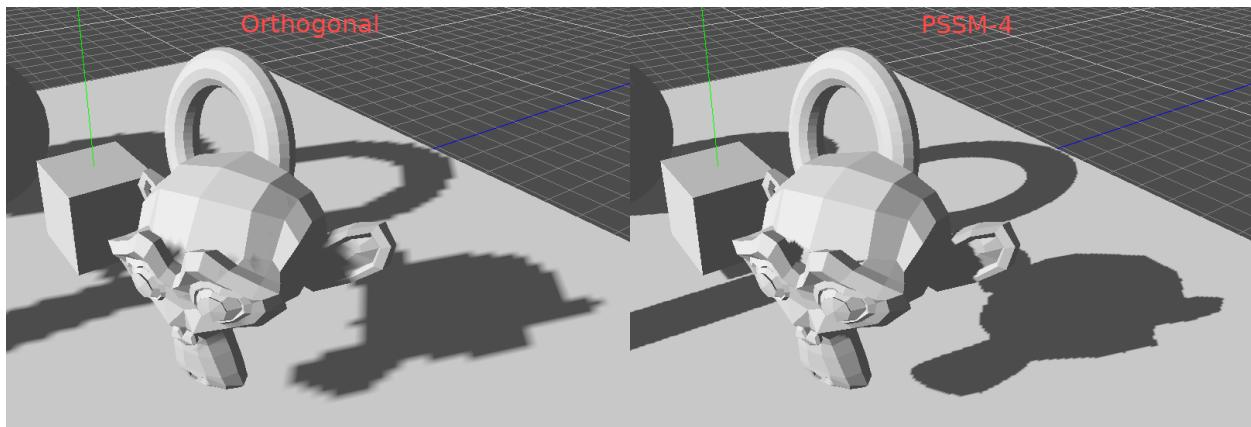


To fix this, a technique named “Parallel Split Shadow Maps” (or PSSM) is used. This splits the view frustum in 2 or 4 areas. Each area gets it’s own shadow map. This allows small, close areas to the viewer to have the same shadow resolution as a huge, far-away area.

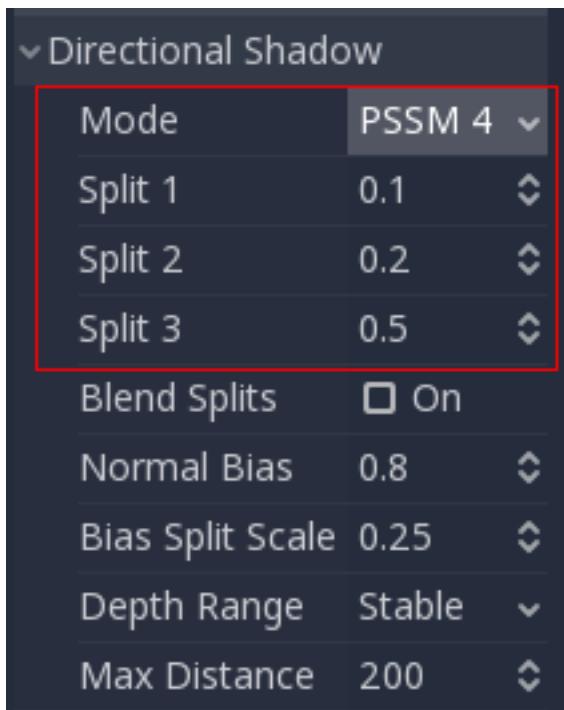
Parallel Split Shadow Mapping



With this, shadows become more detailed:



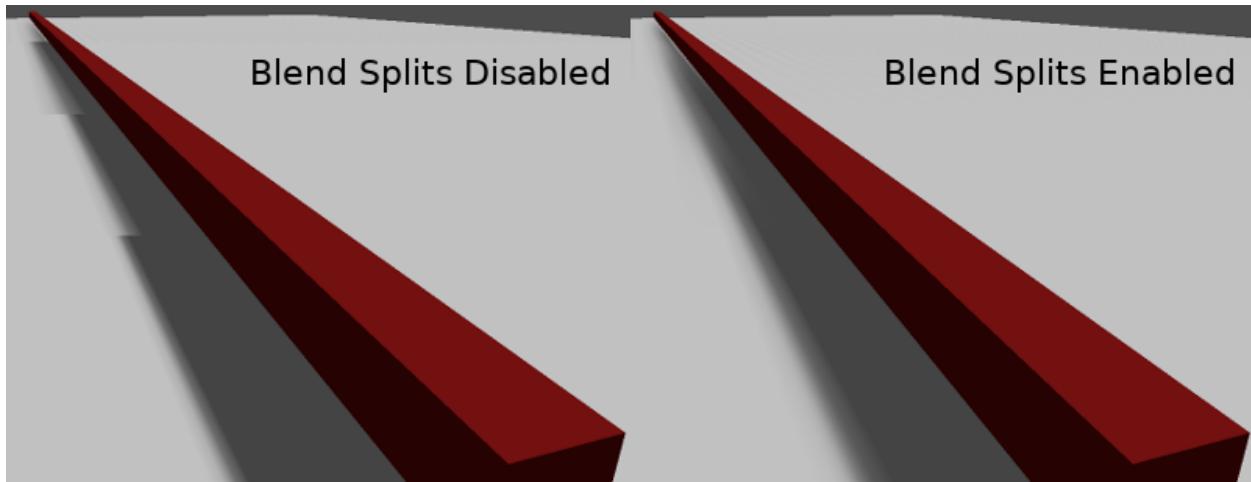
To control PSSM, a number of parameters are exposed:



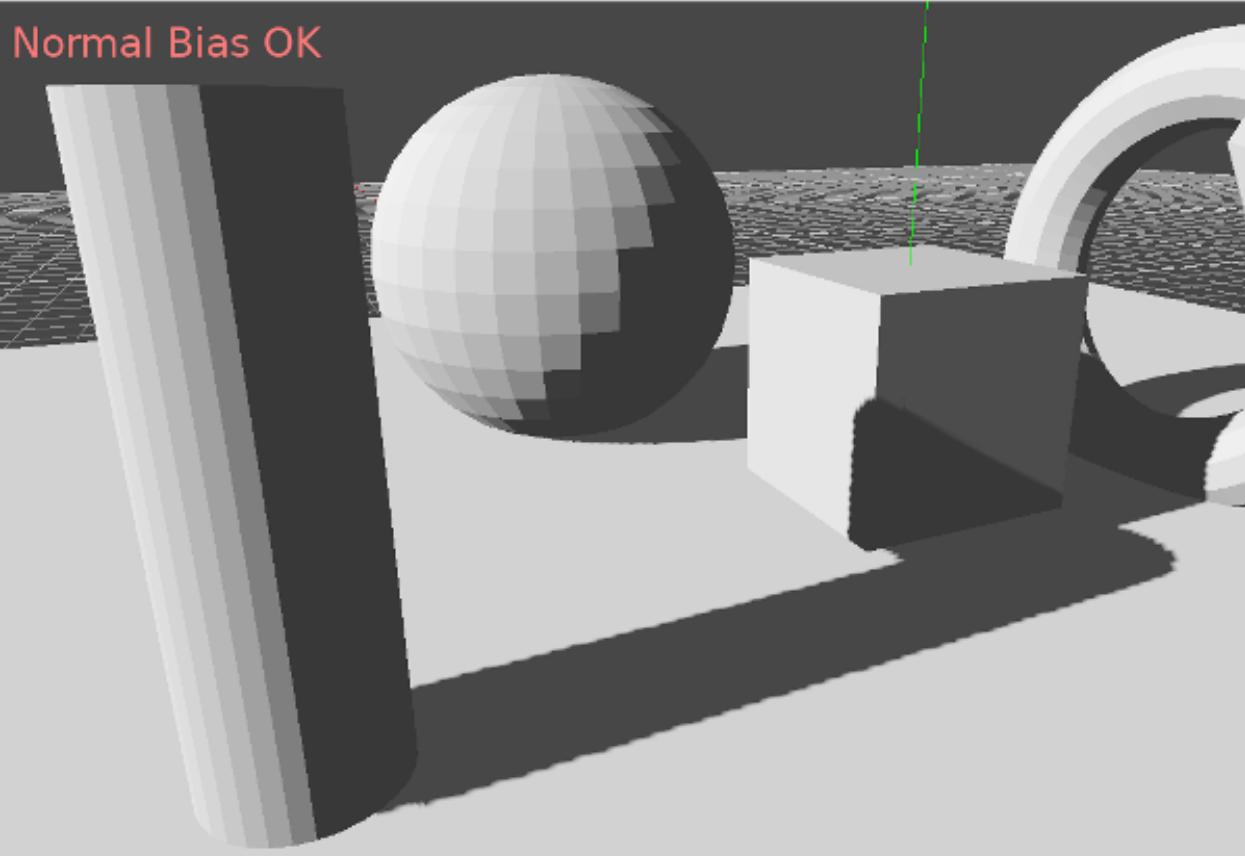
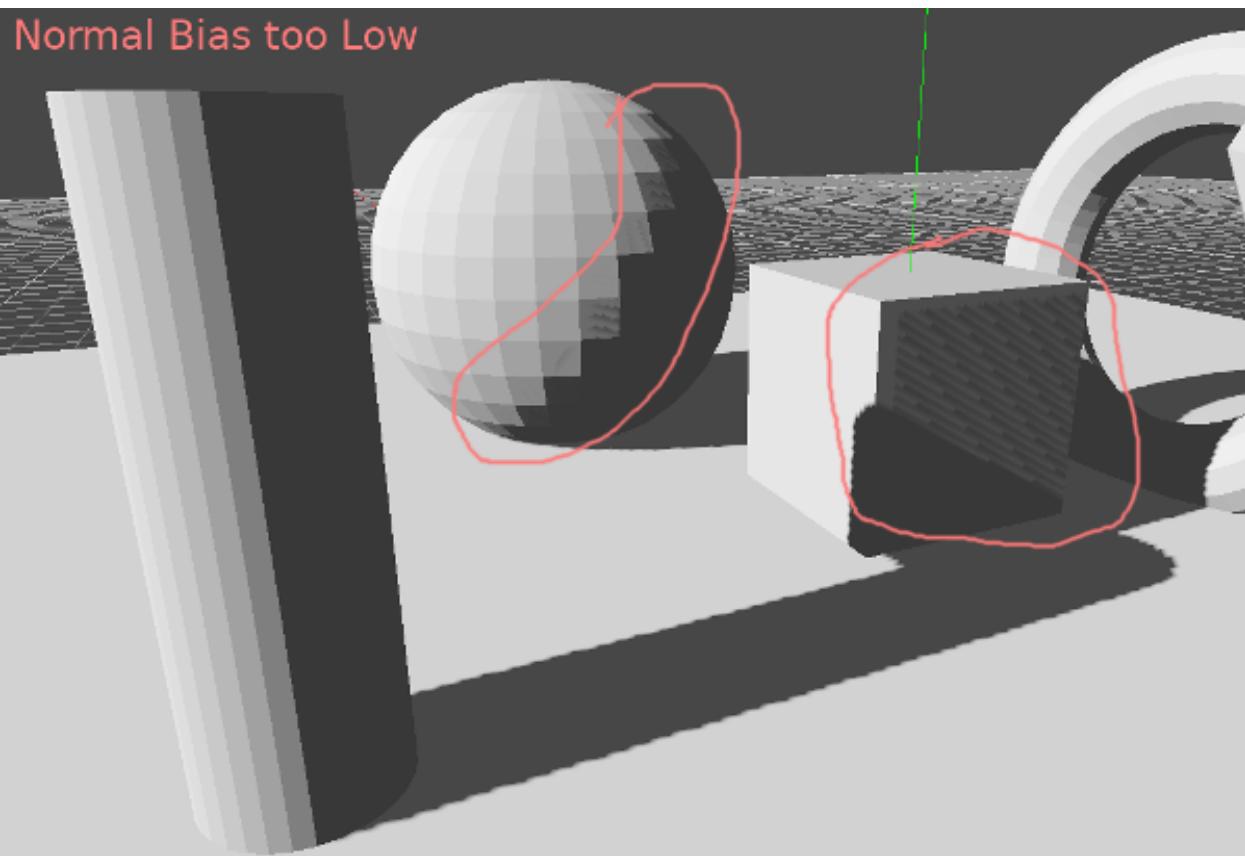
Each split distance is controlled relative to the camera far (or shadow **Max Distance** if greater than zero), so *0.0* is the eye position and *1.0* is where the shadow ends at a distance. Splits are in-between. Default values generally work well, but tweaking the first split a bit is common to give more detail to close objects (like a character in a third person game).

Always make sure to set a shadow *Max Distance* according to what the scene needs. The closer the max distance, the higher quality they shadows will have.

Sometimes, the transition between a split and the next can look bad. To fix this, the “**Blend Splits**” option can be turned on, which sacrifices detail in exchange for smoother transitions:



The “**Normal Bias**” parameter can be used to fix special cases of self shadowing when objects are perpendicular to the light. The only downside is that it makes the shadow a bit thinner.



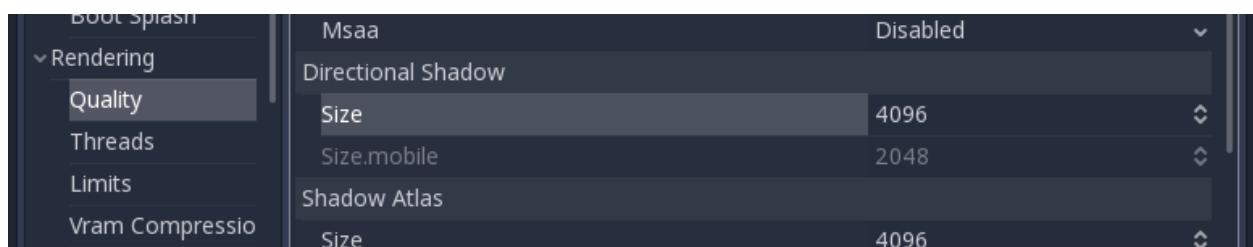
The “**Bias Split Scale**” parameter can control extra bias for the splits that are far away. If self shadowing occurs only on the splits far away, this value can fix them.

Finally, the “**Depth Range**” has to settings:

- **Stable**: Keeps the shadow stable while the camera moves, the blocks that appear in the outline when close to the shadow edges remain in-place. This is the default and generally desired, but it reduces the effective shadow resolution.
- **Optimized**: Tries to achieve the maximum resolution available at any given time. This may result in a “moving saw” effect on shadow edges, but at the same time the shadow looks more detailed (so this effect may be subtle enough to be forgiven).

Just experiment which setting works better for your scene.

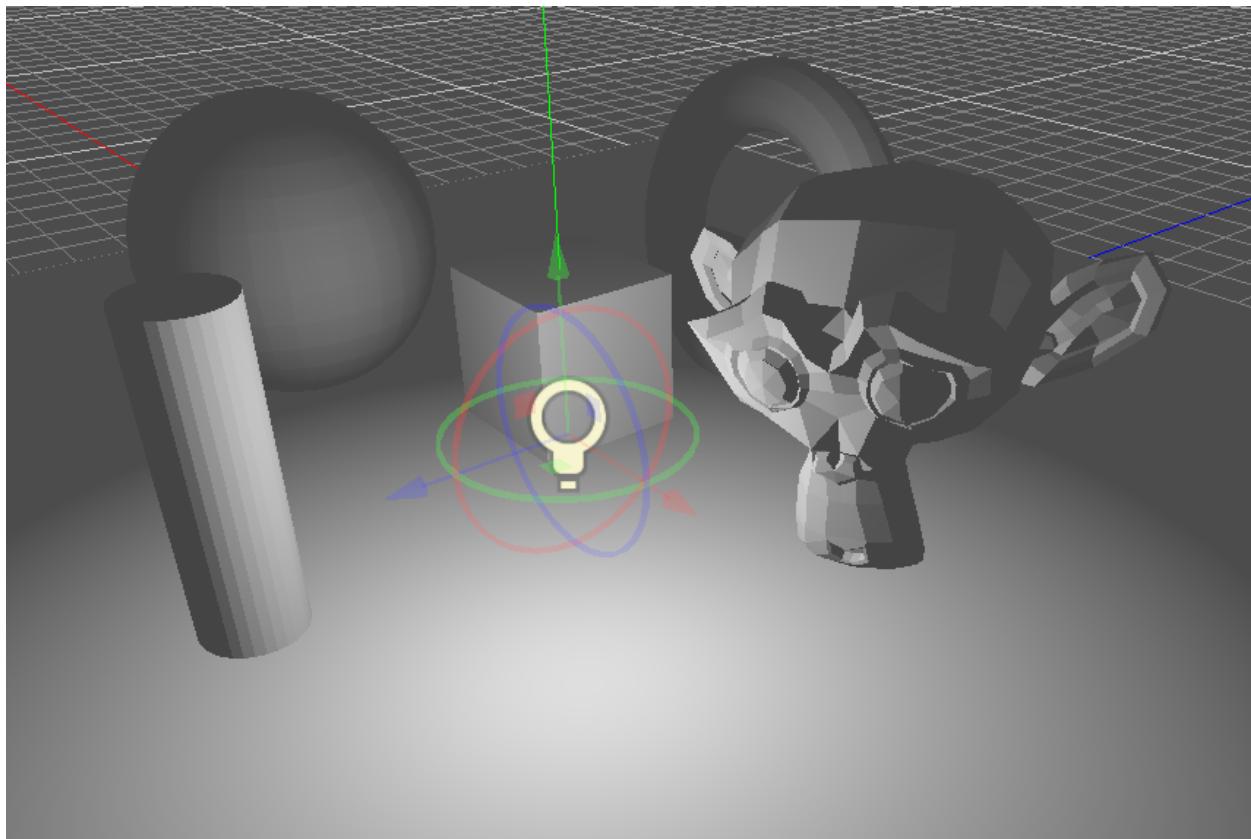
Shadowmap size for directional lights can be changed in Project Settings -> Rendering -> Quality:



Increasing it can solve bias problems, but reduce performance. Shadow mapping is an art of tweaking.

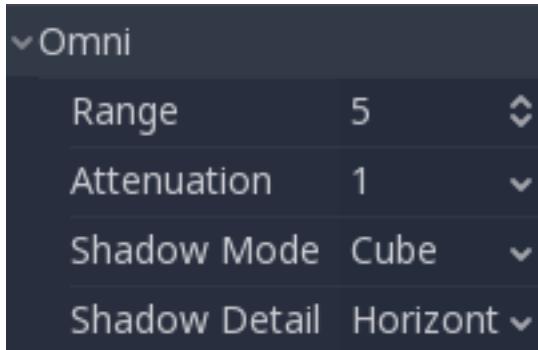
Omni light

Omni light is a point source that emits light spherically in all directions up to a given radius .

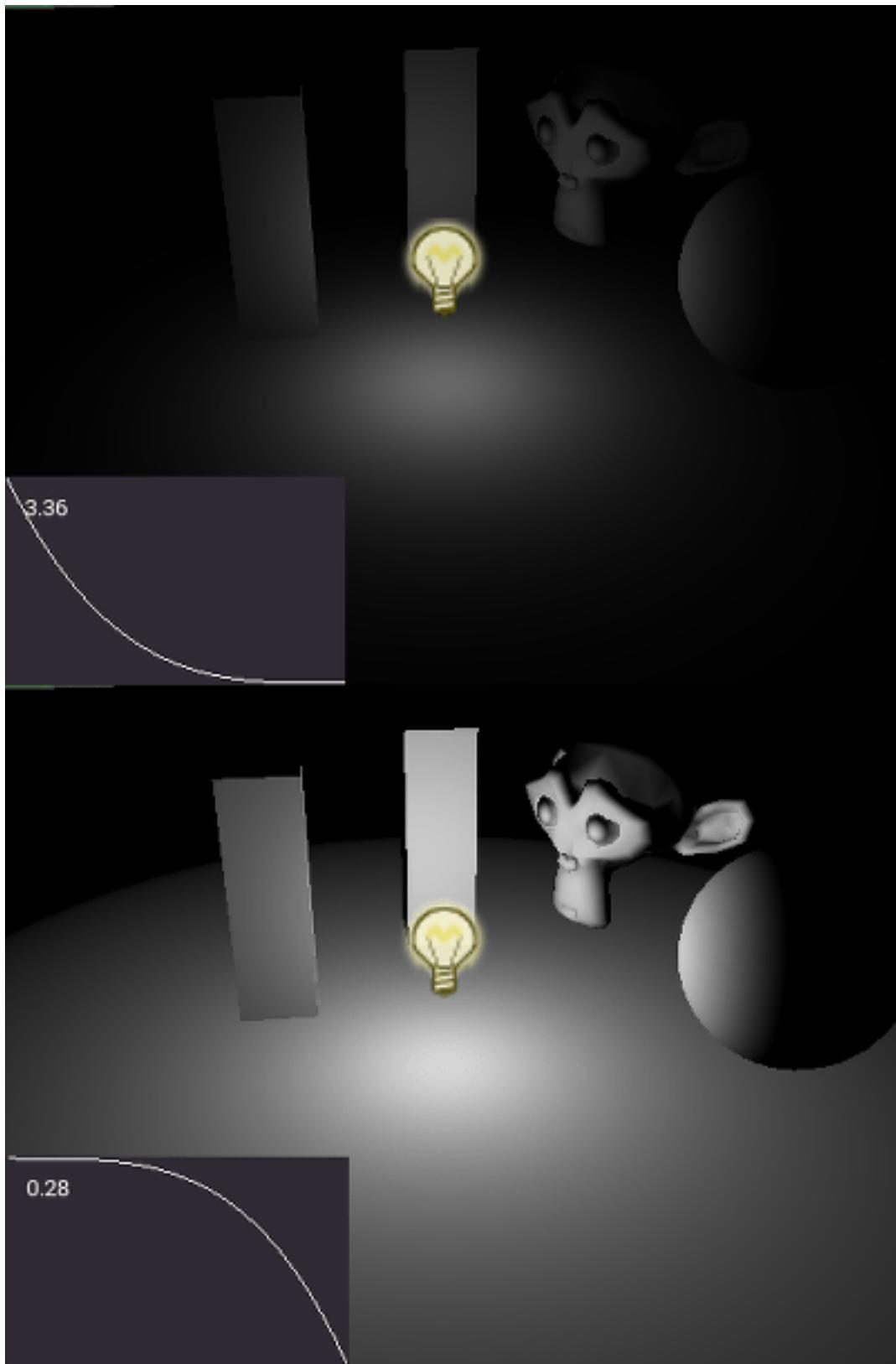


In real life, light attenuation is an inverse function, which means omni lights don't really have a radius. This is a problem, because it means computing several omni lights would become really demanding.

To solve this, a *Range* is introduced, together with an attenuation function.



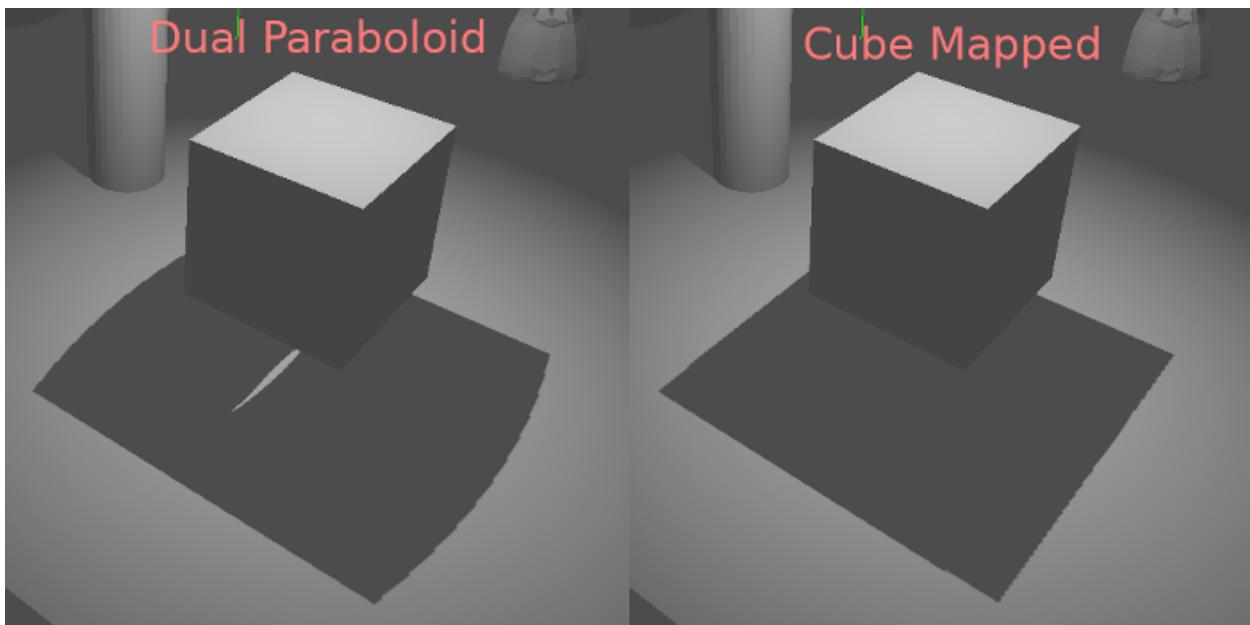
These two parameters allow tweaking how this works visually, in order to find aesthetically pleasing results.



Omni Shadow Mapping

Omni light shadow mapping is relatively straightforward, as it just works. The main issue that needs to be considered is the algorithm used to render it.

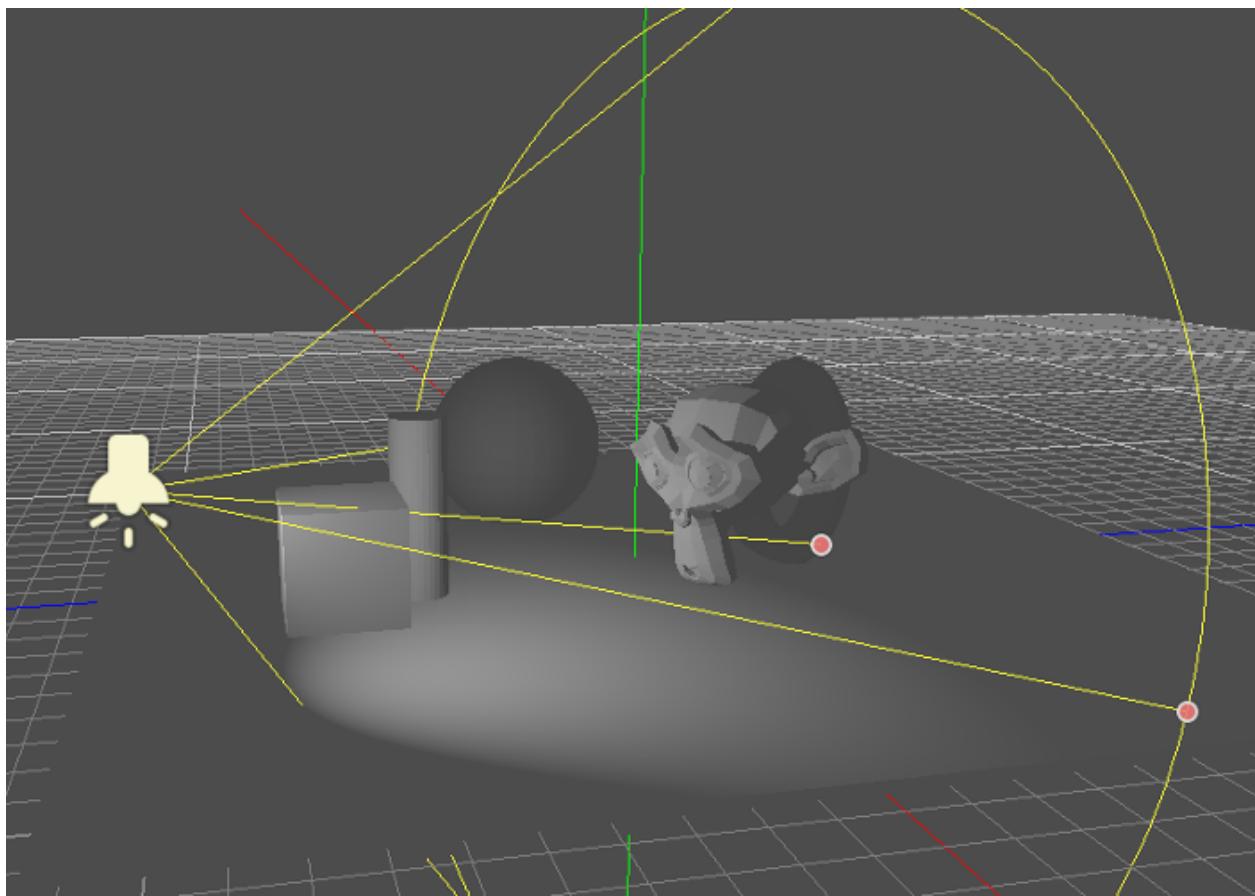
Omni Shadows can be rendered as either “**Dual Paraboloid**” or “**Cube Mapped**”. The former renders very quickly but can cause deformations, while the later is more correct but more costly.



If the objects being rendered are mostly irregular, Dual Paraboloid is usually enough. In any case, as these shadows are cached in a shadow atlas (more on that at the end), it may not make a difference in performance for most scenes.

Spot light

Spot lights are similar to omni lights, except they emit light only into a cone (or “cutoff”). They are useful to simulate flashlights, car lights, reflectors, spots, etc. This type of light is also attenuated towards the opposite direction it points to.



Spot lights share the same **Range** and **Attenuation** as **OmniLight**, and add two extra parameters:

- **Angle:** The aperture angle of the light
- **Angle Attenuation:** The cone attenuation, which helps soften the cone borders.

Spot Shadow Mapping

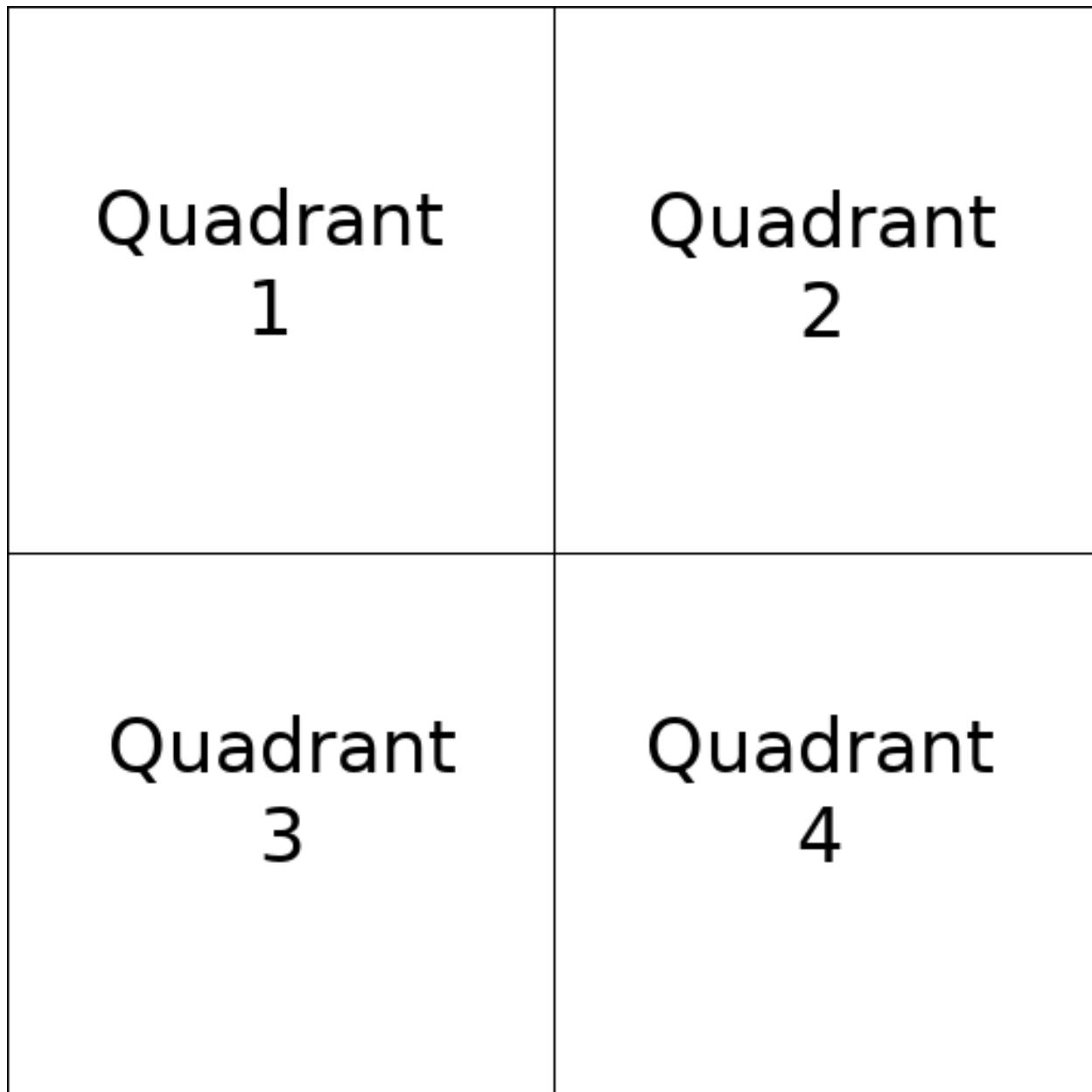
Spots don't need any parameters for shadow mapping, they should just work. Keep in mind that, at more than 89 degrees of aperture, shadows stop functioning for spots, and you should consider using an Omni light.

Shadow Atlas

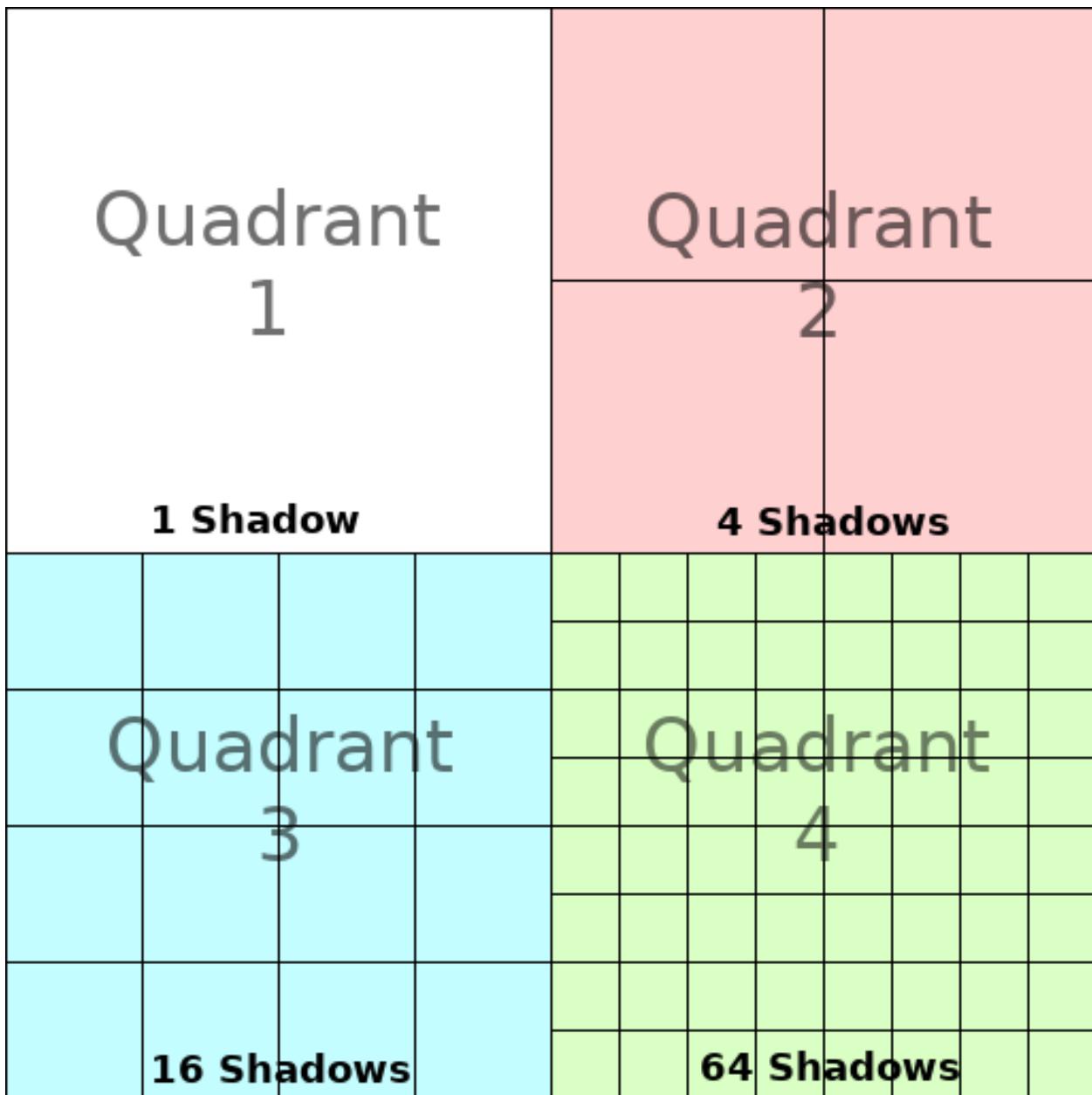
Unlike Directional lights, which have their own shadow texture, Omni and Spot lights are assigned to slots of a shadow atlas. This atlas can be configured in Project Settings -> Rendering -> Quality -> Shadow Atlas.

Rendering	Size.mobile	2048
Quality	Shadow Atlas	
Threads	Size	4096
Limits	Size.mobile	2048
Vram Compressio	Quadrant 0 Subdiv	1 Shadow
Environment	Quadrant 1 Subdiv	4 Shadows
Debug	Quadrant 2 Subdiv	16 Shadows
Settings	Quadrant 3 Subdiv	64 Shadows
Shapes	Shadows	

The resolution applies to the whole Shadow Atlas. This atlas is divided in four quadrants:



Each quadrant, can be subdivided to allocate any number of shadow maps, following is the default subdivision:



The allocation logic is simple, the biggest shadow map size (when no subdivision is used) represents a light the size of the screen (or bigger). Subdivisions (smaller maps) represent shadows for lights that are further away from view and proportionally smaller.

Every frame, the following logic is done for all lights:

1. Check if the light is on a slot of the right size, if not, re-render it and move it to a larger/smaller slot.
2. Check if any object affecting the shadow map has changed, if it did, re-render the light.
3. If neither of the above has happened, nothing is done and the shadow is left untouched.

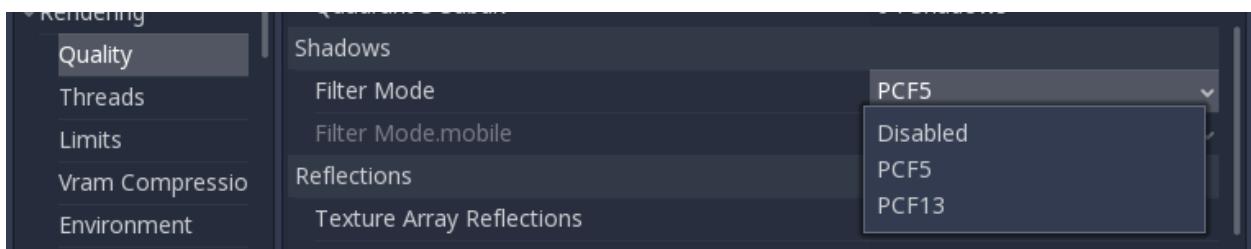
If the slots in a quadrant are full, lights are pushed back to smaller slots depending on size and distance.

This allocation strategy works for most games, but you may want to use a separate one in some cases (as example, a

top-down game where all lights are around the same size and quadrants may have all the same subdivision).

Shadow Filter Quality

The filter quality of shadows can be tweaked. This can be found in Project Settings -> Rendering -> Quality -> Shadows. Godot supports no filter, PCF5 and PCF13.



It affects the blockyness of the shadow outline:



7.5 Reflection Probes

7.5.1 Introduction

As stated in the *Spatial Material*, objects can show reflected or diffuse light. Reflection Probes are used as a source of reflected and ambient light for objects inside their area of influence.

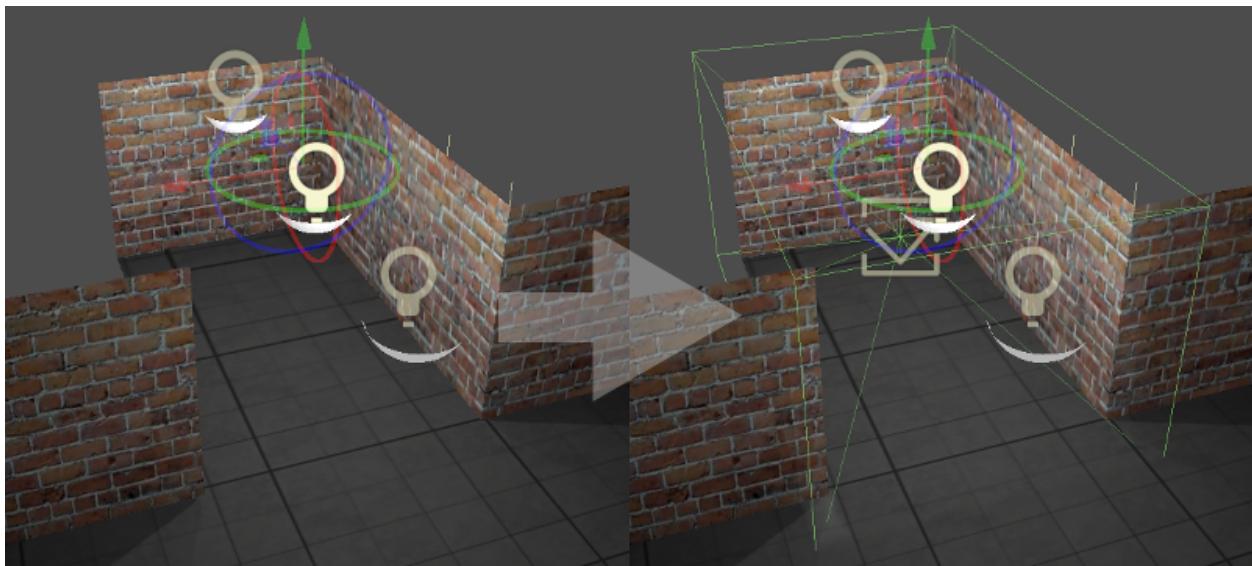
A probe of this type captures the surroundings (as a sort of 360 degrees image), and stores versions of it with increasing levels of *blur*. This is used to simulate roughness in materials, as well as ambient lighting.

While these probes are a very efficient way of storing reflections, they have a few shortcomings:

- They are efficient to render, but expensive to compute. This leads to a default behavior where they only capture on scene load.
- They work best for rectangular shaped rooms or places, otherwise the reflections shown are not as faithful (specially when roughness is 0).

7.5.2 Setting Up

Setting up reflection probes is really easy! Just create a `ReflectionProbe` node, and wrap it around the area where you want to have reflections:



This should result in immediate local reflections. If you are using a Sky texture, reflections are by default blended with it.

By default, on interiors, reflections may appear to not have much consistence. In this scenario, make sure to tick the “*Box Correct*” property.

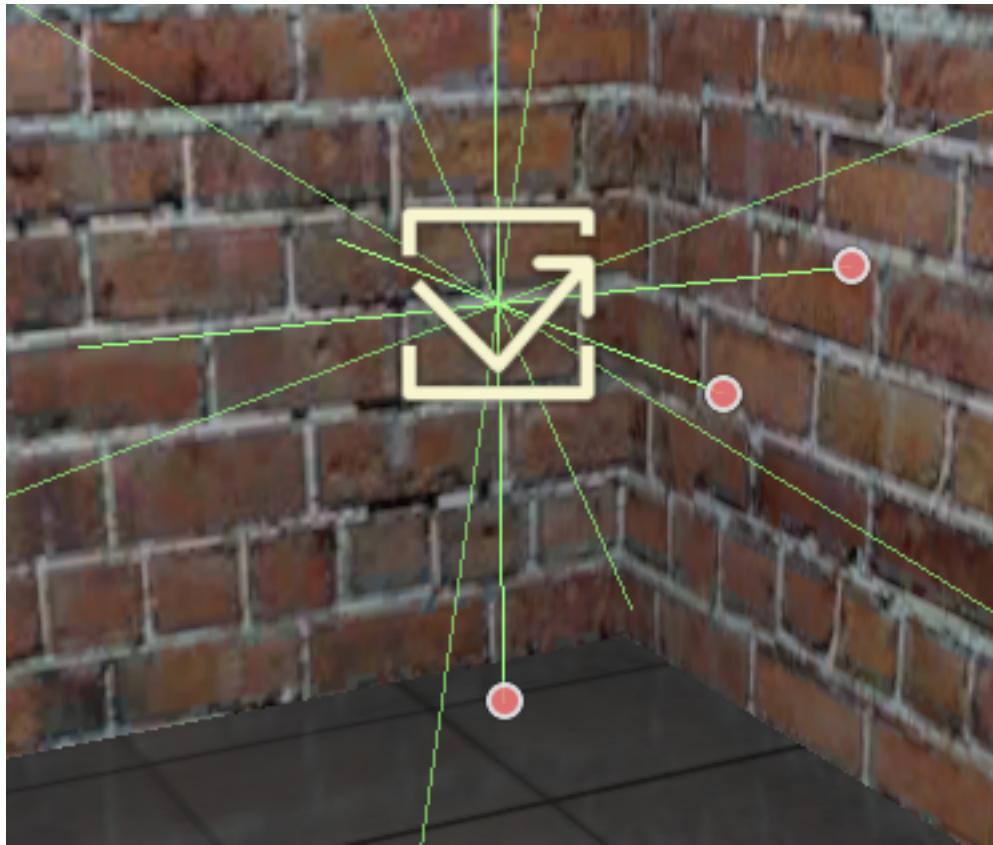
Box Projection **On**

This setting changes the reflection from an infinite skybox to reflecting a box the size of the probe:



Adjusting the box walls may help improve the reflection a bit, but it will always look the best in box shaped rooms.

The probe captures the surrounding from the center of the gizmo. If, for some reason, the room shape or contents occlude the center, it can be displaced to an empty place by moving the handles in the center:

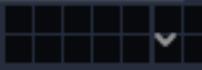


By default, shadow mapping is disabled when rendering probes (only in the rendered image inside the probe, not the actual scene). This is a simple way to save on performance and memory. If you really want shadows in the probe, they can be toggled on/off with the *Enable Shadow* setting:

Enable Shadow On

Finally, keep in mind that you may not want the Reflection Probe to render some objects. A typical scenario is an enemy inside the room which will move around. To keep objects from being rendered in the reflections, use the *Cull Mask* setting:

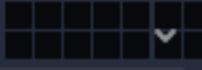
Cull Mask



7.5.3 Interior vs Exterior

If you are using reflection probes in an interior setting, it is recommended that the **Interior** property is enabled. This makes the probe not render the sky, and also allows custom ambient lighting settings.

Cull Mask

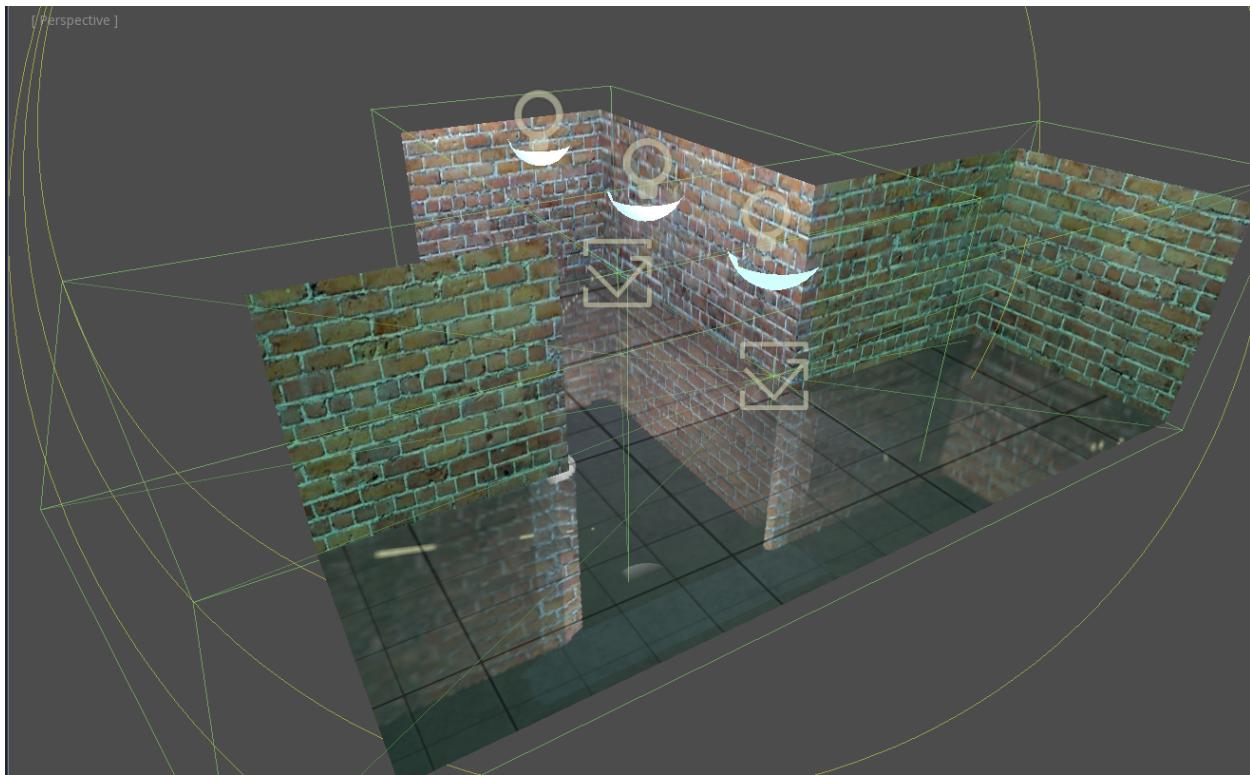


When probes are set to **Interior**, custom constant ambient lighting can be specified per probe. Just choose a color and an energy.

Optionally, you can blend this ambient light with the probe diffuse capture by tweaking the **Ambient Contribution** property (0.0 means, pure ambient color, while 1.0 means pure diffuse capture).

7.5.4 Blending

Multiple reflection probes can be used and Godot will blend them where they overlap using a smart algorithm:



As you can see, this blending is never perfect (after all, these are box reflections, not real reflections), but these artifacts are only visible when using perfectly mirrored reflections. Normally, scenes have normal mapping and varying levels of roughness which can hide this.

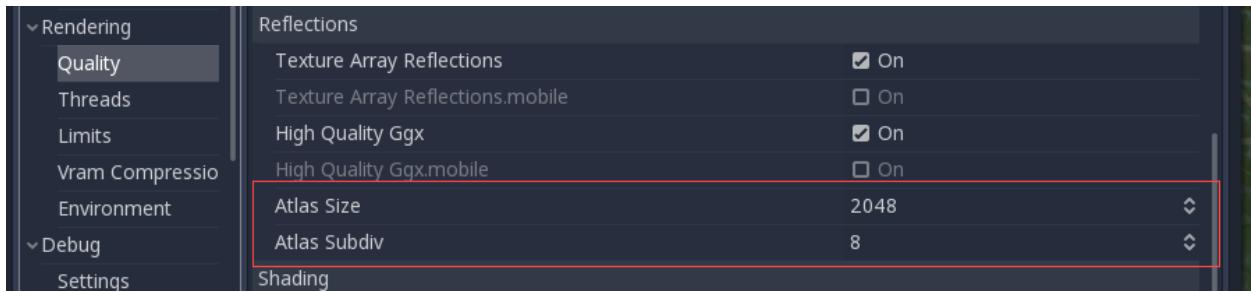
Alternatively, Reflection Probes work very well blended together with Screen Space Reflections to solve these problems. Combining them makes local reflections appear more faithful, while probes only used as fallback when no screen-space information is found:



Finally, blending interior and exterior probes is a recommended approach when making levels that combine both interiors and exteriors. Near the door, a probe can be marked as *exterior* (so it will get sky reflections), while on the inside it can be interior.

7.5.5 Reflection Atlas

In the current renderer implementation, all probes are the same size and they are fit into a Reflection Atlas. The size and amount of probes can be customized in Project Settings -> Quality -> Reflections



7.6 GI Probes

7.6.1 Introduction

Just like with *Reflection Probes*, and as stated in the *Spatial Material*, objects can show reflected or diffuse light. GI Probes are similar to Reflection Probes, but they use a different and more complex technique to produce indirect light and reflections.

The strength of GI Probes are real-time, high quality, indirect light. While the scene needs a quick pre-bake for the static objects that will be used, lights can be added, changed or removed and this will be updated in real-time. Dynamic objects that move within one of these probes will also receive indirect lighting from the scene automatically.

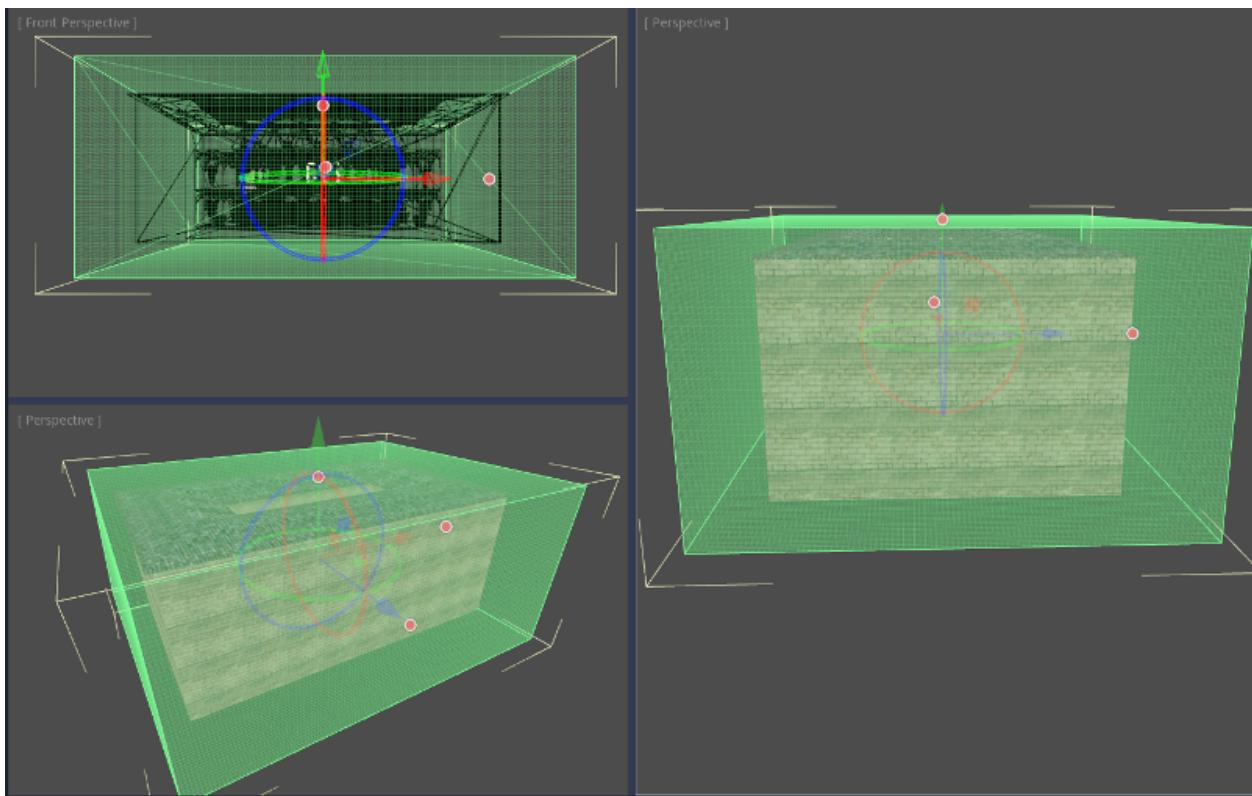
Just like with *ReflectionProbe*, *GIProbes* can be blended (in a bit more limited way), so it is possible to provide full real-time lighting for a stage without having to resort to lightmaps.

The main downside of *GIProbes* are:

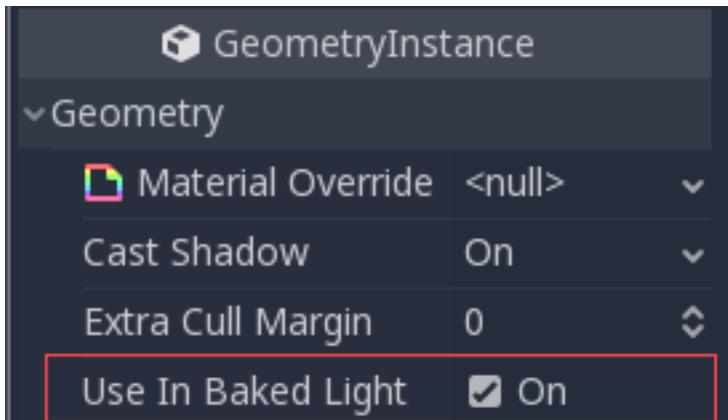
- A small amount of light leaking can occur if the level is not carefully designed. this must be artist-tweaked.
- Performance requirements are higher than for lightmaps, so it may not run properly in low end integrated GPUs (may need to reduce resolution).
- Reflections are voxelized, so they don't look as sharp as with *ReflectionProbe*, but in exchange they are volumetric so any room size or shape works for them. Mixing them with Screen Space Reflection also works very well.
- They consume considerably more video memory than *Reflection Probes*, so they must be used by care in the right subdivision sizes.

7.6.2 Setting Up

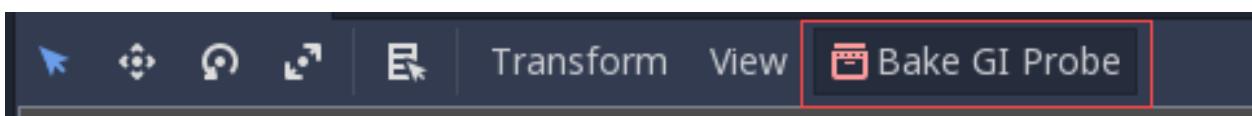
Just like a *ReflectionProbe*, simply set up the *GIProbe* by wrapping it around the geometry that will be affected.



Afterwards, make sure to enable the geometry will be baked. This is important in order for GIPRoe to recognize objects, otherwise they will be ignored:



Once the geometry is set-up, push the Bake button that appears on the 3D editor toolbar to begin the pre-baking process:



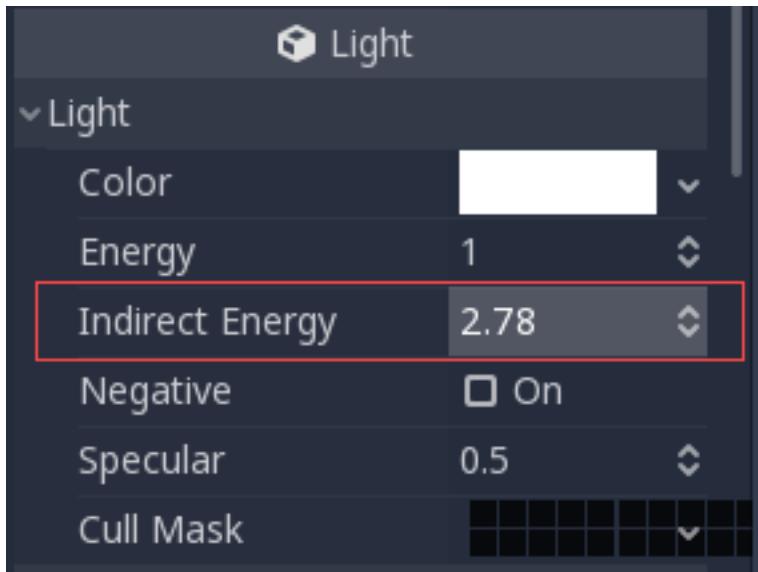
7.6.3 Adding Lights

Unless there are materials with emission, GIPRoe does nothing by default. Lights need to be added to the scene to have an effect.

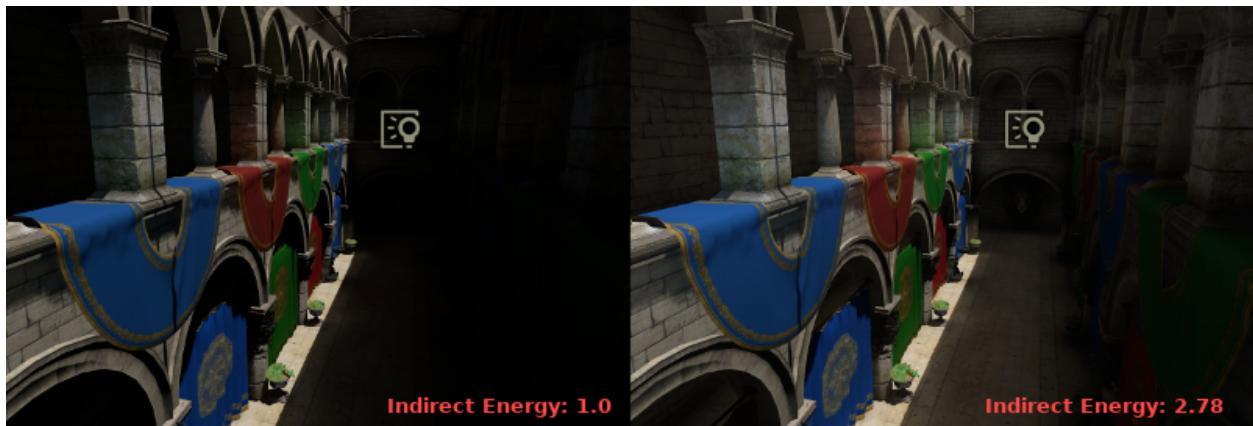
The effect of indirect light can be viewed quickly (it is recommended you turn off all ambient/sky lighting to tweak this, though as in the picture):



In some situations, though, indirect light may be too weak. Lights have an indirect multiplier to tweak this:



And, as GIPRope lighting updates in real-time, this effect is immediate:



7.6.4 Reflections

For materials with high metalness and low roughness, it's possible to appreciate voxel reflections. Keep in mind that these have far less detail than Reflection Probes or Screen Space Reflections, but fully reflect volumetrically.



GIProbes can easily be mixed with Reflection Probes and Screen Space Reflections, as a full 3-stage fallback-chain. This allows to have precise reflections where needed:



7.6.5 Interior vs Exterior

GI Probes normally allow mixing with lighting from the sky. This can be disabled when turning on the *Interior* setting.



The difference becomes clear in the image below, where light from the sky goes from spreading inside to being ignored.



As complex buildings may mix interiors with exteriors, combining GIProbes for both parts works very well.

7.6.6 Tweaking

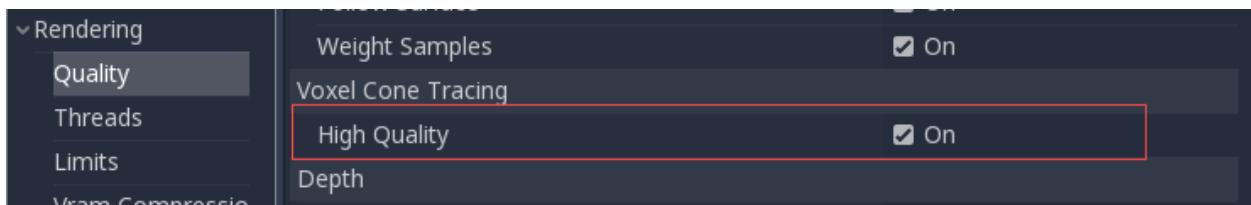
GI Probes support a few parameters for tweaking:



- **Subdiv** Subdivision used for the probe. The default (128) is generally good for small to medium size areas. Bigger subdivisions use more memory.
- **Extents** Size of the probe, can be tweaked from the gizmo.
- **Dynamic Range** Maximum light energy the probe can absorb. Higher values allow brighter light, but with less color detail.
- **Energy** Multiplier for all the probe. Can be used to make the indirect light brighter (although it's better to tweak this from the light itself).
- **Propagation** How much light propagates through the probe internally.
- **Bias** Value used to avoid self-occlusion when doing voxel cone tracing, should generally be above 1.0 (1==voxel size).
- **Normal Bias** Alternative type of bias useful for some scenes. Experiment with this one if regular bias does not work.

7.6.7 Quality

GIProbes are quite demanding. It is possible to use lower quality voxel cone tracing in exchange of more performance.



7.7 Baked Lightmaps

7.7.1 Introduction

Baked lightmaps are an alternative workflow for adding indirect (or baked) lighting to a scene. Unlike the *GI Probes* approach, baked lightmaps work fine on very low end PCs and mobile as they consume almost no resources in run-time.

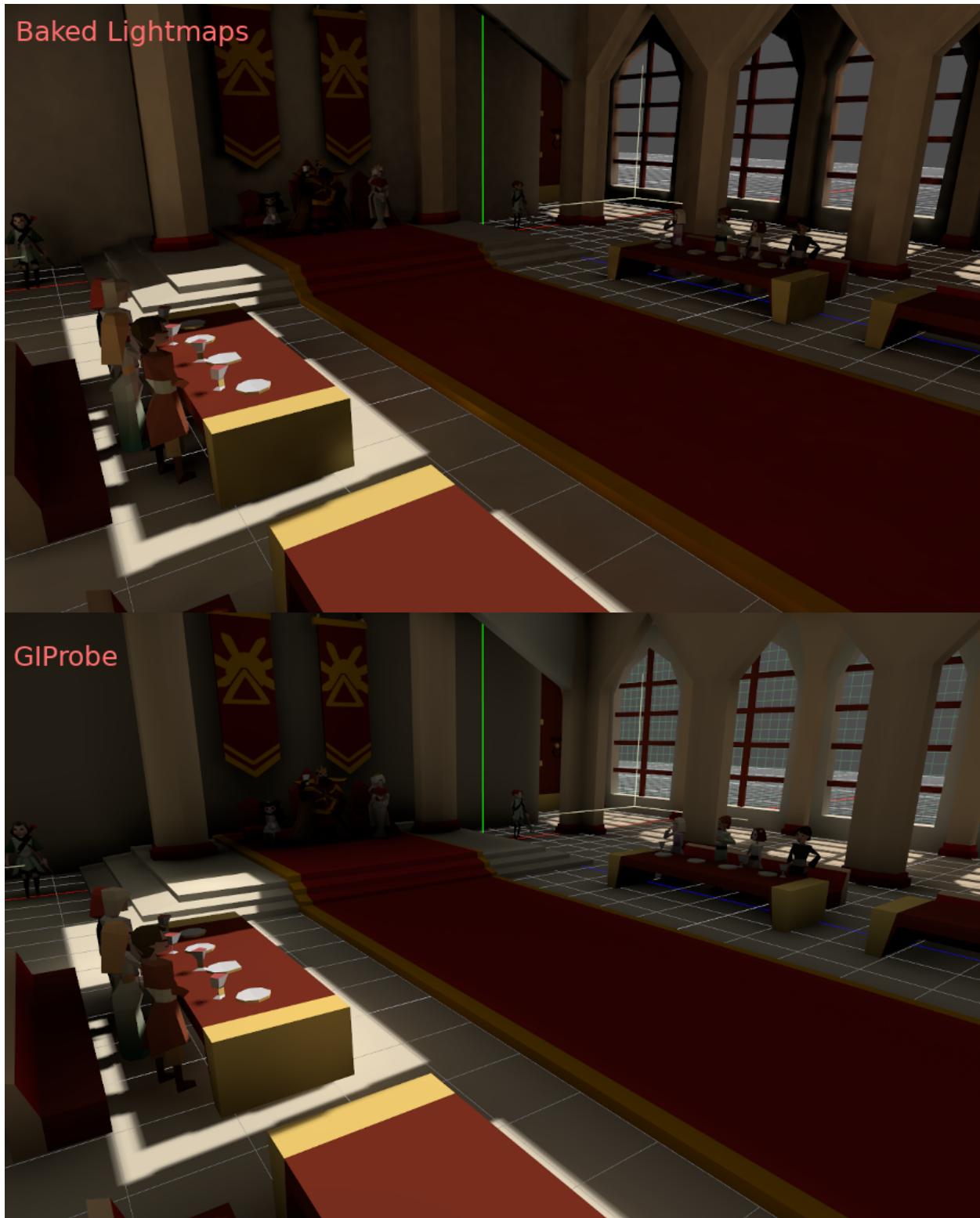
Unlike GIProbes, Baked Lightmaps are completely static, once baked they can't be modified at all. They also don't provide the scene with reflections, so using *Reflection Probes* together with it on interiors (or using a Sky on exteriors) is a requirement to get good quality.

As they are baked, they have less problems regarding to light bleeding than GIProbe and indirect light can look better if using Raytrace mode on high quality setting (but baking can take a while to bake).

In the end, deciding which indirect lighting approach is better depends on your use case. In general GIProbe looks better and is much easier to set up. For low end compatibility or mobile, though, Baked Lightmaps are your only choice.

7.7.2 Visual Comparison

Here are some comparisons of how Baked Lightmaps vs GIProbe look. Notice that lightmaps are more accurate, but also suffer from the fact that lighting is on an unwrapped texture, so transitions and resolution may not be that good. GIProbe looks less accurate (as it's an approximation), but more smooth overall.



7.7.3 Setting Up

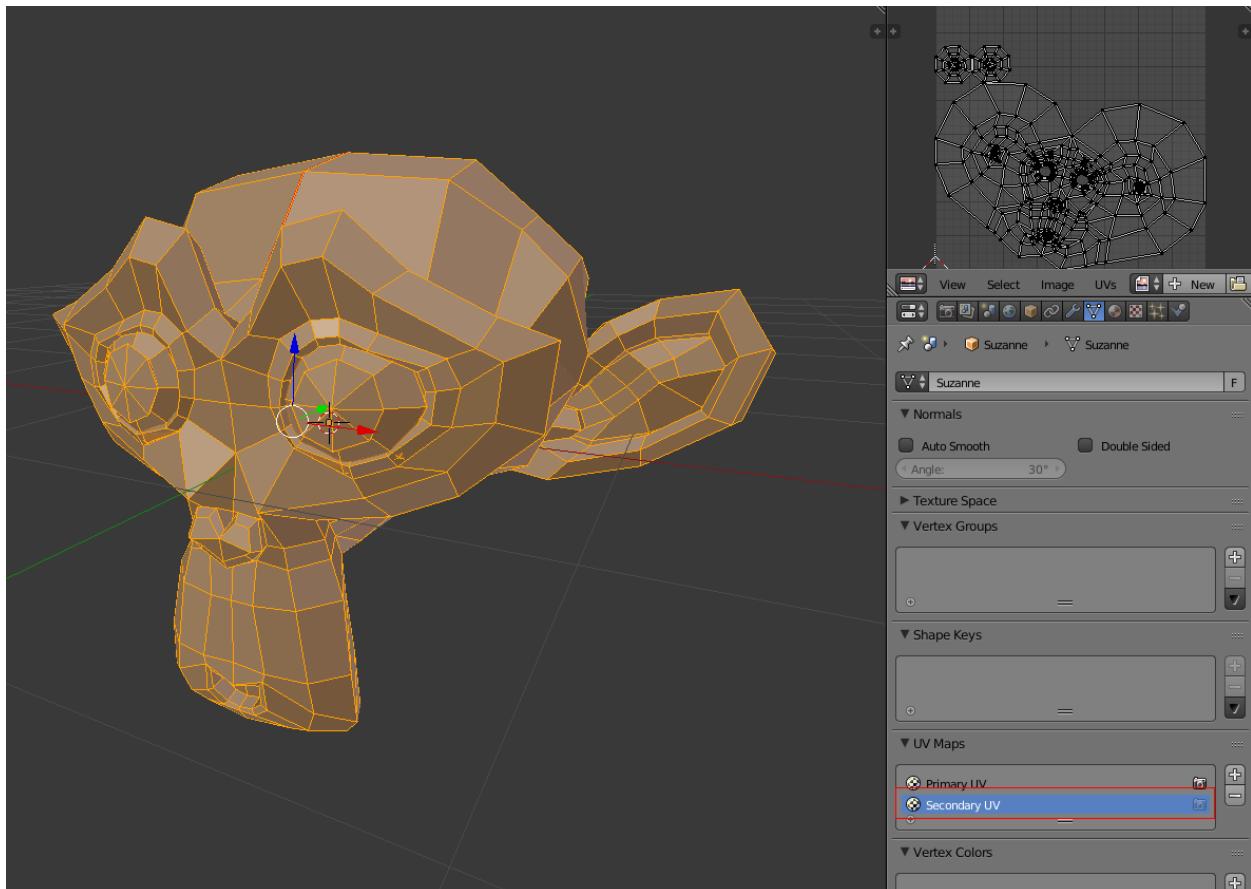
First of all, before the lightmapper can do anything, objects to be baked need an UV2 layer and a texture size. An UV2 layer is a set of secondary texture coordinates that ensures any face in the object has its own place in the UV map. Faces must not share pixels in the texture.

There are a few ways to ensure your object has a unique UV2 layer and texture size

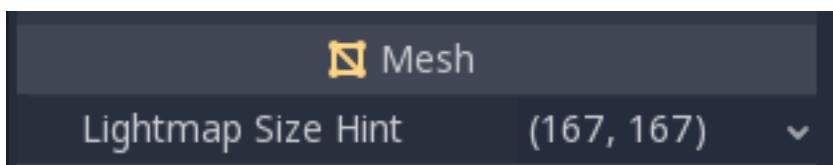
Unwrap from your 3D DCC

One option is to do it from your favorite 3D app. This approach is generally not recommended but it's explained first so you know it exists. The main advantage is that, on very complex objects that you may want to re-import a lot, the texture generation process can be quite costly within Godot, so having it unwrapped before import can be faster.

Simply do an unwrap on the second UV2 layer.



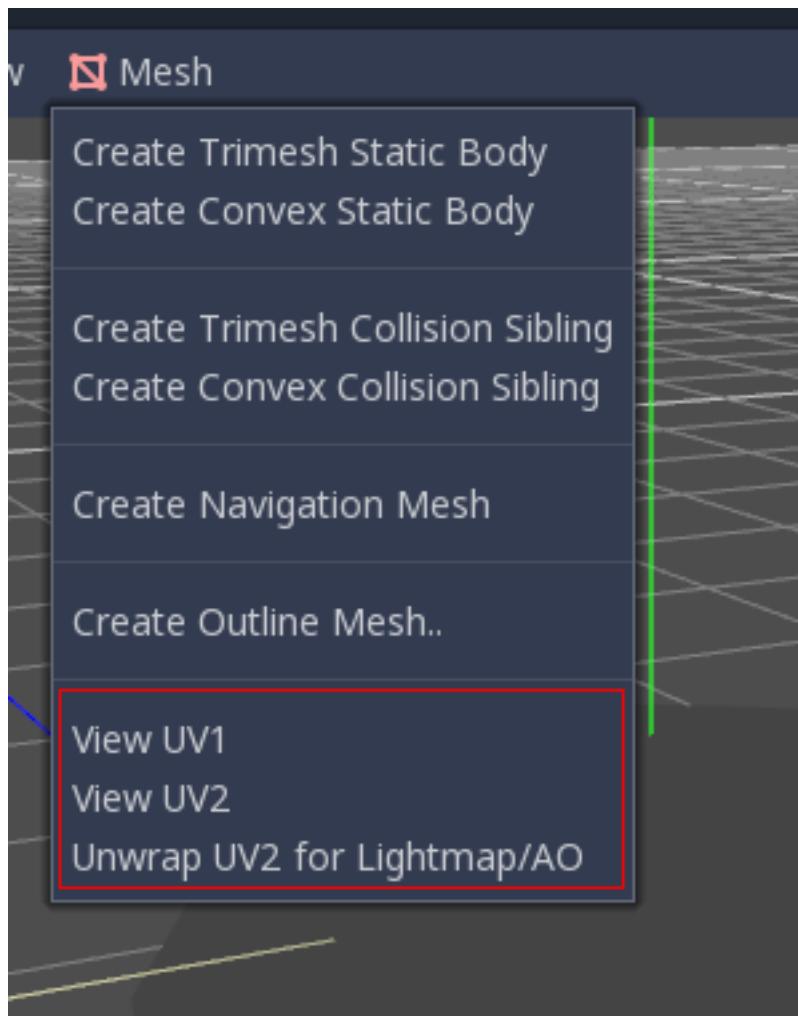
And import normally. Remember you will need to set the texture size on the mesh after import.



If you use external meshes on import, the size will be kept. Be wary that most unwrappers in 3D DCCs are not very quality oriented, as they are meant to work quick. You will mostly need to use seams or other techniques to create better unwrapping.

Unwrap from within Godot

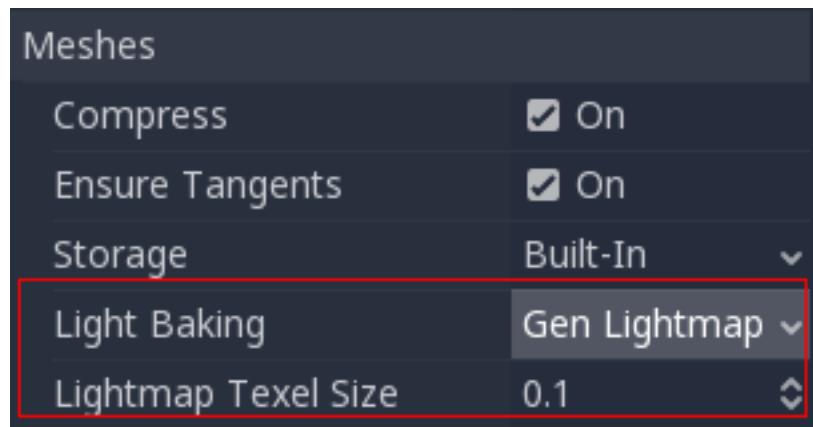
Godot has an option to unwrap meshes and visualize the UV channels. It can be found in the Mesh menu:



This will generate a second set of UV2 coordinates, which can be used for baking and it will set the texture size automatically.

Unwrap on Scene import

This is probably the best approach overall. The only downside is that, on large models, unwrap can take a while on import. Just select the imported scene in the filesystem dock, then go to the Import tab. There, the following option can be modified:



The **Light Baking** mode needs to be set to “**Gen Lightmaps**”. A texel size in world units must also be provided, as this will determine the final size of the lightmap texture (and, in consequence, the UV padding in the map).

The effect of setting this option is that all meshes within the scene will have their UV2 maps properly generated.

As a word of warning: When reusing a mesh within a scene, keep in mind that UVs will be generated for the first instance found. If the mesh is re-used with different scales (and the scales are wildly different, more than half or twice), this will result in inefficient lightmaps. Just don't reuse a source mesh at very different scales if you are planning to use lightmapping.

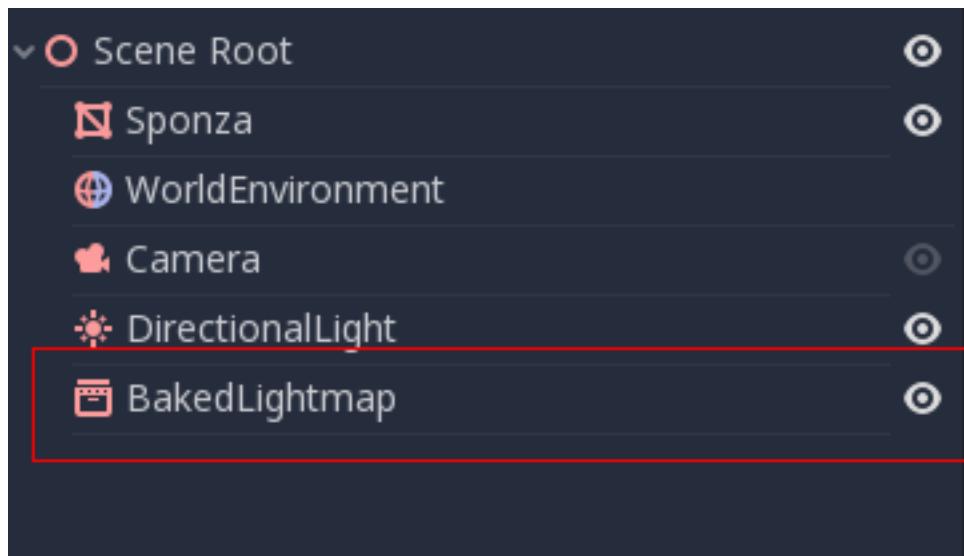
Checking UV2

In the mesh menu mentioned before, the UV2 texture coordinates can be visualized. Make sure, if something is failing, to check that the meshes have these UV2 coordinates:



7.7.4 Setting up the Scene

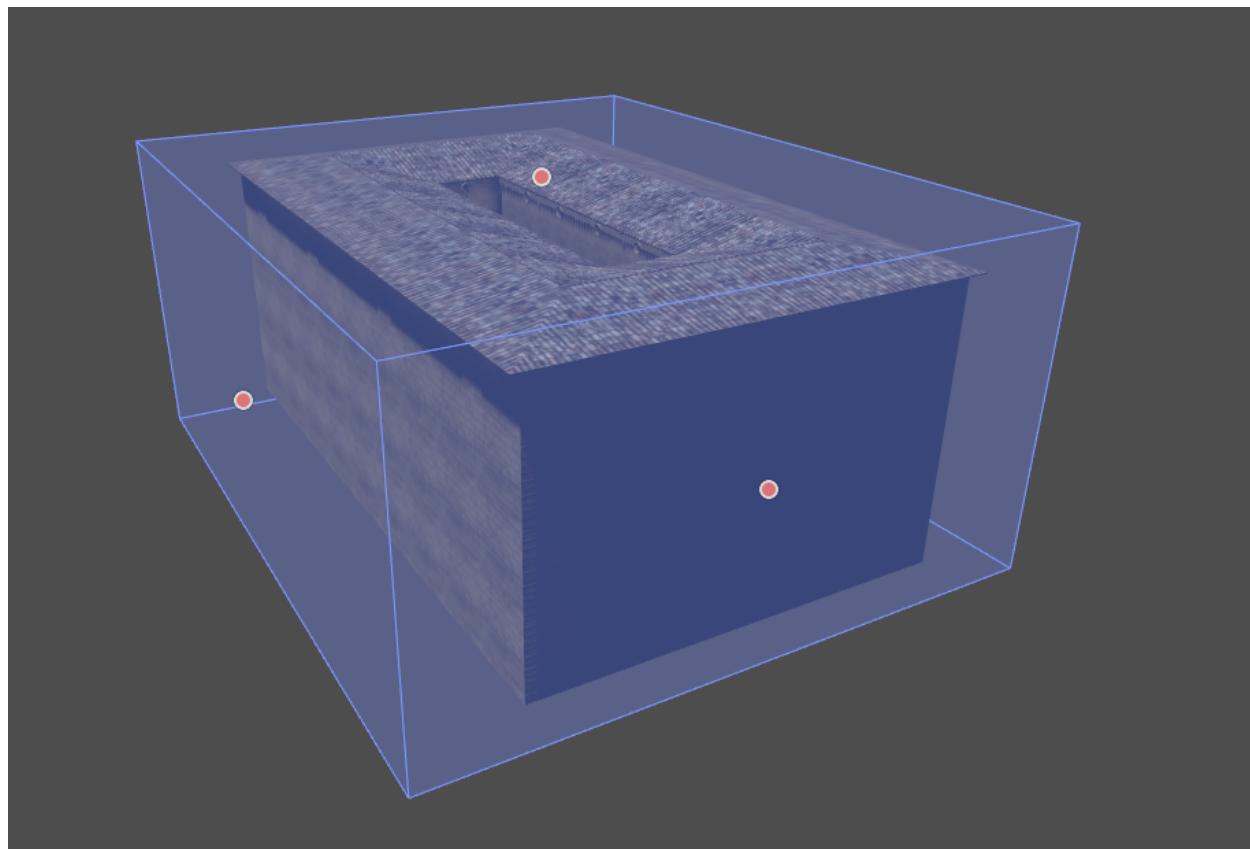
Before anything is done, a **BakedLight** Node needs to be added to a scene. This will enable light baking on all nodes (and sub-nodes) in that scene, even on instanced scenes.



A sub-scene can be instanced several times, as this is supported by the baker and each will be assigned a lightmap of its own (just make sure to respect the rule about scaling mentioned before):

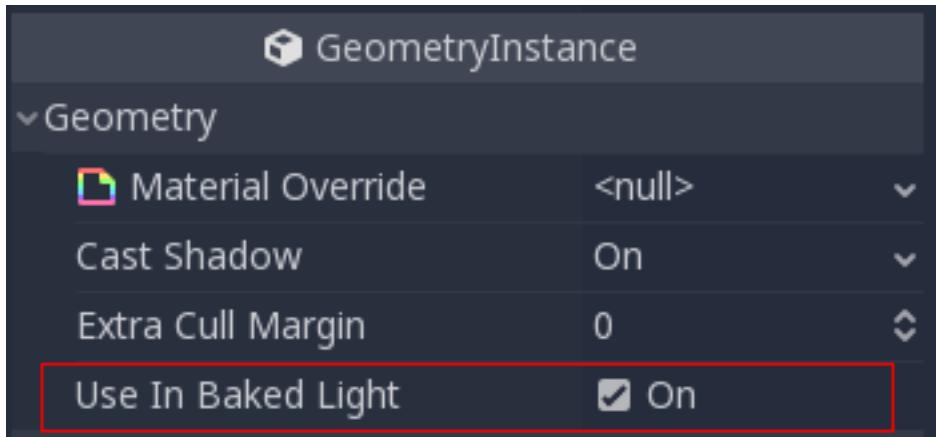
Configure Bounds

Lightmap needs an approximate volume of the area affected, because it uses it to transfer light to dynamic objects inside (more on that later). Just cover the scene with the volume, as you do with GIProbe:



Setting Up Meshes

For a **MeshInstance** node to take part in the baking process, it needs to have the “Use in Baked Light” property enabled.

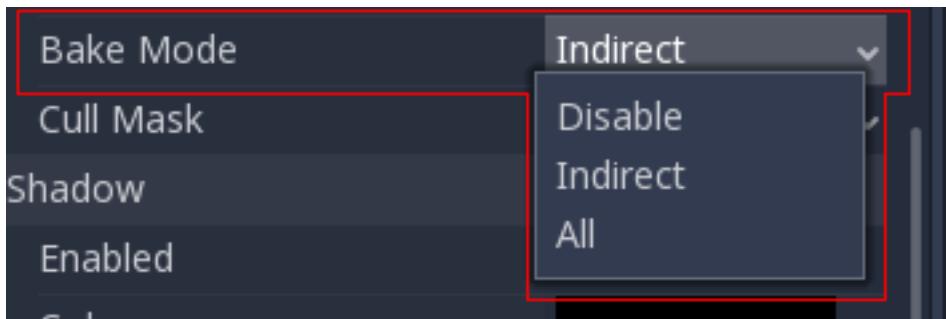


When auto-generating lightmaps on scene import, this is enabled automatically.

Setting up Lights

Lights are baked with indirect light by default. This means that shadowmapping and lighting are still dynamic and affect moving objects, but light bounces from that light will be baked.

Lights can be disabled (no bake), or be fully baked (direct and indirect), this can be controlled from the **Bake Mode** menu in lights:



The modes are :

- **Disabled:** Light is ignored in baking. Keep in mind hiding a light will have no effect for baking, so this must be used instead.
- **Indirect:** This is the default mode, only indirect lighting will be baked.
- **All:** Both indirect and direct lighting will be baked. If you don't want the light to appear twice (dynamically and statically), simply hide it.

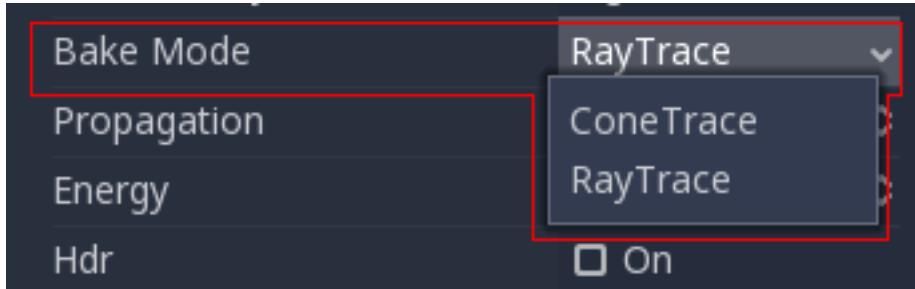
Baking Quality

BakedLightmap uses, for simplicity, a voxelized version of the scene to compute lighting. Voxel size can be adjusted with the **Bake Subdiv** parameter. More subdivision results in more detail, but also takes more time to bake.

In general, the defaults are good enough. There is also a **Capture Subdivision** (that must always be equal or less to the main subdivision), which is used for capturing light in dynamic objects (more on that later). Its default value is also good enough for more cases.

Bake Subdiv	256	▼
Capture Subdiv	128	▼

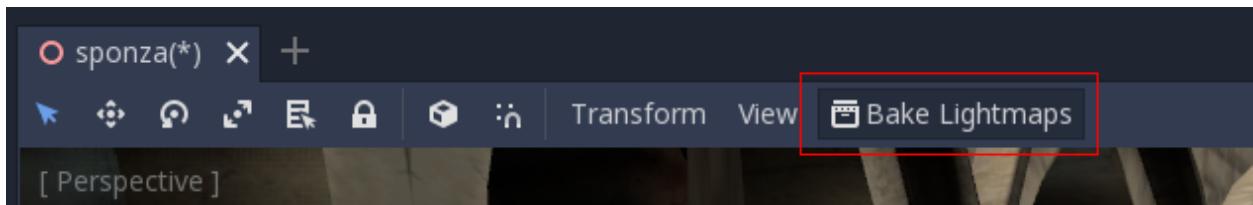
Besides the capture size, quality can be modified by setting the **Bake Mode**. Two modes of capturing indirect are provided:



- **Voxel Cone**: Trace: Is the default one, it's less precise but very fast. Look similar (but slightly better) to GIProbe.
- **Ray Tracing**: This method is more precise, but can take considerably longer to bake. If used in low or medium quality, some scenes may produce grain.

7.7.5 Baking

To begin the bake process, just push the big **Bake Lightmaps** button on top, when selecting the BakedLightmap node:



This can take from seconds to minutes (or hours) depending on scene size, bake method and quality selected.

Configuring Bake

Several more options are present for baking:

- **Bake Subdiv**: Godot lightmapper uses a grid to transfer light information around. The default value is fine and should work for most cases. Increase it in case you want better lighting on very small details or your scene is very large.
- **Capture Subdiv**: This is the grid used for real-time capture information (lighting dynamic objects). Default value is generally OK, it's usually smaller than Bake Subdiv and can't be larger than it.
- **Bake Quality**: Three bake quality modes are provided, Low, Medium and High. Each takes less and more time.
- **Bake Mode**: The baker can use two different techniques: *Voxel Cone Tracing* (fast but approximate), or *Ray-Tracing* (slow, but accurate).

- **Propagation:** Used for the *Voxel Cone Trace* mode, works just like in GIProbe.
- **HDR:** If disabled, lightmaps are smaller but can't capture any light over white (1.0).
- **Image Path:** Where lightmaps will be saved. By default, on the same directory as the scene (“.”), but can be tweaked.
- **Extents:** Size of the area affected (can be edited visually)
- **Light Data:** Contains the light baked data after baking. Textures are saved to disk, but this also contains the capture data for dynamic objects, which can be a bit heavy. If you are using .tscn formats (instead of .scn) you can save it to disk.

7.7.6 Dynamic Objects

In other engines or lightmapper implementations, you are required to manually place small objects called “lightprobes” all around the level to generate *capture* data. This is used to, then, transfer the light to dynamic objects that move around the scene.

This implementation of lightmapping uses a different method, so this process is automatic and you don't have to do anything. Just move your objects around and they will be lit accordingly. Of course, you have to make sure you set up your scene bounds accordingly or it won't work.

7.8 Environment and Post-Processing

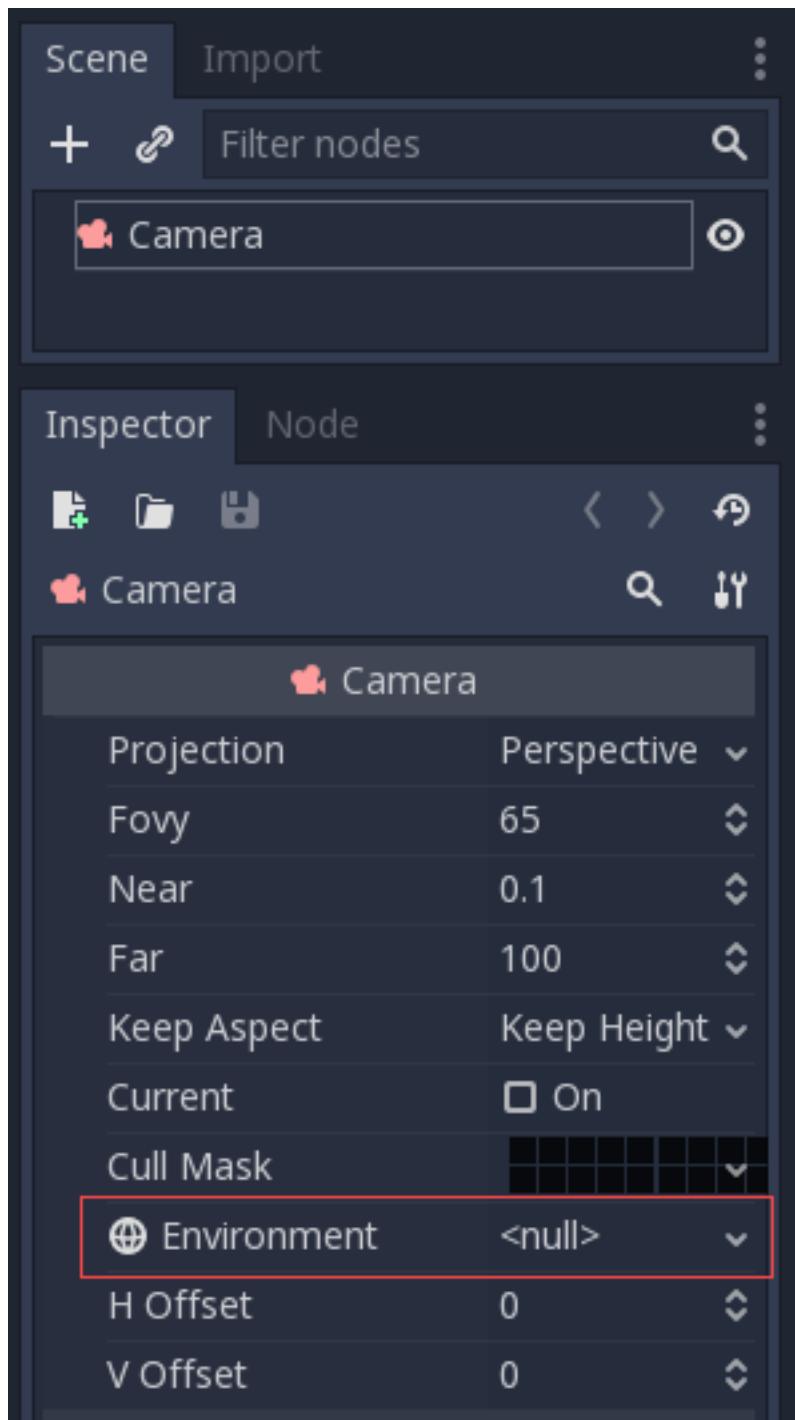
Godot 3 provides a redesigned Environment resource, as well as a brand new post-processing system with many available effects right out of the box.

7.8.1 Environment

The Environment resource stores all the information required for controlling rendering environment. This includes sky, ambient lighting, tone mapping, effects and adjustments. By itself it does nothing, but it becomes enabled once used in one of the following locations, in order of priority:

Camera Node

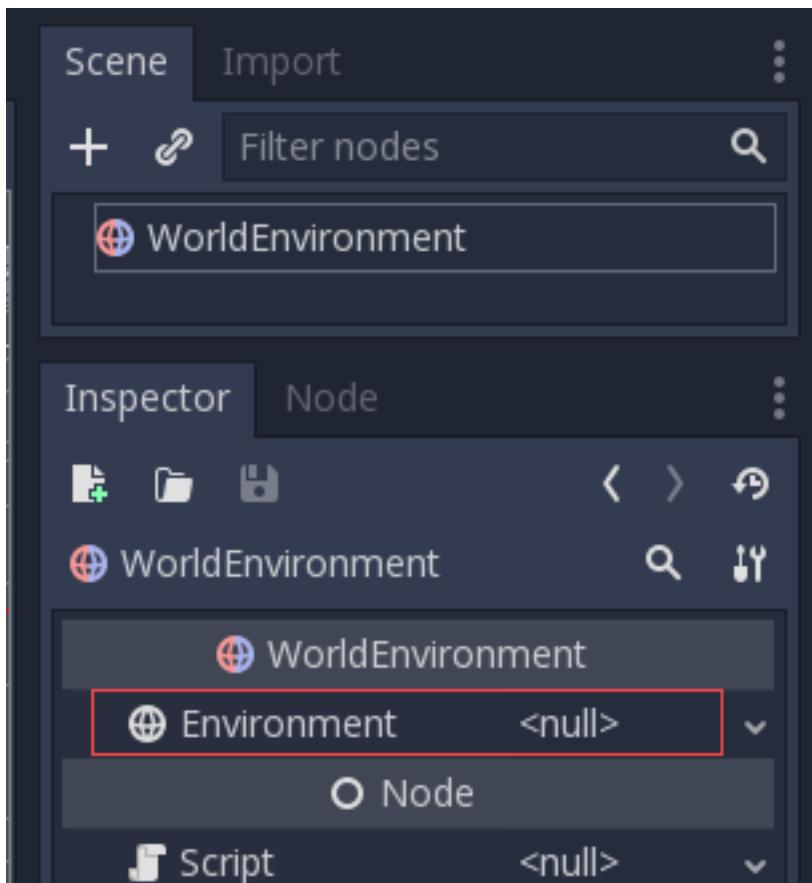
An Environment can be set to a camera. It will have priority over any other setting.



This is mostly useful when wanting to override an existing environment, but in general it's a better idea to use the option below.

WorldEnvironment Node

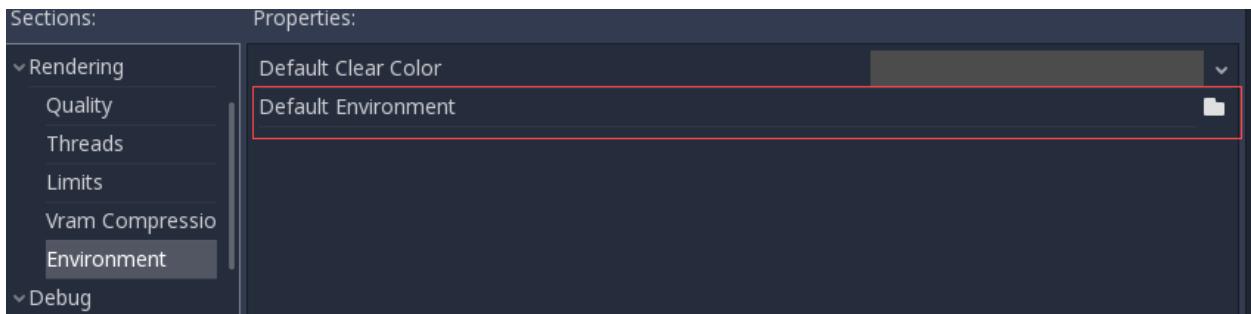
The WorldEnvironment node can be added to any scene, but only one can exist per active scene tree. Adding more than one will result in a warning.



Any Environment added has higher priority than the default Environment (explained below). This means it can be overridden on a per-scene basis, which makes it quite useful.

Default Environment

A default environment can be set, which acts as a fallback when no Environment was set to a Camera or WorldEnvironment. Just head to Project Settings -> Rendering -> Environment:



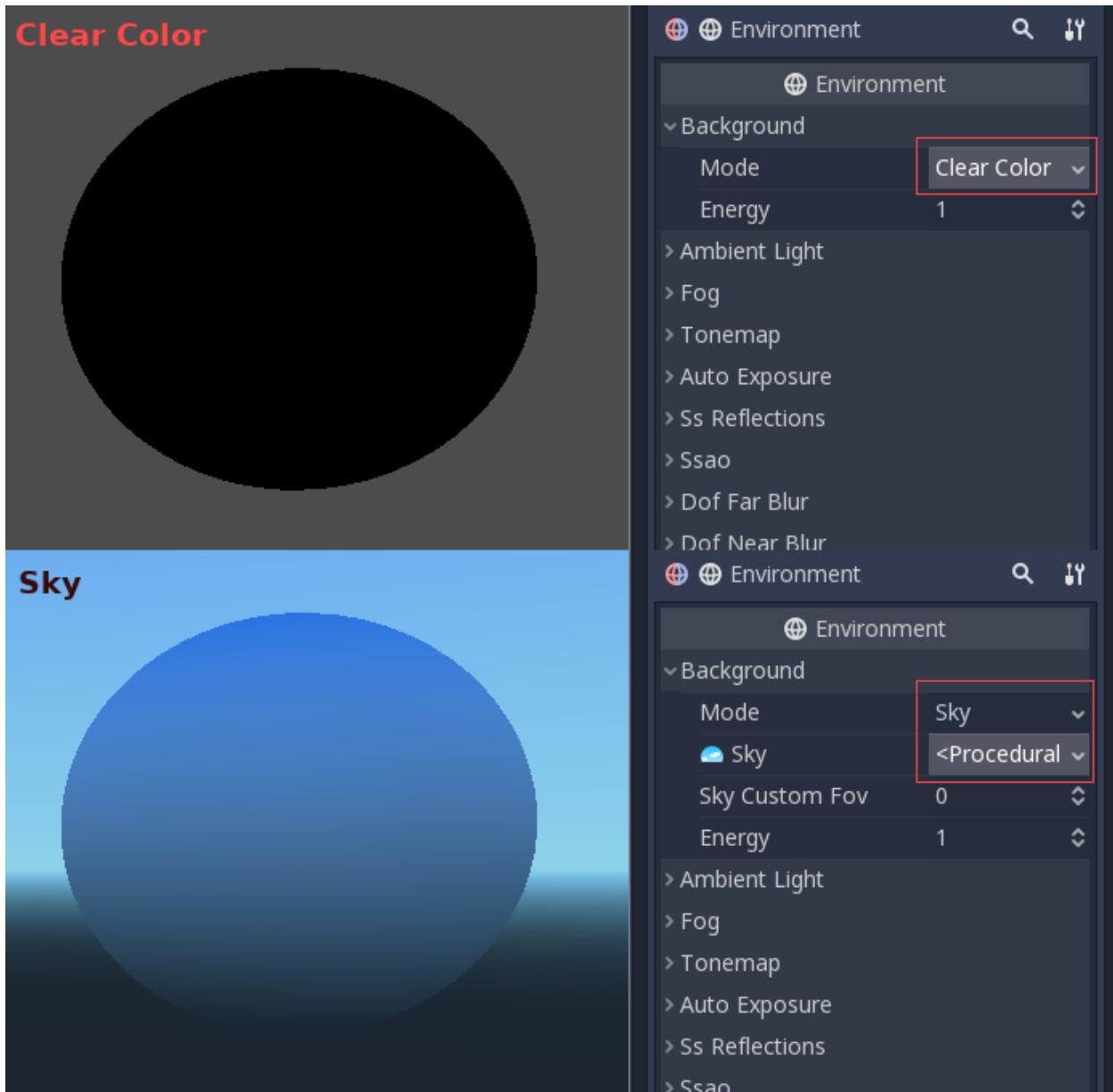
New projects created from the Project Manager come with a default environment (`default_env.tres`). If one needs to be created, save it to disk before referencing it here.

7.8.2 Environment Options

Following is a detailed description of all environment options and how they are intended to be used.

Background

The Background section contains settings on how to fill the background (parts of the screen where objects were not drawn). In Godot 3.0, the background not only serves the purpose of displaying an image or color, it can change how objects are affected by ambient and reflected light.



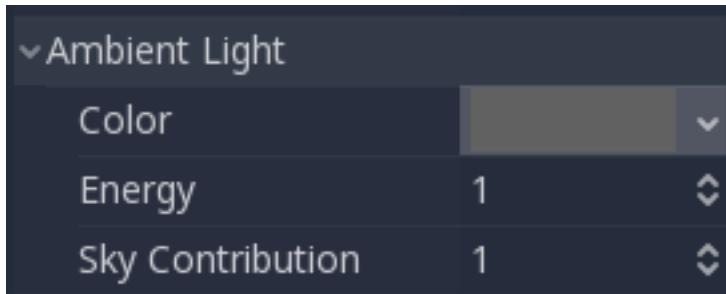
There are many ways to set the background:

- **Clear Color** uses the default clear color defined by the project. The background will be a constant color.
- **Custom Color** is like Clear Color, but with a custom color value.
- **Sky** lets you define a panorama sky (a 360 degree sphere texture) or a procedural sky (a simple sky featuring a gradient and an optional sun). Objects will reflect it and absorb ambient light from it.
- **Color+Sky** lets you define a sky (as above), but uses a constant color value for drawing the background. The sky will only be used for reflection and ambient light.

Ambient Light

Ambient (as defined here) is a type of light that affects every piece of geometry with the same intensity. It is global and independent of lights that might be added to the scene.

There are two types of ambient light, the *Ambient Color* (which is just a constant color multiplied by the material albedo), and then one obtained from the *Sky* (as described before, but a sky needs to be set as background for this to be enabled).



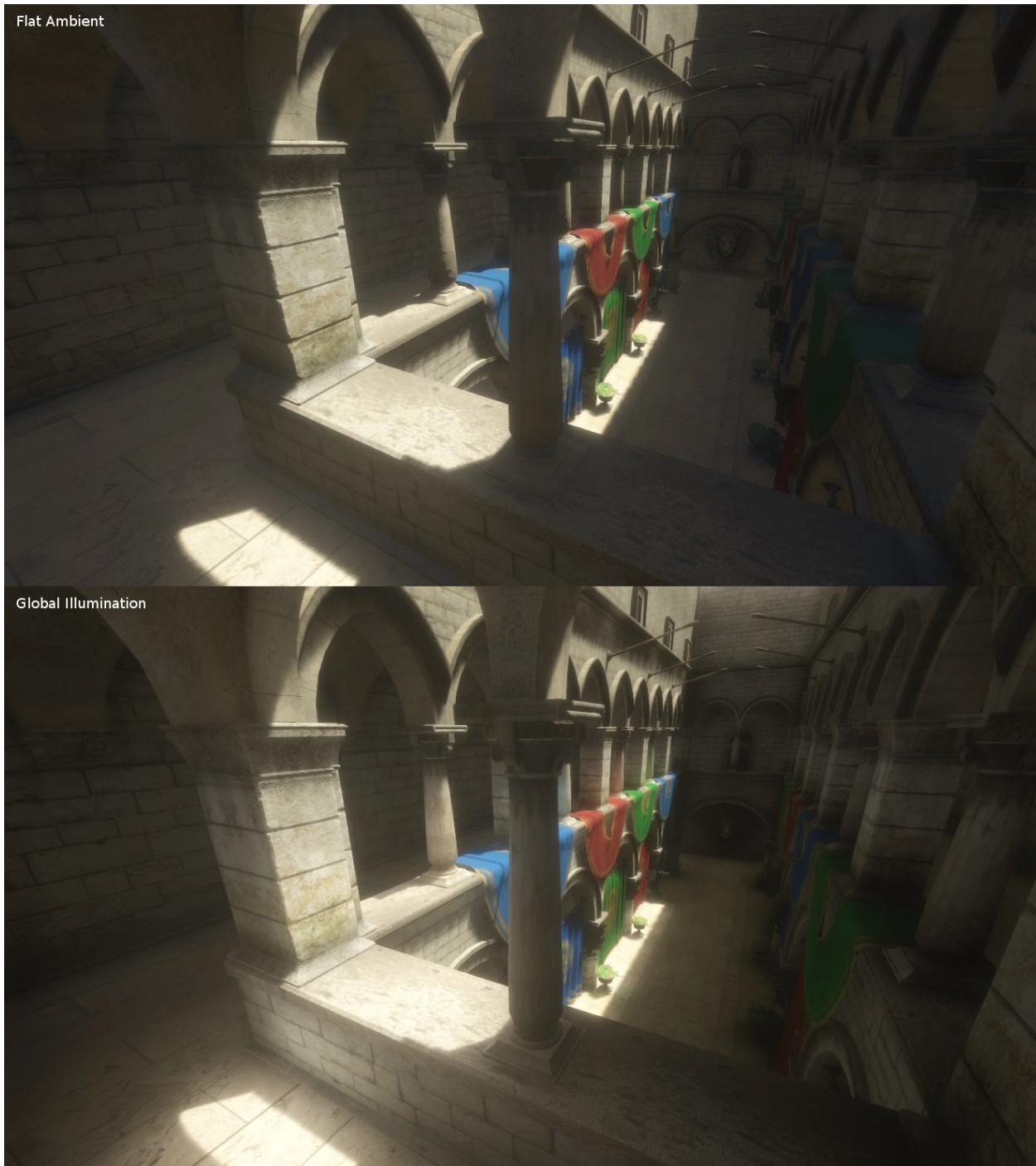
When a *Sky* is set as background, it's possible to blend between ambient color and sky using the **Sky Contribution** setting (this value is 1.0 by default for convenience, so only sky affects objects).

Here is a comparison of how different ambient light affects a scene:



Finally there is a **Energy** setting, which is just a multiplier, useful when working with HDR.

In general, ambient light should only be used for simple scenes, large exteriors or for performance reasons (ambient light is cheap), as it does not provide the best lighting quality. It's better to generate ambient light from ReflectionProbe or GIProbe, which will more faithfully simulate how indirect light propagates. Below is a comparison in quality between using a flat ambient color and a GIProbe:



Using one of the methods described above, objects get constant ambient lighting replaced by ambient light from the probes.

Fog

Fog, just as in real life, makes distant objects fade away into an uniform color. The physical effect is actually pretty complex, but Godot provides a good approximation. There are two kinds of fog in Godot:

- **Depth Fog:** This one is applied based on the distance from the camera.

- **Height Fog:** This one is applied to any objects below (or above) a certain height, regardless of the distance from the camera.



Both of these fog types can have their curve tweaked, making their transition more or less sharp.

Two properties can be tweaked to make the fog effect more interesting:

The first is **Sun Amount**, which makes use of the Sun Color property of the fog. When looking towards a directional light (usually a sun), the color of the fog will be changed, simulating the sunlight passing through the fog.

The second is **Transmit Enabled** which simulates more realistic light transmittance. In practice, it makes light stand out more across the fog.



Tonemap

Selects the tone-mapping curve that will be applied to the scene, from a short list of standard curves used in the film and game industry. Tone mapping can make light and dark areas more homogeneous, even though the result is not that strong. Tone mapping options are:

- **Mode:** Tone mapping mode, which can be Linear, Reinhart, Filmic or Aces.
- **Exposure:** Tone mapping exposure, which simulates amount of light received over time.
- **White:** Tone mapping white, which simulates where in the scale is white located (by default 1.0).

Auto Exposure (HDR)

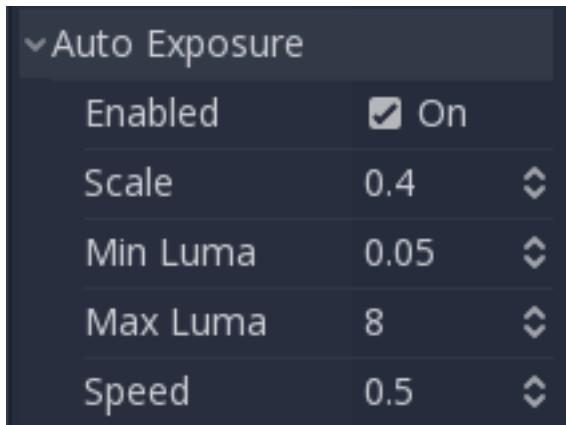
Even though, in most cases, lighting and texturing are heavily artist controlled, Godot supports a simple high dynamic range implementation with auto exposure mechanism. This is generally used for the sake of realism, when combining interior areas with low light and outdoors. Auto exposure simulates the camera (or eye) effort to adapt between light and dark locations and their different amount of light.

The simplest way to use auto exposure is to make sure outdoor lights (or other strong lights) have energy beyond 1.0. This is done by tweaking their **Energy** multiplier (on the Light itself). To make it consistent, the **Sky** usually needs to use the energy multiplier too, to match with the directional light. Normally, values between 3.0 and 6.0 are enough to simulate indoor-outdoor conditions.

By combining Auto Exposure with *Glow* post processing (more on that below), pixels that go over the tonemap **White** will bleed to the glow buffer, creating the typical bloom effect in photography.



The user-controllable values in the Auto Exposure section come with sensible defaults, but you can still tweak them:



- **Scale:** Value to scale the lighting. Brighter values produce brighter images, smaller ones produce darker ones.
- **Min Luma:** Minimum luminance that auto exposure will aim to adjust for. Luminance is the average of the light in all the pixels of the screen.
- **Max Luma:** Maximum luminance that auto exposure will aim to adjust for.
- **Speed:** Speed at which luminance corrects itself. The higher the value, the faster correction happens.

7.8.3 Mid and Post-Processing Effects

A large amount of widely-used mid and post-processing effects are supported in Environment.

Screen-Space Reflections (SSR)

While Godot supports three sources of reflection data (Sky, ReflectionProbe and GIProbe), they may not provide enough detail for all situations. Scenarios where Screen Space Reflections make the most sense are when objects are in contact with each other (object over floor, over a table, floating on water, etc).



The other advantage (even if only enabled to a minimum), is that it works in real-time (while the other types of reflections are pre-computed). This is great to make characters, cars, etc. reflect when moving around.

A few user-controlled parameters are available to better tweak the technique:

- **Max Steps** determines the length of the reflection. The bigger this number, the more costly it is to compute.
- **Fade In** allows adjusting the fade-in curve, which is useful to make the contact area softer.
- **Fade Out** allows adjusting the fade-out curve, so the step limit fades out softly.

- **Depth Tolerance** can be used for screen-space-ray hit tolerance to gaps. The bigger the value, the more gaps will be ignored.
- **Roughness** will apply a screen-space blur to approximate roughness in objects with this material characteristic.

Keep in mind that screen-space-reflections only work for reflecting opaque geometry. Transparent objects can't be reflected.

Screen-Space Ambient Occlusion (SSAO)

As mentioned in the **Ambient** section, areas where light from light nodes does not reach (either because it's outside the radius or shadowed) are lit with ambient light. Godot can simulate this using GIProbe, ReflectionProbe, the Sky or a constant ambient color. The problem, however, is that all the methods proposed before act more on larger scale (large regions) than at the smaller geometry level.

Constant ambient color and Sky are uniform and the same everywhere, while GI and Reflection probes have more local detail, but not enough to simulate situations where light is not able to fill inside hollow or concave features.

This can be simulated with Screen Space Ambient Occlusion. As you can see in the image below, the goal of it is to make sure concave areas are darker, simulating a narrower path for the light to enter:



It is a common mistake to enable this effect, turn on a light and not be able to appreciate it. This is because SSAO only acts on *ambient* light, not direct light.

This is why, in the image above, the effect is less noticeable under the direct light (at the left). If you want to force SSAO to work with direct light too, just use the **Light Affect** parameter (even though this is not correct, some artists like how it looks).

SSAO looks best when combined with a real source of indirect light, like GIProbe:



Tweaking SSAO is possible with several parameters:

Ssao		
Enabled	<input checked="" type="checkbox"/> On	
Radius	6.3	▼
Intensity	1.5	▼
Radius 2	0.2	▼
Intensity 2	0.5	▼
Bias	0.01	▼
Light Affect	0	▼
Color	<div style="background-color: black; width: 100px; height: 20px;"></div>	▼
Quality	Low	▼
Blur	3x3	▼
Edge Sharpness	4	▼

- **Radius/Intensity:** To control the radius or intensity of the occlusion, these two parameters are available. Radius

is in world (Metric) units.

- **Radius2/Intensity2:** A Secondary radius/intensity can be used. Combining a large and a small radius AO generally works well.
- **Bias:** This can be tweaked to solve self occlusion, though the default generally works well enough.
- **Light Affect:** SSAO only affects ambient light, but increasing this slider can make it also affect direct light. Some artists prefer this effect.
- **Quality:** Depending on quality, SSAO will do more samplings over a sphere for every pixel. High quality only works well on modern GPUs.
- **Blur:** Type of blur kernel used. The 1x1 kernel is a simple blur that preserves local detail better, but is not as efficient (generally works better with high quality setting above), while 3x3 will soften the image better (with a bit of dithering-like effect), but does not preserve local detail as well.
- **Edge Sharpness:** This can be used to preserve the sharpness of edges (avoids areas without AO on creases).

Depth of Field / Far Blur

This effect simulates focal distance on high end cameras. It blurs objects behind a given range. It has an initial **Distance** with a **Transition** region (in world units):



The **Amount** parameter controls the amount of blur. For larger blurs, tweaking the **Quality** may be needed in order to avoid artifacts.

Depth of Field / Near Blur

This effect simulates focal distance on high end cameras. It blurs objects close to the camera (acts in the opposite direction as far blur). It has an initial **Distance** with a **Transition** region (in world units):



The **Amount** parameter controls the amount of blur. For larger blurs, tweaking the **Quality** may be needed in order to avoid artifacts.

It is very common to use both blurs together to focus the viewer's attention on a given object:



Glow

In photography and film, when light amount exceeds the maximum supported by the media (be it analog or digital), it generally bleeds outwards to darker regions of the image. This is simulated in Godot with the **Glow** effect.

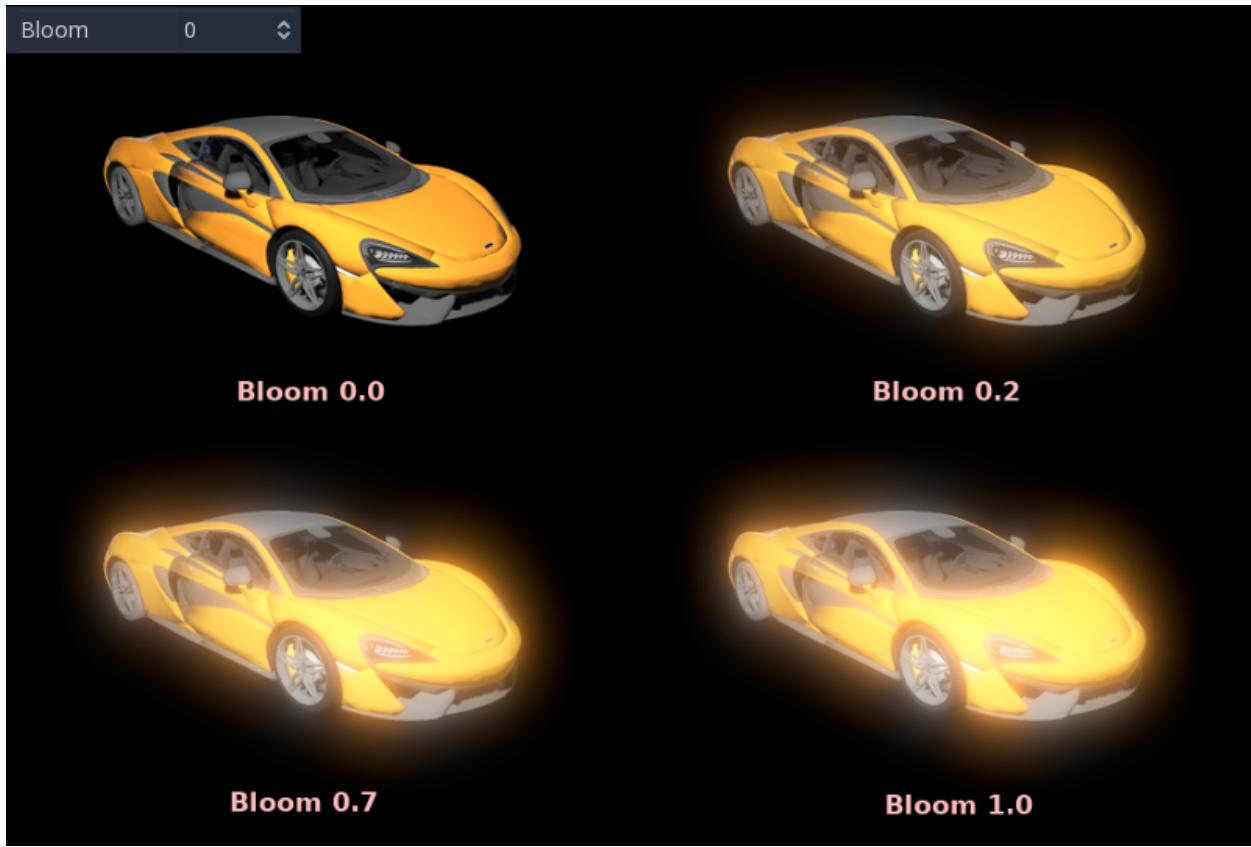


By default, even if the effect is enabled, it will be very weak or invisible. One of two conditions need to happen for it to actually show:

- 1. The light in a pixel surpasses the **HDR Threshold** (where 0 is all light surpasses it, and 1.0 is light over the tonemapper **White** value). Normally this value is expected to be at 1.0, but it can be lowered to allow more light to bleed. There is also an extra parameter, **HDR Scale** that allows scaling (making brighter or darker) the light surpassing the threshold.



- 2. The Bloom effect has a value set greater than 0. As it increases, it sends the whole screen to the glow processor at higher amounts.



Both will cause the light to start bleeding out of the brighter areas.

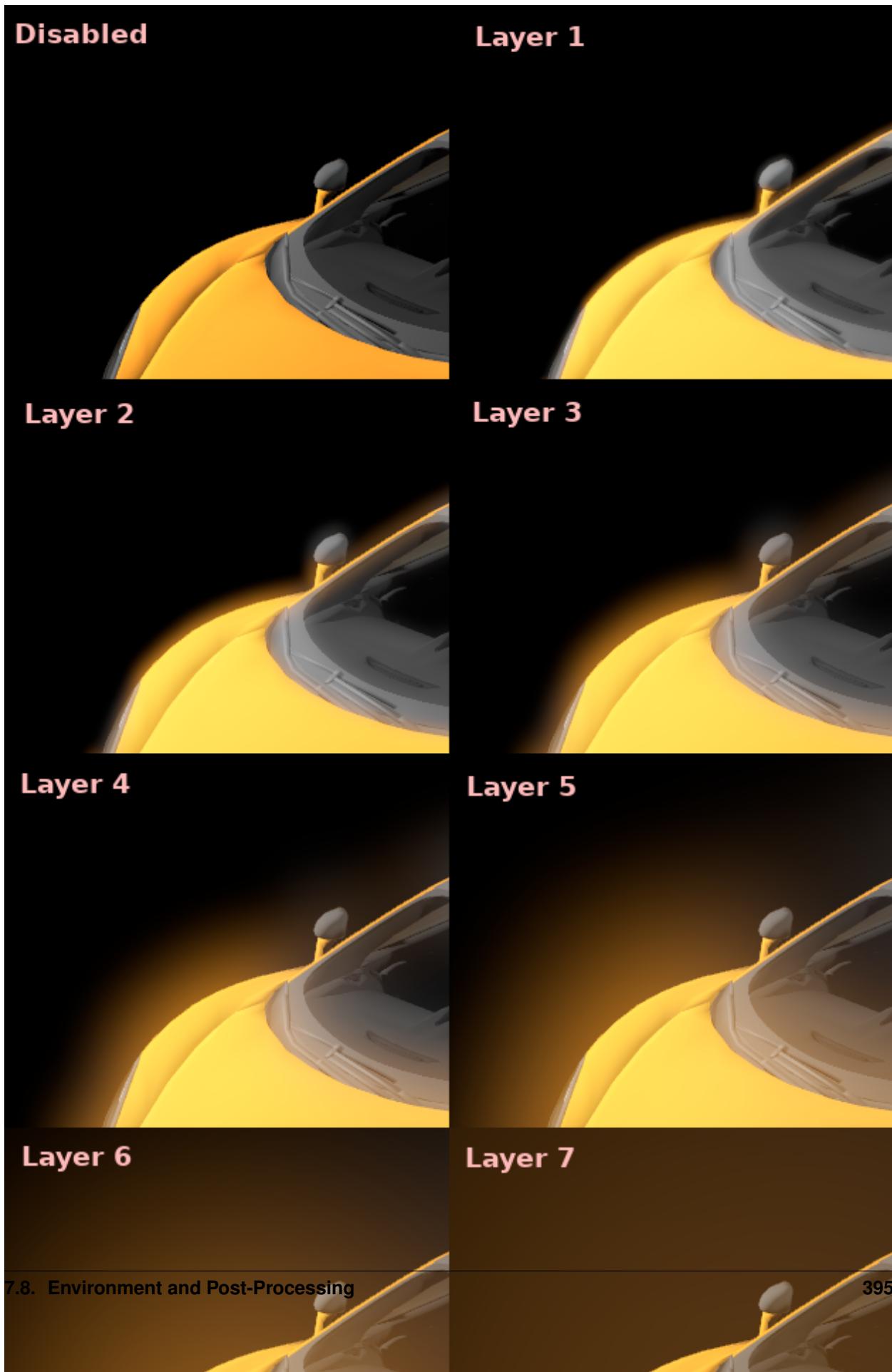
Once glow is visible, it can be controlled with a few extra parameters:

- **Intensity** is an overall scale for the effect, it can be made stronger or weaker (0.0 removes it).
- **Strength** is how strong the gaussian filter kernel is processed. Greater values make the filter saturate and expand outwards. In general changing this is not needed, as the size can be more efficiently adjusted with the **Levels**.

The **Blend Mode** of the effect can also be changed:

- **Additive** is the strongest one, as it just adds the glow effect over the image with no blending involved. In general, it's too strong to be used, but can look good with low intensity Bloom (produces a dream-like effect).
- **Screen** is the default one. It ensures glow never brights more than itself, and works great as an all around.
- **Softlight** is the weakest one, producing only a subtle color disturbance around the objects. This mode works best on dark scenes.
- **Replace** can be used to blur the whole screen or debug the effect. It just shows the glow effect without the image below.

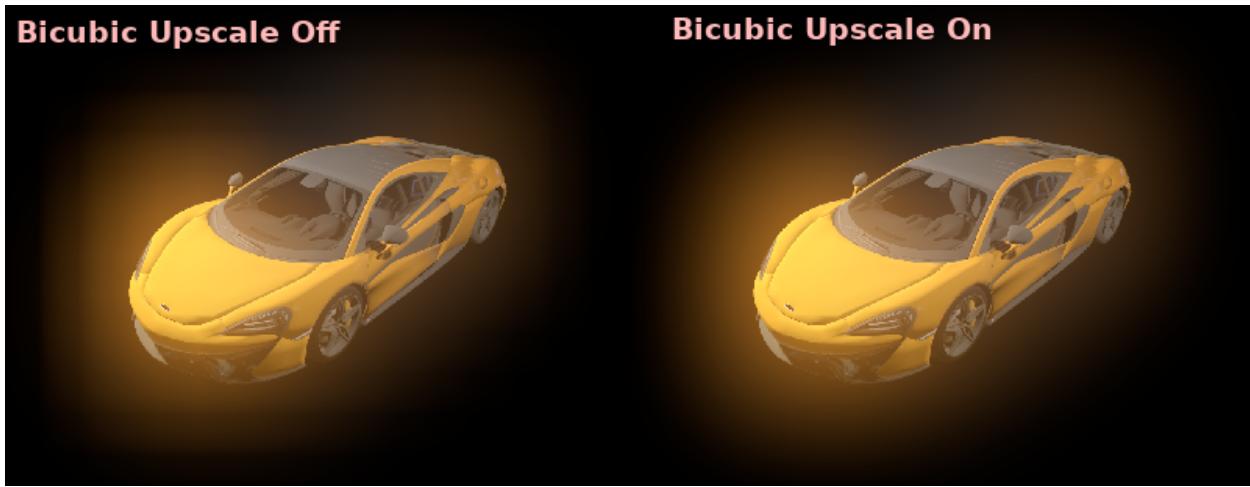
To change the glow effect size and shape, Godot provides **Levels**. Smaller levels are strong glows that appear around objects, while large levels are hazy glows covering the whole screen:



The real strength of this system, though, is to combine levels to create more interesting glow patterns:

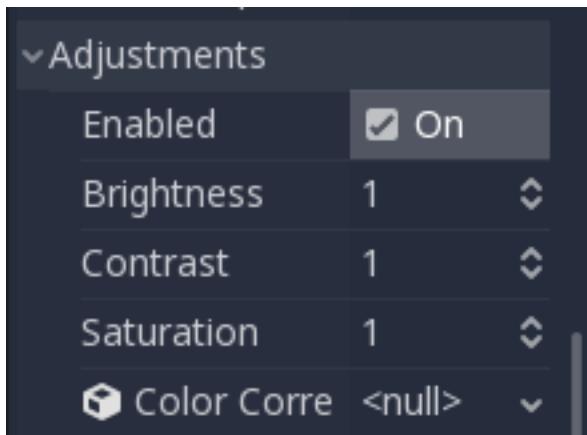


Finally, as the highest layers are created by stretching small blurred images, it is possible that some blockyness may be visible. Enabling **Bicubic Upscaling** gets rid of the it, at a minimal performance cost.



Adjustments

At the end of processing, Godot offers the possibility to do some standard image adjustments.

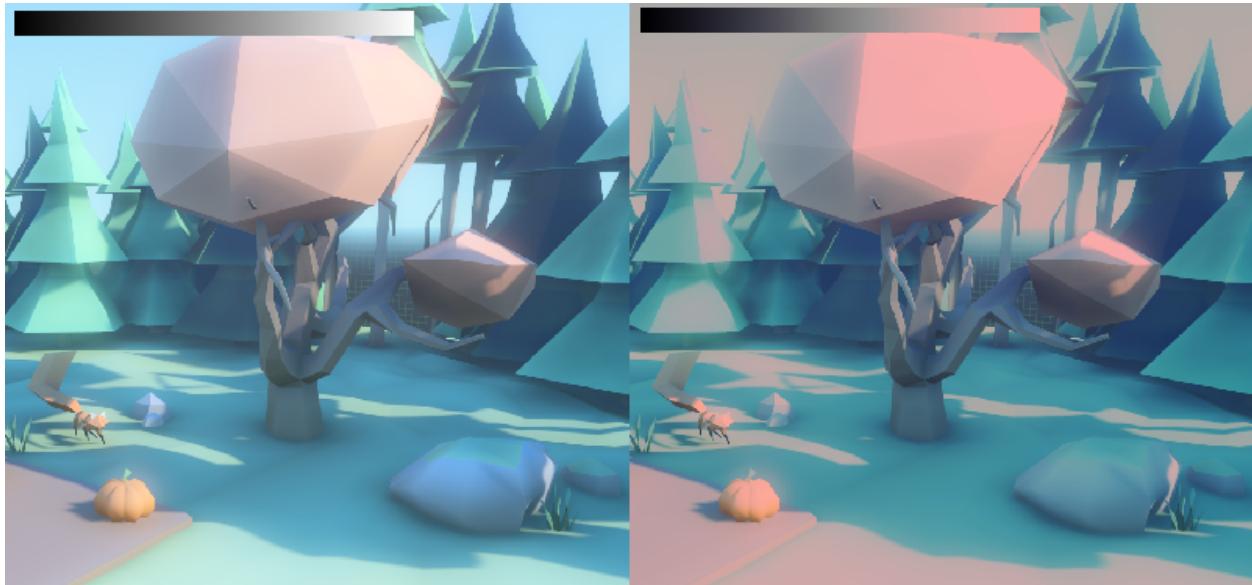


The first one is being able to change the typical Brightness, Contrast and Saturation:



The second is by supplying a color correction gradient. A regular black to white gradient like the following one will produce no effect:

But creating custom ones will allow to map each channel to a different color:



7.9 High dynamic range

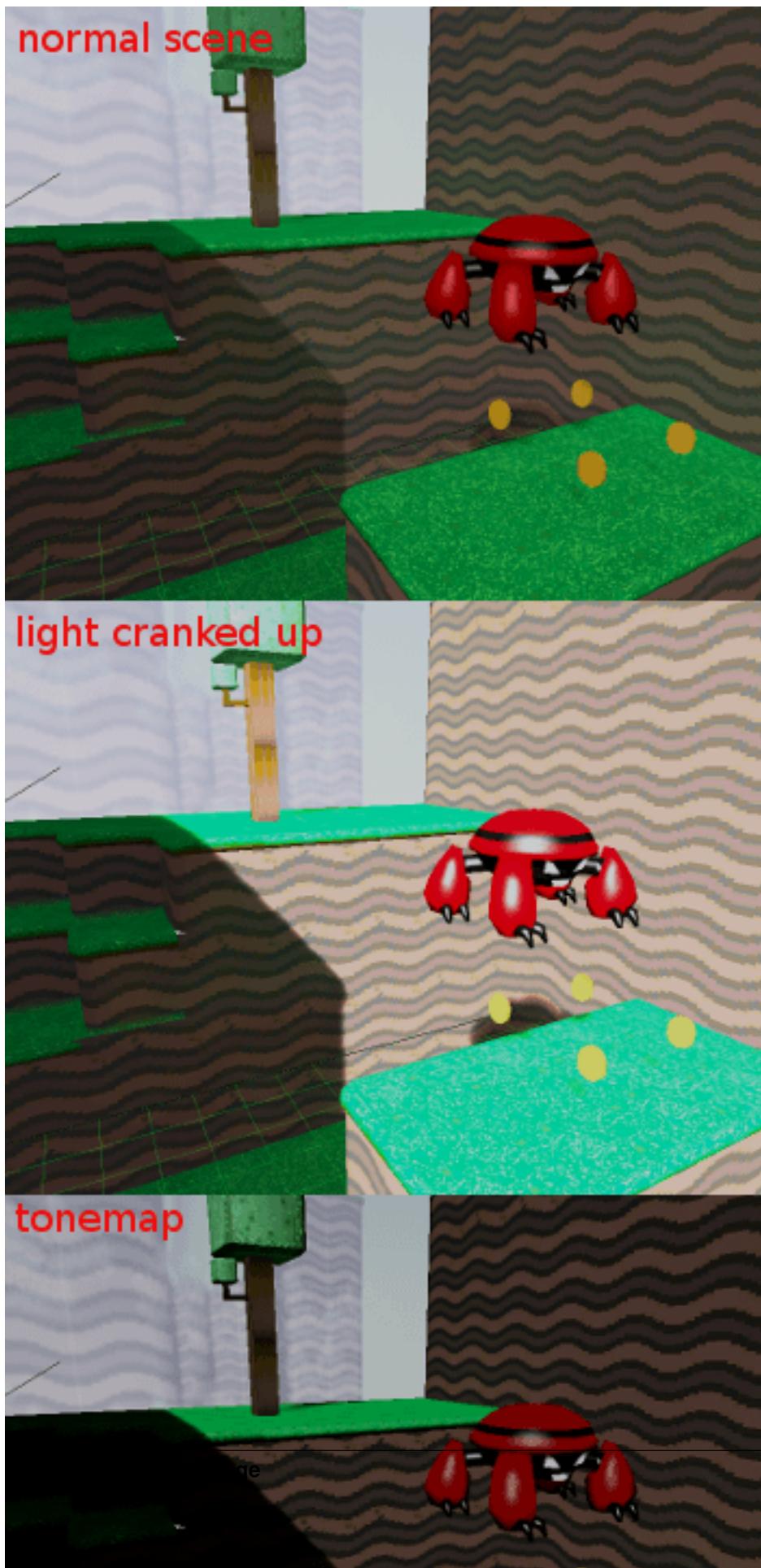
7.9.1 Introduction

Normally, an artist does all the 3D modelling, then all the texturing, looks at their awesome looking model in the 3D DCC and says “looks fantastic, ready for integration!” then goes into the game, lighting is setup and the game runs.

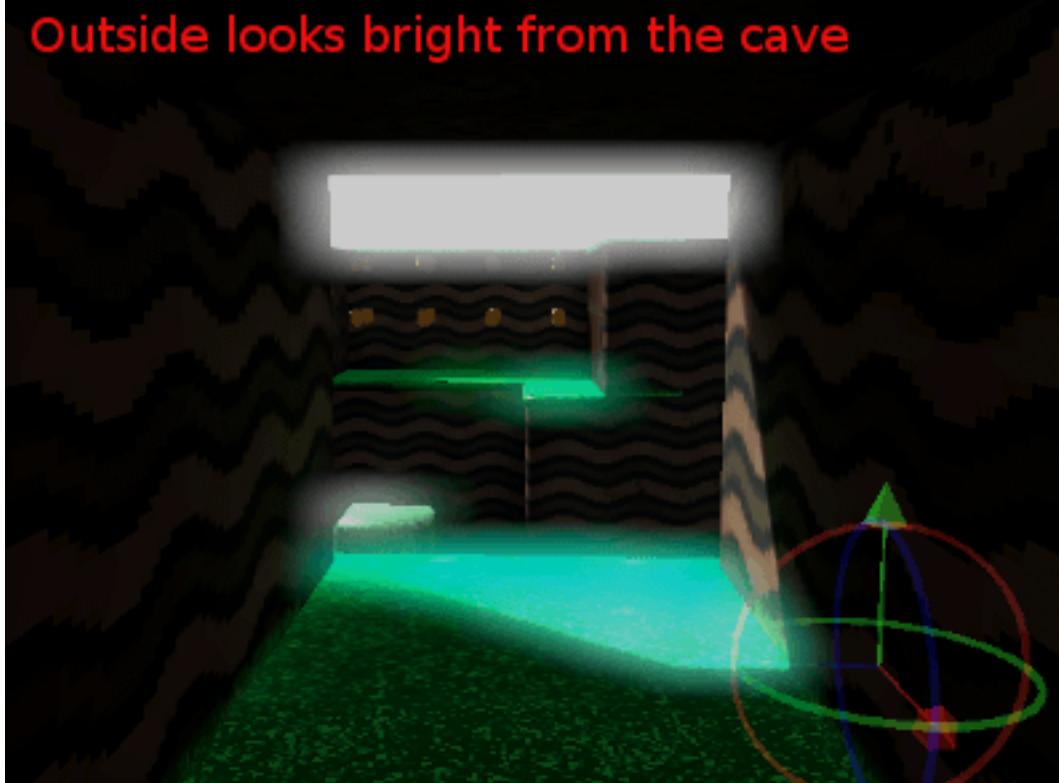
So where does all this HDR stuff thing come from? The idea is that instead of dealing with colors that go from black to white (0 to 1), we use colors whiter than white (for example, 0 to 8 times white).

To be more practical, imagine that in a regular scene, the intensity of a light (generally 1.0) is set to 5.0. The whole scene will turn very bright (towards white) and look horrible.

After this the luminance of the scene is computed by averaging the luminance of every pixel of it, and this value is used to bring the scene back to normal ranges. This last operation is called tone-mapping. Finally, we are at a similar place from where we started:



Except the scene is more contrasted, because there is a higher light range in play. What is this all useful for? The idea is that the scene luminance will change while you move through the world, allowing situations like this to happen:

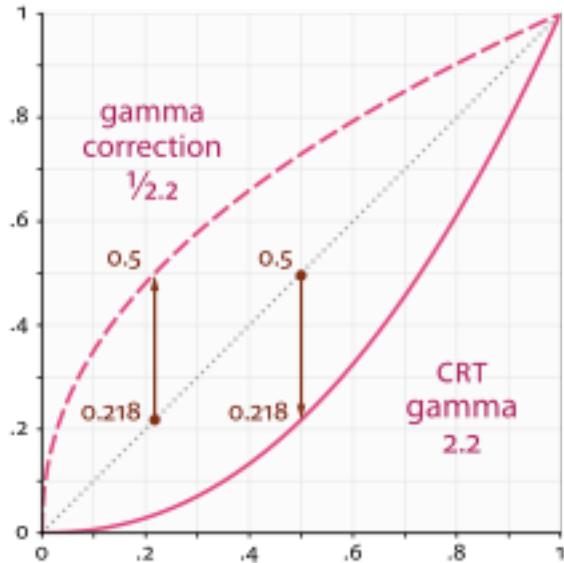


Additionally, it is possible to set a threshold value to send to the glow buffer depending on the pixel luminance. This allows for more realistic light bleeding effects in the scene.

7.9.2 Linear color space

The problem with this technique is that computer monitors apply a gamma curve to adapt better to the way the human eye sees. Artists create their art on the screen too, so their art has an implicit gamma curve applied to it.

The color space where images created in computer monitors exist is called “sRGB”. All visual content that people have on their computers or download from the internet (such as pictures, movies, etc.) is in this colorspace.



The mathematics of HDR require that we multiply the scene by different values to adjust the luminance and exposure to different light ranges, and this curve gets in the way as we need colors in linear space for this.

7.9.3 Linear color space & asset pipeline

Working in HDR is not just pressing a switch. First, imported image assets must be converted to linear space on import. There are two ways to do this:

SRGB -> linear conversion on image import

This is the most compatible way of using linear-space assets and it will work everywhere including all mobile devices. The main issue with this is loss of quality, as sRGB exists to avoid this same problem. Using 8 bits per channel to represent linear colors is inefficient from the point of view of the human eye. These textures might be later compressed too, which makes the problem worse.

In any case though, this is the easy solution that works everywhere.

Hardware sRGB -> linear conversion.

This is the most correct way to use assets in linear-space, as the texture sampler on the GPU will do the conversion after reading the texel using floating point. This works fine on PC and consoles, but most mobile devices do not support it, or do not support it on compressed texture format (iOS for example).

Linear -> sRGB at the end.

After all the rendering is done, the linear-space rendered image must be converted back to sRGB. To do this, simply enable sRGB conversion in the current *Environment* (more on that below).

Keep in mind that sRGB -> Linear and Linear -> sRGB conversions must always be **both** enabled. Failing to enable one of them will result in horrible visuals suitable only for avant garde experimental indie games.

7.9.4 Parameters of HDR

HDR is found in the *Environment* resource. These are found most of the time inside a *WorldEnvironment* node, or set in a camera. There are many parameters for HDR:



ToneMapper

The ToneMapper is the heart of the algorithm. Many options for tonemappers are provided:

- Linear: Simplest tonemapper. It does its job for adjusting scene brightness, but if the differences in light are too big, it will cause colors to be too saturated.
- Log: Similar to linear, but not as extreme.
- Reinhardt: Classical tonemapper (modified so it will not desaturate as much)
- ReinhardtAutoWhite: Same as above, but uses the max scene luminance to adjust the white value.

Exposure

The same exposure parameter as in real cameras. Controls how much light enters the camera. Higher values will result in a brighter scene and lower values will result in a darker scene.

White

Maximum value of white.

Glow threshold

Determine above which value (from 0 to 1 after the scene is tonemapped), light will start bleeding.

Glow scale

Determine how much light will bleed.

Min luminance

Lower bound value of light for the scene at which the tonemapper stops working. This allows dark scenes to remain dark.

Max luminance

Upper bound value of light for the scene at which the tonemapper stops working. This allows bright scenes to remain saturated.

Exposure adjustment speed

Auto-exposure will change slowly and will take a while to adjust (like in real cameras). Bigger values means faster adjustment.

7.10 Using gridmaps

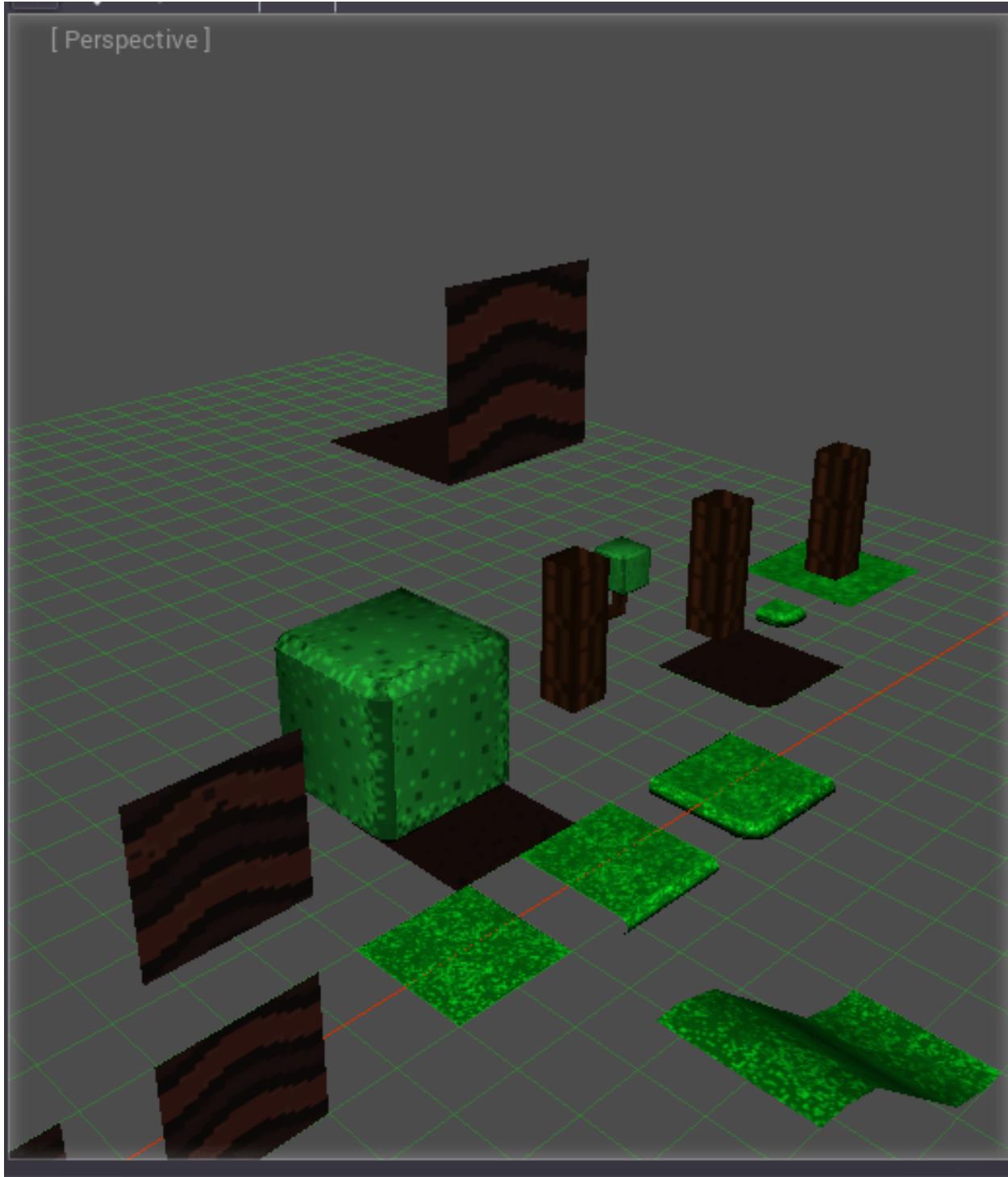
7.10.1 Introduction

Gridmaps are a simple and fast way to create 3D game levels. Think of it as a 3D version of the *TileMap* node. Similarly, you start with a predefined library of 3D meshes that can be put on a grid, just like if you were building a level with an unlimited amount of Lego blocks.

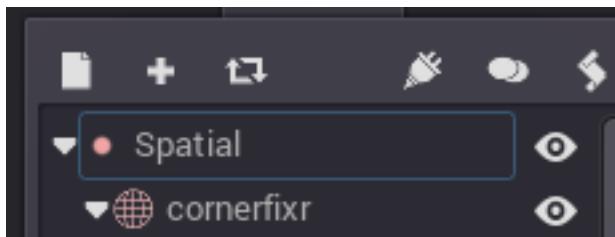
Collisions can also be added to the meshes, just like you would do with the tiles of a tilemap.

7.10.2 Creating a MeshLibrary

To begin, you need a *MeshLibrary*, which is a collection of meshes that can be used in the gridmap. Here are some meshes you can use to set it up.



Open a new scene and create a root node (this is important, as without the root node, it will not be able to generate the `MeshLibrary`!). Then, create a `MeshInstance` node:

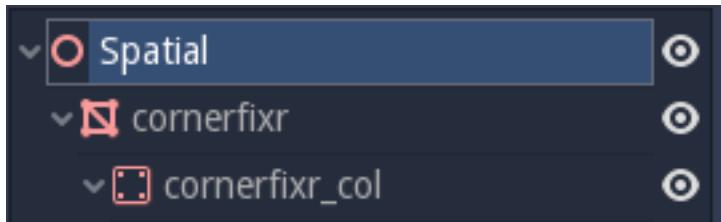


If you don't need to apply physics to the building blocks, that's all you need to do. In most cases though, you will need your block to generate collisions, so let's see how to add them.

7.10.3 Collisions

To assign a *CollisionShape* and *PhysicsBody* to the meshes, the simplest way is to do it while creating the MeshLibrary. Alternatively, you can also edit an existing MeshLibrary from within the GridMap inspector, but only CollisionShapes can be defined there and not PhysicsBody.

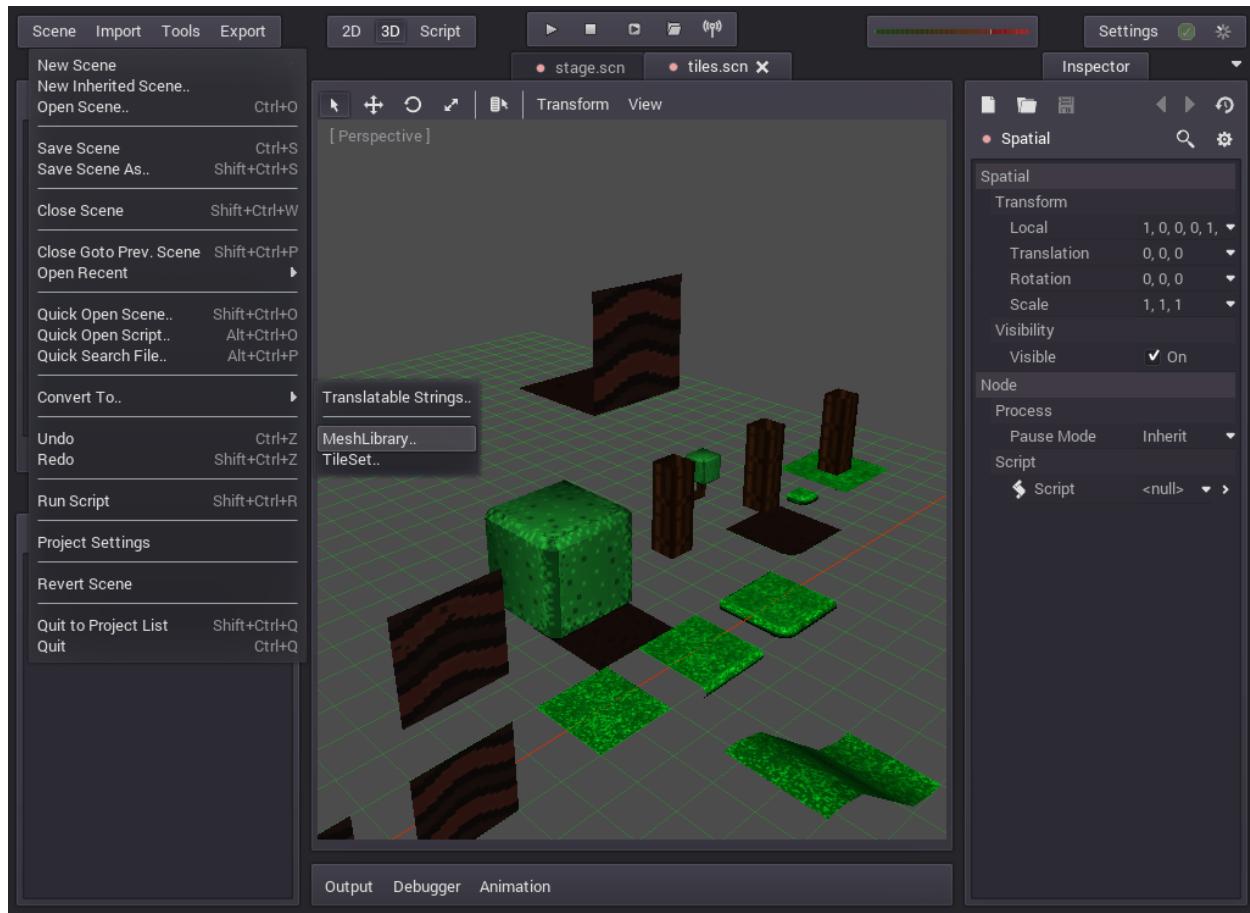
To give the meshes a CollisionShape, you simply add children nodes to the MeshInstance node. You would typically add the desired PhysicsBody and CollisionShape in this order:



You can adjust the order according to your needs.

7.10.4 Exporting the MeshLibrary

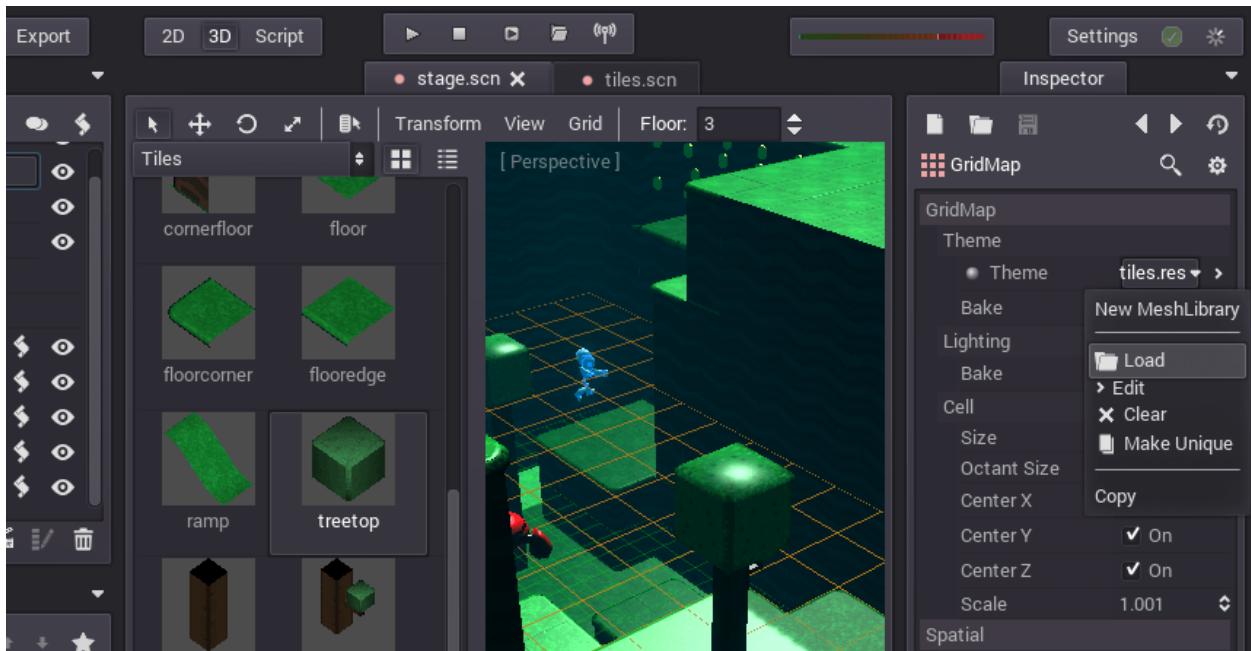
To export, go to Scene > Convert To... > MeshLibrary..., and save it as a resource.



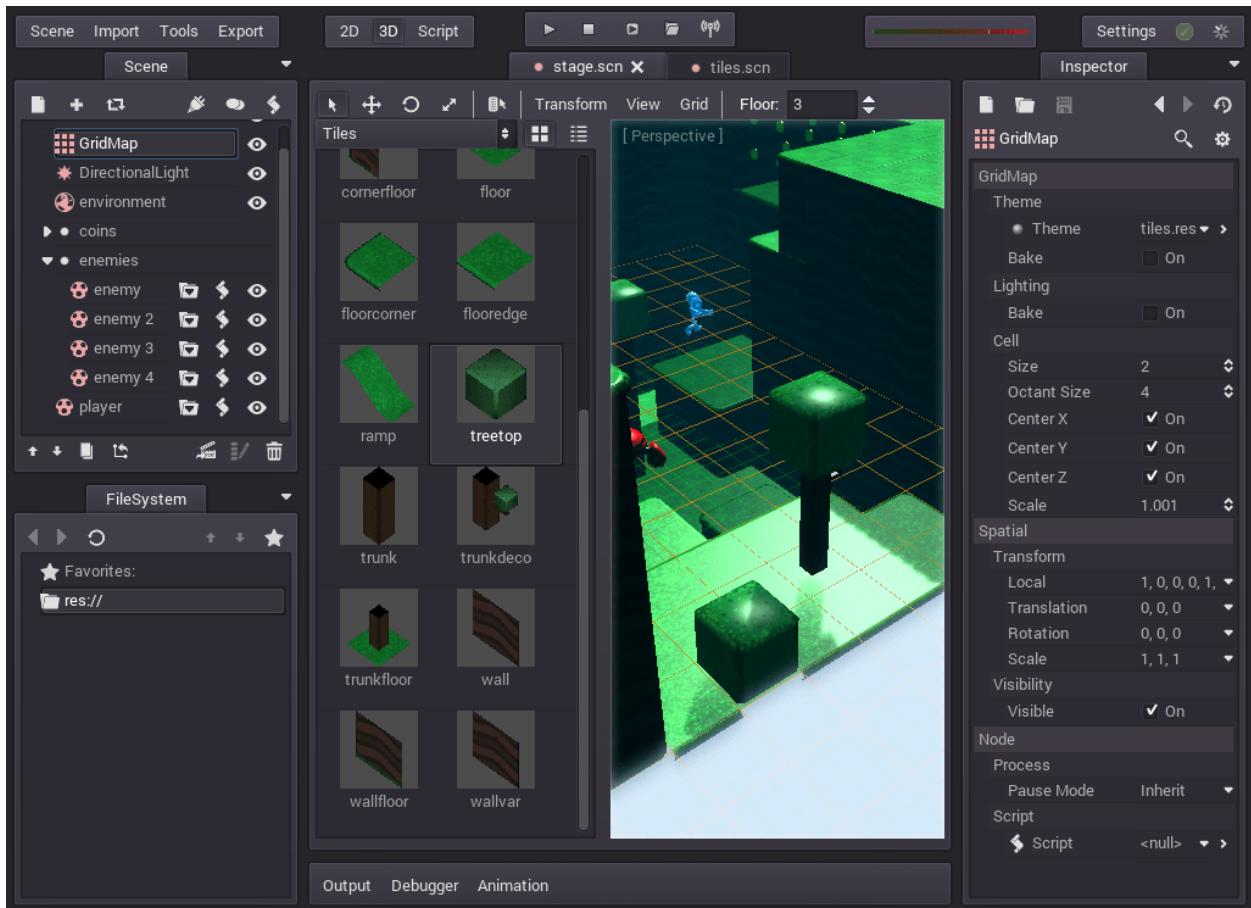
You are now ready to use the GridMap node.

7.10.5 Using the MeshLibrary in a GridMap

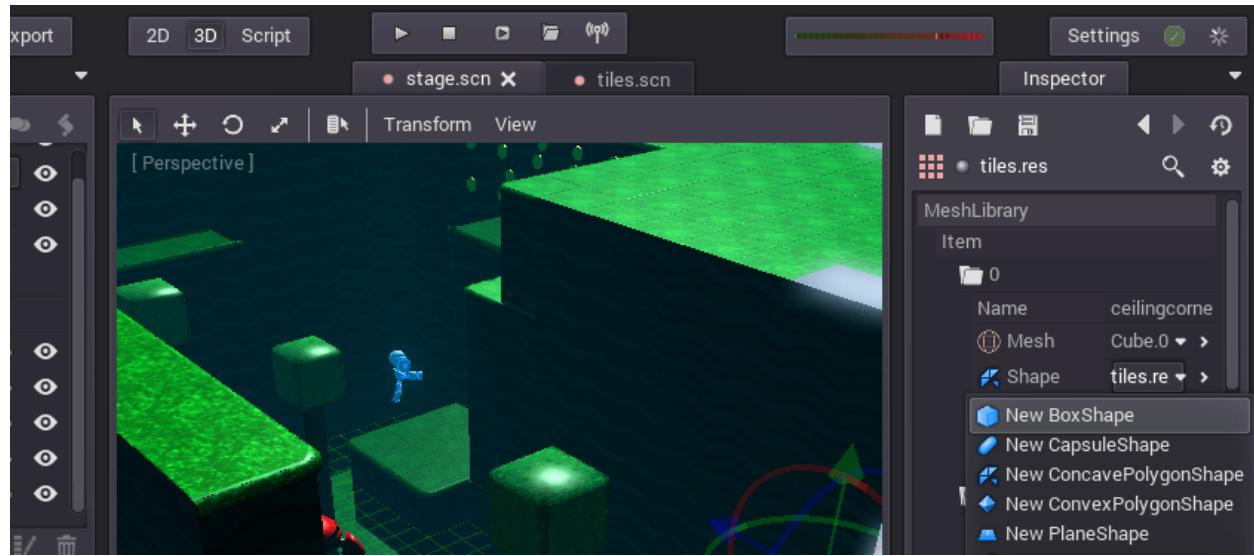
Create a new scene using any node as root, then add a Gridmap node. Then, load the MeshLibrary that you just exported.



Now, you can build your own level as you see best fit. Use left click to add tiles and right click to remove them. You can adjust the floor level when you need to put meshes at specific heights.



As mentioned above, you can also define new CollisionShapes at this stage by doing the following steps:



There you are!

7.10.6 Reminder

- Be cautious before scaling meshes if you are not using uniform meshes.
- There are many ways to make use of gridmaps, be creative!

7.11 Working with 3D skeletons

Godot 3D skeleton support is currently quite rudimentary. The *Skeleton* node and class were designed mainly to support importing skeletal animations as a set of transformation matrices.

7.11.1 Skeleton node

Skeleton node can be directly added anywhere you want on scene. Usually mesh is a child of *Skeleton*, as it easier to manipulate this way, as Transforms within skeleton are relative to where *Skeleton* is. But you can specify *Skeleton* node in every *MeshInstance*.

Being obvious, *Skeleton* is intended to deform meshes, and consists of structures called “bones”. Each “bone” is represented as *Transform*, which is applied to a group of vertices within a mesh. You can directly control a group of vertices from Godot. For that please reference *MeshDataTool* class, method *set_vertex_bones*. This class is very powerful.

The “bones” are organized in hierarchy, every bone, except for root bone(s) have parent. Every bone have associated name you can use to refer to it (e.g. “root” or “hand.L”, etc.). Also bones are all numbered, these numbers are bone IDs. Bone parents are referred by their numbered IDs.

For the rest of the article we consider the following scene:

```
main (Spatial) - script is always here
-- skel (Skeleton)
===== mesh (MeshInstance)
```

This scene is imported from Blender. It contains arm mesh with 2 bones - upperarm and lowerarm, with lowerarm parented to upperarm.

7.11.2 Skeleton class

You can view Godot internal help for descriptions of every function. Basically all operations on bones are done using their numeric ID. You can convert from name to numeric ID and vice versa.

To find number of bones in skeleton we use get_bone_count() function

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("upperarm")
    print("bone id:", id)
    var parent = skel.get_bone_parent(id)
    print("bone parent id:", id)
```

to find ID for the bone, use find_bone() function

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("upperarm")
    print("bone id:", id)
```

Now, we want to do something interesting with ID except for printing it. Also, we might need additional information - to find bone parents to complete chain, etc. This all is done with get/set_bone_* functions.

To find bone parent we use get_bone_parent(id) function

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("upperarm")
    print("bone id:", id)
    var parent = skel.get_bone_parent(id)
    print("bone parent id:", id)
```

Bone transforms is the thing why we're here at all. There are 3 kind of transforms - local, global, custom.

To find bone local Transform we use get_bone_pose(id) function

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("upperarm")
    print("bone id:", id)
    var parent = skel.get_bone_parent(id)
    print("bone parent id:", id)
```

```
var t = skel.get_bone_pose(id)
print("bone transform: ", t)
```

So we see 3x4 matrix there, with first column of 1s. What can we do about that? It is a Transform, so we can do everything we can do with Transform, basically translate, rotate and scale. Also we can multiply transforms to have complex transforms. Remember, “bones” in Godot are just Transforms over a group of vertices. Also we can copy Transforms of other objects there. So lets rotate our “upperarm” bone:

```
extends Spatial
var skel
var id

func _ready():
    skel = get_node("skel")
    id = skel.find_bone("upperarm")
    print("bone id:", id)
    var parent = skel.get_bone_parent(id)
    print("bone parent id:", id)
    var t = skel.get_bone_pose(id)
    print("bone transform: ", t)
    set_process(true)

func _process(dt):
    var t = skel.get_bone_pose(id)
    t = t.rotated(Vector3(0.0, 1.0, 0.0), 0.1 * dt)
    skel.set_bone_pose(id, t)
```

Now we can rotate individual bones. The same happens for scale and translate - try these on your own and see results.

What we used now was local pose. By default all bones are not modified. But this Transform tells us nothing about relationship between bones. This information is needed for quite a number of tasks. How can we get it? Here comes global transform:

To find bone global Transform we use `get_bone_global_pose(id)` function

We will find global Transform for lowerarm bone:

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("lowerarm")
    print("bone id:", id)
    var parent = skel.get_bone_parent(id)
    print("bone parent id:", id)
    var t = skel.get_bone_global_pose(id)
    print("bone transform: ", t)
```

As you see, this transform is not zeroed. While being called global, it is actually relative to Skeleton origin. For root bone, origin is always at 0 if not modified. Lets print origin for our lowerarm bone:

```
extends Spatial
var skel

func _ready():
    skel = get_node("skel")
    var id = skel.find_bone("lowerarm")
```

```

print("bone id:", id)
var parent = skel.get_bone_parent(id)
print("bone parent id:", id)
var t = skel.get_bone_global_pose(id)
print("bone origin: ", t.origin)

```

You will see a number. What does this number mean? It is a rotation point of Transform. So it is base part of the bone. In Blender you can go to Pose mode and try there to rotate bones - they will rotate around their origin. But what about tip? We can't know things like bone length, which we need for many things, without knowing tip location. For all bones in chain except for last one we can calculate tip location - it is simply a child bone origin. Yes, there are situations when this is not true, for non-connected bones. But that is OK for us for now, as it is not important regarding Transforms. But the leaf bone tip is nowhere to be found. Leaf bone is a bone without children. So you don't have any information about its tip. But this is not a showstopper. You can overcome this by either adding extra bone to the chain or just calculating leaf bone length in Blender and store the value in your script.

7.11.3 Using 3D “bones” for mesh control

Now as you know basics we can apply these to make full FK-control of our arm (FK is forward-kinematics)

To fully control our arm we need the following parameters:

- Upperarm angle x, y, z
- Lowerarm angle x, y, z

All of these parameters can be set, incremented and decremented.

Create the following node tree:

```

main (Spatial) <- script is here
+-arm (arm scene)
+ DirectionLight (DirectionLight)
+ Camera

```

Set up Camera so that arm is properly visible. Rotate DirectionLight so that arm is properly lit while in scene play mode.

Now we need to create new script under main:

First we setup parameters:

```

var lowerarm_angle = Vector3var upperarm_angle = Vector3()

```

Now we need to setup a way to change them. Let us use keys for that.

Please create 7 actions under project settings:

- **select_x** - bind to X key
- **select_y** - bind to Y key
- **select_z** - bind to Z key
- **select_upperarm** - bind to key 1
- **select_lowerarm** - bind to key 2
- **increment** - bind to key numpad +
- **decrement** - bind to key numpad -

So now we want to adjust the above parameters. Therefore we create code which does that:

```
func _ready():
    set_process(true)
var bone = "upperarm"
var coordinate = 0
func _process(dt):
    if Input.is_action_pressed("select_x"):
        coordinate = 0
    elif Input.is_action_pressed("select_y"):
        coordinate = 1
    elif Input.is_action_pressed("select_z"):
        coordinate = 2
    elif Input.is_action_pressed("select_upperarm"):
        bone = "upperarm"
    elif Input.is_action_pressed("select_lowerarm"):
        bone = "lowerarm"
    elif Input.is_action_pressed("increment"):
        if bone == "lowerarm":
            lowerarm_angle[coordinate] += 1
        elif bone == "upperarm":
            upperarm_angle[coordinate] += 1
```

The full code for arm control is this:

```
extends Spatial

# member variables here, example:
# var a=2
# var b="textvar"
var upperarm_angle = Vector3()
var lowerarm_angle = Vector3()
var skel

func _ready():
    skel = get_node("arm/Armature/Skeleton")
    set_process(true)
var bone = "upperarm"
var coordinate = 0
func set_bone_rot(bone, ang):
    var b = skel.find_bone(bone)
    var rest = skel.get_bone_rest(b)
    var newpose = rest.rotated(Vector3(1.0, 0.0, 0.0), ang.x)
    var newpose = newpose.rotated(Vector3(0.0, 1.0, 0.0), ang.y)
    var newpose = newpose.rotated(Vector3(0.0, 0.0, 1.0), ang.z)
    skel.set_bone_pose(b, newpose)

func _process(dt):
    if Input.is_action_pressed("select_x"):
        coordinate = 0
    elif Input.is_action_pressed("select_y"):
        coordinate = 1
    elif Input.is_action_pressed("select_z"):
        coordinate = 2
    elif Input.is_action_pressed("select_upperarm"):
        bone = "upperarm"
    elif Input.is_action_pressed("select_lowerarm"):
        bone = "lowerarm"
    elif Input.is_action_pressed("increment"):
```

```

if bone == "lowerarm":
    lowerarm_angle[coordinate] += 1
elif bone == "upperarm":
    upperarm_angle[coordinate] += 1
elif Input.is_action_pressed("decrement"):
    if bone == "lowerarm":
        lowerarm_angle[coordinate] -= 1
    elif bone == "upperarm":
        upperarm_angle[coordinate] -= 1
set_bone_rot("lowerarm", lowerarm_angle)
set_bone_rot("upperarm", upperarm_angle)

```

Pressing keys 1/2 select upperarm/lowerarm, select axis by pressing x, y, z, rotate using numpad “+”/-“

This way you fully control your arm in FK mode using 2 bones. You can add additional bones and/or improve “feel” of the interface by using coefficients for the change. I recommend you play with this example a lot before going to next part.

You can clone the demo code for this chapter using

```
git clone git@github.com:slapin/godot-skel3d.git
cd demo1
```

Or you can browse it using web-interface:

<https://github.com/slavin/godot-skel3d>

7.11.4 Using 3D “bones” to implement Inverse Kinematics

See *Inverse kinematics*.

7.11.5 Using 3D “bones” to implement ragdoll-like physics

TODO.

7.12 Inverse kinematics

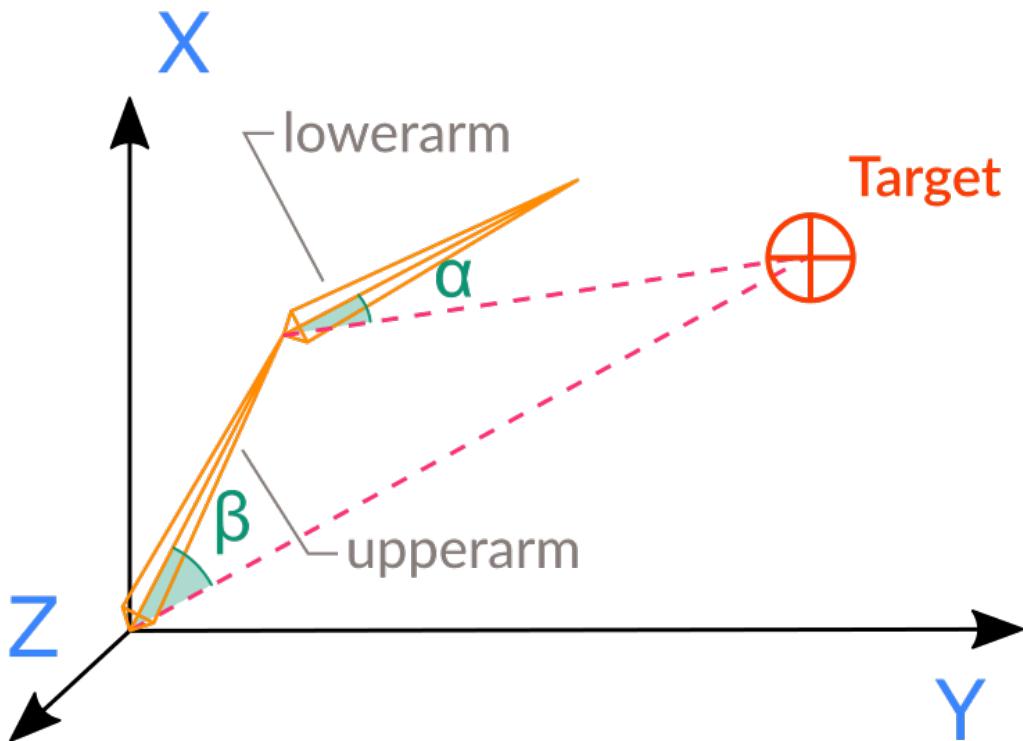
This tutorial is a follow-up of *Working with 3D skeletons*.

Before continuing on, I’d recommend reading some theory, the simplest article I find is this:

http://freespace.virgin.net/hugo.elias/models/m_ik2.htm

7.12.1 Initial problem

Talking in Godot terminology, the task we want to solve here is position our 2 angles we talked about above so, that the tip of lowerarm bone is as close to target point, which is set by target Vector3() as possible using only rotations. This task is very calculation-intensive and never resolved by analytical equation solve. So, it is an underconstrained problem, which means there is unlimited number of solutions to the equation.



For easy calculation, for this chapter we consider target is also child of Skeleton. If it is not the case for your setup you can always reparent it in your script, as you will save on calculations if you do.

In the picture you see angles alpha and beta. In this case we don't use poles and constraints, so we need to add our own. On the picture the angles are 2D angles living in plane which is defined by bone base, bone tip and target.

The rotation axis is easily calculated using cross-product of bone vector and target vector. The rotation in this case will be always in positive direction. If t is the Transform which we get from `get_bone_global_pose()` function, the bone vector is

```
t.basis[2]
```

So we have all information here to execute our algorithm.

In game dev it is common to resolve this problem by iteratively closing to the desired location, adding/subtracting small numbers to the angles until the distance change achieved is less than some small error value. Sounds easy enough, but there are Godot problems we need to resolve there to achieve our goal.

- **How to find coordinates of the tip of the bone?**
- **How to find vector from bone base to target?**

For our goal (tip of the bone moved within area of target), we need to know where the tip of our IK bone is. As we don't use a leaf bone as IK bone, we know the coordinate of the bone base is the tip of parent bone. All these calculations are quite dependent on the skeleton's structure. You can use pre-calculated constants as well. You can add an extra bone for the tip of IK and calculate using that.

7.12.2 Implementation

We will just use exported variable for bone length to be easy.

```
export var IK_bone="lowerarm"
export var IK_bone_length=1.0
export var IK_error = 0.1
```

Now, we need to apply our transformations from IK bone to the base of chain. So we apply rotation to IK bone then move from our IK bone up to its parent, then apply rotation again, then move to the parent of current bone again, etc. So we need to limit our chain somewhat.

```
export var IK_limit = 2
```

For `_ready()` function:

```
var skel
func _ready():
    skel = get_node("arm/Armature/Skeleton")
    set_process(true)
```

Now we can write our chain-passing function:

```
func pass_chain():
    var b = skel.find_bone(IK_bone)
    var l = IK_limit
    while b >= 0 and l > 0:
        print( "name:", skel.get_bone_name(b) )
        print( "local transform:", skel.get_bone_pose(b) )
        print( "global transform:", skel.get_bone_global_pose(b) )
        b = skel.get_bone_parent(b)
        l = l - 1
```

And for the `_process()` function:

```
func _process(dt):
    pass_chain(dt)
```

Executing this script will just pass through bone chain printing bone transforms.

```
extends Spatial

export var IK_bone="lowerarm"
export var IK_bone_length=1.0
export var IK_error = 0.1
export var IK_limit = 2
var skel
func _ready():
    skel = get_node("arm/Armature/Skeleton")
    set_process(true)
func pass_chain(dt):
    var b = skel.find_bone(IK_bone)
    var l = IK_limit
    while b >= 0 and l > 0:
        print("name: ", skel.get_bone_name(b) )
        print("local transform: ", skel.get_bone_pose(b) )
        print( "global transform:", skel.get_bone_global_pose(b) )
        b = skel.get_bone_parent(b)
```

```
    l = l - 1
func _process(dt):
    pass_chain(dt)
```

Now we need to actually work with target. The target should be placed somewhere accessible. Since “arm” is imported scene, we better place target node within our top level scene. But for us to work with target easily its Transform should be on the same level as Skeleton.

To cope with this problem we create “target” node under our scene root node and at script run we will reparent it copying global transform, which will achieve wanted effect.

Create new Spatial node under root node and rename it to “target”. Then modify `_ready()` function to look like this:

```
var skel
var target
func _ready():
    skel = get_node("arm/Armature/Skeleton")
    target = get_node("target")
    var ttrans = target.get_global_transform()
    remove_child(target)
    skel.add_child(target)
    target.set_global_transform(ttrans)
    set_process(true)
```

7.13 Mesh generation with heightmap and shaders

7.13.1 Introduction

This tutorial will help you to use Godot shaders to deform a plane mesh so it appears like a basic terrain. Remember that this solution has pros and cons.

Pros:

- Pretty easy to do.
- This approach allows computation of LOD terrains.
- The heightmap can be used in Godot to create a normal map.

Cons:

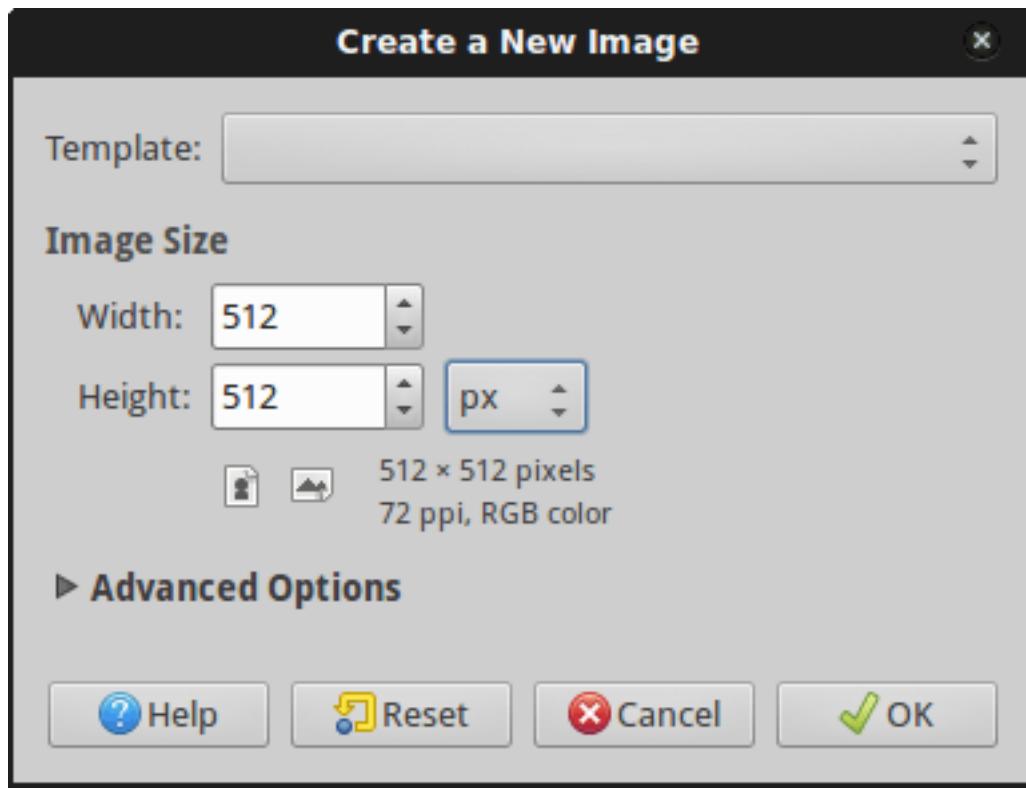
- The Vertex Shader can't re-compute normals of the faces. Thus, if your mesh is not static, this method will **not** work with shaded materials.
- This tutorial uses a plane mesh imported from Blender to Godot Engine. Godot is able to create meshes as well.

See this tutorial as an introduction, not a method that you should employ in your games, except if you intend to do LOD. Otherwise, this is probably not the best way.

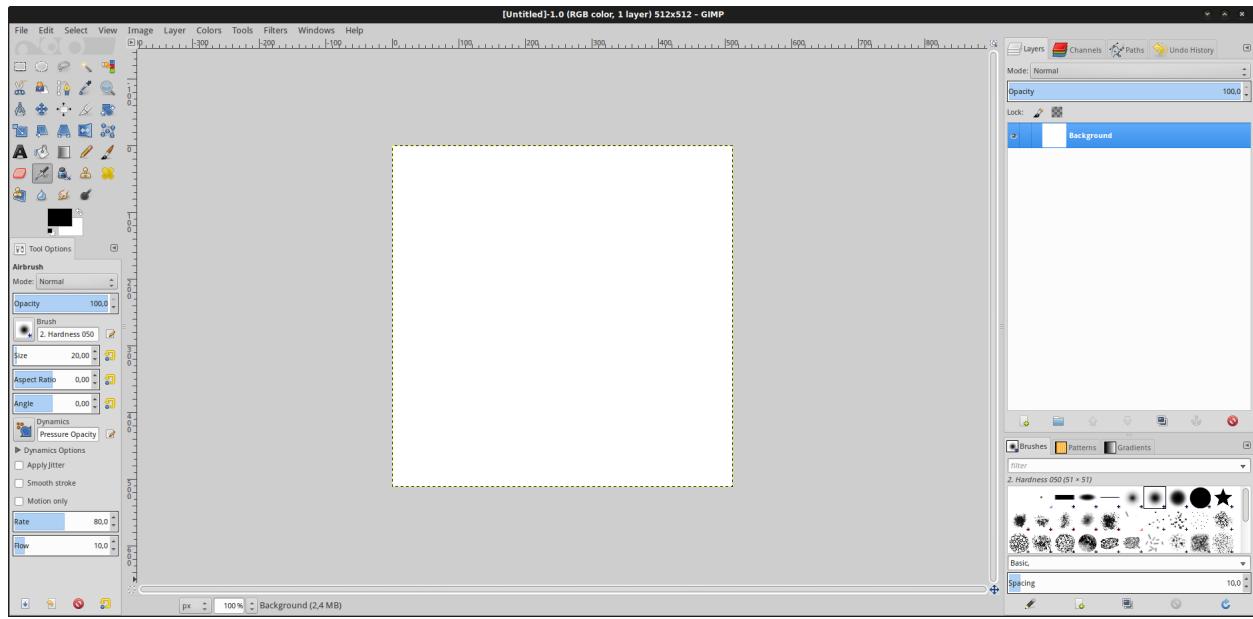
However, let's first create a heightmap, or a 2D representation of the terrain. To do this, I'll use GIMP, but you can use any image editor you like.

7.13.2 The heightmap

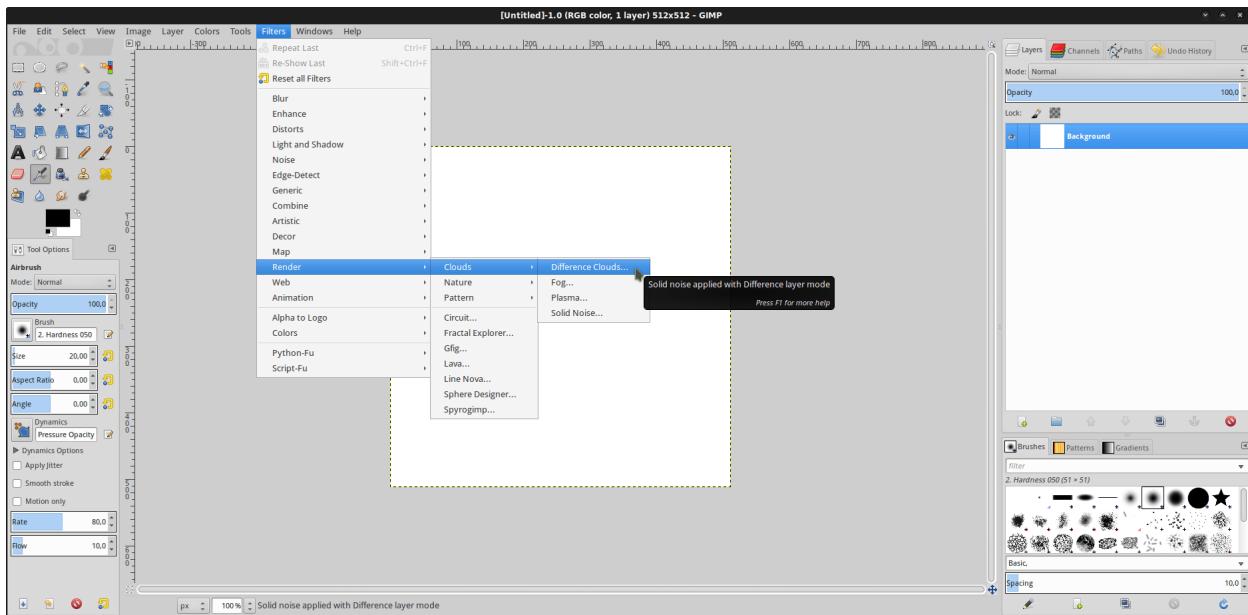
We will use a few functions of GIMP image editor to produce a simple heightmap. Start GIMP and create a square image of 512x512 pixels.



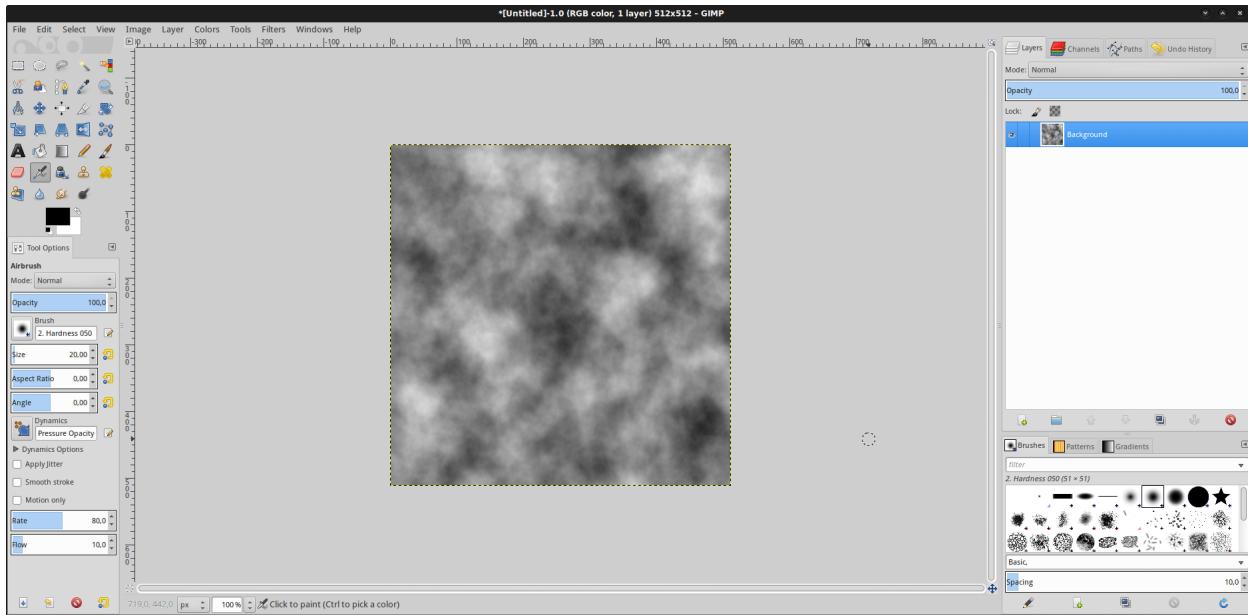
You are now in front of a new, blank, square image.



Then, use a filter to render some clouds on this new image.



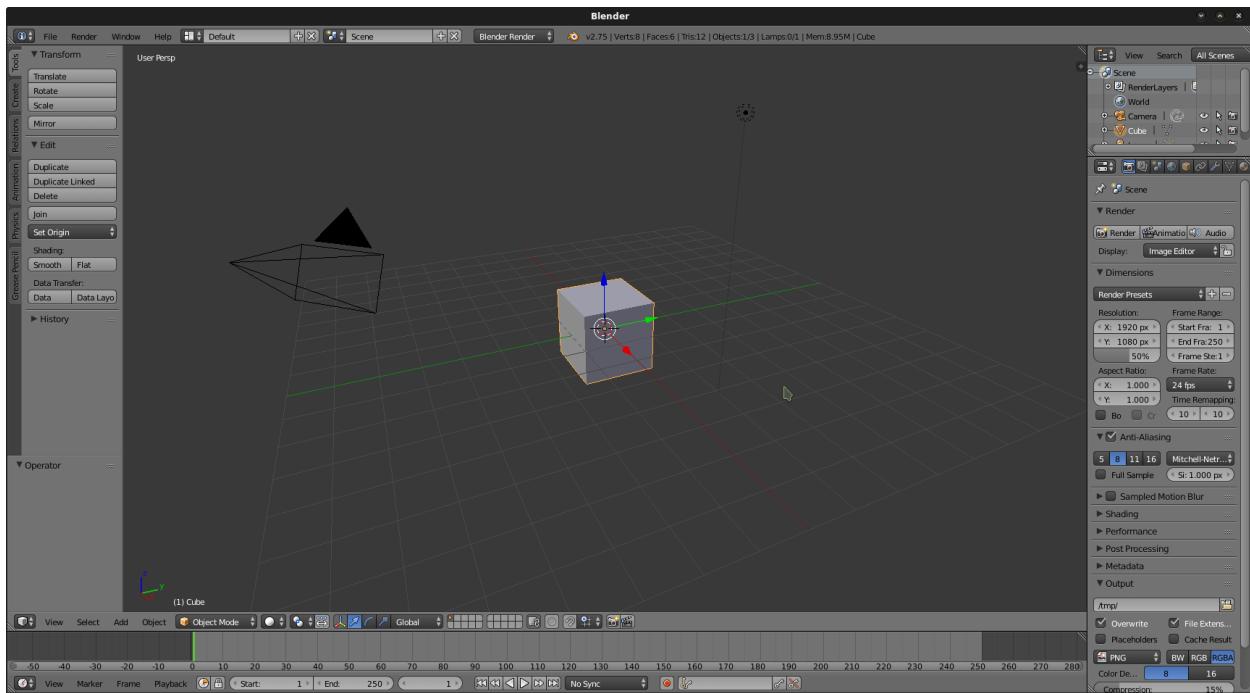
Parameter this filter to whatever you want. A white pixel corresponds to the highest point of the heightmap, a black pixel corresponds to the lowest one. So, darker regions are valleys and brighter are mountains. If you want, you can check “tileable” to render a heightmap that can be cloned and tiled close together with another one. X and Y size don’t matter a lot as long as they are big enough to provide a decent ground. A value of 4.0 or 5.0 for both is nice. Click on the “New Seed” button to roll a dice and GIMP will create a new random heightmap. Once you are happy with the result, click “OK”.



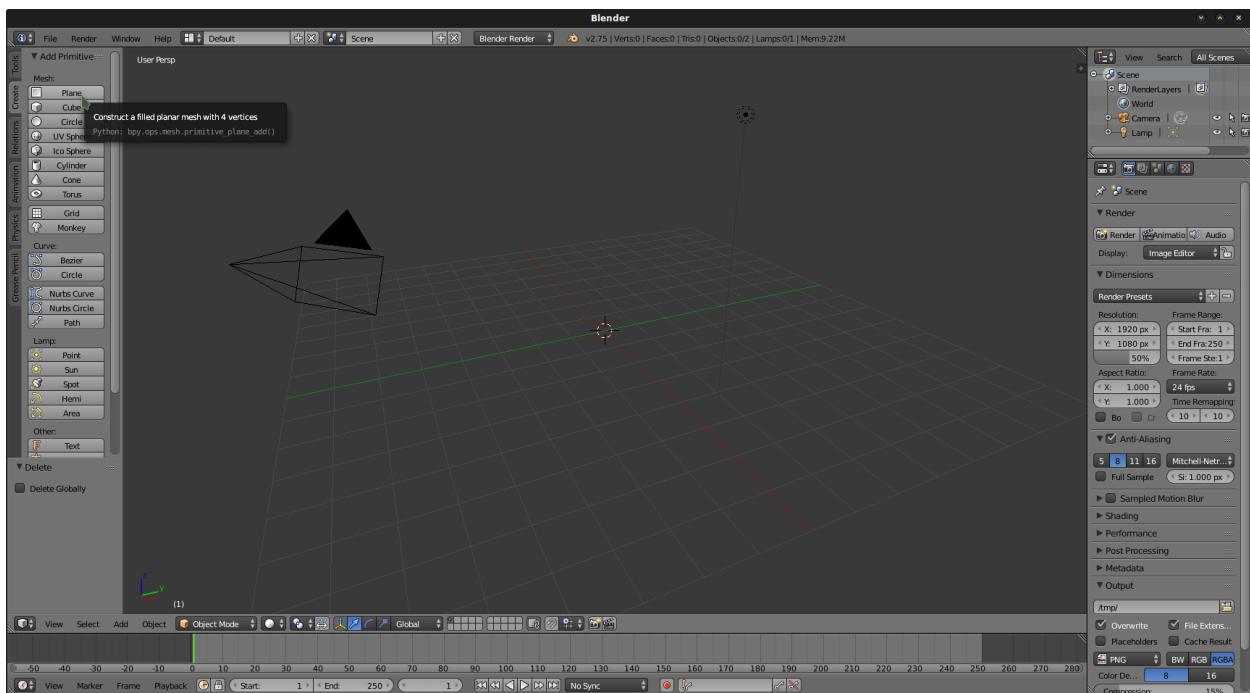
You can continue to edit your image if you wish. For our example, let’s keep the heightmap as is, and let’s export it to a PNG file, say “heightmap.png”. Save it in your Godot project folder.

7.13.3 The plane mesh

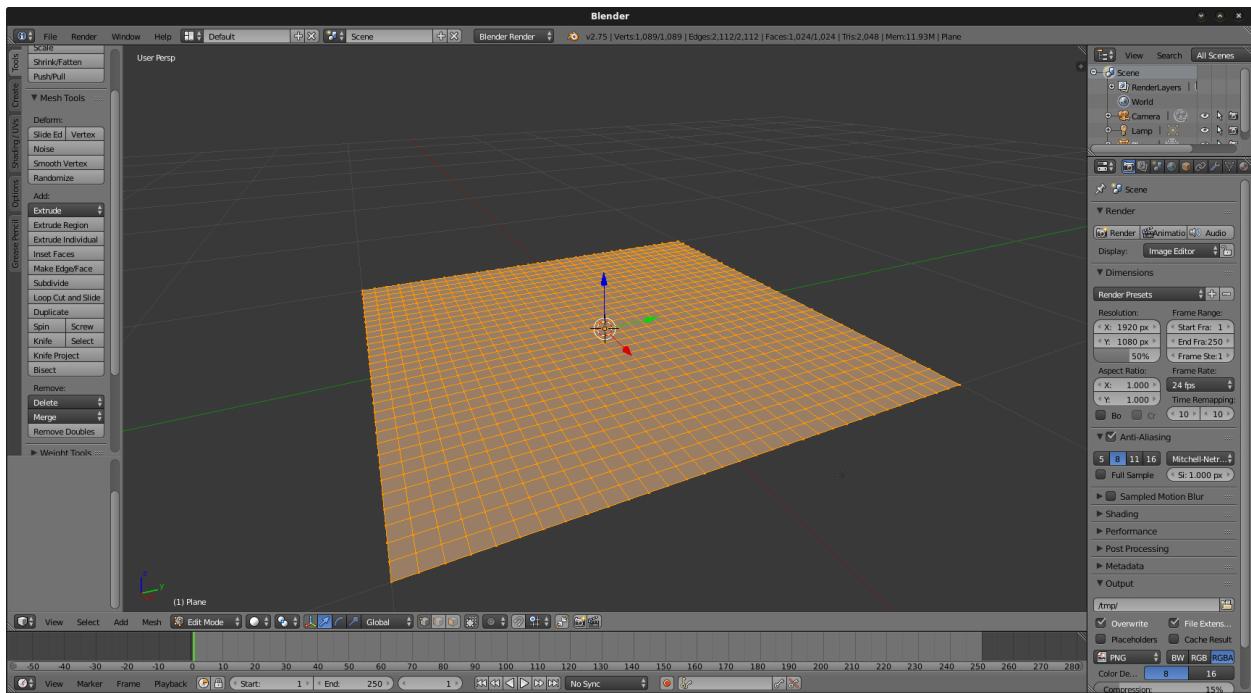
Now, we will need a plane mesh to import in Godot. Let’s run Blender.



Remove the start cube mesh, then add a new plane to the scene.

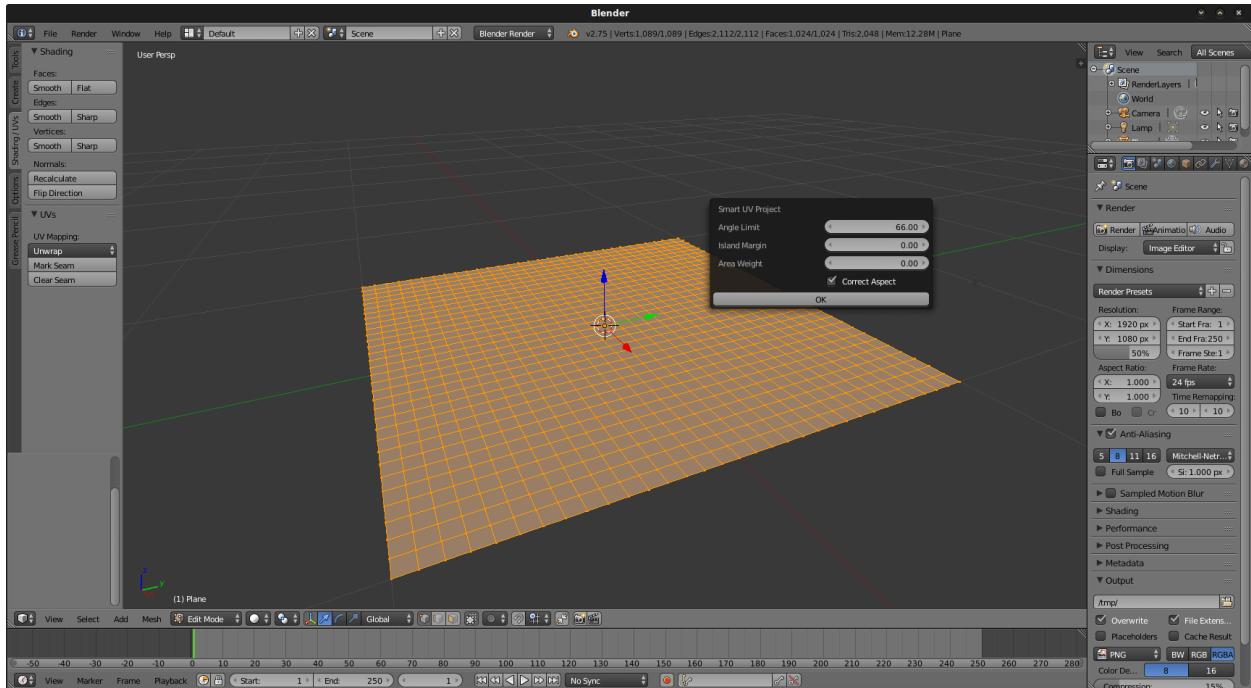


Zoom a bit, then switch to Edit mode (Tab key) and in the Tools button group at the left, hit “Subdivide” 5 or 6 times.

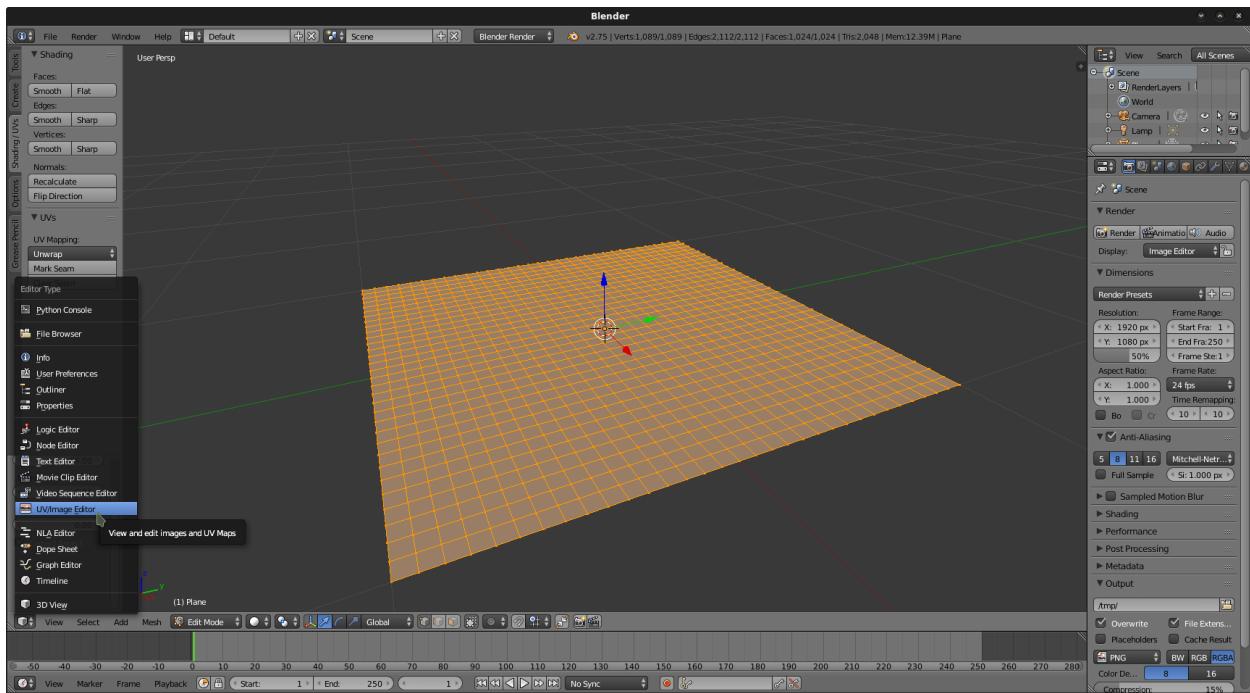


Your mesh is now subdivided, which means we added vertices to the plane mesh that we will later be able to move. Job's not finished yet: in order to texture this mesh a proper UV map is necessary. Currently, the default UV map contains only the 4 corner vertices we had at the beginning. However, we now have more, and we want to be able to texture over the whole mesh correctly.

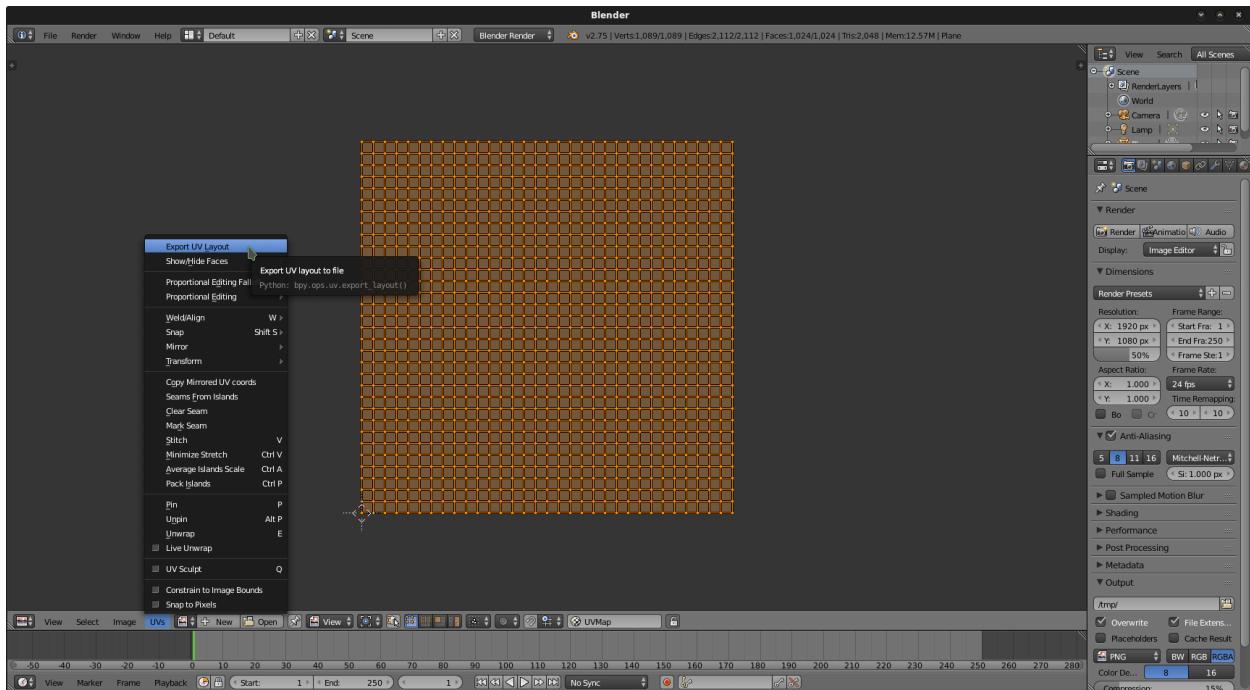
If all the vertices of your mesh are not selected, select them all (hit "A"). They must appear orange, not black. Then, in the Shading/UVs button group to the left, click the "Unwrap" button (or simply hit "U") and select "Smart UV Project". Keep the default options and hit "Ok".



Now, we need to switch our view to “UV/Image editor”.



Select all the vertices again (“A”) then in the UV menu, select “Export UV Layout”.

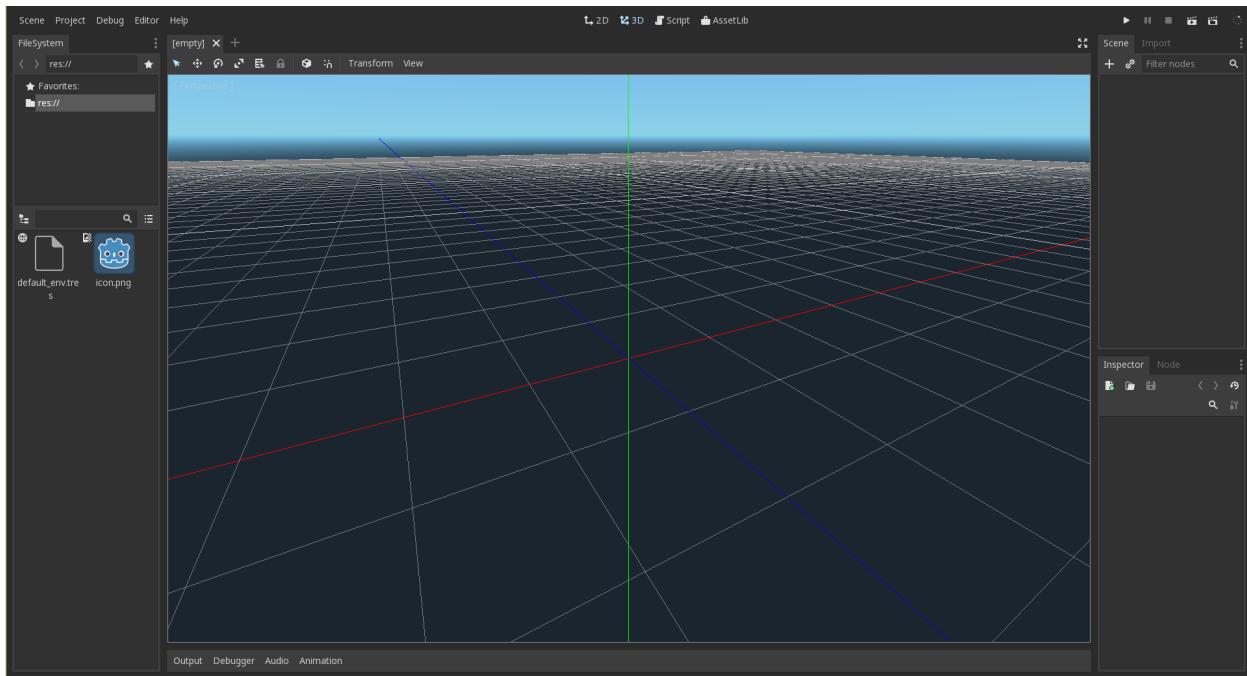


Export the layout as a PNG file. Name it “plane.png” and save it in your Godot project folder. Now, let’s export our mesh as an OBJ file. Top of the screen, click “File/Export/Wavefront (obj)”. Save your object as “plane.obj” in your Godot project folder.

7.13.4 Shader magic

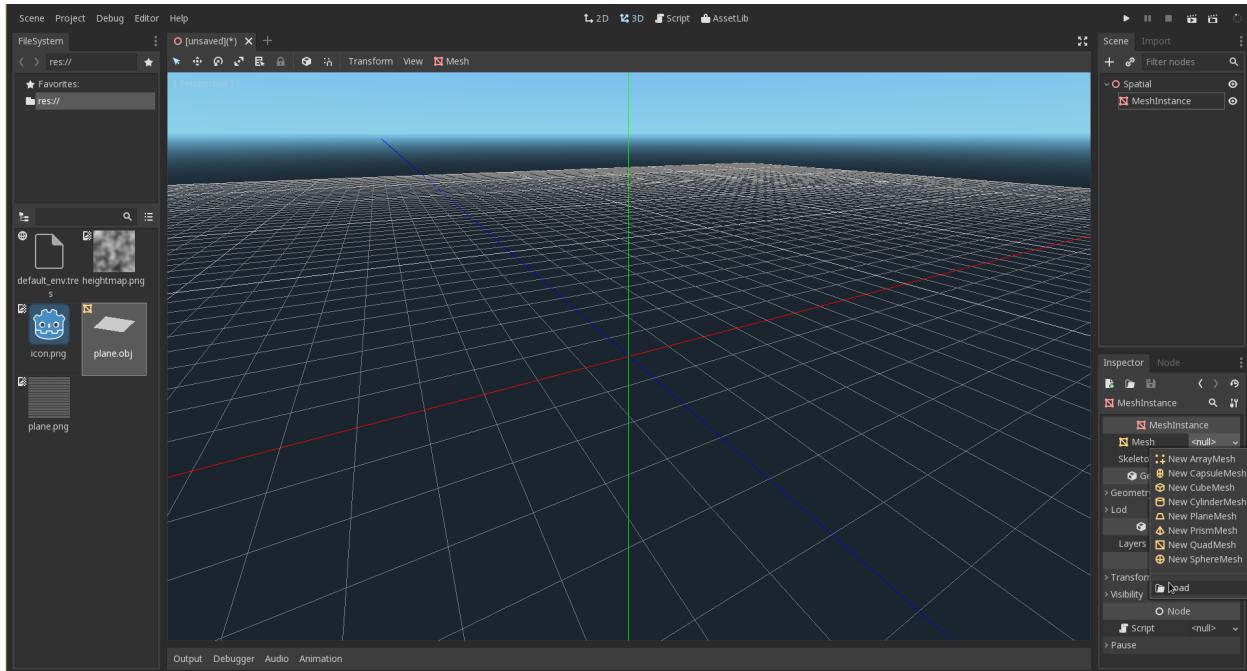
Let’s now open Godot Editor.

Create a new project in the folder you previously created and name it what you want.

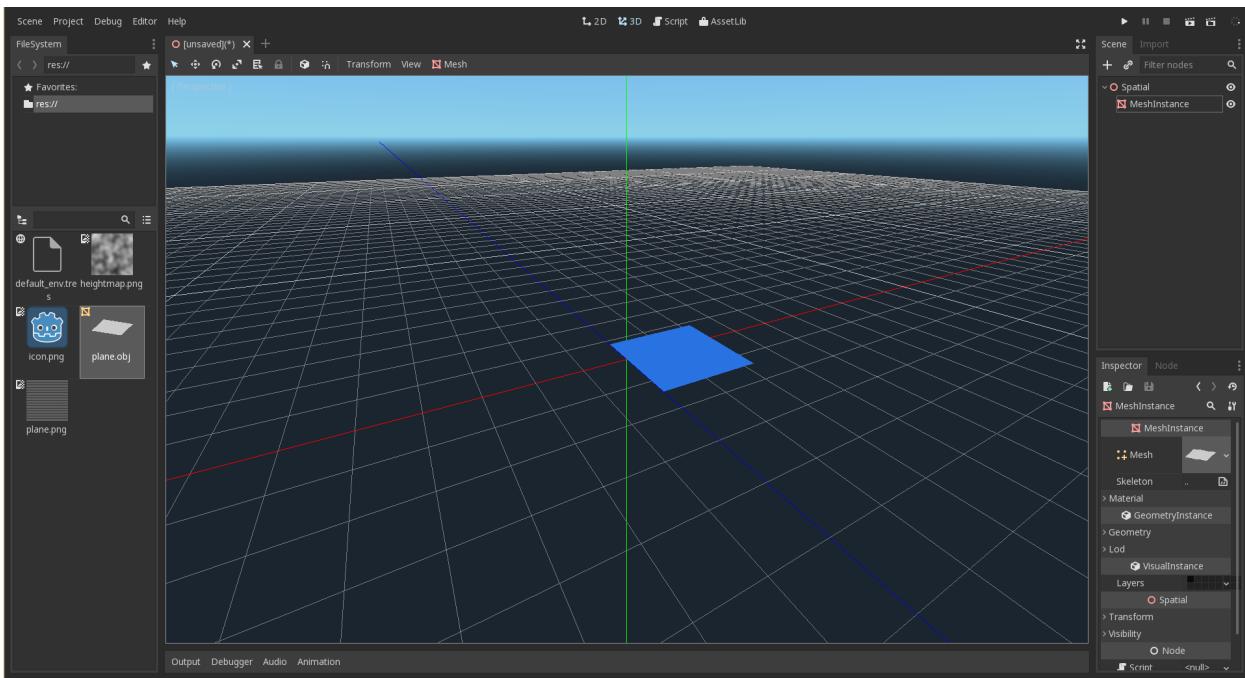


In our default scene (3D), create a root node “Spatial”.

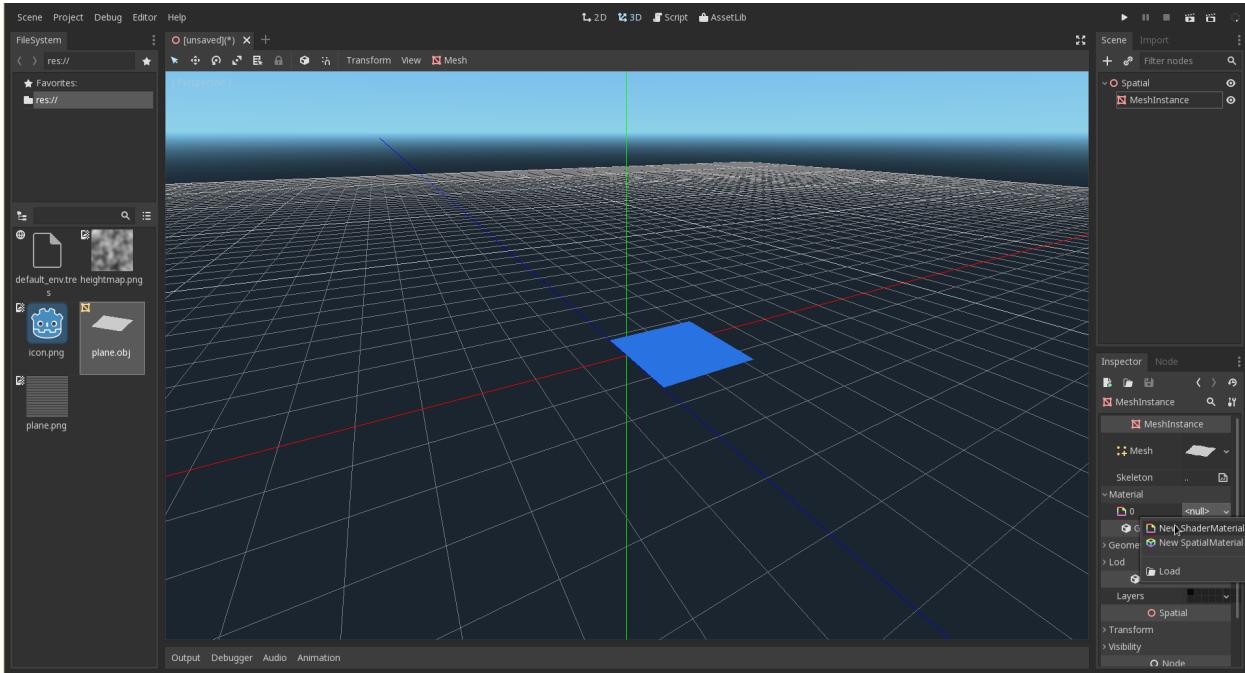
Create a MeshInstance node as a child of the node we just created. Then, load the Mesh selecting “Load” and then our “plane.obj” file.



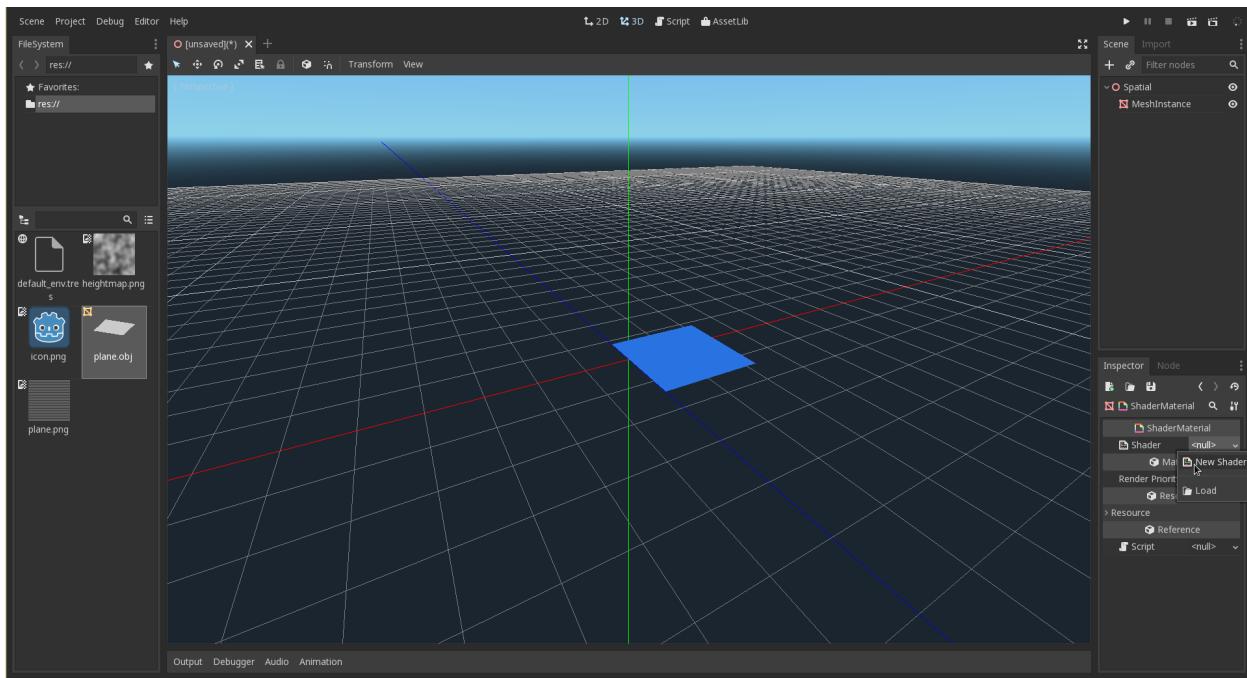
Great! Our plane is now rendered in the 3D view.



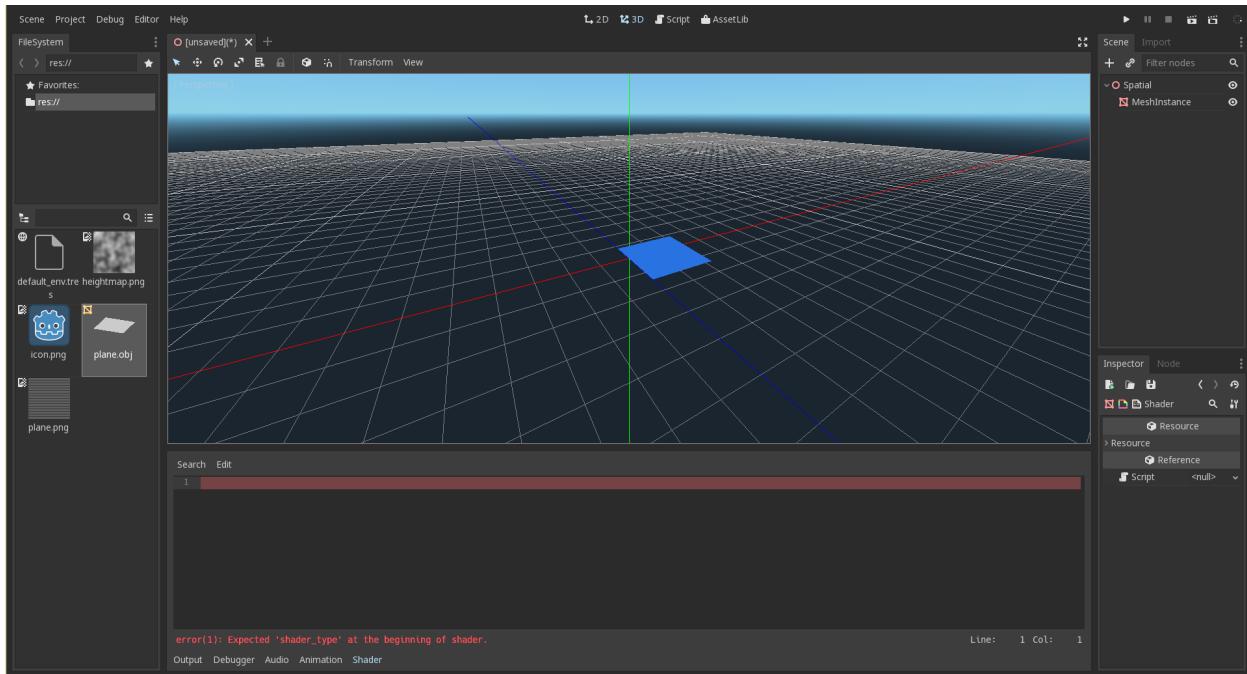
It is time to add some shader stuff. In the Inspector, in the “Material” line, add a “New ShaderMaterial”. Edit it by clicking it and then selecting the “Edit” option.



We need now to create the actual shader. On the Inspector, select the “Shader” line and click on “New Shader”.



Edit it by clicking the “Edit” option just like we did before. The Shader editor opens.



Let's start writing our shader. If you don't know how to use shaders in Godot you can check the *Shading language* page.

Let's start with the Fragment part. This one is used to texture the plane using an image. For this example, we will texture it with the heightmap image itself, so we'll actually see mountains as brighter regions and canyons as darker regions. Use this code:

```
shader_type spatial;
render_mode unshaded;
```

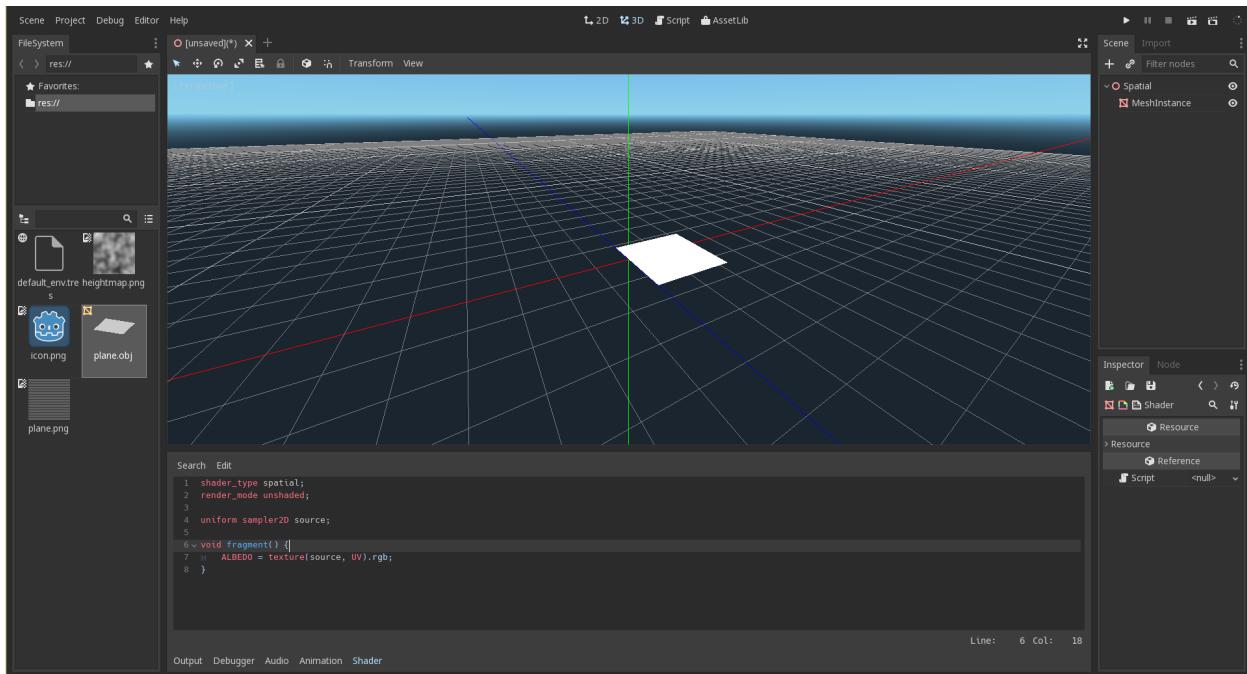
```
uniform sampler2D source;

void fragment() {
    ALBEDO = texture(source, UV).rgb;
}
```

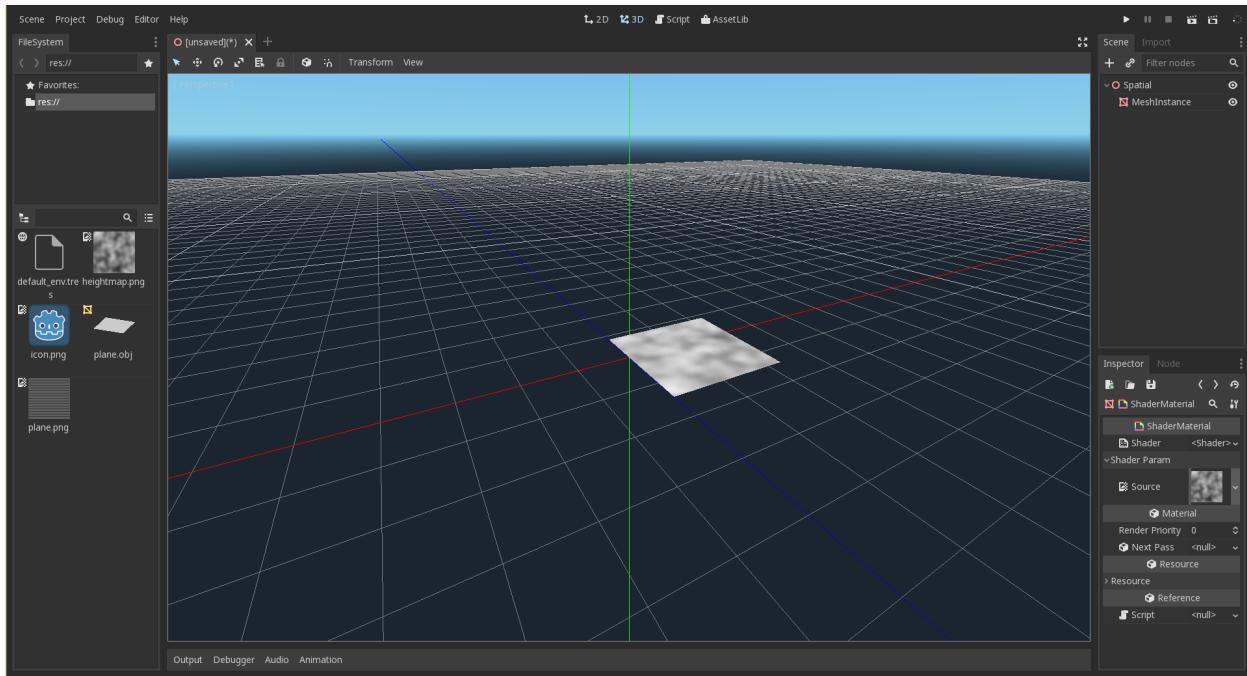
First, we set the shader type as `spatial` (for 3D). The `render_mode unshaded` line makes our `MeshInstance` be unaffected by the lighting in our world. It doesn't matter since it is a greyscale image. We take a parameter (`uniform`) as a `sampler2D`, which will be the texture of our heightmap.

Then, we set the color of every pixel of the image given by `texture(source, UV).rgb` setting it to the `ALBEDO` variable. Remember that the `UV` variable is a shader variable that returns the 2D position of the pixel in the texture image, according to the vertex we are currently dealing with. That is the use of the `UV Layout` we made before.

However, the plane is displayed white! This is because we didn't set the texture file and the color to use.



In the Inspector, click the back arrow to get back to the `ShaderMaterial`. This is where you want to set the texture and the color. Click the “Shader Param” line, in “Source”, click “Load” and select the texture file “heightmap.png”. Now you will see our heightmap.



Good. Now, the Vertex part.

The Vertex Shader is the first shader to be executed by the pipeline. It deals with vertices.

Insert a new “uniform” variable right after the one that we introduced before:

```
uniform float height_range;
```

The `height_range` parameter is a parameter that we will use to increase the height effect.

Then, insert the following function before the fragment function we wrote before.

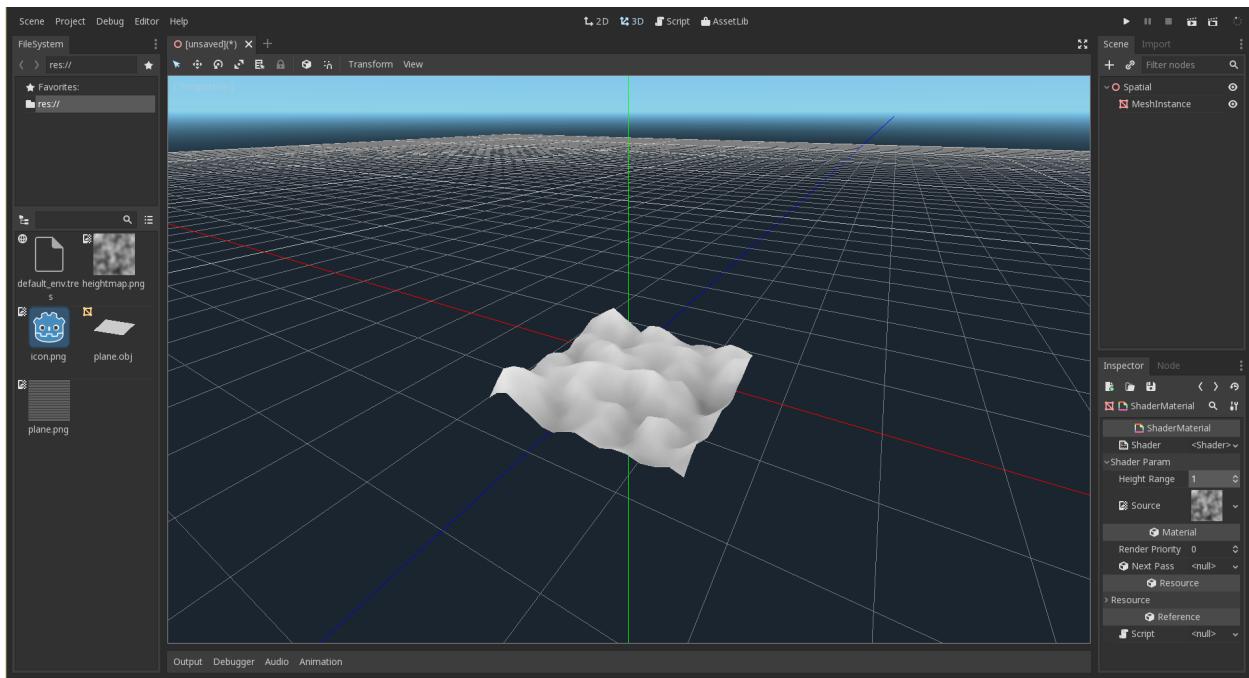
```
void vertex() {
    vec2 xz = VERTEX.xz;
    float h = texture(source, UV).g * height_range;
    VERTEX = vec3(xz.x, h, xz.y);
}
```

First, we save the x and z position of the `VERTEX`, because we do not want them to change: the plane must remain square. Remember that Y axis corresponds to the “altitude”, which is the only one we want to change with the heightmap.

Then, we compute an `h` variable by multiplying the pixel value at the UV position and the `height_range`. As the heightmap is a greyscale image, all r, g and b channels contain the same value. We use `g`, but any of r, g and b have the same effect.

After that, we set the current vertex’ position at (`xz.x`, `h`, `xz.y`) position. Concerning `xz.y` remember that its type is “`vec2`”. Thus, its components are `x` and `y`.

That’s all good, but our plane remains flat. This is because the `height_range` value is 0. Increase this value to observe the mesh distort and take to form of the terrain we set before:



CHAPTER 8

Audio

8.1 Audio Buses

8.1.1 Introduction

Beginning Godot 3.0, the audio engine has been rewritten from scratch. The aim now is to present an interface much friendlier to sound design professionals. To achieve this, the audio engine contains a virtual rack where unlimited audio buses can be created and, on each of it, unlimited amount of effect processors can be added (or more like, as long as your CPU holds up!)

The implementation in Godot is pretty efficient and has been written entirely from the ground up, without relying on any existing audio libraries.

Even the effect processors were written exclusively for Godot (save for the pitch shifting library), with games in mind. This allows a very efficient tradeoff between performance and sound quality.

8.1.2 Decibel Scale

The new audio engine works primarily using the decibel scale. We have chosen this over linear representation of amplitude because it's more intuitive for audio professionals.

For those unfamiliar with it, it can be explained with a few facts:

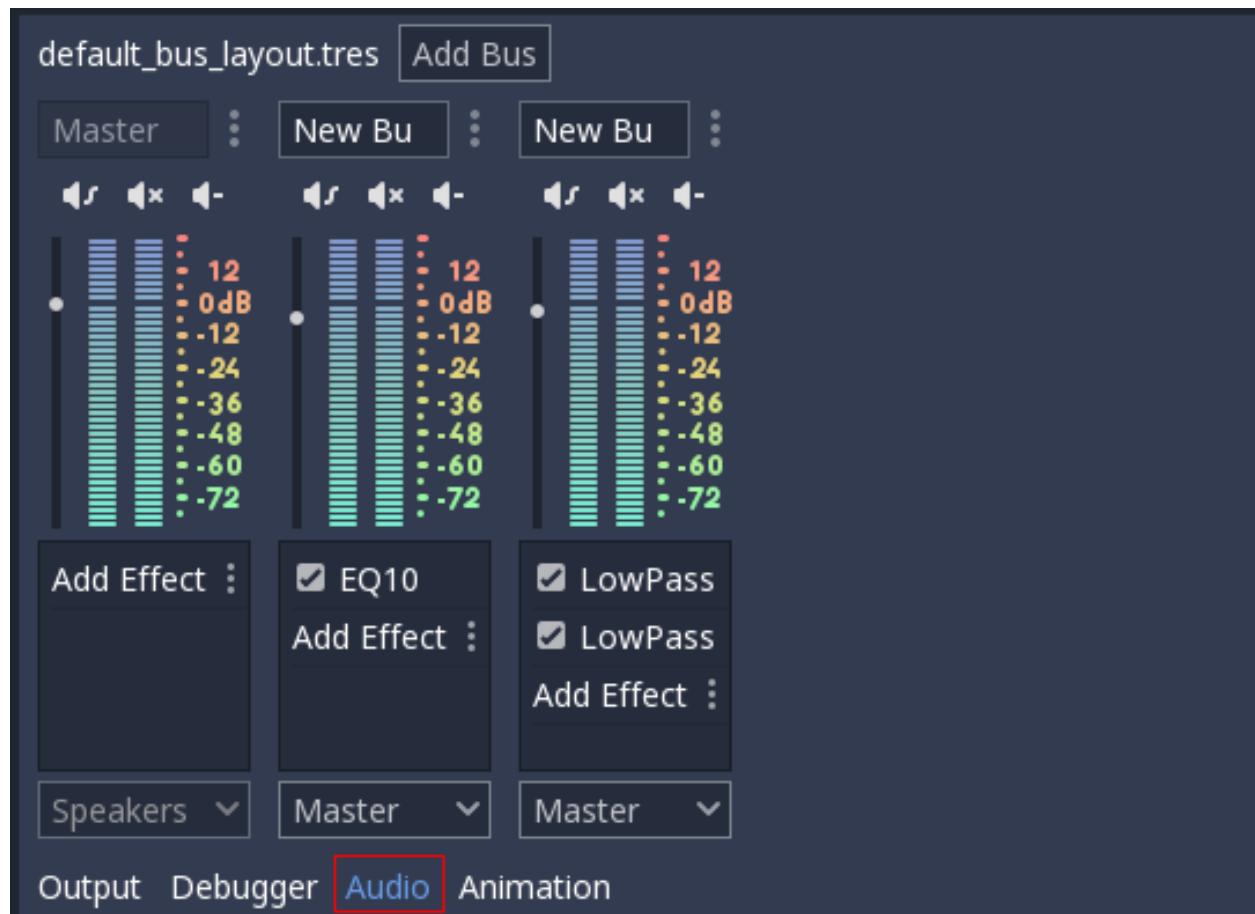
- The decibel (dB) scale is a relative scale. It represents the ratio of sound power by using 10 times the base 10 logarithm of the ratio ($10 \times \log_{10}(P/P_0)$).
- For every 3dB, sound doubles or halves. 6dB represents a factor 4, 9dB a factor 8, 10dB a factor 10, 20dB a factor 100, etc.
- Since the scale is logarithmic, true zero (no audio) can't be represented.
- 0dB is considered the maximum audible volume without *clipping*. This limit is not the human limit but a limit from the sound hardware. Your sound output simply can't output any sound louder than 0dB without distorting it (clipping it).

- Because of the above, your sound mix should work in a way where the sound output of the *Master Bus* (more on that later), should never be above 0dB.
- Every 3dB below the 0dB limit, sound energy is *halved*. It means the sound volume at -3dB is half as loud as 0dB. -6dB is half as loud as -3dB and so on.
- When working with decibels, sound is considered no longer audible between -60dB and -80dB. This makes your working range generally between -60dB and 0dB.

This can take a bit getting used to, but it's friendlier in the end and will allow you to communicate better with audio professionals.

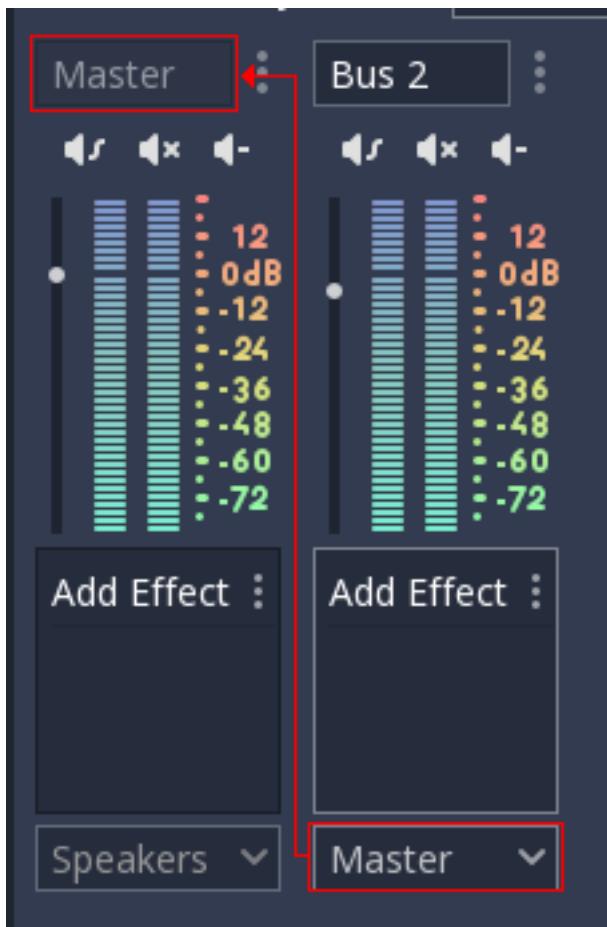
8.1.3 Audio Buses

Audio buses can be found in the bottom panel of Godot Editor:



An *Audio Bus* bus (often called “Audio Channels” too) is a device where audio is channeled. Audio data passes through it and can be *modified* and *re-routed*. A VU-Meter (the bars that go up and down when sound is played) can measure the loudness of the sound in Decibel scale.

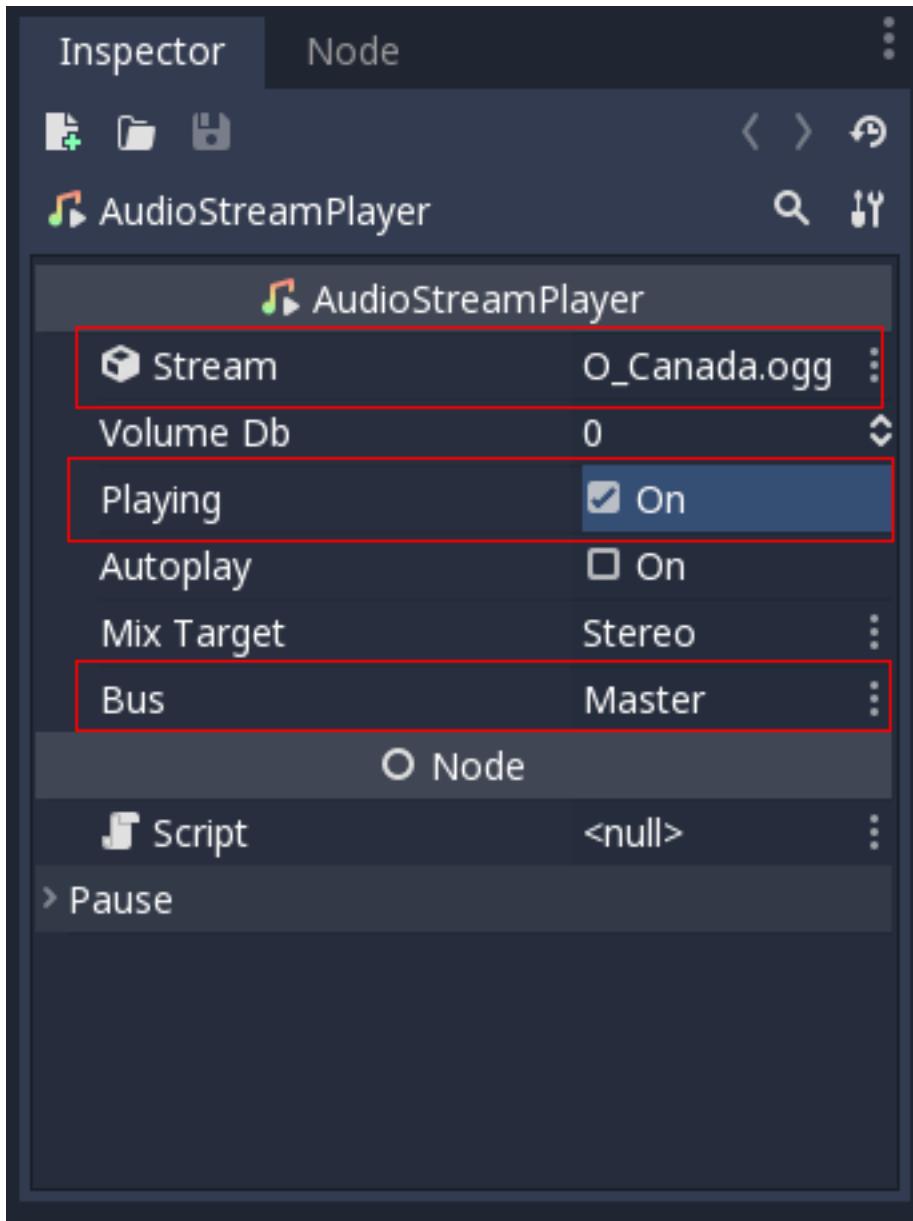
The leftmost bus is the *Master Bus*. This bus outputs the mix to your speakers so, as mentioned in the item above (Decibel Scale), make sure that your mix rarely or never goes above 0dB in this bus. The rest of the audio buses are used for *routing*. This means that, after modifying the sound, they must send it to another bus to the left. Routing is always from right to left without exception as this avoids creating infinite routing loops!



In the above image, *Bus 2* is routing its output to *Master* bus.

8.1.4 Playback of Audio to a Bus

To test playback to a bus, create an `AudioStreamPlayer` node, load an `AudioStream` and select a target bus for playback:



Finally toggle the “playing” property to on and sound will flow.

To learn more about *Audio Streams*, please read the related tutorial later! (@TODO link to audio streams tute)

8.1.5 Adding Effects

Audio buses can contain all sorts of effects. These effects modify the sound in one way or another and are applied in order.

```
Amplify
BandLimitFilter
BandPassFilter
Chorus
Compressor
Delay
Distortion
EQ10
EQ21
EQ6
HighPassFilter
HighShelfFilter
Limiter
LowPassFilter
LowShelfFilter
NotchFilter
Panner
Phaser
PitchShift
Reverb
StereoEnhance
```

Following is a short description of available effects:

Amplify

It's the most basic effect. It changes the sound volume. Amplifying too much can make the sound clip, so be wary of that.

BandLimit and BandPass

These are resonant filters which block frequencies around the *Cutoff* point. BandPass is resonant, while BandLimit stretches to the sides.

Chorus

This effect adds extra voices, detuned by LFO and with a small delay, to add more richness to the sound harmonics and stereo width.

Compressor

The aim of a dynamic range compressor is to reduce the level of the sound when the amplitude goes over a certain threshold in Decibels. One of the main uses of a compressor is to increase the dynamic range while clipping the least possible (when sound goes over 0dB).

Compressor has many uses in the mix, for example:

- * It can be used in the Master bus to compress the whole output (Although a Limiter is probably better)
- * It can be used in voice channels to ensure they sound as even as possible.
- * It can be *Sidechained*. This means, it can reduce the sound level using another audio bus for threshold detection. This technique is very common in video game mixing to download the level of Music/SFX while voices are being heard.
- * It can accentuate transients by using a bit wider attack, meaning it can make sound effects sound more punchy.

There is a lot of bibliography written about compressors, and Godot implementation is rather standard.

Delay

Adds an “Echo” effect with a feedback loop. It can be used, together with Reverb, to simulate very wide rooms, canyons, etc. where sound bounces are far apart.

Distortion

Adds classical effects to modify the sound and make it dirty. Godot supports effects like overdrive, tan, or bit crushing. For games, it can simulate sound coming from some saturated device or speaker very efficiently.

EQ6, EQ10, EQ21

Godot provides three models of equalizers with different band counts. Equalizers are very useful on the Master Bus to completely master a mix and give it character. They are also very useful when a game is run on a mobile device, to adjust the mix to that kind of speakers (it can be added but disabled when headphones are plugged).

HighPassFilter, HighShelfFilter

These are filters that cut frequencies below a specific *Cutoff*. A common use of high pass filters is to add it to effects (or voice) that were recorded too close to a mic and need to sound more realistic. It is commonly used for some types of environment like space.

Limiter

A limiter is similar to a compressor, but it's less flexible and designed to disallow sound going over a given dB threshold. Adding one in the *Master Bus* is always recommended to reduce the effects of clipping.

LowPassFilter, LowShelfFilter

These are the most common filters, they cut frequencies above a specific *Cutoff* and can also resonate. They can be used for a wide amount of effects, from underwater sound to simulating a sound coming from far away.

NotchFilter

The opposite to the BandPassFilter, it removes a band of sound from the frequency spectrum at a given *Cutoff*.

Panner

This is a simple helper to pan sound left or right.

Phaser

It probably does not make much sense to explain that this effect is formed by two signals being dephased and cancelling each other out. It will be sufficient to note that you can make a Darth Vader voice with it, or jet-like sounds.

PitchShift

This effect allows for modulating pitch independently of tempo. All frequencies can be increased/decreased with minimal effect on transients. Can be used for effects such as voice modulation.

Reverb

Reverb simulates rooms of different sizes. It has adjustable parameters that can be tweaked to obtain the sound of a specific room. Reverb is commonly outputted from Areas (@TODO LINK TO TUTORIAL WHEN DONE), or to apply chamber feel to all sounds.

StereoEnhance

This effect has a few algorithms available to enhance the stereo spectrum, in case this is needed.

8.1.6 Automatic Bus Disabling

There is no need to disable buses manually when not in use, Godot detects that the bus has been silent for a few seconds and disable it (including all effects).



8.1.7 Bus Rearrangement

Stream Players use bus names to identify a bus, which allows adding, removing and moving buses around while the reference to them is kept. If a bus is renamed, however, the reference will be lost and the Stream Player will output to Master. This system was chosen because rearranging buses is a more common process than renaming them.

8.1.8 Default Bus Layout

The default bus layout is automatically saved to the “`res://default_bus_layout.res`” file. Other bus layouts can be saved/retrieved from files in case of having to change snapshots, but in most cases this is not necessary.

8.2 Audio Streams

8.2.1 Introduction

As you might have read already in the *Audio Buses Tutorial*, sound is sent to each bus via an `AudioStreamPlayer`.

There are many types of `AudioStreamPlayers` which will be explained in detail. Each of it loads an `AudioStream` and plays it back.

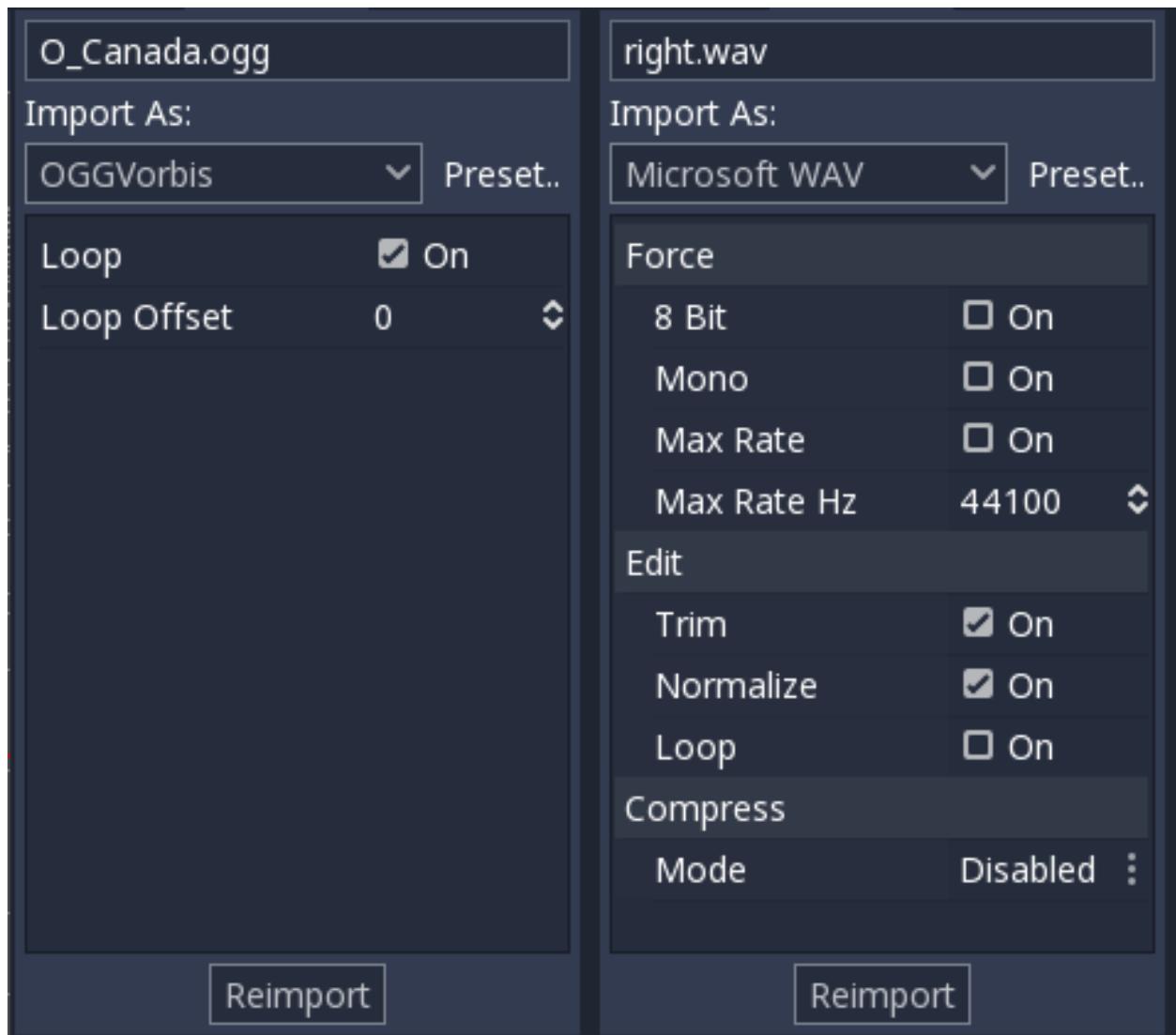
8.2.2 AudioStream

An audio stream is an abstract object that emits sound. It can come from many places, but most commonly from the filesystem. Audio files such as `.wav` or `.ogg` can be loaded as `AudioStreams` and placed inside an `AudioStreamPlayer`.

When using these kind of files, you should choose which one is best for your specific use case:

- Audio files of type `.wav` are considerably big, but use very little amount of your CPU to play. Hundreds of them can be played simultaneously with little impact to performance. This format is usually best for short sound effects, as the importer will trim them and convert them to IMA-ADPCM.
- Audio files of type `.ogg` are much smaller, but use considerably more amount of CPU power to play back, so only a few can be played back (especially on mobile!). This format is usually best for music or long sound effect sequences. It also works well for voice at relatively low bitrates.

Keep in mind both `.wav` and `.ogg` generally don't contain looping information, so this information must be set on the import options of each:



There are other types of AudioStreams, such as `AudioStreamRandomPitch`, which takes an existing `AudioStream` and modulates the pitch every time it's played back randomly (great for some sound effects), and more will keep appearing in the future.

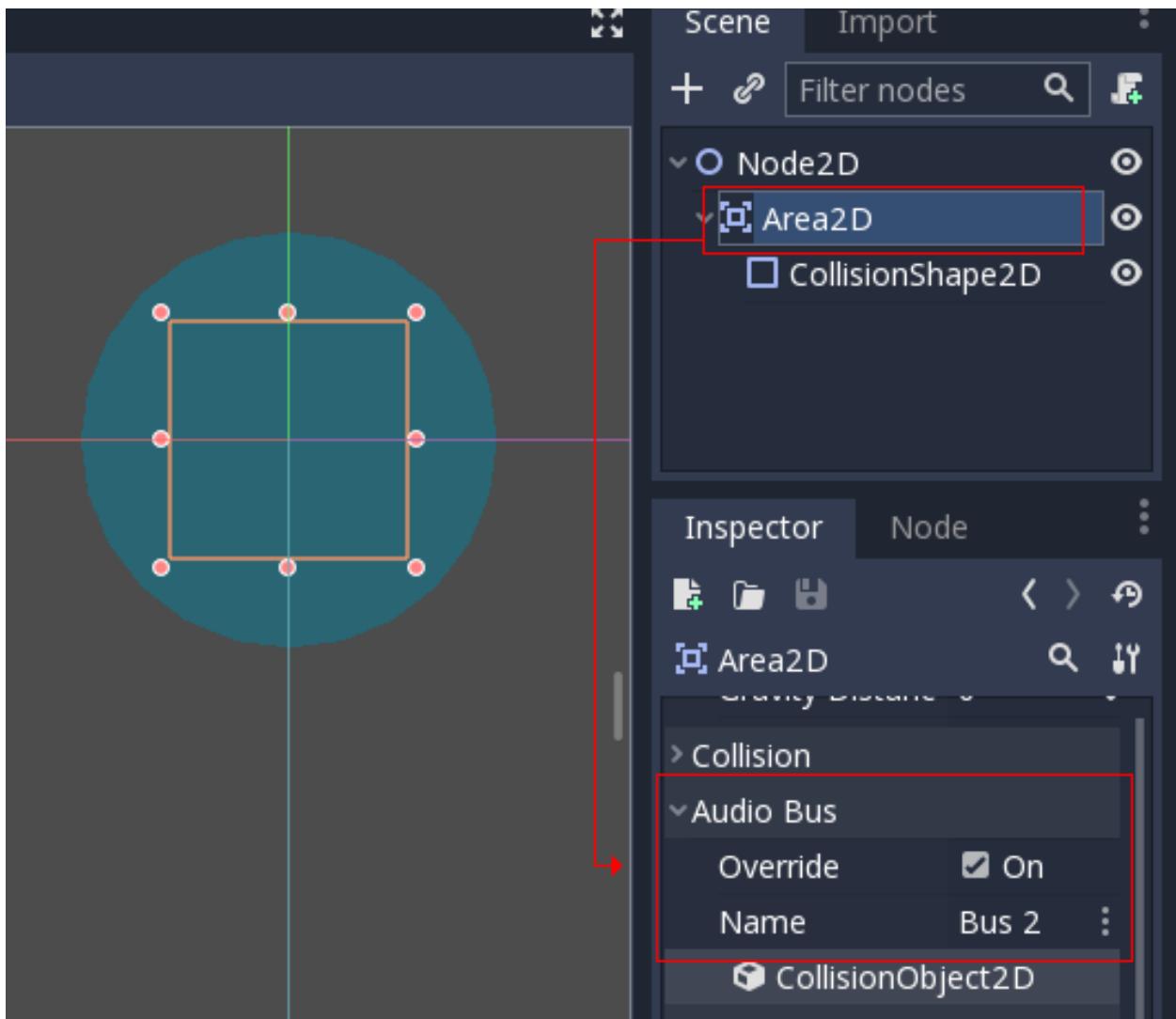
8.2.3 AudioStreamPlayer

This is the standard stream player, it can play to any given bus. In 5.1 sound, it can send to stereo mix or front speakers.

8.2.4 AudioStreamPlayer2D

This is a variant of `AudioStreamPlayer` but emits sound in a 2D positional environment. When close to the left of the screen, the panning will go left. When close to the right side, it will go right.

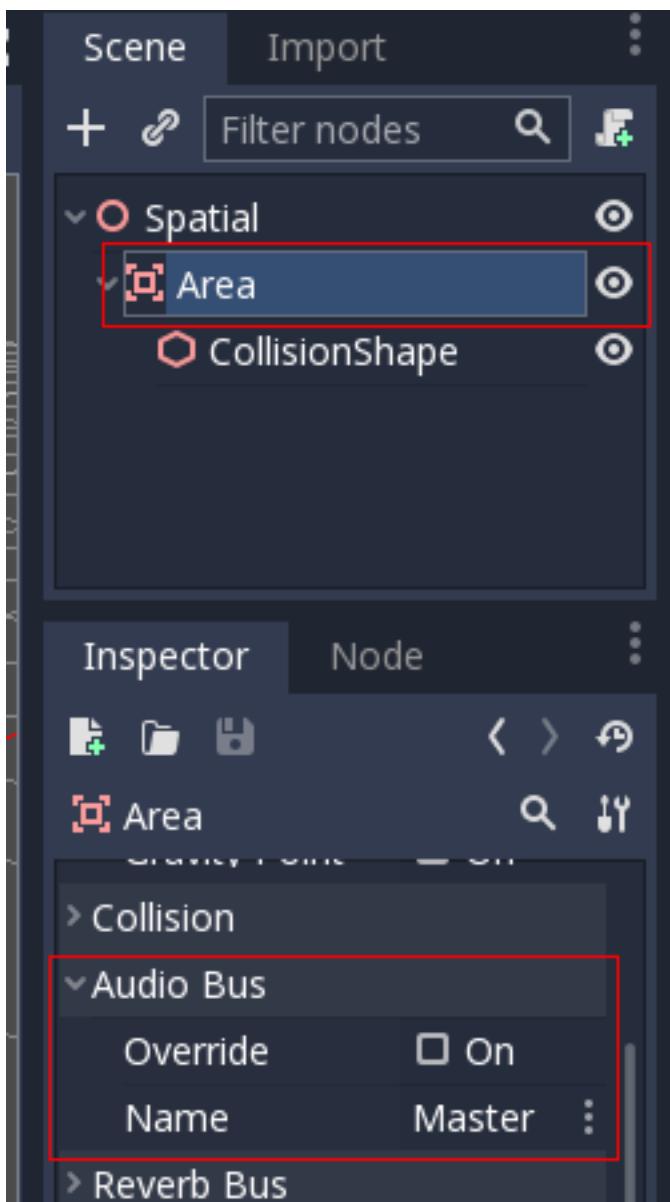
While it's possible to send these effects to specific audio buses, one of the best strategies is to use an `Area2D` to divert sound to a specific bus. This allows to create buses with different reverb or sound qualities and make the emitter will automatically send to them when entering the `Area2D` shapes.



8.2.5 AudioStreamPlayer3D

This is a variant of `AudioStreamPlayer` but emits sound in a 3D positional environment. Depending on the location of the player relative of the screen, it can position sound in Stereo, 5.1 or 7.1 depending on the chosen audio setup.

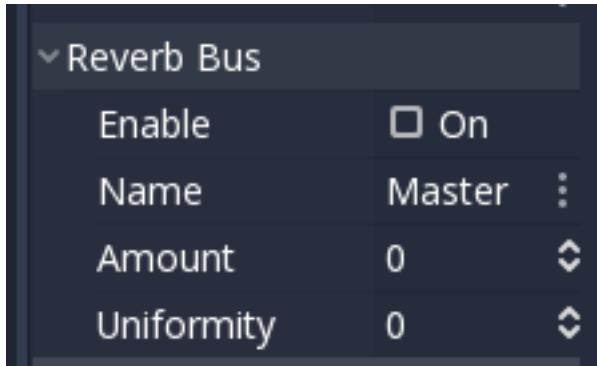
Similar to `AudioStreamPlayer2D`, an Area can divert the sound to an audio bus.



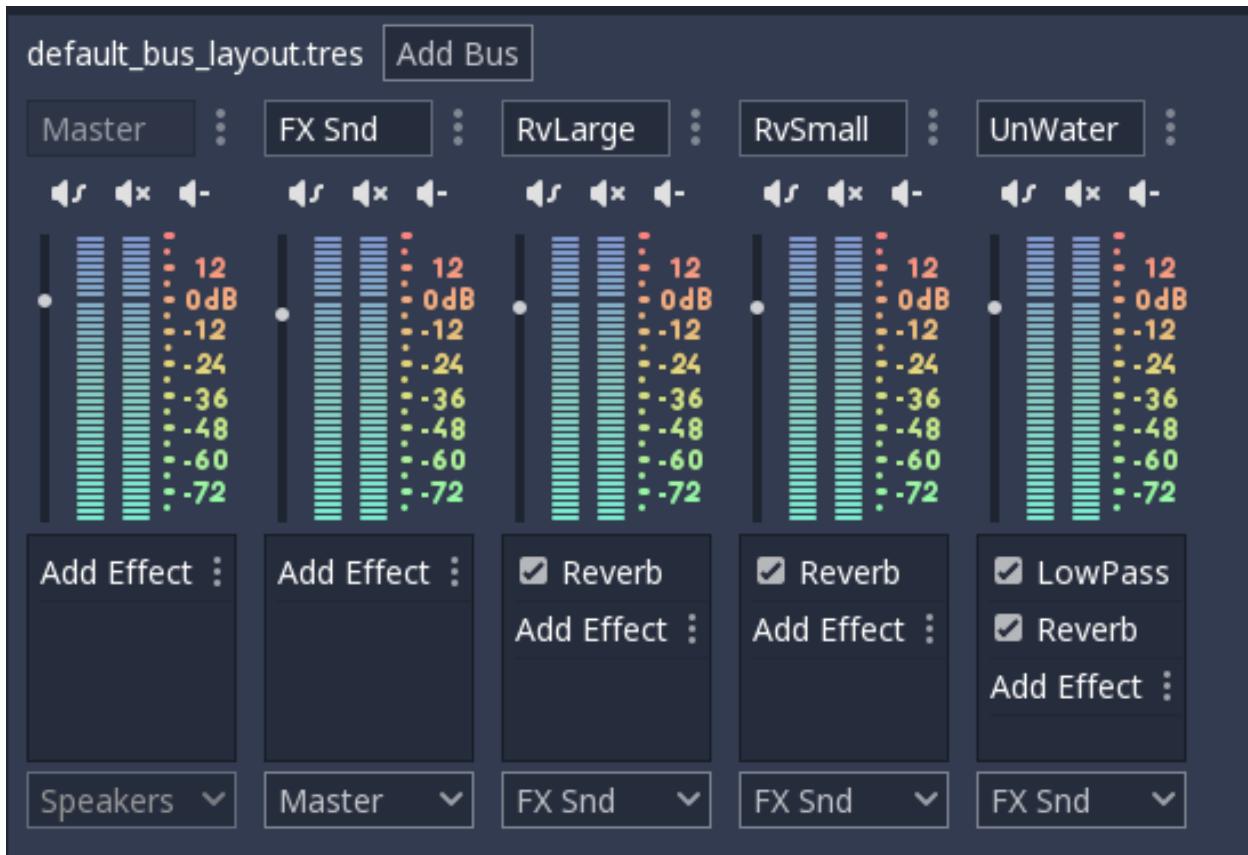
Unlike for 2D, the 3D version of `AudioStreamPlayer` has a few more advanced options:

Reverb Buses

Godot allows 3D Audio Streams that enter a specific *Area* to send dry and wet audio to separate buses. This is really useful when you have several reverb configurations for different types of rooms. This is done by enabling this type of reverb in the *Reverb Bus* section of *Area* properties:



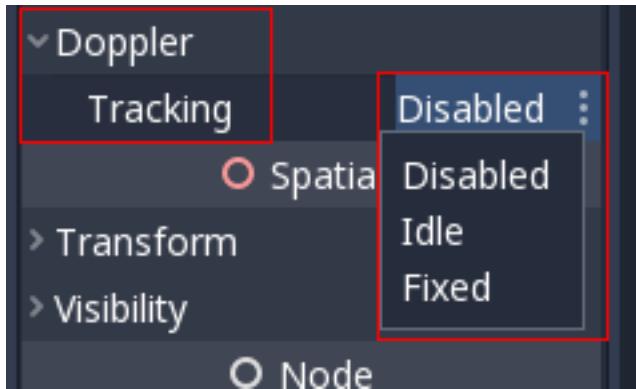
At the same time, a special bus layout is created where each area receives the reverb info from each area. Of course, an actual Reverb effect must be created in that bus for anything to happen:



The Area Reverb Bus section has also a specific parameter named “Uniformity”. Some types of rooms bounce sounds more than others (like for example, a typical warehouse), so reverberation can be heard almost uniformly across the room even though the source is far away. Playing around with this parameter can simulate that effect.

Doppler

When the relative velocity between an emitter and listener changes, this is perceived as an increase or decrease of the pitch shift. Godot can track changes in velocities of *AudioStreamPlayer3D* or *Camera*. Both have this property, which must be enabled manually:



Simply enable it by setting it depending on how objects will be moved (whether on regular *process* or *physics_process* step) and the tracking will happen automatically!

9.1 Physics introduction

Our world is made of tangible matter. In our world, a piano can't go through a wall when going into a house. It needs to use the door. Video games are often like the real world and Pac-Man can't go through the walls of his maze (although he can teleport from the left to the right side of the screen and back).

Anyway, moving sprites around is nice but one day they have to collide properly, so let's get to the point.

9.1.1 Shapes

The base collidable object in Godot's 2D world is a *Shape2D*. There are many types of shapes, all of them inherit this base class:

- *CircleShape2D*
- *RectangleShape2D*
- *CapsuleShape2D*
- *ConvexPolygonShape2D*
- *ConcavePolygonShape2D*
- etc. (there are others check the class list).

Shapes are of type *Resource*, but they can be created via code easily. For example:

```
# Create a circle
var c = CircleShape2D.new()
c.set_radius(20)

# Create a box
var b = RectangleShape2D.new()
b.set_extents(Vector2(20, 10))
```

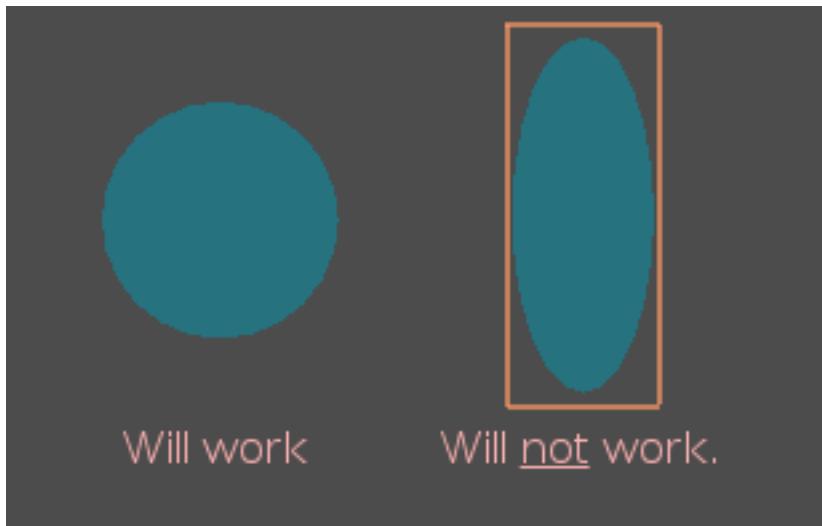
The main use for shapes is checking collision/intersection and getting resolution information. Shapes are mostly convex, (except the concavepolygon one, which is just a list of segments to check collision against). This collision check is done easily with the built-in functions like:

```
# Check if there is a collision between two shapes, each with a transform
if b.collide(b_xform, a, a_xform):
    print("OMG Collision!")
```

Godot will return correct collision and collision info from the different calls to the Shape2D api. Collision between all shapes and transforms can be done this way, or even obtaining contact information, motion casting, etc.

Transforming shapes

As seen before in the collide functions, 2D shapes in Godot can be transformed by using a regular *Transform2D* transform, meaning the functions can check for collisions while the shapes are scaled, moved and rotated. The only limitation to this is that shapes with curved sections (such as circle and capsule) can only be scaled uniformly. This means that circle or capsule shapes scaled in the form of an ellipse **will not work properly**. This is a limitation on the collision algorithm used (SAT), so make sure that your circle and capsule shapes are always scaled uniformly!



9.1.2 When problems begin

Even though this sounds good, reality is that collision detection alone is usually not enough in most scenarios. Many problems start arising as long as the development of the game is in progress:

Too many combinations!

Games have several dozens, hundreds, thousands! of objects that can collide and be collided. The typical approach is to test everything against everything in two for loops like this:

```
for i in colliders:
    for j in colliders:
        if (i.collides(j)):
            do_collision_code()
```

But this scales really bad. Let's imagine there are only 100 objects in the game. This means that $100 \times 100 = 10000$ collisions will need to be tested each frame. This is a lot!

Visual aid

Most of the time, creating a shape via code is not enough. We need to visually place it over a sprite, draw a collision polygon, etc. It is obvious that we need nodes to create the proper collision shapes in a scene.

Collision resolution

Imagine we solved the collision issue, we can tell easily and quickly which shapes overlap. If many of them are dynamic objects that move around, or move according to newtonian physics, solving a collision of multiple objects can be really difficult code-wise.

9.1.3 Introducing... Godot's physics engine!

To solve all these problems, Godot has a physics and collision engine that is well integrated into the scene system, yet it allows different levels and layers of functionality. The built-in physics engine can be used for:

- Simple Collision Detection: See *Shape2D* API.
- Scene Kinematics: Handle shapes, collisions, broadphase, etc as nodes. See *Area2D*.
- Scene Physics: Rigid bodies and constraints as nodes. See *RigidBody2D*, and the joint nodes.

Units of measure

It is often a problem when integrating a 2D physics engine to a game that such engines are optimized to work using meters as unit of measure. Godot uses a built-in custom 2D physics engine that is designed to function properly in pixels, so all units and default values used for stabilization are tuned for this, making development more straightforward.

9.1.4 CollisionObject2D

CollisionObject2D is the (virtual) base node for everything that can be collided in 2D. *Area2D*, *StaticBody2D*, *KinematicBody2D* and *RigidBody2D* all inherit from it. This node contains a list of shapes (*Shape2D*) and a relative transform. This means that all collisionable objects in Godot can use multiple shapes at different transforms (offset/scale/rotation). Just remember that, as mentioned before, **non-uniform scale will not work for circle and capsule shapes**.



A *CollisionObject2D* comes in different flavors: *StaticBody2D*, *RigidBody2D*, *KinematicBody2D* and *Area2D*. However, before we dig into them, let's have a look how to define the shape of a collision object. There are two special nodes for that.

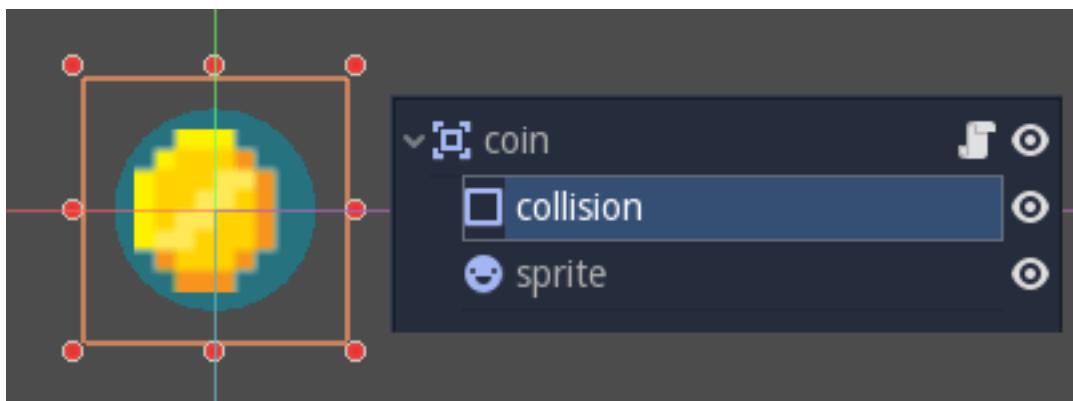
CollisionShape2D

This node is a helper node, which must be created as a direct child of a CollisionObject2D-derived node: *Area2D*, *StaticBody2D*, *KinematicBody2D*, *RigidBody2D*.

The purpose of a CollisionShape2D instance is to add collision shapes to its parent class. Multiple children of type CollisionShape2D can be added to a CollisionObject2D-derived object with the effect that the parent will simply get more collision shapes. When a CollisionShape2D is edited (or added, moved, deleted) the list of shapes in the parent node is updated.

At run time, though, this node does not exist and it can't be accessed with `get_node()`. This is because it is a helper node only for editing the collision shapes. To access the shapes created at runtime, use the CollisionObject2D API directly.

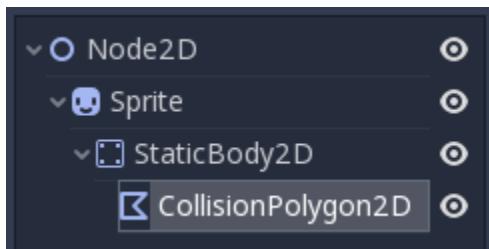
As an example, here's the scene from the platformer, containing an Area2D (named 'coin') having two children: a CollisionShape2D (named 'collision') and a sprite (called 'sprite'):



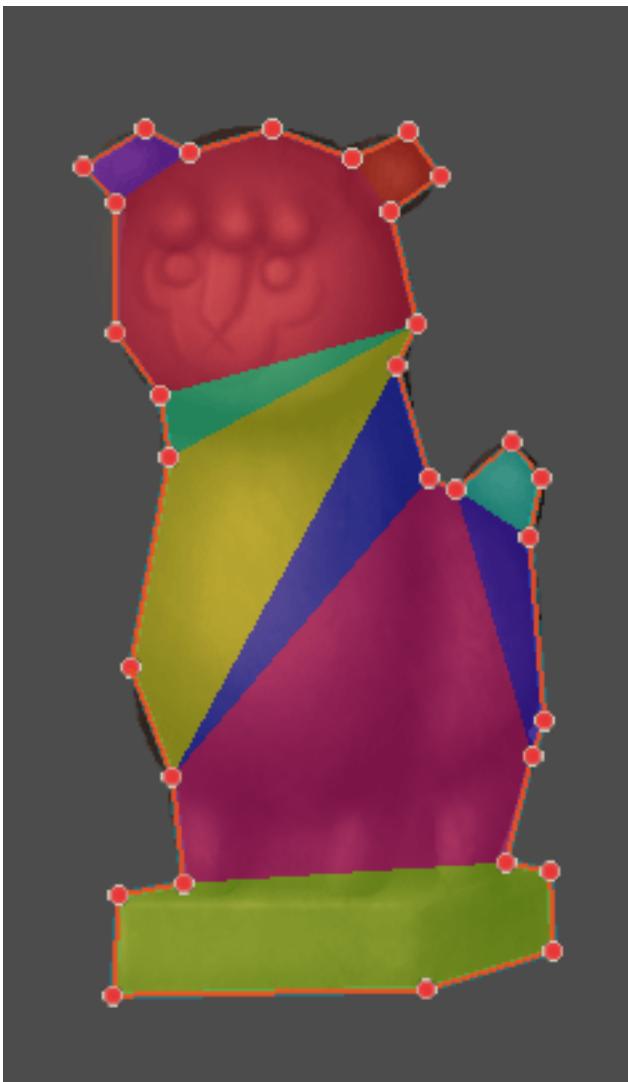
CollisionPolygon2D

This one is similar to CollisionShape2D, except that instead of assigning a shape, a polygon can be edited (drawn by the user) to determine the shape. The polygon can be convex or concave, it doesn't matter.

Here is another scene involving a CollisionPolygon2D: A StaticBody2D has been added as a child of a sprite so that the collision object moves together with the sprite. In turn, the CollisionPolygon is a child of StaticBody2D, meaning it adds collision shapes to it.



The CollisionPolygon2D will decompose the user-defined polygon into convex shapes (shapes can only be convex, remember?) before adding them to the CollisionObject2D. The following image shows such a decomposition:



Triggers

A CollisionShape2D or CollisionPolygon2D can be set as a trigger by setting the boolean flag with the same name. When used in a RigidBody2D or KinematicBody2D, “trigger” shapes take part in collision detection but are unaffected by physics (they don’t block movement etc). Defining a collision shape as a trigger is mostly useful in two situations:

- Collisions for a specific shape shall be disabled.
- An Area2D shall send `body_enter` and `body_exit` signals when the trigger shape enters it (useful in several situations).

StaticBody2D

The simplest node in the physics engine is the *StaticBody2D*. This node takes part in collision detection. However, it does not move or rotate after a collision, so physics do not influence them. The node is static. Other nodes can collide against it and will be influenced accordingly.

The platformer example uses StaticBody2D objects for the floors and walls. These are the static parts of a level and shall stay right where they are even if the player jumps onto them.

KinematicBody2D

Similar to StaticBody2D, objects of type *KinematicBody2D* are not affected by physics (although they take part in collision detection, of course). However, KinematicBody2D are not static but can be moved via code or an animation. They have two main uses:

- **Simulated Motion:** When these bodies are moved manually, either from code or from an *AnimationPlayer* (with process mode set to physics!), the physics will automatically compute an estimate of their linear and angular velocity. This makes them very useful for moving platforms or other AnimationPlayer-controlled objects (like a door, a bridge that opens, etc). As an example, the 2d/platformer demo uses them for moving platforms.
- **Kinematic Characters:** KinematicBody2D also has an API for moving objects (the `move()` function) while performing collision tests. This makes them really useful to implement characters that collide against a world, but that don't require advanced physics. There is a tutorial about it *Kinematic Character (2D)*.

RigidBody2D

This type of body simulates Newtonian physics. It has mass, friction, bounce, and the (0,0) coordinates simulate the center of mass. When real physics are needed, *RigidBody2D* is the node to use. The motion of this body is affected by gravity and/or other bodies.

Rigid bodies are usually active all the time, but when they end up in resting position and don't move for a while, they are put to sleep until something else wakes them up. This saves an enormous amount of CPU.

RigidBody2D nodes update their transform constantly, as it is generated by the simulation from a position, linear velocity and angular velocity. As a result, [STRIKEOUT:this node can't be scaled]. Scaling the children nodes should work fine though.

A RigidBody2D has a `Mode` flag to change its behavior (something which is very common in games). It can behave like a rigid body, a character (a rigid body without the ability to rotate so that it is always upright), a kinematic body or a static body. This flag can be changed dynamically according to the current situation. For example, an enemy frozen by an ice beam becomes a static body.

The best way to interact with a RigidBody2D is during the force integration callback by overriding the function

```
func _integrate_forces(state):  
    [use state to change the object]
```

The “state” parameter is of type *Physics2DDirectBodyState*. Please do not use the state object outside the callback as it will result in an error.

During the evaluation of the aforementioned function, the physics engine synchronizes state with the scene and allows full modification of the object's parameters. Since physics may run in its own thread, parameter changes outside that callback will not take place until the next frame.

Note: When a RigidBody goes to sleep then the `_integrate_forces()` method will not be called (I.E. they act like a static body until a collision or a force is applied to them). To override this behavior you will need to keep the rigid body “awake” by creating a collision, applying a force to it (e.g. `linear_velocity`) or by disabling the `can_sleep` property (see `can_sleep`). Be aware that this can have an effect on performance.

9.1.5 Contact reporting

In general, RigidBody2D will not keep track of the contacts, because this can require a huge amount of memory if thousands of rigid bodies are in the scene. To get contacts reported, simply increase the amount of the “contacts

reported” property from zero to a meaningful value (depending on how many you are expecting to get). The contacts can be later obtained via the *Physics2DDirectBodyState.get_contact_count()* and related functions.

Contact monitoring via signals is also available (signals similar to the ones in Area2D, described below) via a boolean property.

Area2D

Areas in Godot physics have three main roles:

1. Override the space parameters for objects entering them (ie. gravity, gravity direction, gravity type, density, etc).
2. Monitor when rigid or kinematic bodies enter or exit the area.
3. Monitor other areas (this is the simplest way to get overlap test)

The second function is the most common. For it to work, the “monitoring” property must be enabled (it is by default). There are two types of signals emitted by this node:

```
# Simple, high level notification
body_enter(body:PhysicsBody2D)
body_exit(body:PhysicsBody2D)
area_enter(area:Area2D)
area_exit(body:Area2D)

# Low level shape-based notification
# Notifies which shape specifically in both the body and area are in contact
body_enter_shape(body_id:int, body:PhysicsBody2D, body_shape_index:int, area_shape_
↳ index:idx)
body_exit_shape(body_id:int, body:PhysicsBody2D, body_shape_index:int, area_shape_
↳ index:idx)
area_enter_shape(area_id:int, area:Area2D, area_shape_index:int, self_shape_index:idx)
area_exit_shape(area_id:int, area:Area2D, area_shape_index:int, self_shape_index:idx)
```

By default, areas also receive mouse/touchscreen input, providing a lower-level way than controls to implement this kind of input in a game. Bodies support this but it's disabled by default.

In case of overlap, who receives collision information?

Remember that not every combination of two bodies can “report” contacts. Static bodies are passive and will not report contacts when hit. Kinematic Bodies will report contacts but only against Rigid/Character bodies. Area2D will report overlap (not detailed contacts) with bodies and with other areas. The following table should make it more visual:

Type	RigidBody	CharacterBody	KinematicBody	StaticBody	Area
RigidBody	Both	Both	Both	Rigidbody	Area
CharacterBody	Both	Both	Both	CharacterBody	Area
KinematicBody	Both	Both	None	None	Area
StaticBody	RigidBody	CharacterBody	None	None	None
Area	Area	Area	Area	None	Both

9.1.6 Physics global variables

A few global variables can be tweaked in the project settings for adjusting how 2D physics works:

Physics Engine	DEFAULT
Thread Model	Single-Safe
Sleep Threshold Linear	2
Sleep Threshold Angular	0.14
Time Before Sleep	0.5
Bp Hash Table Size	4096
Cell Size	128
Large Object Surface Threshold In Cells	512
Default Gravity	98
Default Gravity Vector	(0, 1)
Default Linear Damp	0.1
Default Angular Damp	1

Leaving them alone is best (except for the gravity, that needs to be adjusted in most games), but there is one specific parameter that might need tweaking which is the “cell_size”. Godot 2D physics engine used by default a space hashing algorithm that divides space in cells to compute close collision pairs more efficiently.

If a game uses several colliders that are really small and occupy a small portion of the screen, it might be necessary to shrink that value (always to a power of 2) to improve efficiency. Likewise if a game uses few large colliders that span a huge map (of several screens of size), increasing that value a bit might help save resources.

9.1.7 Fixed process callback

The physics engine may spawn multiple threads to improve performance, so it can use up to a full frame to process physics. Because of this, when accessing physics variables such as position, linear velocity, etc. they might not be representative of what is going on in the current frame.

To solve this, Godot has a physics process callback, which is like process but it’s called before each physics step always at the same frame rate (by default 60 times per second). During this time, the physics engine is in *synchronization* state and can be accessed directly and without delays.

To enable a physics process callback, use the `set_physics_process()` function, example:

```
extends KinematicBody2D

func _physics_process(delta):
    move(direction * delta)

func _ready():
    set_physics_process(true)
```

9.1.8 Casting rays and motion queries

It is very often desired to “explore” the world around from our code. Throwing rays is the most common way to do it. The simplest way to do this is by using the RayCast2D node, which will throw a ray every frame and record the intersection.

At the moment there isn’t a high level API for this, so the physics server must be used directly. For this, the `Physics2DDirectSpaceState` class must be used. To obtain it, the following steps must be taken:

1. It must be used inside the `_physics_process()` callback, or at `_integrate_forces()`
2. The 2D RIDs for the space and physics server must be obtained.

The following code should work:

```
func _physics_process(delta):
    var space = get_world_2d().get_space()
    var space_state = Physics2DServer.space_get_direct_state(space)
```

Enjoy doing space queries!

9.2 Ray-casting

9.2.1 Introduction

One of the most common tasks in game development is casting a ray (or custom shaped object) and checking what it hits. This enables complex behaviors, AI, etc. to take place. This tutorial will explain how to do this in 2D and 3D.

Godot stores all the low level game information in servers, while the scene is just a frontend. As such, ray casting is generally a lower-level task. For simple raycasts, node such as *RayCast* and *RayCast2D* will work, as they will return every frame what the result of a raycast is.

Many times, though, ray-casting needs to be a more interactive process so a way to do this by code must exist.

9.2.2 Space

In the physics world, Godot stores all the low level collision and physics information in a *space*. The current 2d space (for 2D Physics) can be obtained by calling *CanvasItem.get_world_2d().get_space()*. For 3D, it's *Spatial.get_world().get_space()*.

The resulting space *RID* can be used in *PhysicsServer* and *Physics2DServer* respectively for 3D and 2D.

9.2.3 Accessing space

Godot physics runs by default in the same thread as game logic, but may be set to run on a separate thread to work more efficiently. Due to this, the only time accessing space is safe is during the *Node._physics_process()* callback. Accessing it from outside this function may result in an error due to space being *locked*.

To perform queries into physics space, the *Physics2DDirectSpaceState* and *PhysicsDirectSpaceState* must be used.

In code, for 2D spacestate, this code must be used:

```
func _physics_process(delta):
    var space_rid = get_world_2d().get_space()
    var space_state = Physics2DServer.space_get_direct_state(space_rid)
```

Of course, there is a simpler shortcut:

```
func _physics_process(delta):
    var space_state = get_world_2d().get_direct_space_state()
```

For 3D:

```
func _physics_process(delta):
    var space_state = get_world().get_direct_space_state()
```

9.2.4 Raycast query

For performing a 2D raycast query, the method *Physics2DDirectSpaceState.intersect_ray()* must be used, for example:

```
func _physics_process(delta):
    var space_state = get_world().get_direct_space_state()
    # use global coordinates, not local to node
    var result = space_state.intersect_ray( Vector2(0,0), Vector2(50,100) )
```

Result is a dictionary. If the ray didn't hit anything, the dictionary will be empty. If it did hit something it will contain collision information:

```
if (not result.empty()):
    print("Hit at point: ",result.position)
```

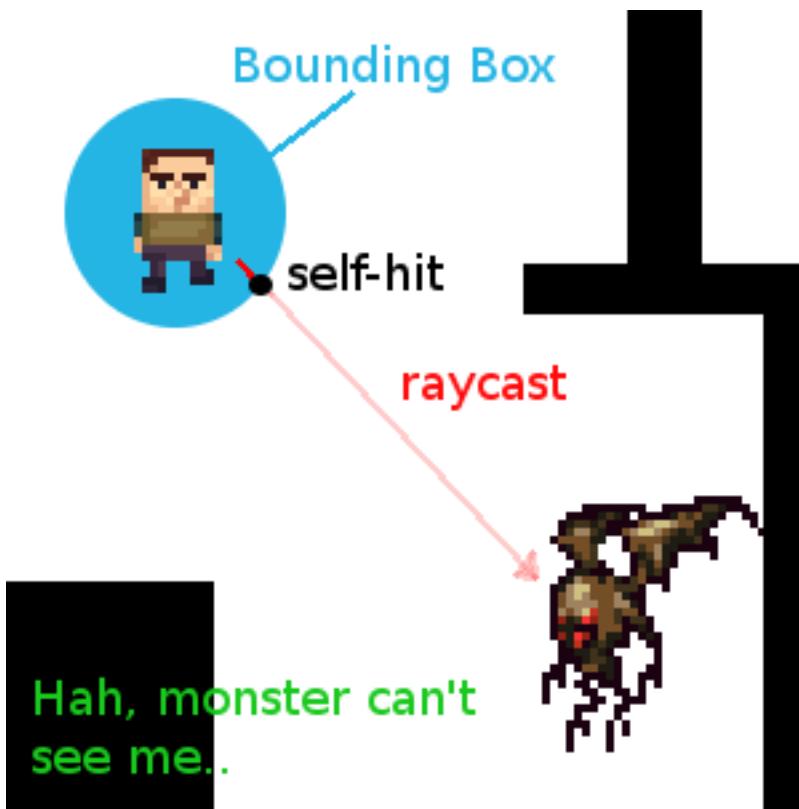
The collision result dictionary, when something hit, has this format:

```
{
    position:Vector2 # point in world space for collision
    normal:Vector2 # normal in world space for collision
    collider:Object # Object collided or null (if unassociated)
    collider_id:ObjectID # Object it collided against
    rid:RID # RID it collided against
    shape:int # shape index of collider
    metadata:Variant() # metadata of collider
}

# in case of 3D, Vector3 is returned.
```

9.2.5 Collision exceptions

It is a very common case to attempt casting a ray from a character or another game scene to try to infer properties of the world around it. The problem with this is that the same character has a collider, so the ray can never leave the origin (it will keep hitting its own collider), as evidenced in the following image.



To avoid self-intersection, the `intersect_ray()` function can take an optional third parameter which is an array of exceptions. This is an example of how to use it from a `KinematicBody2D` or any other `CollisionObject` based node:

```
extends KinematicBody2D

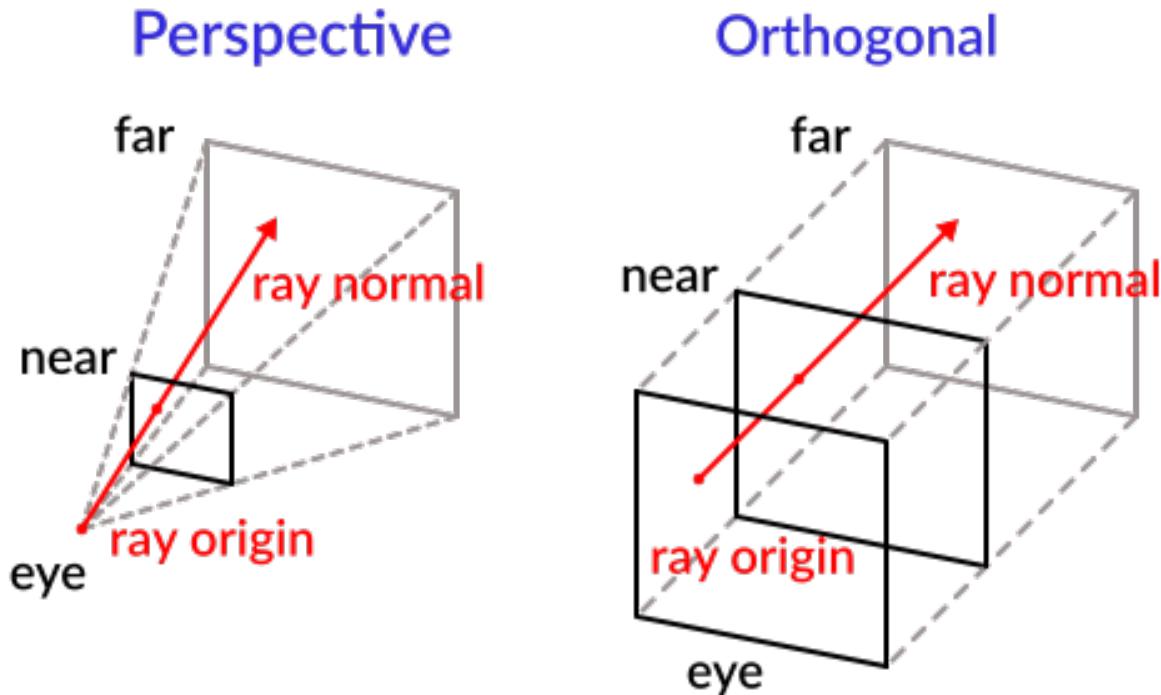
func _physics_process(delta):
    var space_state = get_world().get_direct_space_state()
    var result = space_state.intersect_ray( get_global_pos(), enemy_pos, [ self ] )
```

The extra argument is a list of exceptions, can be objects or RIDs.

9.2.6 3D ray casting from screen

Casting a ray from screen to 3D physics space is useful for object picking. There is not much of a need to do this because `CollisionObject` has an “`input_event`” signal that will let you know when it was clicked, but in case there is any desire to do it manually, here’s how.

To cast a ray from the screen, the `Camera` node is needed. Camera can be in two projection modes, perspective and orthogonal. Because of this, both the ray origin and direction must be obtained. (origin changes in orthogonal, while direction changes in perspective):



To obtain it using a camera, the following code can be used:

```
const ray_length = 1000

func _input(ev):
    if ev is InputEventMouseButton and ev.pressed and ev.button_index==1:

        var camera = get_node("camera")
        var from = camera.project_ray_origin(ev.position)
        var to = from + camera.project_ray_normal(ev.position) * ray_length
```

Of course, remember that during `_input()`, space may be locked, so save your query for `_physics_process()`.

9.3 Kinematic Character (2D)

9.3.1 Introduction

Yes, the name sounds strange. “Kinematic Character”. What is that? The reason is that when physics engines came out, they were called “Dynamics” engines (because they dealt mainly with collision responses). Many attempts were made to create a character controller using the dynamics engines but it wasn’t as easy as it seems. Godot has one of the best implementations of dynamic character controller you can find (as it can be seen in the 2d/platformer demo), but using it requires a considerable level of skill and understanding of physics engines (or a lot of patience with trial and error).

Some physics engines such as Havok seem to swear by dynamic character controllers as the best alternative, while others (PhysX) would rather promote the Kinematic one.

So, what is really the difference?:

- A **dynamic character controller** uses a rigid body with infinite inertial tensor. Basically, it's a rigid body that can't rotate. Physics engines always let objects collide, then solve their collisions all together. This makes dynamic character controllers able to interact with other physics objects seamlessly (as seen in the platformer demo), however these interactions are not always predictable. Collisions also can take more than one frame to be solved, so a few collisions may seem to displace a tiny bit. Those problems can be fixed, but require a certain amount of skill.
- A **kinematic character controller** is assumed to always begin in a non-colliding state, and will always move to a non colliding state. If it starts in a colliding state, it will try to free itself (like rigid bodies do) but this is the exception, not the rule. This makes their control and motion a lot more predictable and easier to program. However, as a downside, they can't directly interact with other physics objects (unless done by hand in code).

This short tutorial will focus on the kinematic character controller. Basically, the oldschool way of handling collisions (which is not necessarily simpler under the hood, but well hidden and presented as a nice and simple API).

9.3.2 Physics process

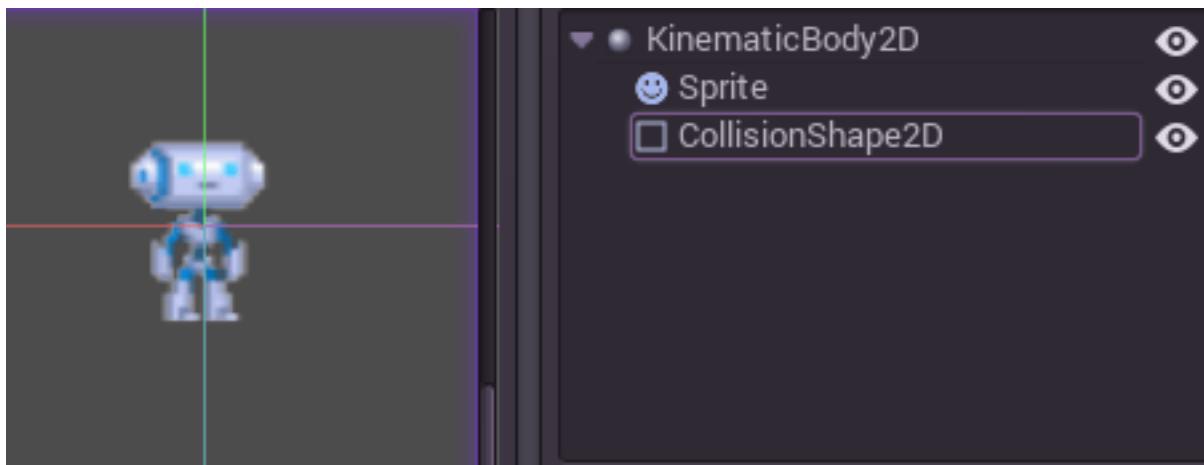
To manage the logic of a kinematic body or character, it is always advised to use physics process, because it's called before physics step and its execution is in sync with physics server, also it is called the same amount of times per second, always. This makes physics and motion calculation work in a more predictable way than using regular process, which might have spikes or lose precision if the frame rate is too high or too low.

```
extends KinematicBody2D

func _physics_process(delta):
    pass
```

9.3.3 Scene setup

To have something to test, here's the scene (from the tilemap tutorial): `kbscene.zip`. We'll be creating a new scene for the character. Use the robot sprite and create a scene like this:



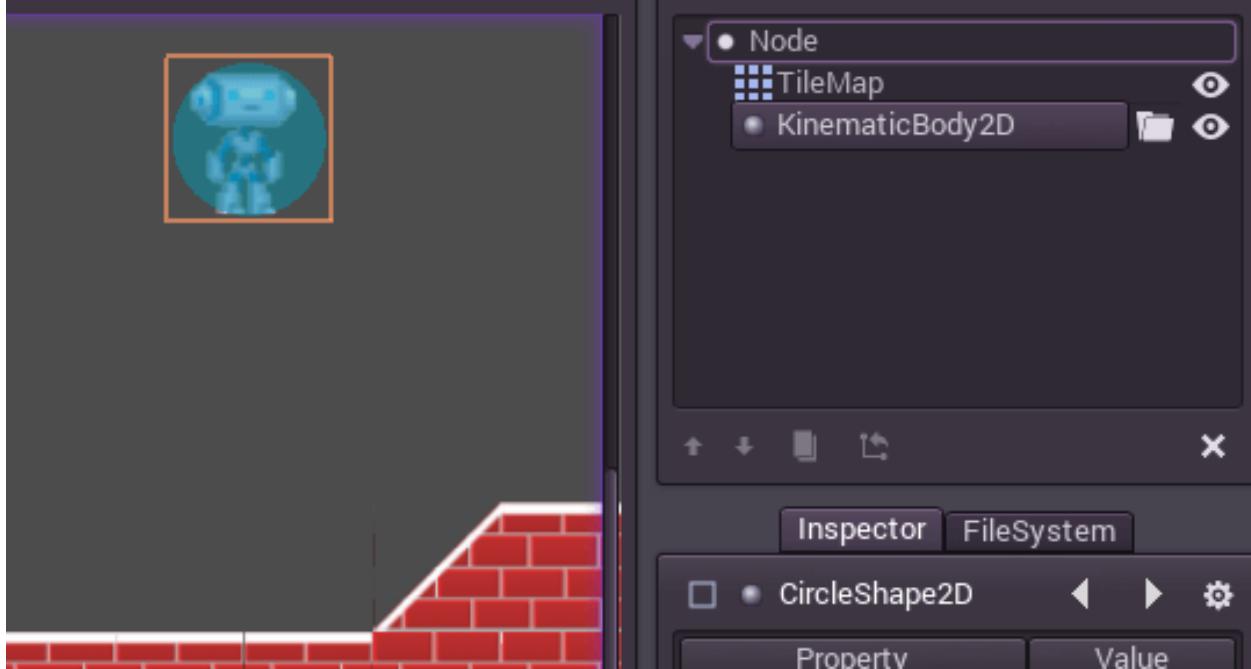
Let's add a circular collision shape to the collision body, create a new CircleShape2D in the shape property of CollisionShape2D. Set the radius to 30:



Note: As mentioned before in the physics tutorial, the physics engine can't handle scale on most types of shapes (only collision polygons, planes and segments work), so always change the parameters (such as radius) of the shape instead of scaling it. The same is also true for the kinematic/rigid/static bodies themselves, as their scale affect the shape scale.

Now create a script for the character, the one used as an example above should work as a base.

Finally, instance that character scene in the tilemap, and make the map scene the main one, so it runs when pressing play.



9.3.4 Moving the Kinematic character

Go back to the character scene, and open the script, the magic begins now! Kinematic body will do nothing by default, but it has a really useful function called `KinematicBody2D.move_and_collide()`. This function takes a `Vector2` as an

argument, and tries to apply that motion to the kinematic body. If a collision happens, it stops right at the moment of the collision.

So, let's move our sprite downwards until it hits the floor:

```
extends KinematicBody2D

func _physics_process(delta):
    move_and_collide(Vector2(0,1)) # Move down 1 pixel per physics frame
```

The result is that the character will move, but stop right when hitting the floor. Pretty cool, huh?

The next step will be adding gravity to the mix, this way it behaves a little more like an actual game character:

```
extends KinematicBody2D

const GRAVITY = 200.0
var velocity = Vector2()

func _physics_process(delta):
    velocity.y += delta * GRAVITY
    var motion = velocity * delta
    move_and_collide(motion)
```

Now the character falls smoothly. Let's make it walk to the sides, left and right when touching the directional keys. Remember that the values being used (for speed at least) is pixels/second.

This adds simple walking support by pressing left and right:

```
extends KinematicBody2D

const GRAVITY = 200.0
const WALK_SPEED = 200

var velocity = Vector2()

func _physics_process(delta):
    velocity.y += delta * GRAVITY

    if (Input.is_action_pressed("ui_left")):
        velocity.x = -WALK_SPEED
    elif (Input.is_action_pressed("ui_right")):
        velocity.x = WALK_SPEED
    else:
        velocity.x = 0

    var motion = velocity * delta

    # The second parameter of move_and_slide is the normal pointing up.
    # In the case of a 2d platformer, in Godot upward is negative y, which translates
    # to -1 as a normal.
    move_and_slide(motion, Vector2(0, -1))
```

And give it a try.

This is a good starting point for a platformer. A more complete demo can be found in the demo zip distributed with the engine, or in the https://github.com/godotengine/godot-demo-projects/tree/master/2d/kinematic_character.

CHAPTER 10

Math

10.1 Vector math

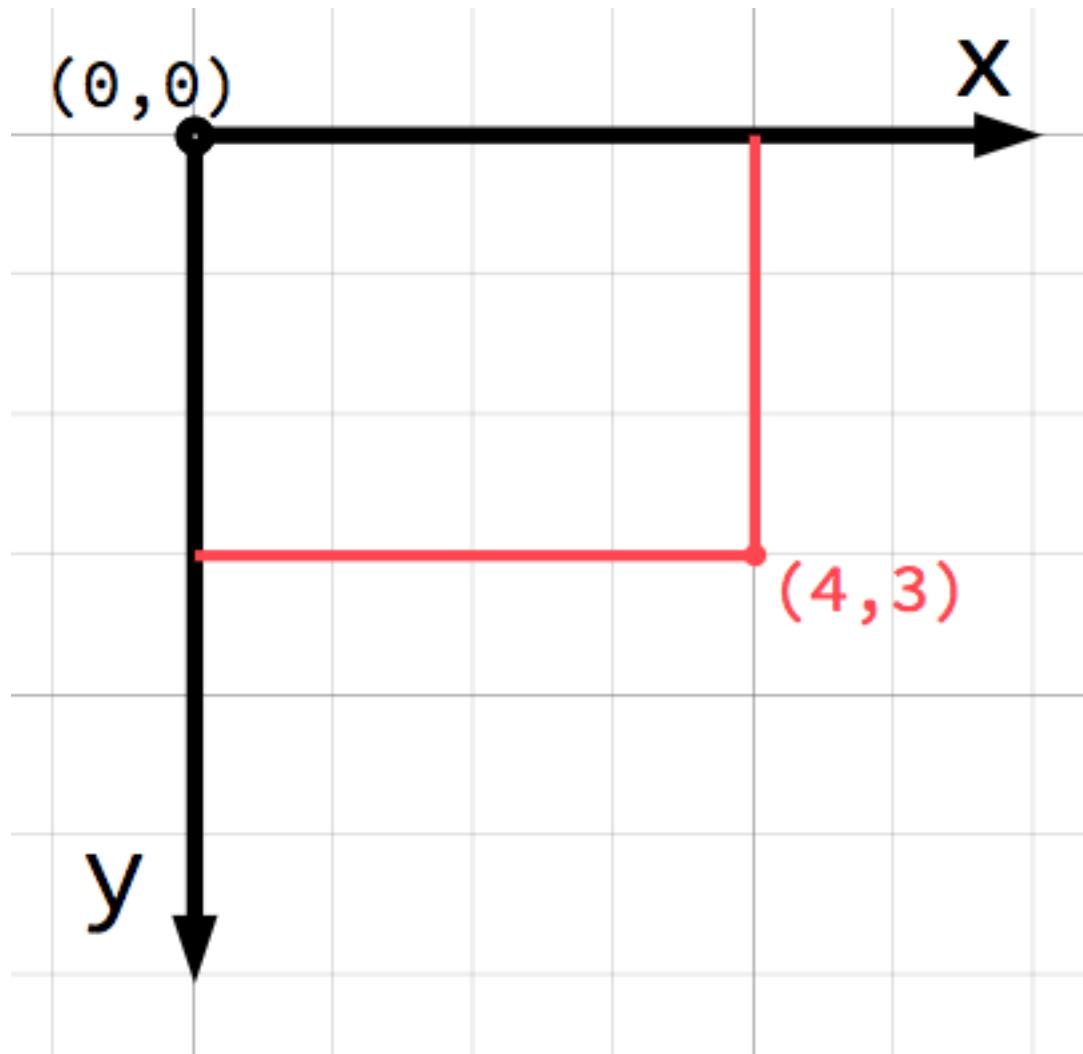
10.1.1 Introduction

This tutorial is a short and practical introduction to linear algebra as it applies to game development. Linear algebra is the study of vectors and their uses. Vectors have many applications in both 2D and 3D development and Godot uses them extensively. Developing a good understanding of vector math is essential to becoming a strong game developer.

Note: This tutorial is **not** a formal textbook on linear algebra. We will only be looking at how it is applied to game development. For a broader look at the mathematics, see <https://www.khanacademy.org/math/linear-algebra>

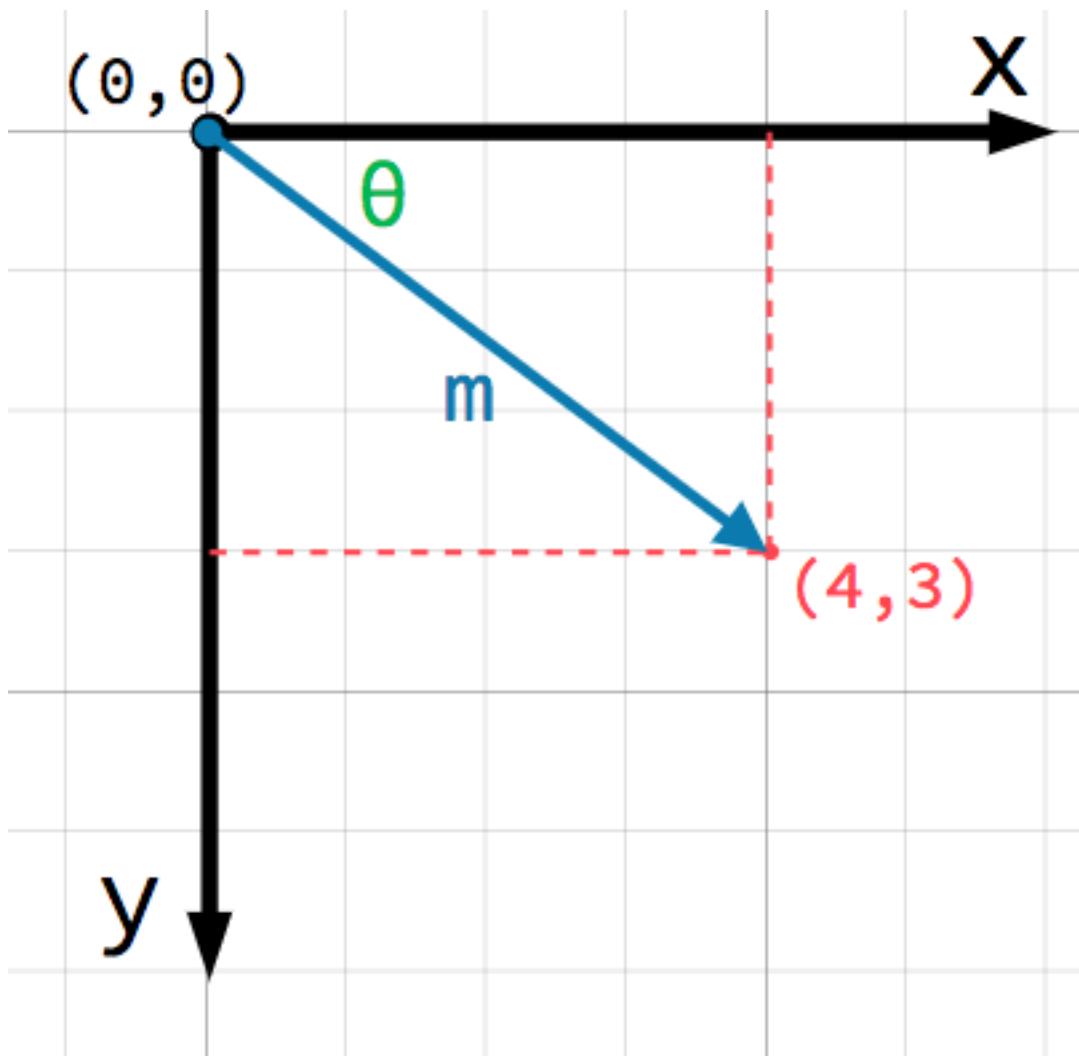
10.1.2 Coordinate systems (2D)

In 2D space, coordinates are defined using a horizontal axis (x) and a vertical axis (y). A particular position in 2D space is written as a pair of values such as $(4, -3)$.



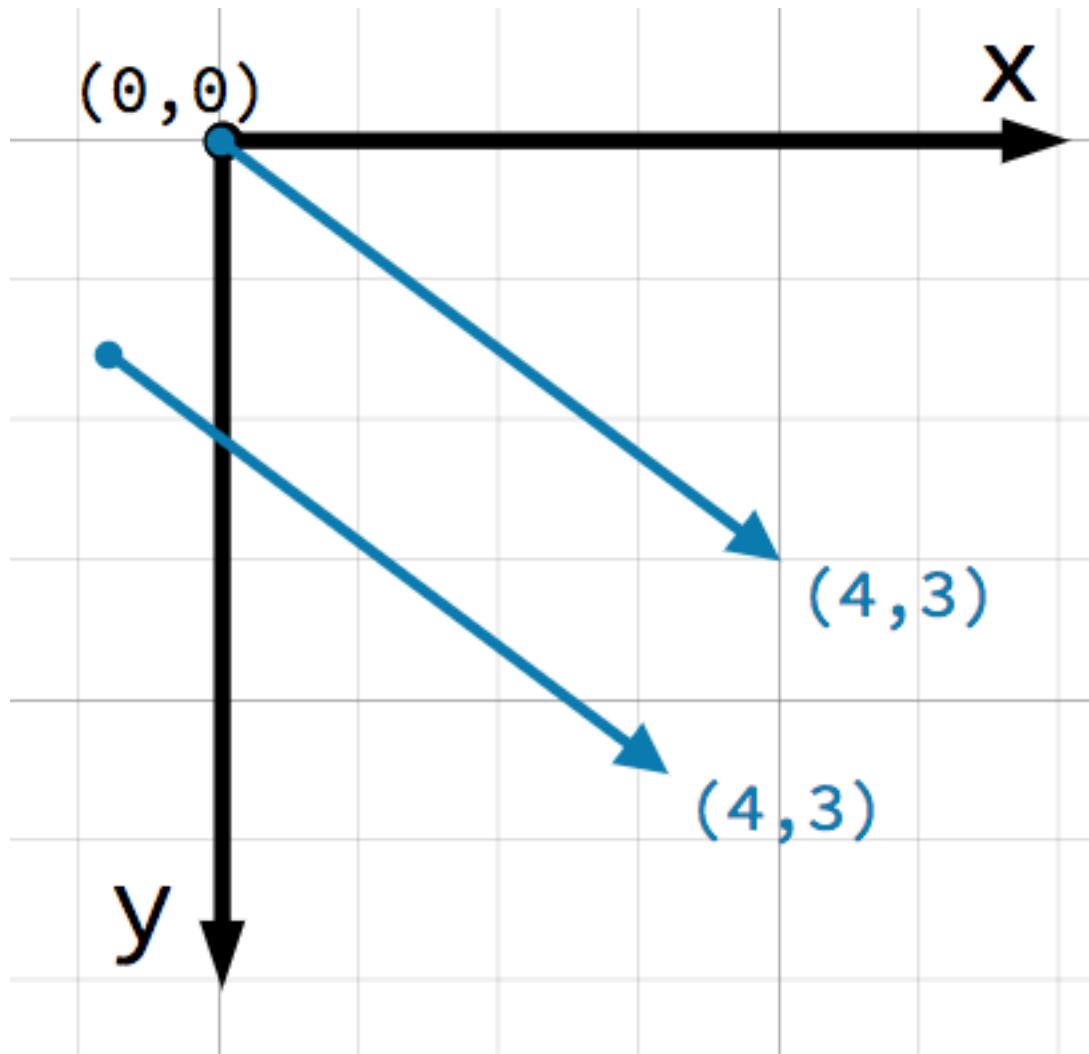
Note: If you're new to computer graphics, it might seem odd that the positive y axis points **downwards** instead of upwards, as you probably learned in math class. However, this is common in most computer graphics applications.

Any position in the 2D plane can be identified by a pair of numbers in this way. However, we can also think of the position $(4, 3)$ as an **offset** from the $(0, 0)$ point, or **origin**. Draw an arrow pointing from the origin to the point:



This is a **vector**. A vector represents a lot of useful information. As well as telling us that the point is at $(4, 3)$, we can also think of it as an angle θ and a length (or magnitude) m . In this case, the arrow is a **position vector** - it denotes a position in space, relative to the origin.

A very important point to consider about vectors is that they only represent **relative** direction and magnitude. There is no concept of a vector's position. The following two vectors are identical:



Both vectors represent a point 4 units to the right and 3 units below some starting point. It does not matter where on the plane you draw the vector, it always represents a relative direction and magnitude.

10.1.3 Vector Operations

You can use either method (x and y coordinates or angle and magnitude) to refer to a vector, but for convenience programmers typically use the coordinate notation. For example, in Godot the origin is the top-left corner of the screen, so to place a 2D node named Node2D 400 pixels to the right and 300 pixels down, use the following code:

```
$Node2D.position = Vector2(400, 300)
```

Godot supports both *Vector2* and *Vector3* for 2D and 3D usage respectively. The same mathematical rules discussed in this article apply for both types.

- Member access

The individual components of the vector can be accessed directly by name.

```
# create a vector with coordinates (2, 5)
var a = Vector2(2, 5)
# create a vector and assign x and y manually
```

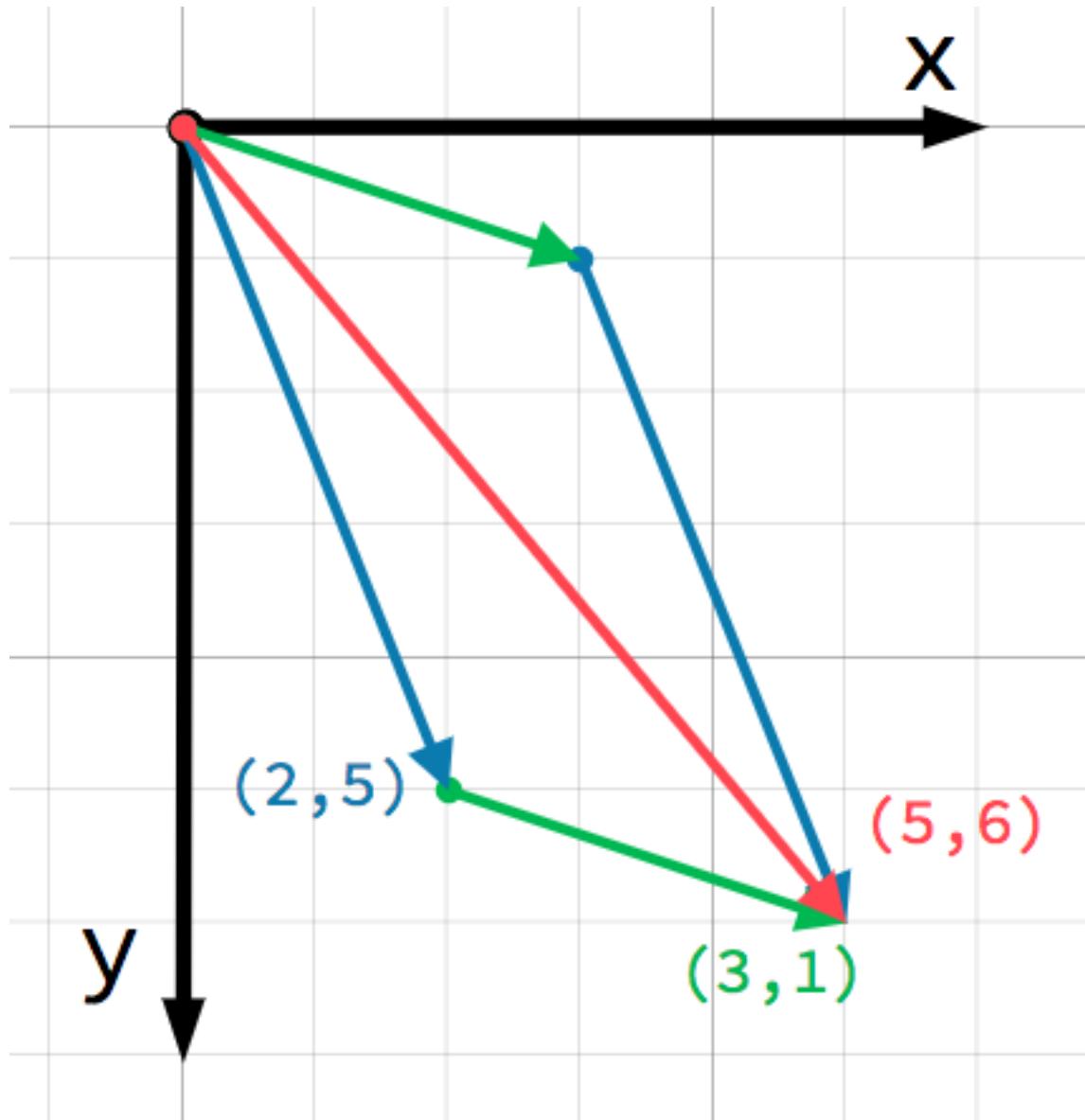
```
var b = Vector2()
b.x = 3
b.y = 1
```

- Adding vectors

When adding or subtracting two vectors, the corresponding components are added:

```
var c = a + b # (2, 5) + (3, 1) = (5, 6)
```

We can also see this visually by adding the second vector at the end of the first:



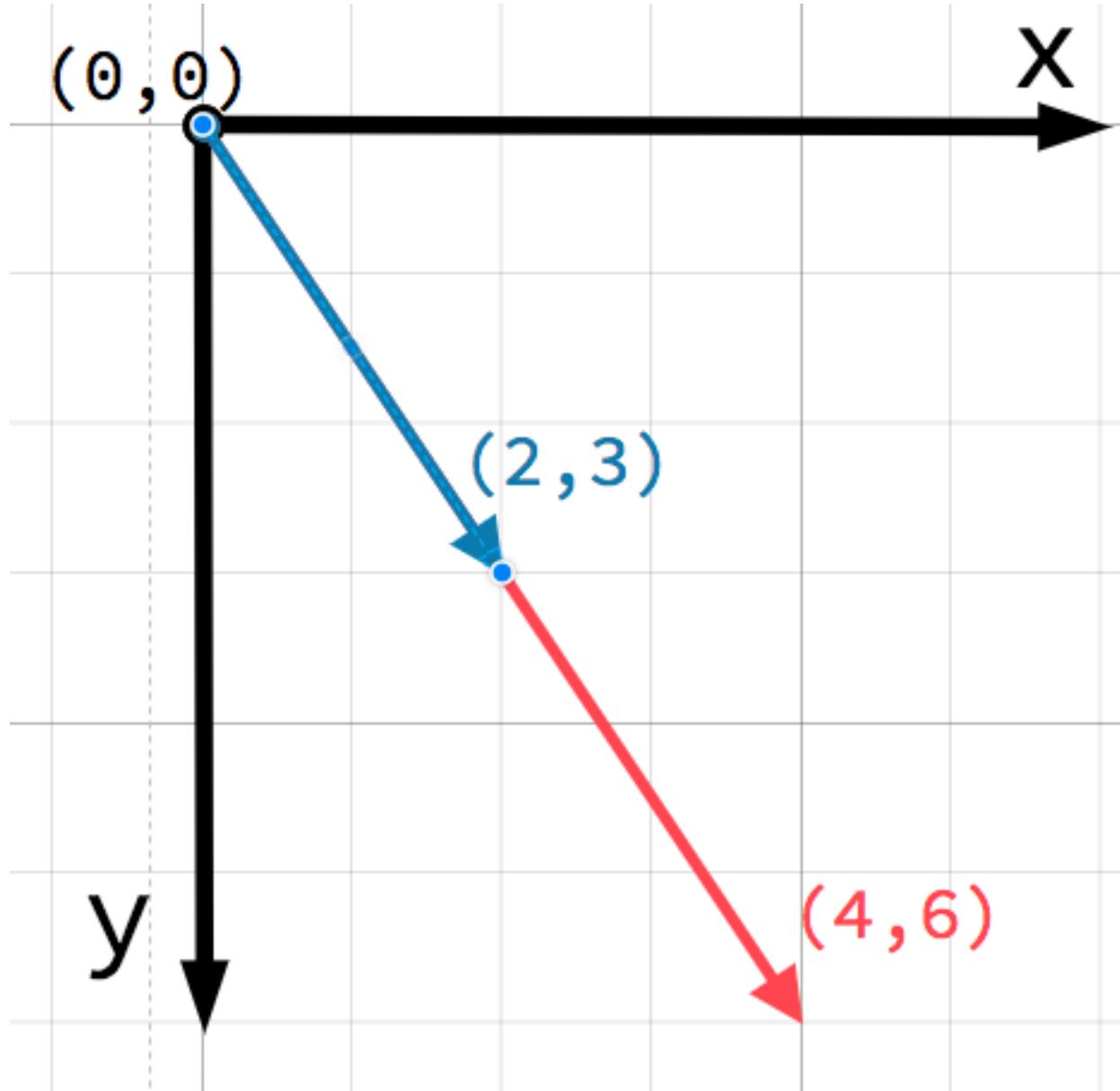
Note that adding $a + b$ gives the same result as $b + a$.

- Scalar multiplication

Note: Vectors represent both direction and magnitude. A value representing only magnitude is called a **scalar**.

A vector can be multiplied by a **scalar**:

```
var c = a * 2 # (2, 5) * 2 = (4, 10)
var d = b / 3 # (3, 6) / 3 = (1, 2)
```



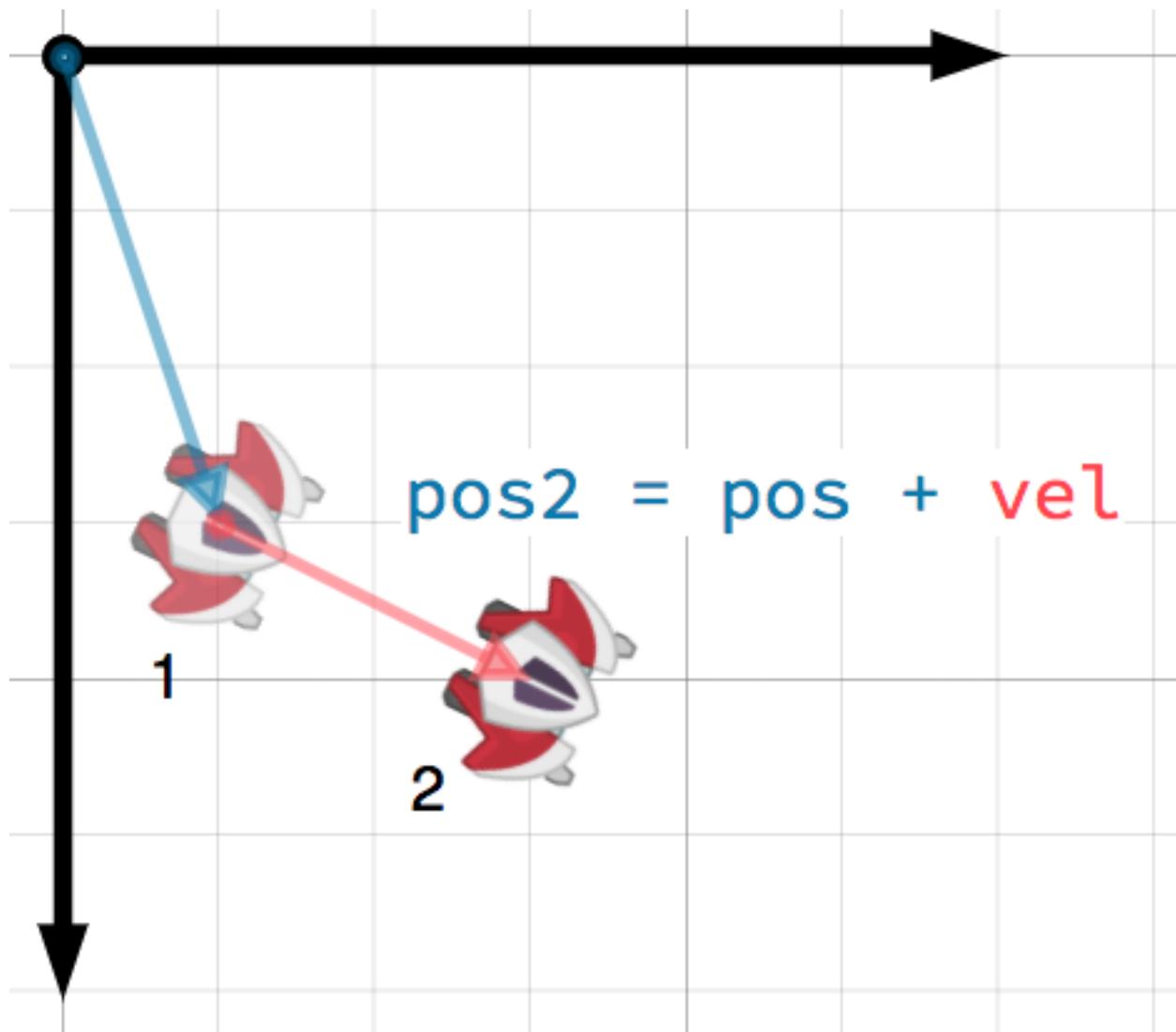
Note: Multiplying a vector by a scalar does not change its direction, only its magnitude. This is how you **scale** a vector.

10.1.4 Practical applications

Let's look at two common uses for vector addition and subtraction.

- Movement

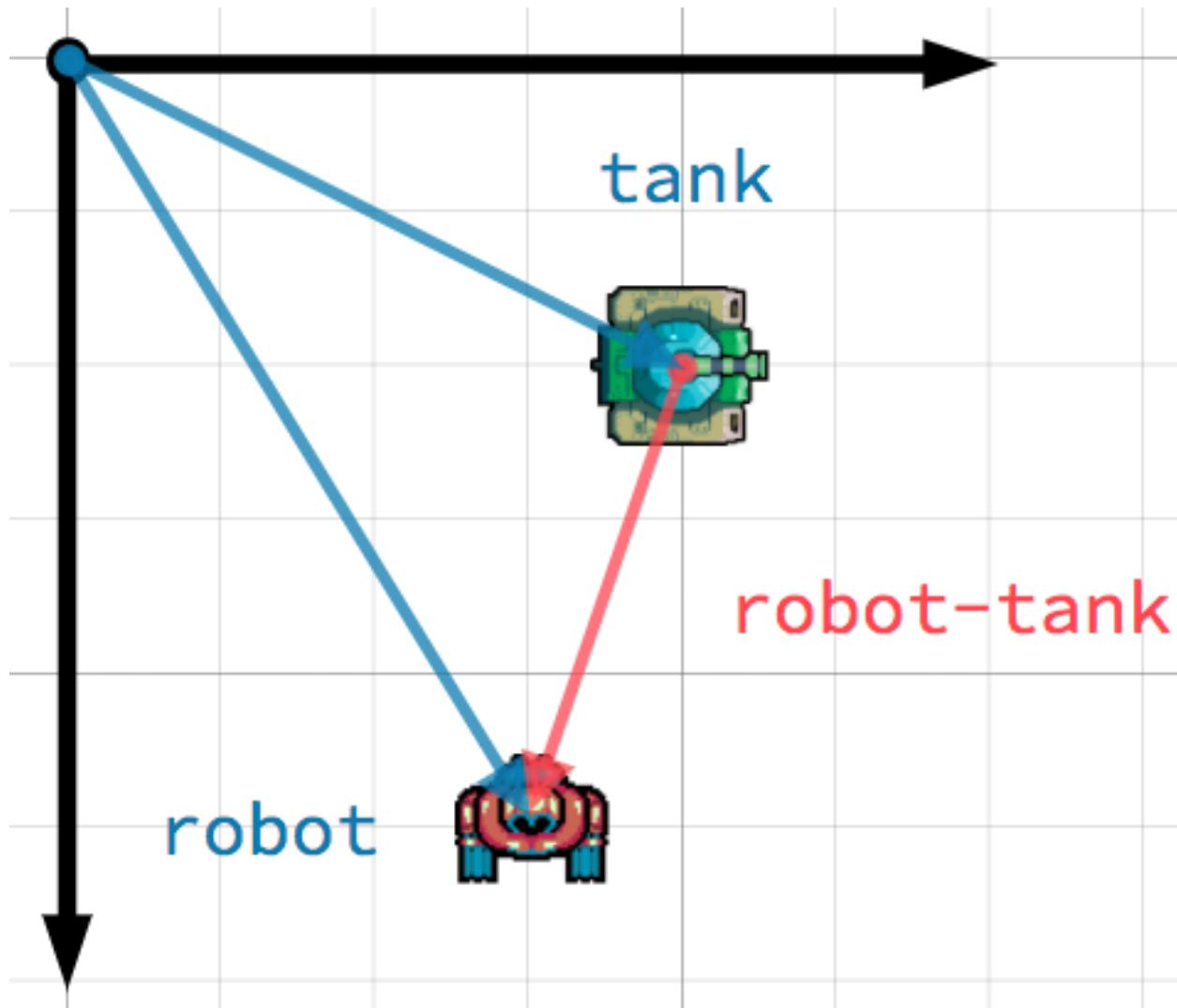
A vector can represent **any** quantity with a magnitude and direction. Typical examples are: position, velocity, acceleration, and force. In this image, the spaceship at step 1 has a position vector of $(1, 3)$ and a velocity vector of $(2, 1)$. The velocity vector represents how far the ship moves each step. We can find the position for step 2 by adding the velocity to the current position.



Tip: Velocity measures the **change** in position per unit of time. The new position is found by adding velocity to the previous position.

- Pointing toward a target

In this scenario, you have a tank that wishes to point its turret at a robot. Subtracting the tank's position from the robot's position gives the vector pointing from the tank to the robot.



Tip: To find a vector pointing from A to B use $B - A$.

10.1.5 Unit vectors

A vector with **magnitude** of 1 is called a **unit vector**. They are also sometimes referred to as **direction vectors** or **normals**. Unit vectors are helpful when you need to keep track of a direction.

Normalization

Normalizing a vector means reducing its length to 1 while preserving its direction. This is done by dividing each of its components by its magnitude:

```
var a = Vector2(2, 4)
var m = sqrt(a.x*a.x + a.y*a.y) # get magnitude "m" using the Pythagorean theorem
a.x /= m
a.y /= m
```

Because this is such a common operation, `Vector2` and `Vector3` provide a method for normalizing:

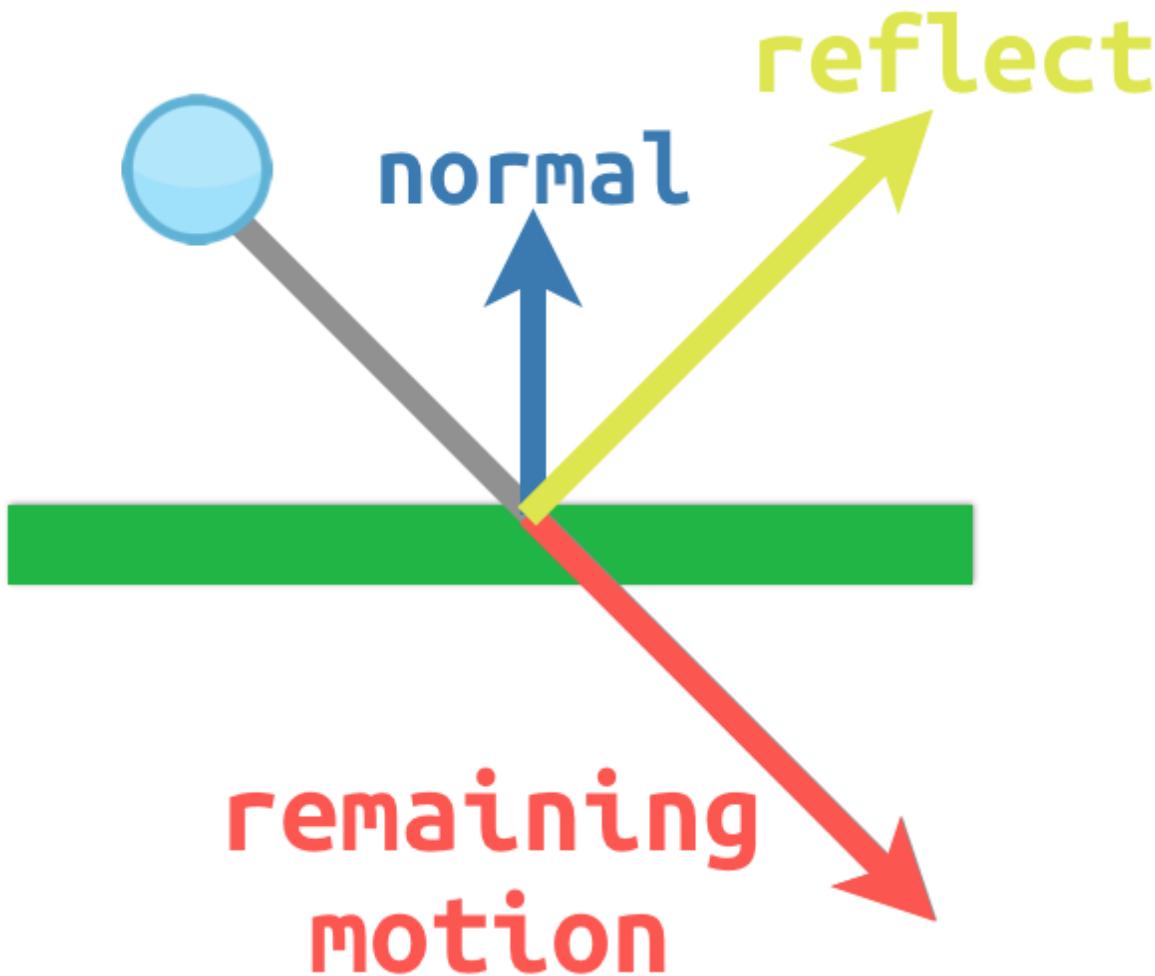
```
a = a.normalized()
```

Warning: Because normalization involves dividing by the vector's length, you cannot normalize a vector of length 0. Attempting to do so will result in an error.

Reflection

A common use of unit vectors is to indicate **normals**. Normal vectors are unit vectors aligned perpendicularly to a surface, defining its direction. They are commonly used for lighting, collisions, and other operations involving surfaces.

For example, imagine we have a moving ball that we want to bounce off a wall or other object:



The surface normal has a value of $(0, -1)$ because this is a horizontal surface. When the ball collides, we take its remaining motion (the amount left over when it hits the surface) and reflect it using the normal. In Godot, the `Vector2` class has a `bounce()` method to handle this. Here is a GDScript example of the diagram above using a `KinematicBody2D`:

```
var collision = move_and_collide(velocity * delta) # object "collision" contains
//information about the collision
if collision:
    var reflect = collision.remainder.bounce(collision.normal)
    velocity = velocity.bounce(collision.normal)
    move_and_collide(reflect)
```

10.1.6 Dot product

The **dot product** is one of the most important concepts in vector math, but is often misunderstood. Dot product is an operation on two vectors that returns a **scalar**. Unlike a vector, which contains both magnitude and direction, a scalar value has only magnitude.

The formula for dot product takes two common forms:

$$A \cdot B = \|A\| \|B\| \cos \Theta$$

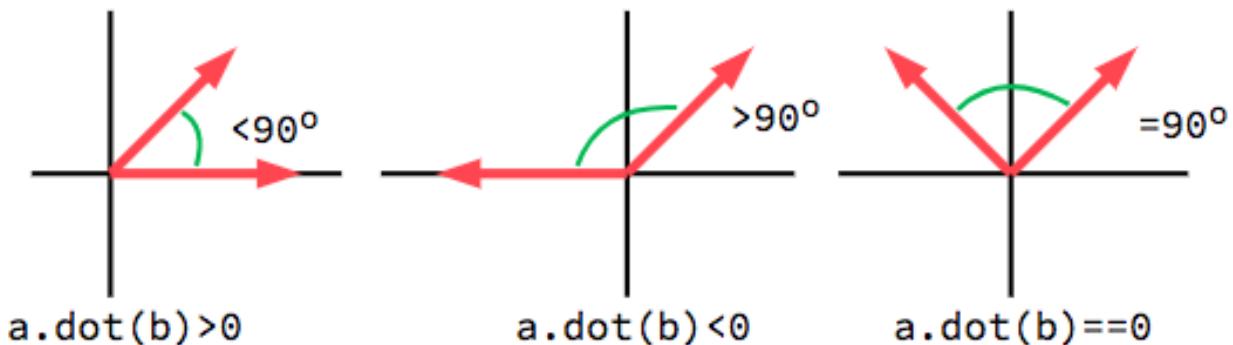
and

$$A \cdot B = A_x B_x + A_y B_y$$

However, in most cases it is easiest to use the built-in method. Note that the order of the two vectors does not matter:

```
var c = a.dot(b)
var d = b.dot(a) # these are equivalent
```

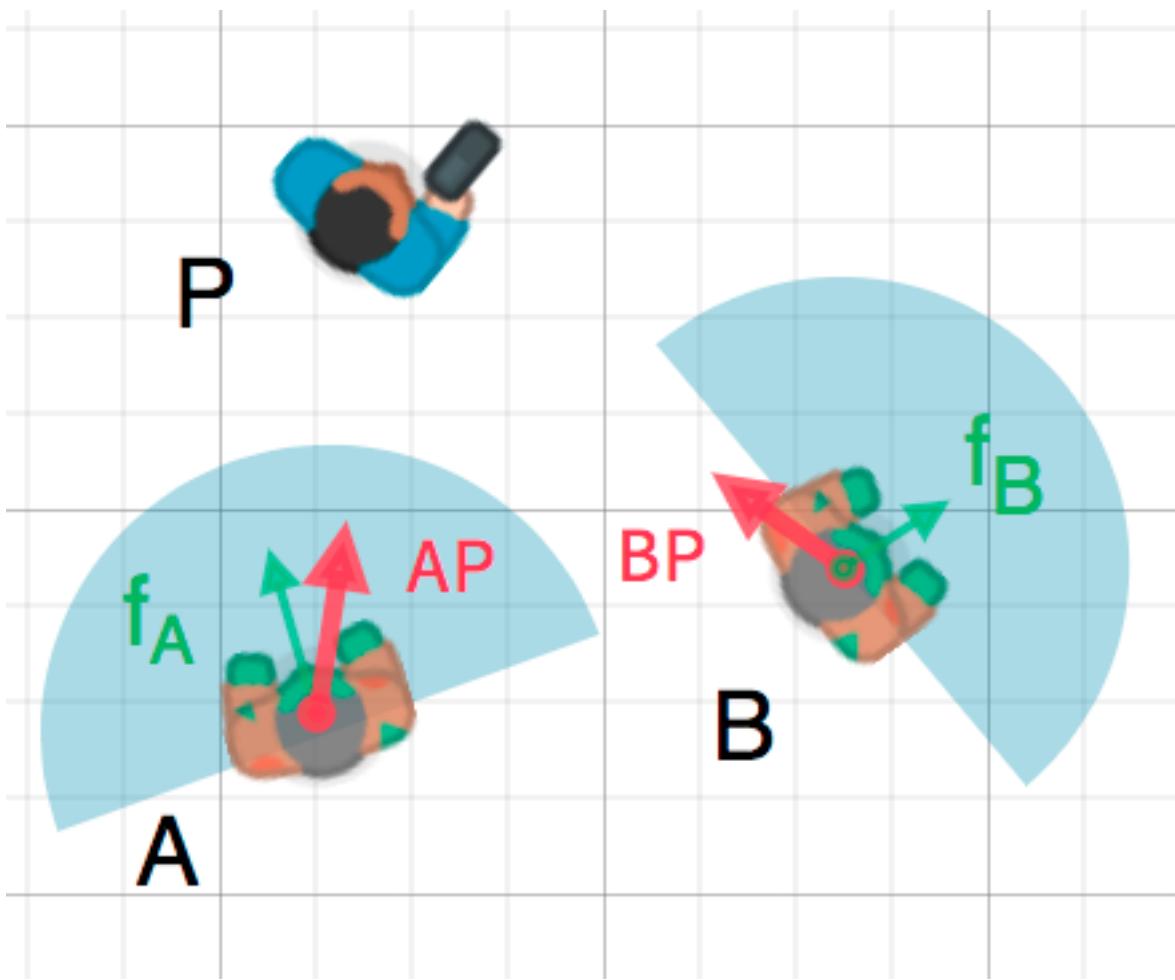
The dot product is most useful when used with unit vectors, making the first formula reduce to just $\cos \theta$. This means we can use the dot product to tell us something about the angle between two vectors:



When using unit vectors, the result will always be between -1 (180°) and 1 (0°).

Facing

We can use this fact to detect whether an object is facing toward another object. In the diagram below, the player P is trying to avoid the zombies A and B. Assuming a zombie's field of view is 180° , can they see the player?



The green arrows f_A and f_B are **unit vectors** representing the zombies' facing directions and the blue semicircle represents its field of view. For zombie A, we find the direction vector AP pointing to the player using $P - A$ and normalize it. If the angle between this vector and the facing vector is less than 90° , then the zombie can see the player.

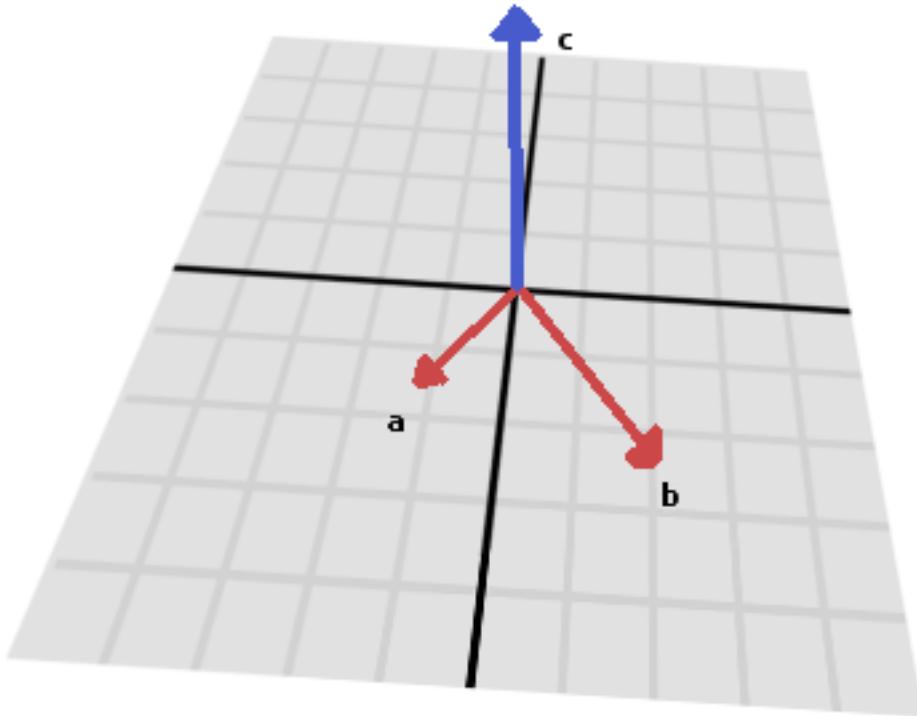
In GDScript it would look like this:

```
var AP = (P - A).normalized()
if AP.dot(fA) > 0:
    print("A sees P!")
```

10.1.7 Cross product

Like the dot product, the **cross product** is an operation on two vectors. However, the result of the cross product is a vector with a direction that is perpendicular to both. Its magnitude depends on their relative angle. If two vectors are parallel, the result of their cross product will be null vector.

$$\|a \times b\| = \|a\| \|b\| |\sin(a, b)|$$



The cross product is calculated like this:

```
var c = Vector3()  
c.x = (a.y * b.z) - (a.z * b.y)  
c.y = (a.z * b.x) - (a.x * b.z)  
c.z = (a.x * b.y) - (a.y * b.x)
```

In GDScript, you can use the built-in method:

```
var c = a.cross(b)
```

Note: In the cross product, order matters. `a.cross(b)` does not give the same result as `b.cross(a)`. The resulting vectors point in **opposite** directions.

Calculating Normals

One common use of cross products is to find the surface normal of a plane or surface in 3D space. If we have the triangle ABC we can use vector subtraction to find two edges AB and AC. Using the cross product, $\text{AB} \times \text{AC}$ produces

a vector perpendicular to both: the surface normal.

Here is a function to calculate a triangle's normal in GDScript:

```
func get_triangle_normal(a, b, c):
    # find the surface normal given 3 vertices
    var side1 = b - a
    var side2 = c - a
    var normal = side1.cross(side2)
    return normal
```

Pointing to a Target

In the dot product section above, we saw how it could be used to find the angle between two vectors. However, in 3D this is not enough information. We also need to know what axis to rotate around. We can find that by calculating the cross product of the current facing direction and the target direction. The resulting perpendicular vector is the axis of rotation.

10.1.8 More Information

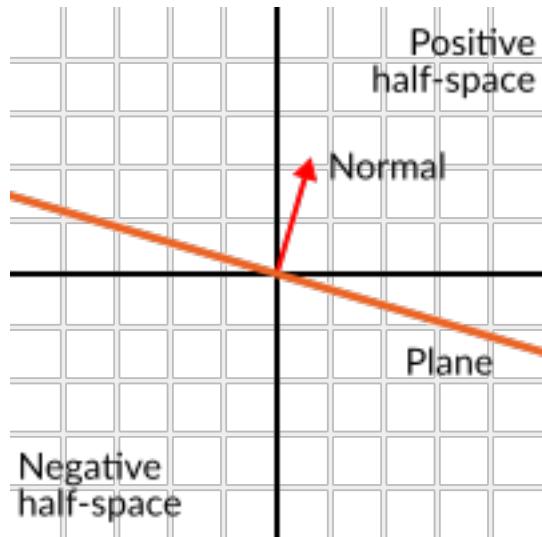
For more information on using vector math in Godot, see the following articles:

- *Advanced Vector Math*
- *Matrices and transforms*

10.2 Advanced Vector Math

10.2.1 Planes

The dot product has another interesting property with unit vectors. Imagine that perpendicular to that vector (and through the origin) passes a plane. Planes divide the entire space into positive (over the plane) and negative (under the plane), and (contrary to popular belief) you can also use their math in 2D:



Unit vectors that are perpendicular to a surface (so, they describe the orientation of the surface) are called **unit normal vectors**. Though, usually they are just abbreviated as *normals. Normals appear in planes, 3D geometry (to determine where each face or vertex is siding), etc. A **normal** is a **unit vector**, but it's called *normal* because of its usage. (Just like we call Origin to (0,0)!).

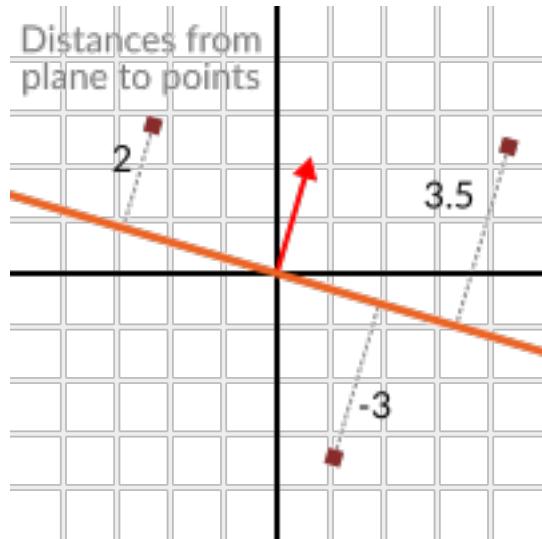
It's as simple as it looks. The plane passes by the origin and the surface of it is perpendicular to the unit vector (or *normal*). The side towards the vector points to is the positive half-space, while the other side is the negative half-space. In 3D this is exactly the same, except that the plane is an infinite surface (imagine an infinite, flat sheet of paper that you can orient and is pinned to the origin) instead of a line.

Distance to plane

Now that it's clear what a plane is, let's go back to the dot product. The dot product between a **unit vector** and any **point in space** (yes, this time we do dot product between vector and position), returns the **distance from the point to the plane**:

```
var distance = normal.dot(point)
```

But not just the absolute distance, if the point is in the negative half space the distance will be negative, too:



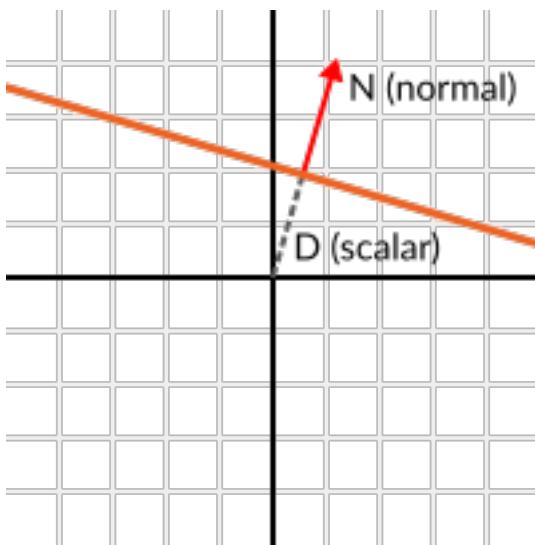
This allows us to tell which side of the plane a point is.

Away from the origin

I know what you are thinking! So far this is nice, but *real* planes are everywhere in space, not only passing through the origin. You want real *plane* action and you want it *now*.

Remember that planes not only split space in two, but they also have *polarity*. This means that it is possible to have perfectly overlapping planes, but their negative and positive half-spaces are swapped.

With this in mind, let's describe a full plane as a **normal** N and a **distance from the origin** scalar D . Thus, our plane is represented by N and D . For example:



For 3D math, Godot provides a *Plane* built-in type that handles this.

Basically, N and D can represent any plane in space, be it for 2D or 3D (depending on the amount of dimensions of N) and the math is the same for both. It's the same as before, but D is the distance from the origin to the plane, travelling in N direction. As an example, imagine you want to reach a point in the plane, you will just do:

```
var point_in_plane = N*D
```

This will stretch (resize) the normal vector and make it touch the plane. This math might seem confusing, but it's actually much simpler than it seems. If we want to tell, again, the distance from the point to the plane, we do the same but adjusting for distance:

```
var distance = N.dot(point) - D
```

The same thing, using a built-in function:

```
var distance = plane.distance_to(point)
```

This will, again, return either a positive or negative distance.

Flipping the polarity of the plane is also very simple, just negate both N and D. This will result in a plane in the same position, but with inverted negative and positive half spaces:

```
N = -N  
D = -D
```

Of course, Godot also implements this operator in *Plane*, so doing:

```
var inverted_plane = -plane
```

Will work as expected.

So, remember, a plane is just that and its main practical use is calculating the distance to it. So, why is it useful to calculate the distance from a point to a plane? It's extremely useful! Let's see some simple examples..

Constructing a plane in 2D

Planes clearly don't come out of nowhere, so they must be built. Constructing them in 2D is easy, this can be done from either a normal (unit vector) and a point, or from two points in space.

In the case of a normal and a point, most of the work is done, as the normal is already computed, so just calculate D from the dot product of the normal and the point.

```
var N = normal  
var D = normal.dot(point)
```

For two points in space, there are actually two planes that pass through them, sharing the same space but with normal pointing to the opposite directions. To compute the normal from the two points, the direction vector must be obtained first, and then it needs to be rotated 90° degrees to either side:

```
# calculate vector from a to b  
var dvec = (point_b - point_a).normalized()  
# rotate 90 degrees  
var normal = Vector2(dvec.y, -dvec.x)  
# or alternatively  
# var normal = Vector2(-dvec.y, dvec.x)  
# depending the desired side of the normal
```

The rest is the same as the previous example, either point_a or point_b will work since they are in the same plane:

```
var N = normal  
var D = normal.dot(point_a)  
# this works the same  
# var D = normal.dot(point_b)
```

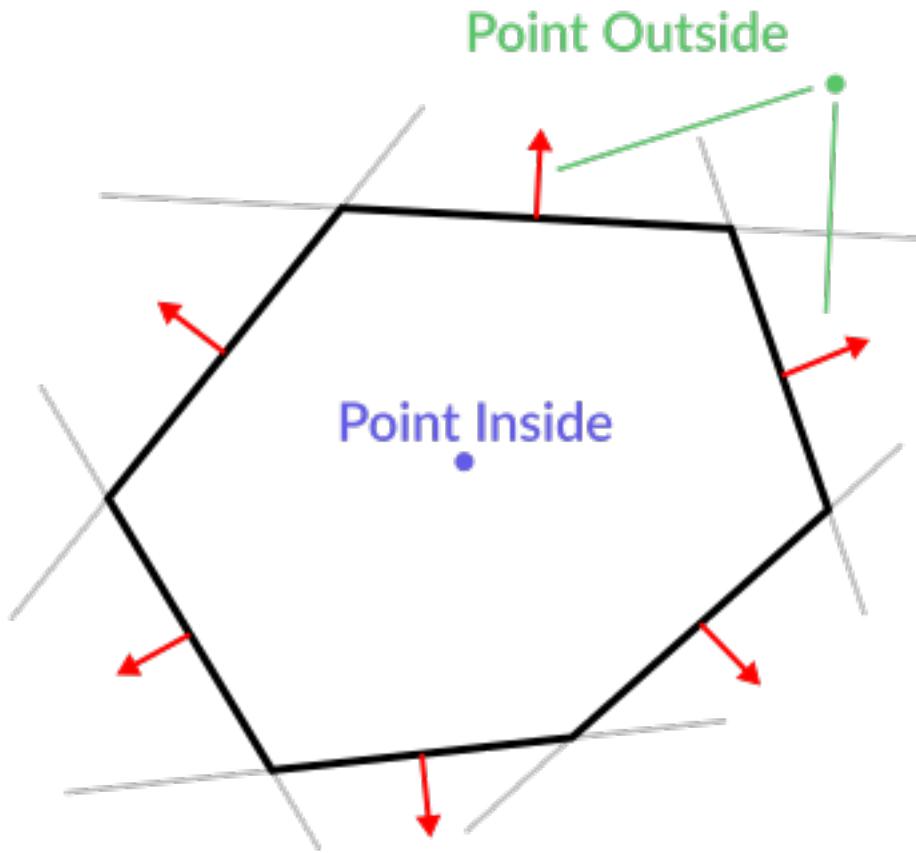
Doing the same in 3D is a little more complex and will be explained further down.

Some examples of planes

Here is a simple example of what planes are useful for. Imagine you have a [convex](#) polygon. For example, a rectangle, a trapezoid, a triangle, or just any polygon where no faces bend inwards.

For every segment of the polygon, we compute the plane that passes by that segment. Once we have the list of planes, we can do neat things, for example checking if a point is inside the polygon.

We go through all planes, if we can find a plane where the distance to the point is positive, then the point is outside the polygon. If we can't, then the point is inside.

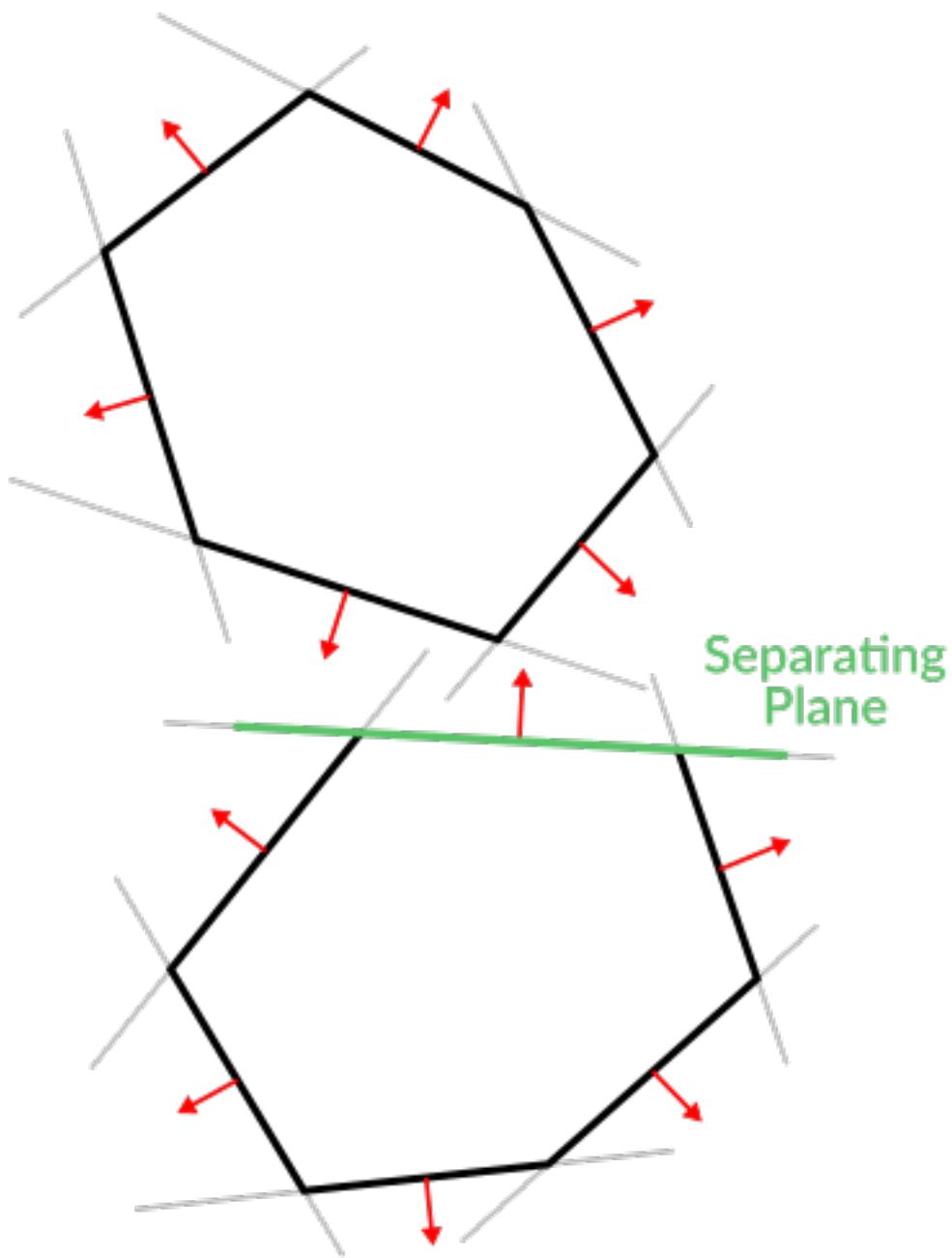


Code should be something like this:

```
var inside = true
for p in planes:
    # check if distance to plane is positive
    if (N.dot(point) - D > 0):
        inside = false
        break # with one that fails, it's enough
```

Pretty cool, huh? But this gets much better! With a little more effort, similar logic will let us know when two convex polygons are overlapping too. This is called the Separating Axis Theorem (or SAT) and most physics engines use this to detect collision.

The idea is really simple! With a point, just checking if a plane returns a positive distance is enough to tell if the point is outside. With another polygon, we must find a plane where *all the *other* polygon points* return a positive distance to it. This check is performed with the planes of A against the points of B, and then with the planes of B against the points of A:



Code should be something like this:

```
var overlapping = true

for p in planes_of_A:
    var all_out = true
    for v in points_of_B:
        if (p.distance_to(v) < 0):
            all_out = false
            break

    if (all_out):
```

```

# a separating plane was found
# do not continue testing
overlapping = false
break

if (overlapping):
    # only do this check if no separating plane
    # was found in planes of A
    for p in planes_of_B:
        var all_out = true
        for v in points_of_A:
            if (p.distance_to(v) < 0):
                all_out = false
                break

        if (all_out):
            overlapping = false
            break

if (overlapping):
    print("Polygons Collided!")

```

As you can see, planes are quite useful, and this is the tip of the iceberg. You might be wondering what happens with non convex polygons. This is usually just handled by splitting the concave polygon into smaller convex polygons, or using a technique such as BSP (which is not used much nowadays).

10.2.2 Collision detection in 3D

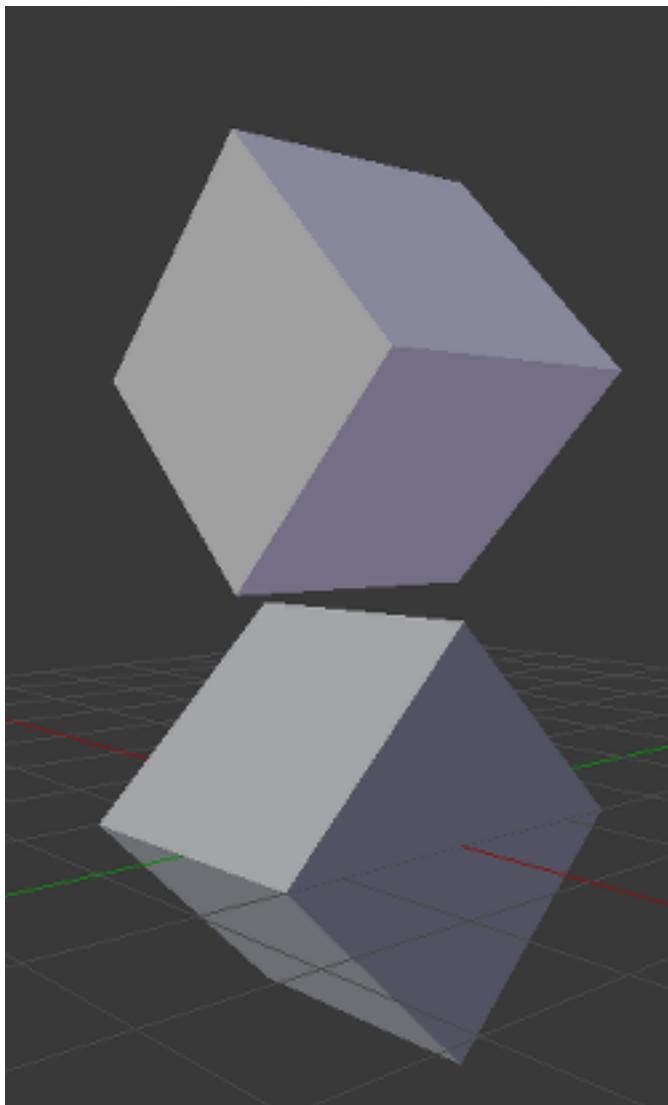
This is another bonus bit, a reward for being patient and keeping up with this long tutorial. Here is another piece of wisdom. This might not be something with a direct use case (Godot already does collision detection pretty well) but It's a really cool algorithm to understand anyway, because it's used by almost all physics engines and collision detection libraries :)

Remember that converting a convex shape in 2D to an array of 2D planes was useful for collision detection? You could detect if a point was inside any convex shape, or if two 2D convex shapes were overlapping.

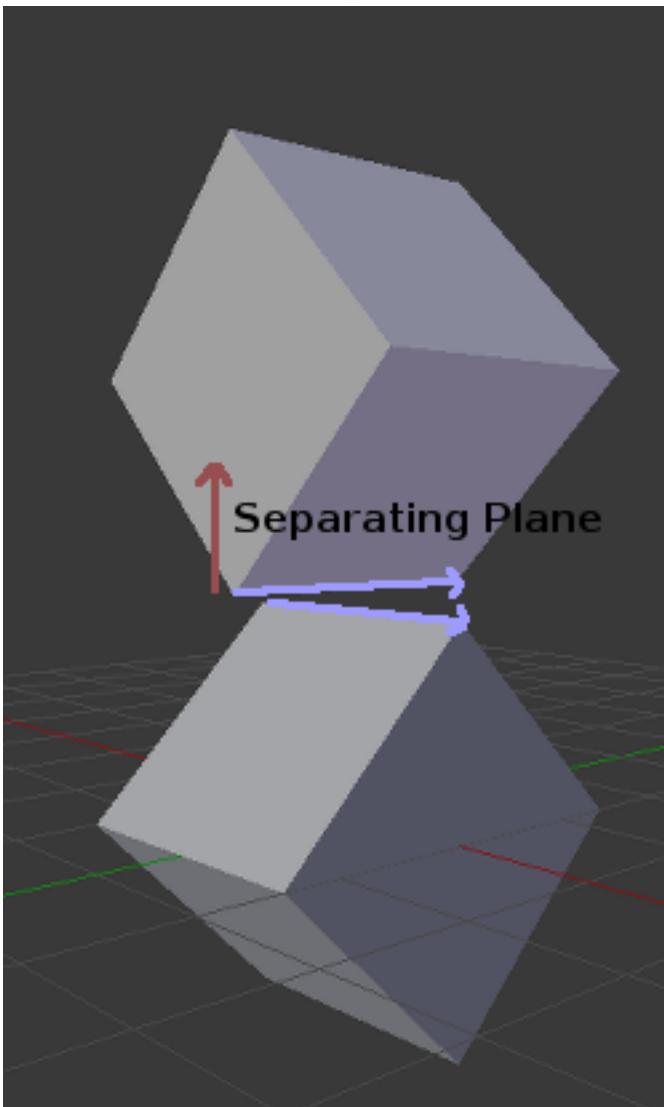
Well, this works in 3D too, if two 3D polyhedral shapes are colliding, you won't be able to find a separating plane. If a separating plane is found, then the shapes are definitely not colliding.

To refresh a bit a separating plane means that all vertices of polygon A are in one side of the plane, and all vertices of polygon B are in the other side. This plane is always one of the face-planes of either polygon A or polygon B.

In 3D though, there is a problem to this approach, because it is possible that, in some cases a separating plane can't be found. This is an example of such situation:



To avoid it, some extra planes need to be tested as separators, these planes are the cross product between the edges of polygon A and the edges of polygon B



So the final algorithm is something like:

```
var overlapping = true

for p in planes_of_A:
    var all_out = true
    for v in points_of_B:
        if (p.distance_to(v) < 0):
            all_out = false
            break

    if (all_out):
        # a separating plane was found
        # do not continue testing
        overlapping = false
        break

if (overlapping):
    # only do this check if no separating plane
    # was found in planes of A
```

```

for p in planes_of_B:
    var all_out = true
    for v in points_of_A:
        if (p.distance_to(v) < 0):
            all_out = false
            break

    if (all_out):
        overlapping = false
        break

if (overlapping):
    for ea in edges_of_A:
        for eb in edges_of_B:
            var n = ea.cross(eb)
            if (n.length() == 0):
                continue

            var max_A = -1e20 # tiny number
            var min_A = 1e20 # huge number

            # we are using the dot product directly
            # so we can map a maximum and minimum range
            # for each polygon, then check if they
            # overlap.

            for v in points_of_A:
                var d = n.dot(v)
                if (d > max_A):
                    max_A = d
                if (d < min_A):
                    min_A = d

            var max_B = -1e20 # tiny number
            var min_B = 1e20 # huge number

            for v in points_of_B:
                var d = n.dot(v)
                if (d > max_B):
                    max_B = d
                if (d < min_B):
                    min_B = d

            if (min_A > max_B or min_B > max_A):
                # not overlapping!
                overlapping = false
                break

            if (not overlapping):
                break

if (overlapping):
    print("Polygons collided!")

```

10.3 Matrices and transforms

10.3.1 Introduction

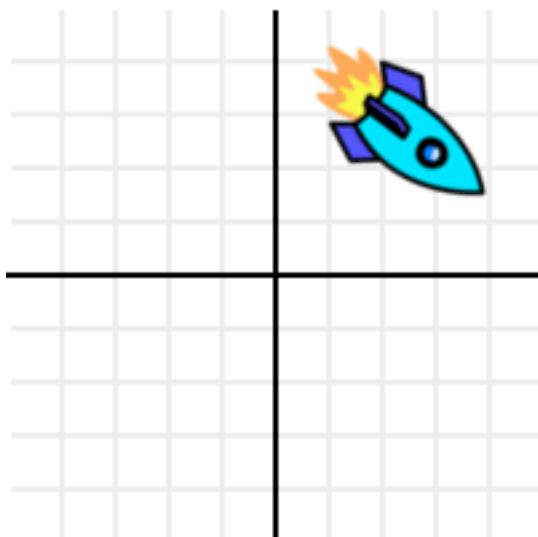
Before reading this tutorial, it is advised to read the previous one about *Vector math* as this one is a direct continuation.

This tutorial will be about *transformations* and will cover a little about matrices (but not in-depth).

Transformations are most of the time applied as translation, rotation and scale so they will be considered as priority here.

10.3.2 Oriented coordinate system (OCS)

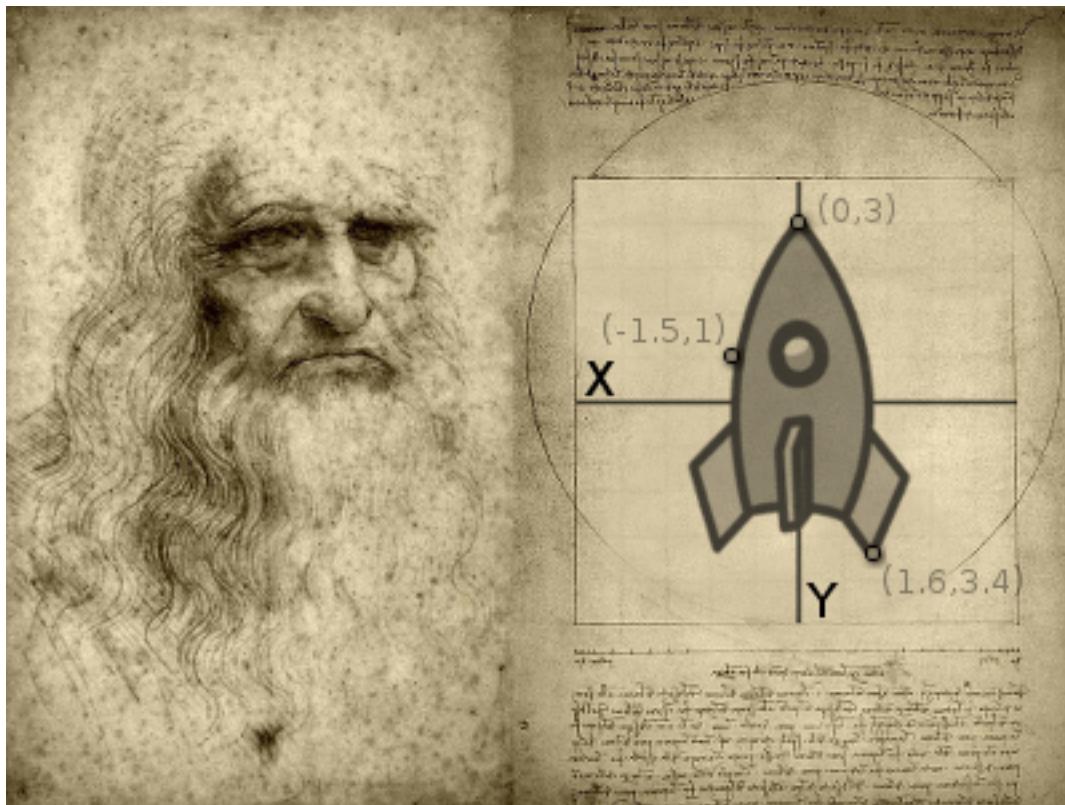
Imagine we have a spaceship somewhere in space. In Godot this is easy, just move the ship somewhere and rotate it:



Ok, so in 2D this looks simple, a position and an angle for a rotation. But remember, we are grown ups here and don't use angles (plus, angles are not really even that useful when working in 3D).

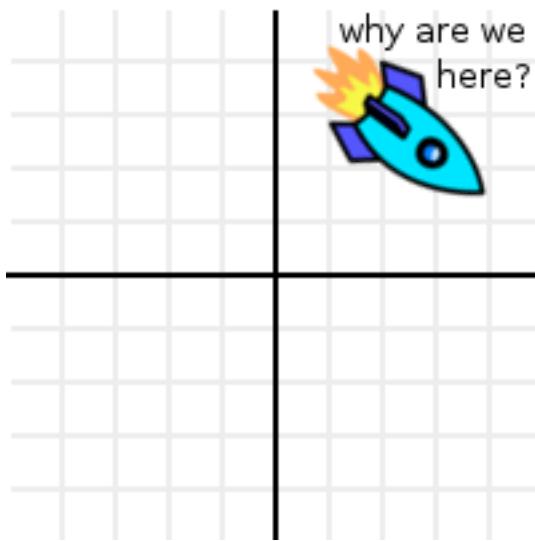
We should realize that at some point, someone *designed* this spaceship. Be it for 2D in a drawing such as Paint.net, Gimp, Photoshop, etc. or in 3D through a 3D DCC tool such as Blender, Max, Maya, etc.

When it was designed, it was not rotated. It was designed in its own *coordinate system*.



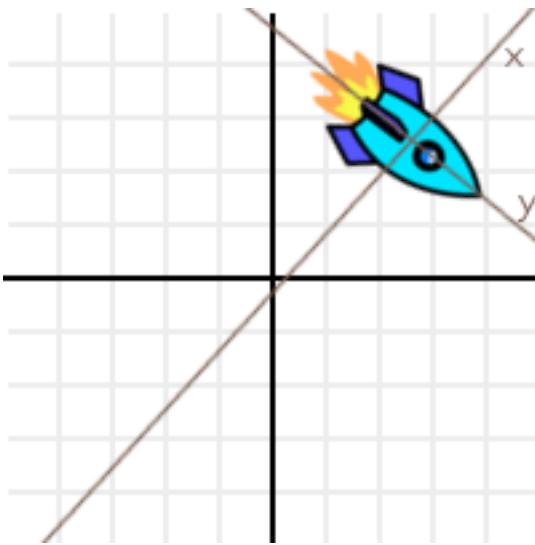
This means that the tip of the ship has a coordinate, the fin has another, etc. Be it in pixels (2D) or vertices (3D).

So, let's recall again that the ship was somewhere in space:

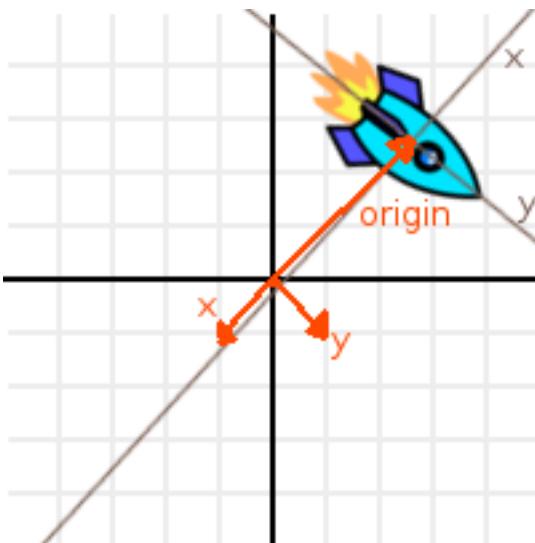


How did it get there? What moved it and rotated it from the place it was designed to its current position? The answer is... a **transform**, the ship was *transformed* from their original position to the new one. This allows the ship to be displayed where it is.

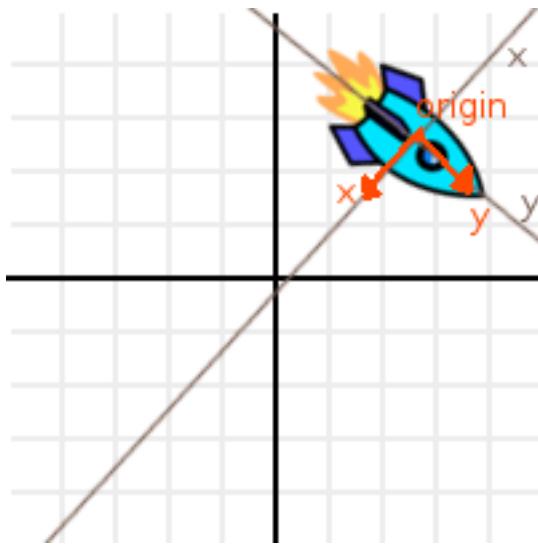
But transform is too generic of a term to describe this process. To solve this puzzle, we will superimpose the ship's original design position at their current position:



So, we can see that the “design space” has been transformed too. How can we best represent this transformation? Let’s use 3 vectors for this (in 2D), a unit vector pointing towards X positive, a unit vector pointing towards Y positive and a translation.

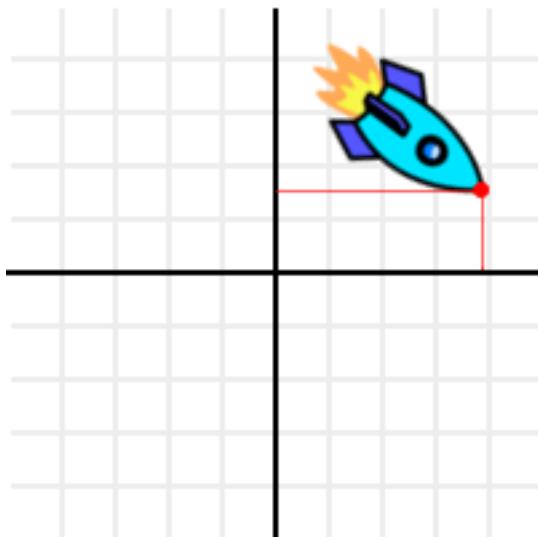


Let’s call the 3 vectors “X”, “Y” and “Origin”, and let’s also superimpose them over the ship so it makes more sense:



Ok, this is nicer, but it still does not make sense. What do X,Y and Origin have to do with how the ship got there?

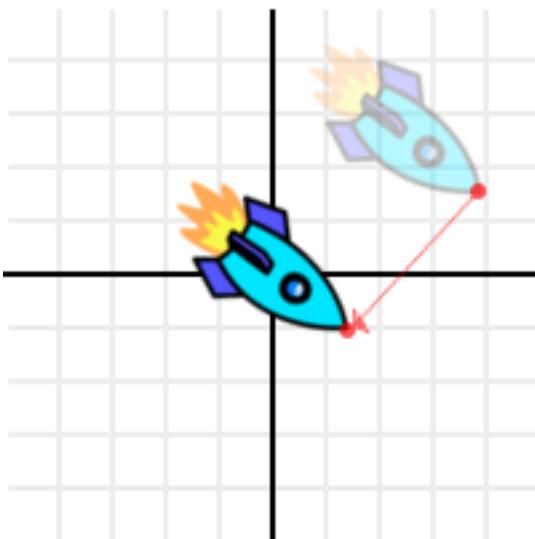
Well, let's take the point from top tip of the ship as reference:



And let's apply the following operation to it (and to all the points in the ship too, but we'll track the top tip as our reference point):

```
var new_pos = pos - origin
```

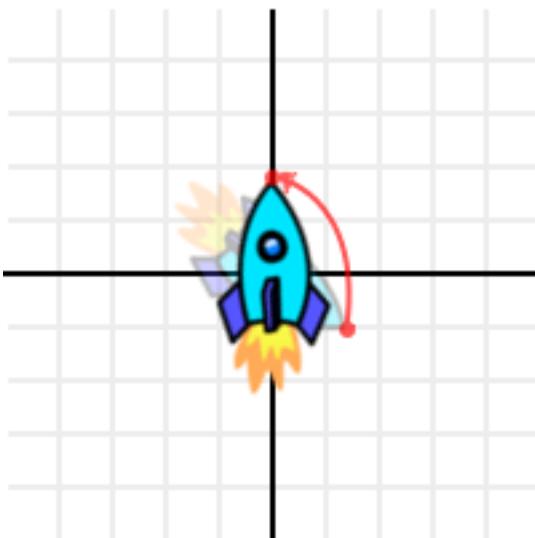
Doing this to the selected point will move it back to the center:



This was expected, but then let's do something more interesting. Use the dot product of X and the point, and add it to the dot product of Y and the point:

```
var final_pos = Vector2(x.dot(new_pos), y.dot(new_pos))
```

Then what we have is.. wait a minute, it's the ship in its design position!



How did this black magic happen? The ship was lost in space, and now it's back home!

It might seem strange, but it does have plenty of logic. Remember, as we have seen in the *Vector math*, what happened is that the distance to X axis, and the distance to Y axis were computed. Calculating distance in a direction or plane was one of the uses for the dot product. This was enough to obtain back the design coordinates for every point in the ship.

So, what we have been working with so far (with X, Y and Origin) is an *Oriented Coordinate System*. X and Y are the **Basis**, and *Origin* is the offset.

10.3.3 Basis

We know what the Origin is. It's where the 0,0 (origin) of the design coordinate system ended up after being transformed to a new position. This is why it's called *Origin*, But in practice, it's just an offset to the new position.

The Basis is more interesting. The basis is the direction of X and Y in the OCS from the new, transformed location. It tells what has changed, in either 2D or 3D. The Origin (offset) and Basis (direction) communicate “Hey, the original X and Y axes of your design are *right here*, pointing towards *these directions*.”

So, let's change the representation of the basis. Instead of 2 vectors, let's use a *matrix*.

$$M = \begin{Bmatrix} X_x & X_y \\ Y_x & Y_y \end{Bmatrix}$$

The vectors are up there in the matrix, horizontally. The next problem now is that.. what is this matrix thing? Well, we'll assume you've never heard of a matrix.

10.3.4 Transforms in Godot

This tutorial will not explain matrix math (and their operations) in depth, only its practical use. There is plenty of material for that, which should be a lot simpler to understand after completing this tutorial. We'll just explain how to use transforms.

10.3.5 Transform2D

Transform2D is a 3x2 matrix. It has 3 Vector2 elements and it's used for 2D. The “X” axis is the element 0, “Y” axis is the element 1 and “Origin” is element 2. It's not divided in basis/origin for convenience, due to its simplicity.

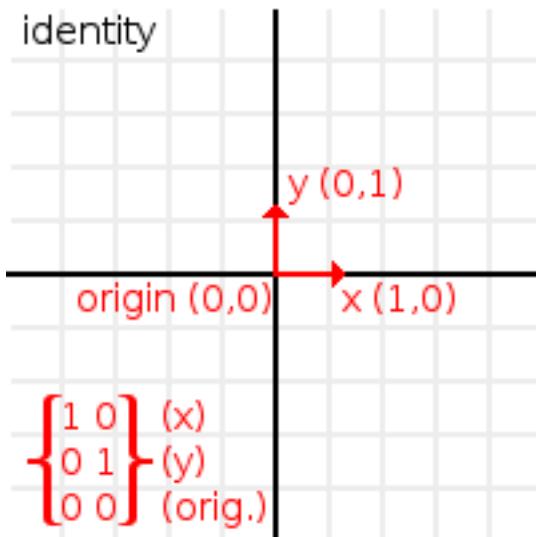
```
var m = Transform2D()
var x = m[0] # 'X'
var y = m[1] # 'Y'
var o = m[2] # 'Origin'
```

Most operations will be explained with this datatype (*Transform2D*), but the same logic applies to 3D.

10.3.6 Identity

By default, *Transform2D* is created as an “identity” matrix. This means:

- ‘X’ Points right: *Vector2(1,0)*
- ‘Y’ Points up (or down in pixels): *Vector2(0,1)*
- ‘Origin’ is the origin *Vector2(0,0)*



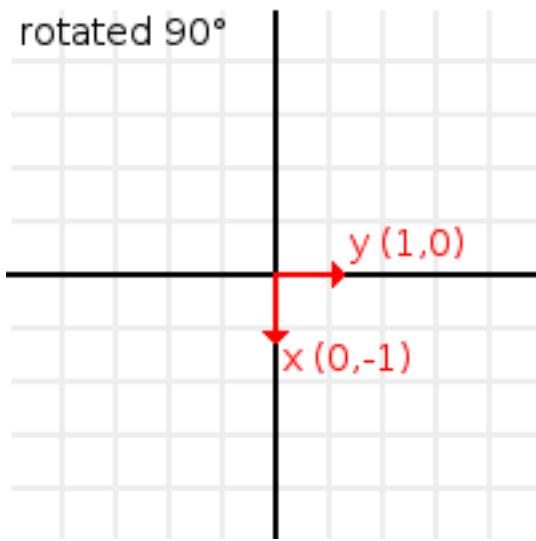
It's easy to guess that an *identity* matrix is just a matrix that aligns the transform to its parent coordinate system. It's an *OCS* that hasn't been translated, rotated or scaled. All transform types in Godot are created with *identity*.

10.3.7 Operations

10.3.8 Rotation

Rotating Transform2D is done by using the “rotated” function:

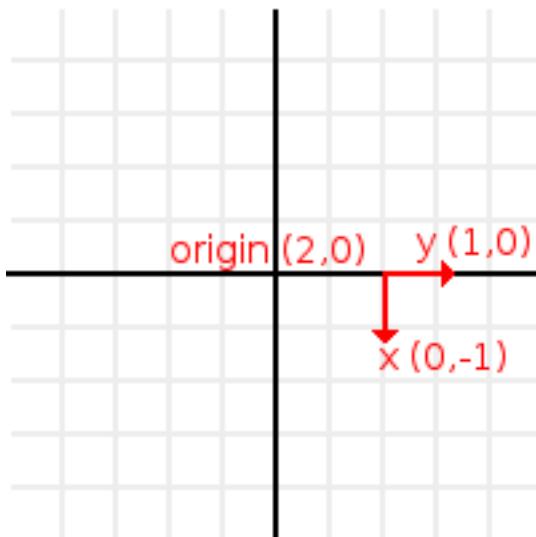
```
var m = Transform2D()
m = m.rotated(PI/2) # rotate 90°
```



10.3.9 Translation

There are two ways to translate a Transform2D, the first one is just moving the origin:

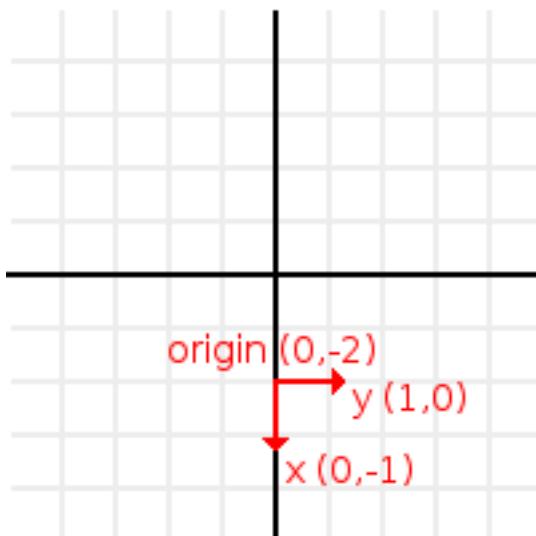
```
# Move 2 units to the right
var m = Transform2D()
m = m.rotated(PI/2) # rotate 90°
m[2] += Vector2(2, 0)
```



This will always work in global coordinates.

If instead, translation is desired in *local* coordinates of the matrix (towards where the *basis* is oriented), there is the `Transform2D.translated()` method:

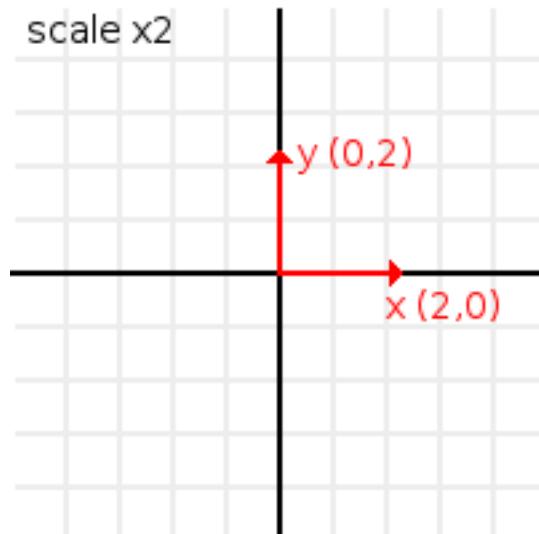
```
# Move 2 units towards where the basis is oriented
var m = Transform2D()
m = m.rotated(PI/2) # rotate 90°
m = m.translated( Vector2(2, 0) )
```



10.3.10 Scale

A matrix can be scaled too. Scaling will multiply the basis vectors by a vector (X vector by x component of the scale, Y vector by y component of the scale). It will leave the origin alone:

```
# Make the basis twice its size.  
var m = Transform2D()  
m = m.scaled( Vector2(2,2) )
```



These kind of operations in matrices are accumulative. It means every one starts relative to the previous one. For those who have been living on this planet long enough, a good reference of how transform works is this:



2BK 50
2CS

A matrix is used similarly to a turtle. The turtle most likely had a matrix inside (and you are likely learning this many years *after* discovering Santa is not real).

10.3.11 Transform

Transform is the act of switching between coordinate systems. To convert a position (either 2D or 3D) from “designer” coordinate system to the OCS, the “xform” method is used.

```
var new_pos = m.xform(pos)
```

And only for basis (no translation):

```
var new_pos = m.basis_xform(pos)
```

Post - multiplying is also valid:

```
var new_pos = m * pos
```

10.3.12 Inverse transform

To do the opposite operation (what we did up there with the rocket), the “xform_inv” method is used:

```
var new_pos = m.xform_inv(pos)
```

Only for Basis:

```
var new_pos = m.basis_xform_inv(pos)
```

Or pre-multiplication:

```
var new_pos = pos * m
```

10.3.13 Orthonormal matrices

However, if the matrix has been scaled (vectors are not unit length), or the basis vectors are not orthogonal (90°), the inverse transform will not work.

In other words, inverse transform is only valid in *orthonormal* matrices. For this, these cases an affine inverse must be computed.

The transform, or inverse transform of an identity matrix will return the position unchanged:

```
# Does nothing, pos is unchanged
pos = Transform2D().xform(pos)
```

10.3.14 Affine inverse

The affine inverse is a matrix that does the inverse operation of another matrix, no matter if the matrix has scale or the axis vectors are not orthogonal. The affine inverse is calculated with the `affine_inverse()` method:

```
var mi = m.affine_inverse()
var pos = m.xform(pos)
pos = mi.xform(pos)
# pos is unchanged
```

If the matrix is orthonormal, then:

```
# if m is orthonormal, then
pos = mi.xform(pos)
# is the same is
pos = m.xform_inv(pos)
```

10.3.15 Matrix multiplication

Matrices can be multiplied. Multiplication of two matrices “chains” (concatenates) their transforms.

However, as per convention, multiplication takes place in reverse order.

Example:

```
var m = more_transforms * some_transforms
```

To make it a little clearer, this:

```
pos = transform1.xform(pos)
pos = transform2.xform(pos)
```

Is the same as:

```
# note the inverse order
pos = (transform2 * transform1).xform(pos)
```

However, this is not the same:

```
# yields a different results
pos = (transform1 * transform2).xform(pos)
```

Because in matrix math, $A * B$ is not the same as $B * A$.

10.3.16 Multiplication by inverse

Multiplying a matrix by its inverse, results in identity:

```
# No matter what A is, B will be identity
B = A.affine_inverse() * A
```

10.3.17 Multiplication by identity

Multiplying a matrix by identity, will result in the unchanged matrix:

```
# B will be equal to A
B = A * Transform2D()
```

10.3.18 Matrix tips

When using a transform hierarchy, remember that matrix multiplication is reversed! To obtain the global transform for a hierarchy, do:

```
var global_xform = parent_matrix * child_matrix
```

For 3 levels:

```
var global_xform = gradparent_matrix * parent_matrix * child_matrix
```

To make a matrix relative to the parent, use the affine inverse (or regular inverse for orthonormal matrices).

```
# transform B from a global matrix to one local to A
var B_local_to_A = A.affine_inverse() * B
```

Revert it just like the example above:

```
# transform back local B to global B
var B = A * B_local_to_A
```

OK, hopefully this should be enough! Let's complete the tutorial by moving to 3D matrices.

10.3.19 Matrices & transforms in 3D

As mentioned before, for 3D, we deal with 3 *Vector3* vectors for the rotation matrix, and an extra one for the origin.

10.3.20 Matrix3

Godot has a special type for a 3x3 matrix, named *Matrix3*. It can be used to represent a 3D rotation and scale. Sub vectors can be accessed as:

```
var m = Matrix3()
var x = m[0] # Vector3
var y = m[1] # Vector3
var z = m[2] # Vector3
```

Or, alternatively as:

```
var m = Matrix3()
var x = m.x # Vector3
var y = m.y # Vector3
var z = m.z # Vector3
```

Matrix3 is also initialized to Identity by default:

$$\left\{ \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \right\} \begin{matrix} (X) \\ (Y) \\ (Z) \end{matrix}$$

10.3.21 Rotation in 3D

Rotation in 3D is more complex than in 2D (translation and scale are the same), because rotation is an implicit 2D operation. To rotate in 3D, an *axis*, must be picked. Rotation, then, happens around this axis.

The axis for the rotation must be a *normal vector*. As in, a vector that can point to any direction, but length must be one (1.0).

```
#rotate in Y axis
var m3 = Matrix3()
m3 = m3.rotated( Vector3(0,1,0), PI/2 )
```

10.3.22 Transform

To add the final component to the mix, Godot provides the *Transform* type. Transform has two members:

- *basis* (of type *Basis*)
- *origin* (of type *Vector3*)

Any 3D transform can be represented with Transform, and the separation of basis and origin makes it easier to work translation and rotation separately.

An example:

```
var t = Transform()
pos = t.xform(pos) # transform 3D position
pos = t.basis.xform(pos) # (only rotate)
pos = t.origin + pos # (only translate)
```


CHAPTER 11

Animation

11.1 Introduction to the 2D animation features

11.1.1 Overview

The *AnimationPlayer* node allows you to create anything from simple to complex animations.

In this guide you learn to:

- Work with the Animation Panel
- Animate any property of any node
- Create a simple animation
- Call functions with the powerful Call Function Tracks

In Godot you can animate anything you find in the Inspector. Animations are changes to properties over time. This means you can animate anything visual in nature: sprites, UI elements, particles, the visibility, and color of textures. But not only. You can also control gameplay related values and call any function.

11.1.2 Create an *AnimationPlayer* node

To use the animation tools we first have to create an *AnimationPlayer* node.

The *AnimationPlayer* node type is the data container for your animations. One *AnimationPlayer* node can hold multiple animations, that can automatically transition to one another.

Click on the *AnimationPlayer* node in the Node tab to open the Animation Panel at the bottom of the viewport.

It consists of five parts:

- Animation controls (i.e. add, load, save, and delete animations)
- The tracks listing
- The timeline with keyframes

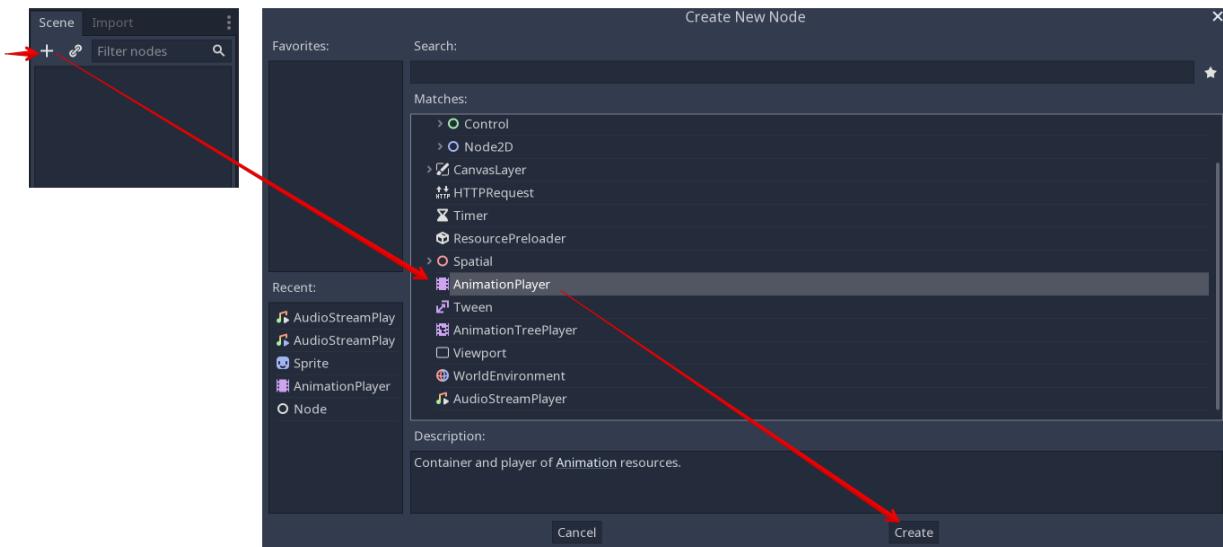


Fig. 11.1: The AnimationPlayer node



Fig. 11.2: The animation panel position

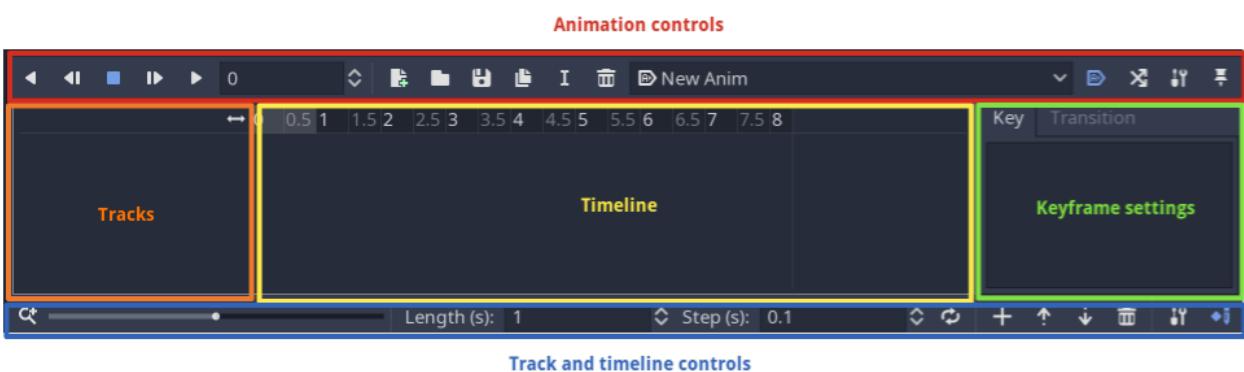


Fig. 11.3: The animation panel

- The keyframe editor (when enabled)
- The timeline and track controls, where you can zoom the timeline and edit tracks for example.

See the *animation panel reference* below for details.

11.1.3 Computer animation relies on keyframes

A keyframe defines the value of a property at a certain point in time.

White and blue diamond shapes represent keyframes in the timeline.

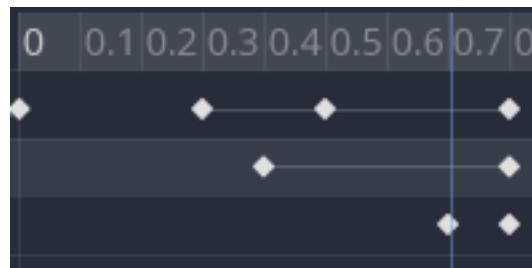


Fig. 11.4: Keyframes in Godot

The engine interpolates values between keyframes, resulting in a gradual change in values over time.

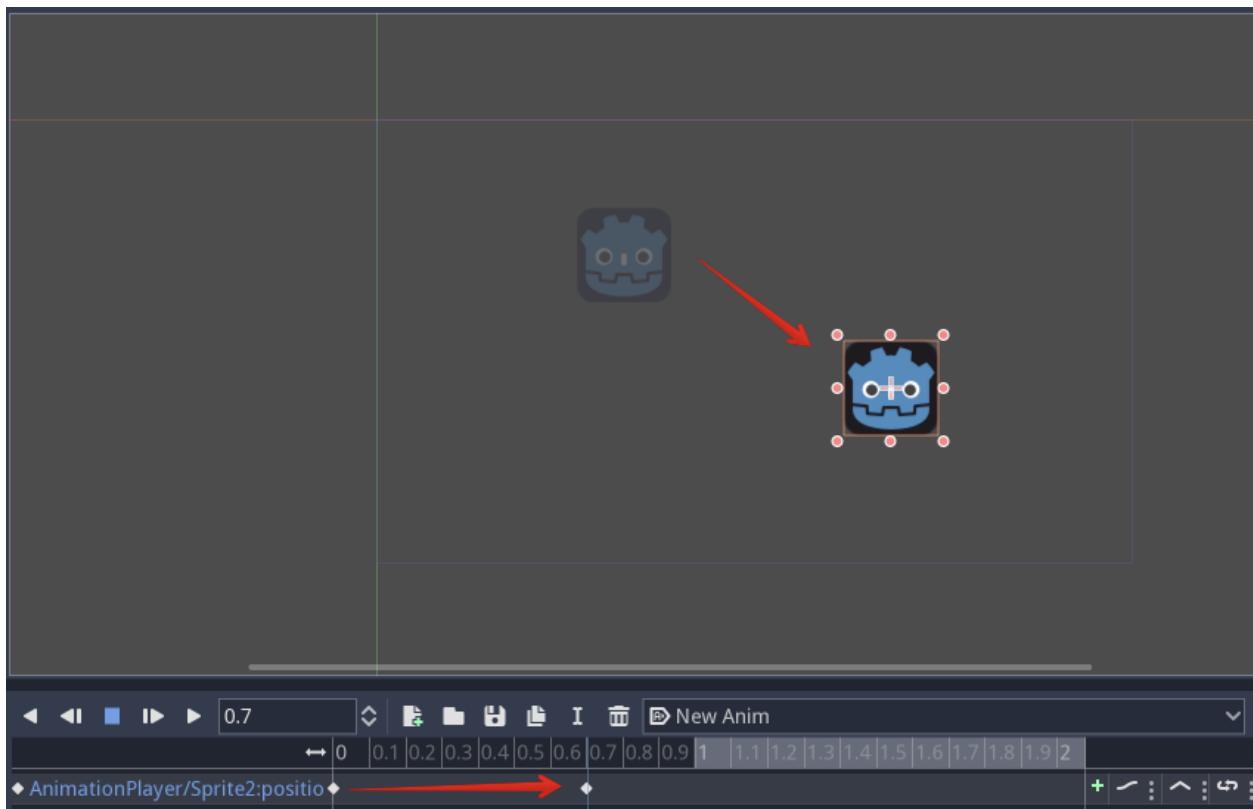


Fig. 11.5: Two keyframes are all it takes to obtain a smooth motion

The timeline lets you insert keyframes and change their timing. It also defines how long the animation is.

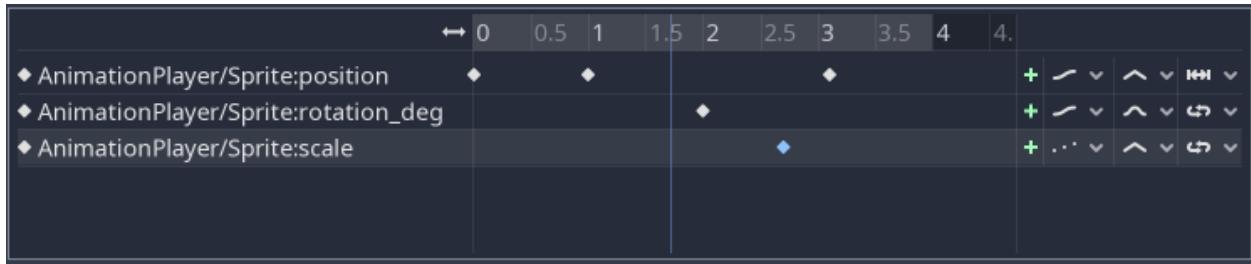


Fig. 11.6: The timeline in the animation panel

Each line of the Animation Panel is an animation track. Normal and Transform tracks reference node properties. Their name or id is a path to the node and the affected property.

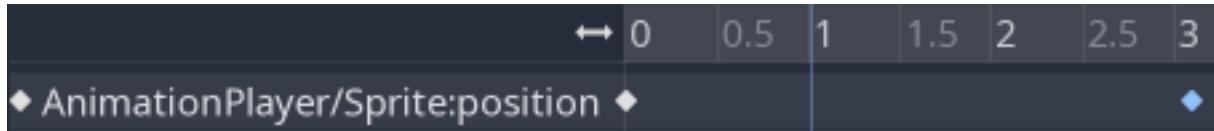


Fig. 11.7: Example of Normal animation tracks

Tip: If you animate the wrong property, you can edit a track's path anytime. Double click on it and type the new path.

Play the animation using the “Play from beginning” button (Default shortcut: Shift+D) to see the changes instantly.

11.1.4 Tutorial: Creating a simple animation

Scene setup

For this tutorial, we'll going to create an AnimationPlayer node and a sprite node as an AnimationPlayer node child.

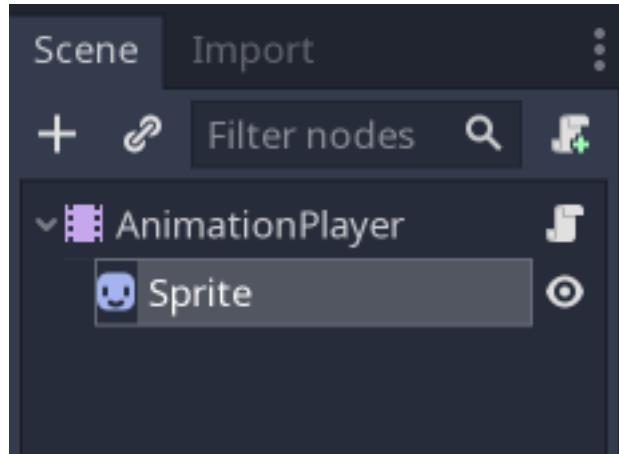


Fig. 11.8: Our scene setup

The sprite holds an image texture and we animate that sprite to move between two points on the screen. As a starting point, move the sprite to a left position on the screen.

Tip: Adding animated nodes as children to the AnimationPlayer node is not required, but it is a nice way of distinguishing animated nodes from non-animated nodes in the Scene Tree.

Select the AnimationPlayer node and click on “Add animation” () in the animation tab to add a new animation. Enter a name for the animation in the dialog box.

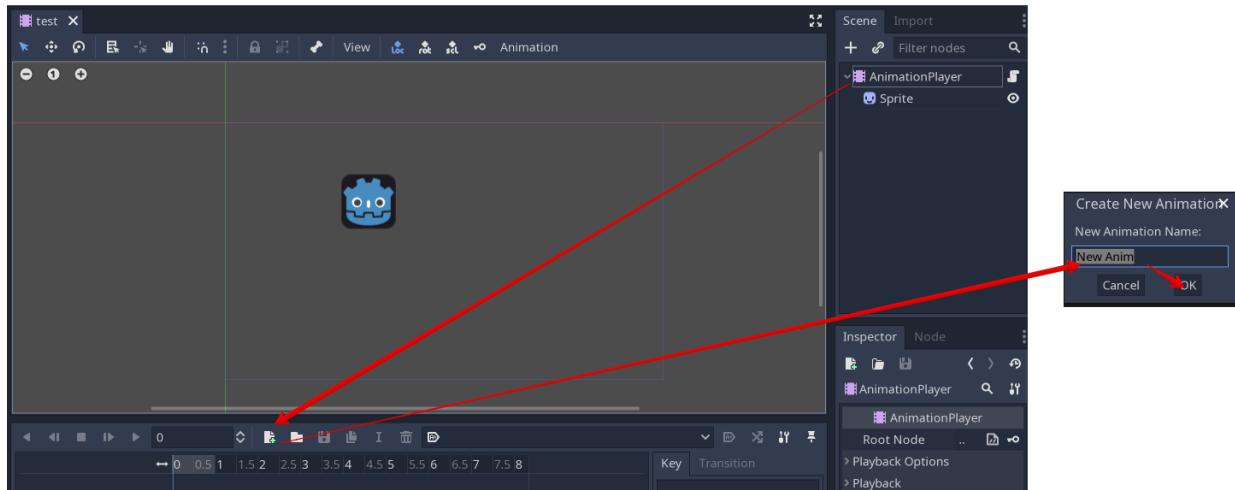


Fig. 11.9: Add a new animation

Adding a track

To add a new track for our sprite, select it and take a look in the toolbar:



Fig. 11.10: Convenience buttons

These switches and buttons allow you to add keyframes for the selected node’s location, rotation, and scale respectively.

Deselect rotation, because we are only interested in the location of our sprite for this tutorial and click on the key button.

As we don’t have a track already set up for the transform/location property, Godot asks, whether it should set it up for us. Click on “Create”.

This creates a new track and our very first keyframe at the beginning of the timeline:

The track name consists of a Node Path, followed by a colon, followed by a reference to its property, that we would like to modify.



Fig. 11.11: The sprite track

In our example, the path is `AnimationPlayer/Sprite` and the property is `position`.

The path always starts at the `AnimationPlayer` node's parent (so paths always have to include the `AnimationPlayer` node itself).

Note: Don't worry if you change the names of nodes in the Scene Tree, that you already have tracks for. Godot automatically updates the paths in the tracks.

The second keyframe

Now we need to set the destination where our sprite should be headed and how much time it takes to get there.

Let's say, we want it to take 2 seconds to go to the other point. By default the animation is set to last only 1 second, so change this in the timeline controls in animation panel's lower panel to 2.



Fig. 11.12: Animation length

Click on the timeline header near the 2 second mark and move the sprite to the target destination on the right side.

Again, click the key button in the toolbar. This creates our second keyframe.

Run the animation



Click on the "Play from beginning" () button.

Yay! Our animation runs:

Fig. 11.13: The animation

Back and forth

As you can see, the "loop" button is enabled by default and our animation loops. Godot has an additional feature here. Like said before, Godot always calculates the frames between two keyframes. In a loop, the first keyframe is also the last keyframe, if no keyframe is specified at the end.

If you set the animation length to 4 seconds now, the animation moves back and forth. You can change this behaviour if you change the track's loop mode. This is covered in the next chapter.



Fig. 11.14: Animation loop



Fig. 11.15: Track settings

Track settings

Each track has a settings panel at the end, where you can set the update rate, the track interpolation, and the loop mode.

The update rate of a track tells Godot when to update the property values. This can be:

- Continuous: Update the property on each frame
- Discrete: Only update the property on keyframes
- Trigger: Only update the property on keyframes or triggers



Fig. 11.16: Track rate

In normal animations, you usually use “Continuous”. The other types are used to script complex animations.

The interpolation tells Godot how to calculate the frame values between the keyframes. These interpolation modes are supported:

- Nearest: Set the nearest keyframe value
- Linear: Set the value based on a linear function calculation between the two keyframes
- Cubic: Set the value based on a cubic function calculation between the two keyframes

Cubic interpolation leads to a more natural movement, where the animation is slower at a keyframe and faster between keyframes. This is usually used for character animation. Linear interpolation creates more of a robotic movement.

Godot supports two loop modes, which affect the animation if it's set to loop:

- Clamp loop interpolation: When this is selected, the animation stops after the last keyframe for this track. When the first keyframe is reached again, the animation will reset to its values.
- Wrap loop interpolation: When this is selected, Godot calculates the animation after the last keyframe to reach the values of the first keyframe again.

11.1.5 Keyframes for other properties

Godot doesn't restrict you to only edit transform properties. Every property can be used as a track where you can set keyframes.

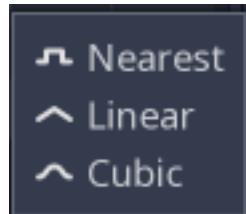


Fig. 11.17: Track interpolation



Fig. 11.18: Loop modes

If you select your sprite while the animation panel is visible, you get a small keyframe button for all of the sprite's properties. Click on this button and Godot automatically adds a track and keyframe to the current animation.

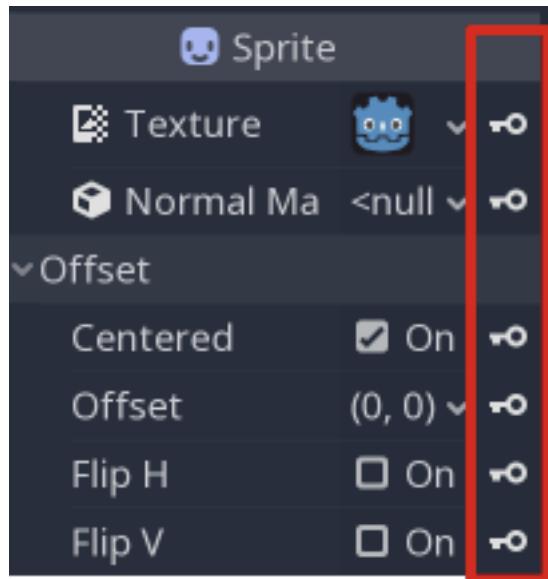


Fig. 11.19: Keyframes for other properties

11.1.6 Edit keyframes



For advanced use and to edit keyframe in detail, enable the keyframe editor ().

This adds an editor pane on the right side of the track settings. When you select a keyframe, you can directly edit its values in this editor:

Additionally, you can also edit the transition value for this keyframe:

This tells Godot, how to change the property values when it reaches this keyframe.

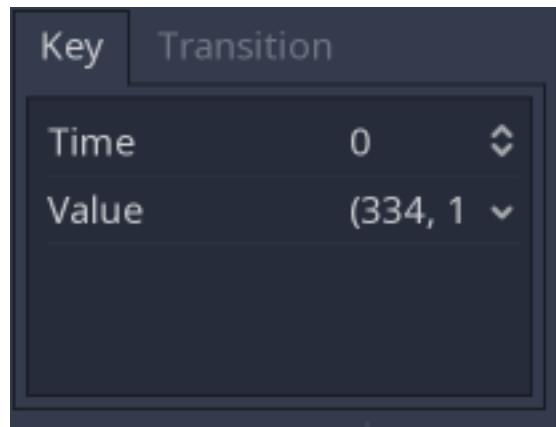


Fig. 11.20: Keyframe editor editing a key



Fig. 11.21: Keyframe editor editing a transition

You usually tweak your animations this way, when the movement doesn't "look right".

11.1.7 Advanced: Call Func tracks

Godot's animation engine doesn't stop here. If you're already comfortable with Godot's scripting language *GDScript* and *Godot API* you know that each node type is a class and has a bunch of callable functions.

For example, the *AudioStreamPlayer* node type has a function to play an audio stream.

Wouldn't it be great to play a stream at a specific keyframe in an animation? This is where "Call Func Tracks" come in handy. These tracks reference a node again, this time without a reference to a property. Instead, a keyframe holds the name and arguments of a function, that Godot should call when it reaches this keyframe.

To let Godot play a sample when it reaches a keyframe, follow this list:

Add a *AudioStreamPlayer* to the Scene Tree and setup a stream using an audio file you put in your project.

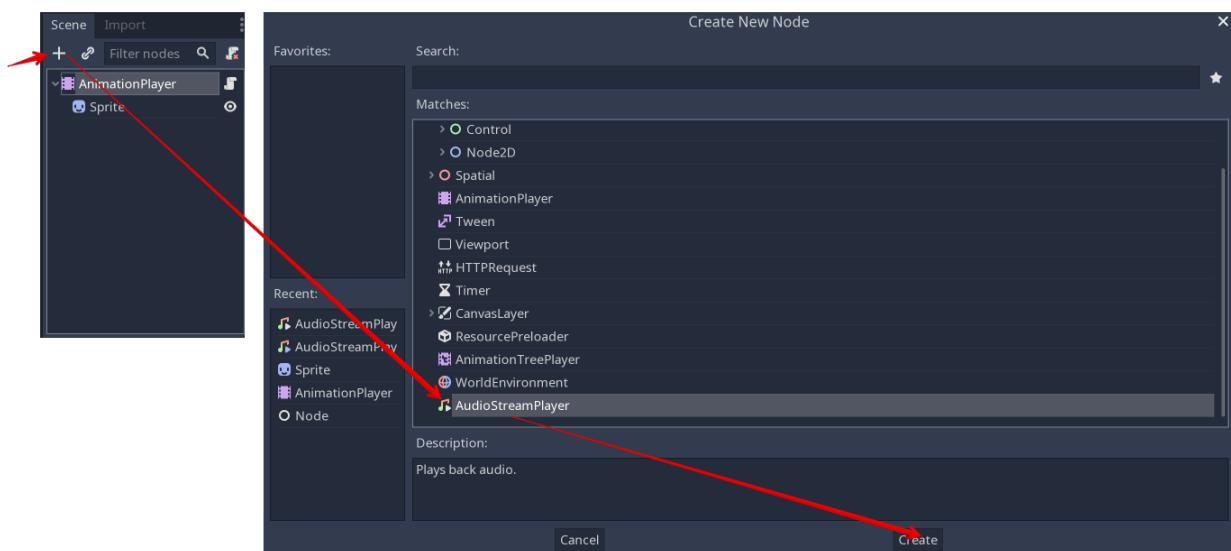


Fig. 11.22: Add AudioStreamPlayer



Click on "Add track" () on the animation panel's track controls.

Select "Add Call Func Track" from the list of possible track types.

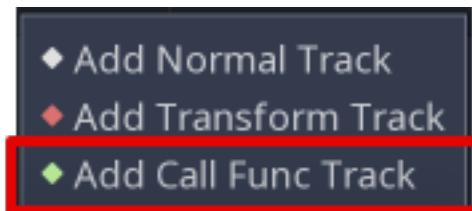


Fig. 11.23: Add Call Func Track

Select the *AudioStreamPlayer* node in the selection window. Godot adds the track with the reference to the node.



Fig. 11.24: Select AudioStreamPlayer

Select the timeline position, where Godot should play the sample by clicking on the timeline header.



Enable the Keyframe Editor by clicking on .



Click on “Add keyframe” near the settings of our func track () and select the keyframe.

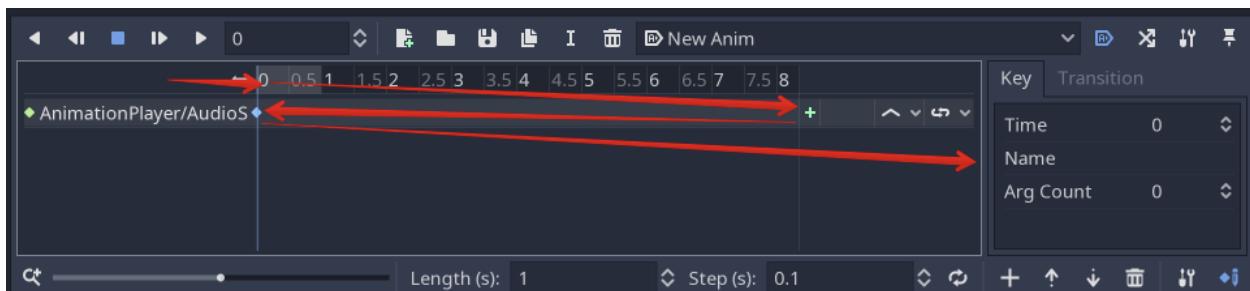


Fig. 11.25: Add a keyframe to the call func track

Enter “play” as the function name.

When Godot reaches the keyframe, Godot calls the *AnimationPlayer* node’s “play” function and the stream plays.

11.1.8 References

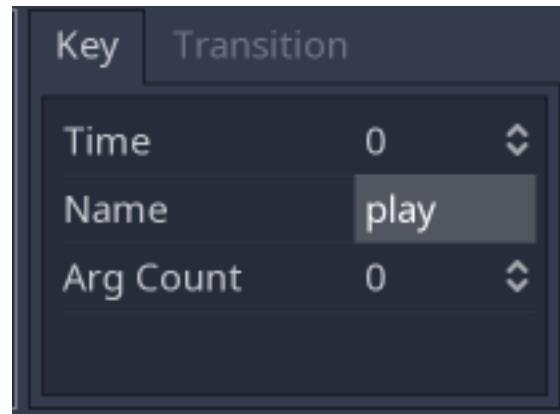


Fig. 11.26: Keyframe settings of a call func track

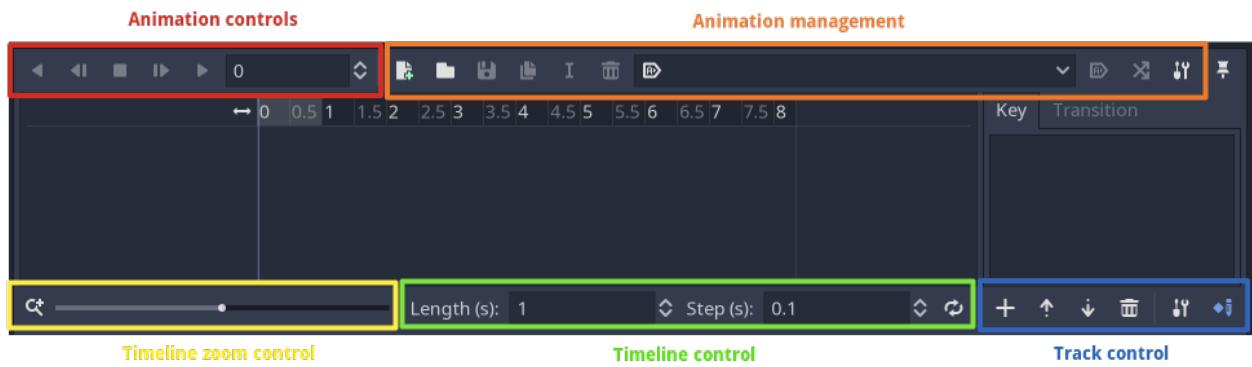


Fig. 11.27: The animation panel reference

Animation panel reference

The animation panel has the following parts (from left to right):

Animation controls

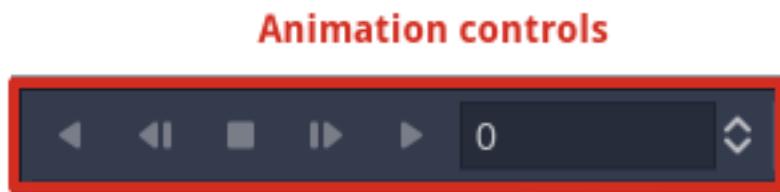


Fig. 11.28: Animation controls

- Play animation backwards from current position
- Play animation backwards from the animation end
- Stop animation
- Play animation forwards from the animation beginning
- Play animation forwards from the current position
- Direct time selection

Animation management

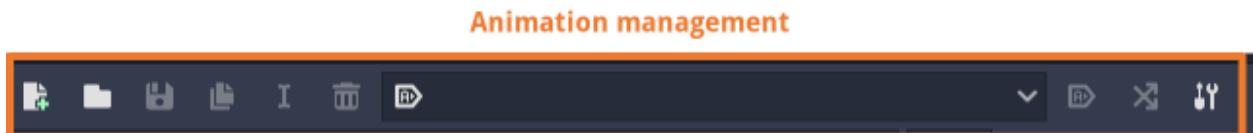


Fig. 11.29: Animation management

- Create a new animation
- Load animation
- Save animation
- Duplicate animation
- Rename animation
- Delete animation
- Animation selection
- Automatically play selected animation
- Edit animation blend times
- Extended animation Tools

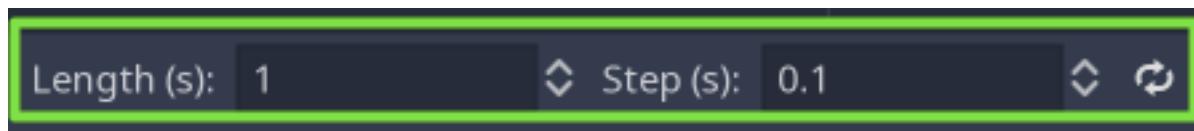
Timeline zoom level control



Timeline zoom control

Fig. 11.30: Timeline zoom level contro

Timeline control

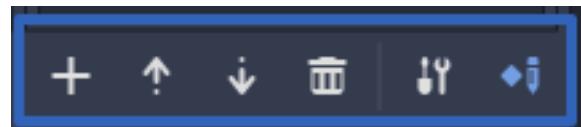


Timeline control

Fig. 11.31: Timeline control

- Length of animation
- Steps of animation
- Toggle loop animation

Track control



Track control

Fig. 11.32: Track control

- Add track
- Move track up
- Move track down
- Delete track
- Extended track tools
- Toggle keyframe editor

11.2 Cutout animation

11.2.1 What is it?

Cut-out is a technique of animating in 2D where pieces of paper (or similar material) are cut in special shapes and laid one over the other. The papers are animated and photographed, frame by frame using a stop motion technique (more info [here](#)).

With the advent of the digital age, this technique became possible using computers, which resulted in an increased amount of animation TV shows using digital Cut-out. Notable examples are [South Park](#) or [Jake and the Never Land Pirates](#).

In video games, this technique also become very popular. Examples of this are [Paper Mario](#) or [Rayman Origins](#).

11.2.2 Cutout in Godot

Godot provides a few tools for working with these kind of assets, but its overall design makes it ideal for the workflow. The reason is that, unlike other tools meant for this, Godot has the following advantages:

- **The animation system is fully integrated with the engine:** This means, animations can control much more than just motion of objects, such as textures, sprite sizes, pivots, opacity, color modulation, etc. Everything can be animated and blended.
- **Mix with Traditional:** AnimatedSprite allows traditional animation to be mixed, very useful for complex objects, such as shape of hands and foot, changing facial expression, etc.
- **Custom Shaped Elements:** Can be created with *Polygon2D* allowing the mixing of UV animation, deformations, etc.
- **Particle Systems:** Can also be mixed with the traditional animation hierarchy, useful for magic effects, jetpacks, etc.
- **Custom Colliders:** Set colliders and influence areas in different parts of the skeletons, great for bosses, fighting games, etc.
- **Animation Tree:** Allows complex combinations and blendings of several animations, the same way it works in 3D.

And much more!

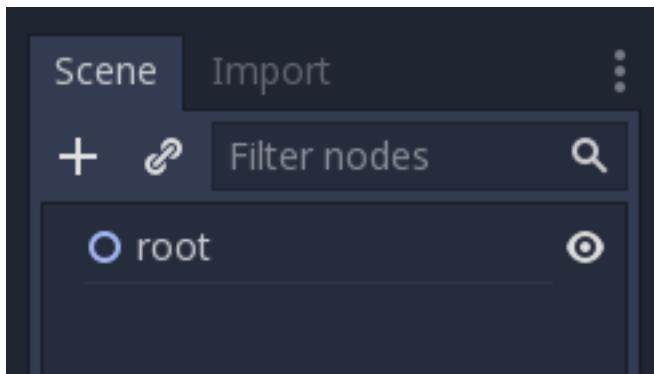
11.2.3 Making of GBot!

For this tutorial, we will use as demo content the pieces of the [GBot](#) character, created by Andreas Esau.

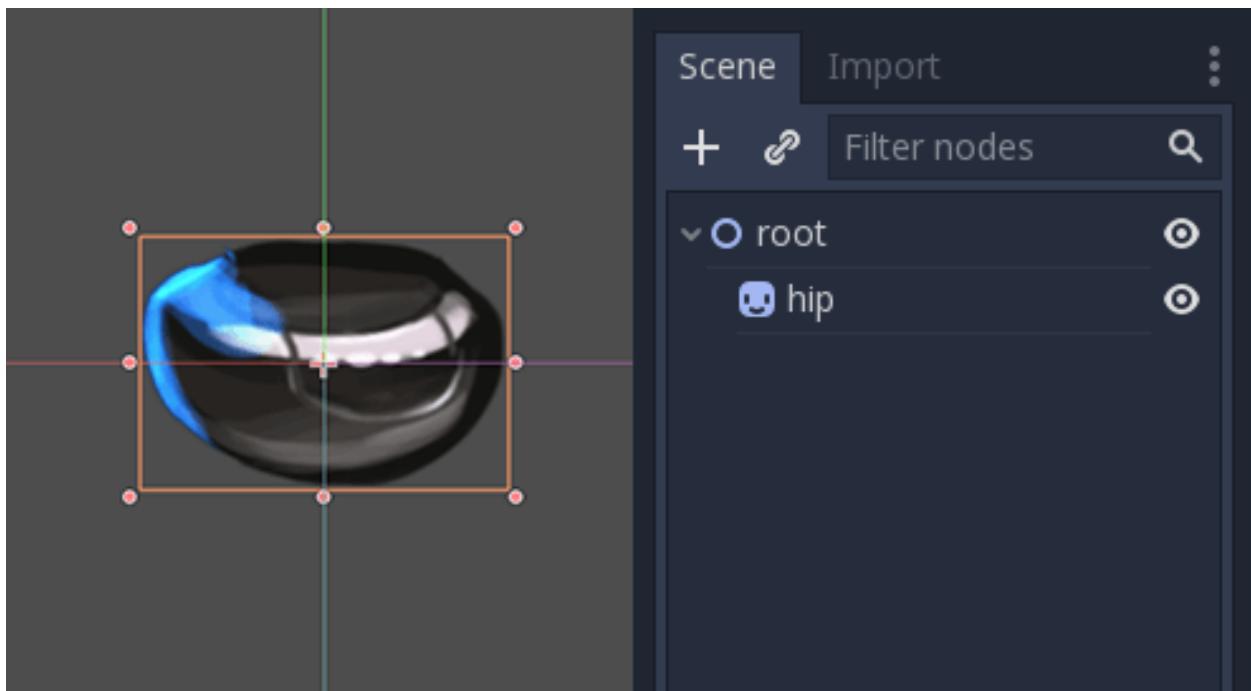
Get your assets: `gbot_resources.zip`.

11.2.4 Setting up the rig

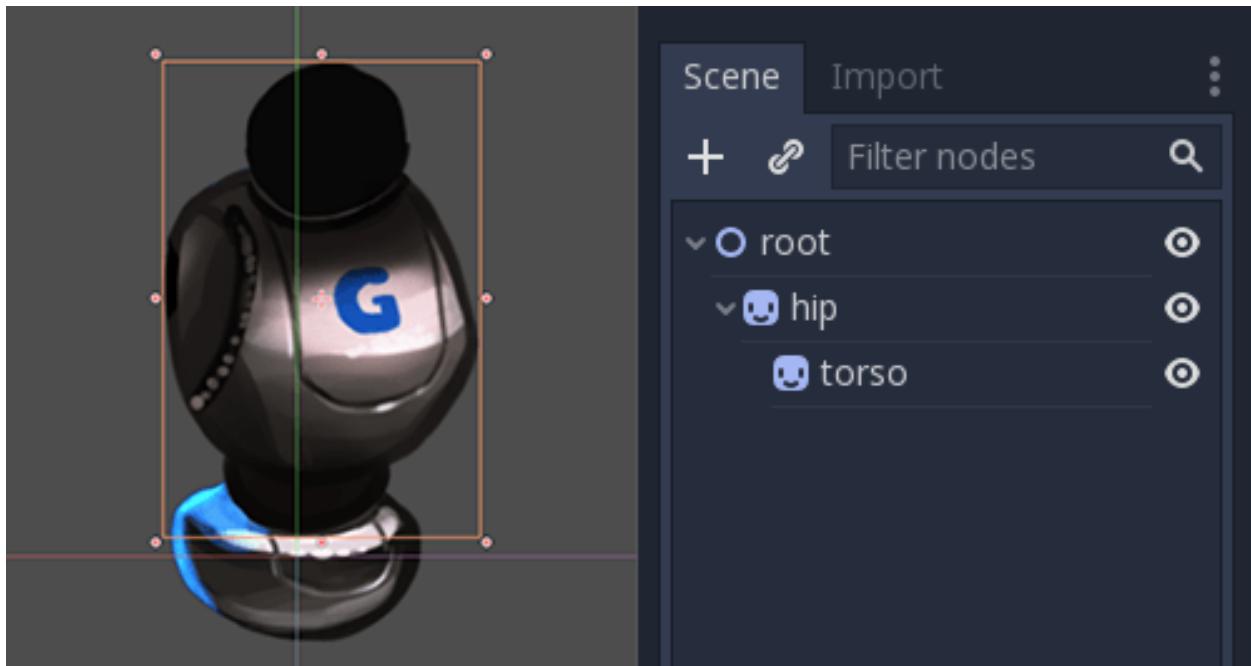
Create an empty Node2D as root of the scene, we will work under it:



OK, the first node of the model that we will create will be the hip. Generally, both in 2D and 3D, the hip is the root of the skeleton. This makes it easier to animate:



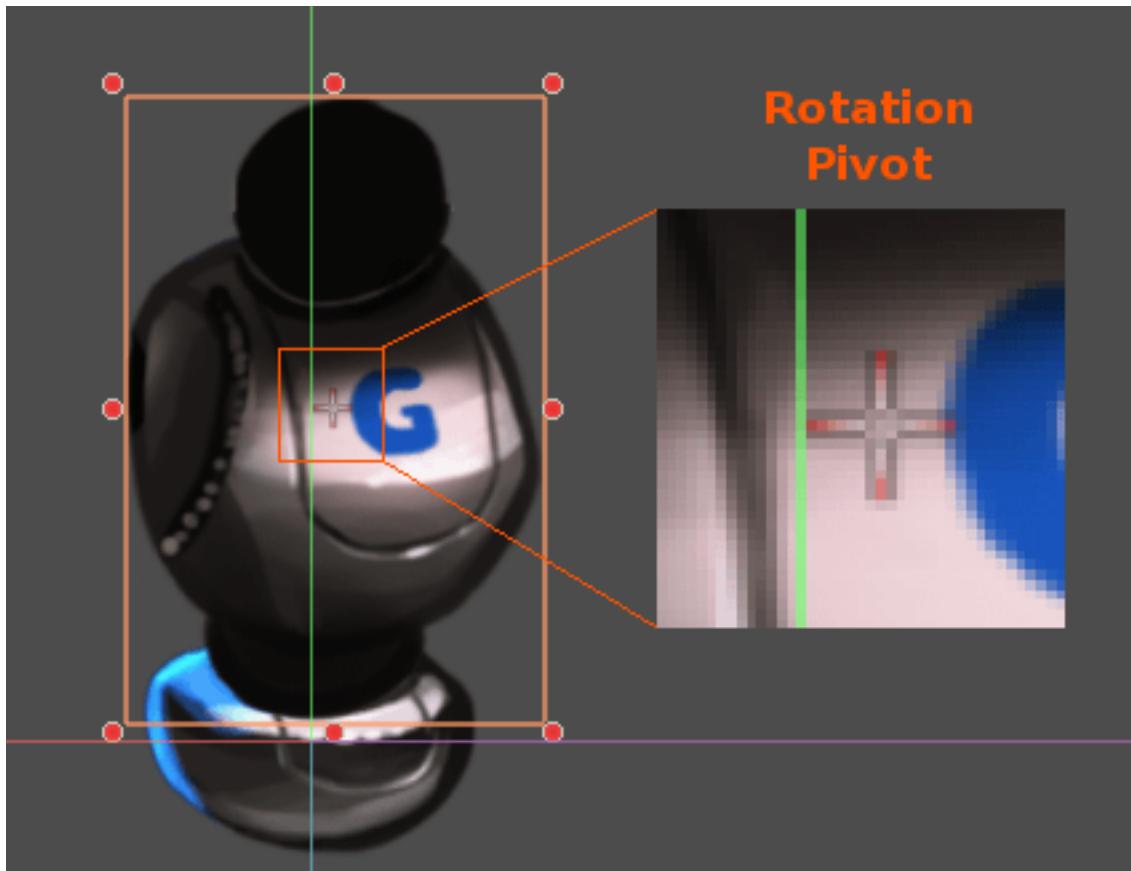
Next will be the torso. The torso needs to be a child of the hip, so create a child sprite and load the torso, later accommodate it properly:



This looks good. Let's see if our hierarchy works as a skeleton by rotating the torso. We can do this by pressing E to enter rotate mode, and dragging with the left mouse button. To exit rotate mode just hit ESC.

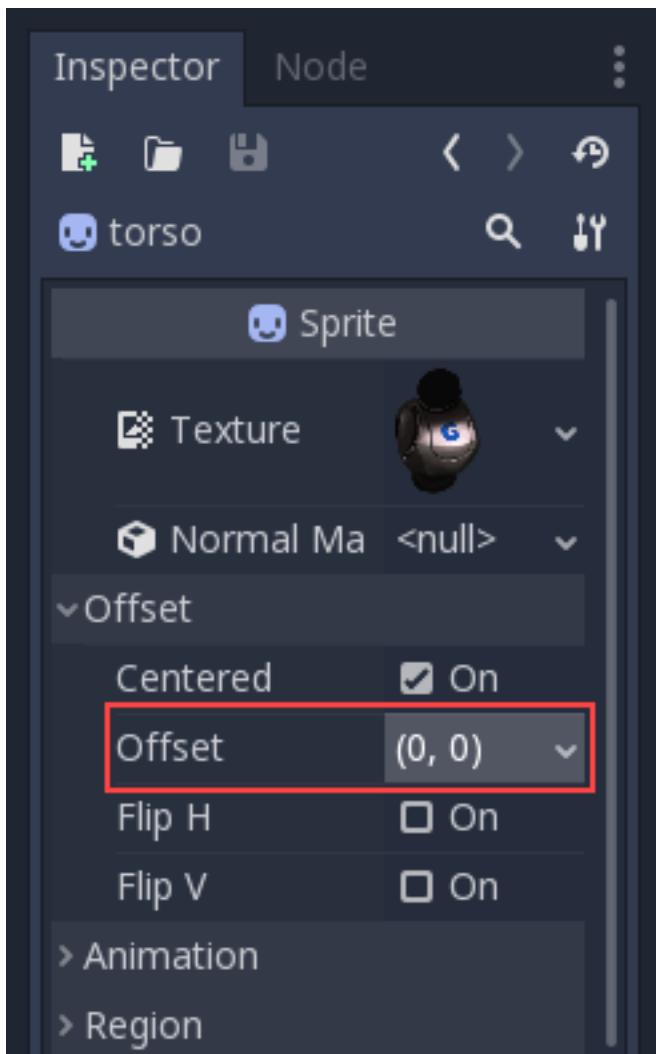
Ouch, that doesn't look good! The rotation pivot is wrong, this means it needs to be adjusted.

This small little cross in the middle of the *Sprite* is the rotation pivot:



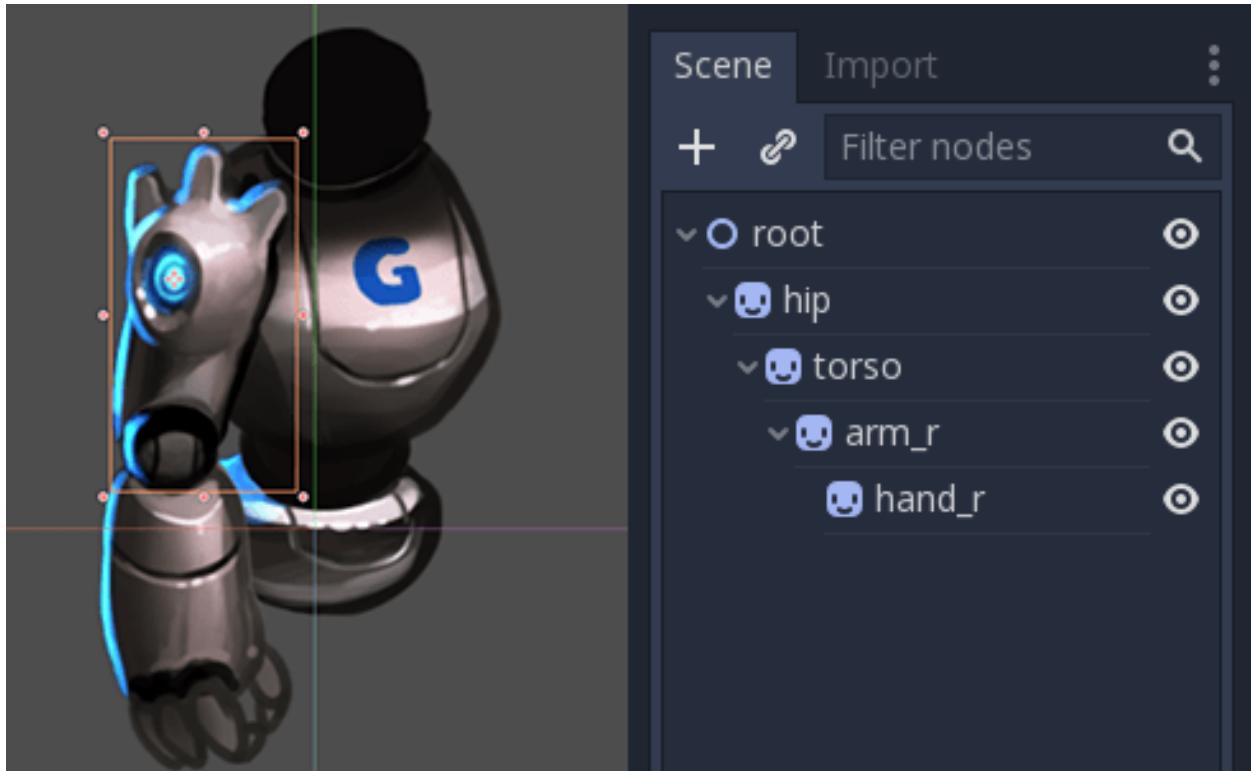
11.2.5 Adjusting the pivot

The pivot can be adjusted by changing the *offset* property in the Sprite:

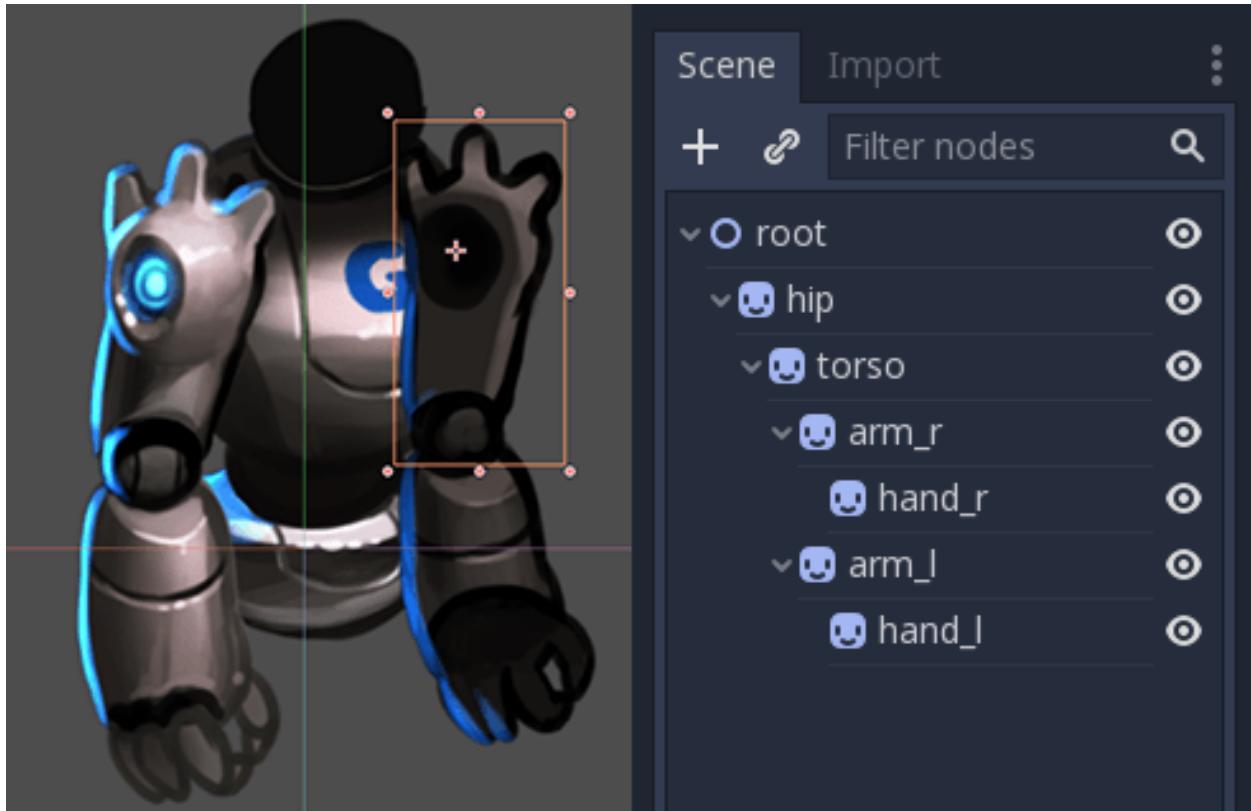


However, there is a way to do it more *visually*. While hovering over the desired pivot point, simply press the “v” key to move the pivot there for the selected Sprite. Alternately, there is a tool in the tool bar that has a similar function.

Now it looks good! Let’s continue adding body pieces, starting by the right arm. Make sure to put the sprites in hierarchy, so their rotations and translations are relative to the parent:



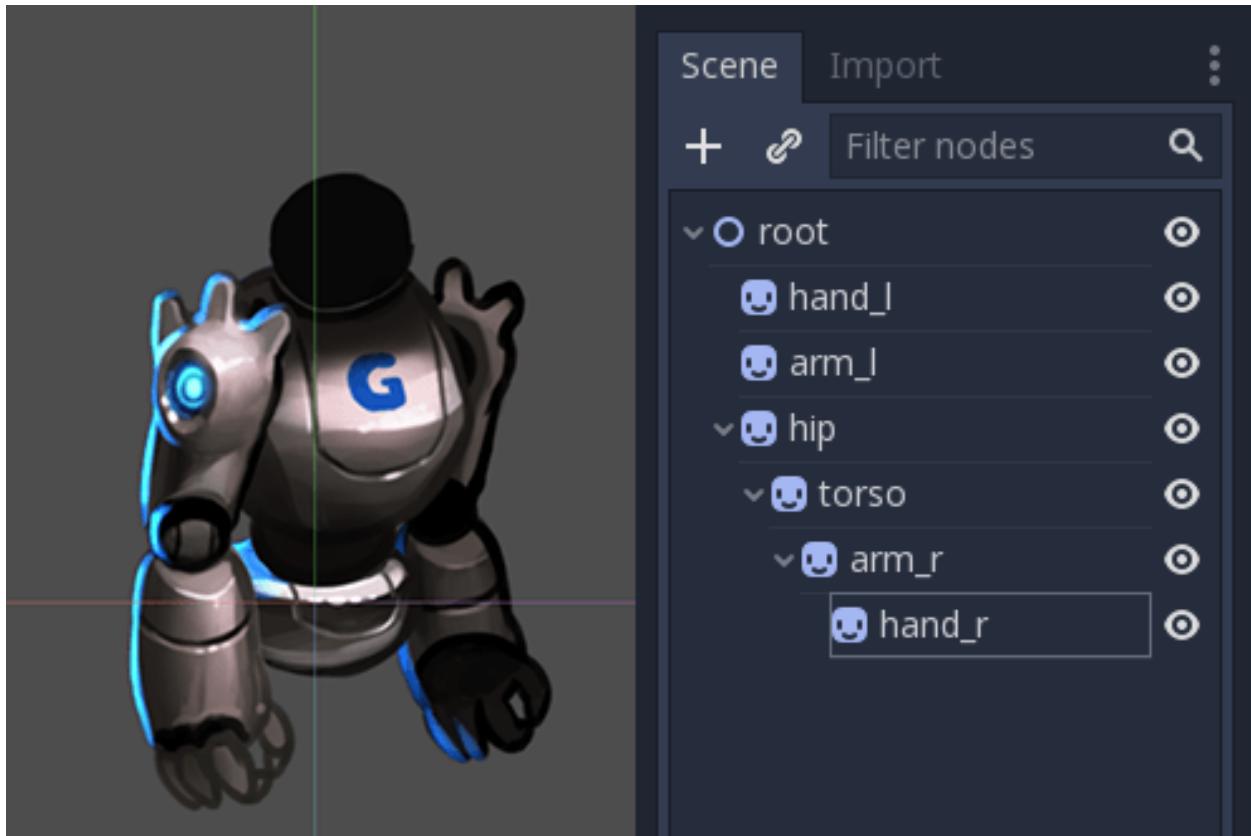
This seems easy, so continue with the left arm. The rest should be simple! Or maybe not:



Right. Remember your tutorials, Luke. In 2D, parent nodes appear below children nodes. Well, this sucks. It seems Godot does not support cutout rigs after all. Come back next year, maybe for 3.0.. no wait. Just Kidding! It works just

fine.

But how can this problem be solved? We want the left arm to appear behind the hip and the torso. For this, we can move the nodes behind the hip (note that you can bypass this by setting the Node2D Z property, but then you won't learn about all this!):



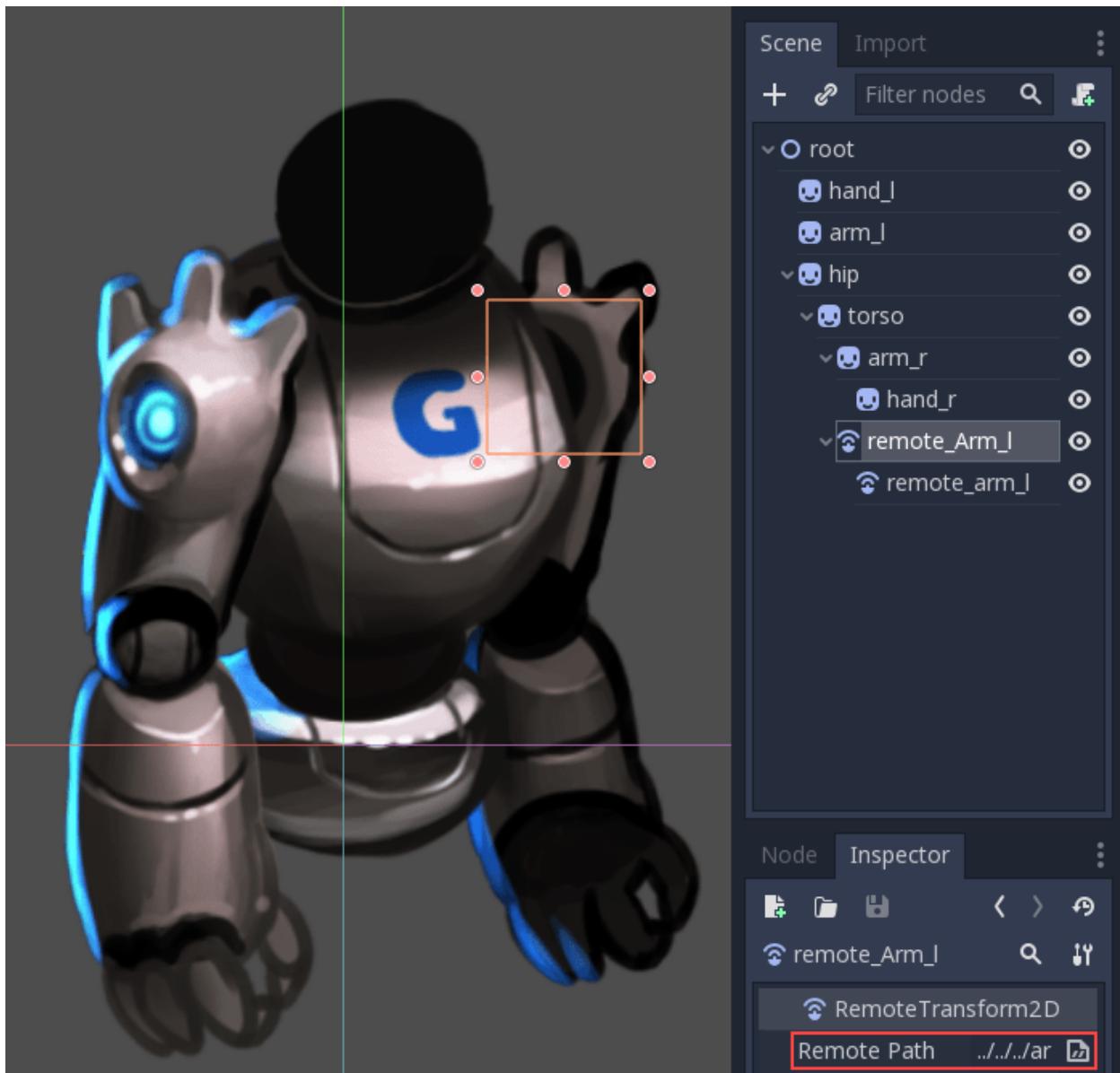
But then, we lose the hierarchy layout, which allows to control the skeleton like.. a skeleton. Is there any hope?.. Of Course!

11.2.6 RemoteTransform2D node

Godot provides a special node, *RemoteTransform2D*. This node will transform nodes that are sitting somewhere else in the hierarchy, by applying the transform to the remote nodes.

This enables to have a visibility order independent from the hierarchy.

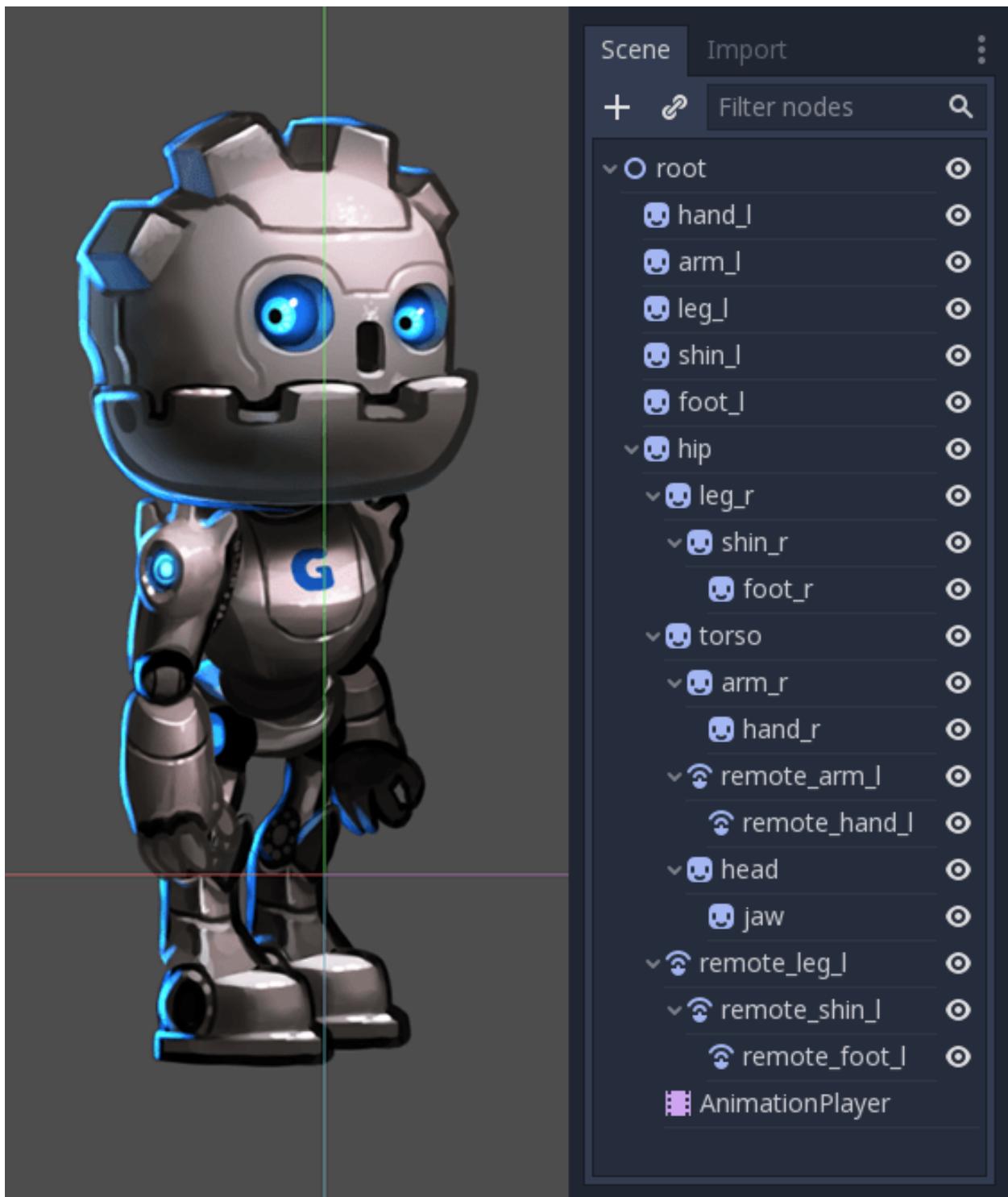
Simply create two more nodes as children from torso, `remote_arm_l` and `remote_hand_l` and link them to the actual sprites:



Moving the remote transform nodes will move the sprites, allowing you to easily animate and pose the character:

11.2.7 Completing the skeleton

Complete the skeleton by following the same steps for the rest of the parts. The resulting scene should look similar to this:



The resulting rig will be easy to animate. By selecting the nodes and rotating them you can animate forward kinematics (FK) efficiently.

For simple objects and rigs this is fine, however the following problems are common:

- Selecting sprites can become difficult for complex rigs, and the scene tree ends being used due to the difficulty of clicking over the proper sprite.

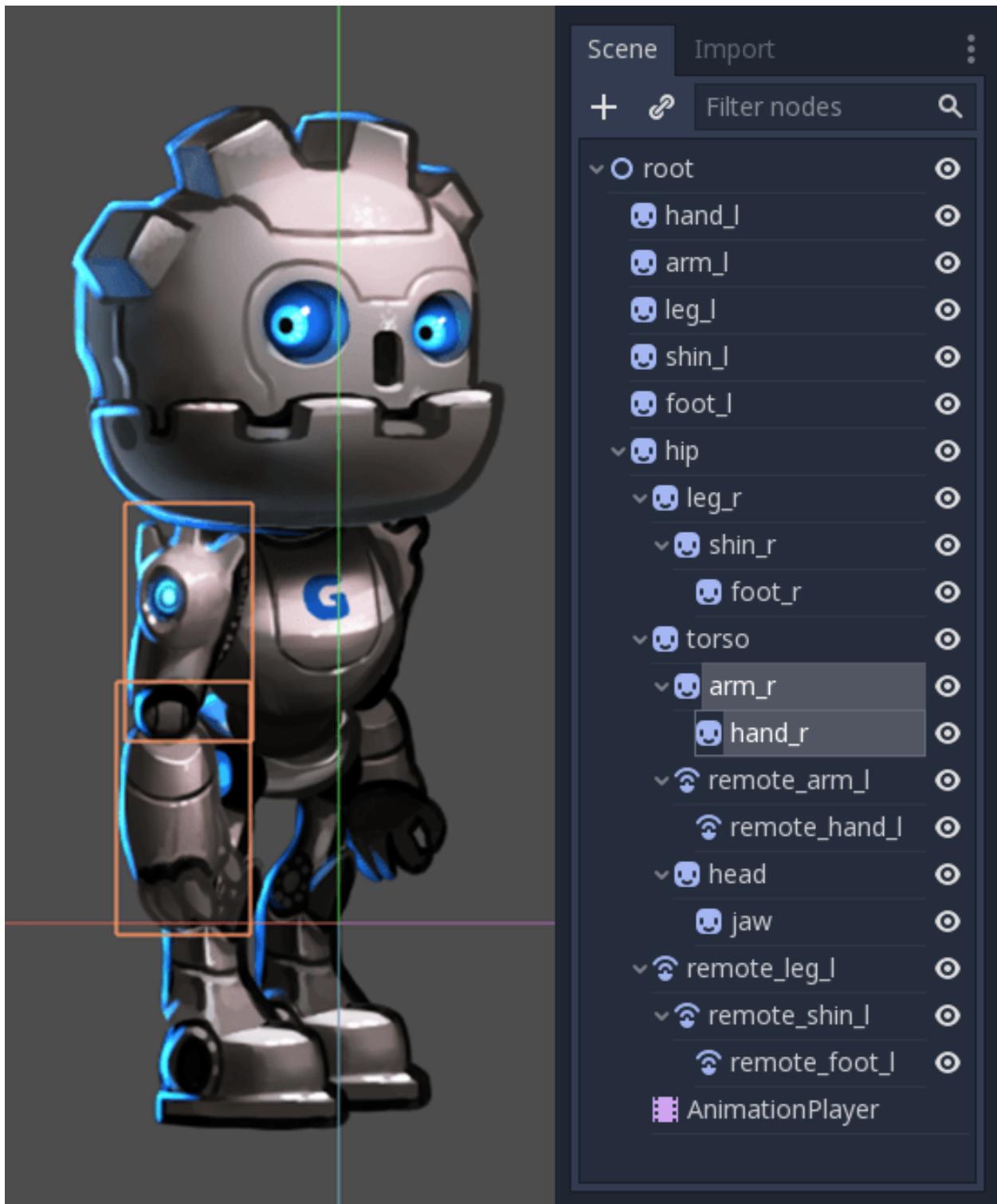
- Inverse Kinematics is often desired for extremities.

To solve these problems, Godot supports a simple method of skeletons.

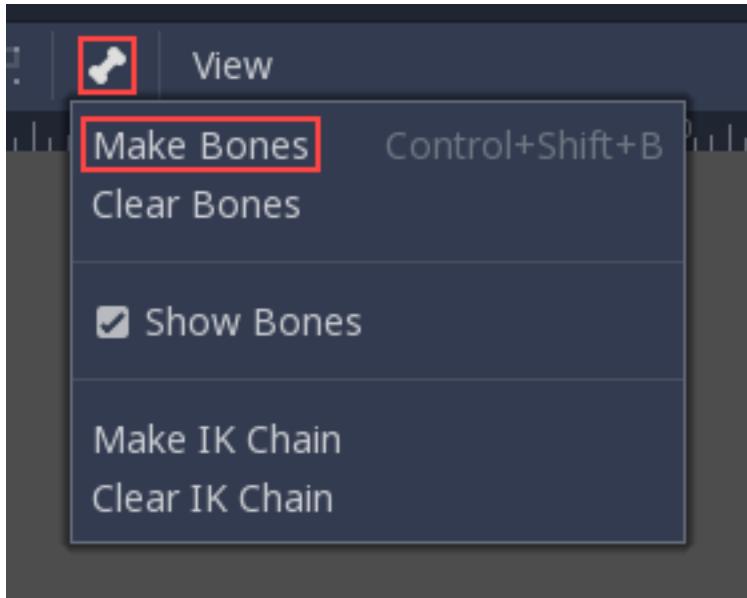
11.2.8 Skeletons

Godot doesn't actually support *true* Skeletons, but it does feature a helper to create “bones” between nodes. This is enough for most cases, but the way it works is not completely obvious.

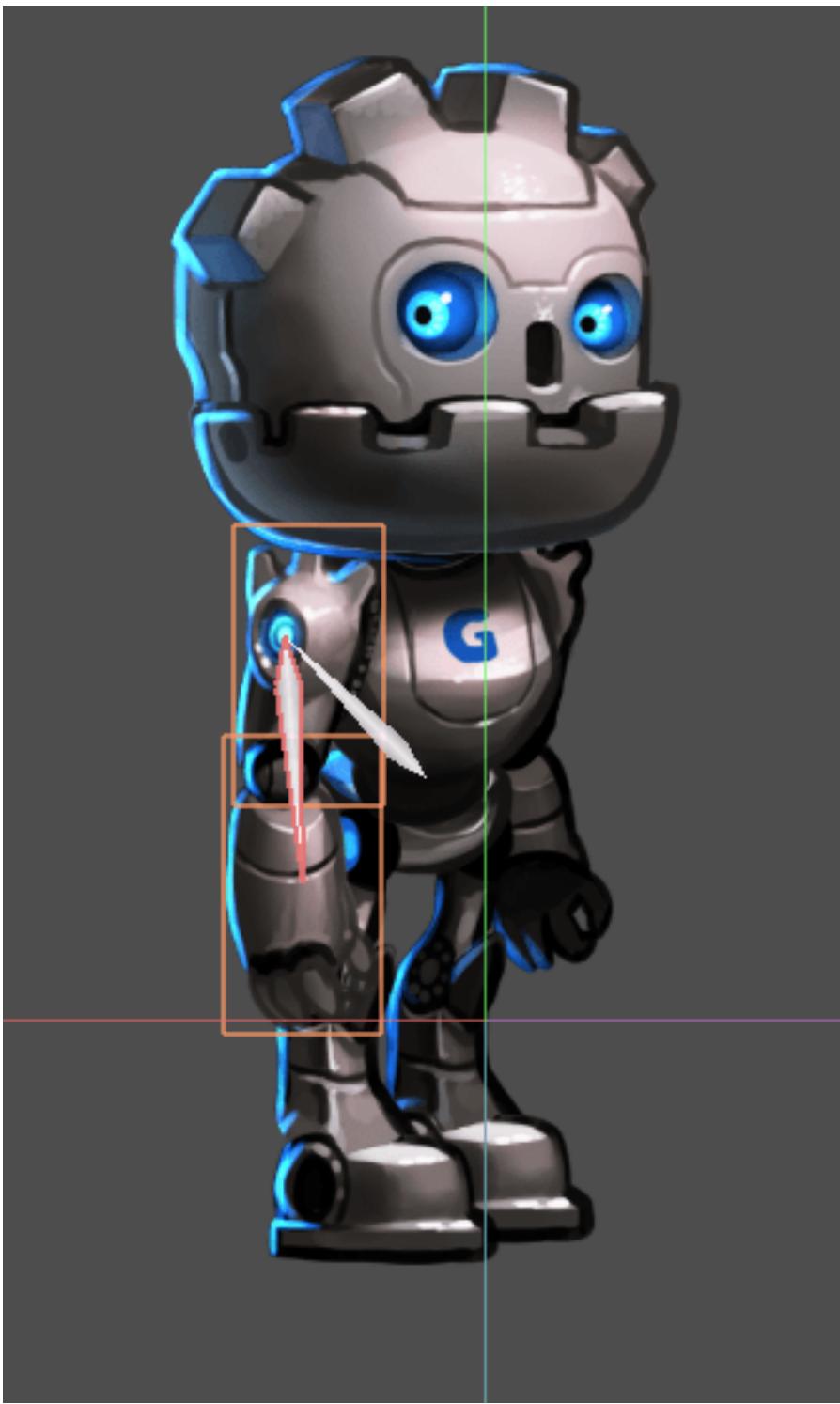
As an example, let's turn the right arm into a skeleton. To create skeletons, a chain of nodes must be selected from top to bottom:



Then, click on the Skeleton menu and select Make Bones.



This will add bones covering the arm, but the result is not quite what is expected.



It looks like the bones are shifted up in the hierarchy. The hand connects to the arm, and the arm to the body. So the question is:

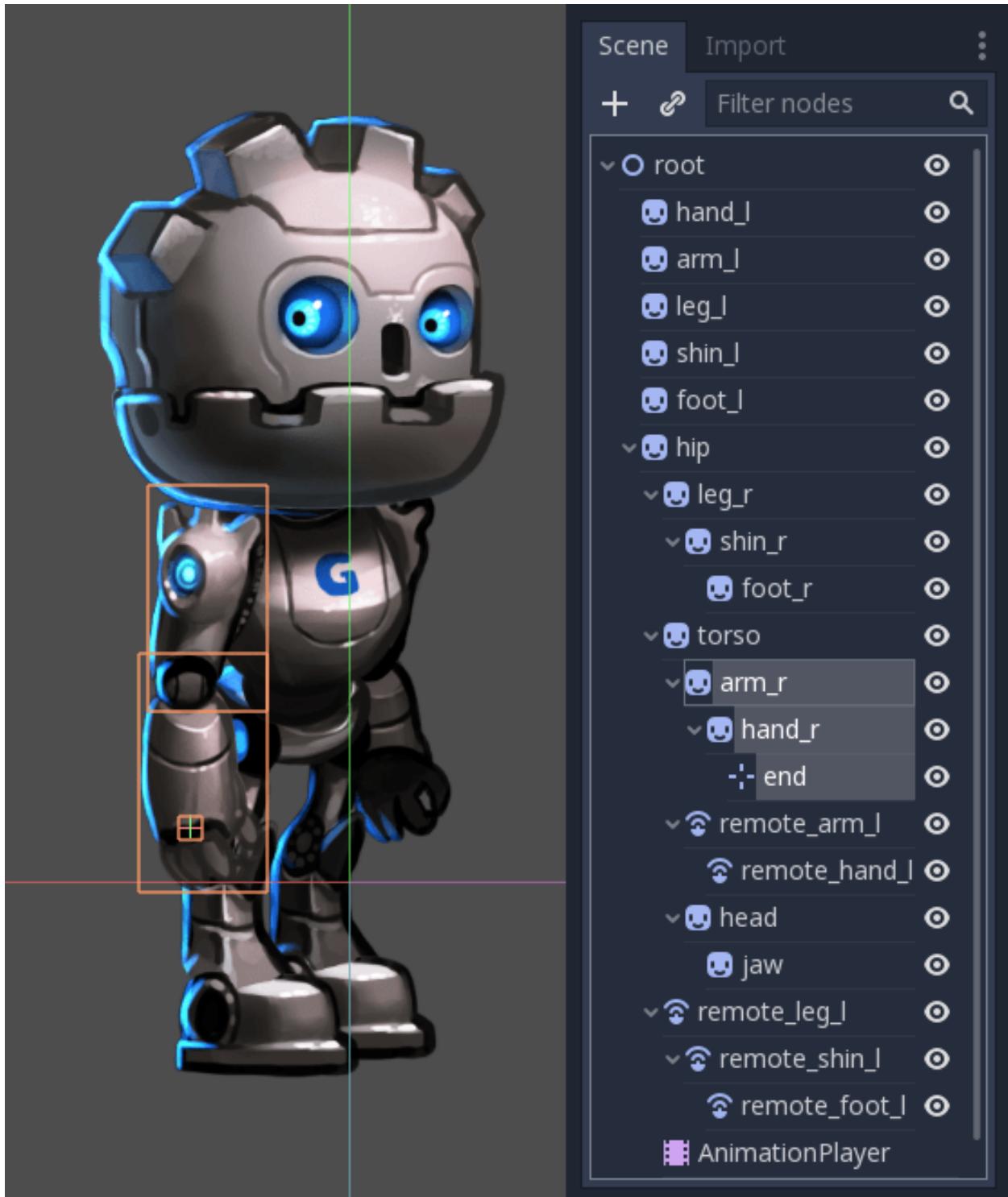
- Why does the hand lack a bone?
- Why does the arm connect to the body?

This might seem strange at first, but will make sense later on. In traditional skeleton systems, bones have a position, an

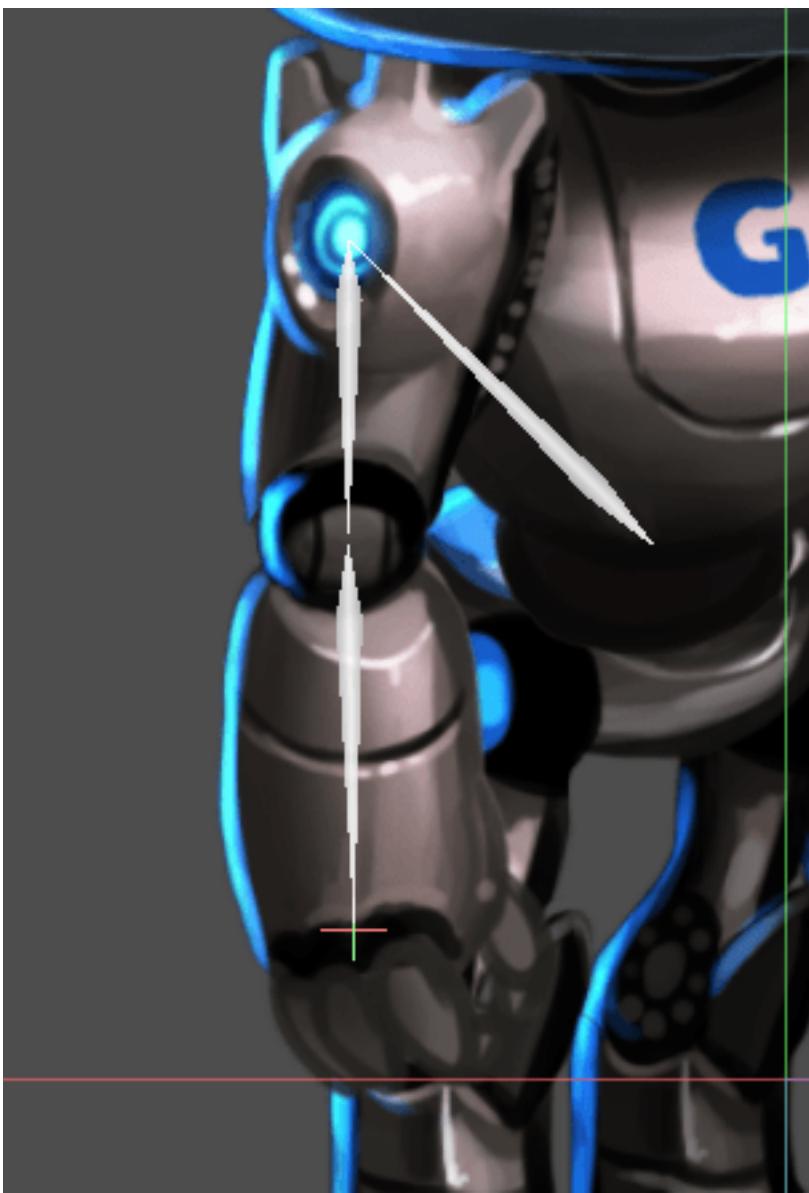
orientation and a length. In Godot, bones are mostly helpers so they connect the current node with the parent. Because of this, **toggling a node as a bone will just connect it to the parent**.

So, with this knowledge. Let's do the same again so we have an actual, useful skeleton.

The first step is creating an endpoint node. Any kind of node will do, but *Position2D* is preferred because it's visible in the editor. The endpoint node will ensure that the last bone has orientation.



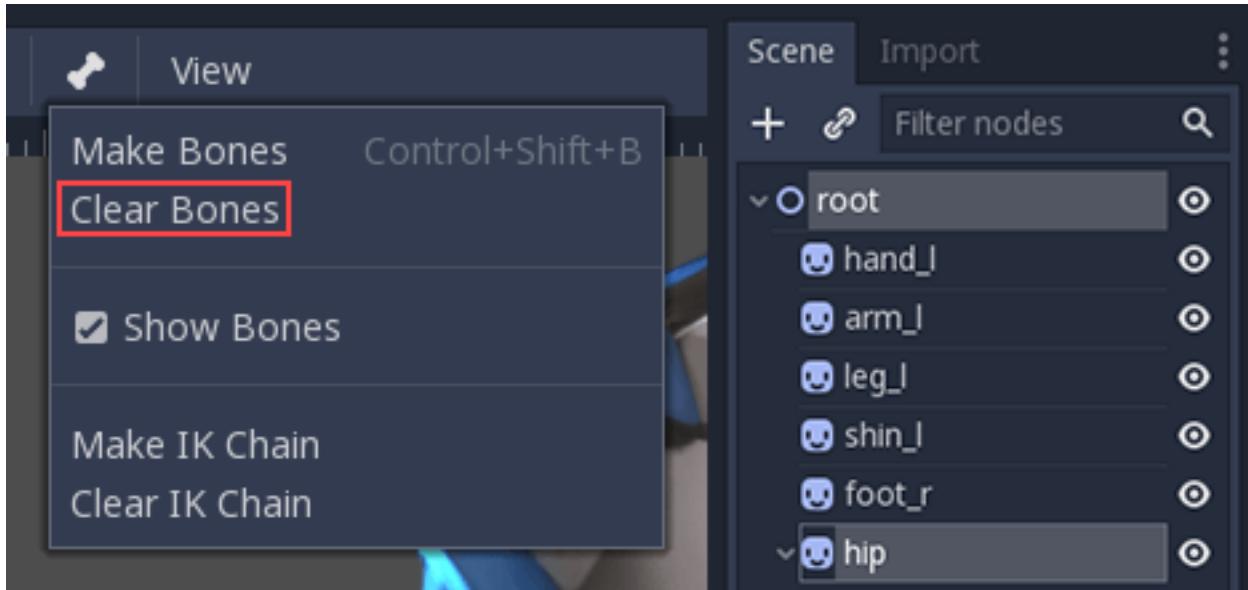
Now select the whole chain, from the endpoint to the arm and create bones:



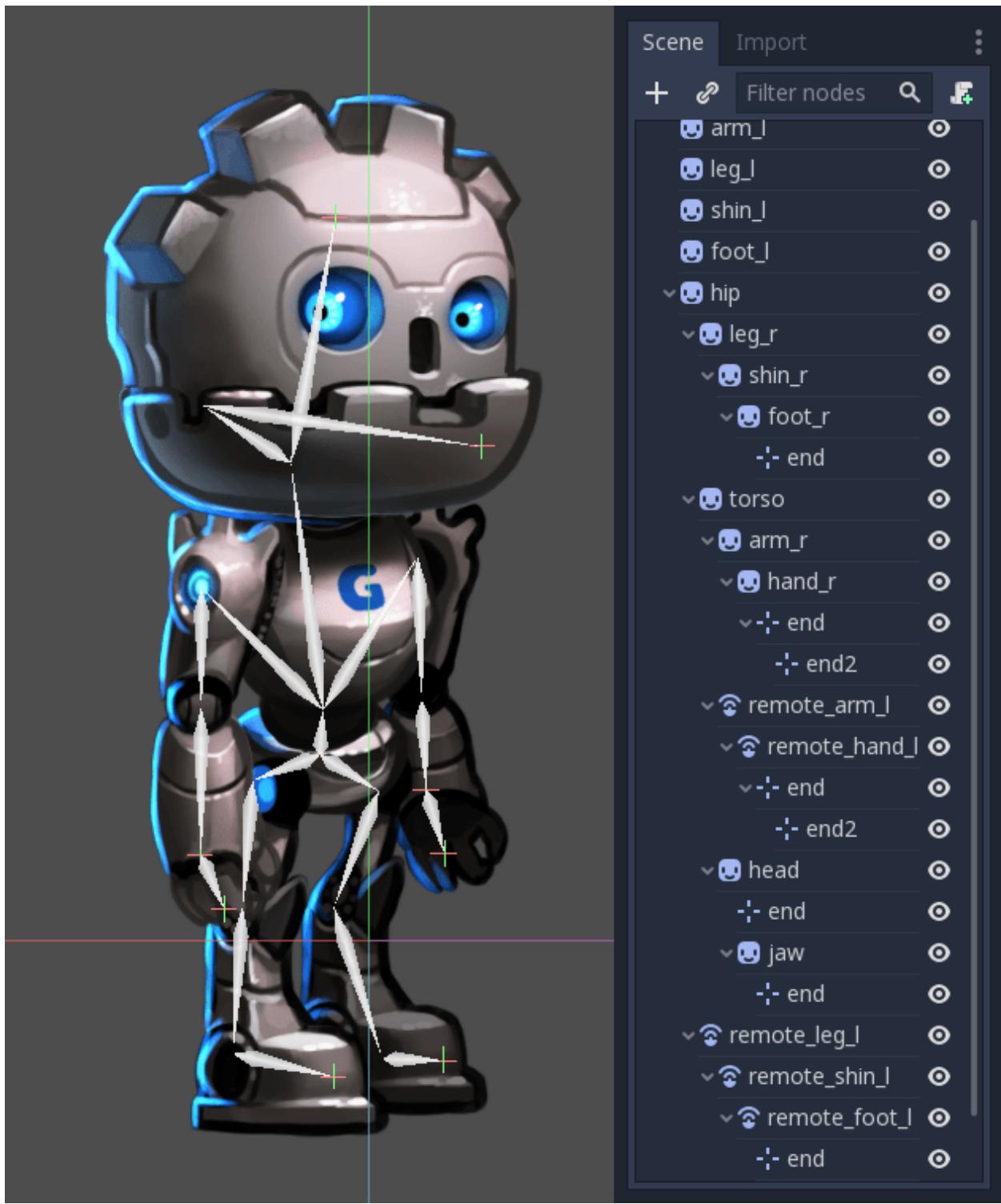
The result resembles a skeleton a lot more, and now the arm and forearm can be selected and animated.

Finally, create endpoints in all meaningful extremities and connect the whole skeleton with bones up to the hip.

You may notice when connecting the hip and torso, that an extra bone is created. To fix this, select the root and hip node, open the Skeleton menu, click clear bones.



After fixing that your final skeleton should look something like this:



Finally! the whole skeleton is rigged! On close look, it is noticeable that there is a second set of endpoints in the hands. This will make sense soon.

Now that a whole skeleton is rigged, the next step is setting up the IK chains. IK chains allow for more natural control of extremities.

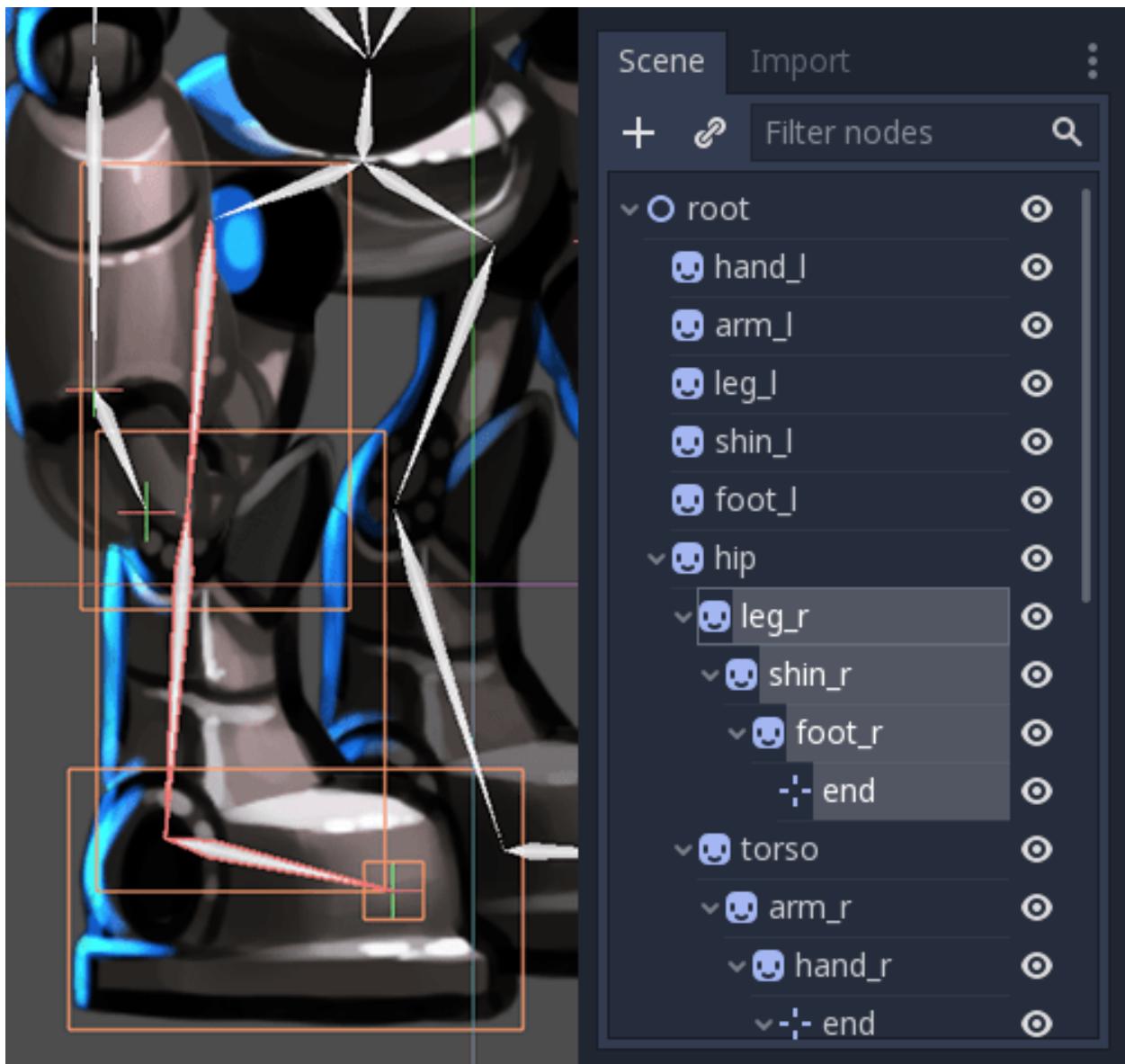
11.2.9 IK chains

IK chains are a powerful animation tool. Imagine you want to pose a character's foot in a specific position on the ground. Without IK chains, each motion of the foot would require rotating and positioning several other bones. This would be quite complex and lead to imprecise results.

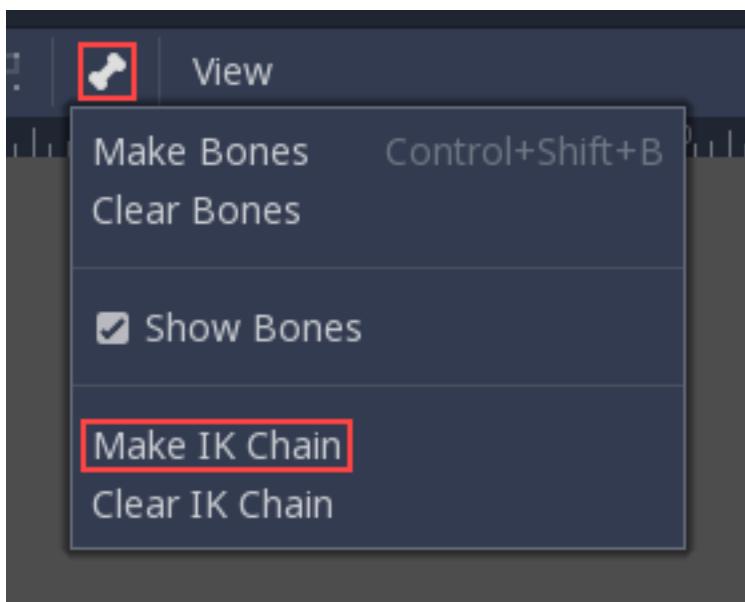
What if we could move the foot and let the rest of the leg self-adjust?

This type of posing is called IK (Inverse Kinematic).

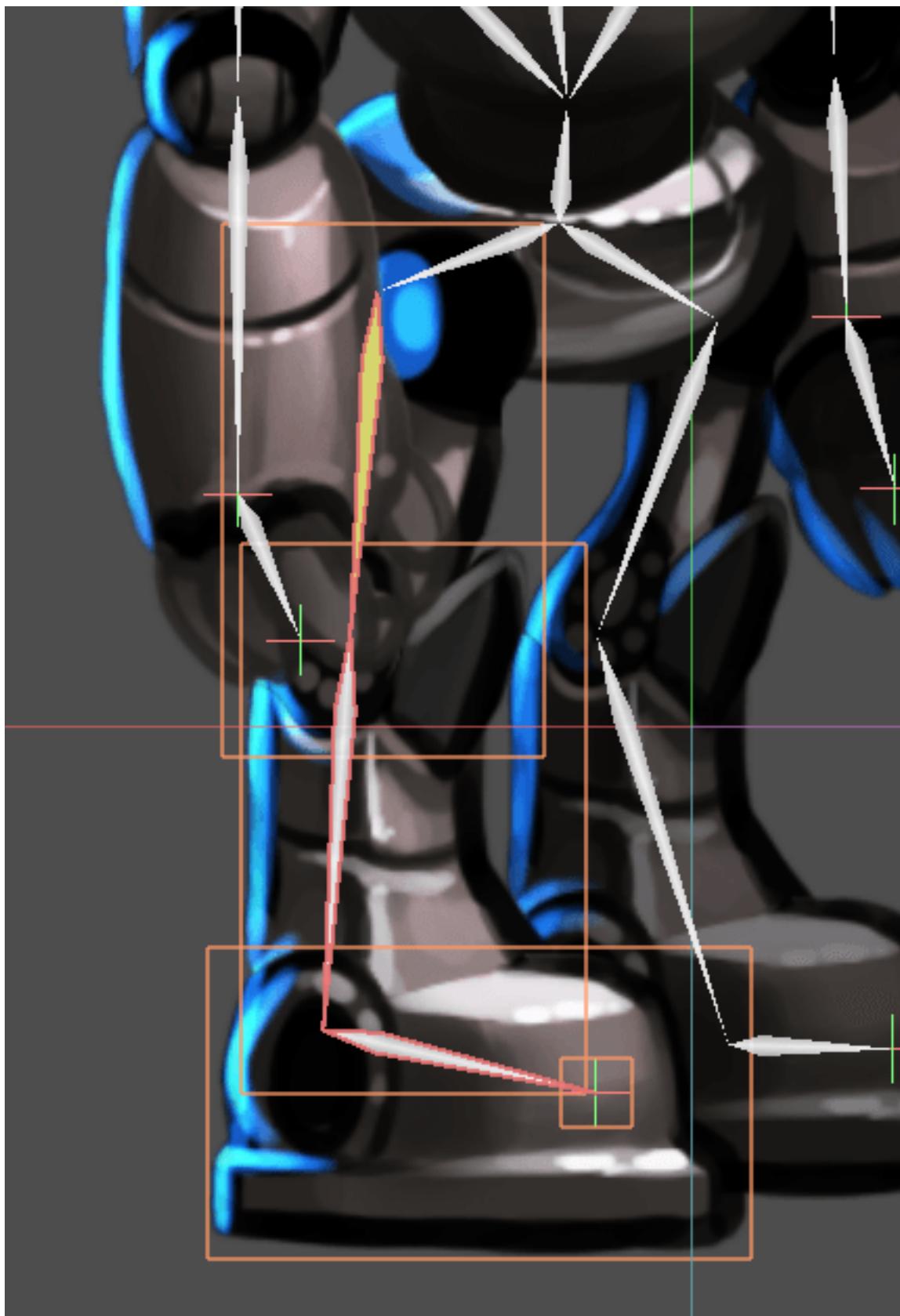
To create an IK chain, simply select a chain of bones from endpoint to the base for the chain. For example, to create an IK chain for the right leg, select the following:



Then enable this chain for IK. Go to Edit > Make IK Chain.



As a result, the base of the chain will turn *Yellow*.



Once the IK chain is set-up, simply grab any of the bones in the extremity, any child or grand-child of the base of the

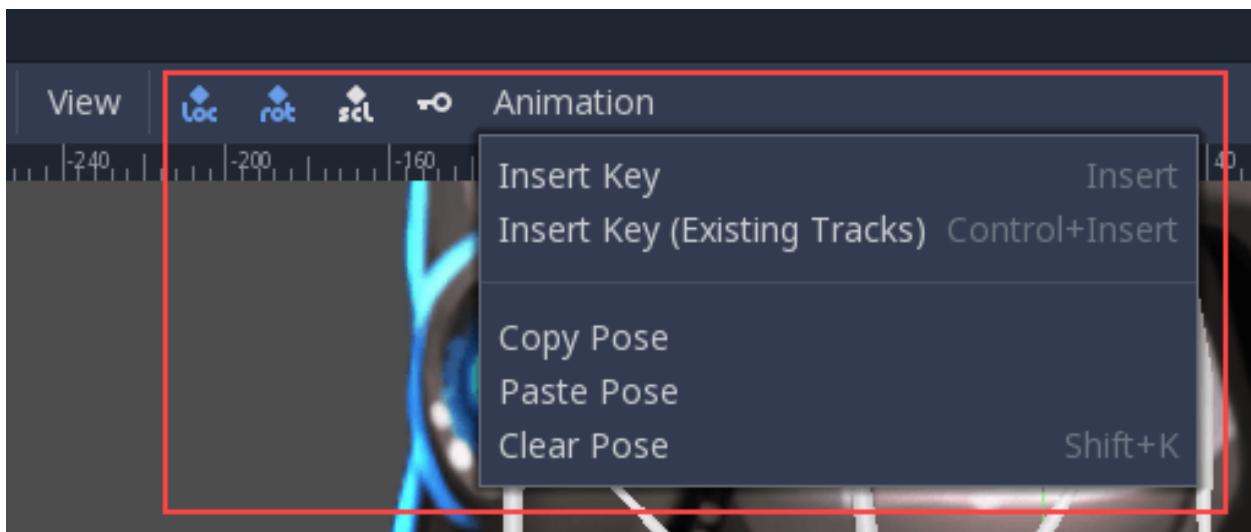
chain and try to grab it and move it. Result will be pleasant, satisfaction warranted!

11.2.10 Animation

The following section will be a collection of tips for creating animation for your rigs. If unsure about how the animation system in Godot works, refresh it by checking again the *Animations*.

2D animation

When doing animation in 2D, a helper will be present in the top menu. This helper only appears when the animation editor window is opened:



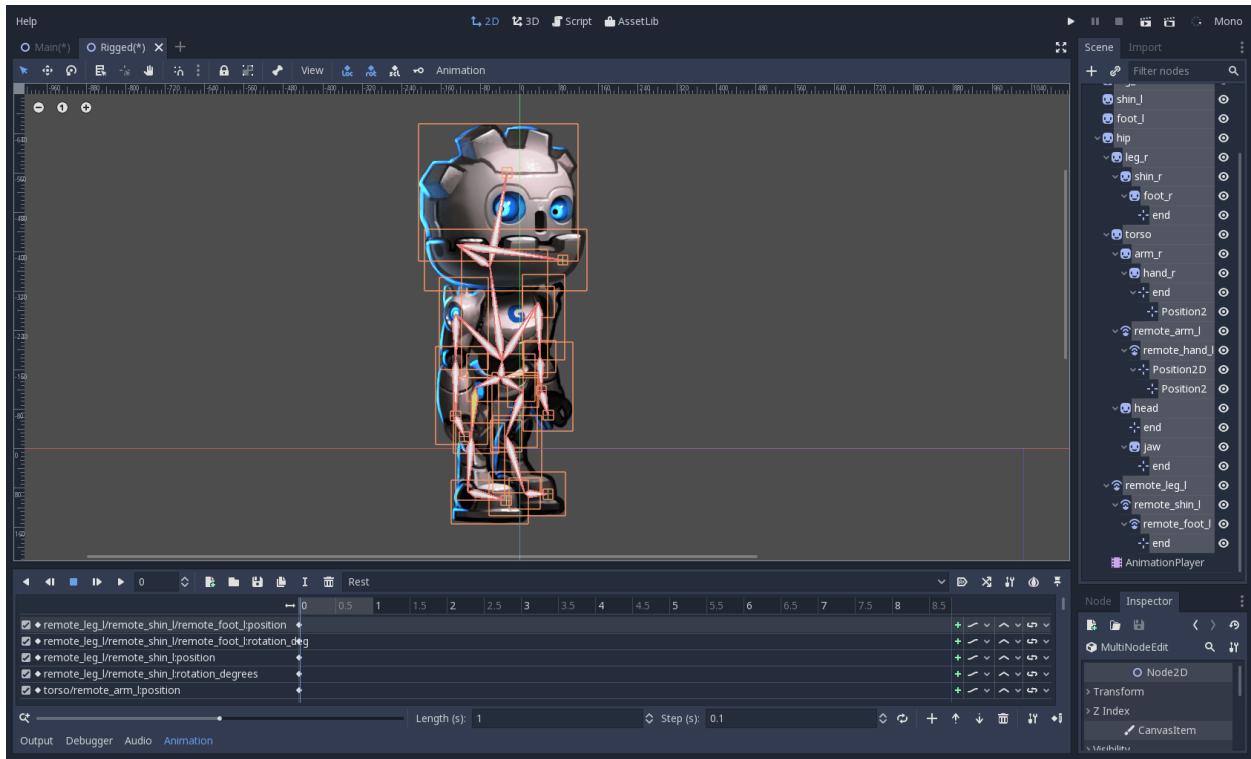
The key button will insert location/rotation/scale keyframes to the selected objects or bones. This depends on the mask enabled. Green items will insert keys while red ones will not, so modify the key insertion mask to your preference.

11.2.11 Rest pose

These kind of rigs do not have a “rest” pose, so it’s recommended to create a reference rest pose in one of the animations.

Simply do the following steps:

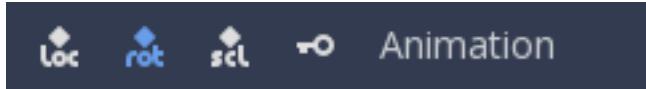
1. Make sure the rig is in “rest” (not doing any specific pose).
2. Create a new animation, rename it to “rest”.
3. Select all nodes (box selection should work fine).
4. Select “loc” and “rot” on the top menu.
5. Push the key button. Keys will be inserted for everything, creating a default pose.



11.2.12 Rotation

Animating these models means only modifying the rotation of the nodes. Location and scale are rarely used, with the only exception of moving the entire rig from the hip (which is the root node).

As a result, when inserting keys, only the “rot” button needs to be pressed most of the time:



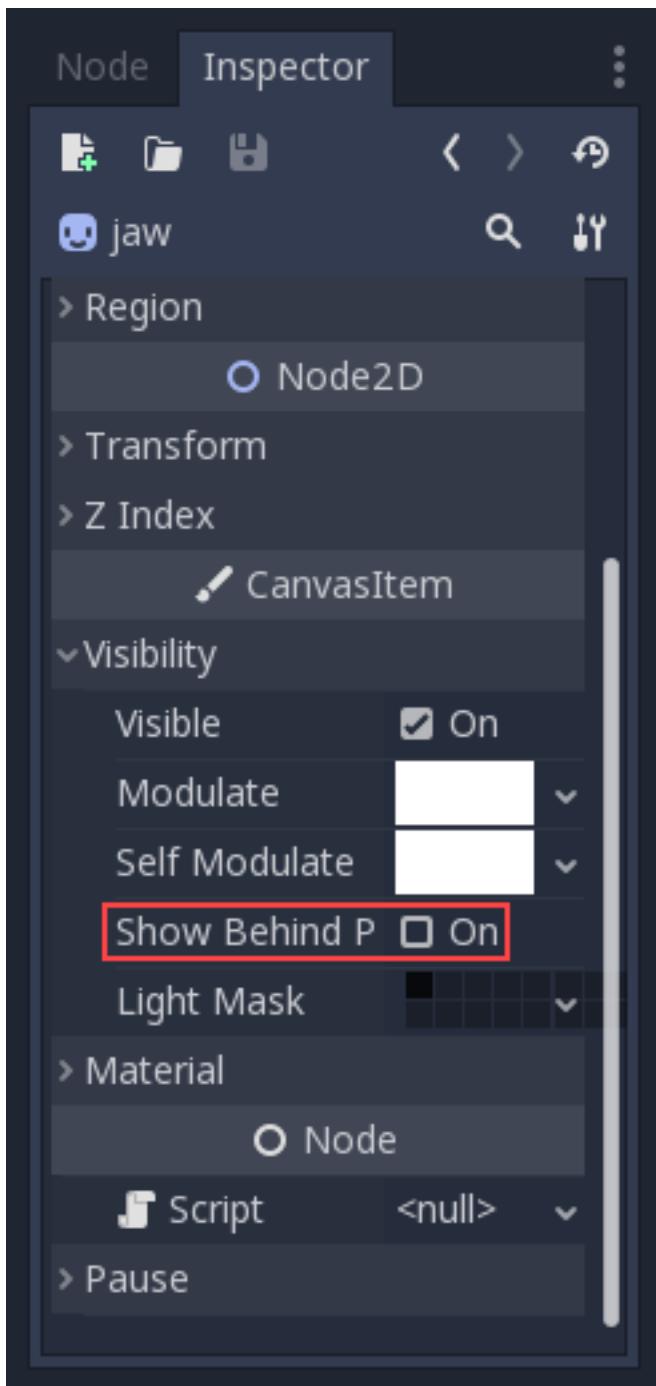
This will avoid the creation of extra animation tracks for the position that will remain unused.

11.2.13 Keyframing IK

When editing IK chains, it is not necessary to select the whole chain to add keyframes. Selecting the endpoint of the chain and inserting a keyframe will automatically insert keyframes until the chain base too. This makes the task of animating extremities much simpler.

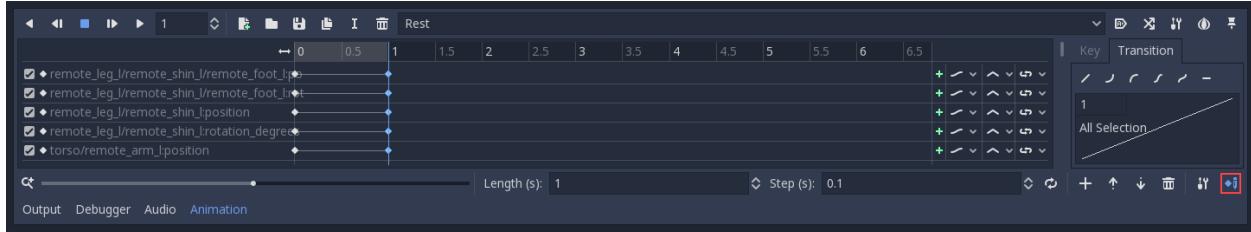
11.2.14 Moving sprites above and behind others.

RemoteTransform2D works in most cases, but sometimes it is really necessary to have a node above and below others during an animation. To aid on this the “Behind Parent” property exists on any Node2D:



11.2.15 Batch setting transition curves

When creating really complex animations and inserting lots of keyframes, editing the individual keyframe curves for each can become an endless task. For this, the Animation Editor has a small menu where changing animation curves is easy. First select the appropriate keys. Next click on the pencil icon in the bottom right of the animation panel, this will open the transition editor. Now click on one of the curve options most appropriate for your animation.



CHAPTER 12

Inputs

12.1 InputEvent

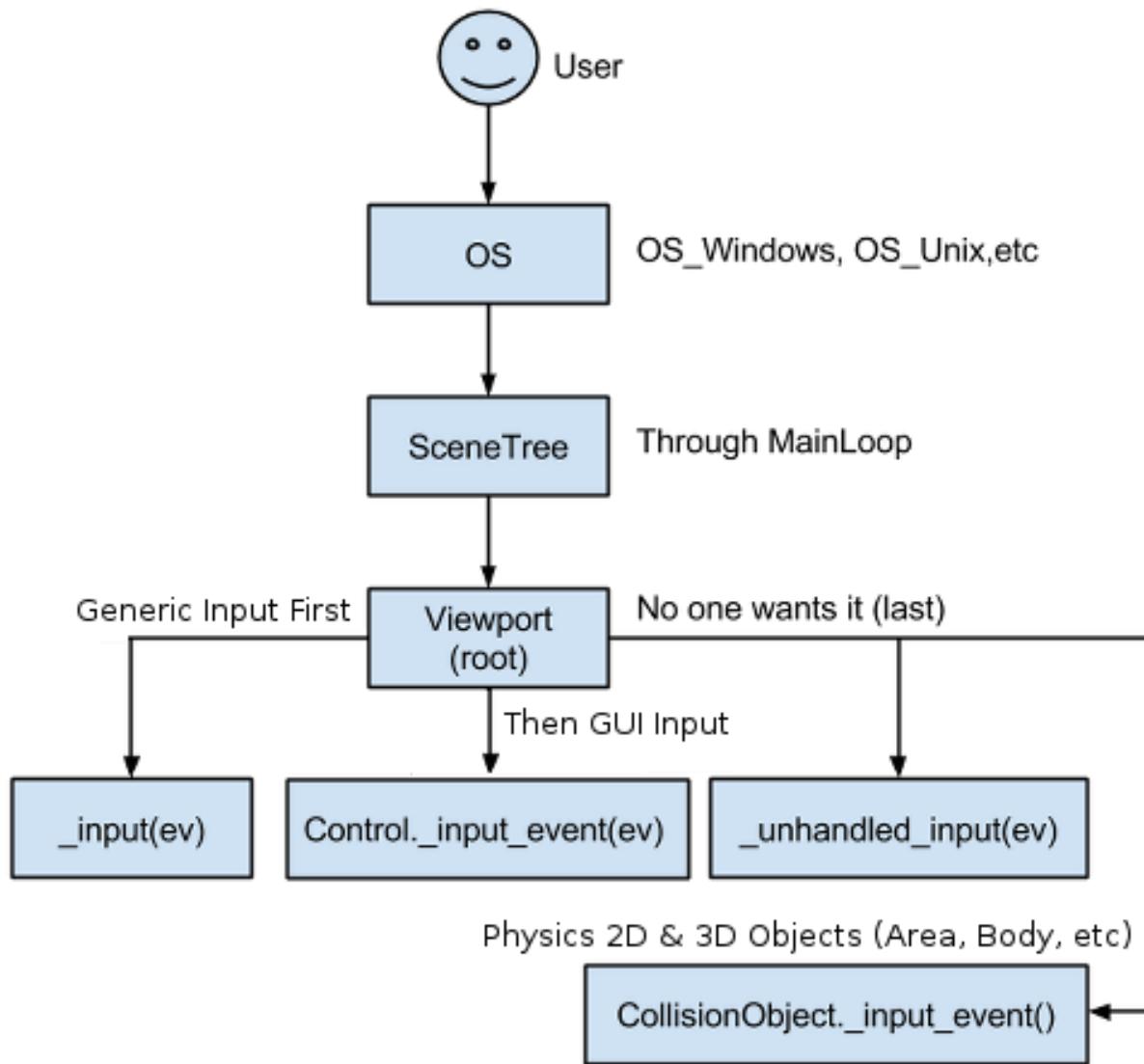
12.1.1 What is it?

Managing input is usually complex, no matter the OS or platform. To ease this a little, a special built-in type is provided, *InputEvent*. This datatype can be configured to contain several types of input events. Input Events travel through the engine and can be received in multiple locations, depending on the purpose.

12.1.2 How does it work?

Every input event is originated from the user/player (though it's possible to generate an *InputEvent* and feed them back to the engine, which is useful for gestures). The OS object for each platform will read events from the device, then feed them to MainLoop. As *SceneTree* is the default MainLoop implementation, events are fed to it. Godot provides a function to get the current *SceneTree* object : `get_tree()`.

But *SceneTree* does not know what to do with the event, so it will give it to the viewports, starting by the “root” *Viewport* (the first node of the scene tree). *Viewport* does quite a lot of stuff with the received input, in order:



1. First of all, the standard `Node._input()` function will be called in any node that overrides it (and hasn't disabled input processing with `Node.set_process_input()`). If any function consumes the event, it can call `SceneTree.set_input_as_handled()`, and the event will not spread any more. This ensures that you can filter all events of interest, even before the GUI. For gameplay input, `Node._unhandled_input()` is generally a better fit, because it allows the GUI to intercept the events.
2. Second, it will try to feed the input to the GUI, and see if any control can receive it. If so, the `Control` will be called via the virtual function `Control._gui_input()` and the signal “`input_event`” will be emitted (this function is re-implementable by script by inheriting from it). If the control wants to “consume” the event, it will call `Control.accept_event()` and the event will not spread any more. Events that are not consumed will propagate **up**, to `Control`'s ancestors. Use `Control.mouse_filter` property to control whether a `Control` is notified of mouse events via `Control._gui_input()` callback, and whether these events are propagated further.
3. If so far no one consumed the event, the unhandled input callback will be called if overridden (and not disabled with `Node.set_processUnhandledInput()`). If any function consumes the event, it can call `SceneTree.set_input_as_handled()`, and the event will not spread any more. The unhandled input callback is ideal for full-screen gameplay events, so they are not received when a GUI is active.

4. If no one wanted the event so far, and a *Camera* is assigned to the Viewport, a ray to the physics world (in the ray direction from the click) will be cast. If this ray hits an object, it will call the *CollisionObject._input_event()* function in the relevant physics object (bodies receive this callback by default, but areas do not). This can be configured through *Area* properties).
5. Finally, if the event was unhandled, it will be passed to the next Viewport in the tree, otherwise it will be ignored.

12.1.3 Anatomy of an InputEvent

InputEvent is just a base built-in type, it does not represent anything and only contains some basic information, such as event ID (which is increased for each event), device index, etc.

InputEvent has a “type” member. By assigning it, it can become different types of input event. Every type of *InputEvent* has different properties, according to its role.

Example of changing event type.

```
# create event
var ev = InputEventMouseButton.new()
# button_index is only available for the above type
ev.button_index = BUTTON_LEFT
```

There are several types of *InputEvent*, described in the table below:

Event	Type Index	Description
<i>InputEvent</i>	NONE	Empty Input Event.
<i>InputEventKey</i>	KEY	Contains a scancode and unicode value, as well as modifiers.
<i>InputEvent-MouseButton</i>	MOUSE_BUTTON	Contains click information, such as button, modifiers, etc.
<i>InputEvent-MouseMotion</i>	MOUSE_MOTION	Contains motion information, such as relative, absolute positions and speed.
<i>InputEventJoy-padMotion</i>	JOY-STICK_MOTION	Contains Joystick/Joypad analog axis information.
<i>InputEventJoy-padButton</i>	JOY-STICK_BUTTON	Contains Joystick/Joypad button information.
<i>InputEventScreen-Touch</i>	SCREEN_TOUCH	Contains multi-touch press/release information. (only available on mobile devices)
<i>InputEventScreen-Drag</i>	SCREEN_DRAG	Contains multi-touch drag information. (only available on mobile devices)
<i>InputEventAction</i>	SCREEN_ACTION	Contains a generic action. These events are often generated by the programmer as feedback. (more on this below)

12.1.4 Actions

An *InputEvent* may or may not represent a pre-defined action. Actions are useful because they abstract the input device when programming the game logic. This allows for:

- The same code to work on different devices with different inputs (e.g., keyboard on PC, Joypad on console).
- Input to be reconfigured at run-time.

Actions can be created from the Project Settings menu in the Actions tab. Read [doc_simple_2d_game-input_actions_setup](#) for an explanation on how the action editor works.

Any event has the methods `InputEvent.is_action()`, `InputEvent.is_pressed()` and `InputEvent`.

Alternatively, it may be desired to supply the game back with an action from the game code (a good example of this is detecting gestures). The Input singleton has a method for this: `Input.parse_input_event()`. You would normally use it like this:

```
var ev = InputEventAction.new()
# set as move_left, pressed
ev.set_as_action("move_left", true)
# feedback
Input.parse_input_event(ev)
```

12.1.5 InputMap

Customizing and re-mapping input from code is often desired. If your whole workflow depends on actions, the `InputMap` singleton is ideal for reassigning or creating different actions at run-time. This singleton is not saved (must be modified manually) and its state is run from the project settings (`project.godot`). So any dynamic system of this type needs to store settings in the way the programmer best sees fit.

12.2 Mouse and input coordinates

12.2.1 About

The reason for this small tutorial is to clear up many common mistakes about input coordinates, obtaining mouse position and screen resolution, etc.

12.2.2 Hardware display coordinates

Using hardware coordinates makes sense in the case of writing complex UIs meant to run on PC, such as editors, MMOs, tools, etc. Yet, it does not make as much sense outside of that scope.

12.2.3 Viewport display coordinates

Godot uses viewports to display content, and viewports can be scaled by several options (see *Multiple resolutions* tutorial). Use, then, the functions in nodes to obtain the mouse coordinates and viewport size, for example:

```
func _input(ev):
    # Mouse in viewport coordinates

    if (ev is InputEventMouseButton):
        print("Mouse Click/Unclick at: ", ev.position)
    elif (ev is InputEventMotion):
        print("Mouse Motion at: ", ev.position)

    # Print the size of the viewport

    print("Viewport Resolution is: ", get_viewport_rect().size)

func _ready():
    set_process_input(true)
```

Alternatively it's possible to ask the viewport for the mouse position:

```
get_viewport().get_mouse_position()
```


CHAPTER 13

GUI

13.1 GUI skinning

13.1.1 Oh beautiful GUI!

This tutorial is about advanced skinning of an user interface. Most games generally don't need this, as they end up just relying on *Label*, *TextureRect*, *TextureButton* and *TextureProgress*.

However, many types of games often need complex user interfaces, like MMOs, traditional RPGs, Simulators, Strategy, etc. These kind of interfaces are also common in some games that include editors to create content, or interfaces for network connectivity.

Godot user interface uses these kind of controls with the default theme, but they can be skinned to resemble pretty much any kind of user interface.

13.1.2 Theme

The GUI is skinned through the *Theme* resource. Theme contains all the information required to change the entire visual styling of all controls. Theme options are named, so it's not obvious which name changes what (specially from code), but several tools are provided. The ultimate place to look at what each theme option is for each control, which will always be more up to date than any documentation is the file [scene/resources/default_theme/default_theme.cpp](#). The rest of this document will explain the different tools used to customize the theme.

A Theme can be applied to any control in the scene. As a result, all children and grand-children controls will use that same theme too (unless another theme is specified further down the tree). If a value is not found in a theme, it will be searched in themes higher up in the hierarchy towards the root. If nothing was found, the default theme is used. This system allows for flexible overriding of themes in complex user interfaces.

13.1.3 Theme options

Each kind of option in a theme can be:

- **A integer constant:** A single numerical constant. Generally used to define spacing between components or alignment.
- **A Color:** A single color, with or without transparency. Colors are usually applied to fonts and icons.
- **A Texture:** A single image. Textures are not often used, but when they are they represent handles to pick or icons in a complex control (such as file dialog).
- **A Font:** Every control that uses text can be assigned the fonts used to draw strings.
- **A StyleBox:** Stylebox is a resource that defines how to draw a panel in varying sizes (more information on them later).

Every option is associated to:

- A name (the name of the option)
- A Control (the name of the control)

An example usage:

```
var t = Theme.new()
t.set_color("font_color", "Label", Color(1.0, 1.0, 1.0))

var l = Label.new()
l.set_theme(t)
```

In the example above, a new theme is created. The “font_color” option is changed and then applied to a label. As a result, the label (and all children and grand children labels) will use that color.

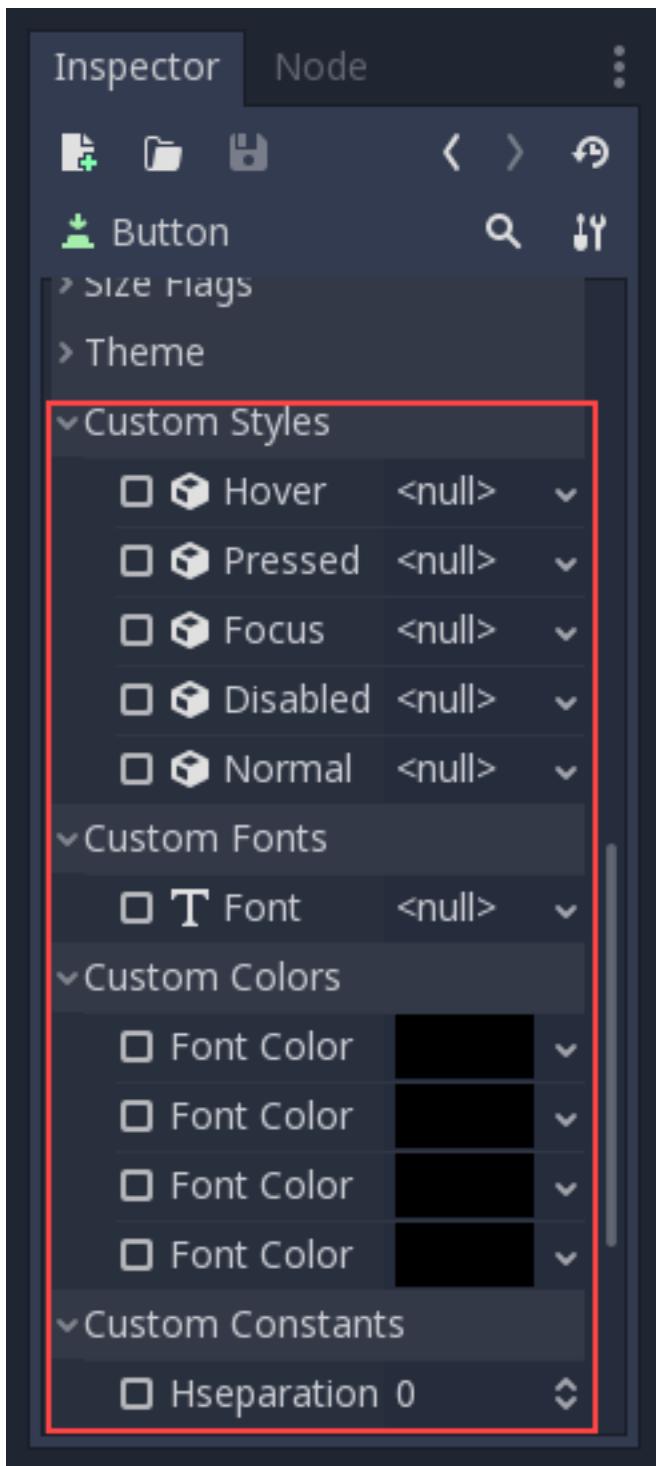
It is possible to override those options without using the theme directly and only for a specific control by using the override API in *Control.add_color_override()*:

```
var l = Label.new()
l.add_color_override("font_color", Color(1.0, 1.0, 1.0))
```

In the inline help of Godot (in the script tab) you can check which theme options are overrideable, or check the *Control* class reference.

13.1.4 Customizing a control

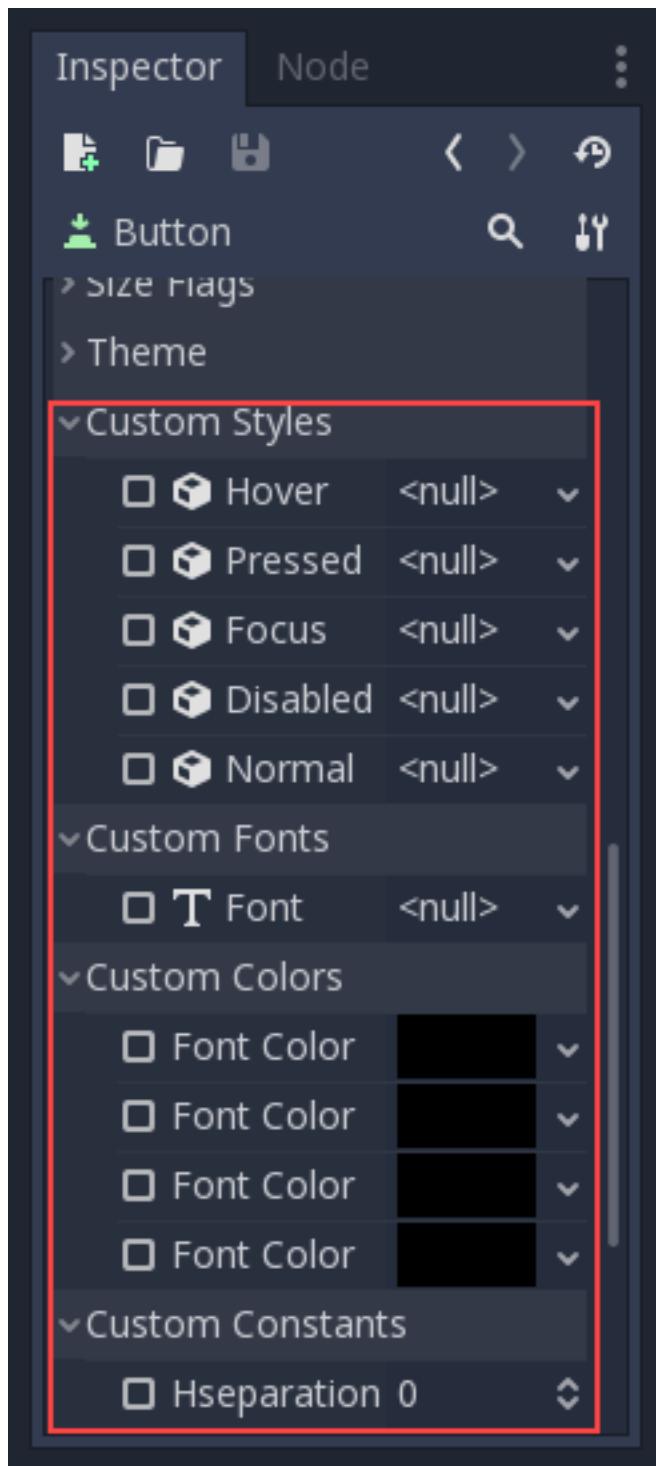
If only a few controls need to be skinned, it is often not necessary to create a new theme. Controls offer their theme options as special kinds of properties. If checked, overriding will take place:



As can be seen in the image above, theme options have little check-boxes. If checked, they can be used to override the value of the theme just for that control.

13.1.5 Creating a theme

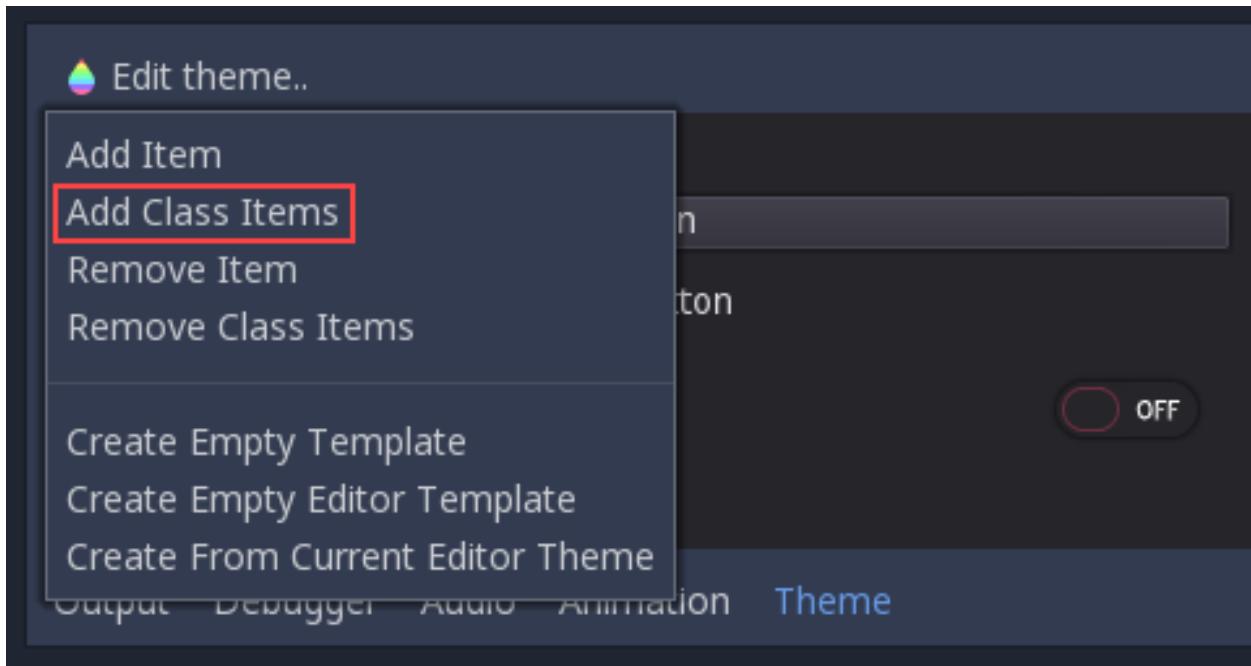
The simplest way to create a theme is to edit a theme resource. Create a Theme from the resource menu, the editor will appear immediately. Following to this, save it (to, as example, mytheme.thm):



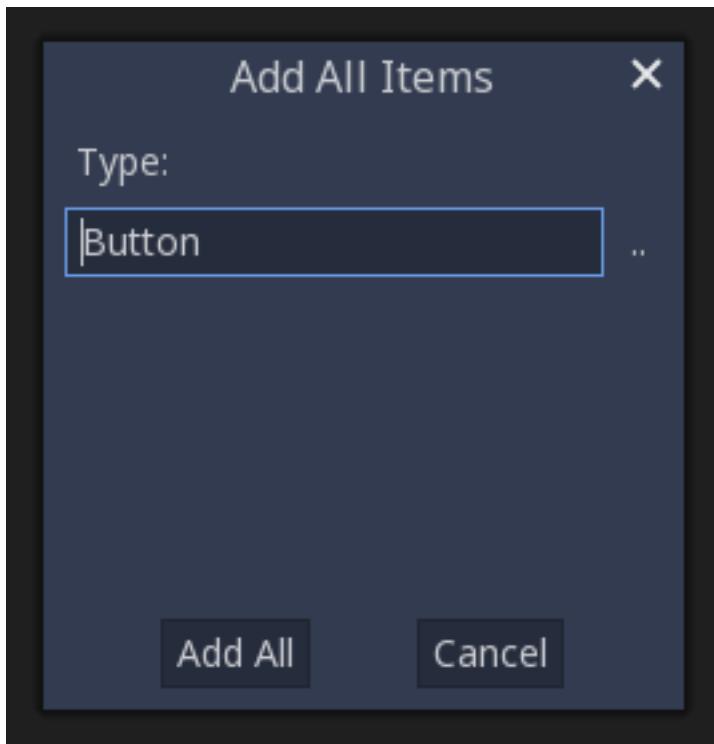
This will create an empty theme that can later be loaded and assigned to controls.

13.1.6 Example: theming a button

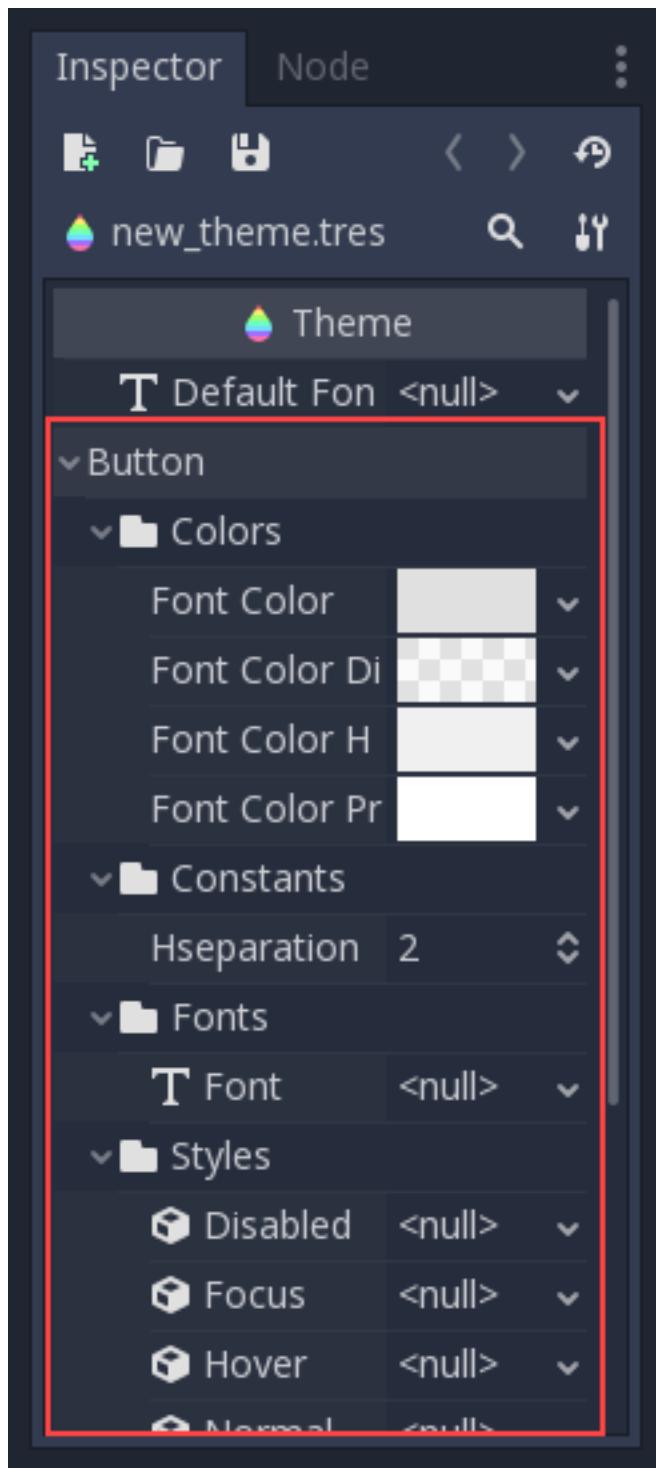
Take some assets (`skin_assets.zip`), go to the “theme” menu and select “Add Class Item”:



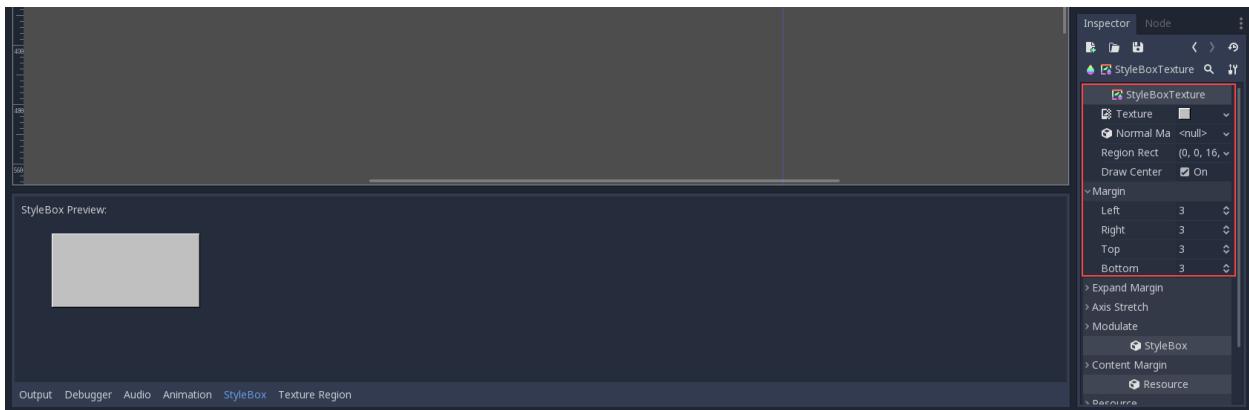
A menu will appear prompting the type of control to create. Select “Button”:



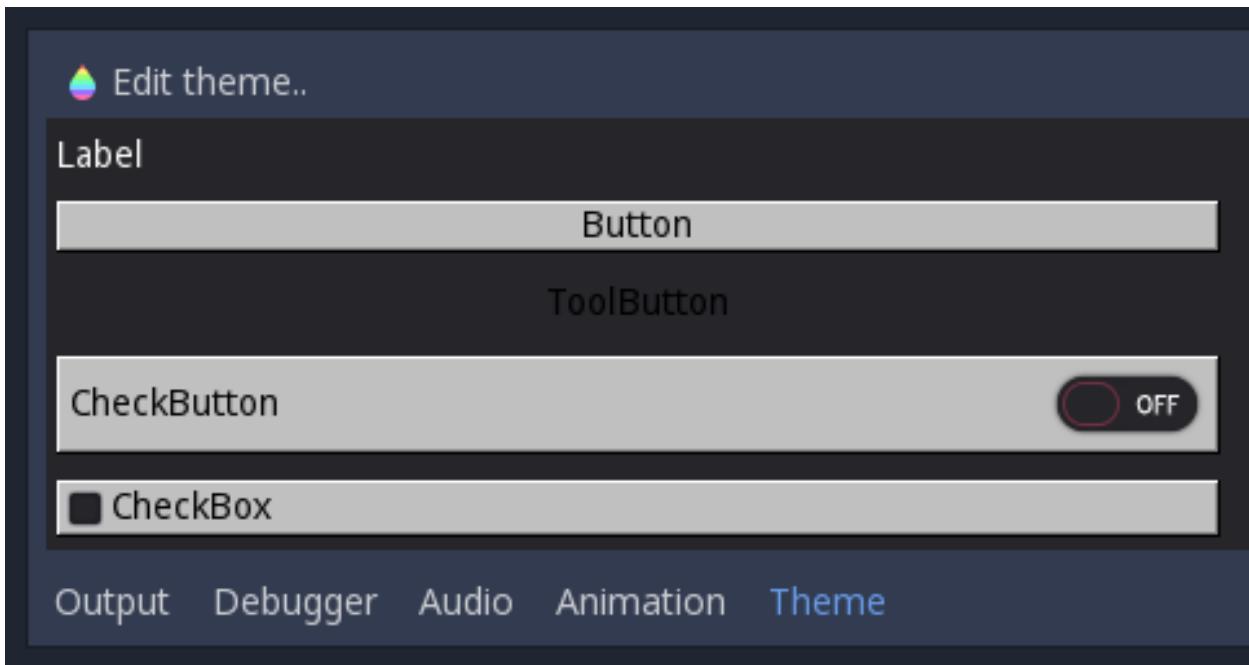
Immediately, all button theme options will appear in the property editor, where they can be edited:



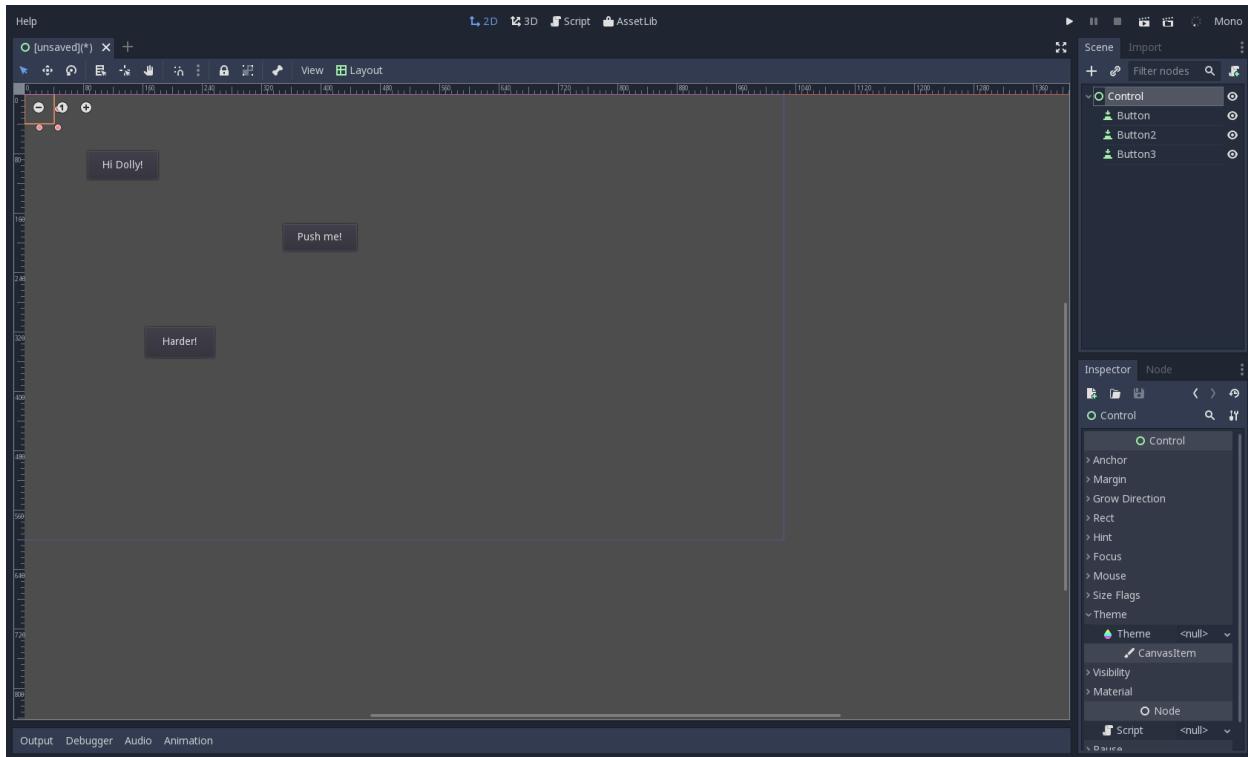
Select the “normal” stylebox and create a new “StyleBoxTexture”, then edit it. A texture stylebox basically contains a texture and the size of the margins that will not stretch when the texture is stretched. This is called “3x3” stretching:



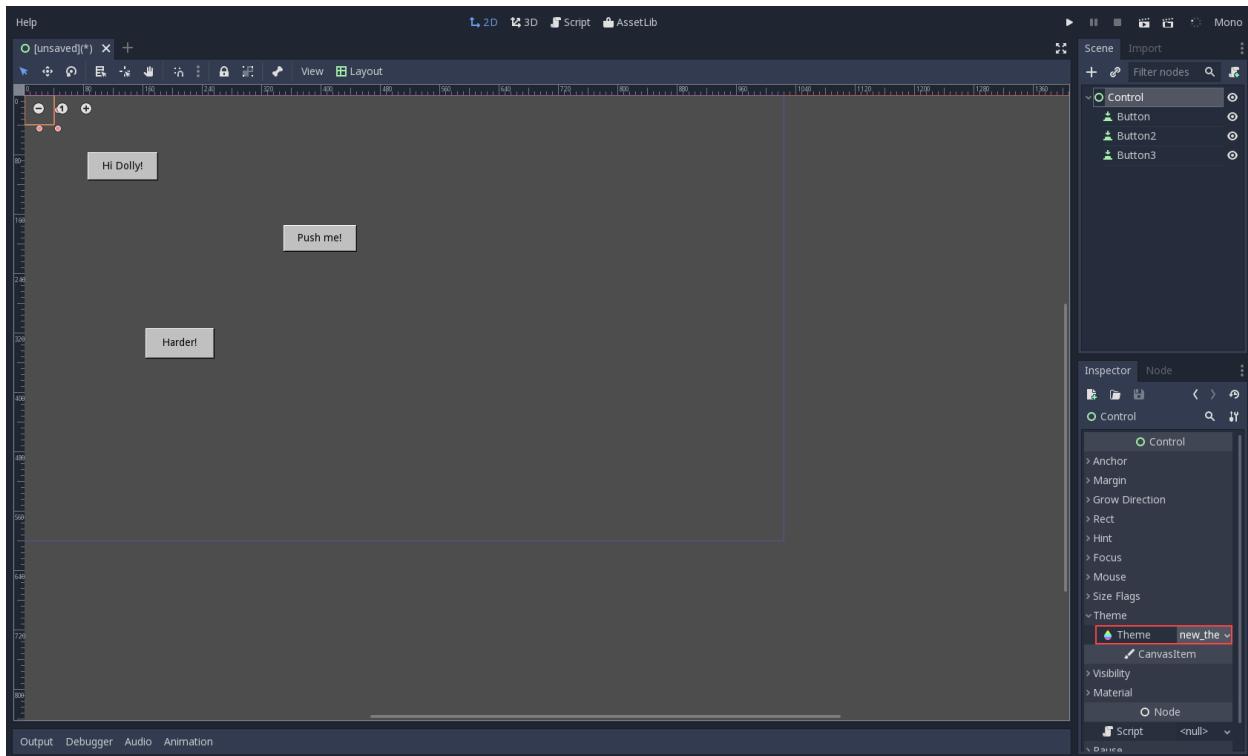
Repeat the steps and add the other assets. There is no hover or disabled image in the example files, so use the same stylebox as in normal. Set the supplied font as the button font and change the font color to black. Soon, your button will look different and retro:



Save this theme to the .thm file. Go to the 2D editor and create a few buttons:



Now, go to the root node of the scene and locate the “theme” property, replace it by the theme that was just created. It should look like this:



Congratulations! You have created a reusable GUI Theme!

13.2 Custom GUI controls

13.2.1 So many controls...

Yet there are never enough. Creating your own custom controls that act just the way you want them is an obsession of almost every GUI programmer. Godot provides plenty of them, but they may not work exactly the way you want. Before contacting the developers with a pull-request to support diagonal scrollbars, at least it will be good to know how to create these controls easily from script.

13.2.2 Drawing

For drawing, it is recommended to check the *Custom drawing in 2D* tutorial. The same applies. Some functions are worth mentioning due to their usefulness when drawing, so they will be detailed next:

Checking control size

Unlike 2D nodes, “size” is very important with controls, as it helps to organize them in proper layouts. For this, the *Control.rect_size* member variable is provided. Checking it during *_draw()* is vital to ensure everything is kept in-bounds.

Checking focus

Some controls (such as buttons or text editors) might provide input focus for keyboard or joypad input. Examples of this are entering text or pressing a button. This is controlled with the *Control.focus_mode* member variable. When drawing, and if the control supports input focus, it is always desired to show some sort of indicator (highlight, box, etc) to indicate that this is the currently focused control. To check for this status, the *Control.has_focus()* method exists. Example

```
func _draw():
    if (has_focus()):
        draw_selected()
    else:
        draw_normal()
```

13.2.3 Sizing

As mentioned before, size is very important to controls. This allows them to lay out properly, when set into grids, containers, or anchored. Controls most of the time provide a *minimum size* to help to properly lay them out. For example, if controls are placed vertically on top of each other using a *VBoxContainer*, the minimum size will make sure your custom control is not squished by the other controls in the container.

To provide this callback, just override *Control.get_minimum_size()*, for example:

```
func get_minimum_size():
    return Vector2(30, 30)
```

Or alternatively, set it via function:

```
func _ready():
    set_custom_minimum_size( Vector2(30, 30) )
```

13.2.4 Input

Controls provide a few helpers to make managing input events much easier than regular nodes.

Input events

There are a few tutorials about input before this one, but it's worth mentioning that controls have a special input method that only works when:

- The mouse pointer is over the control.
- The button was pressed over this control (control always captures input until button is released)
- Control provides keyboard/joypad focus via *Control.focus_mode*.

This function is *Control._gui_input()*. Simply override it in your control. No processing needs to be set.

```
extends Control

func _gui_input(ev):
    if (ev is InputEventMouseButton and ev.button_index==BUTTON_LEFT and ev.pressed):
        print("Left mouse button was pressed!")
```

For more information about events themselves, check the *InputEvent* tutorial.

Notifications

Controls also have many useful notifications for which no callback exists, but can be checked with the *_notification* callback:

```
func _notification(what):

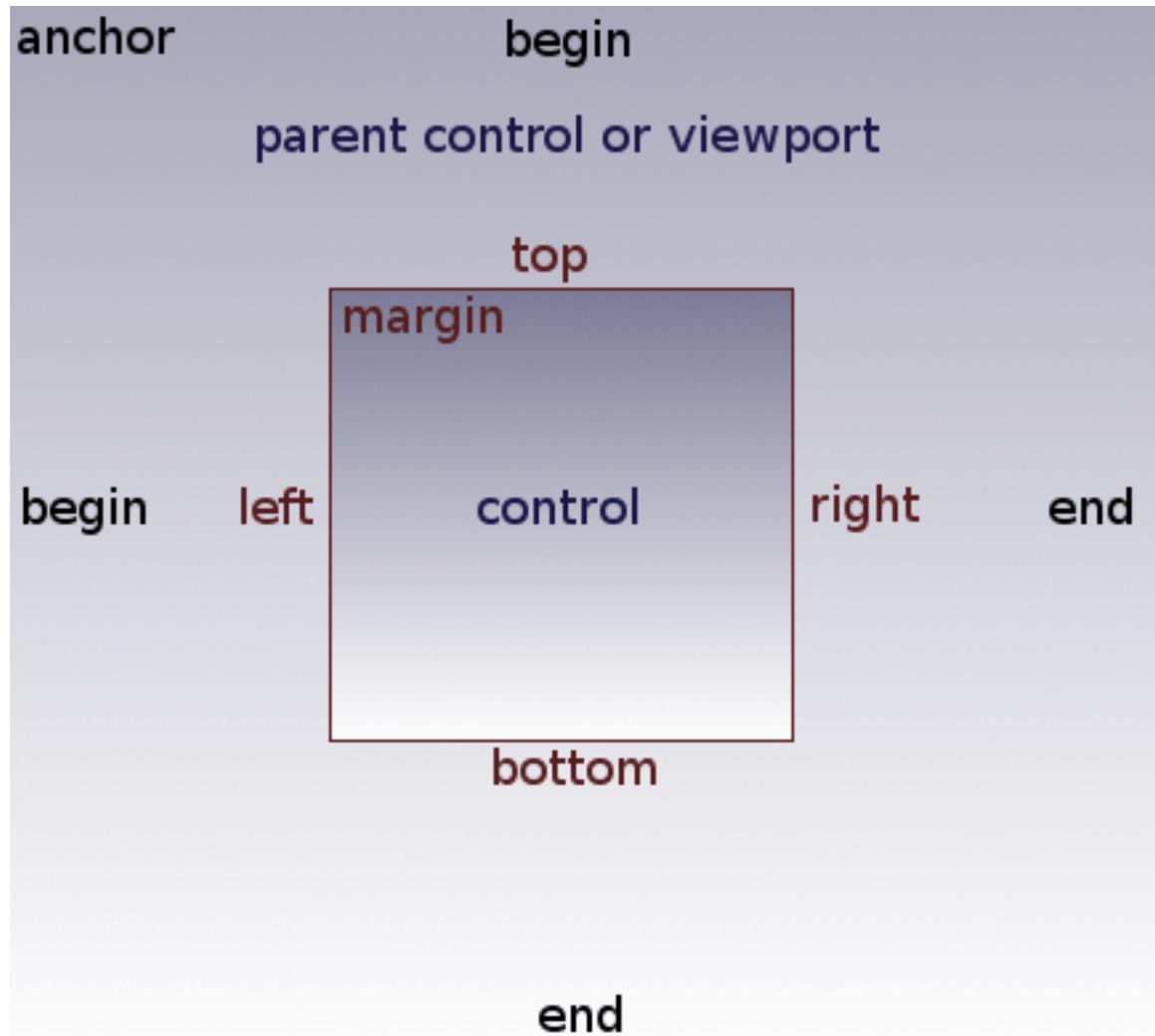
    if (what==NOTIFICATION_MOUSE_ENTER):
        pass # mouse entered the area of this control
    elif (what==NOTIFICATION_MOUSE_EXIT):
        pass # mouse exited the area of this control
    elif (what==NOTIFICATION_FOCUS_ENTER):
        pass # control gained focus
    elif (what==NOTIFICATION_FOCUS_EXIT):
        pass # control lost focus
    elif (what==NOTIFICATION_THEME_CHANGED):
        pass # theme used to draw the control changed
        # update and redraw is recommended if using a theme
    elif (what==NOTIFICATION_VISIBILITY_CHANGED):
        pass # control became visible/invisible
        # check new status with is_visible()
    elif (what==NOTIFICATION_RESIZED):
        pass # control changed size, check new size
        # with get_size()
    elif (what==NOTIFICATION_MODAL_CLOSED):
        pass # for modal popups, notification
        # that the popup was closed
```

13.3 Size and anchors

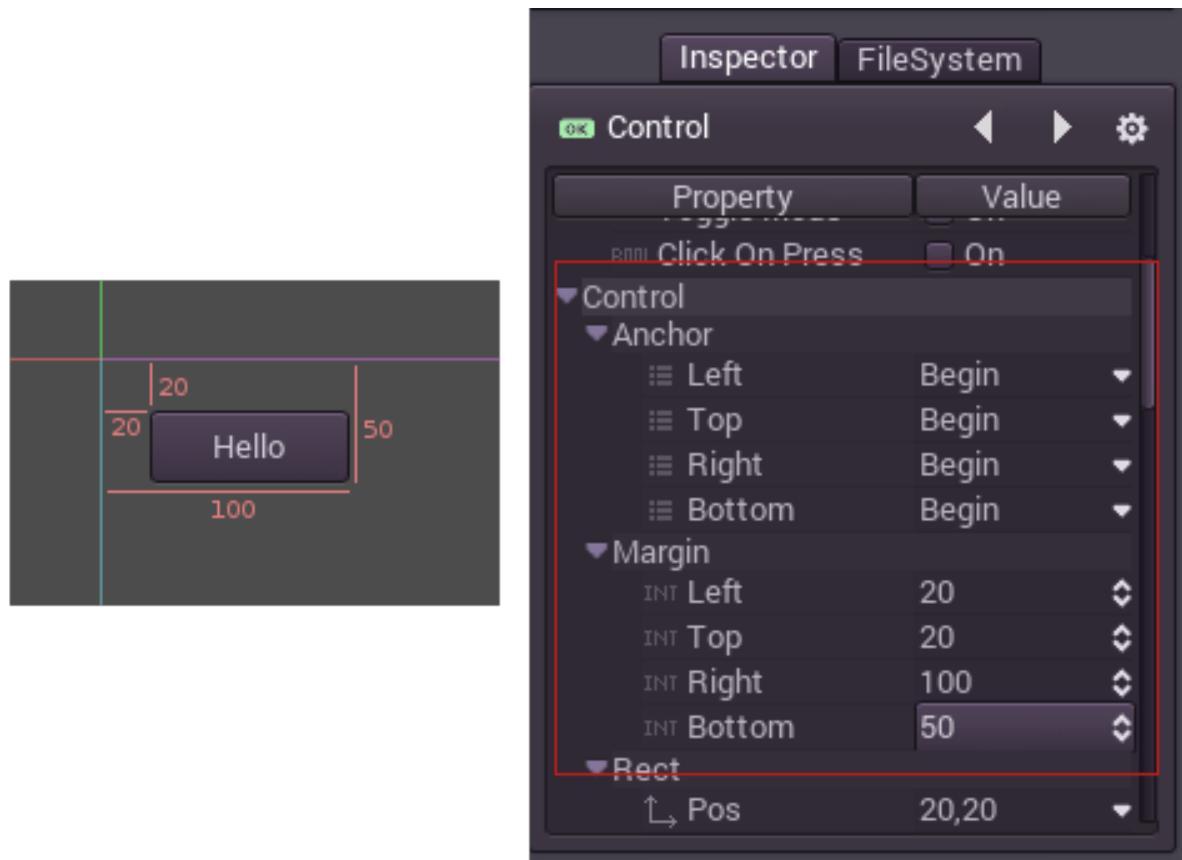
If a game was always going to be run on the same device and at the same resolution, positioning controls would be a simple matter of setting the position and size of each one of them. Unfortunately, that is rarely the case.

Only TVs nowadays have a standard resolution and aspect ratio. Everything else, from computer monitors to tablets, portable consoles and mobile phones have different resolutions and aspect ratios.

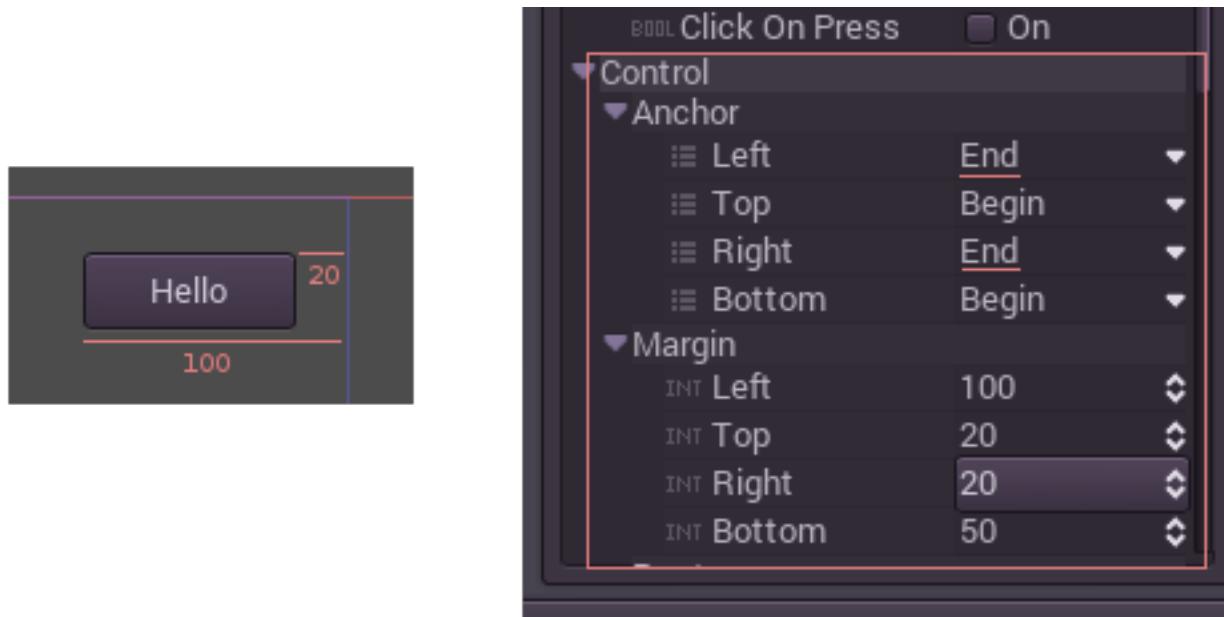
There are several ways to handle this, but for now let's just imagine that the screen resolution has changed and the controls need to be re-positioned. Some will need to follow the bottom of the screen, others the top of the screen, or maybe the right or left margins.



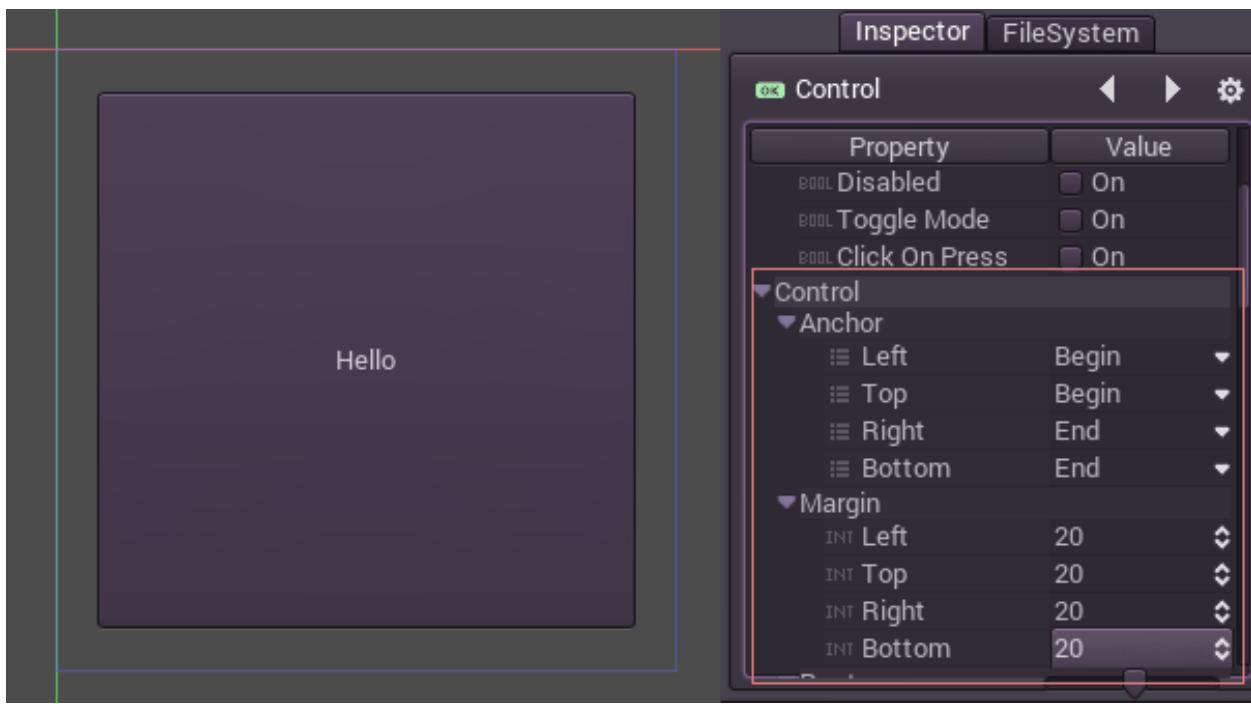
This is done by editing the *margin* properties of controls. Each control has four margins: left, right, bottom and top. By default all of them represent a distance in pixels relative to the top-left corner of the parent control or (in case there is no parent control) the viewport.



When horizontal (left,right) and/or vertical (top,bottom) anchors are changed to END, the margin values become relative to the bottom-right corner of the parent control or viewport.



Here the control is set to expand its bottom-right corner with that of the parent, so when re-sizing the parent, the control will always cover it, leaving a 20 pixel margin:



13.4 BBCode in RichTextLabel

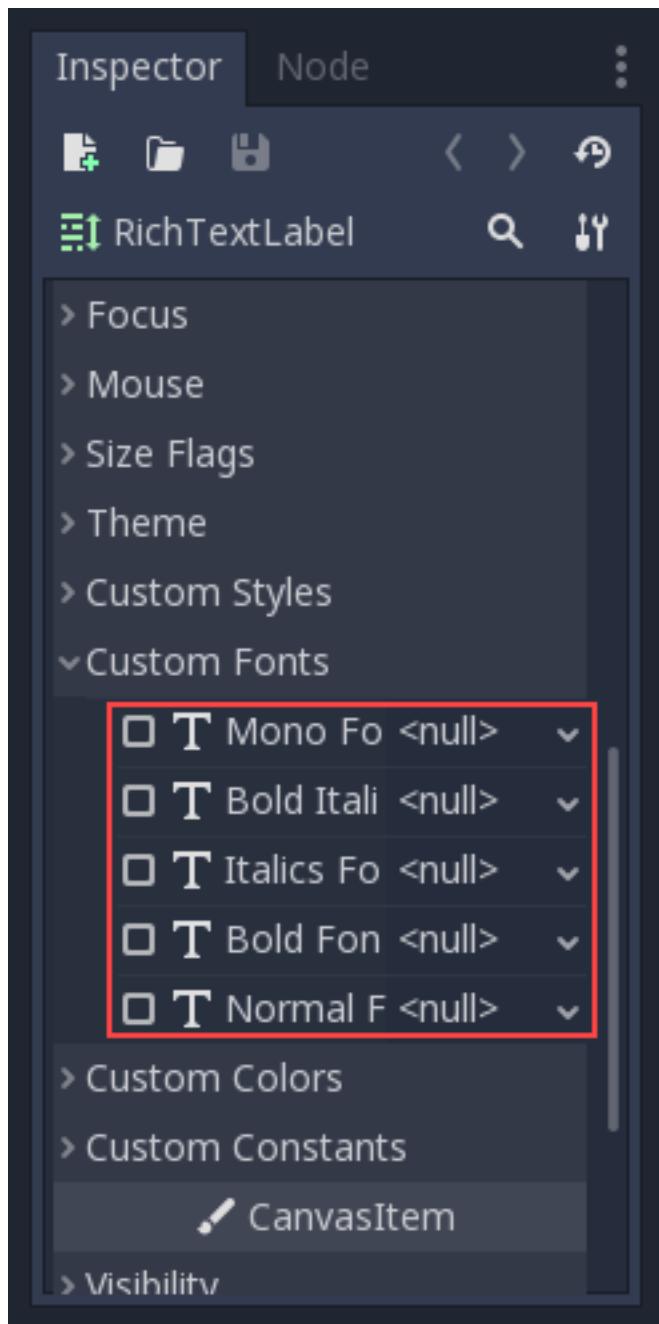
13.4.1 Introduction

RichTextLabel allows to display complex text markup in a control. It has a built-in API for generating the markup, but can also parse a BBCode.

Note that the BBCode tags can also be used to some extent in the *XML source of the class reference*.

13.4.2 Setting up

For *RichTextLabel* to work properly, it must be set up. This means loading the intended fonts in the relevant properties:



13.4.3 Reference

Com-mand	Tag	Description
bold	[b]{text}[/b]	Makes {text} bold.
italics	[i]{text}[/i]	Makes {text} italics.
under-line	[u]{text}[/u]	Makes {text} underline.
code	[code]{text}[/code]	Makes {text} monospace.
center	[center]{text}[/center]	Makes {text} centered.
right	[right]{text}[/right]	Makes {text} right-aligned.
fill	[fill]{text}[/fill]	Makes {text} fill width.
indent	[indent]{text}[/indent]	Increase indent level of {text}.
url	[url]{url}[/url]	Show {url} as such.
url (ref)	[url=<url>]{text}[/url]	Makes {text} reference <url>.
image	[img]{path}[/img]	Insert image at resource {path}.
font	[font=<path>]{text}[/font]	Use custom font at <path> for {text}.
color	[color=<code/name>]{text}[/color]	Change {text} color, use # format such as #ff00ff or name.

Built-in color names

List of valid color names for the [color=<name>] tag:

- aqua
- black
- blue
- fuchsia
- gray
- green
- lime
- maroon
- navy
- purple
- red
- silver
- teal
- white
- yellow

Hexadecimal color codes

For opaque RGB colors, any valid 6-digit hexadecimal code is supported, e.g. [color=#ffffff]white[/color].

For transparent RGB colors, any 8-digit hexadecimal code can be used, e.g. [color=#88ffffff]translucent white[/color]. In this case, note that the alpha channel is the **first** component of the color code, not the last one.

CHAPTER 14

Viewports

14.1 Viewports

14.1.1 Introduction

Godot has a small but very useful feature called viewports. Viewports are, as they name implies, rectangles where the world is drawn. They have three main uses, but can flexibly adapted to a lot more. All this is done via the *Viewport* node.



The main uses in question are:

- **Scene Root:** The root of the active scene is always a Viewport. This is what displays the scenes created by the user. (You should know this by having read previous tutorials!)
- **Sub-Viewports:** These can be created when a Viewport is a child of a *Control*.
- **Render Targets:** Viewports can be set to “RenderTarget” mode. This means that the viewport is not directly visible, but its contents can be accessed via a *Texture*.

14.1.2 Input

Viewports are also responsible of delivering properly adjusted and scaled input events to all its children nodes. Both the root viewport and sub-viewports do this automatically, but render targets do not. Because of this, the user must do it manually via the *Viewport.input()* function if needed.

14.1.3 Listener

Godot supports 3D sound (in both 2D and 3D nodes), more on this can be found in another tutorial (one day..). For this type of sound to be audible, the viewport needs to be enabled as a listener (for 2D or 3D). If you are using a custom viewport to display your world, don't forget to enable this!

14.1.4 Cameras (2D & 3D)

When using a 2D or 3D *Camera* / *Camera2D*, cameras will always display on the closest parent viewport (going towards the root). For example, in the following hierarchy:

- Viewport
 - Camera

Camera will display on the parent viewport, but in the following one:

- Camera
 - Viewport

It will not (or may display in the root viewport if this is a subscene).

There can be only one active camera per viewport, so if there is more than one, make sure that the desired one has the “current” property set, or make it the current camera by calling:

```
camera.make_current()
```

14.1.5 Scale & stretching

Viewports have a “rect” property. X and Y are often not used (only the root viewport really uses them), while WIDTH AND HEIGHT represent the size of the viewport in pixels. For Sub-Viewports, these values are overridden by the ones from the parent control, but for render targets this sets their resolution.

It is also possible to scale the 2D content and make it believe the viewport resolution is other than the one specified in the rect, by calling:

```
viewport.set_size_override(w,h) #custom size for 2D  
viewport.set_size_override_stretch(true/false) #enable stretch for custom size
```

The root viewport uses this for the stretch options in the project settings.

14.1.6 Worlds

For 3D, a Viewport will contain a *World*. This is basically the universe that links physics and rendering together. Spatial-base nodes will register using the World of the closest viewport. By default, newly created viewports do not contain a World but use the same as a parent viewport (root viewport does contain one though, which is the one objects are rendered to by default). A world can be set in a viewport using the “world” property, and that will separate all children nodes of that viewport from interacting with the parent viewport world. This is especially useful in scenarios where, for example, you might want to show a separate character in 3D imposed over the game (like in Starcraft).

As a helper for situations where you want to create viewports that display single objects and don't want to create a world, viewport has the option to use its own World. This is very useful when you want to instance 3D characters or objects in the 2D world.

For 2D, each Viewport always contains its own *World2D*. This suffices in most cases, but in case sharing them may be desired, it is possible to do so by calling the viewport API manually.

14.1.7 Capture

It is possible to query a capture of the viewport contents. For the root viewport this is effectively a screen capture. This is done with the following API:

```
# queues a screen capture, will not happen immediately
viewport.queue_screen_capture()
```

After a frame or two (check `_process()`), the capture will be ready, get it back by using:

```
var capture = viewport.get_screen_capture()
```

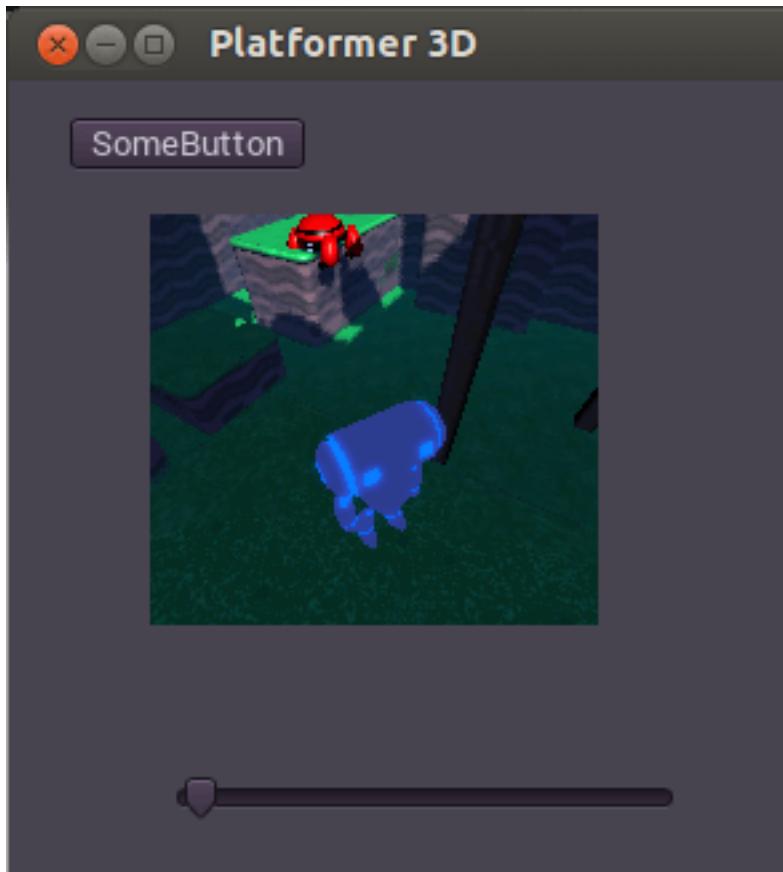
If the returned image is empty, capture still didn't happen, wait a little more, as this API is asynchronous.

14.1.8 Sub-viewport

If the viewport is a child of a control, it will become active and display anything it has inside. The layout is something like this:

- Control
 - Viewport

The viewport will cover the area of its parent control completely.



14.1.9 Render target

To set as a render target, just toggle the “render target” property of the viewport to enabled. Note that whatever is inside will not be visible in the scene editor. To display the contents, the render target texture must be used. This can be requested via code using (for example):

```
var rtt = viewport.get_render_target_texture()
sprite.set_texture(rtt)
```

By default, re-rendering of the render target happens when the render target texture has been drawn in a frame. If visible, it will be rendered, otherwise it will not. This behavior can be changed to manual rendering (once), or always render, no matter if visible or not.

A few classes are created to make this easier in most common cases inside the editor:

- ViewportSprite (for 2D).
- ViewportQuad (for 3D).
- ViewportFrame (for GUI).

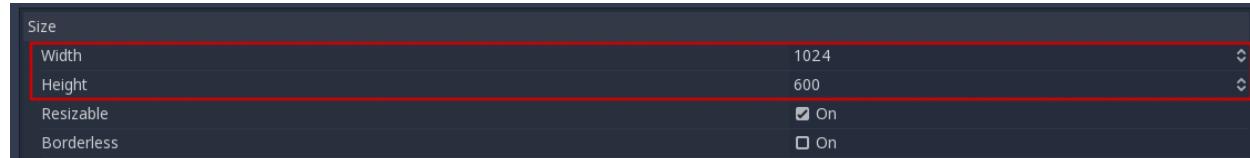
TODO: Review the doc, ViewportQuad and ViewportFrame don't exist in 2.0.

Make sure to check the viewport demos! Viewport folder in the demos archive available to download, or <https://github.com/godotengine/godot-demo-projects/tree/master/viewport>

14.2 Multiple resolutions

14.2.1 Base resolution

A base screen resolution for the project can be specified in the project settings under `display, window`



However, what it does is not completely obvious. When running on PC, the engine will attempt to set this resolution (or use something smaller if it fails). On mobile, consoles or devices with a fixed resolution or full screen rendering, this resolution will be ignored and the native resolution will be used instead. To compensate for this, Godot offers many ways to control how the screen will resize and stretch to different screen sizes.

14.2.2 Resizing

There are several types of devices, with several types of screens, which in turn have different pixel density and resolutions. Handling all of them can be a lot of work, so Godot tries to make the developer's life a little easier. The `Viewport` node has several functions to handle resizing, and the root node of the scene tree is always a viewport (scenes loaded are instanced as a child of it, and it can always be accessed by calling `get_tree().get_root()` or `get_node("/root")`).

In any case, while changing the root Viewport params is probably the most flexible way to deal with the problem, it can be a lot of work, code and guessing, so Godot provides a simple set of parameters in the project settings to handle multiple resolutions.

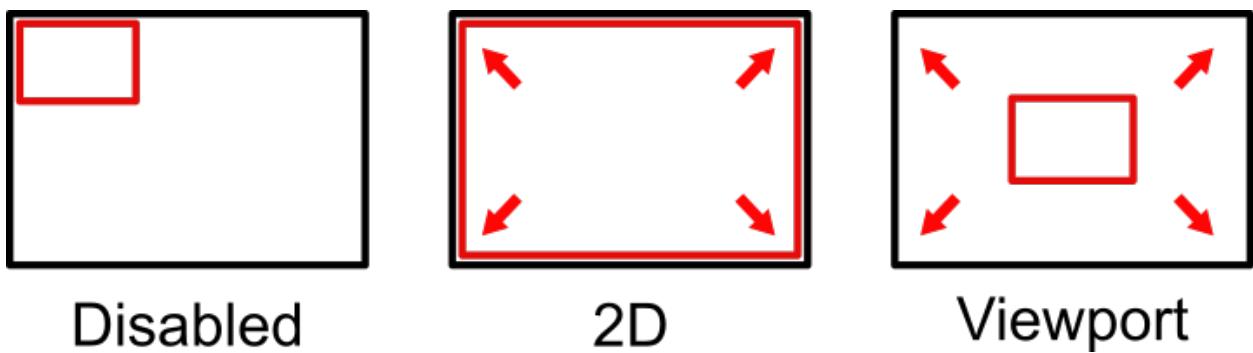
14.2.3 Stretch settings

Stretch settings are located in the project settings, it's just a bunch of configuration variables that provide several options:

Stretch	
Mode	disabled
Aspect	ignore
Shrink	1

14.2.4 Stretch mode

- **Disabled:** The first is the stretch mode. By default this is disabled, which means no stretching happens (the bigger the screen or window, the bigger the resolution, always matching pixels 1:1).
- **2D:** In this mode, the resolution specified in display/width and display/height in the project settings will be stretched to cover the whole screen. This means that 3D will be unaffected (will just render to higher-res) and 2D will also be rendered at higher-res, just enlarged.
- **Viewport:** Viewport scaling is different, the root *Viewport* is set as a render target, and still renders precisely to the resolution specified in the display/ section of the project settings. Finally, this viewport is copied and scaled to fit the screen. This mode is useful when working with pixel-precise games, or just for the sake of rendering to a lower resolution for improving performance.



14.2.5 Stretch aspect

- **Ignore:** Ignore the aspect ratio when stretching the screen. This means that the original resolution will be stretched to fit the new one, even if it's wider or narrower.
- **Keep:** Keep aspect ratio when stretching the screen. This means that the original resolution will be kept when fitting the new one, and black bars will be added to the sides or the top/bottom of the screen.
- **Keep Width:** Keep aspect ratio when stretching the screen, but if the resulting screen is taller than the specified resolution, it will be stretched vertically (and more vertical resolution will be reported in the viewport, proportionally). This is usually the best option for creating GUIs or HUDs that scale, so some controls can be anchored to the bottom (*Size and anchors*).
- **Keep Height:** Keep aspect ratio when stretching the screen, but if the resulting screen is wider than the specified resolution, it will be stretched horizontally (and more horizontal resolution will be reported in the viewport, proportionally). This is usually the best option for 2D games that scroll horizontally (like runners or platformers).

CHAPTER 15

Shading

15.1 Shading language

15.1.1 Introduction

Godot uses a shading language very similar to GLSL ES 3.0. Most datatypes and functions are supported, and the remaining will likely be added over time.

Unlike the shader language in Godot 2.x, this implementation is much closer to the original.

15.1.2 Shader Types

Instead of supplying a general purpose configuration, Godot Shading Language must specify what shader is intended for. Depending on the type, different render modes, built-in variables and processing functions are supported.

Any shader needs a first line specifying this type, in the following format:

```
shader_type <type>;
```

Valid types are:

- “spatial”: For 3D rendering.
- “canvas_item”: For 2D rendering.
- “particles”: For particle systems.

15.1.3 Render Modes

Different shader types support different render modes. They are optional but, if specified, must be after the *shader_type*. Example syntax is:

```
shader_type spatial;
render_mode unshaded, cull_disabled;
```

15.1.4 Data types:

Most GLSL ES 3.0 datatypes are supported. Following is the list:

Type	Description
void	Void datatype, useful only for functions that return nothing.
bool	Boolean datatype, can only contain “true” or “false”
bvec2	Two component vector of booleans.
bvec3	Three component vector of booleans.
bvec4	Four component vector of booleans.
int	Signed scalar integer.
ivec2	Two component vector of signed integers.
ivec3	Three component vector of signed integers.
ivec4	Four component vector of signed integers.
uint	Unsigned scalar integer, can’t contain negative numbers.
uvec2	Two component vector of unsigned integers.
uvec3	Three component vector of unsigned integers.
uvec4	Four component vector of unsigned integers.
float	Floating point scalar.
vec2	Two component vector of floating point values.
vec3	Three component vector of floating point values.
vec4	Four component vector of floating point values.
mat2	2x2 matrix, in column major order.
mat3	3x3 matrix, in column major order.
mat4	4x4 matrix, in column major order.
sampler2D	Sampler type, for binding 2D textures, which are read as float.
isampler2D	Sampler type for binding 2D textures, which are read as signed integer.
usampler2D	Sampler type for binding 2D textures, which are read as unsigned integer.
samplerCube	Sampler type for binding Cubemaps, which are read as floats.

Casting

Just like GLSL ES 3.0, implicit casting is not allowed between scalars and vectors of the same size but different type. Casting of types of different size is also not allowed. Conversion must be done explicitly via constructors.

Example:

```
float a = 2; // valid
float a = 2.0; // valid
float a = float(2); // valid
```

Default integer constants are signed, so casting is always needed to convert to unsigned:

```
int a = 2; // valid
uint a = 2; // invalid
uint a = uint(2); // valid
```

Members

Individual scalar members of vector types are accessed via the “x”, “y”, “z” and “w” members. Alternatively, using “r”, “g”, “b” and “a” also works and is equivalent. Use whatever fits best for your use case.

For matrices, use [idx] indexing syntax to access each vector.

Constructing

Construction of vector types must always pass:

```
// The required amount of scalars
vec4 a = vec4(0.0, 1.0, 2.0, 3.0);
// Complementary vectors and/or scalars
vec4 a = vec4(vec2(0.0, 1.0), vec2(2.0, 3.0));
vec4 a = vec4(vec3(0.0, 1.0, 2.0), 3.0);
// A single scalar for the whole vector
vec4 a = vec4(0.0);
```

Swizzling

It is possible to obtain any combination of them in any order, as long as the result is another vector type (or scalar). This is easier shown than explained:

```
vec4 a = vec4(0.0, 1.0, 2.0, 3.0);
vec3 b = a.rgb; // Creates a vec3 with vec4 components
vec3 b = a.aaa; // Also valid, creates a vec3 and fills it with "a".
vec3 b = a.bgr; // Order does not matter
vec3 b = a.xyz; // Also rgba, xyzw are equivalent
float c = b.w; // Invalid, because "w" is not present in vec3 b
```

Precision

It is possible to add precision modifiers to datatypes, use them for uniforms, variables, arguments and varyings:

```
lowp vec4 a = vec4(0.0, 1.0, 2.0, 3.0); // low precision, usually 8 bits per_
    ↵component mapped to 0-1
mediump vec4 a = vec4(0.0, 1.0, 2.0, 3.0); // medium precision, usually 16 bits or_
    ↵half float
highp vec4 a = vec4(0.0, 1.0, 2.0, 3.0); // high precision, uses full float or_
    ↵integer range (default)
```

Using lower precision for some operations can speed up the math involved (at the cost of, of course, less precision). This is rarely needed in the vertex shader (where full precision is needed most of the time), but often needed in the fragment one.

Keep in mind that some architectures (mainly mobile) benefit a lot from this, but are also restricted (conversion between precisions has a cost). Please read the relevant documentation on the target architecture to find out more. In all honesty though, mobile drivers are really buggy so just stay out of trouble and make simple shaders without specifying precision unless you *really* need to.

15.1.5 Operators:

Godot shading language supports the same set of operators as GLSL ES 3.0. Below is the list of them in precedence order:

Precedence	Class	Operator
1 (highest)	parenthetical grouping	()
2	unary	+,-,!,-~
3	multiplicative	/,*,%
4	additive	+, -
5	bit-wise shift	<<,>>
6	relational	<,>,<=,>=
7	equality	==, !=
8	bit-wise and	&
9	bit-wise exclusive or	^
10	bit-wise inclusive or	
11	logical and	&&
12 (lowest)	logical inclusive or	

15.1.6 Flow Control

Godot Shading language supports the most common types of flow control:

```
// if and else
if (cond) {

} else {

}

// for loops
for      (int i = 0; i < 10; i++) {

}

// while
while  (true) {

}
```

Keep in mind that, in modern GPUs, an infinite loop can exist and can freeze your application (including editor). Godot can't protect you from this, so be careful to not make this mistake!

15.1.7 Discarding

Fragment and light functions can use the *discard* keyword. If used, the fragment is discarded and nothing is written.

15.1.8 Functions

It's possible to define any function in a Godot shader. They take the following syntax:

```
ret_type func_name(args) {
    return ret_type; // if returning a value
}

// a better example:

int sum2(int a, int b) {
    return a + b;
}
```

Functions can be used from any other function that is below it.

Function argument can have special qualifiers:

- **in**: Means the argument is only for reading (default).
- **out**: Means the argument is only for writing.
- **inout**: Means the argument is fully passed via reference.

Example below:

```
void sum2(int a, int b, inout int result) {
    result = a + b;
}
```

15.1.9 Processor Functions

Depending on shader type, processor functions may be available to optionally override. For “spatial” and “canvas_item”, it is possible to override “vertex”, “fragment” and “light”. For “particles”, only “vertex” can be overridden.

Vertex Processor

The “vertex” processing function is called for every vertex, 2D or 3D. For particles, it’s called for every particle.

Depending on shader type, a different set of built-in inputs and outputs are provided. In general, vertex functions are not that commonly used.

```
shader_type spatial;

void vertex() {
    VERTEX.x += sin(TIME); // offset vertex x by sine function on time elapsed
}
```

Fragment Processor

The “fragment” processor is used to set up the Godot material parameters per pixel. This code runs on every visible pixel the object or primitive is drawn to.

```
shader_type spatial;

void fragment() {
    ALBEDO = vec3(1.0, 0.0, 0.0); // use red for material albedo
}
```

Light Processor

The “light” processor runs per pixel too, but also runs for every light that affects the object (and does not run if no lights affect the object).

```
shader_type spatial;

void light() {
    COLOR = vec3(0.0, 1.0, 0.0);
}
```

Varyings

To send data from vertex to fragment shader, *varyings* are used. They are set for every primitive vertex in the *vertex processor*, and the value is interpolated (and perspective corrected) when reaching every pixel in the fragment processor.

```
shader_type spatial;

varying vec3 some_color;
void vertex() {
    some_color = NORMAL; // make the normal the color
}

void fragment() {
    ALBEDO = some_color;
}
```

Interpolation qualifiers

Certain values are interpolated during the shading pipeline. You can modify how these interpolations are done by using *interpolation qualifiers*.

```
shader_type spatial;

varying flat vec3 our_color;

void vertex() {
    our_color = COLOR.rgb;
}

void fragment() {
    ALBEDO = our_color;
}
```

There are three possible interpolation qualifiers:

Qualifier	Description
flat	The value is not interpolated
noperspective	The value is linearly interpolated in window-space
smooth	The value is interpolated in a perspective-correct fashion. This is the default

Uniforms

Passing values to shaders is possible. These are global to the whole shader and called *uniforms*. When a shader is later assigned to a material, the uniforms will appear as editable parameters on it. Uniforms can't be written from within the shader.

```
shader_type spatial;

uniform float some_value;
```

Any type except for *void* can be a uniform. Additionally, Godot provides optional shader hints to make the compiler understand what the uniform is used for.

```
shader_type spatial;

uniform vec4 color : hint_color;
uniform float amount : hint_range(0, 1);
```

Full list of hints below:

Type	Hint	Description
vec4	hint_color	Used as color
int, float	hint_range(min,max [,step])	Used as range (with min/max/step)
sampler2D	hint_albedo	Used as albedo color, default white
sampler2D	hint_black_albedo	Used as albedo color, default black
sampler2D	hint_normal	Used as normalmap
sampler2D	hint_white	As value, default to white.
sampler2D	hint_black	As value, default to black
sampler2D	hint_aniso	As flowmap, default to right.

As Godot 3D engine renders in linear color space, it's important to understand that textures that are supplied as color (ie, albedo) need to be specified as such for proper SRGB->linear conversion.

Uniforms can also be assigned default values:

```
shader_type spatial;

uniform vec4 some_vector = vec4(0.0);
```

15.1.10 Built-in Functions

A large number of built-in functions are supported, conforming mostly to GLSL ES 3.0. When *vec_type* (*float*), *vec_int_type*, *vec_uint_type*, *vec_bool_type* nomenclature is used, it can be scalar or vector.

Function	Description
float sin (float)	Sine
float cos (float)	Cosine
float tan (float)	Tangent
float asin (float)	arc-Sine
float acos (float)	arc-Cosine
float atan (float)	arc-Tangent
float atan2 (float x, float y)	arc-Tangent to convert vector to angle

Continued on next page

Table 15.1 – continued from previous page

Function	Description
float sinh (float)	Hyperbolic-Sine
float cosh (float)	Hyperbolic-Cosine
float tanh (float)	Hyperbolic-Tangent
vec_type pow (float x, float y)	Power, x elevated to y
vec_type pow (vec_type, vec_type)	Power (Vec. Exponent)
vec_type exp (vec_type)	Base-e Exponential
vec_type log (vec_type)	Natural Logarithm
vec_type sqrt (vec_type)	Square Root
vec_type inversesqrt (vec_type)	Inverse Square Root
vec_type abs (vec_type)	Absolute
vec_type sign (vec_type)	Sign
vec_type floor (vec_type)	Floor
vec_type round (vec_type)	Round
vec_type trunc (vec_type)	Trunc
vec_type ceil (vec_type)	Ceiling
vec_type fract (vec_type)	Fractional
vec_type mod (vec_type, vec_type)	Remainder
vec_type modf (vec_type x, out vec_type i)	Fractional of x, with i has integer part
vec_type min (vec_type, vec_type)	Minimum
vec_type max (vec_type, vec_type)	Maximum
vec_type clamp (vec_type value, vec_type min, vec_type max)	Clamp to Min-Max
vec_type mix (vec_type a, vec_type b, float c)	Linear Interpolate
vec_type mix (vec_type a, vec_type b, vec_type c)	Linear Interpolate (Vector Coef.)
vec_type step (vec_type a, vec_type b)	'a[i] < b[i] ? 0.0 : 1.0'
vec_type smoothstep (vec_type a, vec_type b, vec_type c)	
vec_bool_type isnan (vec_type)	scalar, or vector component being nan
vec_bool_type isinf (vec_type)	scalar, or vector component being inf
vec_int_type floatBitsToInt (vec_type)	Float->Int bit copying, no conversion
vec_uint_type floatBitsToUInt (vec_type)	Float->UInt bit copying, no conversion
vec_type intBitsToFloat (vec_int_type)	Int->Float bit copying, no conversion
vec_type uintBitsToFloat (vec_uint_type)	UInt->Float bit copying, no conversion
float length (vec_type)	Vector Length
float distance (vec_type, vec_type)	Distance between vector.
float dot (vec_type, vec_type)	Dot Product
vec3 cross (vec3, vec3)	Cross Product
vec_type normalize (vec_type)	Normalize to unit length
vec3 reflect (vec3, vec3)	Reflect
vec3 refract (vec3, vec3)	Refract
vec_type faceforward (vec_type N, vec_type I, vec_type NRef)	If dot(Nref, I) < 0 return N, otherwise -N
mat_type matrixCompMult (mat_type, mat_type)	Matrix Component Multiplication
mat_type outerProduct (vec_type, vec_type)	Matrix Outer Product
mat_type transpose (mat_type)	Transpose Matrix
float determinant (mat_type)	Matrix Determinant
mat_type inverse (mat_type)	Inverse Matrix
vec_bool_type lessThan (vec_scalar_type)	Bool vector cmp on < int/uint/float vectors
vec_bool_type greaterThan (vec_scalar_type)	Bool vector cmp on > int/uint/float vectors
vec_bool_type lessThanEqual (vec_scalar_type)	Bool vector cmp on <= int/uint/float vectors
vec_bool_type greaterThanEqual (vec_scalar_type)	Bool vector cmp on >= int/uint/float vectors
vec_bool_type equal (vec_scalar_type)	Bool vector cmp on == int/uint/float vectors

Continued on next page

Table 15.1 – continued from previous page

Function	Description
<code>vec_bool_type notEqual (vec_scalar_type)</code>	Bool vector cmp on != int/uint/float vectors
<code>bool any (vec_bool_type)</code>	Any component is true
<code>bool all (vec_bool_type)</code>	All components are true
<code>bool not (vec_bool_type)</code>	No components are true
<code>ivec2 textureSize (sampler2D_type s, int lod)</code>	Get the size of a texture
<code>ivec2 textureSize (samplerCube s, int lod)</code>	Get the size of a cubemap
<code>vec4_type texture (sampler2D_type s, vec2 uv [, float bias])</code>	Perform a 2D texture read
<code>vec4_type texture (samplerCube s, vec3 uv [, float bias])</code>	Perform a Cube texture read
<code>vec4_type textureProj (sampler2d_type s, vec3 uv [, float bias])</code>	Perform a texture read with projection
<code>vec4_type textureProj (sampler2d_type s, vec4 uv [, float bias])</code>	Perform a texture read with projection
<code>vec4_type textureLod (sampler2D_type s, vec2 uv , float lod)</code>	Perform a 2D texture read at custom mipmap
<code>vec4_type textureProjLod (sampler2d_type s, vec3 uv , float lod)</code>	Perform a texture read with projection/lod
<code>vec4_type textureProjLod (sampler2d_type s, vec4 uv , float lod)</code>	Perform a texture read with projection/lod
<code>vec_type texelFetch (samplerCube s, ivec2 uv, int lod)</code>	Fetch a single texel using integer coords
<code>vec_type dFdx (vec_type)</code>	Derivative in x using local differencing
<code>vec_type dFdy (vec_type)</code>	Derivative in y using local differencing
<code>vec_type fwidth (vec_type)</code>	Sum of absolute derivative in x and y

15.1.11 Shader Types In-Depth

Spatial

Accepted render modes and built-ins for “`shader_type spatial;`”.

Render Modes

Render Mode	Description
<code>blend_mix</code>	Mix blend mode (alpha is transparency), default.
<code>blend_add</code>	Additive blend mode.
<code>blend_sub</code>	Substractive blend mode.
<code>blend_mul</code>	Multiplicative blend mode.
<code>depth_draw_opaque</code>	Only draw depth for opaque geometry (not transparent).
<code>depth_draw_always</code>	Always draw depth (opaque and transparent).
<code>depth_draw_never</code>	Never draw depth.
<code>depth_draw_alpha_prepass</code>	Do opaque depth pre-pass for transparent geometry.
<code>depth_test_disable</code>	Disable depth testing.
<code>cull_front</code>	Cull front-faces.
<code>cull_back</code>	Cull back-faces (default).
<code>cull_disabled</code>	Culling disabled (double sided).
<code>unshaded</code>	Result is just albedo. No lighting/shading happens in material.
<code>diffuse_lambert</code>	Lambert shading for diffuse (default).
<code>diffuse_lambert_wrap</code>	Lambert wrapping (roughness dependent) for diffuse.
<code>diffuse_oren_nayar</code>	Oren Nayar for diffuse.
<code>diffuse_burley</code>	Burley (Disney PBS) for diffuse.
<code>diffuse_toon</code>	Toon shading for diffuse.
<code>specular_schlick_ggx</code>	Schlick-GGX for specular (default).
<code>specular_blinn</code>	Blinn for specular (compatibility).
<code>specular_phong</code>	Phong for specular (compatibility).
<code>specular_toon</code>	Toon for specular.
<code>specular_disabled</code>	Disable specular.
<code>skip_vertex_transform</code>	VERTEX/NORMAL/etc need to be transformed manually in VS.
<code>world_vertex_coords</code>	VERTEX/NORMAL/etc are modified in world coordinates instead of local.
<code>vertex_lighting</code>	Use vertex-based lighting.

Vertex Built-Ins

Built-In	Object vertex to world space transform.
mat4 WORLD_MATRIX	Model space to world space transform.
mat4 INV_CAMERA_MATRIX	World space to view space transform.
mat4 PROJECTION_MATRIX	View space to clip space transform.
mat4 CAMERA_MATRIX	View space to world space transform.
mat4 MODELVIEW_MATRIX	Model space to view space transform (use if possible).
mat4 INV_PROJECTION_MATRIX	Clip space to view space transform.
float TIME	Elapsed total time in seconds.
vec2 VIEWPORT_SIZE	Size of viewport (in pixels).
vec3 VERTEX	Vertex in local coords (see doc below).
vec3 NORMAL	Normal in local coords.
vec3 TANGENT	Tangent in local coords.
vec3 BINORMAL	Binormal in local coords.
vec2 UV	UV main channel.
vec2 UV2	UV secondary channel.
vec4 COLOR	Color from vertices.
vec2 POINT_SIZE	Point size for point rendering.
int INSTANCE_ID	Instance ID for instancing.
vec4 INSTANCE_CUSTOM	Instance custom data (for particles, mostly).
float ROUGHNESS	Roughness for vertex lighting.

Vertex data (VERTEX, NORMAL, TANGENT, BITANGENT) is presented in local model space. If not written to, these values will not be modified and be passed through as they came.

They can be optionally set to be presented in world space (after being transformed by world) by adding the *world_vertex_coords* render mode.

It is also possible to completely disable the built-in modelview transform (projection will still happen later, though) with the following code, so it can be done manually:

```
shader_type spatial;
render_mode skip_vertex_transform;

void vertex() {
    VERTEX = (MODELVIEW_MATRIX * vec4(VERTEX, 1.0)).xyz;
    NORMAL = (MODELVIEW_MATRIX * vec4(VERTEX, 0.0)).xyz;
    // same as above for binormal and tangent, if normal mapping is used
}
```

Other built-ins such as UV, UV2 and COLOR are also passed through to the fragment function if not modified.

For instancing, the INSTANCE_CUSTOM variable contains the instance custom data. When using particles, this information is usually:

- **x**: Rotation angle in radians.
- **y**: Phase during lifetime (0 to 1).
- **z**: Animation frame.

This allows to easily adjust the shader to a particle system using default particles material. When writing a custom particles shader, this value can be used as desired.

Fragment Built-Ins

	Built-In	Description
in mat4 WORLD_MATRIX		Model space to world space transform.
in mat4 INV_CAMERA_MATRIX		World space to view space transform.
in mat4 PROJECTION_MATRIX		View space to clip space transform.
in mat4 CAMERA_MATRIX		View space to world space transform.
in mat4 INV_PROJECTION_MATRIX		Clip space to view space transform.
float TIME		Elapsed total time in seconds.
vec2 VIEWPORT_SIZE		Size of viewport (in pixels).
vec3 VERTEX		Vertex that comes from vertex function, in view space.
in vec4 FRAGCOORD		Fragment coordinate, pixel adjusted.
in bool FRONT_FACING		true whether current face is front face.
inout vec3 NORMAL		Normal that comes from vertex function, in view space.
inout vec3 TANGENT		Tangent that comes from vertex function.
inout vec3 BINORMAL		Binormal that comes from vertex function.
out vec3 NORMALMAP		Output this if reading normal from a texture instead of NORMAL.
out float NORMALMAP_DEPTH		Depth from variable above. Defaults to 1.0.
in vec2 UV		UV that comes from vertex function.
in vec2 UV2		UV2 that comes from vertex function.
in vec4 COLOR		COLOR that comes from vertex function.
out vec3 ALBEDO		Albedo (default white).
out float ALPHA		Alpha (0..1), if written to the material will go to transparent pipeline.
out float METALLIC		Metallic (0..1).
out float SPECULAR		Specular. Defaults to 0.5, best to not modify unless you want to change IOR.
out float ROUGHNESS		Roughness (0..1).
out float RIM		Rim (0..1).
out float RIM_TINT		Rim Tint, goes from 0 (white) to 1 (albedo).
out float CLEARCOAT		Small added specular blob.
out float CLEARCOAT_GLOSS		Gloss of Clearcoat.
out float ANISOTROPY		For distorting the specular blob according to tangent space.
out float ANISOTROPY_FLOW		Distortion direction, use with flowmaps.
out float SSS_STRENGTH		Strength of Subsurface Scattering (default 0).
out vec3 TRANSMISSION		Transmission mask (default 0,0,0).
out float AO		Ambient Occlusion (pre-baked).
out float AO_LIGHT_AFFECT		How much AO affects lights (0..1. default 0, none).
out vec3 EMISSION		Emission color (can go over 1,1,1 for HDR).
in sampler2D SCREEN_TEXTURE		Built-in Texture for reading from the screen. Mipmaps contain increasingly blurred copies.
in sampler2D DEPTH_TEXTURE		Built-in Texture for reading depth from the screen. Must convert to linear using INV_PROJ .
in vec2 SCREEN_UV		Screen UV coordinate for current pixel.
in vec2 POINT_COORD		Point Coord for drawing points with POINT_SIZE .
in float SIDE		SIDE multiplier, for double sided materials.
out float ALPHA_SCISSOR		If written to, values below a certain amount of alpha are discarded.

Values marked as “in” are read-only. Values marked as “out” are for optional writing. If nothing is written, a default value is used.

Light Built-Ins

Built-in	Description
in mat4 WORLD_MATRIX	Model space to world space transform.
in mat4 INV_CAMERA_MATRIX	World space to view space transform.
in mat4 PROJECTION_MATRIX	View space to clip space transform.
in mat4 CAMERA_MATRIX	View space to world space transform.
in mat4 INV_PROJECTION_MATRIX	Clip space to view space transform.
in float TIME	Elapsed total time in seconds.
in vec2 VIEWPORT_SIZE	Size of viewport (in pixels).
in vec3 NORMAL	Normal vector.
in vec3 VIEW	View vector.
in vec3 LIGHT	Light Vector.
in vec3 LIGHT_COLOR	Color of light multiplied by energy.
in vec3 ATTENUATION	Attenuation based on distance or shadow.
in vec3 ALBEDO	Base albedo.
in vec3 TRANSMISSION	Transmission mask.
in float ROUGHNESS	Roughness.
out vec3 DIFFUSE_LIGHT	Diffuse light result.
out vec3 SPECULAR_LIGHT	Specular light result.

Writing light shaders is completely optional. Unlike other game engines, they don't affect performance or force a specific pipeline.

To write a light shader, simply make sure to assign something to DIFFUSE_LIGHT or SPECULAR_LIGHT. Assigning nothing means no light is processed.

Canvas Item

Accepted render modes and built-ins for “`shader_type canvas_item;`”.

Render Modes

Render Mode	Description
blend_mix	Mix blend mode (alpha is transparency), default.
blend_add	Additive blend mode.
blend_sub	Subtractive blend mode.
blend_mul	Multiplicative blend mode.
blend_premul_alpha	Premultiplied alpha blend mode.
unshaded	Result is just albedo. No lighting/shading happens in material.
light_only	Only draw for light pass (when multipass is used).
skip_vertex_transform	VERTEX/NORMAL/etc need to be transformed manually in VS.

Vertex Built-Ins

Built-In	Description
in mat4 WORLD_MATRIX	Image to World transform.
in mat4 EXTRA_MATRIX	Extra transform.
in mat4 PROJECTION_MATRIX	World to view transform.
in float TIME	Global time, in seconds.
in vec4 INSTANCE_CUSTOM	Instance custom data.
in bool AT_LIGHT_PASS	True if this is a light pass (for multi-pass light rendering).
inout vec2 VERTEX	Vertex in image space.
inout vec2 UV	UV.
inout vec4 COLOR	Color from vertex primitive.
out vec2 POINT_SIZE	Point size for point drawing.

Vertex data (VERTEX) is presented in local space. If not written to, these values will not be modified and be passed through as they came.

It is possible to completely disable the built-in modelview transform (projection will still happen later, though) with the following code, so it can be done manually:

```
shader_type spatial;
render_mode skip_vertex_transform;

void vertex() {
    VERTEX = (EXTRA_MATRIX * (WORLD_MATRIX * vec4(VERTEX, 0.0, 1.0))).xy;
}
```

Other built-ins such as UV and COLOR are also passed through to the fragment function if not modified.

For instancing, the INSTANCE_CUSTOM variable contains the instance custom data. When using particles, this information is usually:

- **x**: Rotation angle in radians.
- **y**: Phase during lifetime (0 to 1).
- **z**: Animation frame.

This allows to easily adjust the shader to a particle system using default particles material. When writing a custom particles shader, this value can be used as desired.

Fragment Built-Ins

Built-In	Description
in vec4 FRAGCOORD	Fragment coordinate, pixel adjusted.
out vec3 NORMAL	Normal, writable.
out vec3 NORMALMAP	Normal from texture, default is read from NORMAL_TEXTURE .
out float NORMALMAP_DEPTH	Normalmap depth for scaling.
in vec2 UV	UV from vertex function.
in vec4 COLOR	Color from vertex function.
in sampler2D TEXTURE	Default 2D texture.
in sampler2D NORMAL_TEXTURE	Default 2D normal texture.
in vec2 TEXTURE_PIXEL_SIZE	Default 2D texture pixel size.
in vec2 SCREEN_UV	Screen UV for use with SCREEN_TEXTURE .
in vec2 SCREEN_PIXEL_SIZE	Screen pixel size.
in vec2 POINT_COORD	Coordinate for drawing points.
in float TIME	Global time in seconds.
in sampler2D SCREEN_TEXTURE	Screen texture, mipmaps contain gaussian blurred versions.

Light Built-Ins

Built-In	Description
in vec4 POSITION	Screen Position.
in vec3 NORMAL	Input Normal.
in vec2 UV	UV.
in vec4 COLOR	Input Color.
in sampler2D TEXTURE	Current texture in use for CanvasItem.
in vec2 TEXTURE_PIXEL_SIZE	Pixel size for current 2D texture.
in vec2 SCREEN_UV	Screen Texture Coordinate (for using with texscreen).
in vec2 POINT_COORD	Current UV for Point Sprite.
in float TIME	Time (in seconds).
vec2 LIGHT_VEC	Vector from light to fragment, can be modified to alter shadow computation.
in float LIGHT_HEIGHT	Height of Light.
in color LIGHT_COLOR	Color of Light.
in color LIGHT_SHADOW_COLOR	Color of Light shadow.
in vec2 LIGHT_UV	UV for light image.
in vec4 SHADOW	Light shadow color override.
out vec4 LIGHT	Light Output (shader is ignored if this is not used).

Particles

Accepted render modes and built-ins for “`shader_type particles;`”.

Render Modes

Render Mode	Description
billboard	Particle will be shown as a billboard.
keep_data	Do not clear previous data on restart.

Vertex Built-Ins

Built-In	Description
inout vec4 COLOR	Particle color, can be written to.
inout vec3 VELOCITY	Particle velocity, can be modified.
out float MASS	Particle mass, use for attractors (default 1).
inout bool ACTIVE	Particle is active, can be set to false.
in bool RESTART	Set to true when particle must restart (lifetime cycled).
inout vec4 CUSTOM	Custom particle data.
inout mat4 TRANSFORM	Particle transform.
in float TIME	Global time in seconds.
in float DELTA	Delta process time.
in uint NUMBER	Unique number since emission start.
in int INDEX	Particle index (from total particles).
in mat4 EMISSION_TRANSFORM	Emitter transform (used for non-local systems).
in uint RANDOM_SEED	Random seed used as base for random.

Particle shades only support vertex processing. They are drawn with any regular material for CanvasItem or Spatial, depending on whether they are 2D or 3D.

15.2 Shader materials

15.2.1 Introduction

For the most common cases, Godot provides ready to use materials for most types of shaders, such as SpatialMaterial, CanvasItemMaterial and ParticlesMaterial (@TODO link to tutorials/classes). They are flexible implementations that cover most use cases.

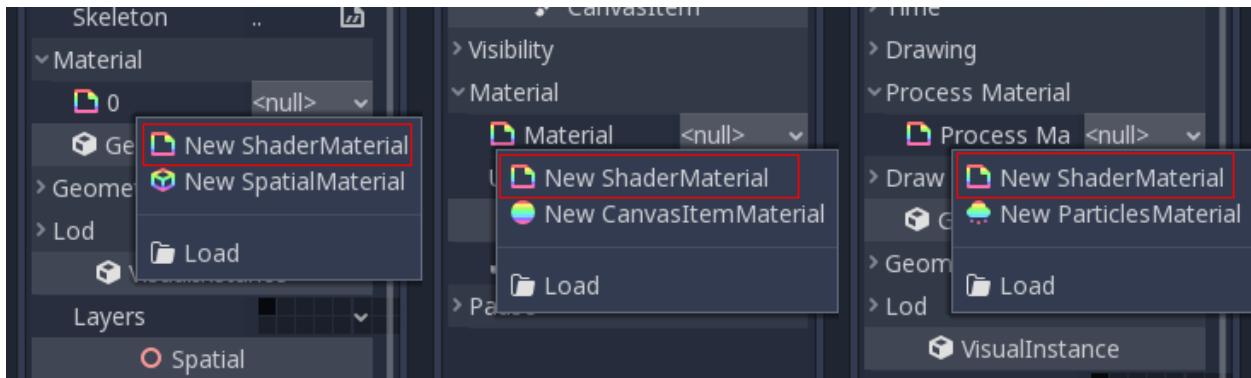
Shader materials allow writing a custom shader directly, for maximum flexibility. Examples of this are:

- Create procedural textures.
- Create complex texture blendings.
- Create animated materials, or materials that change with time.
- Create refractive effects or other advanced effects.
- Create special lighting shaders for more exotic materials.
- Animate vertices, like tree leaves or grass.
- Create custom particle code, that responds to baked animations or force fields.
- And much more!

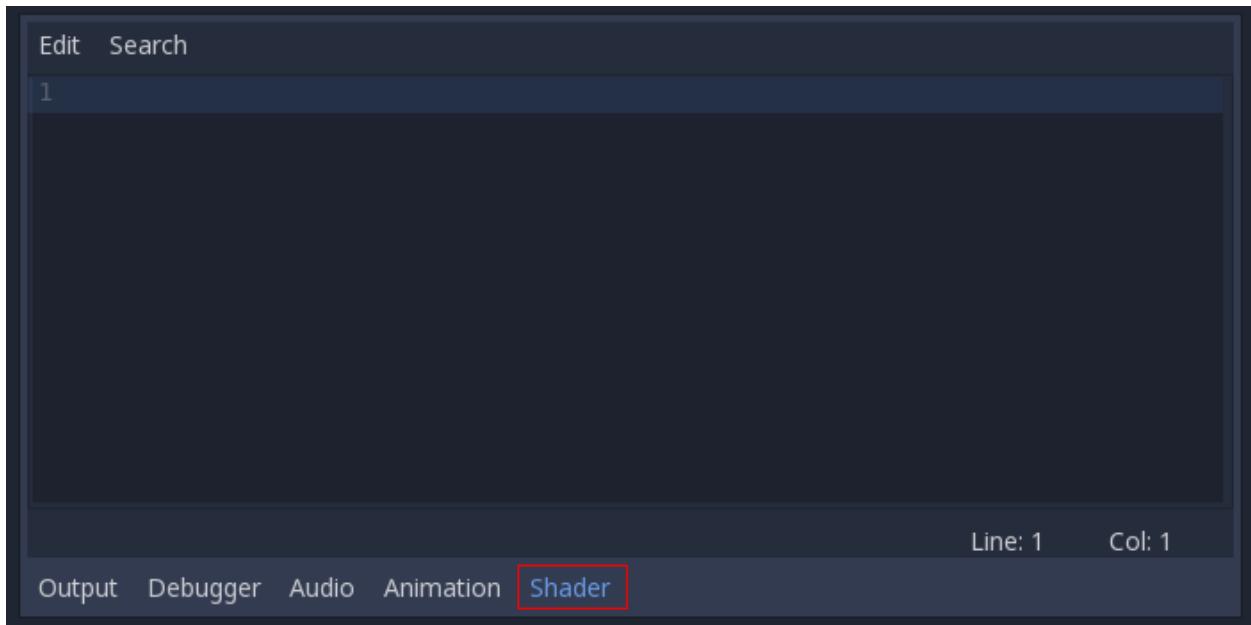
Traditionally, most engines will ask you to learn GLSL, HLSL or CG, which are pretty complex for the skillset of most artists. Godot uses a simplified version of a shader language that will detect errors as you type, so you can see your edited shaders in real-time. Additionally, it is possible to edit shaders using a visual, node-based graph editor.

15.2.2 Creating a ShaderMaterial

Create a new ShaderMaterial in some object of your choice. Go to the “Material” property and create a ShaderMaterial.

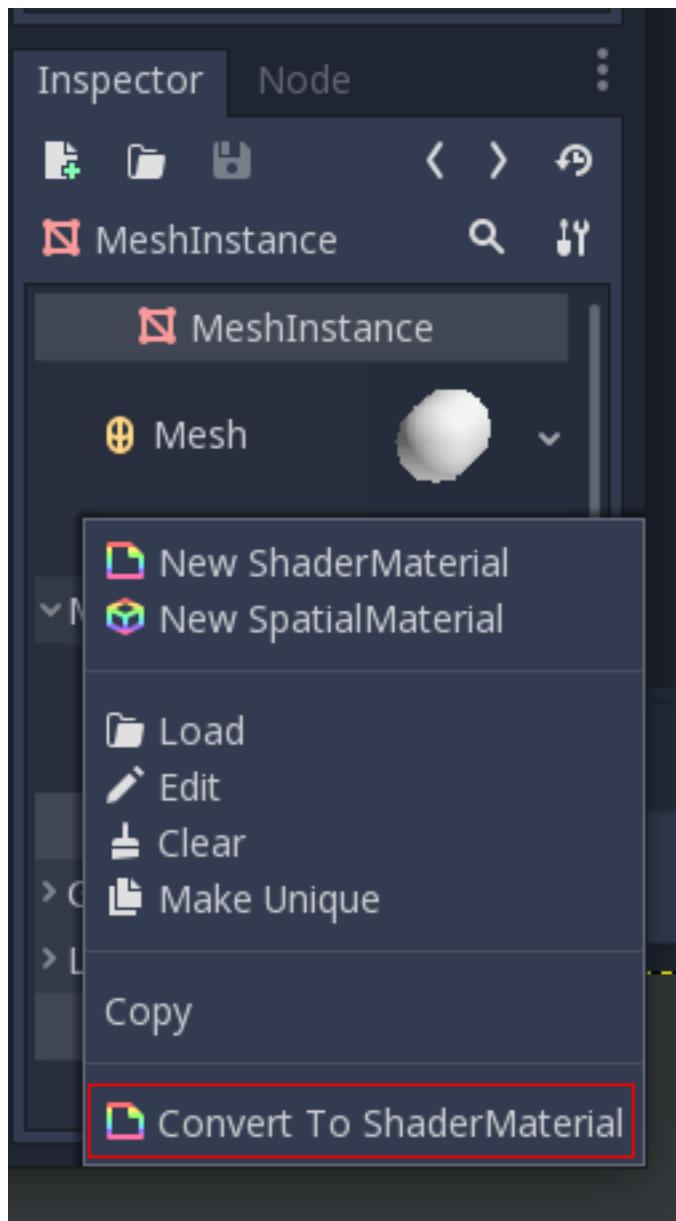


Edit the newly created shader, and the shader editor will open:



15.2.3 Converting to ShaderMaterial

It is possible to convert from SpatialMaterial, CanvasItemMaterial and ParticlesMaterial to ShaderMaterial. Just go to the material properties and enable the convert option.



15.3 Screen-reading shaders

15.3.1 Introduction

Very often it is desired to make a shader that reads from the same screen it's writing to. 3D APIs such as OpenGL or DirectX make this very difficult because of internal hardware limitations. GPUs are extremely parallel, so reading and writing causes all sort of cache and coherency problems. As a result, not even the most modern hardware supports this properly.

The workaround is to make a copy of the screen, or a part of the screen, to a back-buffer and then read from it while drawing. Godot provides a few tools that makes this process easy!

15.3.2 SCREEN_TEXTURE built-in texture.

Godot *Shading language* has a special texture, “SCREEN_TEXTURE” (and “DEPTH_TEXTURE” for depth, in case of 3D). It takes as parameter the UV of the screen and returns a vec3 RGB with the color. A special built-in varying: SCREEN_UV can be used to obtain the UV for the current fragment. As a result, this simple 2D fragment shader:

```
COLOR=textureLod( SCREEN_TEXTURE, SCREEN_UV, 0.0 );
```

results in an invisible object, because it just shows what lies behind.

The reason why textureLod must be used is because, when Godot copies back a chunk of the screen, it also does an efficient separable gaussian blur to its mipmaps.

This allows for not only reading from the screen, but reading from it with different amounts of blur at no cost.

15.3.3 SCREEN_TEXTURE example

SCREEN_TEXTURE can be used for a lot of things. There is a special demo for *Screen Space Shaders*, that you can download to see and learn. One example is a simple shader to adjust brightness, contrast and saturation:

```
shader_type canvas_item;

uniform float brightness = 1.0;
uniform float contrast = 1.0;
uniform float saturation = 1.0;

vec3 c = textureLod(SCREEN_TEXTURE, SCREEN_UV, 0.0).rgb;

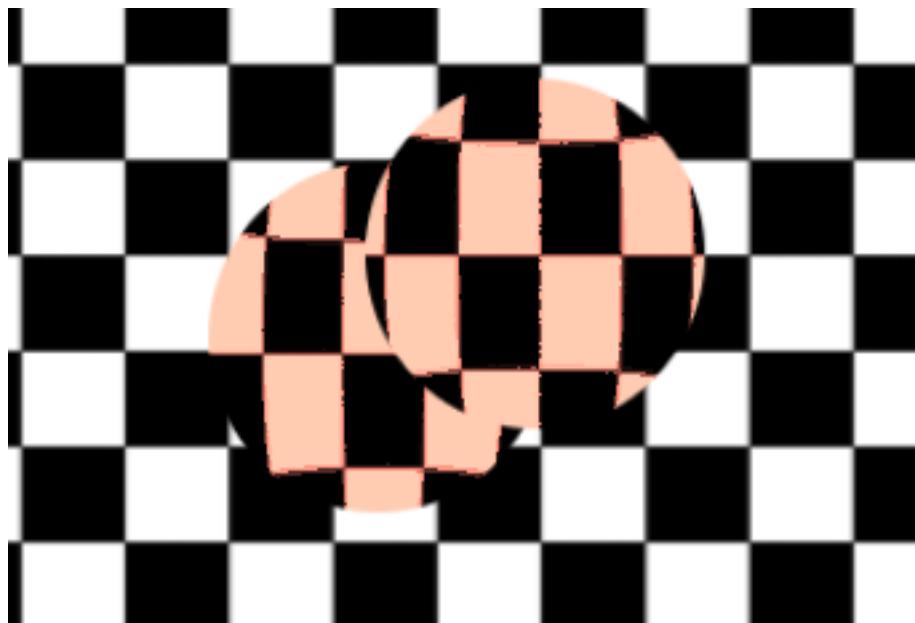
c.rgb = mix(vec3(0.0), c.rgb, brightness);
c.rgb = mix(vec3(0.5), c.rgb, contrast);
c.rgb = mix(vec3(dot(vec3(1.0), c.rgb)*0.33333), c.rgb, saturation);

COLOR.rgb = c;
```

15.3.4 Behind the scenes

While this seems magical, it’s not. The SCREEN_TEXTURE built-in, when first found in a node that is about to be drawn, does a full-screen copy to a back-buffer. Subsequent nodes that use it in shaders will not have the screen copied for them, because this ends up being very inefficient.

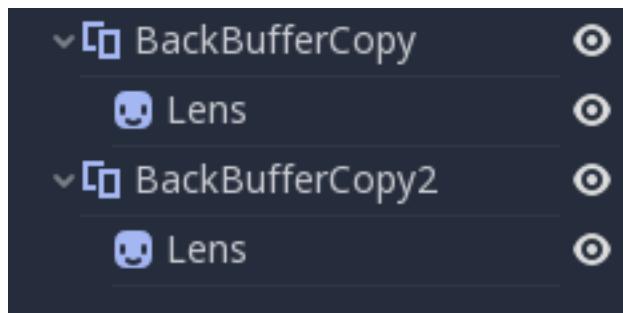
As a result, if shaders that use SCREEN_TEXTURE overlap, the second one will not use the result of the first one, resulting in unexpected visuals:



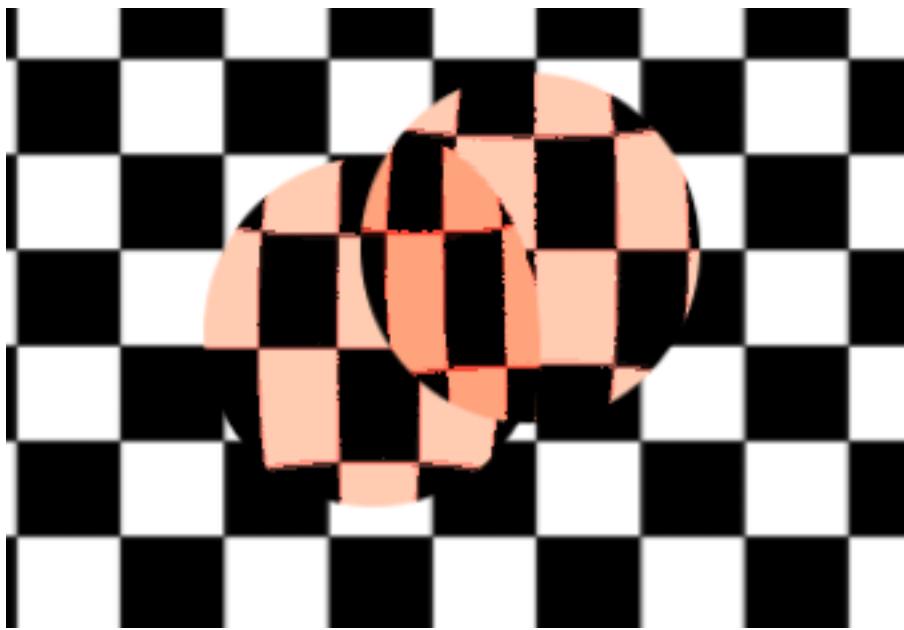
In the above image, the second sphere (top right) is using the same source for SCREEN_TEXTURE as the first one below, so the first one “disappears”, or is not visible.

In 3D, this is unavoidable because copying happens when opaque rendering completes.

In 2D this can be corrected via the *BackBufferCopy* node, which can be instantiated between both spheres. BackBufferCopy can work by either specifying a screen region or the whole screen:



With correct back-buffer copying, the two spheres blend correctly:



15.3.5 Back-buffer logic

So, to make it clearer, here's how the backbuffer copying logic works in Godot:

- If a node uses the SCREEN_TEXTURE, the entire screen is copied to the back buffer before drawing that node. This only happens the first time, subsequent nodes do not trigger this.
- If a BackBufferCopy node was processed before the situation in the point above (even if SCREEN_TEXTURE was not used), this behavior described in the point above does not happen. In other words, automatic copying of the entire screen only happens if SCREEN_TEXTURE is used in a node for the first time and no BackBufferCopy node (not disabled) was found before in tree-order.
- BackBufferCopy can copy either the entire screen or a region. If set to only a region (not the whole screen) and your shader uses pixels not in the region copied, the result of that read is undefined (most likely garbage from previous frames). In other words, it's possible to use BackBufferCopy to copy back a region of the screen and then use SCREEN_TEXTURE on a different region. Avoid this behavior!

15.3.6 DEPTH_TEXTURE

For 3D Shaders, it's also possible to access the screen depth buffer. For this, the DEPTH_TEXTURE built-in is used. This texture is not linear, it must be converted via the inverse projection matrix.

The following code retrieves the 3D position below the pixel being drawn:

```
float depth = textureLod(DEPTH_TEXTURE, SCREEN_UV, 0.0).r;
vec4 upos = INV_PROJECTION_MATRIX * vec4(SCREEN_UV*2.0-1.0, depth*2.0-1.0, 1.0);
vec3 pixel_position = upos.xyz/upos.w;
```


CHAPTER 16

Networking

16.1 High level multiplayer

16.1.1 High level vs low level API

The following explains the differences of high- and low-level networking in Godot as well as some fundamentals. If you want to jump in head-first and add networking to your first nodes, skip to *Initializing the network* below. But make sure to read the rest later on!

Godot always supported standard low-level networking via UDP, TCP and some higher level protocols such as SSL and HTTP. These protocols are flexible and can be used for almost anything. However using them to synchronize game state manually can be a large amount of work. Sometimes that work can't be avoided or is worth it, for example when working with a custom server implementation on the backend. But in most cases it's worthwhile to consider Godot's high-level networking API, which sacrifices some of the fine-grained control of low-level networking for greater ease of use.

This is due to the inherent limitations of the low-level protocols:

- TCP ensures packets will always arrive reliably and in order, but latency is generally higher due to error correction. It's also quite a complex protocol because it understands what a "connection" is, and optimizes for goals that often don't suit applications like multiplayer games. Packets are buffered to be sent in larger batches, trading less per-packet overhead for higher latency. This can be useful for things like HTTP, but generally not for games. Some of this can be configured and disabled (e.g. by disabling "Nagle's algorithm" for the TCP connection).
- UDP is a simpler protocol which only sends packets (and has no concept of a "connection"). No error correction makes it pretty quick (low latency), but packets may be lost along the way or received in the wrong order. Added to that, the MTU (maximum packet size) for UDP is generally low (only a few hundred bytes), so transmitting larger packets means splitting them, reorganizing them and retrying if a part fails.

In general, TCP can be thought of as reliable, ordered, and slow; UDP as unreliable, unordered and fast. Because of the large difference in performance it often makes sense to re-build the parts of TCP wanted for games (optional reliability and packet order) while avoiding the unwanted parts (congestion/traffic control features, Nagle's algorithm, etc). Due to this most game engines come with such an implementation, and Godot is no exception.

In summary you can use the low-level networking API for maximum control and implement everything on top of bare network protocols or use the high-level API based on *SceneTree* that does most of the heavy lifting behind the scenes in a generally optimized way.

Note: Most of Godot's supported platforms offer all or most of the mentioned high- and low-level networking features. As networking is always largely hardware and operating system dependent, however, some features may change or not be available on some target platforms. Most notably, the HTML5 platform currently only offers WebSocket support and lacks some of the higher level features as well as raw access to low-level protocols like TCP and UDP.

Note: More about TCP/IP, UDP, and networking: https://gafferongames.com/post/udp_vs_tcp/

Gaffer On Games has a lot of useful articles about networking in Games ([here](#)), including the comprehensive [introduction to networking models in games](#).

If you want to use your low-level networking library of choice instead of Godot's built-in networking, see [here](#) for an example: <https://github.com/PerduGames/gdnet3>

Warning: Adding networking to your game comes with some responsibility. It can make your application vulnerable if done wrong and may lead to cheats or exploits. It may even allow an attacker to compromise the machines your application runs on and use your servers to send spam, attack others or steal your users data if they play your game.

This is always the case when networking is involved and has nothing to do with Godot. You can of course experiment, but when you release a networked application, always take care of any possible security concerns.

16.1.2 Mid level abstraction

Before going into how we would like to synchronize a game across the network, it can be helpful to understand how the base network API for synchronization works.

Godot uses a mid-level object *NetworkedMultiplayerPeer*. This object is not meant to be created directly, but is designed so that several implementations can provide it:

Class: NetworkedMultiplayerPeer

Inherits: [PacketPeer](#) , [Reference](#) , [Object](#)

Inherited by: [NetworkedMultiplayerENet](#)

Public Methods:

```
int    get_connection_status() const
int    get_packet_peer() const
int    get_unique_id() const
bool   is_refusing_new_connections() const
void   poll()
void   set_refuse_new_connections( bool enable )
void   set_target_peer( int id )
void   set_transfer_mode( int mode )
```

Signals:

```
connection_failed()
connection_succeeded()
peer_connected( int id )
peer_disconnected( int id )
server_disconnected()
```

Constants:

```
TRANSFER_MODE_UNRELIABLE = 0
TRANSFER_MODE_UNRELIABLE_ORDERED = 1
TRANSFER_MODE_RELIABLE = 2
CONNECTION_DISCONNECTED = 0
CONNECTION_CONNECTING = 1
CONNECTION_CONNECTED = 2
TARGET_PEER_BROADCAST = 0
TARGET_PEER_SERVER = 1
```

This object extends from *PacketPeer*, so it inherits all the useful methods for serializing, sending and receiving data. On top of that, it adds methods to set a peer, transfer mode, etc. It also includes signals that will let you know when peers connect or disconnect.

This class interface can abstract most types of network layers, topologies and libraries. By default Godot provides an implementation based on ENet (*NetworkedMultiplayerENet*), but this could be used to implement mobile APIs (for adhoc WiFi, Bluetooth) or custom device/console-specific networking APIs.

For most common cases, using this object directly is discouraged, as Godot provides even higher level networking facilities. Yet it is made available in case a game has specific needs for a lower level API.

16.1.3 Initializing the network

The object that controls networking in Godot is the same one that controls everything tree-related: *SceneTree*.

To initialize high level networking, the *SceneTree* must be provided a *NetworkedMultiplayerPeer* object.

To create that object it first has to be initialized as a server or client.

Initializing as a server, listening on the given port, with a given maximum number of peers:

```
var peer = NetworkedMultiplayerENet.new()
peer.create_server(SERVER_PORT, MAX_PLAYERS)
get_tree().set_network_peer(peer)
```

Initializing as a client, connecting to a given IP and port:

```
var peer = NetworkedMultiplayerENet.new()
peer.create_client(SERVER_IP, SERVER_PORT)
get_tree().set_network_peer(peer)
```

Note that it may make sense to store the local network peer instance on the *SceneTree* to be able to access it later, as there currently is no *get_tree().get_network_peer()*. This can be done via *SceneTree*'s metadata feature:

```
get_tree().set_meta("network_peer", peer)
```

Checking whether the tree is initialized as a server or client:

```
get_tree().is_networking_server()
```

Terminating the networking feature:

```
get_tree().set_network_peer(null)
```

(Although it may make sense to send a message first to let the other peers know you're going away instead of letting the connection close or timeout, depending on your game.)

16.1.4 Managing connections

Some games accept connections at any time, others during the lobby phase. Godot can be requested to no longer accept connections at any point (see *set_refuse_new_network_connections(bool)* and related methods on *SceneTree*). To manage who connects, Godot provides the following signals in *SceneTree*:

Server and Clients:

- *network_peer_connected(int id)*
- *network_peer_disconnected(int id)*

The above signals are called on every peer connected to the server (including on the server) when a new peer connects or disconnects. Clients will connect with a unique ID greater than 1, while network peer ID 1 is always the server. Anything below 1 should be handled as invalid. You can retrieve the ID for the local system via `SceneTree.get_network_unique_id()`. These IDs will be useful mostly for lobby management and should generally be stored as they identify connected peers and thus players. You can also use IDs to send messages only to certain peers.

Clients:

- `connected_to_server`
- `connection_failed`
- `server_disconnected`

Again, all these functions are mainly useful for lobby management or for adding/removing players on the fly. For these tasks the server clearly has to work as a server and you have do tasks manually such as sending a newly connected player information about other already connected players (e.g. their names, stats, etc).

Lobbies can be implemented any way you want, but the most common way is to use a node with the same name across scenes in all peers. Generally, an autoloaded node/singleton is a great fit for this, to always have access to, e.g. “/root/lobby”.

16.1.5 RPC

To communicate between peers, the easiest way is to use RPCs (remote procedure calls). This is implemented as a set of functions in `Node`:

- `rpc(“function_name”, <optional_args>)`
- `rpc_id(<peer_id>, “function_name”, <optional_args>)`
- `rpc_unreliable(“function_name”, <optional_args>)`
- `rpc_unreliable_id(<peer_id>, “function_name”, <optional_args>)`

Synchronizing member variables is also possible:

- `rset(“variable”, value)`
- `rset_id(<peer_id>, “variable”, value)`
- `rset_unreliable(“variable”, value)`
- `rset_unreliable_id(<peer_id>, “variable”, value)`

Functions can be called in two fashions:

- Reliable: the function call will arrive no matter what, but may take longer because it will be re-transmitted in case of failure.
- Unreliable: if the function call does not arrive, it will not be re-transmitted, but if it arrives it will do it quickly.

In most cases, reliable is desired. Unreliable is mostly useful when synchronizing object positions (sync must happen constantly, and if a packet is lost, it’s not that bad because a new one will eventually arrive and it would likely be outdated because the object moved further in the meantime, even if it was resent reliably).

There is also the `get_rpc_sender_id` function in `SceneTree` which can be used to check which peer (or peer ID) sent a RPC call.

16.1.6 Back to lobby

Let's get back to the lobby. Imagine that each player that connects to the server will tell everyone about it.

```
# Typical lobby implementation, imagine this being in /root/lobby

extends Node

# Connect all functions

func _ready():
    get_tree().connect("network_peer_connected", self, "_player_connected")
    get_tree().connect("network_peer_disconnected", self, "_player_disconnected")
    get_tree().connect("connected_to_server", self, "_connected_ok")
    get_tree().connect("connection_failed", self, "_connected_fail")
    get_tree().connect("server_disconnected", self, "_server_disconnected")

# Player info, associate ID to data
var player_info = {}
# Info we send to other players
var my_info = { name = "Johnson Magenta", favorite_color = Color8(255, 0, 255) }

func _player_connected(id):
    pass # Will go unused, not useful here

func _player_disconnected(id):
    player_info.erase(id) # Erase player from info

func _connected_ok():
    # Only called on clients, not server. Send my ID and info to all the other peers
    rpc("register_player", get_tree().get_network_unique_id(), my_info)

func _server_disconnected():
    pass # Server kicked us, show error and abort

func _connected_fail():
    pass # Could not even connect to server, abort

remote func register_player(id, info):
    # Store the info
    player_info[id] = info
    # If I'm the server, let the new guy know about existing players
    if (get_tree().is_network_server()):
        # Send my info to new player
        rpc_id(id, "register_player", 1, my_info)
        # Send the info of existing players
        for peer_id in player_info:
            rpc_id(id, "register_player", peer_id, player_info[peer_id])

    # Call function to update lobby UI here
```

You might have noticed already something different, which is the usage of the *remote* keyword on the *register_player* function:

```
remote func register_player(id, info):
```

This keyword has two main uses. The first is to let Godot know that this function can be called from RPC. If no keywords are added Godot will block any attempts to call functions for security. This makes security work a lot easier

(so a client can't call a function to delete a file on another client's system).

The second use is to specify how the function will be called via RPC. There are four different keywords:

- *remote*
- *sync*
- *master*
- *slave*

The *remote* keyword means that the *rpc()* call will go via network and execute remotely.

The *sync* keyword means that the *rpc()* call will go via network and execute remotely, but will also execute locally (do a normal function call).

The others will be explained further down. Note that you could also use the *get_rpc_sender_id* function on *SceneTree* to check which peer actually made the RPC call to *register_player*.

With this, lobby management should be more or less explained. Once you have your game going you will most likely want to add some extra security to make sure clients don't do anything funny (just validate the info they send from time to time, or before game start). For the sake of simplicity and because each game will share different information, this is not shown here.

16.1.7 Starting the game

Once enough players have gathered in the lobby, the server should probably start the game. This is nothing special in itself, but we'll explain a few nice tricks that can be done at this point to make your life much easier.

Player scenes

In most games, each player will likely have its own scene. Remember that this is a multiplayer game, so in every peer you need to instance **one scene for each player connected to it**. For a 4 player game, each peer needs to instance 4 player nodes.

So, how to name such nodes? In Godot nodes need to have an unique name. It must also be relatively easy for a player to tell which nodes represent each player ID.

The solution is to simply name the *root nodes of the instanced player scenes as their network ID*. This way, they will be the same in every peer and RPC will work great! Here is an example:

```
remote func pre_configure_game():
    var selfPeerID = get_tree().get_network_unique_id()

    # Load world
    var world = load(which_level).instance()
    get_node("/root").add_child(world)

    # Load my player
    var my_player = preload("res://player.tscn").instance()
    my_player.set_name(str(selfPeerID))
    my_player.set_network_master(selfPeerID) # Will be explained later
    get_node("/root/world/players").add_child(my_player)

    # Load other players
    for p in player_info:
        var player = preload("res://player.tscn").instance()
        player.set_name(str(p))
```

```

get_node("/root/world/players").add_child(player)

# Tell server (remember, server is always ID=1) that this peer is done pre-
→configuring
rpc_id(1, "done_preconfiguring", selfPeerID)

```

Note: Depending on when you execute `pre_configure_game()` you may need to change any calls to `add_child()` to be deferred via `call_deferred()` as the SceneTree is locked while the scene is being created (e.g. when `_ready()` is being called).

Synchronizing game start

Setting up players might take different amount of time on every peer due to lag, different hardware, or other reasons. To make sure the game will actually start when everyone is ready, pausing the game until all players are ready can be very useful:

```

remote func pre_configure_game():
    get_tree().set_pause(true) # Pre-pause
    # The rest is the same as in the code in the previous section (look above)

```

When the server gets the OK from all the peers, it can tell them to start, as for example:

```

var players_done = []
remote func done_preconfiguring(who):
    # Here is some checks you can do, as example
    assert(get_tree().is_network_server())
    assert(who in player_info) # Exists
    assert(not who in players_done) # Was not added yet

    players_done.append(who)

    if (players_done.size() == player_info.size()):
        rpc("post_configure_game")

remote func post_configure_game():
    get_tree().set_pause(false)
    # Game starts now!

```

16.1.8 Synchronizing the game

In most games the goal of multiplayer networking is that the game runs synchronized on all the peers playing it. Besides supplying an RPC and remote member variable set implementation, Godot adds the concept network masters.

Network master

The network master of a node is the peer that has the ultimate authority over it.

When not explicitly set the network master is inherited from the parent node, which if not changed is always going to be the server (ID 1). Thus the server has authority over all nodes by default.

The network master can be set with the function `Node.set_network_master(id, recursive)` (recursive is true by default and means the network master is recursively set on all child nodes of the node as well).

Checking that a specific node instance on a peer is the network master for this node for all connected peers is done by calling `Node.is_network_master()`. This will return true when executed on the server and false on all client peers.

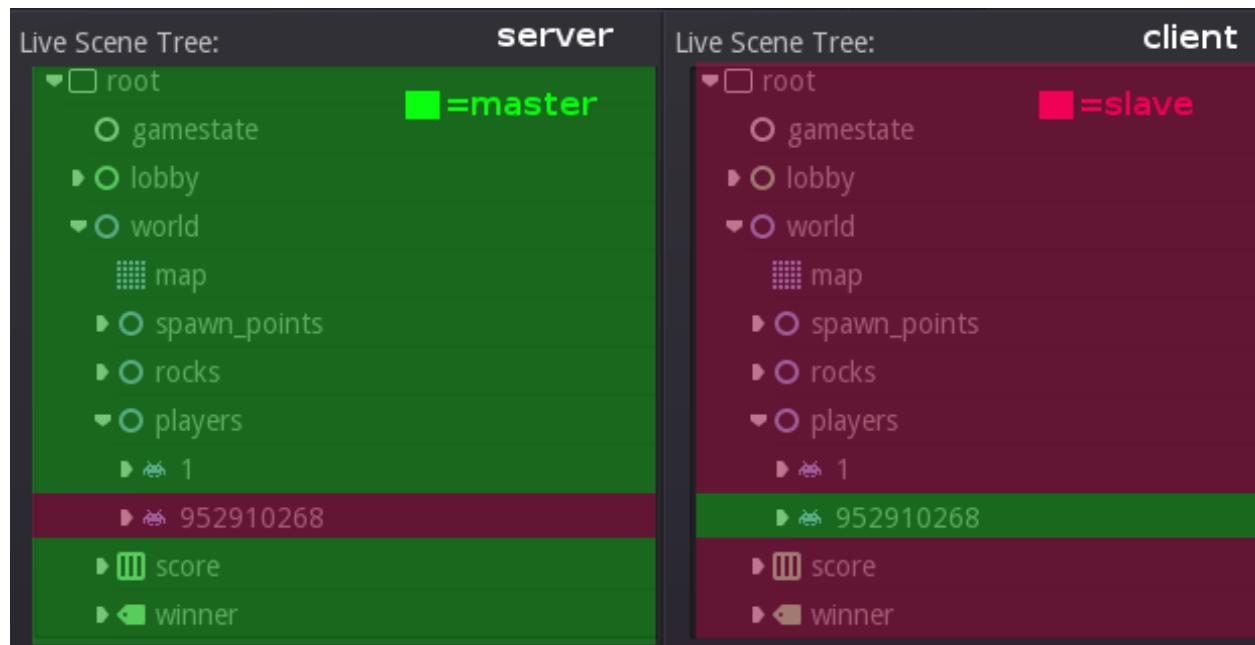
If you have paid attention to the previous example, it's possible you noticed that the local peer is set to have network master authority for their own player instead of the server:

```
[...]
# Load my player
var my_player = preload("res://player.tscn").instance()
my_player.set_name(str(selfPeerID))
my_player.set_network_master(selfPeerID) # The player belongs to this peer, it has_
→the authority
get_node("/root/world/players").add_child(my_player)

# Load other players
for p in player_info:
    var player = preload("res://player.tscn").instance()
    player.set_name(str(p))
    get_node("/root/world/players").add_child(player)
[...]
```

Each time this piece of code is executed on each peer, the peer makes itself master on the node it controls, and all other nodes remain as slaves with the server being their network master.

To clarify, here is an example of how this looks in the `bomber` demo:



Master and slave keywords

The real advantage of this model is when used with the `master/slave` keywords in GDScript (or their equivalent in C# and Visual Script). Similarly to the `remote` keyword, functions can also be tagged with them:

Example bomb code:

```
for p in bodies_in_area:
    if (p.has_method("exploded")):
        p.rpc("exploded", bomb_owner)
```

Example player code:

```
slave func stun():
    stunned = true

master func exploded(by_who):
    if (stunned):
        return # Already stunned

    rpc("stun")
    stun() # Stun myself, could have used sync keyword too.
```

In the above example, a bomb explodes somewhere (likely managed by whoever is master). The bomb knows the bodies in the area, so it checks them and checks that they contain an *exploded* function.

If they do, the bomb calls *exploded* on it. However, the *exploded* method in the player has a *master* keyword. This means that only the player who is master for that instance will actually get the function.

This instance, then, calls the *stun* function in the same instances of that same player (but in different peers), and only those which are set as slave, making the player look stunned in all the peers (as well as the current, master one).

Note that you could also send the *stun()* message only to a specific player by using *rpc_id(<id>, “exploded”, bomb_owner)*. This may not make much sense for an area-of-effect case like the bomb but in other cases, like single target damage.

```
rpc_id(TARGET_PEER_ID, "stun") # Only stun the target peer
```

16.2 HTTP client class

Here's an example of using the *HTTPClient* class. It's just a script, so it can be run by executing:

```
c:\godot> godot -s http_test.gd
```

It will connect and fetch a website.

```
extends SceneTree

# HTTPClient demo
# This simple class can do HTTP requests, it will not block but it needs to be polled

func _init():

    var err=0
    var http = HTTPClient.new() # Create the Client

    err = http.connect_to_host("www.php.net",80) # Connect to host/port
    assert(err==OK) # Make sure connection was OK

    # Wait until resolved and connected
    while ( http.get_status()==HTTPClient.STATUS_CONNECTING or http.get_
    ↪status()==HTTPClient.STATUS_RESOLVING):
        http.poll()
        print("Connecting...")
        OS.delay_msec(500)
```

```

assert( http.get_status() == HTTPClient.STATUS_CONNECTED ) # Could not connect

# Some headers

var headers=[
    "User-Agent: Pirulo/1.0 (Godot)",
    "Accept: */*"
]

err = http.request(HTTPClient.METHOD_GET,"/ChangeLog-5.php",headers) # Request a_
→page from the site (this one was chunked..)

assert( err == OK ) # Make sure all is OK

while (http.get_status() == HTTPClient.STATUS_REQUESTING):
    # Keep polling until the request is going on
    http.poll()
    print("Requesting...")
    OS.delay_msec(500)

assert( http.get_status() == HTTPClient.STATUS_BODY or http.get_status() ==_
→HTTPClient.STATUS_CONNECTED ) # Make sure request finished well.

print("response? ",http.has_response()) # Site might not have a response.

if (http.has_response()):
    # If there is a response..

    headers = http.get_response_headers_as_dictionary() # Get response headers
    print("code: ",http.get_response_code()) # Show response code
    print("**headers:\n",headers) # Show headers

    # Getting the HTTP Body

    if (http.is_response_chunked()):
        # Does it use chunks?
        print("Response is Chunked!")
    else:
        # Or just plain Content-Length
        var bl = http.get_response_body_length()
        print("Response Length: ",bl)

    # This method works for both anyway

    var rb = PoolByteArray() # Array that will hold the data

    while(http.get_status()==HTTPClient.STATUS_BODY):
        # While there is body left to be read
        http.poll()
        var chunk = http.read_response_body_chunk() # Get a chunk
        if (chunk.size()==0):
            # Got nothing, wait for buffers to fill a bit
            OS.delay_usec(1000)
        else:
            rb = rb + chunk # Append to read buffer

```

```

# Done!

print("bytes got: ",rb.size())
var text = rb.get_string_from_ascii()
print("Text: ",text)

quit()

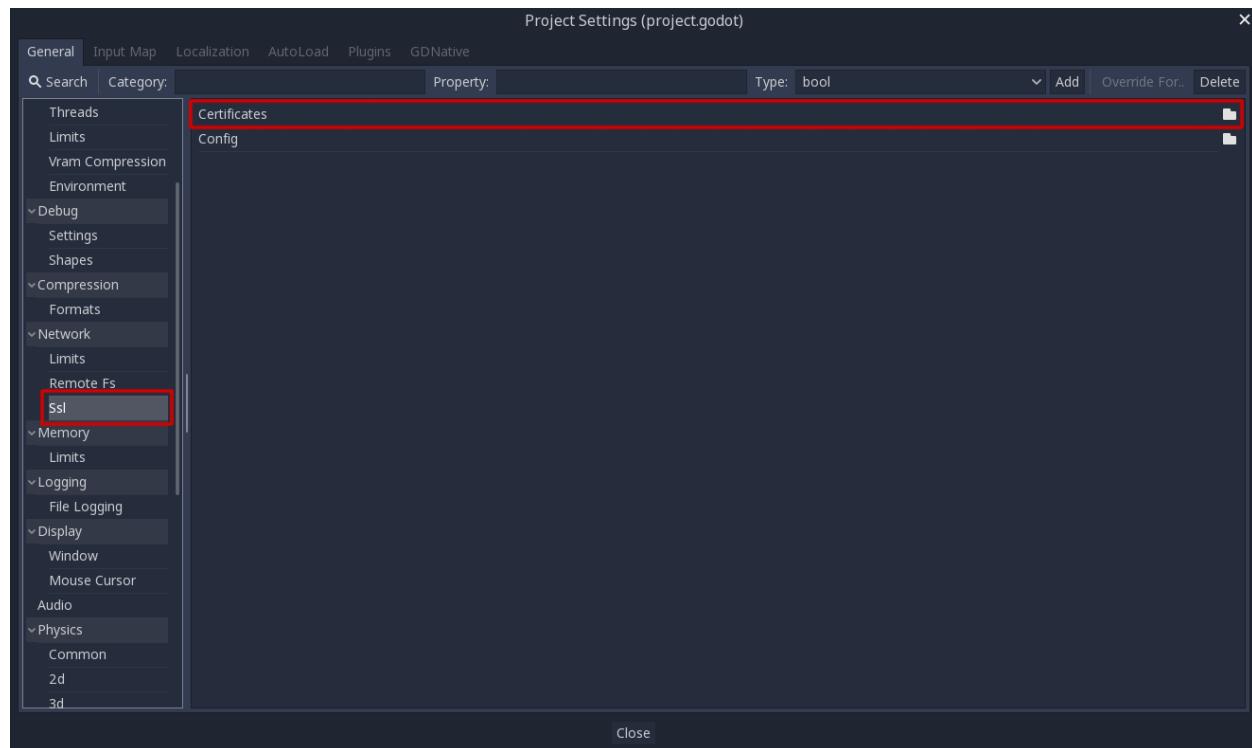
```

16.3 SSL certificates

16.3.1 Introduction

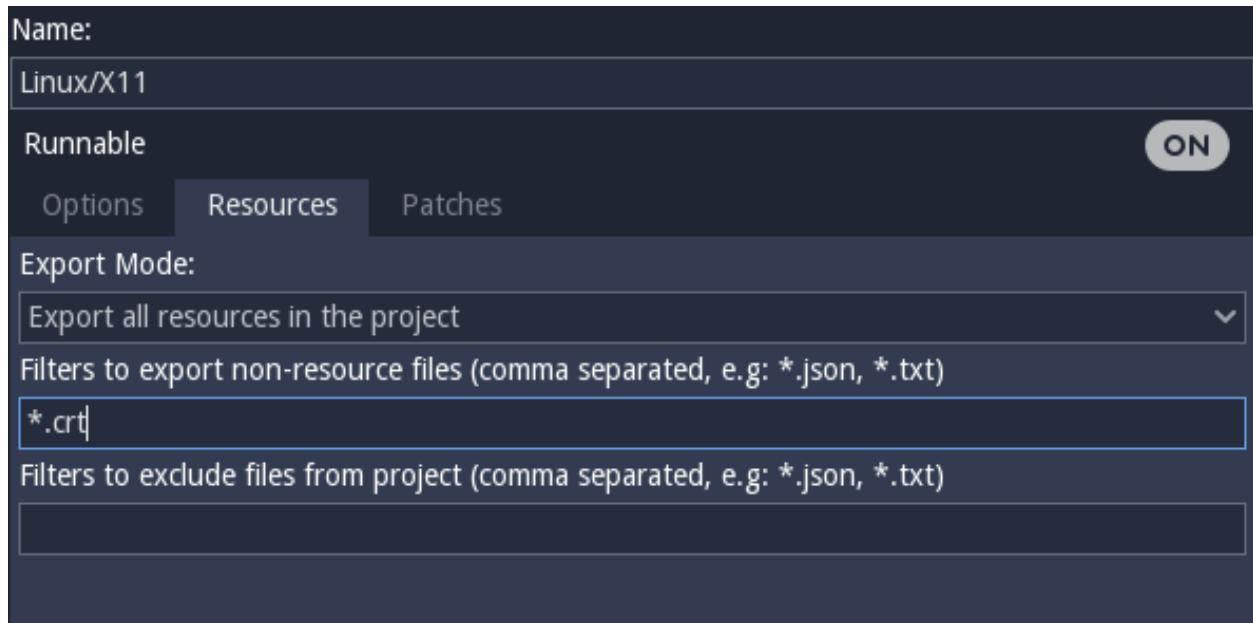
It is often desired to use SSL connections for communications to avoid “man in the middle” attacks. Godot has a connection wrapper, *StreamPeerSSL*, which can take a regular connection and add security around it. The *HTTPClient* class also supports HTTPS by using this same wrapper.

For SSL to work, certificates need to be provided. A .crt file must be specified in the project settings:



This file should contain any number of public certificates in http://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail format.

Of course, remember to add .crt as filter so the exporter recognizes this when exporting your project.



There are two ways to obtain certificates:

16.3.2 Approach 1: self signed cert

The first approach is the simplest, just generate a private and public key pair, and put the public pair in the .crt file (again, in PEM format). The private key should go to your server.

OpenSSL has [some documentation](#) about this. This approach also **does not require domain validation** nor requires you to spend a considerable amount of money in purchasing certificates from a CA.

16.3.3 Approach 2: CA cert

The second approach consists of using a certificate authority (CA) such as Verisign, Geotrust, etc. This is a more cumbersome process, but it's more “official” and ensures your identity is clearly represented.

Unless you are working with large companies or corporations, or need to connect to someone else’s servers (i.e., connecting to Google or some other REST API provider via HTTPS) this method is not as useful.

Also, when using a CA issued cert, **you must enable domain validation**, to ensure the domain you are connecting to is the one intended, otherwise any website can issue any certificate in the same CA and it will work.

If you are using Linux, you can use the supplied certs file, generally located in:

```
/etc/ssl/certs/ca-certificates.crt
```

This file allows HTTPS connections to virtually any website (i.e., Google, Microsoft, etc.).

Or just pick any of the more specific certificates there if you are connecting to a specific one.

CHAPTER 17

Asset Library

17.1 About the Asset Library

The Godot Asset Library, otherwise known as the AssetLib, is a repository of user-submitted Godot addons, scripts, tools and other resources, collectively referred to as assets. They're available to all Godot users for download directly from within the engine, but it can also be accessed at Godot's [official website](#).

On the surface the Asset Library might look and function similar to asset stores available for other engines, such as Unity's Asset Store, or Unreal Engine's Marketplace, where you can submit both freely-available assets, as well as paid, commercial ones. In addition, often times such assets are distributed under non-free, proprietary licenses, limiting what you can do with them.

The Asset Library is different - all assets are distributed free of charge, and under a host of open-source licenses (such as the MIT license, the GPL, and the Boost Software License). This makes the AssetLib more similar to the software repositories of a Linux distribution.

This set of pages will cover how to use the AssetLib (both from inside Godot, and on the website), how you can submit your own assets, and what the guidelines for submission are.

Please note that the AssetLib is relatively young - it may have various pain points, bugs and usability issues. As with all Godot projects, the code repository is available on [GitHub](#), where you can submit pull requests and issues, so please do not hesitate to visit it!

17.1.1 Frequently asked questions

Can paid assets be uploaded to the asset library?

Not to the official one, though in the future there might be other asset libraries which allow it. That said, you are allowed to monetize and sell Godot assets outside the Asset Library.

17.2 Using the AssetLib

17.2.1 On the website

Overview

As mentioned before, you can access the web frontend of the AssetLib on Godot's official website, and this is what it looks like when you first visit it:

The screenshot shows the Godot Asset Library interface. At the top, there is a header bar with tabs for "Header", "Godot Asset Library", and "Assets". A "Login/Register" button is also visible. Below the header is a search bar with dropdown menus for "Category" (set to "Any"), "Support level" (with checkboxes for "Testing", "Community", and "Official" all unchecked), "Godot version" (set to "Any"), and "Order by" (set to "Updated" with the "Reverse" checkbox unchecked). There is also a "Search for..." input field and a "Search" button. To the left of the main content area, there is a sidebar labeled "Assets / Search results" with a "Search bar + settings" section. The main content area displays a grid of 10 asset cards, each with a thumbnail, title, category, version, rating, and submission details. The assets listed are: 2D Hexagonal Map Demo, 2D Isometric Demo, 2D Kinematic Character Demo, 2D Kinematic Collision Demo, 2D Light Masks Demo, 2D Lights and Shadows Demo, 2D Navigation Demo, 2D Particles Demo, 2D Physics Platformer Demo (Rigidbody), and 2D Platformer Demo (Kinematicbody). Each card includes a "Demos" button, a "3.0" button, and an "Official" button. Below the grid is a pagination bar with buttons for "«", "1", "2", "3", "...", "15", and "»". A note indicates "10 items per page, 147 items total." and a dropdown for "Change items per page to: 10 | 20 | 50 | 100 | 200 | 500". A "View Pending" button is also present. At the bottom, there is a footer bar with "Copyright © 2017 The Godot Engine community - MIT Licensed.", "Currently running version 9cd0e06.", and links to "GitHub repository" and "Godot Engine homepage".

At the top you see the **header** which takes you to various other parts of the AssetLib - at the moment it's empty as we are not logged in.

Searching

In the center is the **search bar + settings** section, and the **assets** section below it - this lets you filter out certain kinds of assets based on a host of criteria. These include the asset **category** (such as 2D tools, scripts and demos), **engine version** they are intended for, **sorting order** (by update date, by name, etc.) and **support level**.

While most other filter settings should be fairly self-explanatory, it's worth going over what "support level" means in the Asset Library. Currently there are three support levels, and each asset can belong to only one.

Official assets are created and maintained by the official Godot Engine developers. Currently, these include the official engine demos, which showcase how various areas of the engine work.

Community assets are submitted and maintained by the members of the Godot community.

Testing assets are works-in-progress, and may contain bugs and usability issues. They are not recommended for use in serious projects, but you are encouraged to download, test them, and submit issues to the original authors.

You can mix and match any of the search filters and criteria, and upon clicking the Search button, receive the list of all assets in the Library that match them.

The screenshot shows the Godot AssetLib search interface. At the top, there are filters for 'Category' (set to 'Scripts'), 'Support level' (checkboxes for 'Testing' and 'Community' checked), 'Official' (checkbox checked), 'Godot version' (set to '3.0'), 'Order by' (set to 'Updated'), and 'Reverse' (checkbox checked). A search bar contains the query 'Godot'. Below the filters, the results for 'Godot' are listed:

- Godot NExt (3.0) 3.0**: Scripts, 3.0, Community. Submitted by user willnationsdev; MIT, 2017-09-18. Rating: ★★★★★. Preview image shows a network graph.
- godot-behavior-tree-plugin 1.0.4**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-12. Rating: ★★★★★. Preview image shows a blue gear-like icon.
- godot-quest-system 1.0.1**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-12. Rating: ★★★★★. Preview image shows a blue gear-like icon.
- Godot Screen Manager 1.0.1**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-18. Rating: ★★★★★. Preview image shows a blue gear-like icon.
- Godot Object Pool 1.0.8**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-12. Rating: ★★★★★. Preview image shows a blue gear-like icon.
- Godot Finite State Machine 1.0.4**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-12. Rating: ★★★★★. Preview image shows a blue gear-like icon.
- godot-hotbars 1.0.0**: Scripts, 3.0, Community. Submitted by user brandonlamb; MIT, 2017-12-18. Rating: ★★★★★. Preview image shows a blue gear-like icon.

At the bottom left, there are navigation buttons: «, 1 (highlighted in blue), and ».

Note that the search results are not updated in real-time, so you will have to re-submit the search query each time you change the query settings.

Breakdown of an asset

Now let's take a look at what an asset's page looks like and what it contains.

The screenshot shows the Godot AssetLib page for the asset '2D Hexagonal Map Demo'. The asset details are as follows:

- Thumbnail/icon**: A green hexagonal icon labeled '1'.
- Name**: '2D Hexagonal Map Demo'.
- Version**: '3.0-2183e'.
- Average rating**: ★★4★☆☆ (unimplemented).
- Category**: 'Demos'.
- Godot version**: '3.0'.
- Official**: Yes.
- Submitted by**: 'Godot Engine' (user ID 7).
- Date submitted**: '2018-01-17'.
- Description**: 'Very simple demo showing Hexagonal TileMap and TileSet.'

Below the details, there are several buttons:

- View files**
- Download 10**
- Submit an issue**
- Recent Edits**

Below the buttons, the SHA256 hash is shown: 'd506f7844886cf52d77bc1978aece27989a354276960857b7c0f564c7e0f0fe'.

At the bottom, there is a preview image of a hexagonal map with green terrain and small trees, labeled '12'.

1. Asset's thumbnail/icon.
2. Asset's name.
3. Current version number of the asset.
4. Asset's average rating, displayed in stars. (This is currently unimplemented.)
5. Asset's category, Godot version, and support status.
6. Asset's original author/submitter.
7. The license the asset is distributed under.

8. The date of the asset's latest edit/update.
9. A textual description of the asset.
10. Links related to the asset (download link, file list, issue tracker).
11. A SHA-256 hash of the asset, for download validation purposes.
12. Images and videos showcasing the asset.

Registering and logging in

In order to upload assets to the AssetLib, you need to be logged in, and to do that, you need a registered user account. In the future, this may also give you access to other features, such as commenting on or rating the existing assets. You do *not* need to be logged in to browse and download the assets.

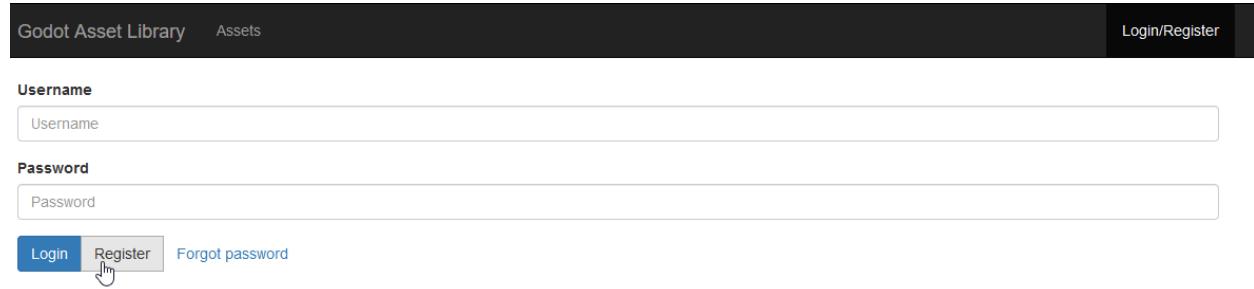
The login/registration page can be accessed from the AssetLib header.



From here, you can register your account, which requires a valid email address, a username, and a (preferably strong) password.

A screenshot of the registration form. It has fields for "Username" and "Email", both with placeholder text "Username" and "Email". Below them is a "Password" field with placeholder text "Password". At the bottom are two buttons: "Login" and "Register", with "Register" being highlighted with a blue background. To the right of the form is a "Login/Register" button in the header.

Then, you can use your username and password to log in.



This will change the look of the AssetLib header. Now you get access to a handful new functions:

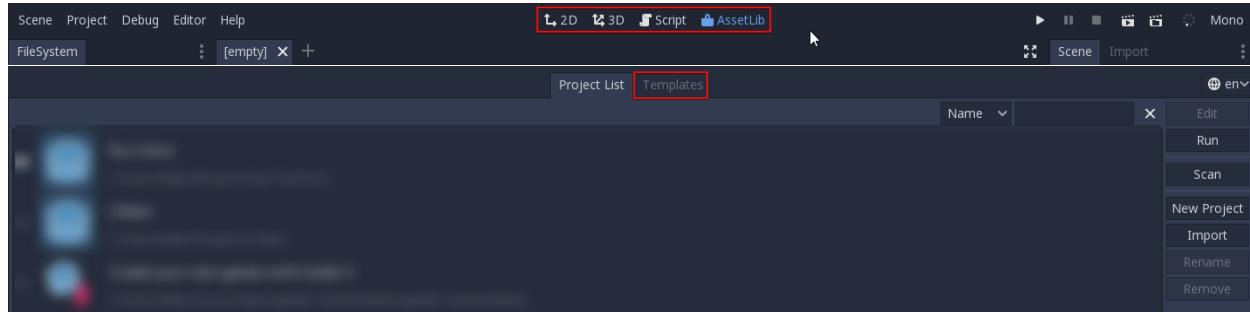
- The feed, which shows a list of status updates on your submitted assets (and possibly more in the future).
- A list of your uploaded assets.
- The ability to submit new assets.



You can learn how to submit assets to the Library, and what the asset submission guidelines are, in the next part of this tutorial, *:Submitting to the Asset Library*.

17.2.2 In the editor

You can also access the AssetLib directly from Godot. It is available from two places - in the Project Manager's Templates tab, and inside of a project, from the workspaces list.



Click on it, and Godot will fetch info about the assets from the AssetLib. Once it's finished, you will see a window similar to what the AssetLib website looks like, with some differences:

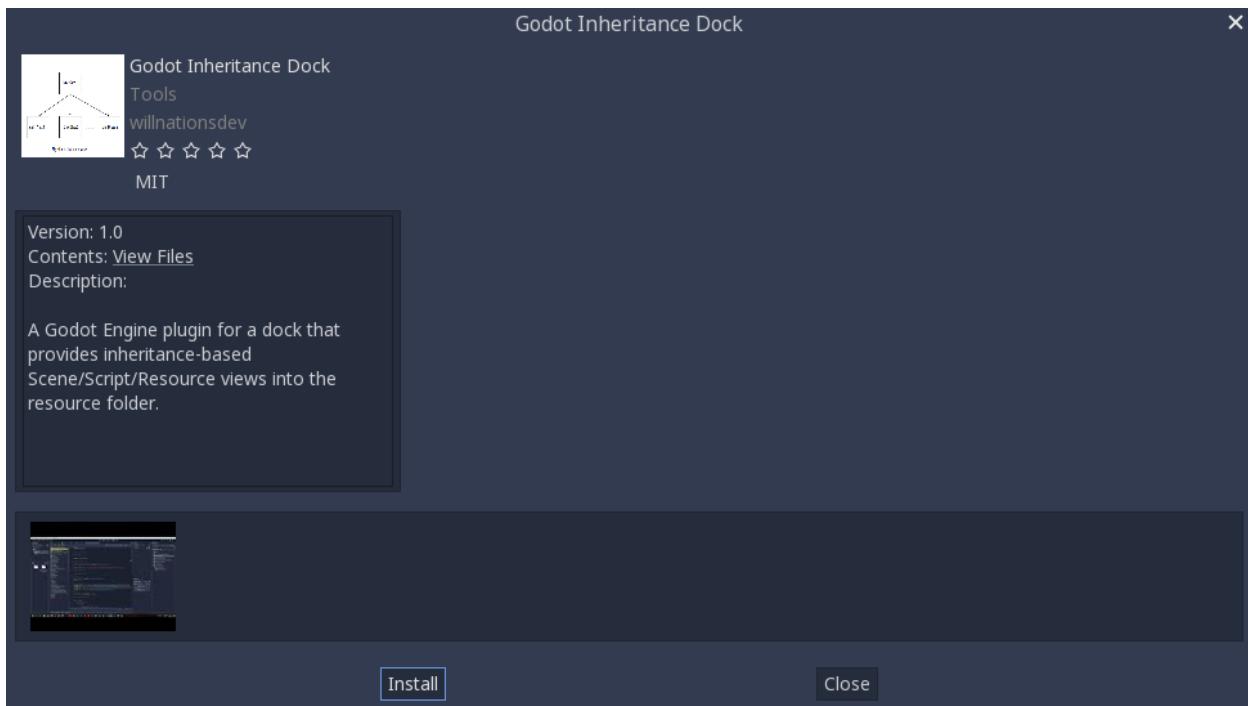
Asset Name	Type	Author	Rating	Licence
Godot Object Pool	Scripts	brandonlamb	5 stars	MIT
Godot Finite State Machine	Scripts	brandonlamb	5 stars	MIT
Godot NExt (3.0)	Scripts	willnationsdev	5 stars	MIT
Noise Shaders	Shaders	curly-brace	5 stars	MIT
Fractal Shaders	Shaders	Chaosus	5 stars	MIT
FSM (Finite State Machine)	Tools	kubecz3k	5 stars	MIT
WAD Level Loader	Scripts	Chaosus	5 stars	MIT
PCKManager	Tools	MrJustreborn	5 stars	MIT
godot-behavior-tree-plugin	Scripts	brandonlamb	5 stars	
godot-quest-system	Scripts	brandonlamb	5 stars	

Similarly to the web version of the AssetLib, here you can search for assets by category, name, and sort them by factors such as name or edit date.

Notably, you can only fetch assets for the current version of Godot you are running. Also, you can only download Projects, Demos and Templates from the Project Manager view of the AssetLib, while Addons (tools, scripts, materials etc.) can only be downloaded from the in-project AssetLib. In addition, unlike when using the web frontend, the search results are updated in real-time (you do not have to press Search after every change to your search query for the changes to take place).

In the future, you will be able to choose a different AssetLib provider to fetch assets from (using the Site dropdown menu), however currently only the official [Godot website](#) version of the AssetLib is supported, as well as the version that may be running on your local machine's web server (the localhost option).

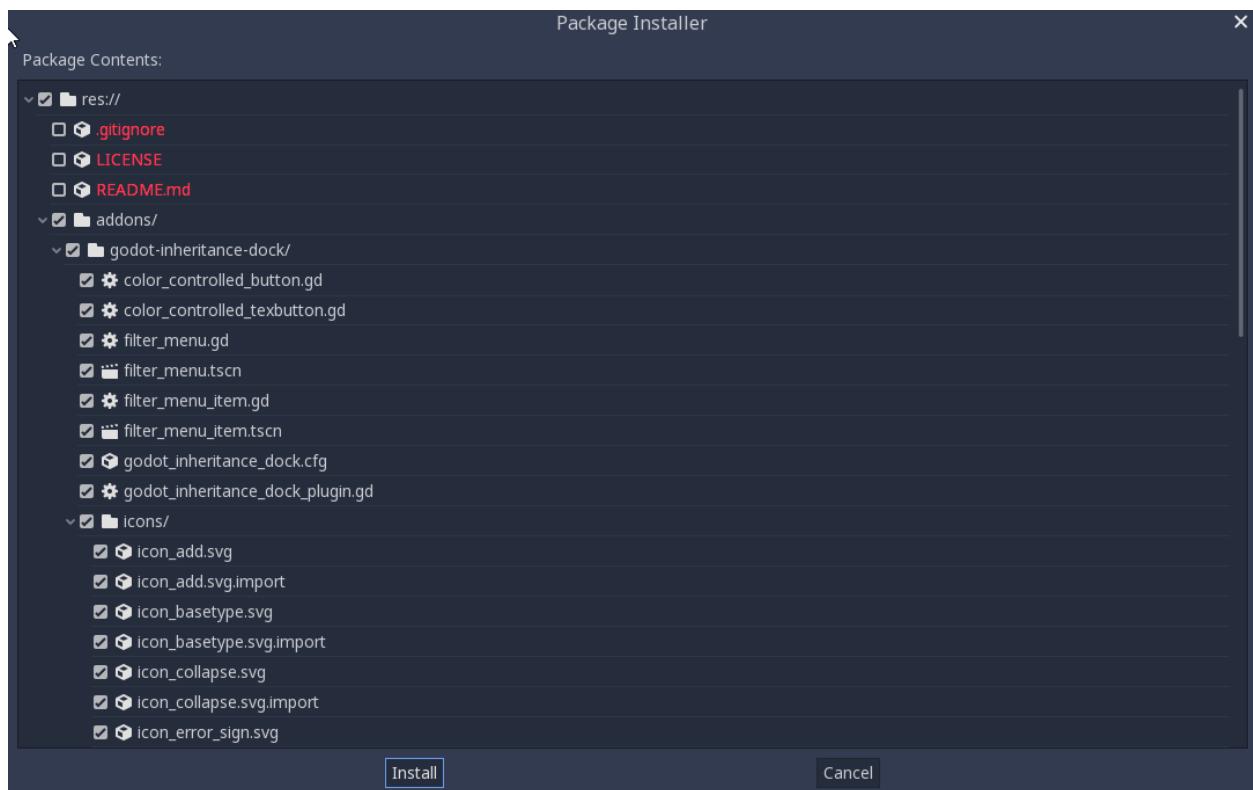
When you click on an asset, you will see more information about it.



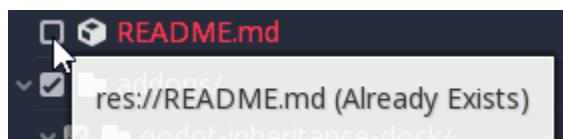
If you click on the Install button, Godot will fetch an archive of the asset, and will track download progress of it at the bottom of the editor window. If the download fails, you can retry it using the Retry button.



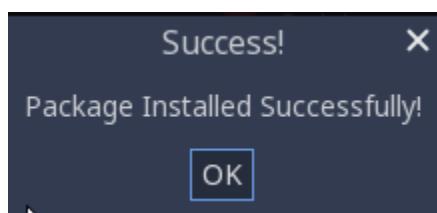
When it finishes, you can proceed to install it using the Install button. This will bring up the Package Installer window.



Here you can see a list of all the files that will be installed. You can tick off any of the files that you do not wish to install, and Godot will also inform you about any problems with files that it cannot install. These files will be shown in red, and hovering over them will show you a message stating why it cannot be installed.



Once you are done, you can press the Install button, which will unzip all the files in the archive, and import any assets contained therein, such as images or 3D models. Once this is done, you should see a message stating that the Package installation is complete.



You may also use the Import button to import asset archives obtained elsewhere (such as downloading them directly from the AssetLib web frontend), which will take you through the same package installation procedure as with the assets downloaded directly via Godot that we just covered.

17.3 Submitting to the Asset Library

17.3.1 Introduction

This tutorial aims to serve as both a guide on how you can submit your own assets to the Godot Asset Library and share them with the Godot community.

As mentioned in the *Using the AssetLib* document, in order to be able to submit assets to the AssetLib, you need to have a registered account, and be logged in.

17.3.2 Submitting

Once you are logged in, you will be able to head over to the “Submit Assets” page of the AssetLib, which will look like this:

The screenshot shows the 'Submit Asset' form on the Godot Asset Library website. The form fields are as follows:

- Asset Name ***: A text input field labeled "Asset Name" with the placeholder "The name of the asset".
- Category ***: A dropdown menu labeled "2D Tools" with the placeholder "The category the asset belongs to".
- Godot version ***: A dropdown menu labeled "Unknown" with the placeholder "The version of Godot the asset works with".
- Version ***: A text input field labeled "Version" with the placeholder "The version of the asset".
- Repository host ***: A dropdown menu labeled "GitHub" with the placeholder "We need to know where your repository is hosted in order to compute the final download URL. If your Git repository provider is missing, you might like to open an issue about it".
- Repository URL ***: A text input field with the placeholder "The url to browse the asset's files (webpage). Should be similar to <https://github.com/<user>/<project>> (Might be different depending on your "provider")".
- Issues URL**: A text input field with the placeholder "The url to issues list of the asset. Should be similar to <https://github.com/<user>/<project>/issues> (Might be different depending on your "provider"). If you use your provider's issue list, you".

While it may look like a lot (and there is more as you scroll down), each field is described in terms of what you should put in. We will nonetheless go over what is required in the submission form here as well.

- **Asset Name:** The name of your asset. Should be a unique, descriptive title of what your asset is.
- **Category:** The category that your asset belongs to, and will be shown in search results. The category is split into **Addons** and **Projects**. In-editor, assets of the Project type (Templates, Demos, Projects) only show up when viewing the AssetLib from the Project Manager, while assets of the Addon type will only be visible from inside a project.
- **Godot version:** The version of the engine that the asset works with. Currently it's not possible to have a single asset entry contain downloads for multiple engine versions, so you may need to re-submit the asset multiple times, with an entry for each Godot version it supports. This is particularly important when dealing with major versions of the engine, such as Godot 2.x and Godot 3.x.
- **Version:** The version number of the asset. While you are free to choose and use any versioning scheme that you like, you may want to look into something such as **SemVer** if you want your asset's versioning scheme

to be clear and consistent. Note that there is also an internal version number, incremented every time the asset download URL is changed/updated.

- **Repository host:** Assets uploaded to the AssetLib are not hosted on it directly, and instead point to repositories hosted on third-party Git providers, such as GitHub, GitLab or Bitbucket. This is where you choose which provider your asset uses, so the final download link can be computed.
- **Repository URL:** The URL to your asset's files/webpage. This will vary based on your choice of provider, but it should look similar to <https://github.com/<user>/<project>>.
- **Issues URL:** The URL to your asset's issue tracker. Again, this will differ from repository host to repository host, but will likely look similar to <https://github.com/<user>/<project>/issues>. You may leave this field empty if you use your provider's issue tracker, and it's part of the same repository.
- **Download Commit/Tag:** The commit or tag of the asset. For example, `b1d3172f89b86e52465a74f63a74ac84c491d3e1` or `v1.0`. From this, the actual download URL will be computed.
- **Icon URL:** The URL to your asset's icon (which will be used as a thumbnail in the AssetLib search results, and on the asset's page). Should be an image in either the PNG or JPG format.
- **License:** The license under which you are distributing the asset. The list includes a variety of free and open-source software licenses, such as GPL (v2 and v3), MIT, BSD and Boost Software License. You can visit OpenSource.org for a detailed description of each of the listed licenses.
- **Description:** Finally, you can use the Description field for a textual overview of your asset, its features and behavior, a changelog, et cetera. In the future, formatting with Markdown will be supported, but currently, your only option is plain text.

You may also include up to three video and/or image previews, which will be shown at the bottom of the asset page. Use the “Enable” checkbox on each of the preview submission boxes to enable them.

- **Type:** Either an image, or a video.
- **Image/YouTube URL:** Either a link to the image, or to a video, hosted on YouTube.
- **Thumbnail URL:** An URL to a image that will be used as a thumbnail for the preview. This option will be removed eventually, and thumbnails will be automatically computed instead.

Once you are done, hit Submit. Your asset will be entered into the pending queue, which you can visit on the AssetLib [here](#). The approval process is manual and may take up to a few days for your addon to be accepted (or rejected), so please be patient! You will be informed when your asset is reviewed. If it was rejected, you will be told why that may have been, and you will be able to submit it again with the appropriate changes. You may have some luck accelerating the approval process by messaging the moderators/assetlib reviewers on IRC (the #godotengine-atelier channel on Freenode), or the official Discord server.

17.3.3 Submission guidelines

[TODO]

CHAPTER 18

VR

18.1 An AR/VR Primer for Godot

This tutorial gives you a springboard into the world of AR and VR in the Godot game engine.

A new architecture was introduced in Godot 3 called the AR/VR Server. On top of this architecture specific implementations are available as interfaces most of which are plugins based on GDNative. This tutorial focuses purely on the core elements abstracted by the core architecture. This architecture has enough features for you to create an entire VR experience that can then be deployed for various interfaces. However each platform often has some unique features that are impossible to abstract. Such features will be documented on the relevant interfaces and fall outside of the scope of this primer.

18.1.1 AR/VR Server

When Godot starts each available interface will make itself known to the AR/VR server. GDNative interfaces are setup as singletons, as long as they are added to the list of GDNative singletons in your project they will make themselves known to the server.

You can use the function `get_interfaces` to return a list of available interfaces but for this tutorial we're going to use the *native mobile VR interface* in our examples. This interface is a very simple and straight forward implementation that uses the 3DOF sensors on your phone for orientation and outputs a stereo scopic image to screen. It is also available in the Godot core and outputs to screen on desktop which makes it ideal for prototyping or a tutorial such as this one.

To enable an interface you execute the following code:

```
var arvr_interface = ARVRServer.find_interface("Native mobile")
if arvr_interface and arvr_interface.initialize():
    get_viewport().arvr = true
```

This code finds the interface we wish to use, initializes it and if that is successful binds the main viewport to the interface. This last step gives some control over the viewport to the interface which automatically enables things like stereo scopic rendering on the viewport.

For our mobile vr interface, and any interface where the main input is directly displayed on screen, the main viewport needs to be the viewport where arvr is set to true. But for interfaces that render on an externally attached device you can use a secondary viewport. In this later case a viewport that shows its output on screen will show an undistorted version of the left eye while showing the fully processed stereo scopic output on the device.

Finally you should only initialize an interface once, switching scenes and reinitializing interfaces will just introduce a lot of overhead. If you want to turn the headset off temporarily just disable the viewport or set arvr to false on the viewport. In most scenarios though you wouldn't disable the headset once you're in VR, this can be very disconcerting to the gamer.

18.1.2 New AR/VR Nodes

Three new node types have been added for supporting AR and VR in Godot and one additional node type especially for AR. These are:

- *ARVROrigin* - our origin point in the world
- *ARVRCamera* - a special subclass of the camera which is positionally tracked
- *ARVRCController* - a new spatial class that tracks the location of a controller
- *ARVRAnchor* - an anchor point for an AR implementation mapping a real world location into your virtual world

The first two must exist in your scene for AR/VR to work and this tutorial focuses purely on them.

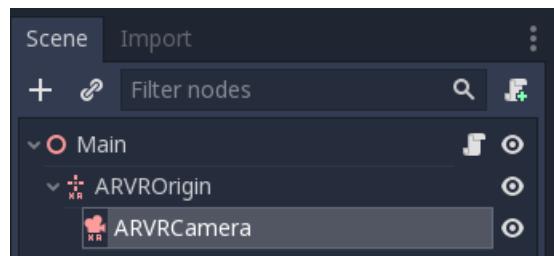
ARVROrigin is an important node, you must have one and only one of these somewhere in your scene. This node maps the center of your real world tracking space to a location in your virtual world. Everything else is positionally tracked in relation to this point. Where this point lies exactly differs from one implementation to another but the best example to understand how this node works is to take a look at a room scale location. While we have functions to adjust the point to center it on the player by default the origin point will be the center location of the room you are in. As you physically walk around the room the location of the HMD is tracked in relation to this center position and the tracking is mirror in the virtual world.

To keep things simple, when you physically move around your room the ARVR Origin point stays where it is, the position of the camera and controllers will be adjusted according to your movements. When you move through the virtual world, either through controller input or when you implement a teleport system it is the origin point which you will have to adjust the position of.

ARVRCamera is the second node that must always be a part of your scene and it must always be a child node of your origin node. It is a subclass of Godots normal camera however its position is automatically updated each frame based on the physical orientation and position of the HMD. Also due to the precision required for rendering to an HMD or rendering an AR overlay over a real world camera most of the standard camera properties are ignored. The only properties of the camera that are used are the near and far plane settings. The FOV, aspect ratio and projection mode are all ignored.

Note that for our native mobile VR implementation there is no positional tracking, only the orientation of the phone and by extension the HMD is tracked. This implementation artificially places the camera at a height (Y) of 1.85.

Conclusion, your minimum setup in your scene to make AR or VR work should look like this:



And that's all you need to get started. Obviously you need to add something more into your scene so there is something to see but after that you can export the game to your phone of choice, pop it into a viewer and away you go.

18.1.3 Other things to consider

There are a few other subjects that we need to briefly touch upon in this primer that are important to know.

The first are our units. In normal 3D games you don't have to think a lot about units. As long as everything is at the same scale a box sized 1 unit by 1 unit by 1 unit can be any size from a cube you can hold in your hand to something the size of a building. In AR and VR this changes because things in your virtual world are mapped to things in the real world. If you step 1 meter forward in the real world, but you only move 1 cm forward in your virtual world, you have a problem. The same with the position of your controllers, if they don't appear in the right relative space it breaks the immersion for the player. Most VR platforms including our AR/VR Server assumes that 1 unit = 1 meter. The AR/VR server however has a property that for convenience is also exposed on the ARVROrigin node called world scale. For instance setting this to a value of 10 it changes our coordinate system so 10 units = 1 meter.

Performance is another thing that needs to be carefully considered. Especially VR taxes your game a lot more than most people realise. For mobile VR you have to be extra careful here but even for desktop games there are three factors that make life extra difficult:

- You are rendering stereoscopic, two for the price of one. While not exactly doubling the work load and with things in the pipeline such as supporting the new MultiView OpenGL extension in mind, there still is an extra workload in rendering images for both eyes
- A normal game will run acceptable on 30fps and ideally manages 60fps. That gives you a big range to play with between lower end and higher end hardware. For any HMD application of AR or VR however 60fps is the absolute minimum and you really should target your games to running 90fps stably to ensure your users don't get motion sickness right off the bat.
- The high FOV and related lens distortion effect require many VR experiences to render at double the resolution. Yes a VIVE may only have a resolution of 1080x1200 per eye, we're rendering each eye at 2160x2400 as a result. This is less of an issue for most AR applications.

All in all, the workload your GPU has in comparison with a normal 3D game is a fair amount higher. While things are in the pipeline to improve this such as MultiView and foveated rendering these aren't supported on all devices. This is why you see many VR games using a more art style and if you pay close attention to those VR games that go for realism, you'll probably notice they're a bit more conservative on the effects or use some good old optical trickery.

CHAPTER 19

Plugins

19.1 Editor plugins

19.1.1 Making Plugins

About Plugins

A plugin is a great way to extend the editor with useful tools. It can be made entirely with GDScript and standard scenes, without even reloading the editor. Unlike modules, you don't need to create C++ code nor recompile the engine. While this makes plugins not as powerful, there's still a lot of things you can do with them. Note that a plugin is not different from any scene you already can make, except that it is made via script to add functionality.

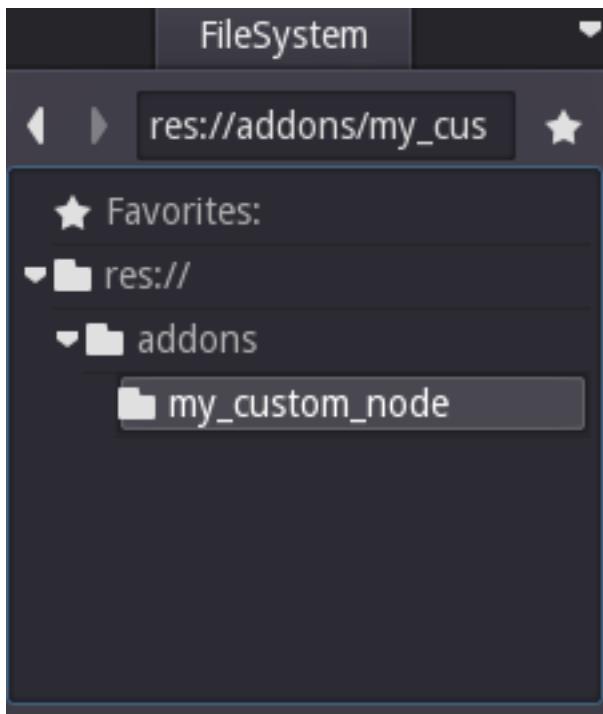
This tutorial will guide you through the creation of two simple plugins so you can understand how they work and be able to develop your own. The first will be a custom node that you can add to any scene in the project and the other will be a custom dock added to the editor.

Creating a plugin

Before starting, create a new empty project wherever you want. This will serve as base to develop and test the plugins.

The first thing you need to do is to create a new plugin that the editor can understand as such. For that you need two files: `plugin.cfg` for the configuration and a custom GDScript with the functionality.

Plugins have a standard path like `addons/plugin_name` inside the project folder. So create the folder `my_custom_node` inside `addons`. So you'll have a directory structure like this:



To make the `plugin.cfg` file, open your favorite text editor with a blank file. Godot is not able (yet) to open text files besides scripts, so this must be done in an external editor. Add the following structure to your `plugin.cfg`:

```
[plugin]
name="My Custom Node"
description="A custom node made to extend the Godot Engine."
author="Your Name Here"
version="1.0"
script="custom_node.gd"
```

This is a simple `ini` file with metadata about your plugin. You need to set up the name and description so users can understand what it does. Add your own name so you can be properly credited. A version number is useful so users can see if they have an outdated version (if you are unsure on how to come up with the version number, check [SemVer](#)). And finally a main script file to load when your plugin is active.

The script file

Open the script editor (F3) and create a new GDScript file called `custom_node.gd` inside the `my_custom_node` folder. This script is special and it has two requirements: it must be a `tool` script and it has to inherit from `EditorPlugin`.

It's important to deal with initialization and clean-up of resources. So a good practice is to use the virtual function `_enter_tree()` to initialize your plugin and `_exit_tree()` to clean it up. You can delete the default GDScript template from your file and replace it with the following structure:

```
tool
extends EditorPlugin

func _enter_tree():
    # Initialization of the plugin goes here
    pass
```

```
func _exit_tree():
    # Clean-up of the plugin goes here
    pass
```

This is a good template to use when devising new plugins.

A custom node

Sometimes you want a certain behavior in many nodes. Maybe a custom scene or control that can be reused. Instancing is helpful in a lot of cases but sometimes it can be cumbersome, especially if you're using it between many projects. A good solution to this is to make a plugin that adds a node with a custom behavior.

To create a new node type, you can avail of the function `add_custom_type()` from the `EditorPlugin` class. This function can add new types to the editor, be it nodes or resources. But before you can create the type you need a script that will act as the logic for the type. While such script doesn't need to have the `tool` keyword, it is interesting to use it so the user can see it acting on the editor.

For this tutorial, we'll create a simple button that prints a message when clicked. And for that we'll need a simple script that extends from `Button`. It could also extend `BaseButton` if you prefer:

```
tool
extends Button

func _enter_tree():
    connect("pressed", self, "clicked")

func clicked():
    print("You clicked me!")
```

That's it for our basic button. You can save this as `button.gd` inside the plugin folder. You'll also need a 16x16 icon to show in the scene tree. If you don't have one, you can grab the default one from the engine:



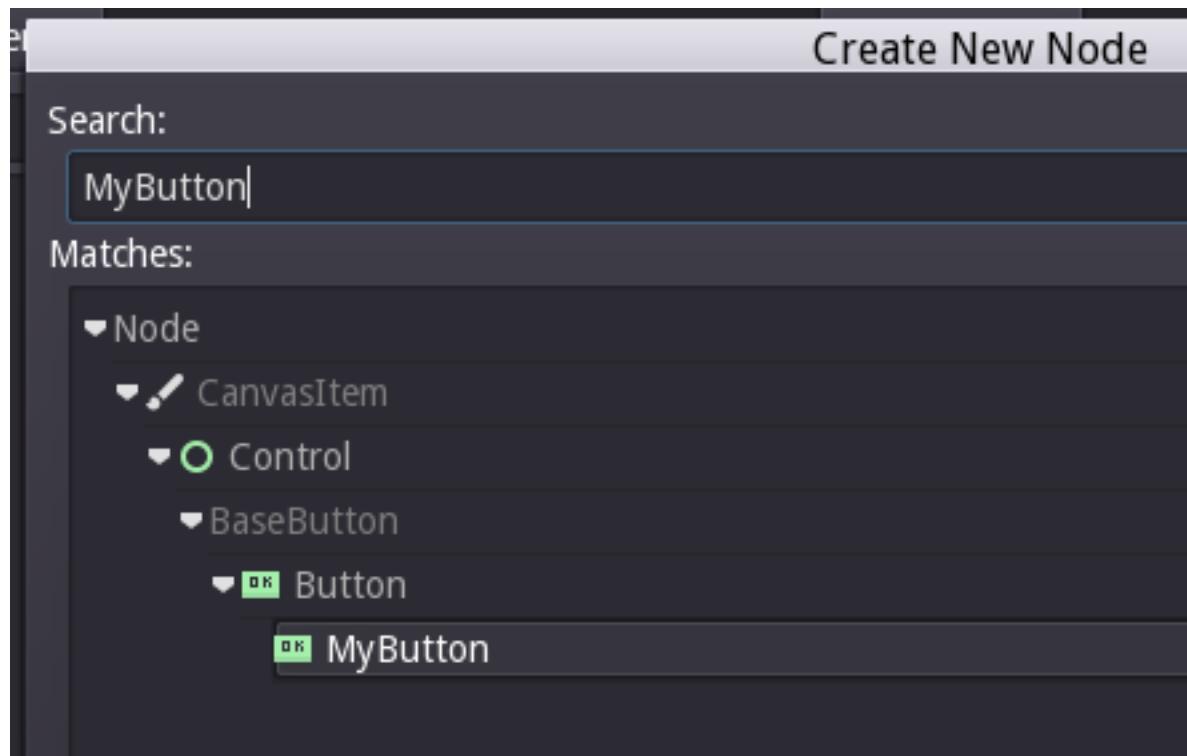
Now we need to add it as a custom type so it shows on the Create New Node dialog. For that, change the `custom_node.gd` script to the following:

```
tool
extends EditorPlugin

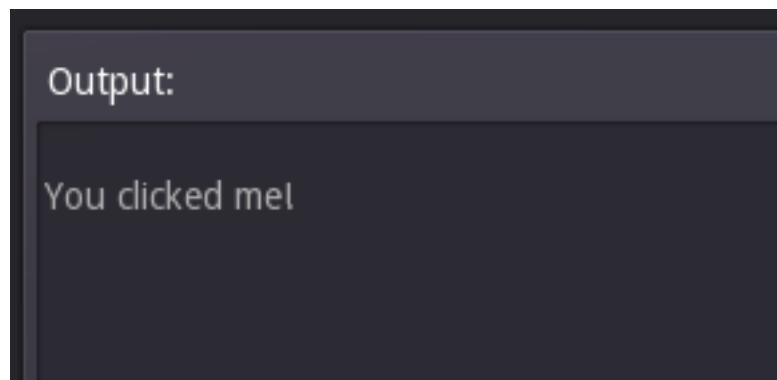
func _enter_tree():
    # Initialization of the plugin goes here
    # Add the new type with a name, a parent type, a script and an icon
    add_custom_type("MyButton", "Button", preload("button.gd"), preload("icon.png"))

func _exit_tree():
    # Clean-up of the plugin goes here
    # Always remember to remove it from the engine when deactivated
    remove_custom_type("MyButton")
```

With that done, the plugin should already be available in the plugin list at Project Settings. So activate it and try to add a new node to see the result:



When you add the node, you can see that it already have the script you created attached to it. Set a text to the button, save and run the scene. When you click the button, you can see a text in the console:



A custom dock

Maybe you need to extend the editor and add tools that are always available. An easy way to do it is to add a new dock with a plugin. Docks are just scenes based on control, so how to create them is not far beyond your knowledge.

The way to start this plugin is similar to the custom node. So create a new `plugin.cfg` file in the `addons/my_custom_dock` folder. And then with your favorite text editor add the following content to it:

```
[plugin]

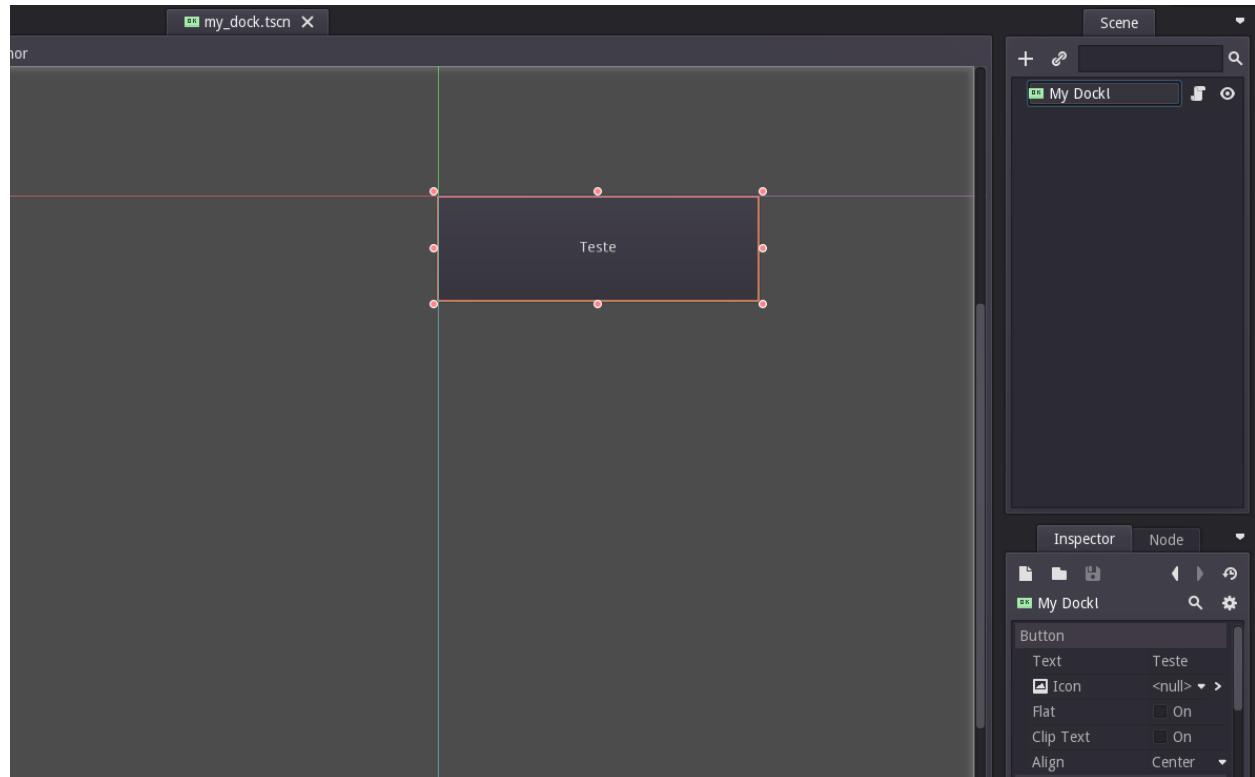
name="My Custom Dock"
description="A custom dock made so I can learn how to make plugins."
author="Your Name Here"
version="1.0"
```

```
script="custom_dock.gd"
```

Then create the script `custom_dock.gd` in the same folder. Fill with the *template we've seen before* to get a good start.

Since we're trying to add a new custom dock, we need to create the contents of such dock. This is nothing more than a standard Godot scene. So you can create a new scene in the editor and start creating it.

For an editor dock, it is mandatory that the root of the scene is a *Control* or one of its child classes. For this tutorial, you can make a single button. The name of the root node will also be the name that appears on the dock tab, so be sure to put a descriptive but short one. Don't forget to add a text to your button.



Save this scene as `my_dock.tscn`.

Now you need to grab that scene you just created and add it as a dock in the editor. For this you can rely on the function `add_control_to_dock()` from the `EditorPlugin` class.

The code is very straightforward, you just need to select a dock position to add it and have a control to add (which is the scene you just created). It is also very important that you remember to **remove the dock** when the plugin is deactivated. The code can be like this:

```
tool
extends EditorPlugin

var dock # A class member to hold the dock during the plugin lifecycle

func _enter_tree():
    # Initialization of the plugin goes here
    # First load the dock scene and instance it:
    dock = preload("res://addons/my_custom_dock/my_dock.tscn").instance()

    # Add the loaded scene to the docks:
```

```

add_control_to_dock( DOCK_SLOT_LEFT_UL, dock)
# Note that LEFT_UL means the left of the editor, upper-left dock

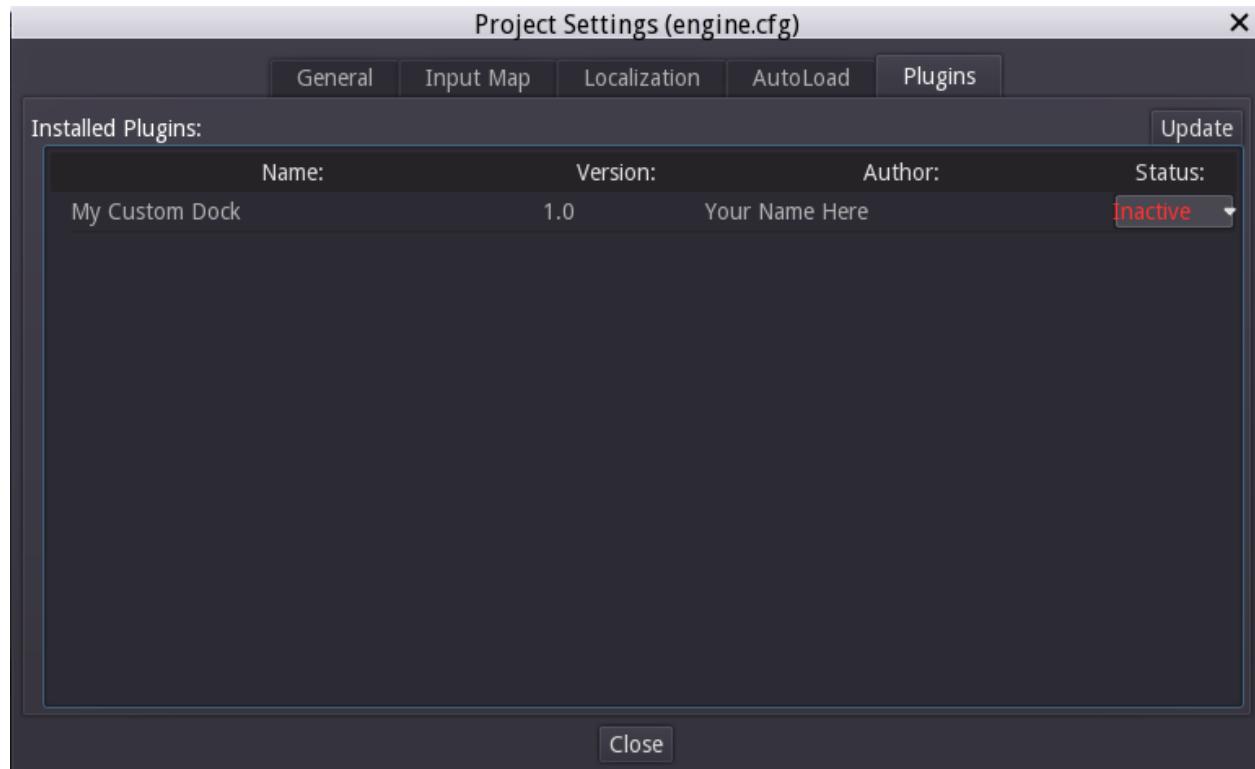
func _exit_tree():
    # Clean-up of the plugin goes here
    # Remove the scene from the docks:
    remove_control_from_docks( dock ) # Remove the dock
    dock.free() # Erase the control from the memory

```

While the dock position is chosen when adding it, the user is free to move it and save the layout with the dock in any position.

Checking the results

Now it is the moment to check the results of your work. Open the *Project Settings* and click on the *Plugins* tab. Your plugin should be the only on the list. If it is not showing, click on the *Update* button at the top right corner.



At the *Status* column, you can see that the plugin is inactive. So you just need to click on the status to select *Active*. The dock should be immediately visible, even before you close the settings window. And now, lo and behold, you have a custom dock! In just a bit of coding and a simple scene.



Going beyond

Now that you learned how to make basic plugins, you can extend the editor in many nice ways. Many functions can be added to editor on the fly with GDScript, it is a powerful way to create special editors without having to delve into C++ modules.

You can make your own plugins to help you and also share them in Godot's Asset Library so many people can benefit of your work.

19.1.2 Import plugins

Introduction

An import plugin is a special type of editor tool that allows custom resources to be imported by Godot and be treated as first-class resources. The editor itself comes bundled with a lot of import plugins to handle the common resources like PNG images, Collada and glTF models, OGG Vorbis sounds, and many more.

This tutorial will show you how to create a simple import plugin to load a custom text file as a material resource. This text file will contain three numeric values separated by comma, which represents the three channels of a color, and the resulting color will be used as the albedo (main color) of the imported material.

Note: This tutorial assumes you already know how to make generic plugins. If in doubt, refer to the *Making Plugins* page. This also assumes you are acquainted with Godot's import system.

The sample file to import contains only a line representing the pure blue color (zero red, zero green, and full blue):

```
0,0,255
```

Configuration

First we need a generic plugin that will handle the initialization and destruction of our import plugin. Let's add the `plugin.cfg` file first:

```
[plugin]
name="Silly Material Importer"
description="Imports a 3D Material from an external text file."
author="Yours Truly"
version="1.0"
script="material_import.gd"
```

Then we need the `material_import.gd` file to add and remove the import plugin when needed:

```
# material_import.gd
tool
extends EditorPlugin

var import_plugin

func _enter_tree():
    import_plugin = preload("import_plugin.gd").new()
    add_import_plugin(import_plugin)

func _exit_tree():
    remove_import_plugin(import_plugin)
    import_plugin = null
```

When this plugin is activated, it will create a new instance of the import plugin (which we'll soon make) and add it to the editor using the `add_import_plugin` method. We store a reference to it in a class member `import_plugin` so we can refer to it later when removing it. The `remove_import_plugin` method is called when the plugin is deactivated to clean up the memory and let the editor know the import plugin isn't available anymore.

Note that the import plugin is a reference type so it doesn't need to be explicitly released from the memory with the `free()` function. It will be released automatically by the engine when it goes out of scope.

The EditorImportPlugin class

The main character of the show is the *EditorImportPlugin* class. It is responsible to implement the methods that are called by Godot when it needs to know how to deal with files.

Let's begin to code our plugin, one method at time:

```
# import_plugin.gd
tool
extends EditorImportPlugin

func get_importer_name():
    return "demos.sillymaterial"
```

The first method is the *get_importer_name*. This is a unique name to your plugin that is used by Godot to know which import was used in a certain file. When the files needs to be reimported, the editor will know which plugin to call.

```
func get_visible_name():
    return "Silly Material"
```

The *get_visible_name* method is responsible to inform the name of the type it imports and will be shown to the user in the Import dock.

You should choose this name as a continuation to “Import as”. Eg. “*Import as Silly Material*”. Yes, this one is a bit silly, but you certainly can come up with a descriptive name for your plugin.

```
func get_recognized_extensions():
    return ["mtxt"]
```

Godot's import system detects file types by their extension. In the *get_recognized_extensions* method you return an array of strings to represent each extension that this plugin can understand. If an extension is recognized by more than one plugin, the user can select which one to use when importing the files.

Tip: Common extensions like `.json` and `.txt` might be used by many plugins. Also, there could be files in the project that are just data for the game and should not be imported. You have to be careful when importing to validate the data. Never expect the file to be well-formed.

```
func get_save_extension():
    return "material"
```

The imported files are saved in the `.import` folder at the project's root. Their extension should match the type of resource you are importing, but since Godot can't tell what you'll use (because there might be multiple valid extensions for the same resource), you need to inform what will be the used in the import.

Since we're importing a Material, we'll use the special extension for such resource types. If you are importing a scene, you can use `scn`. Generic resources can use the `res` extension. However, this is not enforced in any way by the engine.

```
func get_resource_type():
    return "SpatialMaterial"
```

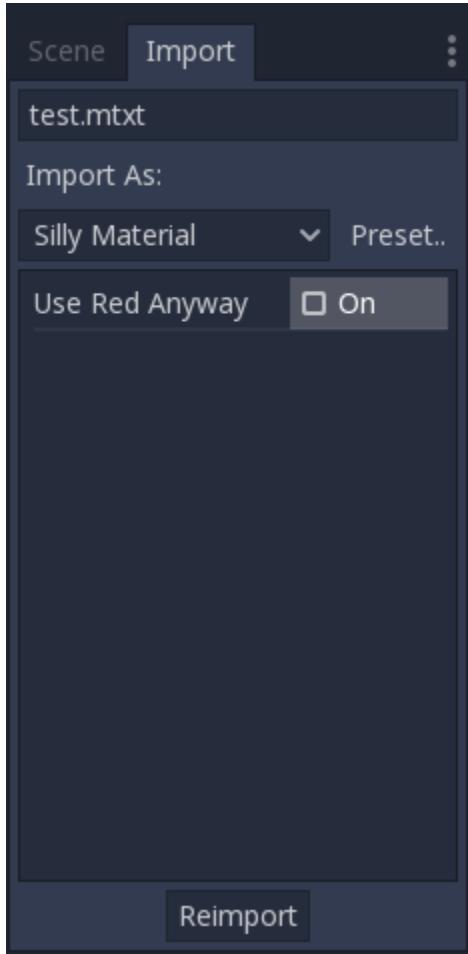
The imported resource has a specific type, so the editor can know which property slot it belongs to. This allows drag and drop from the FileSystem dock to a property in the Inspector.

In our case it's a *SpatialMaterial*, which can be applied to 3D objects.

Note: If you need to import different types from the same extension, you have to create multiple import plugins. You can abstract the import code on another file to avoid duplication in this regard.

Options and presets

Your plugin can provide different options to allow the user to control how the resource will be imported. If a set of selected options is common, you can also create different presets to make it easier for the user. The following image shows how the options will appear in the editor:



Since there might be many presets and they are identified with a number, it's a good practice to use an enum so you can refer to them using names.

```
tool
extends EditorImportPlugin

enum Presets { PRESET_DEFAULT }

...
```

Now that the enum is defined, let's keep looking at the methods of an import plugin:

```
func get_preset_count():
    return Presets.size()
```

The `get_preset_count` method returns the amount of presets that this plugin defines. We only have one preset now, but we can make this method future-proof by returning the size of our `Presets` enumeration.

```
func get_preset_name(preset):
    match preset:
        PRESET_DEFAULT: return "Default"
        _: return "Unknown"
```

Here we have the `get_preset_name` method, which gives names to the presets as they will be presented to the user, so be sure to use short and clear names.

We can use the `match` statement here to make the code more structured. This way it's easy to add new presets in the future. We use the catch all pattern to return something too. Although Godot won't ask for presets beyond the preset count you defined, it's always better to be on the safe side.

If you have only one preset you could simply return its name directly, but if you do this you have to be careful when you add more presets.

```
func get_import_options(preset):
    match preset:
        PRESET_DEFAULT:
            return [
                {
                    "name": "use_red_anyway",
                    "default_value": false
                }
            ]
        _: return []
```

This is the method which defines the available options. `get_import_options` returns an array of dictionaries, and each dictionary contains a few keys that are checked to customize the option as it's shown to the user. The following table shows the possible keys:

Key	Type	Description
name	String	The name of the option. When showed, underscores become spaces and first letters are capitalized.
default_value	Any	The default value of the option for this preset.
property_hint	Enum value	One of the <code>PropertyHint</code> values to use as hint.
hint_string	String	The hint text of the property. The same as you'd add in the <code>export</code> statement in GDScript.
usage	Enum value	One of the <code>PropertyUsageFlags</code> values to define the usage.

The `name` and `default_value` keys are **mandatory**, the rest are optional.

Note that the `get_import_options` method receives the preset number, so you can configure the options for each different preset (especially the default value). In this example we use the `match` statement, but if you have lots of options and the presets only change the value you may want to create the array of options first and then just change it based on the preset.

Warning: The `get_import_options` method is called even if you don't define presets (by making `get_preset_count` return zero). You have to return an array even it's empty, otherwise you can get errors.

```
func get_option_visibility(option, options):
    return true
```

For the `get_option_visibility` method, we simply return `true` because all of our options (i.e. the single one we defined) are visible all the time.

If you need to make certain option visible only if another is set with a certain value, you can add the logic in this method.

The `import` method

The heavy part of the process, responsible for the converting the files into resources, is covered by the `import` method. Our sample code is a bit long, so let's split in a few parts:

```
func import(source_file, save_path, options, r_platform_variants, r_gen_files):
    var file = File.new()
    var err = file.open(source_file, File.READ)
    if (err != OK):
        return err

    var line = file.get_line()

    file.close()
```

The first part of our import method opens and reads the source file. We use the `File` class to do that, passing the `source_file` parameter which is provided by the editor.

If there's an error when opening the file, we return it to let the editor know that the import wasn't successful.

```
var channels = line.split(",")
if channels.size() != 3:
    return ERR_PARSE_ERROR

var color
if options.use_red_anyway:
    color = Color8(255, 0, 0)
else:
    color = Color8(int(channels[0]), int(channels[1]), int(channels[2]))
```

This code takes the line of the file it read before and splits it in pieces that are separated by a comma. If there are more or less than the three values, it considers the file invalid and reports an error.

Then it creates a new `Color` variable and sets its values according to the input file. If the `use_red_anyway` option is enabled, then it sets the color as a pure red instead.

```
var material = SpatialMaterial.new()
material.albedo_color = color
```

This part makes a new `SpatialMaterial` that is the imported resource. We create a new instance of it and then set its albedo color as the value we got before.

```
return ResourceSaver.save("%s.%s" % [save_path, get_save_extension()], material)
```

This is the last part and quite an important one, because here we save the made resource to the disk. The path of the saved file is generated and informed by the editor via the `save_path` parameter. Note that this comes **without** the extension, so we add it using *string formatting*. For this we call the `get_save_extension` method that we defined earlier, so we can be sure that they won't get out of sync.

We also return the result from the `ResourceSaver:save` method, so if there's an error in this step, the editor will know about it.

Platform variants and generated files

You may have noticed that our plugin ignored two arguments of the `import` method. Those are *return arguments* (hence the `r` at the beginning of their name), which means that the editor will read from them after calling your import method. Both of them are arrays that you can fill with information.

The `r_platform_variants` argument is used if you need to import the resource differently depending on the target platform. While it's called *platform* variants, it is based on the presence of *feature tags*, so even the same platform can have multiple variants depending on the setup.

To import a platform variant, you need to save it with the feature tag before the extension, and then push the tag to the `r_platform_variants` array so the editor can know that you did.

For an example, let's say we save a different material for mobile platform. We would need to do something like the following:

```
r_platform_variants.push_back("mobile")
return ResourceSaver.save("%s.%s.%s" % [save_path, "mobile", get_save_extension()],_
                           ↴mobile_material)
```

The `r_gen_files` argument is meant for extra files that are generated during your import process and need to be kept. The editor will look at it to understand the dependencies and make sure the extra file is not inadvertently deleted.

This is also an array and should be filled with full paths of the files you save. As an example, let's create another material for the next pass and save it in a different file:

```
var next_pass = SpatialMaterial.new()
next_pass.albedo_color = color.inverted()
var next_pass_path = "%s.next_pass.%s" % [save_path, get_save_extension()]

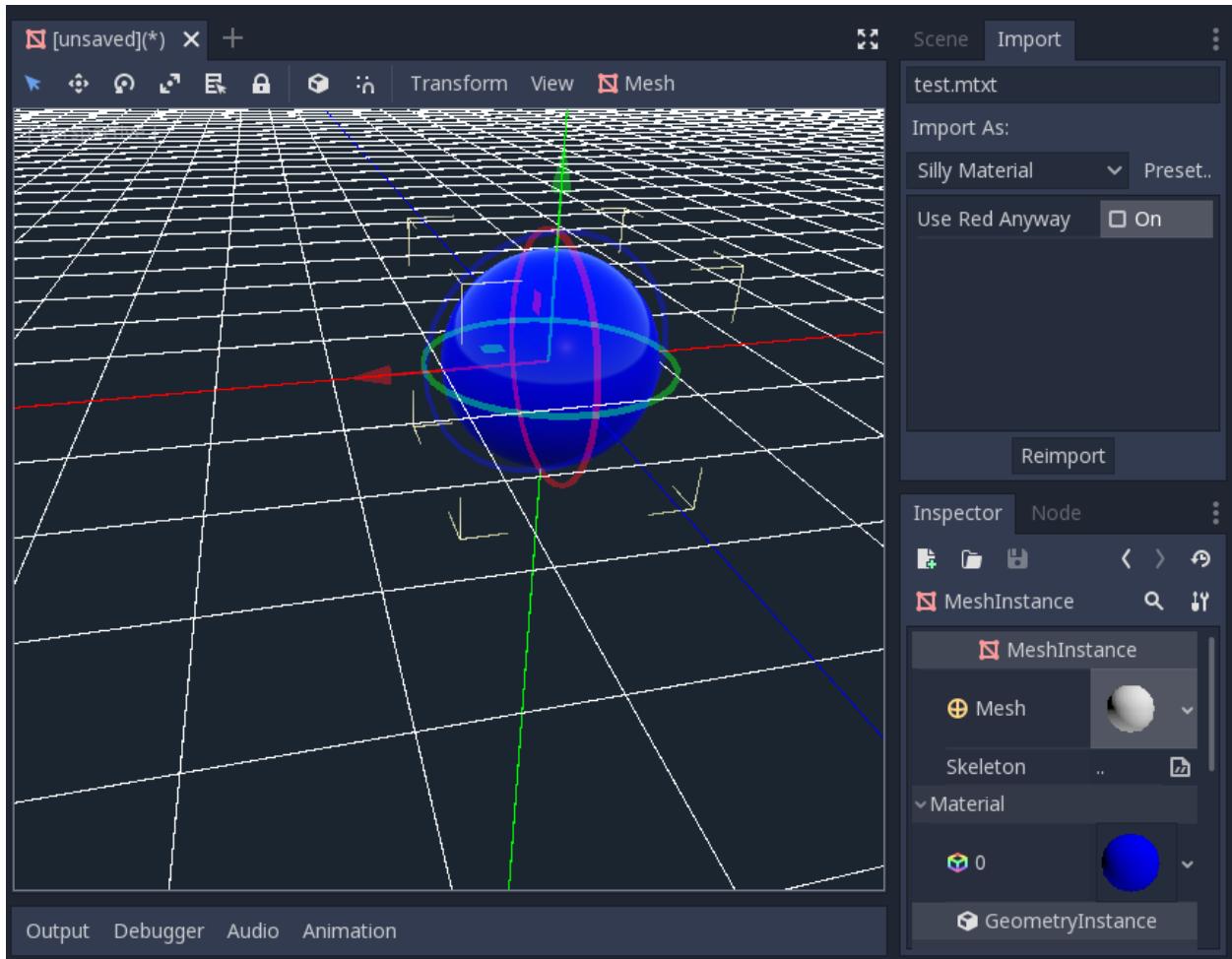
err = ResourceSaver.save(next_pass_path, next_pass)
if err != OK:
    return err
r_gen_files.push_back(next_pass_path)
```

Trying the plugin

This has been very theoretical, but now that the import plugin is done, let's test it. Make sure you created the sample file (with the contents described in the introduction section) and save it as `test.mtxt`. Then activate the plugin in the Project Settings.

If everything goes well, the import plugin is added to the editor and the file system is scanned, making the custom resource appear on the FileSystem dock. If you select it and focus the Import dock, you can see the only option to select there.

Create a `MeshInstance` node in the scene and for its `Mesh` property set up a new `SphereMesh`. Unfold the `Material` section in the Inspector and then drag the file from the FileSystem dock to the `material` property. The object will update in the viewport with the blue color of the imported material.



Go to Import dock, enable the “Use Red Anyway” option, and click on “Reimport”. This will update the imported material and should automatically update the view showing the red color instead.

And that’s it! Your first import plugin is done! Now get creative and make plugins for your own beloved formats. This can be quite useful to write your data in a custom format and then use it in Godot as if they were native resources. This shows how the import system is powerful and extendable.

19.2 GDNative

19.2.1 GDNative C example

Introduction

This tutorial will introduce you to the bare minimum required to create GDNative modules. This should be your starting point into the world of GDNative, understanding the contents of this tutorial will help you in understanding all that is to come after this.

Before we begin, you can download the source code to the example object we’ll be describing here by following this link: <https://github.com/GodotNativeTools/GDNative-demos/tree/master/c/SimpleDemo>

This example project also contains a SConstruct file that makes compiling a little easier but in this tutorial we’ll be doing things by hand.

GDNative can be used to create several types of additions to Godot, from PluginScript to ARVR interfaces. In this tutorial we are going to look at creating a *NativeScript* module. NativeScript allows you to write logic in C or C++ in similar fashion as you would write a GDScript file. We'll be creating the C equivalent of this GDScript:

```
extends Reference

var data

func _ready():
    data = "World from GDScript!"

func get_data():
    return data
```

We'll be writing separate tutorials on the other types of GDNative modules and explain what each of them is for as we go through them.

Prerequisites

Before we start you'll need a few things.

1. A Godot 3.0 executable
2. A C compiler
3. A copy of this repository: https://github.com/GodotNativeTools/godot_headers

The first two pretty much speak for themselves. On Linux, you'll likely have a C compiler, on macOS, it's easiest to just install Xcode from the Mac App Store and, on Windows, we've tested this with both MSVC 2015 and 2017.

For number 3, we suggest that you create a folder somewhere that you use to store your code, open up a terminal and CD into that folder. Then execute:

```
git clone https://github.com/GodotNativeTools/godot_headers
```

This will download the required files into that folder.

If you are building Godot from source, you may need a newer version of these files. You can find this at <godotsource>/modules/gdnative/include

Our C source

Let's start by writing our main code. Ideally, we want to end up with a file structure that looks something like this:

```
+ <your development folder>
  + godot_headers
    - <lots of files here>
  + simple
    + bin
      - libsimple.dll/so/dylib
      - libsimple.gdnlib
      - simple.gdns
    + src
      - .gdignore
      - simple.c
  main.tscn
  project.godot
```

Open up Godot and create a new project called simple. This will create the simple folder and project.godot file. Then manually create a bin and src subfolder in this folder.

We're going to start by having a look at what our simple.c file contains. Now, for our example here we're making a single C source file without a header just to keep things simple. Once you start writing bigger projects it is advisable you break your project up into multiple files. That however falls outside of the scope of this tutorial.

We'll be looking at the source code bit by bit so all the parts below should all be put together into one big file. I'll explain each section as we add it.

The below code includes our header files that we need and then defines two pointers to two different structs. GDNative supports a large collection for functions for calling back into the main Godot executable. In order for your module to have access to these functions, GDNative provides your application with a struct containing pointers to all these functions.

To keep this implementation modular and easily extendable, the core functions are available directly through the “core” API struct, but additional functions have their own “GDNative structs” that are accessible through extensions.

In our example, we access one of these extension to gain access to the functions specifically needed for NativeScript.

```
#include <gdnative_api_struct.gen.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const godot_gdnative_core_api_struct *api = NULL;
const godot_gdnative_ext_nativescript_api_struct *nativescript_api = NULL;
```

A NativeScript behaves like any other script in Godot. Because the NativeScript API is rather low level, it requires the library to specify many things more verbosely than other scripting systems, such as GDScript. When a NativeScript instance gets created, a library-given constructor gets called. When that instance gets destroyed, the given destructor will be executed.

These are forward declarations for the functions we'll be implementing for our object. A constructor and destructor is needed. Additionally, the object will have a single method called get_data.

```
void *simple_constructor(godot_object *p_instance, void *p_method_data);
void simple_destructor(godot_object *p_instance, void *p_method_data, void *p_user_
↪data);
godot_variant simple_get_data(godot_object *p_instance, void *p_method_data
    , void *p_user_data, int p_num_args, godot_variant **p_args);
```

Next up is the first of the entry points Godot will call when our dynamic library is loaded. These methods are all prefixed with godot (you can change this later on) followed by their name. gdnative_init is a function that initialises our dynamic library. Godot will give it a pointer to a structure that contains various bits of information we may find useful amongst which the pointers to our API structures.

For any additional API structures we need to loop through our extensions array and check the type of extension.

```
void GDN_EXPORT godot_gdnative_init(godot_gdnative_init_options *p_options) {
    api = p_options->api_struct;

    // now find our extensions
    for (int i = 0; i < api->num_extensions; i++) {
        switch (api->extensions[i]->type) {
            case GDNATIVE_EXT_NATIVESCRIPT: {
                nativescript_api = (godot_gdnative_ext_nativescript_api_struct *)api->
↪extensions[i];
            }; break;
            default: break;
        }
    }
}
```

```

        }
    }
}
```

Next up is `gdnative_terminate` which is called before the library is unloaded. Godot will unload the library when no object uses it anymore. Here, you can do any cleanup you may need to do. For our example, we're simply going to clear our API pointers.

```
void GDN_EXPORT godot_gdnative_terminate(godot_gdnative_terminate_options *p_options)
{
    api = NULL;
    nativescript_api = NULL;
}
```

Finally we have `nativescript_init` which is the most important function we'll need today. This function will be called by Godot as part of loading a GDNative library and communicates back to Godot what objects we make available to Godot.

We first tell Godot which classes are implemented by calling `nativescript_register_class`. The first parameter here is the handle pointer given to us. The second is the name of our object class. The third is the type of object in Godot that we ‘inherit’ from, this is not true inheritance but it’s close enough. Finally, our fourth and fifth parameters are descriptions for our constructor and destructor.

We then tell Godot about our methods (well our one method in this case), by calling `nativescript_register_method` for each method of our class. In our case, that is just `get_data`. Our first parameter is yet again our handle pointer. The second is again the name of the object class we’re registering. The third is the name of our function as it will be known to GDScript. The fourth is our attributes setting. The fifth and final parameter is a description of which function to call when the method gets called.

The descriptions contain the function pointers to the functions themselves. The other two fields in these structs are for specifying per-method userdata. The value in the `method_data` field will be passed on every function call as the `p_method_data` argument. This is useful to reuse one function for different methods on possibly multiple different script-classes. If the `method_data` value is a pointer to memory that needs to be freed, the `free_func` field can contain a pointer to a function that will free that memory. That free function gets called when the script itself (not instance!) gets unloaded (so usually at library-unload time).

```
void GDN_EXPORT godot_nativescript_init(void *p_handle) {
    godot_instance_create_func create = { NULL, NULL, NULL };
    create.create_func = &simple_constructor;

    godot_instance_destroy_func destroy = { NULL, NULL, NULL };
    destroy.destroy_func = &simple_destructor;

    nativescript_api->godot_nativescript_register_class(p_handle, "SIMPLE", "Reference",
    "create, destroy);

    godot_instance_method get_data = { NULL, NULL, NULL };
    get_data.method = &simple_get_data;

    godot_method_attributes attributes = { GODOT_METHOD_RPC_MODE_DISABLED };

    nativescript_api->godot_nativescript_register_method(p_handle, "SIMPLE", "get_data",
    "attributes, get_data);
}
```

Now, it's time to start working on the functions of our object. First, we define a structure that we use to store the member data of an instance of our GDNative class.

```
typedef struct user_data_struct {
    char data[256];
} user_data_struct;
```

And then, we define our constructor. All we do in our constructor is allocate memory for our structure and fill it with some data. Note that we use Godot's memory functions so the memory gets tracked and then return the pointer to our new structure. This pointer will act as our instance identifier in case multiple objects are instantiated.

This pointer will be passed to any of our functions related to our object as a parameter called `p_user_data`, and can both be used to identify our instance and to access its member data.

```
void *simple_constructor(godot_object *p_instance, void *p_method_data) {
    user_data_struct *user_data = api->godot_alloc(sizeof(user_data_struct));
    strcpy(user_data->data, "World from GDNative!");

    return user_data;
}
```

Our destructor is called when Godot is done with our object and we free our instances' member data.

```
void simple_destructor(godot_object *p_instance, void *p_method_data, void *p_user_
↪data) {
    api->godot_free(p_user_data);
}
```

And finally, we implement our `get_data` function. Data is always sent and returned as variants so in order to return our data, which is a string, we first need to convert our C string to a Godot string object, and then copy that string object into the variant we are returning.

Strings are heap-allocated in Godot, so they have a destructor which frees the memory. Destructors are named `godot_TYPENAME_destroy`. When a Variant gets created with a String, it references the String. That means that the original String can be “destroyed” to decrease the ref-count. If that does not happen the String memory will leak since the ref-count will never be zero and the memory never deallocated. The returned variant gets automatically destroyed by Godot.

(In more complex operations it can be confusing the keep track of which value needs to be deallocated and which does not. As a general rule: call `godot_XXX_destroy` when a C++ destructor would be called instead. The String destructor would be called in C++ after the Variant was created, so the same is necessary in C)

The variant we return is destroyed automatically by Godot.

```
godot_variant simple_get_data(godot_object *p_instance, void *p_method_data,
    void *p_user_data, int p_num_args, godot_variant **p_args) {
    godot_string data;
    godot_variant ret;
    user_data_struct * user_data = (user_data_struct *) p_user_data;

    api->godot_string_new_data(&data, user_data->data, strlen(user_data->data));
    api->godot_variant_new_string(&ret, &data);
    api->godot_string_destroy(&data);

    return ret;
}
```

And that is the whole source code of our module.

If you add a blank `.gdignore` file to the `src` folder, Godot will not try to import the compiler-generated temporary files.

Compiling

We now need to compile our source code. As mentioned our example project on GitHub contains a Scons configuration that does all the hard work for you but for our tutorial here we are going to call the compilers directly.

Assuming you are sticking to the folder structure suggested above it is best to CD into the src subfolder in a terminal session and execute the commands from there. Make sure to create the bin folder before you proceed.

On Linux:

```
clang -std=c11 -fPIC -c -I/PATH/TO/GODOT/HEADERS simple.c -o simple.os
clang -shared simple.os -o ../bin/libsimple.so
```

On Mac OS X:

```
clang -std=c11 -fPIC -c -I/PATH/TO/GODOT/HEADERS simple.c -o simple.os -arch i386 -
clang -dynamiclib simple.os -o ../bin/libsimple.dylib -arch i386 -arch x86_64
```

On Windows:

```
cl /Fosimple.obj /c simple.c /nologo -EHsc -DNDEBUG /MD /I. /IC:/PATH/TO/GODOT/HEADERS
link /nologo /dll /out:..\bin\libsimple.dll /implib:..\bin\libsimple.lib simple.obj
```

Note that on the Windows build you also end up with a libsimple.lib library. This is a library that you can compile into a project to provide access to the DLL. We get it as a bonus and we do not need it :) When exporting your game for release this file will be ignored.

Creating our GDNLIB file

With our module compiled we now need to create a gdnlib file for our module which we place alongside our dynamic libraries. This file tells Godot what dynamic libraries are part of our module and need to be loaded per platform. At the time of writing this tutorial work is still being done on making this configurable from within Godot so for now grab your favourite text editor, create a file called libsimple.gdnlb and add the following into this file:

```
[general]
singleton=false
load_once=true
symbol_prefix="godot_"

[entry]
X11.64="res://bin/libsimple.so"
Windows.64="res://bin/libsimple.dll"
OSX.64="res://bin/libsimple.dylib"

[dependencies]
X11.64=[]
Windows.64=[]
OSX.64=[]
```

This file contains 3 sections.

The **general** section contains some info that tells Godot how to use our module.

If singleton is true our library is automatically loaded and a function called godot_singleton_init is called. We'll leave that for another tutorial.

If load_once is true our library is loaded only once and each individual script that uses our library will use the same data. Any variable you define globally will be accessible from any instance of your object you create. If load_once is false a new copy of the library is loaded into memory each time a script access the library.

The symbol_prefix is a prefix for our core functions. So the godot in godot_nativescript_init for instance. If you use multiple GDNative libraries that you wish to statically link you'll have to use different prefixes. This again is a subject to dive into deeper in a separate tutorial, it is only needed at this time for deployment to iOS as this platform does not like dynamic libraries.

The **entry** section tells us for each platform and feature combination which dynamic library has to be loaded. This also informs the exporter which files need to be exported when exporting to a specific platform.

The **dependencies** section tells Godot what other files need to be exported for each platform in order for our library to work. Say that your GDNative module uses another DLL to implement functionality from a 3rd party library, this is where you list that DLL.

Putting it all together

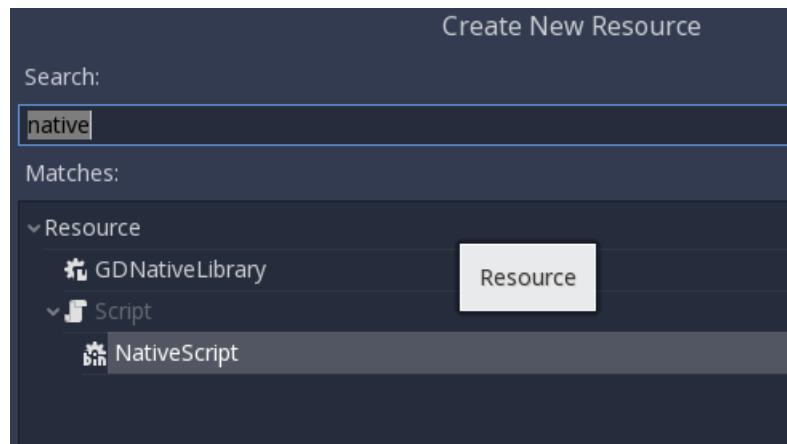
Now that we should have a working GDNative library it is time to fire up Godot and use it. Open up the sample project if you haven't left it open after creating the project all the way at the beginning of this tutorial.

Creating our GDNS file

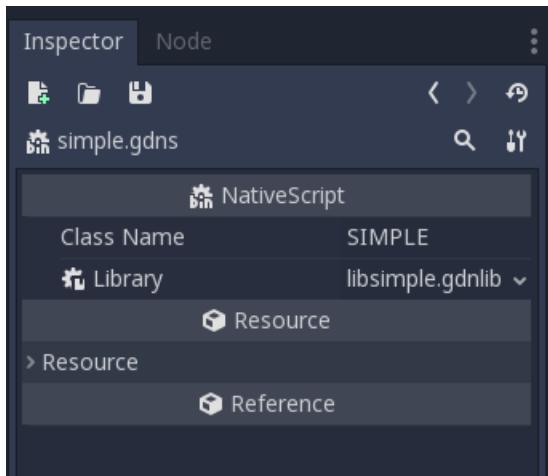
With our GDNLIB file we've told Godot how to load our library, now we need to tell it about our "Simple" object class. This we do by creating a GDNS resource file.

Start by clicking the create resource button in the Inspector:

And select NativeScript:

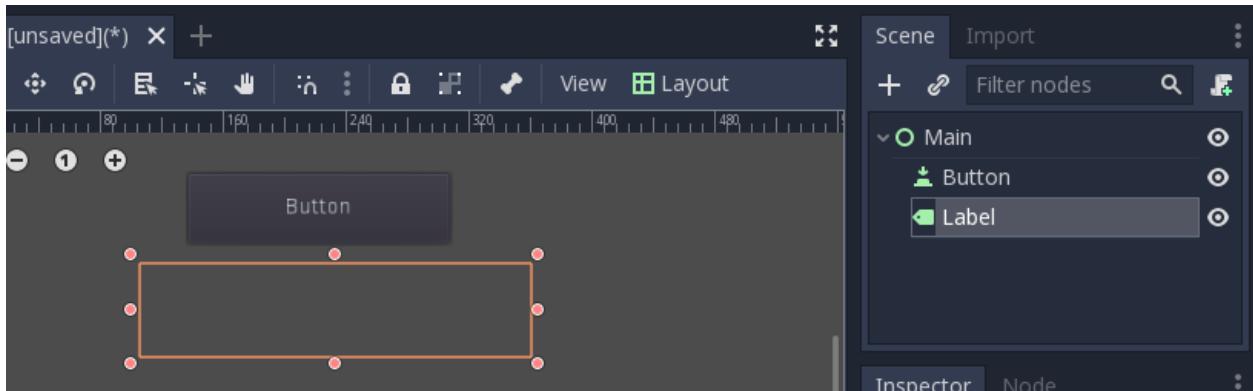


Press Create, now the inspector will show a few fields we need to enter. In Class Name we enter "SIMPLE" which is the object class name we used in our C source when calling godot_nativescript_register_class. We also need to select our GDNLIB file by clicking on Library and selecting Load:



Finally click on the save icon and save this as bin/Simple.gdns:

Now it's time to build our scene. Add a control node to your scene as your root and call it main. Then add a button and a label as subnodes. Place them somewhere nice on screen and give your button a name.



Select the control node and create a script for the control node:

Next link up the pressed signal on the button to your script:

Don't forget to save your scene, call it main.tscn.

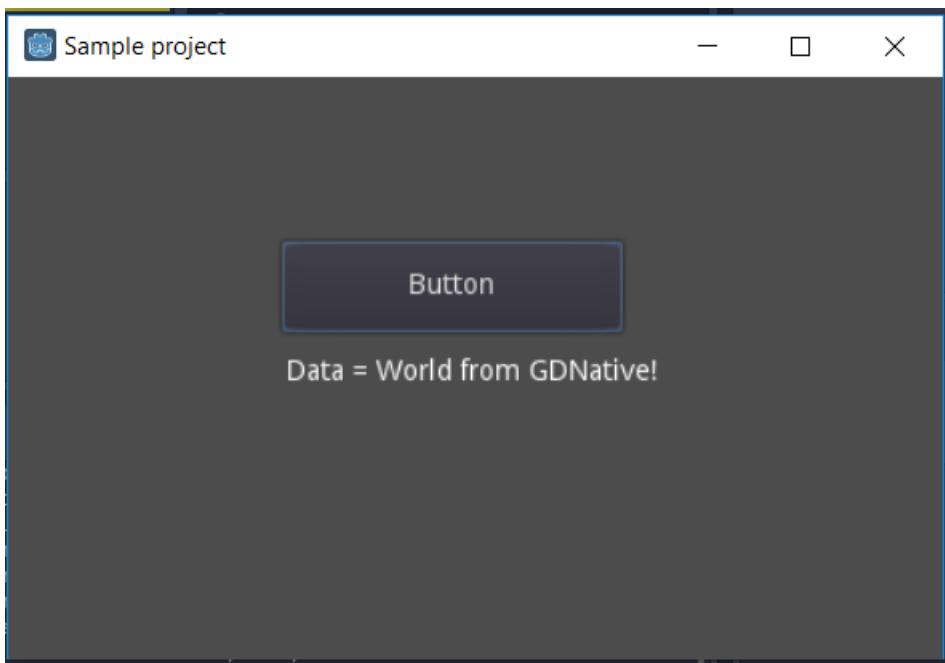
Now we can implement our main.gd code:

```
extends Control

# load the SIMPLE library
onready var data = preload("res://bin/simple.gdns").new()

func _on_Button_pressed():
    $Label.text = "Data = " + data.get_data()
```

After all that, our project should work. The first time you run it Godot will ask you what your main scene is and you select your main.tscn file and presto:



CHAPTER 20

Platform-specific

20.1 Android in-app purchases

Godot engine has integrated GooglePaymentsV3 module with which we can implement in-app purchases in our game. The Godot engine demo project repository has an android-iap example project. It includes a gdscript interface for android IAP.

Check the repository here <https://github.com/godotengine/godot-demo-projects>

Find the iap.gd script in

```
godot-demo-projects/misc/android_iap
```

Add it to the Autoload list and name it as IAP so that we can reference it anywhere in the game.

20.1.1 Getting the product details

When starting our game, we will need to get the item details from Google such as the product price, description and localized price string etc.

```
#First listen to the sku details update callback
IAP.connect("sku_details_complete",self,"sku_details_complete")

#Then ask google the details for these items
IAP.sku_details_query(["pid1","pid2"]) #pid1 and pid2 are our product ids entered in ↵Googleplay dashboard

#This will be called when sku details are retrieved successfully
func sku_details_complete():
    print(IAP.sku_details) #This will print the details as JSON format, refer the ↵format in iap.gd
    print(IAP.sku_details["pid1"].price) #print formatted localized price
```

We can use the IAP details to display the title, price and/or description on our shop scene.

20.1.2 Check if user purchased an item

When starting our game, we can check if the user has purchased any product. YOU SHOULD DO THIS ONLY AFTER 2/3 SECONDS AFTER YOUR GAME IS LOADED. If we do this as the first thing when the game is launched, IAP might not be initialized and our game will crash on start.

```
#Add a listener first
IAP.connect("has_purchased",self,"iap_has_purchased")
IAP.request_purchased() #Ask Google for all purchased items

#This will call for each and every user purchased products
func iap_has_purchased(item_name):
    print(item_name) #print the name of purchased items
```

Google IAP policy says the game should restore the user's purchases if the user replaces their phone or reinstalls the same app. We can use the above code to check what products the user has purchased and we can make our game respond accordingly.

20.1.3 Simple Purchase

We can put this purchase logic on a product's buy button.

```
#First listen for purchase_success callback
IAP.connect("purchase_success",self,"purchase_success_callback")

#Then call purchase like this
IAP.purchase("pid1") #replace pid1 with your product id
IAP.purchase("pid2") #replace pid2 with your another product id

#This function will be called when the purchase is a success
func purchase_success_callback(item):
    print(item + " has purchased")
```

We can also implement other signals for the purchase flow and improve the user experience as you needed.

`purchase_fail` - When the purchase is failed due to any reason

`purchase_cancel` - When the user cancels the purchase

`purchase_owned` - When the user already bought the product earlier

20.1.4 Consumables and Non-Consumables

There are two types of products - consumables and non-consumables. **Consumables** are purchased and used, for eg: healing potions which can be purchased again and again. **Non-consumables** are one time purchases, for eg: Level packs.

Google doesn't have this separation in their dashboard. If our product is a consumable, and if a user has purchased it, it will not be available for purchase until it is consumed. So we should call the consume method for our consumables and don't call consume for your non-consumables.

```
IAP.connect("consume_success", self, "on_consume_success")
IAP.consume("pid")

func on_consume_success(item):
    print(item + " consumed")
```

If our game has only consumables, we don't have to do this. We can set it to consume the item automatically after a purchase.

```
IAP.set_auto_consume(true)
```

If our game has only non-consumables, we can

```
IAP.set_auto_consume(false)
```

We should set the auto consume value only once when the game starts.

20.1.5 Testing

If we add a gmail id as a tester in Google dashboard, that tester can purchase items and they will not be charged. Another way to test IAP is using redeem codes generated by us for our game because the purchase flow is the same.

Third way of testing is in development side. If we put the product ids as shown below, we will get a static fixed response according to the product id. This is a quick way of testing things before going to the dashboard.

- android.test.purchased
- android.test.canceled
- android.test.refunded
- android.test.item_unavailable

20.2 Services for iOS

At the moment, there are 2 iOS APIs partially implemented, GameCenter and Storekit. Both use the same model of asynchronous calls explained below.

20.2.1 Asynchronous methods

When requesting an asynchronous operation, the method will look like this:

```
Error purchase(Variant p_params);
```

The parameter will usually be a Dictionary, with the information necessary to make the request, and the call will have 2 phases. First, the method will immediately return an Error value. If the Error is not 'OK', the call operation is completed, with an error probably caused locally (no internet connection, API incorrectly configured, etc). If the error value is 'OK', a response event will be produced and added to the 'pending events' queue. Example:

```
func on_purchase_pressed():
    var result = InAppStore.purchase( { "product_id": "my_product" } )
    if result == OK:
        animation.play("busy") # show the "waiting for response" animation
    else:
```

```
show_error()

# put this on a 1 second timer or something
func check_events():
    while InAppStore.get_pending_event_count() > 0:
        var event = InAppStore.pop_pending_event()
        if event.type == "purchase":
            if event.result == "ok":
                show_success(event.product_id)
            else:
                show_error()
```

Remember that when a call returns OK, the API will *always* produce an event through the pending_event interface, even if it's an error, or a network timeout, etc. You should be able to, for example, safely block the interface waiting for a reply from the server. If any of the APIs don't behave this way it should be treated as a bug.

The pending event interface consists of 2 methods:

- `get_pending_event_count ()` Returns the number of pending events on the queue.
- `Variant pop_pending_event ()` Pops the first event from the queue and returns it.

20.2.2 Store Kit

Implemented in platform/iphone/in_app_store.mm

The Store Kit API is accessible through the “InAppStore” singleton (will always be available from gdscript). It is initialized automatically. It has 2 methods for purchasing:

- `Error purchase(Variant p_params);`
- `Error request_product_info(Variant p_params);`

and the pending_event interface

```
int get_pending_event_count();
Variant pop_pending_event();
```

purchase

Purchases a product id through the Store Kit API.

Parameters

Takes a Dictionary as a parameter, with one field, `product_id`, a string with your product id. Example:

```
var result = InAppStore.purchase( { "product_id": "my_product" } )
```

Response event

The response event will be a dictionary with the following fields:

On error:

```
{
  "type": "purchase",
  "result": "error",
  "product_id": "the product id requested"
}
```

On success:

```
{
  "type": "purchase",
  "result": "ok",
  "product_id": "the product id requested"
}
```

request_product_info

Requests the product info on a list of product IDs.

Parameters

Takes a Dictionary as a parameter, with one field, `product_ids`, a string array with a list of product ids. Example:

```
var result = InAppStore.request_product_info( { "product_ids": ["my_product1", "my_product2"] } )
```

Response event

The response event will be a dictionary with the following fields:

```
{
  "type": "product_info",
  "result": "ok",
  "invalid_ids": [ list of requested ids that were invalid ],
  "ids": [ list of ids that were valid ],
  "titles": [ list of valid product titles (corresponds with list of valid ids) ],
  "descriptions": [ list of valid product descriptions ],
  "prices": [ list of valid product prices ],
  "localized_prices": [ list of valid product localized prices ],
}
```

20.2.3 Game Center

Implemented in platform/iphone/game_center.mm

The Game Center API is available through the “GameCenter” singleton. It has 6 methods:

- Error post_score(Variant p_score);
- Error award_achievement(Variant p_params);
- Error reset_achievements();
- Error request_achievements();

- Error request_achievement_descriptions();
 - Error show_game_center(Variant p_params);
- plus the standard pending event interface.

post_score

Posts a score to a Game Center leaderboard.

Parameters

Takes a Dictionary as a parameter, with 2 fields:

- score a float number
- category a string with the category name

Example:

```
var result = GameCenter.post_score( { "score": 100, "category": "my_leaderboard", } )
```

Response event

The response event will be a dictionary with the following fields:

On error:

```
{  
    "type": "post_score",  
    "result": "error",  
    "error_code": the value from NSError::code,  
    "error_description": the value from NSError::localizedDescription,  
}
```

On success:

```
{  
    "type": "post_score",  
    "result": "ok",  
}
```

award_achievement

Modifies the progress of a Game Center achievement.

Parameters

Takes a Dictionary as a parameter, with 3 fields:

- name (string) the achievement name
- progress (float) the achievement progress from 0.0 to 100.0 (passed to GKAchievement::percentComplete)

- `show_completion_banner` (bool) whether Game Center should display an achievement banner at the top of the screen

Example:

```
var result = award_achievement( { "name": "hard_mode_completed", "progress": 6.1 } )
```

Response event

The response event will be a dictionary with the following fields:

On error:

```
{  
    "type": "award_achievement",  
    "result": "error",  
    "error_code": the error code taken from NSError::code,  
}
```

On success:

```
{  
    "type": "award_achievement",  
    "result": "ok",  
}
```

reset_achievements

Clears all Game Center achievements. The function takes no parameters.

Response event

The response event will be a dictionary with the following fields:

On error:

```
{  
    "type": "reset_achievements",  
    "result": "error",  
    "error_code": the value from NSError::code  
}
```

On success:

```
{  
    "type": "reset_achievements",  
    "result": "ok",  
}
```

request_achievements

Request all the Game Center achievements the player has made progress on. The function takes no parameters.

Response event

The response event will be a dictionary with the following fields:

On error:

```
{  
    "type": "achievements",  
    "result": "error",  
    "error_code": the value from NSError::code  
}
```

On success:

```
{  
    "type": "achievements",  
    "result": "ok",  
    "names": [ list of the name of each achievement ],  
    "progress": [ list of the progress made on each achievement ]  
}
```

request_achievement_descriptions

Request the descriptions of all existing Game Center achievements regardless of progress. The function takes no parameters.

Response event

The response event will be a dictionary with the following fields:

On error:

```
{  
    "type": "achievement_descriptions",  
    "result": "error",  
    "error_code": the value from NSError::code  
}
```

On success:

```
{  
    "type": "achievement_descriptions",  
    "result": "ok",  
    "names": [ list of the name of each achievement ],  
    "titles": [ list of the title of each achievement ]  
    "unachieved_descriptions": [ list of the description of each achievement when it is  
    ↵unachieved ]  
    "achieved_descriptions": [ list of the description of each achievement when it is  
    ↵achieved ]  
    "maximum_points": [ list of the points earned by completing each achievement ]  
    "hidden": [ list of booleans indicating whether each achievement is initially  
    ↵visible ]  
    "replayable": [ list of booleans indicating whether each achievement can be earned  
    ↵more than once ]  
}
```

show_game_center

Displays the built in Game Center overlay showing leaderboards, achievements, and challenges.

Parameters

Takes a Dictionary as a parameter, with 2 fields:

- `view` (string) (optional) the name of the view to present. Accepts “default”, “leaderboards”, “achievements”, or “challenges”. Defaults to “default”.
- `leaderboard_name` (string) (optional) the name of the leaderboard to present. Only used when “view” is “leaderboards” (or “default” is configured to show leaderboards). If not specified, Game Center will display the aggregate leaderboard.

Examples:

```
var result = show_game_center( { "view": "leaderboards", "leaderboard_name": "best_time_leaderboard" } )
var result = show_game_center( { "view": "achievements" } )
```

Response event

The response event will be a dictionary with the following fields:

On close:

```
{  
    "type": "show_game_center",  
    "result": "ok",  
}
```

20.2.4 Multi-platform games

When working on a multi-platform game, you won’t always have the “GameCenter” singleton available (for example when running on PC or Android). Because the gdscript compiler looks up the singletons at compile time, you can’t just query the singletons to see and use what you need inside a conditional block, you need to also define them as valid identifiers (local variable or class member). This is an example of how to work around this in a class:

```
var GameCenter = null # define it as a class member

func post_score(p_score):
    if GameCenter == null:
        return
    GameCenter.post_score( { "value": p_score, "category": "my_leaderboard" } )

func check_events():
    while GameCenter.get_pending_event_count() > 0:
        # do something with events here
        pass

func _ready():
    # check if the singleton exists
    if Globals.has_singleton("GameCenter"):
```

```
GameCenter = Globals.get_singleton("GameCenter")
# connect your timer here to the "check_events" function
```

20.3 Console Support in Godot

20.3.1 Official Support

Godot does not officially support consoles (save for XBox One via UWP), and this situation will most likely never change.

The reasons for this are:

- To develop for consoles, one must be licensed as a company. Godot, as an open source project, does not have such legal figure.
- Console SDKs are secret, and protected by non-disclosure agreements. Even if we could get access to them, we could not publish the code as open-source.
- Consoles require specialized hardware to develop for, so regular individuals can't create games for them anyway.

This, however, does not mean you can't port your games to console. It's quite the contrary.

20.3.2 Third-Party Support

Console ports of Godot are offered via third party companies (which have ported Godot on their own) and offer porting and publishing services of your games to consoles.

Following is the list of providers:

- [Lone Wolf Technology](#) offers Switch, PS4 and XBox One porting and publishing of Godot games.

Miscellaneous

21.1 Background loading

When switching the main scene of your game (e.g. going to a new level), you might want to show a loading screen with some indication that progress is being made. The main load method (`ResourceLoader::load` or just `load` from GDScript) blocks your thread while the resource is being loaded, so it's not good. This document discusses the `ResourceInteractiveLoader` class for smoother load screens.

21.1.1 ResourceInteractiveLoader

The `ResourceInteractiveLoader` class allows you to load a resource in stages. Every time the method `poll` is called, a new stage is loaded, and control is returned to the caller. Each stage is generally a sub-resource that is loaded by the main resource. For example, if you're loading a scene that loads 10 images, each image will be one stage.

21.1.2 Usage

Usage is generally as follows

Obtaining a ResourceInteractiveLoader

```
Ref<ResourceInteractiveLoader> ResourceLoader::load_interactive(String p_path);
```

This method will give you a `ResourceInteractiveLoader` that you will use to manage the load operation.

Polling

```
Error ResourceInteractiveLoader::poll();
```

Use this method to advance the progress of the load. Each call to `poll` will load the next stage of your resource. Keep in mind that each stage is one entire “atomic” resource, such as an image, or a mesh, so it will take several frames to load.

Returns OK on no errors, ERR_FILE_EOF when loading is finished. Any other return value means there was an error and loading has stopped.

Load progress (optional)

To query the progress of the load, use the following methods:

```
int ResourceInteractiveLoader::get_stage_count() const;
int ResourceInteractiveLoader::get_stage() const;
```

`get_stage_count` returns the total number of stages to load. `get_stage` returns the current stage being loaded.

Forcing completion (optional)

```
Error ResourceInteractiveLoader::wait();
```

Use this method if you need to load the entire resource in the current frame, without any more steps.

Obtaining the resource

```
Ref<Resource> ResourceInteractiveLoader::get_resource();
```

If everything goes well, use this method to retrieve your loaded resource.

21.1.3 Example

This example demonstrates how to load a new scene. Consider it in the context of the *Singletons (AutoLoad)* example.

First we setup some variables and initialize the `current_scene` with the main scene of the game:

```
var loader
var wait_frames
var time_max = 100 # msec
var current_scene

func _ready():
    var root = get_tree().get_root()
    current_scene = root.get_child(root.get_child_count() -1)
```

The function `goto_scene` is called from the game when the scene needs to be switched. It requests an interactive loader, and calls `set_process(true)` to start polling the loader in the `_process` callback. It also starts a “loading” animation, which can show a progress bar or loading screen, etc.

```
func goto_scene(path): # game requests to switch to this scene
    loader = ResourceLoader.load_interactive(path)
    if loader == null: # check for errors
        show_error()
        return
    set_process(true)
```

```

current_scene.queue_free() # get rid of the old scene

# start your "loading..." animation
get_node("animation").play("loading")

wait_frames = 1

```

`_process` is where the loader is polled. `poll` is called, and then we deal with the return value from that call. `OK` means keep polling, `ERR_FILE_EOF` means load is done, anything else means there was an error. Also note we skip one frame (via `wait_frames`, set on the `goto_scene` function) to allow the loading screen to show up.

Note how we use `OS.get_ticks_msec` to control how long we block the thread. Some stages might load really fast, which means we might be able to cram more than one call to `poll` in one frame, some might take way more than your value for `time_max`, so keep in mind we won't have precise control over the timings.

```

func _process(time):
    if loader == null:
        # no need to process anymore
        set_process(false)
        return

    if wait_frames > 0: # wait for frames to let the "loading" animation to show up
        wait_frames -= 1
        return

    var t = OS.get_ticks_msec()
    while OS.get_ticks_msec() < t + time_max: # use "time_max" to control how much
        ↪ time we block this thread

        # poll your loader
        var err = loader.poll()

        if err == ERR_FILE_EOF: # load finished
            var resource = loader.get_resource()
            loader = null
            set_new_scene(resource)
            break
        elif err == OK:
            update_progress()
        else: # error during loading
            show_error()
            loader = null
            break

```

Some extra helper functions. `update_progress` updates a progress bar, or can also update a paused animation (the animation represents the entire load process from beginning to end). `set_new_scene` puts the newly loaded scene on the tree. Because it's a scene being loaded, `instance()` needs to be called on the resource obtained from the loader.

```

func update_progress():
    var progress = float(loader.get_stage()) / loader.get_stage_count()
    # update your progress bar?
    get_node("progress").set_progress(progress)

    # or update a progress animation?
    var len = get_node("animation").get_current_animation_length()

```

```
# call this on a paused animation. use "true" as the second parameter to force
→the animation to update
get_node("animation").seek(progress * len, true)

func set_new_scene(scene_resource):
    current_scene = scene_resource.instance()
    get_node("/root").add_child(current_scene)
```

21.1.4 Using multiple threads

ResourceInteractiveLoader can be used from multiple threads. A couple of things to keep in mind if you attempt it:

Use a Semaphore

While your thread waits for the main thread to request a new resource, use a Semaphore to sleep (instead of a busy loop or anything similar).

Not blocking main thread during the polling

If you have a mutex to allow calls from the main thread to your loader class, don't lock it while you call `poll` on the loader. When a resource is finished loading, it might require some resources from the low level APIs (VisualServer, etc), which might need to lock the main thread to acquire them. This might cause a deadlock if the main thread is waiting for your mutex while your thread is waiting to load a resource.

21.1.5 Example class

You can find an example class for loading resources in threads here: `resource_queue.gd`. Usage is as follows:

```
func start()
```

Call after you instance the class to start the thread.

```
func queue_resource(path, p_in_front = false)
```

Queue a resource. Use optional parameter “`p_in_front`” to put it in front of the queue.

```
func cancel_resource(path)
```

Remove a resource from the queue, discarding any loading done.

```
func is_ready(path)
```

Returns true if a resource is done loading and ready to be retrieved.

```
func get_progress(path)
```

Get the progress of a resource. Returns -1 on error (for example if the resource is not on the queue), or a number between 0.0 and 1.0 with the progress of the load. Use mostly for cosmetic purposes (updating progress bars, etc), use `is_ready` to find out if a resource is actually ready.

```
func get_resource(path)
```

Returns the fully loaded resource, or null on error. If the resource is not done loading (`is_ready` returns false), it will block your thread and finish the load. If the resource is not on the queue, it will call `ResourceLoader::load` to load it normally and return it.

Example:

```
# initialize
queue = preload("res://resource_queue.gd").new()
queue.start()

# suppose your game starts with a 10 second cutscene, during which the user can't
# interact with the game.
# For that time we know they won't use the pause menu, so we can queue it to load
# during the cutscene:
queue.queue_resource("res://pause_menu.tres")
start_curscene()

# later when the user presses the pause button for the first time:
pause_menu = queue.get_resource("res://pause_menu.tres").instance()
pause_menu.show()

# when you need a new scene:
queue.queue_resource("res://level_1.tscn", true) # use "true" as the second parameter
# to put it at the front
# any other resource
# of the queue, pausing the load of

# to check progress
if queue.is_ready("res://level_1.tscn"):
    show_new_level(queue.get_resource("res://level_1.tscn"))
else:
    update_progress(queue.get_process("res://level_1.tscn"))

# when the user walks away from the trigger zone in your Metroidvania game:
queue.cancel_resource("res://zone_2.tscn")
```

Note: this code in its current form is not tested in real world scenarios. Ask punto on IRC (#godotengine on irc.freenode.net) for help.

21.2 Data paths

21.2.1 Path separators

For the sake of supporting as many platforms as possible, Godot only accepts unix style path separators (/). These work everywhere, including Windows.

A path like: C:\Projects will become C:/Projects.

21.2.2 Resource path

As mentioned before. Godot considers that a project exists at any given folder that contains an “`project.godot`” text file, even if such file is empty.

Accessing project files can be done by opening any path with `res://` as a base. For example, a texture located in the root of the project folder may be opened from the following path: `res://sometexture.png`.

21.2.3 Userdata path (persistent data)

While the project is running, it is a very common scenario that the resource path will be read-only, due to it being inside a package, self contained executable, or system wide install location.

Storing persistent files in such scenarios should be done by using the `user://` prefix, for example: `user://gamesave.txt`.

In some devices (for example, mobile ad consoles) this path is unique for the app. Under desktop operating systems, the engine uses the typical `~/.Name` (check the project name under the settings) in macOS and Linux, and `APPDATA/Name` for Windows.

21.3 Saving games

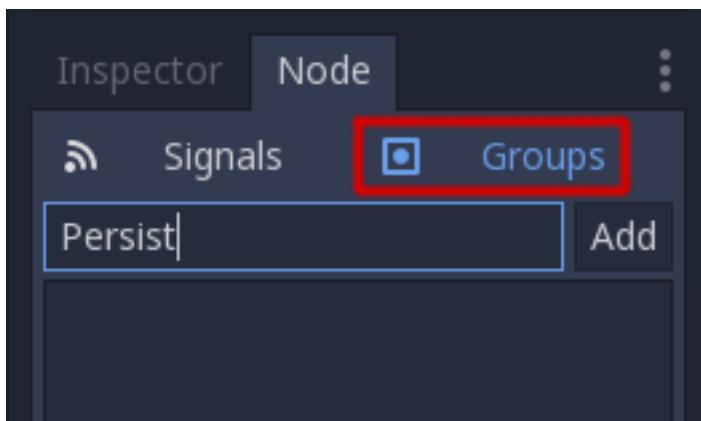
21.3.1 Introduction

Save games can be complicated. It can be desired to store more information than the current level or number of stars earned on a level. More advanced save games may need to store additional information about an arbitrary number of objects. This will allow the save function to scale as the game grows more complex.

21.3.2 Identify persistent objects

First we should identify what objects we want to keep between game sessions and what information we want to keep from those objects. For this tutorial, we will use groups to mark and handle objects to be saved but other methods are certainly possible.

We will start by adding objects we wish to save to the “Persist” group. As in the *Scripting (continued)* tutorial, we can do this through the GUI or through script. Let’s add the relevant nodes using the GUI:



Once this is done when we need to save the game we can get all objects to save them and then tell them all to save with this script:

```
var savenodes = get_tree().get_nodes_in_group("Persist")
for i in savenodes:
    # Now we can call our save function on each node.
```

21.3.3 Serializing

The next step is to serialize the data. This makes it much easier to read and store to disk. In this case, we're assuming each member of group Persist is an instanced node and thus has a path. GDScript has helper functions for this, such as `to_json()` and `parse_json()`, so we will use a dictionary. Our node needs to contain a save function that returns this data. The save function will look like this:

```
func save():
    var savedict = {
        filename=get_filename(),
        parent=get_parent().get_path(),
        posx=get_pos().x, #Vector2 is not supported by json
        posy=get_pos().y,
        attack=attack,
        defense=defense,
        currenthealth=currenthealth,
        maxhealth=maxhealth,
        damage=damage,
        regen=regen,
        experience=experience,
        TNL=TNL,
        level=level,
        AttackGrowth=AttackGrowth,
        DefenseGrowth=DefenseGrowth,
        HealthGrowth=HealthGrowth,
        isalive=isalive,
        last_attack=last_attack
    }
    return savedict
```

This gives us a dictionary with the style `{ "variable_name":that_variables_value }` which will be useful when loading.

21.3.4 Saving and reading data

As covered in the *File system* tutorial, we'll need to open a file and write to it and then later read from it. Now that we have a way to call our groups and get their relevant data, let's use `to_json()` to convert it into an easily stored string and store them in a file. Doing it this way ensures that each line is its own object so we have an easy way to pull the data out of the file as well.

```
# Note: This can be called from anywhere inside the tree. This function is path_
→independent.
# Go through everything in the persist category and ask them to return a dict of_
→relevant variables
func save_game():
    var savegame = File.new()
    savegame.open("user://savegame.save", File.WRITE)
    var savenodes = get_tree().get_nodes_in_group("Persist")
    for i in savenodes:
        var nodedata = i.save()
        savegame.store_line(to_json(nodedata))
    savegame.close()
```

Game saved! Loading is fairly simple as well. For that we'll read each line, use `parse_json()` to read it back to a dict, and then iterate over the dict to read our values. But we'll need to first create the object and we can use the filename and parent values to achieve that. Here is our load function:

```

# Note: This can be called from anywhere inside the tree. This function is path-independent.
func load_game():
    var savegame = File.new()
    if !savegame.file_exists("user://savegame.save"):
        return #Error! We don't have a save to load

    # We need to revert the game state so we're not cloning objects during loading. This will vary wildly depending on the needs of a project, so take care with this step.
    # For our example, we will accomplish this by deleting savable objects.
    var savenodes = get_tree().get_nodes_in_group("Persist")
    for i in savenodes:
        i.queue_free()

    # Load the file line by line and process that dictionary to restore the object it represents
    savegame.open("user://savegame.save", File.READ)
    while (!savegame.eof_reached()):
        var currentLine = parse_json(savegame.get_line())
        # First we need to create the object and add it to the tree and set its position.
        var newobject = load(currentline["filename"]).instance()
        get_node(currentline["parent"]).add_child(newobject)
        newobject.set_position(Vector2(currentline["posx"], currentline["posy"]))
        # Now we set the remaining variables.
        for i in currentline.keys():
            if (i == "filename" or i == "parent" or i == "posx" or i == "posy"):
                continue
            newobject.set(i, currentline[i])
    savegame.close()

```

And now we can save and load an arbitrary number of objects laid out almost anywhere across the scene tree! Each object can store different data depending on what it needs to save.

21.3.5 Some notes

We may have glossed over a step, but setting the game state to one fit to start loading data can be very complicated. This step will need to be heavily customized based on the needs of an individual project.

This implementation assumes no Persist objects are children of other Persist objects. Doing so would create invalid paths. If this is one of the needs of a project this needs to be considered. Saving objects in stages (parent objects first) so they are available when child objects are loaded will make sure they're available for the `add_child()` call. There will also need to be some way to link children to parents as the `NodePath` will likely be invalid.

21.4 Encrypting save games

21.4.1 Why?

Because the world today is not the world of yesterday. A capitalist oligarchy runs the world and forces us to consume in order to keep the gears of this rotten society on track. As such, the biggest market for video game consumption today is the mobile one. It is a market of poor souls forced to compulsively consume digital content in order to forget the misery of their every day life, commute, or just any other brief free moment they have that they are not using to produce goods or services for the ruling class. These individuals need to keep focusing on their video games (because

not doing so will produce them a tremendous existential angst), so they go as far as spending money on them to extend their experience, and their preferred way of doing so is through in-app purchases and virtual currency.

But, imagine if someone was to find a way to edit the saved games and assign the items and currency without effort? This would be terrible, because it would help players consume the content much faster, and as such run out of it sooner than expected. If this happens they will have nothing that avoids them to think, and the tremendous agony of realizing their own irrelevance would again take over their life.

No, we definitely do not want this to happen, so let's see how to encrypt savegames and protect the world order.

21.4.2 How?

The class *File* is simple to use, just open a location and read/write data (integers, strings and variants). To create an encrypted file, a passphrase must be provided, like this:

```
var f = File.new()
var err = f.open_encrypted_with_pass("user://savedata.bin", File.WRITE, "mypass")
f.store_var(game_state)
f.close()
```

This will make the file unreadable to users, but will still not avoid them to share savefiles. To solve this, using the device unique id or some unique user identifier is needed, for example:

```
var f = File.new()
var err = f.open_encrypted_with_pass("user://savedata.bin", File.WRITE, OS.get_unique_
→ID())
f.store_var(game_state)
f.close()
```

Note that `OS.get_unique_ID()` only works on iOS and Android.

This is all! Thanks for your cooperation, citizen.

21.5 Handling quit requests

21.5.1 Quitting

Most platforms have the option to request the application to quit. On desktops, this is usually done with the “x” icon on the window titlebar. On Android, the back button is used to quit when on the main screen (and to go back otherwise).

21.5.2 Handling the notification

The *MainLoop* has a special notification that is sent to all nodes when quit is requested: `MainLoop.NOTIFICATION_WM_QUIT`.

Handling it is done as follows (on any node):

```
func _notification(what):
    if (what == MainLoop.NOTIFICATION_WM_QUIT_REQUEST):
        get_tree().quit() # default behavior
```

When developing mobile apps, quitting is not desired unless the user is on the main screen, so the behavior can be changed.

It is important to note that by default, Godot apps have the built-in behavior to quit when quit is requested, this can be changed:

```
get_tree().set_auto_accept_quit(false)
```

21.6 Pausing games

21.6.1 Pause?

In most games it is desirable to, at some point, interrupt the game to do something else, such as taking a break or changing options. However this is not as simple as it seems. The game might be stopped, but it might be desirable that some menus and animations continue working.

Implementing a fine-grained control for what can be paused (and what can not) is a lot of work, so a simple framework for pausing is provided in Godot.

21.6.2 How pausing works

To set pause mode, the pause state must be set. This is done by assigning “true” to the *SceneTree.paused* member variable:

```
get_tree().paused = true
```

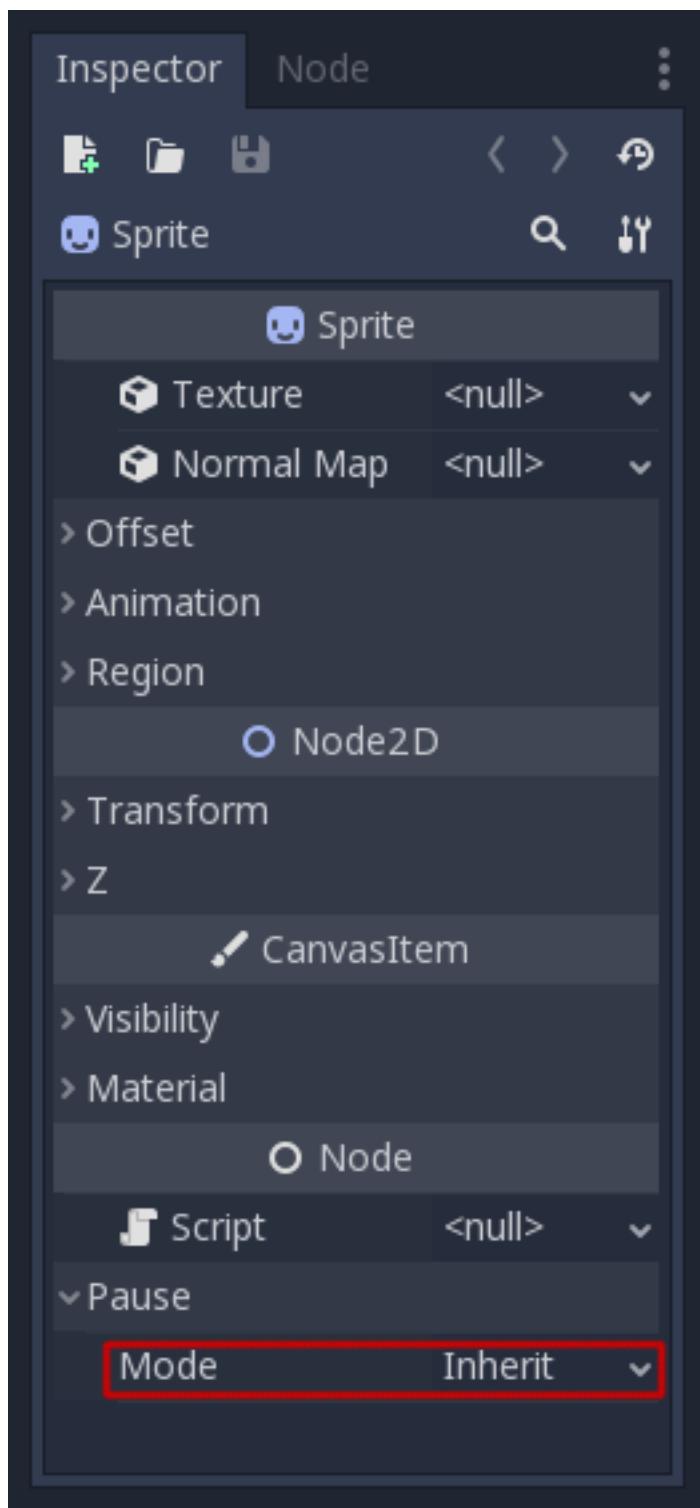
Doing so will have the following behavior:

- 2D and 3D physics will be stopped.
- `_process` and `_physics_process` will not be called anymore in nodes.
- `_input` and `_input_event` will not be called anymore either.

This effectively stops the whole game. Calling this function from a script, by default, will result in an unrecoverable state (nothing will work anymore!).

21.6.3 White-listing nodes

Before enabling pause, make sure that nodes that must keep working during pause are white-listed. This is done by editing the “Pause Mode” property in a node:



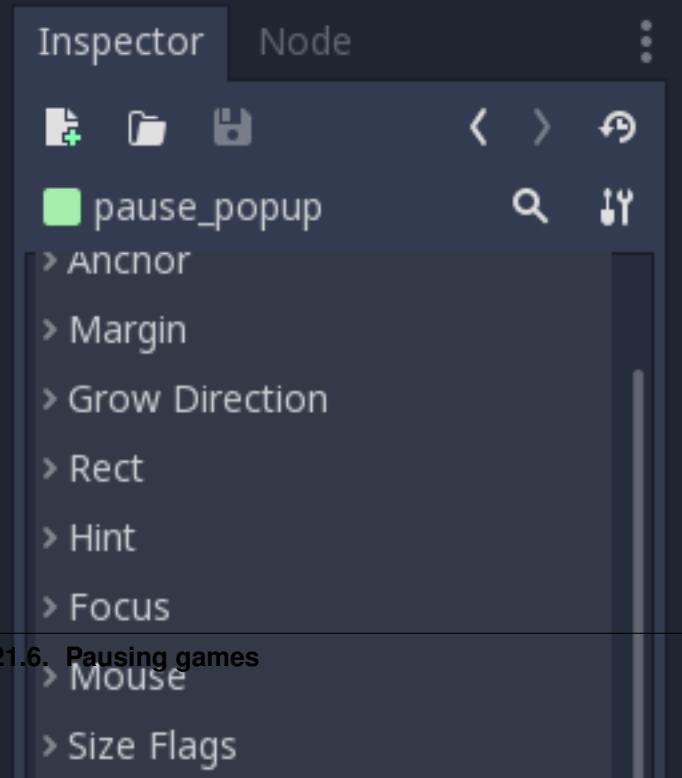
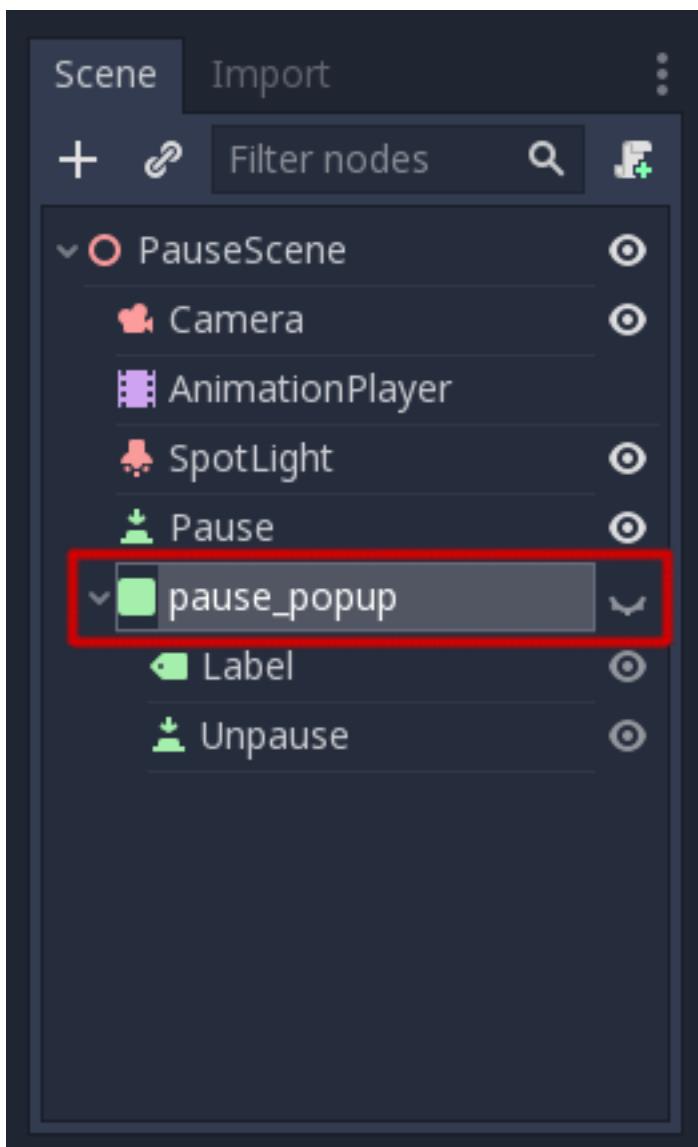
By default all nodes have this property in the “Inherit” state. This means, that they will only process (or not) depending on what this same property is set on the parent node. If the parent is set to “Inherit”, then the grandparent will be checked and so on. Ultimately, if a state can’t be found in any of the grandparents, the pause state in SceneTree is used. This means that, by default, when the game is paused every node will be paused.

So the three possible states for a node are:

- **Inherit:** Process depending on the state of the parent, grandparent, etc. The first parent that has a non-Inherit state.
- **Stop:** Stop the node no matter what (and children in Inherit mode). When paused this node will not process.
- **Process:** Process the node no matter what (and children in Inherit mode). Paused or not this node will process.

21.6.4 Example

An example of this is creating a popup or panel with controls inside, and set its pause mode to “Process” then just hide it:



Just by setting the root of the pause popup to “Process”, all children and grandchildren will inherit that state. This way, this branch of the scene tree will continue working when paused.

Finally, make it so when a pause button is pressed (any button will do), enable the pause and show the pause screen.

```
func _on_pause_button_pressed():
    get_tree().paused = true
    $pause_popup.show()
```

To remove the pause, just do the opposite when the pause screen is closed:

```
func _on_pause_popup_close_pressed():
    $pause_popup.hide()
    get_tree().paused = false
```

And that should be all!

21.7 Internationalizing games

21.7.1 Introduction

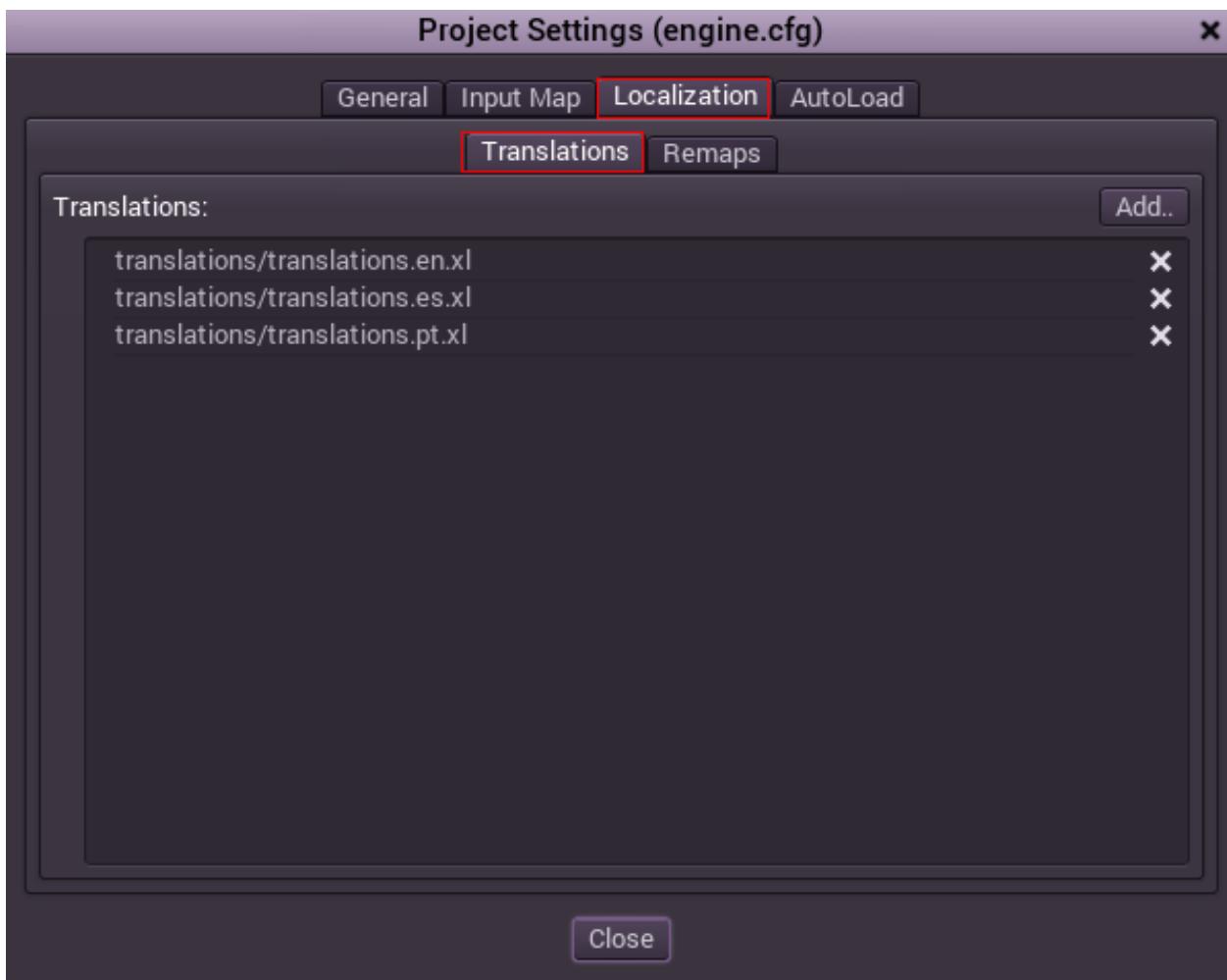
Sería excelente que el mundo hablara solo un idioma. Unfortunately for us developers, that is not the case. While not generally a big requirement when developing indie or niche games, it is also very common that games going into a more massive market require localization.

Godot offers many tools to make this process more straightforward, so this tutorial is more like a collection of tips and tricks.

Localization is usually done by specific studios hired for the job and, despite the huge amount of software and file formats available for this, the most common way to do localization to this day is still with spreadsheets. The process of creating the spreadsheets and importing them is already covered in the *Importing translations* tutorial, so this one could be seen more like a follow up to that one.

21.7.2 Configuring the imported translation

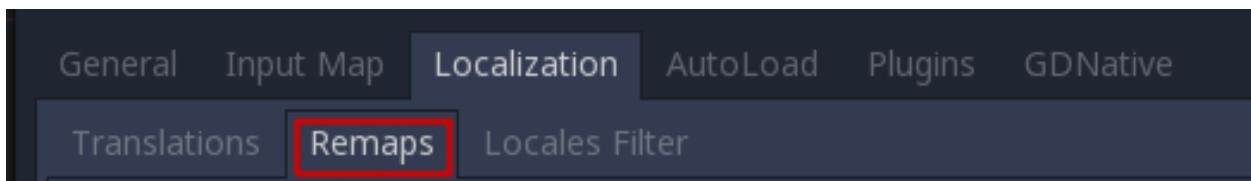
The translations can get updated and re-imported when they change, but they still have to be added to the project. This is done in Scene > Project Settings > Localization:



This dialog allows to add or remove translations project-wide.

21.7.3 Localizing resources

It is also possible to instruct Godot to open alternative versions of assets (resources) depending on the current language. For this the “Remaps” tab exists:



Select the resource to be remapped, and the alternatives for each locale.

21.7.4 Converting keys to text

Some controls such as *Button*, *Label*, etc. will automatically fetch a translation each time they are set a key instead of a text. For example, if a label is assigned “MAIN_SCREEN_GREETING1” and a key to different languages exists in the translations, this will be automatically converted. This process is done upon load though, so if the project in

question has a dialog that allows changing the language in the settings, the scenes (or at least the settings scene) will have to be re-loaded for new text to have effect.

For code, the *Object.tr()* function can be used. This will just look-up the text into the translations and convert it if found:

```
level.set_text(tr("LEVEL_5_NAME"))
status.set_text(tr("GAME_STATUS_" + str(status_index)))
```

21.7.5 Making controls resizable

The same text in different languages can vary greatly in length. For this, make sure to read the tutorial on *Size and anchors*, as having dynamically adjusted control sizes may help. *Container* can be very useful, as well as the multiple options in *Label* for text wrapping.

21.7.6 TranslationServer

Godot has a server for handling the low level translation management called the *TranslationServer*. Translations can be added or removed during run-time, and the current language be changed too.

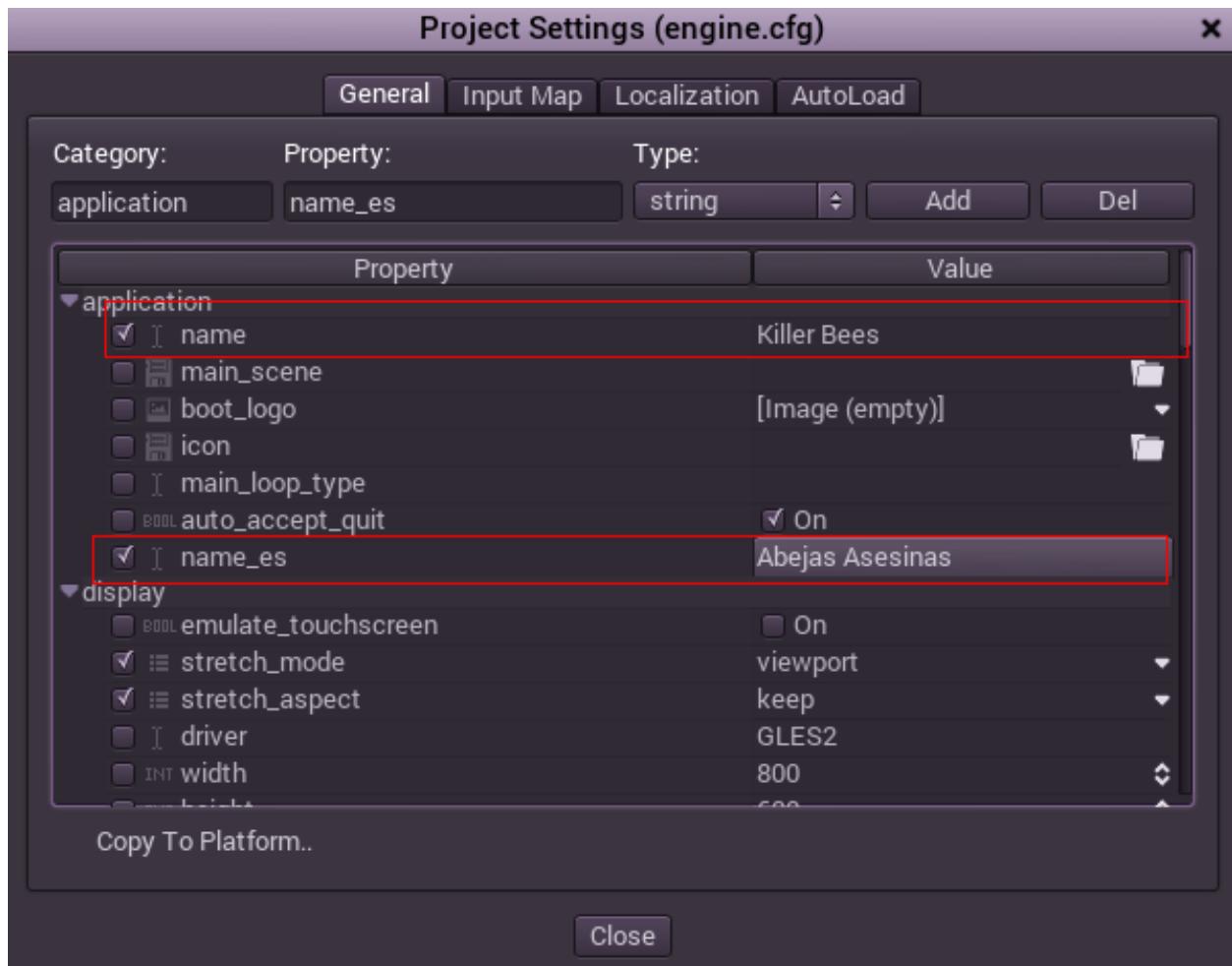
21.7.7 Command line

Language can be tested when running Godot from command line. For example, to test a game in french, the following arguments can be supplied:

```
c:\MyGame> godot -lang fr
```

21.7.8 Translating the project name

The project name becomes the app name when exporting to different operating systems and platforms. To specify the project name in more than one language, create a new setting application/name in the project settings dialog and append the locale identifier to it. For example:



As always, If you don't know the code of a language or zone, *check the list*.

21.8 Locales

This is the list of supported locales and variants in the engine. It's based on the Unix standard locale strings:

Locale	Language and Variant
aa	Afar
aa_DJ	Afar (Djibouti)
aa_ER	Afar (Eritrea)
aa_ET	Afar (Ethiopia)
af	Afrikaans
af_ZA	Afrikaans (South Africa)
agr_PE	Aguaruna (Peru)
ak_GH	Akan (Ghana)
am_ET	Amharic (Ethiopia)
an_ES	Aragonese (Spain)
anp_IN	Angika (India)
ar	Arabic

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
ar_AE	Arabic (United Arab Emirates)
ar_BH	Arabic (Bahrain)
ar_DZ	Arabic (Algeria)
ar_EG	Arabic (Egypt)
ar_IQ	Arabic (Iraq)
ar_JO	Arabic (Jordan)
ar_KW	Arabic (Kuwait)
ar_LB	Arabic (Lebanon)
ar LY	Arabic (Libya)
ar_MA	Arabic (Morocco)
ar_OM	Arabic (Oman)
ar_QA	Arabic (Qatar)
ar_SA	Arabic (Saudi Arabia)
ar_SD	Arabic (Sudan)
ar_SY	Arabic (Syria)
ar_TN	Arabic (Tunisia)
ar_YE	Arabic (Yemen)
as_IN	Assamese (India)
ast_ES	Asturian (Spain)
ayc_PE	Southern Aymara (Peru)
ay_PE	Aymara (Peru)
az_AZ	Azerbaijani (Azerbaijan)
be	Belarusian
be_BY	Belarusian (Belarus)
bem_ZM	Bemba (Zambia)
ber_DZ	Berber languages (Algeria)
ber_MA	Berber languages (Morocco)
bg	Bulgarian
bg_BG	Bulgarian (Bulgaria)
bhb_IN	Bhili (India)
bho_IN	Bhojpuri (India)
bi_TV	Bislama (Tuvalu)
bn	Bengali
bn_BD	Bengali (Bangladesh)
bn_IN	Bengali (India)
bo	Tibetan
bo_CN	Tibetan (China)
bo_IN	Tibetan (India)
br_FR	Breton (France)
brx_IN	Bodo (India)
bs_BA	Bosnian (Bosnia and Herzegovina)
byn_ER	Bilin (Eritrea)
ca	Catalan
ca_AD	Catalan (Andorra)
ca_ES	Catalan (Spain)
ca_FR	Catalan (France)
ca_IT	Catalan (Italy)
ce_RU	Chechen (Russia)
chr_US	Cherokee (United States)

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
cmn_TW	Mandarin Chinese (Taiwan)
crh_UA	Crimean Tatar (Ukraine)
csb_PL	Kashubian (Poland)
cs	Czech
cs_CZ	Czech (Czech Republic)
cv_RU	Chuvash (Russia)
cy_GB	Welsh (United Kingdom)
da	Danish
da_DK	Danish (Denmark)
de	German
de_AT	German (Austria)
de_BE	German (Belgium)
de_CH	German (Switzerland)
de_DE	German (Germany)
de_IT	German (Italy)
de_LU	German (Luxembourg)
doi_IN	Dogri (India)
dv_MV	Dhivehi (Maldives)
dz_BT	Dzongkha (Bhutan)
el	Greek
el_CY	Greek (Cyprus)
el_GR	Greek (Greece)
en	English
en_AG	English (Antigua and Barbuda)
en_AU	English (Australia)
en_BW	English (Botswana)
en_CA	English (Canada)
en_DK	English (Denmark)
en_GB	English (United Kingdom)
en_HK	English (Hong Kong)
en_IE	English (Ireland)
en_IL	English (Israel)
en_IN	English (India)
en_NG	English (Nigeria)
en_MT	English (Malta)
en_NZ	English (New Zealand)
en_PH	English (Philippines)
en_SG	English (Singapore)
en_US	English (United States)
en_ZA	English (South Africa)
en_ZM	English (Zambia)
en_ZW	English (Zimbabwe)
eo	Esperanto
es	Spanish
es_AR	Spanish (Argentina)
es_BO	Spanish (Bolivia)
es_CL	Spanish (Chile)
es_CO	Spanish (Colombia)
es_CR	Spanish (Costa Rica)

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
en_CU	Spanish (Cuba)
es_DO	Spanish (Dominican Republic)
es_EC	Spanish (Ecuador)
es_ES	Spanish (Spain)
es_GT	Spanish (Guatemala)
es_HN	Spanish (Honduras)
es_MX	Spanish (Mexico)
es_NI	Spanish (Nicaragua)
es_PA	Spanish (Panama)
es_PE	Spanish (Peru)
es_PR	Spanish (Puerto Rico)
es_PY	Spanish (Paraguay)
es_SV	Spanish (El Salvador)
es_US	Spanish (United States)
es_UY	Spanish (Uruguay)
es_VE	Spanish (Venezuela)
et	Estonian
et_EE	Estonian (Estonia)
eu	Basque
eu_ES	Basque (Spain)
fa	Persian
fa_IR	Persian (Iran)
ff_SN	Fulah (Senegal)
fi	Finnish
fi_FI	Finnish (Finland)
fil_PH	Filipino (Philippines)
fo_FO	Faroese (Faroe Islands)
fr	French
fr_BE	French (Belgium)
fr_CA	French (Canada)
fr_CH	French (Switzerland)
fr_FR	French (France)
fr_LU	French (Luxembourg)
fur_IT	Friulian (Italy)
Western Frisian (Germany)	
Western Frisian (Netherlands)	
ga	Irish
ga_IE	Irish (Ireland)
gd_GB	Scottish Gaelic (United Kingdom)
gez_ER	Geez (Eritrea)
gez_ET	Geez (Ethiopia)
gl_ES	Galician (Spain)
gu_IN	Gujarati (India)
gv_GB	Manx (United Kingdom)
hak_TW	Hakka Chinese (Taiwan)
ha_NG	Hausa (Nigeria)
he	Hebrew
he_IL	Hebrew (Israel)
hi	Hindi

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
hi_IN	Hindi (India)
hne_IN	Chhattisgarhi (India)
hr	Croatian
hr_HR	Croatian (Croatia)
hsb_DE	Upper Sorbian (Germany)
ht_HT	Haitian (Haiti)
hu	Hungarian
hu_HU	Hungarian (Hungary)
hus_MX	Huastec (Mexico)
hy_AM	Armenian (Armenia)
ia_FR	Interlingua (France)
id	Indonesian
id_ID	Indonesian (Indonesia)
ig_NG	Igbo (Nigeria)
ik_CA	Inupiaq (Canada)
is	Icelandic
is_IS	Icelandic (Iceland)
it	Italian
it_CH	Italian (Switzerland)
it_IT	Italian (Italy)
iu_CA	Inuktitut (Canada)
ja	Japanese
ja_JP	Japanese (Japan)
kab_DZ	Kabyle (Algeria)
ka_GE	Georgian (Georgia)
kk_KZ	Kazakh (Kazakhstan)
kl_GL	Kalaallisut (Greenland)
km_KH	Central Khmer (Cambodia)
kn_IN	Kannada (India)
kok_IN	Konkani (India)
ko	Korean
ko_KR	Korean (South Korea)
ks_IN	Kashmiri (India)
ku	Kurdish
ku_TR	Kurdish (Turkey)
kw_GB	Cornish (United Kingdom)
ky_KG	Kirghiz (Kyrgyzstan)
lb_LU	Luxembourgish (Luxembourg)
lg_UG	Ganda (Uganda)
li_BE	Limburgan (Belgium)
li_NL	Limburgan (Netherlands)
lij_IT	Ligurian (Italy)
ln_CD	Lingala (Congo)
lo_LA	Lao (Laos)
lt	Lithuanian
lt_LT	Lithuanian (Lithuania)
lv	Latvian
lv_LV	Latvian (Latvia)
lzh_TW	Literary Chinese (Taiwan)

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
mag_IN	Magahi (India)
mai_IN	Maithili (India)
mg_MG	Malagasy (Madagascar)
mh_MH	Marshallese (Marshall Islands)
mhr_RU	Eastern Mari (Russia)
mi_NZ	Maori (New Zealand)
miq_NI	Mískito (Nicaragua)
mk	Macedonian
mk_MK	Macedonian (Macedonia)
ml_IN	Malayalam (India)
mni_IN	Manipuri (India)
mn_MN	Mongolian (Mongolia)
mr_IN	Marathi (India)
ms	Malay
ms_MY	Malay (Malaysia)
mt	Maltese
mt_MT	Maltese (Malta)
my_MM	Burmese (Myanmar)
myv_RU	Erzya (Russia)
nah_MX	Nahuatl languages (Mexico)
nan_TW	Min Nan Chinese (Taiwan)
nb	Norwegian Bokmål
nb_NO	Norwegian Bokmål (Norway)
nds_DE	Low German (Germany)
nds_NL	Low German (Netherlands)
ne_NP	Nepali (Nepal)
nhn_MX	Central Nahuatl (Mexico)
niu_NU	Niuean (Niue)
niu_NZ	Niuean (New Zealand)
nl	Dutch
nl_AW	Dutch (Aruba)
nl_BE	Dutch (Belgium)
nl_NL	Dutch (Netherlands)
nn	Norwegian Nynorsk
nn_NO	Norwegian Nynorsk (Norway)
no	Norwegian
no_NO	Norwegian (Norway)
nr_ZA	South Ndebele (South Africa)
nso_ZA	Pedi (South Africa)
oc_FR	Occitan (France)
om	Oromo
om_ET	Oromo (Ethiopia)
om_KE	Oromo (Kenya)
or_IN	Oriya (India)
os_RU	Ossetian (Russia)
pa_IN	Punjabi (India)
pap	Papiamento
pap_AN	Papiamento (Netherlands Antilles)
pap_AW	Papiamento (Aruba)

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
pap_CW	Papiamento (Curaçao)
pa_PK	Punjabi (Pakistan)
pl	Polish
pl_PL	Polish (Poland)
pr	Pirate
ps_AF	Pushto (Afghanistan)
pt	Portuguese
pt_BR	Portuguese (Brazil)
pt_PT	Portuguese (Portugal)
quy_PE	Ayacucho Quechua (Peru)
quz_PE	Cusco Quechua (Peru)
raj_IN	Rajasthani (India)
ro	Romanian
ro_RO	Romanian (Romania)
ru	Russian
ru_RU	Russian (Russia)
ru_UA	Russian (Ukraine)
rw_RW	Kinyarwanda (Rwanda)
sa_IN	Sanskrit (India)
sat_IN	Santali (India)
sc_IT	Sardinian (Italy)
sco	Scots
sd_IN	Sindhi (India)
se_NO	Northern Sami (Norway)
sgs_LT	Samogitian (Lithuania)
shs_CA	Shuswap (Canada)
sid_ET	Sidamo (Ethiopia)
si_LK	Sinhala (Sri Lanka)
sk	Slovak
sk_SK	Slovak (Slovakia)
sl	Slovenian
so	Somali
so_DJ	Somali (Djibouti)
so_ET	Somali (Ethiopia)
so KE	Somali (Kenya)
so_SO	Somali (Somalia)
son_ML	Songhai languages (Mali)
sq	Albanian
sq_AL	Albanian (Albania)
sq_KV	Albanian (Kosovo)
sq_MK	Albanian (Macedonia)
sr	Serbian
sr_BA	Serbian (Bosnia and Herzegovina)
sr_CS	Serbian (Serbia and Montenegro)
sr_ME	Serbian (Montenegro)
sr_RS	Serbian (Serbia)
ss_ZA	Swati (South Africa)
st_ZA	Southern Sotho (South Africa)
sv	Swedish

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
sv_FI	Swedish (Finland)
sv_SE	Swedish (Sweden)
sw_KE	Swahili (Kenya)
sw_TZ	Swahili (Tanzania)
szl_PL	Silesian (Poland)
ta	Tamil
ta_IN	Tamil (India)
ta_LK	Tamil (Sri Lanka)
tcy_IN	Tulu (India)
te_IN	Telugu (India)
tg TJ	Tajik (Tajikistan)
the_NP	Chitwania Tharu (Nepal)
th	Thai
th_TH	Thai (Thailand)
th_TH TH	Thai (Thailand,TH)
ti	Tigrinya
ti_ER	Tigrinya (Eritrea)
ti_ET	Tigrinya (Ethiopia)
tig_ER	Tigre (Eritrea)
tk_TM	Turkmen (Turkmenistan)
tl_PH	Tagalog (Philippines)
tn_ZA	Tswana (South Africa)
tr	Turkish
tr_CY	Turkish (Cyprus)
tr_TR	Turkish (Turkey)
ts_ZA	Tsonga (South Africa)
tt_RU	Tatar (Russia)
ug_CN	Uighur (China)
uk	Ukrainian
uk_UA	Ukrainian (Ukraine)
unm_US	Unami (United States)
ur	Urdu
ur_IN	Urdu (India)
ur_PK	Urdu (Pakistan)
uz	Uzbek
uz_UZ	Uzbek (Uzbekistan)
ve_ZA	Venda (South Africa)
vi	Vietnamese
vi_VN	Vietnamese (Vietnam)
wa_BE	Walloon (Belgium)
wae_CH	Walser (Switzerland)
wal_ET	Wolaytta (Ethiopia)
wo_SN	Wolof (Senegal)
xh_ZA	Xhosa (South Africa)
yi_US	Yiddish (United States)
yo_NG	Yoruba (Nigeria)
yue_HK	Yue Chinese (Hong Kong)
zh	Chinese
zh_CN	Chinese (China)

Continued on next page

Table 21.1 – continued from previous page

Locale	Language and Variant
zh_HK	Chinese (Hong Kong)
zh_SG	Chinese (Singapore)
zh_TW	Chinese (Taiwan)
zu_ZA	Zulu (South Africa)

21.9 Binary serialization API

21.9.1 Introduction

Godot has a simple serialization API based on Variant. It's used for converting data types to an array of bytes efficiently. This API is used in the functions `get_var` and `store_var` of `File` as well as the packet APIs for `PacketPeer`. This format is not used for binary scenes and resources.

21.9.2 Packet specification

The packet is designed to be always padded to 4 bytes. All values are little endian encoded. All packets have a 4 byte header representing an integer, specifying the type of data:

Type	Value
0	null
1	bool
2	integer
3	float
4	string
5	vector2
6	rect2
7	vector3
8	transform2d
9	plane
10	quat
11	aabb
12	basis
13	transform
14	color
15	node path
16	rid
17	object
18	dictionary
19	array
20	raw array
21	int array
22	real array
23	string array
24	vector2 array
25	vector3 array
26	color array
27	max

Following this is the actual packet contents, which varies for each type of packet:

0: null**1: bool**

Offset	Len	Type	Description
4	4	Integer	0 for False, 1 for True

2: int

Offset	Len	Type	Description
4	4	Integer	Signed, 32-Bit Integer

3: float/real

Offset	Len	Type	Description
4	4	Float	IEE 754 32-Bits Float

4: String

Offset	Len	Type	Description
4	4	Integer	String Length (in Bytes)
8	X	Bytes	UTF-8 Encoded String

This field is padded to 4 bytes.

5: Vector2

Offset	Len	Type	Description
4	4	Float	X Coordinate
8	4	Float	Y Coordinate

6: Rect2

Offset	Len	Type	Description
4	4	Float	X Coordinate
8	4	Float	Y Coordinate
12	4	Float	X Size
16	4	Float	Y Size

7: Vector3

Offset	Len	Type	Description
4	4	Float	X Coordinate
8	4	Float	Y Coordinate
12	4	Float	Z Coordinate

8: Transform2D

Offset	Len	Type	Description
4	4	Float	[0][0]
8	4	Float	[0][1]
12	4	Float	[1][0]
16	4	Float	[1][1]
20	4	Float	[2][0]
24	4	Float	[2][1]

9: Plane

Offset	Len	Type	Description
4	4	Float	Normal X
8	4	Float	Normal Y
12	4	Float	Normal Z
16	4	Float	Distance

10: Quat

Offset	Len	Type	Description
4	4	Float	Imaginary X
8	4	Float	Imaginary Y
12	4	Float	Imaginary Z
16	4	Float	Real W

11: AABB

Offset	Len	Type	Description
4	4	Float	X Coordinate
8	4	Float	Y Coordinate
12	4	Float	Z Coordinate
16	4	Float	X Size
20	4	Float	Y Size
24	4	Float	Z Size

12: Basis

Offset	Len	Type	Description
4	4	Float	[0][0]
8	4	Float	[0][1]
12	4	Float	[0][2]
16	4	Float	[1][0]
20	4	Float	[1][1]
24	4	Float	[1][2]
28	4	Float	[2][0]
32	4	Float	[2][1]
36	4	Float	[2][2]

13: Transform

Offset	Len	Type	Description
4	4	Float	[0][0]
8	4	Float	[0][1]
12	4	Float	[0][2]
16	4	Float	[1][0]
20	4	Float	[1][1]
24	4	Float	[1][2]
28	4	Float	[2][0]
32	4	Float	[2][1]
36	4	Float	[2][2]
40	4	Float	[3][0]
44	4	Float	[3][1]
48	4	Float	[3][2]

14: Color

Offset	Len	Type	Description
4	4	Float	Red (0..1)
8	4	Float	Green (0..1)
12	4	Float	Blue (0..1)
16	4	Float	Alpha (0..1)

15: NodePath

Offset	Len	Type	Description
4	4	Integer	String Length, or New Format (val&0x80000000!=0 and Name-Count=val&0x7FFFFFFF)

For old format:

Offset	Len	Type	Description
8	X	Bytes	UTF-8 Encoded String

Padded to 4 bytes.

For new format:

Offset	Len	Type	Description
4	4	Integer	Sub-Name Count
8	4	Integer	Flags (absolute: val&1 != 0)

For each Name and Sub-Name

Offset	Len	Type	Description
X+0	4	Integer	String Length
X+4	X	Bytes	UTF-8 Encoded String

Every name string is padded to 4 bytes.

16: RID (unsupported)**17: Object (unsupported)****18: Dictionary**

Offset	Len	Type	Description
4	4	Integer	val&0x7FFFFFFF = elements, val&0x80000000 = shared (bool)

Then what follows is, for amount of “elements”, pairs of key and value, one after the other, using this same format.

19: Array

Offset	Len	Type	Description
4	4	Integer	val&0x7FFFFFFF = elements, val&0x80000000 = shared (bool)

Then what follows is, for amount of “elements”, values one after the other, using this same format.

20: PoolByteArray

Offset	Len	Type	Description
4	4	Integer	Array Length (Bytes)
8..8+length	1	Byte	Byte (0..255)

The array data is padded to 4 bytes.

21: PoolIntArray

Offset	Len	Type	Description
4	4	Integer	Array Length (Integers)
8..8+length*4	4	Integer	32 Bits Signed Integer

22: PoolRealArray

Offset	Len	Type	Description
4	4	Integer	Array Length (Floats)
8..8+length*4	4	Integer	32 Bits IEE 754 Float

23: PoolStringArray

Offset	Len	Type	Description
4	4	Integer	Array Length (Strings)

For each String:

Offset	Len	Type	Description
X+0	4	Integer	String Length
X+4	X	Bytes	UTF-8 Encoded String

Every string is padded to 4 bytes.

24: PoolVector2Array

Offset	Len	Type	Description
4	4	Integer	Array Length
8..8+length*8	4	Float	X Coordinate
8..12+length*8	4	Float	Y Coordinate

25: PoolVector3Array

Offset	Len	Type	Description
4	4	Integer	Array Length
8..8+length*12	4	Float	X Coordinate
8..12+length*12	4	Float	Y Coordinate
8..16+length*12	4	Float	Z Coordinate

26: PoolColorArray

Offset	Len	Type	Description
4	4	Integer	Array Length
8..8+length*16	4	Float	Red (0..1)
8..12+length*16	4	Float	Green (0..1)
8..16+length*16	4	Float	Blue (0..1)
8..20+length*16	4	Float	Alpha (0..1)

CHAPTER 22

Compiling

22.1 Introduction to the buildsystem

22.1.1 SCons

Godot uses [SCons](#) to build. We love it, we are not changing it for anything else. We are not even sure other build systems are up to the task of building Godot. We constantly get requests to move the build system to CMake, or Visual Studio, but this is not going to happen. There are many reasons why we have chosen SCons over other alternatives, for example:

- Godot can be compiled for a dozen different platforms. All PC platforms, all mobile platforms, many consoles, and many web-based platforms (such as HTML5 and Chrome PNACL).
- Developers often need to compile for several of the platforms **at the same time**, or even different targets of the same platform. They can't afford reconfiguring and rebuilding the project each time. SCons can do this with no sweat, without breaking the builds.
- SCons will *never* break a build no matter how many changes, configurations, additions, removals etc. You have more chances to die struck by lightning than needing to clean and rebuild in SCons.
- Godot build process is not simple. Several files are generated by code (binders), others are parsed (shaders), and others need to offer customization (plugins). This requires complex logic which is easier to write in an actual programming language (like Python) rather than using a mostly macro-based language only meant for building.
- Godot build process makes heavy use of cross compiling tools. Each platform has a specific detection process, and all these must be handled as specific cases with special code written for each.

So, please try to keep an open mind and get at least a little familiar with it if you are planning to build Godot yourself.

22.1.2 Setup

Please refer to the documentation for *Compiling for Android*, *Compiling for iOS*, *Compiling for OSX*, *Compiling for Universal Windows Platform*, *Compiling for the Web*, *Compiling for Windows* and *Compiling for X11 (Linux, *BSD)*.

Note that for **Windows/Visual Studio**, you need to use `x86_x64 Cross Tools Command Prompt` for VS 2017 or similar, depending on your install, instead of the standard Windows command prompt to enter the commands below.

22.1.3 Platform selection

Godot's build system will begin by detecting the platforms it can build for. If not detected, the platform will simply not appear on the list of available platforms. The build requirements for each platform are described in the rest of this tutorial section.

SCons is invoked by just calling `scons`.

However, this will do nothing except list the available platforms, for example:

```
user@host:~/godot$ scons
scons: Reading SConscript files ...
No valid target platform selected.
The following were detected:
    android
    server
    javascript
    windows
    x11

Please run scons again with argument: platform=<string>
scons: done reading SConscript files.
scons: Building targets ...
scons: `.' is up to date.
scons: done building targets.
```

To build for a platform (for example, `x11`), run with the `platform=` (or just `p=` to make it short) argument:

```
user@host:~/godot$ scons platform=x11
```

This will start the build process, which will take a while. If you want `scons` to build faster, use the `-j <cores>` parameter to specify how many cores will be used for the build. Or just leave it using one core, so you can use your computer for something else :)

Example for using 4 cores:

```
user@host:~/godot$ scons platform=x11 -j 4
```

Note that there are currently [issues](#) with parallel builds for at least some users, so if you are running into errors, try building without the `-j` parameter.

22.1.4 Resulting binary

The resulting binaries will be placed in the `bin/` subdirectory, generally with this naming convention:

```
godot.<platform>.[opt].[tools/debug].<architecture>[extension]
```

For the previous build attempt the result would look like this:

```
user@host:~/godot$ ls bin
bin/godot.x11.tools.64
```

This means that the binary is for X11, is not optimized, has tools (the whole editor) compiled in, and is meant for 64 bits.

A Windows binary with the same configuration will look like this.

```
C:\GODOT> DIR BIN/
godot.windows.tools.64.exe
```

Just copy that binary to wherever you like, as it contains the project manager, editor and all means to execute the game. However, it lacks the data to export it to the different platforms. For that the export templates are needed (which can be either downloaded from godotengine.org, or you can build them yourself).

Aside from that, there are a few standard options that can be set in all build targets, and which will be explained below.

22.1.5 Tools

Tools are enabled by default in all PC targets (Linux, Windows, macOS), disabled for everything else. Disabling tools produces a binary that can run projects but that does not include the editor or the project manager.

```
scons platform=<platform> tools=yes/no
```

22.1.6 Target

Target controls optimization and debug flags. Each mode means:

- **debug**: Build with C++ debugging symbols, runtime checks (performs checks and reports error) and none to little optimization.
- **release_debug**: Build without C++ debugging symbols and optimization, but keep the runtime checks (performs checks and reports errors). Official binaries use this configuration.
- **release**: Build without symbols, with optimization and with little to no runtime checks. This target can't be used together with tools=yes, as the tools require some debug functionality and run-time checks to run.

```
scons platform=<platform> target=debug/release_debug/release
```

This flag appends the “.debug” suffix (for debug), or “.tools” (for debug with tools enabled). When optimization is enabled (release) it appends the “.opt” suffix.

22.1.7 Bits

Bits is meant to control the CPU or OS version intended to run the binaries. It is focused mostly on desktop platforms and ignored everywhere else.

- **32**: Build binaries for 32 bits platform.
- **64**: Build binaries for 64 bits platform.
- **default**: Build whatever the build system feels is best. On Linux this depends on the host platform (if not cross compiling), on Mac it defaults to 64 bits and on Windows it defaults to 32 bits.

```
scons platform=<platform> bits=default/32/64
```

This flag appends “.32” or “.64” suffixes to resulting binaries when relevant.

22.1.8 Export templates

Official export templates are downloaded from the Godot Engine site: godotengine.org. However, you might want to build them yourself (in case you want newer ones, you are using custom modules, or simply don't trust your own shadow).

If you download the official export templates package and unzip it, you will notice that most are just optimized binaries or packages for each platform:

```
android_debug.apk  
android_release.apk  
javascript_debug.zip  
javascript_release.zip  
linux_server_32  
linux_server_64  
linux_x11_32_debug  
linux_x11_32_release  
linux_x11_64_debug  
linux_x11_64_release  
osx.zip  
version.txt  
windows_32_debug.exe  
windows_32_release.exe  
windows_64_debug.exe  
windows_64_release.exe
```

To create those yourself, just follow the instructions detailed for each platform in this same tutorial section. Each platform explains how to create its own template.

If you are developing for multiple platforms, macOS is definitely the most convenient host platform for cross compilation, since you can cross-compile for almost every target (except for UWP). Linux and Windows come in second place, but Linux has the advantage of being the easier platform to set this up.

22.2 Compiling for Windows

22.2.1 Requirements

For compiling under Windows, the following is required:

- Visual C++, [Visual Studio Community](#) (recommended), version 2013 (12.0) or later. **Make sure you read [Installing Visual Studio](#) caveats below or you will have to run/download the installer again.**
- Python 2.7+ or Python 3.5+.
- Pywin32 Python Extension for parallel builds (which increase the build speed by a great factor).
- SCons build system.

22.2.2 Setting up SCons

Python adds the interpreter (python.exe) to the path. It usually installs in C:\Python (or C:\Python[Version]). SCons installs inside the Python install (typically in the Scripts folder) and provides a batch file called scons.bat. The location of this file can be added to the path or it can simply be copied to C:\Python together with the interpreter executable.

To check whether you have installed Python and SCons correctly, you can type `python --version` and `scons --version` into the Windows Command Prompt (`cmd.exe`).

If commands above do not work, make sure you add Python to your PATH environment variable after installing it, and check again.

22.2.3 Setting up Pywin32

Pywin32 is required for parallel builds using multiple CPU cores. If SCons is issuing a warning about Pywin32 after parsing SConstruct build instructions, when beginning to build, you need to install it properly from the correct installer executable for your Python version [located at Sourceforge](#).

For example, if you installed a 32-bit version of Python 2.7, you would want to install the latest version of Pywin32 that is built for the mentioned version of Python. That executable installer would be named `pywin32-221.win32-py2.7.exe`.

The `amd64` version of Pywin32 is for a 64-bit version of Python `pywin32-221.win-amd64-py2.7.exe`. Change the `py` number to install for your version of Python (check via `python --version` mentioned above).

22.2.4 Installing Visual Studio caveats

If installing Visual Studio 2015 or later, make sure to run **Custom** installation, not **Typical** and select C++ as language there (and any other things you might need). The installer does not install C++ by default. C++ was the [only language made optional](#) in Visual Studio 2015.

If you have already made the mistake of installing a **Typical**, installation, rerun the executable installer you downloaded from internet, it will give you a **Modify** Button option. Running the install from Add/Remove programs will only give you the “Repair” option, which will do nothing for your problem.

If you’re using Express, make sure you get/have a version that can compile for ***C++, Desktop***.

22.2.5 Downloading Godot’s source

Godot’s source is hosted on GitHub. Downloading it (cloning) via [Git](#) is recommended.

The tutorial will presume from now on that you placed the source into `C:\godot`.

22.2.6 Compiling

SCons will not be able out of the box to compile from the Windows Command Prompt (`cmd.exe`) because SCons and Visual C++ compiler will not be able to locate environment variables and executables they need for compilation.

Therefore, you need to start a Visual Studio command prompt. It sets up environment variables needed by SCons to locate the compiler. It should be called similar to one of the below names (for your respective version of Visual Studio):

- “Developer Command Prompt for VS2013”
- “VS2013 x64 Native Tools Command Prompt”
- “VS2013 x86 Native Tools Command Prompt”
- “VS2013 x64 Cross Tools Command Prompt”
- “VS2013 x86 Cross Tools Command Prompt”

You should be able to find at least the Developer Command Prompt for your version of Visual Studio in your start menu.

However Visual Studio sometimes seems to not install some of the above shortcuts, except the Developer Console at these locations that are automatically searched by the start menu search option:

```
Win 7:  
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2015\Visual Studio  
    ↵Tools  
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2013\Visual Studio  
    ↵Tools
```

If you found the Developer Console, it will do for now to create a 32-bit version of Godot, but if you want the 64-bit version, you might need to setup the prompts manually for easy access.

If you don't see some of the shortcuts, "How the prompts actually work" section below will explain how to setup these prompts if you need them.

About the Developer/Tools Command Prompts and the Visual C++ compiler

There is a few things you need to know about these consoles and the Visual C++ compiler.

Your Visual Studio installation will ship with several Visual C++ compilers, them being more or less identical, however each `cl.exe` (Visual C++ compiler) will compile Godot for a different architecture (32-bit x86 or 64-bit x86; the ARM compiler is not supported).

The **Developer Command Prompt** will build a 32-bit version of Godot by using the 32-bit Visual C++ compiler.

Native Tools Prompts (mentioned above) are used when you want the 32-bit `cl.exe` to compile a 32-bit executable (x86 Native Tools Command Prompt). For the 64-bit `cl.exe`, it will compile a 64-bit executable (x64 Native Tools Command Prompt).

The **Cross Tools** are used when your Windows is using one architecture (32-bit, for example) and you need to compile to a different architecture (64-bit). As you might be familiar, 32-bit Windows can not run 64-bit executables, but you still might need to compile for them.

For example:

- "VS2013 x64 Cross Tools Command Prompt" will use a 32-bit `cl.exe` that will compile a 64 bit application.
- "VS2013 x86 Cross Tools Command Prompt" will use a 64-bit `cl.exe` that will compile a 32-bit application.
This one is useful if you are running a 32-bit Windows.

On a 64-bit Windows, you can run any of above prompts and compilers (`cl.exe` executables) because 64-bit Windows can run any 32-bit application. 32-bit Windows cannot run 64-bit executables, so the Visual Studio installer won't even install shortcuts for some of these prompts.

Note that you need to choose the **Developer Console** or the correct **Tools Prompt** to build Godot for the correct architecture. Use only Native Prompts if you are not sure yet what exactly Cross Compile Prompts do.

Running SCons

Once inside the **Developer Console/Tools Console Prompt**, go to the root directory of the engine source code and type:

```
C:\godot> scons platform=windows
```

Tip: if you installed "Pywin32 Python Extension" you can append the `-j` command to instruct SCons to run parallel builds like this:

```
c:\godot> scons -j6 platform=windows
```

In general, it is OK to have at least as many threads compiling Godot as you have cores in your CPU, if not one or two more. Feel free to add the `-j` option to any SCons command you see below if you setup the “Pywin32 Python Extension”.

If all goes well, the resulting binary executable will be placed in `C:\godot\bin\` with the name of `godot.windows.tools.32.exe` or `godot.windows.tools.64.exe`. SCons will automatically detect what compiler architecture the environment (the prompt) is setup for and will build a corresponding executable.

This executable file contains the whole engine and runs without any dependencies. Executing it will bring up the Project Manager.

How the prompts actually work

The Visual Studio command prompts are just shortcuts that call the standard Command Prompt and have it run a batch file before giving you control. The batch file itself is called `vcvarsall.bat` and it sets up environment variables, including the PATH variable, so that the correct version of the compiler can be run. The Developer Command Prompt calls a different file called `VsDevCmd.bat` but none of the other tools that this batch file enables are needed by Godot/SCons.

Since you are probably using Visual Studio 2013 or 2015, if you need to recreate them manually, use the below folders, or place them on the desktop/taskbar:

```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2015\Visual Studio_
↪Tools
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2013\Visual Studio_
↪Tools
```

Start the creation of the shortcut by pressing the right mouse button/New/Shortcut in an empty place in your desired location.

Then copy one of these commands below for the corresponding tool you need into the “Path” and “Name” sections of the shortcut creation wizard, and fix the path to the batch file if needed.

- Visual Studio 2013 is in the “Microsoft Visual Studio 12.0” folder.
- Visual Studio 2015 is in the “Microsoft Visual Studio 14.0” folder.
- etc.

```
Name: Developer Command Prompt for VS2013
Path: %comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.
↪0\Common7\Tools\VsDevCmd.bat"""

Name: VS2013 x64 Cross Tools Command Prompt
Path: %comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.
↪bat"" x86_amd64

Name: VS2013 x64 Native Tools Command Prompt
Path: %comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.
↪bat"" amd64

Name: VS2013 x86 Native Tools Command Prompt
Path: %comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.
↪bat"" x86
```

```
Name: VS2013 x86 Cross Tools Command Prompt
Path: %comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.
      ↪bat"" amd64_x86
```

After you create the shortcut, in the shortcut's properties, that you can access by right clicking with your mouse on the shortcut itself, you can choose the starting directory of the command prompt ("Start in" field).

Some of these shortcuts (namely the 64-bit compilers) seem to not be available in the Express edition of Visual Studio or Visual C++. Before recreating the commands, make sure that cl.exe executables are present in one of these locations, they are the actual compilers for the architecture you want to build from the command prompt.

```
x86 (32-bit) cl.exe
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\cl.exe

x86 (32-bit) cl.exe for cross-compiling for 64-bit Windows.
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\x86_amd64\cl.exe

x64 (64-bit) cl.exe
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\amd64\cl.exe

x64 (64-bit) cl.exe for cross-compiling for 32-bit Windows.
C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\amd64_x86\cl.exe
```

In case you are wondering what these prompt shortcuts do, they call cmd.exe with the \k option and have it run a Batch file.

```
%comspec% - path to cmd.exe
\k - keep alive option of the command prompt
remainder - command to run via cmd.exe

cmd.exe \k(eep cmd.exe alive after commands behind this option run) ""runme.bat""_
↪with_this_option
```

How to run an automated build of Godot

If you just need to run the compilation process via a Batch file or directly in the Windows Command Prompt you need to use the following command:

```
"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat" x86
```

with one of the following parameters:

- x86 (32-bit cl.exe to compile for the 32-bit architecture)
- amd64 (64-bit cl.exe to compile for the 64-bit architecture)
- x86_amd64 (32-bit cl.exe to compile for the 64-bit architecture)
- amd64_x86 (64-bit cl.exe to compile for the 32-bit architecture)

and after that one, you can run SCons:

```
scons platform=windows
```

or you can run them together:

```

32-bit Godot
"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat" x86 && scons
↳platform=windows

64-bit Godot
"C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\vcvarsall.bat" amd64 && scons
↳platform=windows

```

22.2.7 Development in Visual Studio or other IDEs

For most projects, using only scripting is enough but when development in C++ is needed, for creating modules or extending the engine, working with an IDE is usually desirable.

You can create a Visual Studio solution via SCons by running SCons with the `vsproj=yes` parameter, like this:

```
scons p=windows vsproj=yes
```

You will be able to open Godot's source in a Visual Studio solution now, and able to build Godot via the Visual Studio **Build** button. However, make sure that you have installed Pywin32 so that parallel (-j) builds work properly.

If you need to edit the compilation commands, they are located in “Godot” project settings, NMAKE sheet. SCons is called at the very end of the commands. If you make a mistake, copy the command from one of the other build configurations (debug, release_debug, release) or architectures (Win32/x64). They are equivalent.

22.2.8 Cross-compiling for Windows from other operating systems

If you are a Linux or macOS user, you need to install [MinGW-w64](#), which typically comes in 32-bit and 64-bit variants. The package names may differ based on your distro, here are some known ones:

Arch	<code>pacman -S scons mingw-w64-gcc</code>
Debian / Ubuntu	<code>apt-get install scons mingw-w64</code>
Fedora	<code>dnf install scons mingw32-gcc-c++</code> ↳mingw64-gcc-c++
macOS	<code>brew install scons mingw-w64</code>
Mageia	<code>urpmi scons mingw32-gcc-c++ mingw64-gcc-</code> ↳c++

Before allowing you to attempt the compilation, SCons will check for the following binaries in your \$PATH:

```
i686-w64-mingw32-gcc
x86_64-w64-mingw32-gcc
```

If the binaries are not located in the \$PATH (e.g. /usr/bin), you can define the following environment variables to give a hint to the build system:

```
export MINGW32_PREFIX="/path/to/i686-w64-mingw32-"  
export MINGW64_PREFIX="/path/to/x86_64-w64-mingw32-"
```

To make sure you are doing things correctly, executing the following in the shell should result in a working compiler (the version output may differ based on your system):

```
user@host:~$ ${MINGW32_PREFIX}gcc --version  
i686-w64-mingw32-gcc (GCC) 6.1.0 20160427 (Mageia MinGW 6.1.0-1.mga6)
```

Troubleshooting

Cross-compiling from some versions of Ubuntu may lead to [this bug](#), due to a default configuration lacking support for POSIX threading.

You can change that configuration following those instructions, for 32-bit:

```
sudo update-alternatives --config i686-w64-mingw32-gcc  
<choose i686-w64-mingw32-gcc-posix from the list>  
sudo update-alternatives --config i686-w64-mingw32-g++  
<choose i686-w64-mingw32-g++-posix from the list>
```

And for 64-bit:

```
sudo update-alternatives --config x86_64-w64-mingw32-gcc  
<choose x86_64-w64-mingw32-gcc-posix from the list>  
sudo update-alternatives --config x86_64-w64-mingw32-g++  
<choose x86_64-w64-mingw32-g++-posix from the list>
```

22.2.9 Creating Windows export templates

Windows export templates are created by compiling Godot as release, with the following flags:

- (using Mingw32 command prompt, using the bits parameter)

```
C:\godot> scons platform=windows tools=no target=release bits=32  
C:\godot> scons platform=windows tools=no target=release_debug bits=32
```

- (using Mingw-w64 command prompt, using the bits parameter)

```
C:\godot> scons platform=windows tools=no target=release bits=64  
C:\godot> scons platform=windows tools=no target=release_debug bits=64
```

- (using the Visual Studio command prompts for the correct architecture, notice the lack of bits parameter)

```
C:\godot> scons platform=windows tools=no target=release  
C:\godot> scons platform=windows tools=no target=release_debug
```

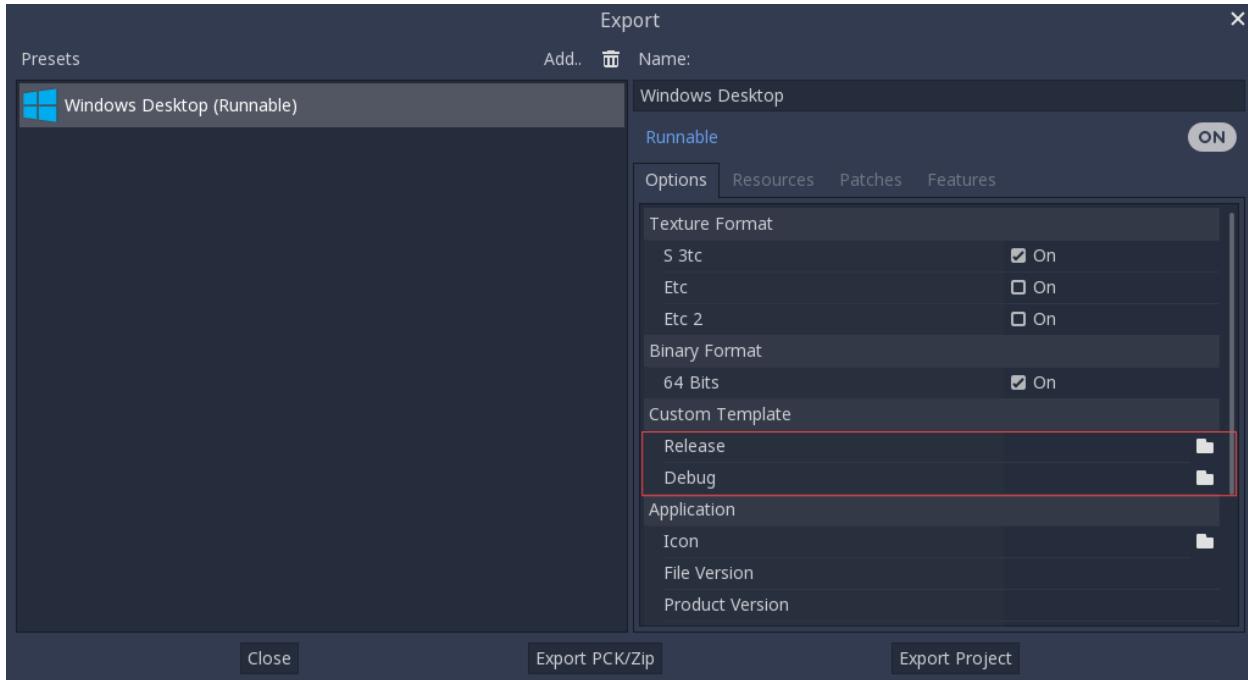
If you plan on replacing the standard templates, copy these to:

```
C:\USERS\YOURUSER\AppData\Roaming\Godot\Templates
```

With the following names:

```
windows_32_debug.exe
windows_32_release.exe
windows_64_debug.exe
windows_64_release.exe
```

However, if you are writing your custom modules or custom C++ code, you might instead want to configure your binaries as custom export templates here:



You don't even need to copy them, you can just reference the resulting files in the `bin\` directory of your Godot source folder, so the next time you build you automatically have the custom templates referenced.

22.3 Compiling for X11 (Linux, *BSD)

22.3.1 Requirements

For compiling under Linux or other Unix variants, the following is required:

- GCC or Clang
- Python 2.7+ (Python 3 only supported as of SCons 3.0)
- SCons build system
- pkg-config (used to detect the dependencies below)
- X11, Xcursor, Xinerama, Xi and XRandR development libraries
- MesaGL development libraries
- ALSA development libraries
- PulseAudio development libraries (for sound support)
- Freetype (for the editor)

- OpenSSL (for HTTPS and TLS)
- libudev (optional, build with *udev=yes*)

Distro-specific oneliners

Arch	pacman -S scons libxcursor libxinerama ↳ libxi libxrandr mesa glu alsalib ↳ pulseaudio freetype2
Debian / Ubuntu	sudo apt-get install build-essential ↳ scons pkg-config libX11-dev libxcursor- ↳ dev libxinerama-dev \ libgl1-mesa-dev libglu-dev ↳ libasound2-dev libpulse-dev ↳ libfreetype6-dev libssl-dev libudev- ↳ dev \ libxi-dev libxrandr-dev
Fedor a	sudo dnf install scons pkgconfig libX11- ↳ devel libXcursor-devel libXrandr-devel ↳ libXinerama-devel \ libXi-devel mesa-libGL-devel alsal- ↳ lib-devel pulseaudio-libs-devel ↳ freetype-devel openssl-devel \ libudev-devel mesa-libGLU-devel
FreeBSD	sudo pkg install scons pkg-config xorg- ↳ libraries libXcursor libXrandr libXi ↳ xineramaproto libglapi \ libGLU freetype2 openssl
Gentoo	emerge -an dev-util/scons x11-libs/ ↳ libX11 x11-libs/libXcursor x11-libs/ ↳ libXinerama x11-libs/libXi \ media-libs/mesa media-libs/glu media- ↳ libs/alsa-lib media-sound/pulseaudio ↳ media-libs/freetype
Mageia	urpmi scons pkgconfig "pkgconfig(alsa)" ↳ "pkgconfig(freetype2)" "pkgconfig(glu)" ↳ "pkgconfig(libpulse)" \ "pkgconfig(openssl)" "pkgconfig(udev)" ↳ "pkgconfig(x11)" "pkgconfig(xcursor)" ↳ "pkgconfig(xinerama)" \ "pkgconfig(xi)" "pkgconfig(xrandr)" ↳ "pkgconfig(zlib)"
OpenBSD	pkg_add python scons png llvm
openSUSE	sudo zypper install scons pkgconfig ↳ libX11-devel libXcursor-devel ↳ libXrandr-devel libXinerama-devel \ libXi-devel Mesa-libGL-devel ↳ alsal-devel libpulse-devel freetype- ↳ devel openssl-devel \ libudev-devel libGLU1
22.3. Compiling for X11 (Linux, *BSD)	68
Solus	sudo eopkg install -c system-devel scons

22.3.2 Compiling

Start a terminal, go to the root dir of the engine source code and type:

```
user@host:~/godot$ scons platform=x11
```

If all goes well, the resulting binary executable will be placed in the “bin” subdirectory. This executable file contains the whole engine and runs without any dependencies. Executing it will bring up the project manager.

Note: If you wish to compile using Clang rather than GCC, use this command:

```
user@host:~/godot$ scons platform=x11 use_llvm=yes
```

Using Clang appears to be a requirement for OpenBSD, otherwise fonts would not build.

22.3.3 Building export templates

To build X11 (Linux, *BSD) export templates, run the build system with the following parameters:

- (32 bits)

```
user@host:~/godot$ scons platform=x11 tools=no target=release bits=32
user@host:~/godot$ scons platform=x11 tools=no target=release_debug bits=32
```

- (64 bits)

```
user@host:~/godot$ scons platform=x11 tools=no target=release bits=64
user@host:~/godot$ scons platform=x11 tools=no target=release_debug bits=64
```

Note that cross compiling for the opposite bits (64/32) as your host platform is not always straight-forward and might need a chroot environment.

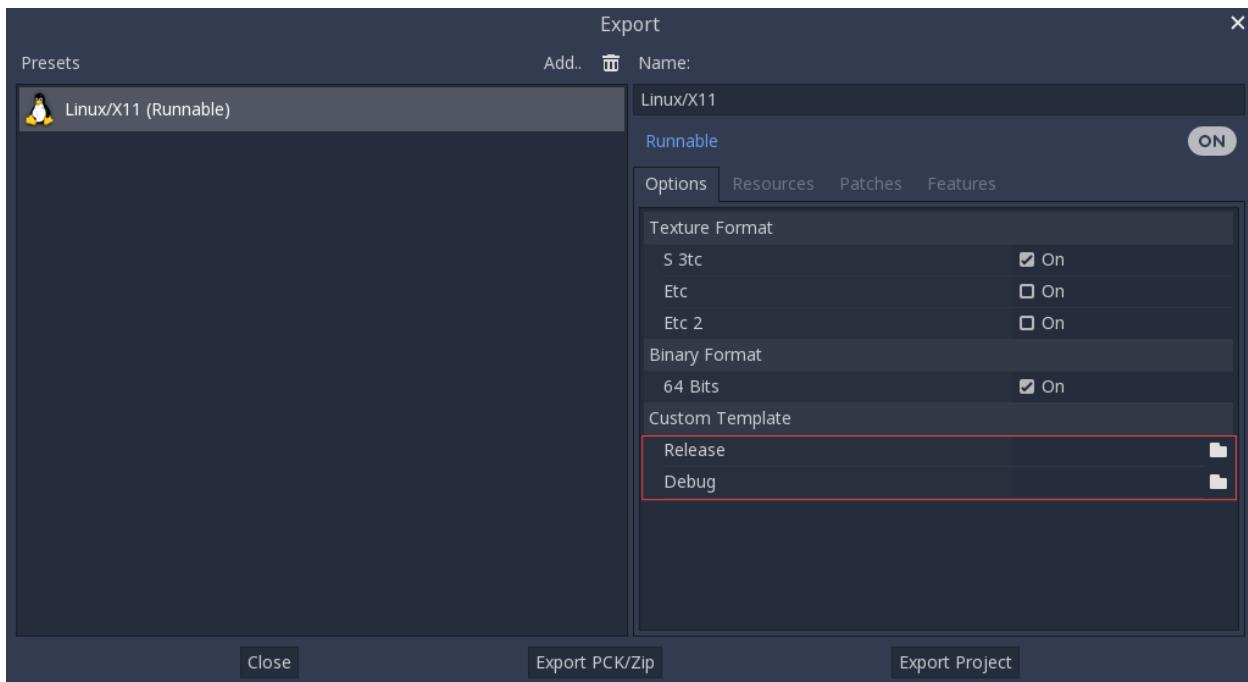
To create standard export templates, the resulting files must be copied to:

```
/home/youruser/.godot/templates
```

and named like this (even for *BSD which is seen as “Linux X11” by Godot):

```
linux_x11_32_debug
linux_x11_32_release
linux_x11_64_debug
linux_x11_64_release
```

However, if you are writing your custom modules or custom C++ code, you might instead want to configure your binaries as custom export templates here:



You don't even need to copy them, you can just reference the resulting files in the bin/ directory of your Godot source folder, so the next time you build you automatically have the custom templates referenced.

22.4 Compiling for OSX

22.4.1 Requirements

For compiling under Linux or other Unix variants, the following is required:

- Python 2.7+ or Python 3.5+
- SCons build system
- Xcode (or the more lightweight Command Line Tools for Xcode)

22.4.2 Compiling

Start a terminal, go to the root dir of the engine source code and type:

```
user@host:~/godot$ scons platform=osx
```

If all goes well, the resulting binary executable will be placed in the “bin” subdirectory. This executable file contains the whole engine and runs without any dependencies. Executing it will bring up the project manager.

To create an .app like in the official builds, you need to use the template located in misc/dist/osx_tools.app. Typically, for a “.64” optimised binary built with `scons p=osx target=release_debug`:

```
user@host:~/godot$ cp -r misc/dist/osx_tools.app ./Godot.app
user@host:~/godot$ mkdir -p Godot.app/Contents/MacOS
user@host:~/godot$ cp bin/godot.osx.opt.tools.64 Godot.app/Contents/MacOS/Godot
user@host:~/godot$ chmod +x Godot.app/Contents/MacOS/Godot
```

22.4.3 Compiling for 32 and 64-bit

All macOS versions after 10.6 are 64-bit exclusive, so the executable will be a “.64” file by default for most users. If you would like to compile a “.fat” executable which contains both 32 and 64-bit code, you can do so by specifying the bits in the scons command like so:

```
user@host:~/godot$ scons platform=osx bits=fat
```

22.4.4 Cross-compiling

It is possible to compile for OSX in a Linux environment (and maybe also in Windows with Cygwin). For that you will need [OSXCross](#) to be able to use OSX as target. First, follow the instructions to install it:

Clone the *OSXCross repository* <<https://github.com/tpoechtrager/osxcross>> somewhere on your machine (or download a zip file and extract it somewhere), e.g.:

```
user@host:~$ git clone https://github.com/tpoechtrager/osxcross.git /home/myuser/  
→sources/osxcross
```

1. Follow the instructions to package the SDK: <https://github.com/tpoechtrager/osxcross#packaging-the-sdk>
2. Follow the instructions to install OSXCross: <https://github.com/tpoechtrager/osxcross#installation>

After that, you will need to define the `OSXCROSS_ROOT` as the path to the OSXCross installation (the same place where you cloned the repository/extracted the zip), e.g.:

```
user@host:~$ export OSXCROSS_ROOT=/home/myuser/sources/osxcross
```

Now you can compile with SCons like you normally would:

```
user@host:~/godot$ scons platform=osx
```

If you have an OSXCross SDK version different from the one expected by the SCons buildsystem, you can specify a custom one with the `osxcross_sdk` argument:

```
user@host:~/godot$ scons platform=osx osxcross_sdk=darwin15
```

22.5 Compiling for Android

22.5.1 Note

For most cases, using the built-in deployer and export templates is good enough. Compiling the Android APK manually is mostly useful for custom builds or custom packages for the deployer.

Also, you still need to do all the steps mentioned in the *Exporting for Android* tutorial before attempting your custom export template.

22.5.2 Requirements

For compiling under Windows, Linux or macOS, the following is required:

- Python 2.7+ or Python 3.5+
- SCons build system

- [Windows only] PyWin32 (optional, for parallel compilation)
- Android SDK version 23.0.3 [Note: Please install all tools and extras of the SDK Manager]
- Android build tools version 19.1
- Android NDK r13 or later
- Gradle (will be downloaded and installed automatically if missing)
- JDK 6 or later (either OpenJDK or Oracle JDK) - JDK 9 is untested as of now

22.5.3 Setting up the buildsystem

Set the environment variable ANDROID_HOME to point to the Android SDK.

Set the environment variable ANDROID_NDK_ROOT to point to the Android NDK.

To set those environment variables on Windows, press Windows+R, type “control system”, then click on **Advanced system settings** in the left pane, then click on **Environment variables** on the window that appears.

To set those environment variables on Unix (e.g. Linux, macOS), use `export ANDROID_HOME=/path/to/android-sdk` and `export ANDROID_NDK_ROOT=/path/to/android-ndk`. Where /path/to/android-sdk and /path/to/android-ndk is the path where Android SDK and Android NDK are placed on your PC.

Toolchain

We usually try to keep the Godot Android build code up to date, but Google changes their toolchain versions very often, so if compilation fails due to wrong toolchain version, go to your NDK directory and check the current number, then set the following environment variable:

```
NDK_TARGET (by default set to "arm-linux-androideabi-4.9")
```

22.5.4 Building the export templates

Godot needs two export templates for Android: the optimized “release” template (*android_release.apk*) and the debug version (*android_debug.apk*). Compiling the standard export templates is done by calling scons with the following arguments:

- Release template (used when exporting with “Debugging Enabled” OFF)

```
C:\godot> scons platform=android target=release
C:\godot> cd platform/android/java
C:\godot\platform\android\java> gradlew build
```

(on Linux or macOS, execute the *gradlew* script with *./gradlew build*)

The resulting APK is in:

```
bin\android_release.apk
```

- Debug template (used when exporting with “Debugging Enabled” ON)

```
C:\godot> scons platform=android target=release_debug
C:\godot> cd platform/android/java
C:\godot\platform\android\java> gradlew build
```

The resulting APK is in:

```
bin\android_debug.apk
```

Faster compilation

If you are on Unix or installed PyWin32 on Windows and have multiple CPU cores available, you can speed up the compilation by adding the `-jX` argument to the SCons command, where X is the number of cores that you want to allocate to the compilation, e.g. `scons -j4`.

Adding support for x86 devices

If you also want to include support for x86 devices, run the `scons` command a second time with the `android_arch=x86` argument before building the APK with Gradle. For example for the release template:

```
C:\godot> scons platform=android target=release
C:\godot> scons platform=android target=release android_arch=x86
C:\godot> cd platform/android/java
C:\godot\platform\android\java> gradlew build
```

This will create a fat binary that works in both platforms, but will add about 6 megabytes to the APK.

Troubleshooting

It might be necessary to clean the build cache between two APK compilations, as some users have reported issues when building the two export templates one after the other.

22.5.5 Using the export templates

As export templates for Android, Godot needs release and debug APKs that were compiled against the same version/commit as the editor. If you are using official binaries for the editor, make sure to install the matching export templates, or to build your own from the same version.

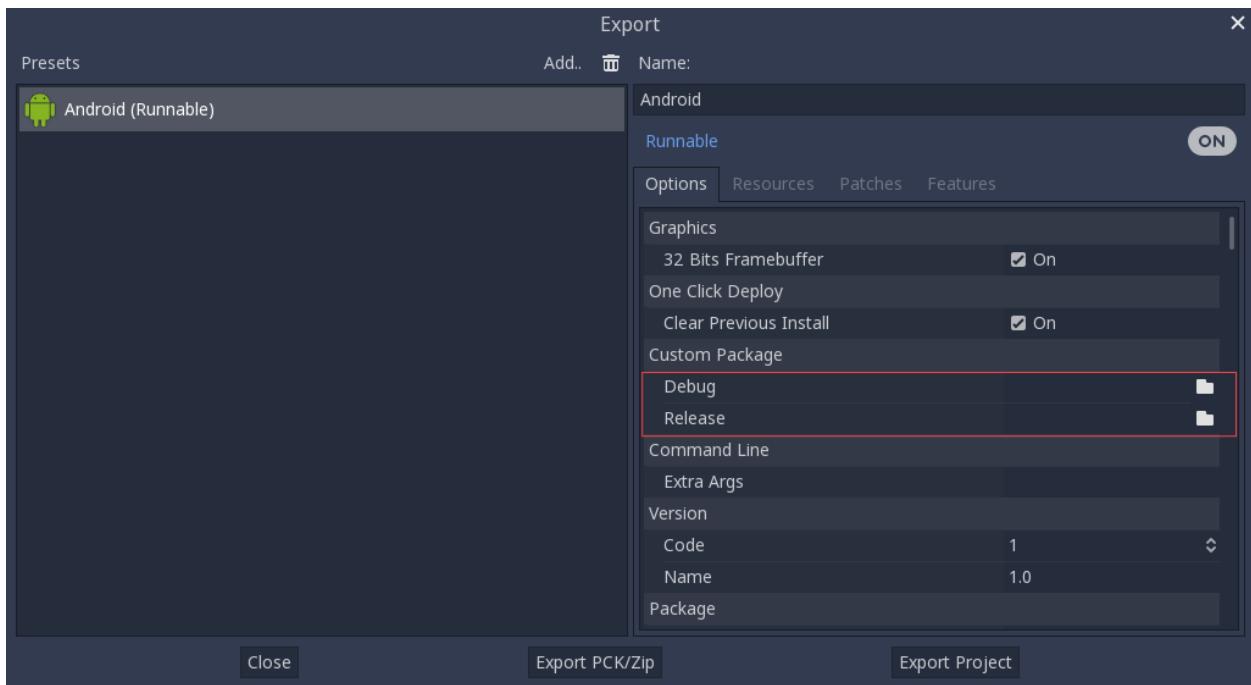
When exporting your game, Godot opens the APK, changes a few things inside, adds your file and spits it back. It's really handy! (and required some reverse engineering of the format).

Installing the templates

The newly-compiled templates (`android_debug.apk` and `android_release.apk`) must be copied to Godot's templates folder with their respective names. The templates folder can be located in:

- Windows: `C:\Users\[username]\AppData\Roaming\Godot\templates`
- Linux: `/home/[username]/.godot/templates`
- macOS: `/users/[username]/.godot/templates`

However, if you are writing your custom modules or custom C++ code, you might instead want to configure your APKs as custom export templates here:



You don't even need to copy them, you can just reference the resulting file in the `bin\` directory of your Godot source folder, so that the next time you build you will automatically have the custom templates referenced.

22.5.6 Troubleshooting

Application not installed

Android might complain the application is not correctly installed. If so, check the following:

- Check that the debug keystore is properly generated.
- Check that jarsigner is from JDK 6, 7 or 8.

If it still fails, open a command line and run logcat:

```
C:\android-sdk\platform-tools> adb logcat
```

And check the output while the application is installed. Reason for failure should be presented there.

Seek assistance if you can't figure it out.

Application exits immediately

If the application runs but exits immediately, there might be one of the following reasons:

- Make sure to use export templates that match your editor version; if you use a new Godot version, you *have* to update the templates too.
- `libgodot_android.so` is not in `lib/armeabi-v7a` or `lib/armeabi`
- Device does not support armv7 (try compiling yourself for armv6)
- Device is Intel, and apk is compiled for ARM.

In any case, `adb logcat` should also show the cause of the error.

Compilation fails

On Linux systems with Kernel version 4.3 or newer, compilation may fail with the error “pthread_create failed: Resource temporarily unavailable.”

This is because of a change in the way Linux limits thread creation. But you can change those limits through the command line. Please read this section thoroughly before beginning.

First open a terminal, then begin compilation as usual (it may be a good idea to run a –clean first). While compiling enter the following in your terminal:

```
user@host:~/ $ top -b -n 1 | grep scons
```

The output should list a scons process, with its PID as the first number in the output. For example the PID 1077 in the output shown below:

```
user@host:~/ $ top -b -n 1 | grep scons
1077 user      20      0    10544    1628    1508 S 0.000 0.027  0:00.00 grep
```

Now you can use another command to increase the number of processes that scons is allowed to spawn. You can check its current limits with:

```
user@host:~/ $ prlimit --pid=1077 --nproc
```

You can increase those limits with the command:

```
user@host:~/ $ prlimit --pid=1077 --nproc=60000:60500
```

Obviously you should substitute the scons PID output by top and a limits that you think suitable. These are in the form –nproc=soft:hard where soft must be lesser than or equal to hard. See the man page for more information.

If all went well, and you entered the prlimit command while scons was running, then your compilation should continue without the error.

22.6 Compiling for iOS

22.6.1 Requirements

- SCons (you can get it from macports, you should be able to run scons in a terminal when installed)
- Xcode with the iOS SDK and the command line tools.

22.6.2 Compiling

Open a Terminal, go to the root dir of the engine source code and type:

```
$ scons p=iphone target=debug
```

for a debug build, or:

```
$ scons p=iphone target=release
```

for a release build (check platform/iphone/detect.py for the compiler flags used for each configuration).

Alternatively, you can run

```
$ scons p=iphone arch=x86 target=debug
```

for a Simulator executable.

Additionally since some time Apple requires 64 bit version of application binary when you are uploading to iStore. The best way to provide these is to create a bundle in which there are both 32bit and 64 binaries, so every device will be able to run the game. It can be done in three steps, first compile 32 bit version, then compile 64 bit version and then use lipo to bundle them into one fat binary, all those steps can be performed with following commands:

```
$ scons p=iphone tools=no bits=32 target=release arch=arm
$ scons p=iphone tools=no bits=64 target=release arch=arm64
$ lipo -create bin/godot.ipHONE.opt.32 bin/godot.ipHONE.opt.64 -output bin/godot.ipHONE.opt.universal
```

22.6.3 Run

To run on a device or simulator, follow these instructions: *Exporting for iOS*.

Replace or add your executable to the Xcode project, and change the “executable name” property on Info.plist accordingly if you use an alternative build.

22.7 Cross-compiling for iOS on Linux

The procedure for this is somewhat complex and requires a lot of steps, but once you have the environment properly configured it will be easy to compile Godot for iOS anytime you want.

22.7.1 Disclaimer

While it is possible to compile for iOS on a Linux environment, Apple is very restrictive about the tools to be used (especially hardware-wise), allowing pretty much only their products to be used for development. So this is **not official**. However, a [statement from Apple in 2010](#) says they relaxed some of the [App Store review guidelines](#) to allow any tool to be used, as long as the resulting binary does not download any code, which means it should be OK to use the procedure described here and cross-compiling the binary.

22.7.2 Requirements

- [XCode with the iOS SDK](#) (a dmg image)
- [Clang >= 3.5](#) for your development machine installed and in the `PATH`. It has to be version ≥ 3.5 to target `arm64` architecture.
- [Fuse](#) for mounting and umounting the dmg image.
- [darling-dmg](#), which needs to be built from source. The procedure for that is explained below.
 - For building darling-dmg, you’ll need the development packages of the following libraries: `fuse`, `icu`, `openssl`, `zlib`, `bzip2`.
- [cctools-port](#) for the needed build tools. The procedure for building is quite peculiar and is described below.
 - This also has some extra dependencies: `automake`, `autogen`, `libtool`.

22.7.3 Configuring the environment

darling-dmg

Clone the repository on your machine:

```
$ git clone https://github.com/darlinghq/darling-dmg.git
```

Build it:

```
$ cd darling-dmg
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ make -j 4 # The number is the amount of cores your processor has, for faster build
$ cd ..
```

Preparing the SDK

Mount the XCode image:

```
$ mkdir xcode
$ ./darling-dmg/build/darling-dmg /path/to/Xcode_7.1.1.dmg xcode
[...]
Everything looks OK, disk mounted
```

Extract the iOS SDK:

```
$ mkdir -p iPhoneSDK/iPhoneOS9.1.sdk
$ cp -r xcode/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/
  ↪iPhoneOS.sdk/* iPhoneSDK/iPhoneOS9.1.sdk
$ cp -r xcode/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/
  ↪include/c++/* iPhoneSDK/iPhoneOS9.1.sdk/usr/include/c++
$ fusermount -u xcode # unmount the image
```

Pack the SDK:

```
$ cd iPhoneSDK
$ tar -cf - * | xz -9 -c - > iPhoneOS9.1.sdk.tar.xz
```

Toolchain

Build cctools:

```
$ git clone https://github.com/tpoechtrager/cctools-port.git
$ cd cctools-port/usage_examples/ios_toolchain
$ ./build.sh /path/iPhoneOS9.1.sdk.tar.xz arm64
```

Copy the tools to a nicer place. Note that the SCons scripts for building will look under `usr/bin` inside the directory you provide for the toolchain binaries, so you must copy to such subdirectory, akin to the following commands:

```
$ mkdir -p /home/user/iostoolchain/usr
$ cp -r target/bin /home/user/iostoolchain/usr/
```

Now you should have the iOS toolchain binaries in `/home/user/iostoolchain/usr/bin`.

22.7.4 Compiling Godot for iPhone

Once you've done the above steps, you should keep two things in your environment: the built toolchain and the iPhoneOS SDK directory. Those can stay anywhere you want since you have to provide their paths to the SCons build command.

For the iPhone platform to be detected, you need the `OSXCROSS_IOS` environment variable defined to anything.

```
$ export OSXCROSS_IOS=anything
```

Now you can compile for iPhone using SCons like the standard Godot way, with some additional arguments to provide the correct paths:

```
$ scons -j 4 platform=iphone arch=arm target=release_debug IPHONESDK="/path/to/iphoneSDK" IPHONEPATH="/path/to/iostoolchain" ios_triple="arm-apple-darwin11-"
$ scons -j 4 platform=iphone arch=arm64 target=release_debug IPHONESDK="/path/to/iphoneSDK" IPHONEPATH="/path/to/iostoolchain" ios_triple="arm-apple-darwin11-"
```

Producing fat binaries

Apple requires a fat binary with both architectures (armv7 and arm64) in a single file. To do this, use the `arm-apple-darwin11-lipo` executable. The following example assumes you are in the root Godot source directory:

```
$ /path/to/iostoolchain/usr/bin/arm-apple-darwin11-lipo -create bin/godot.ipHONE.opt.debug.arm bin/godot.ipHONE.opt.debug.arm64 -output bin/godot.ipHONE.opt.debug.fat
```

Then you will have an iOS fat binary in `bin/godot.ipHONE.opt.debug.fat`.

22.8 Compiling for Universal Windows Platform

22.8.1 Requirements

- SCons (see *Compiling for Windows* for more details).
- Visual Studio 2015 Update 2. It may work with earlier versions. See *Compiling for Windows* about the caveats of installing it and the various prompts.
- Windows 10 SDK (can be selected in Visual Studio installation).
- ANGLE source. Use the `ms_master` (default) branch. Keep it in a path without spaces to avoid problems.

22.8.2 Compiling

You need to open a proper Visual Studio prompt for the target architecture you want to build. Check *Compiling for Windows* to see how these prompts work.

There are three target architectures for UWP: x86 (32-bits), x64 (64-bits) and ARM (32-bits). For the latter, you can run `vcvarsall.bat` with `x86_arm` or `amd64_arm` as argument to set the environment.

Set the `ANGLE_SRC_PATH` to the directory where you downloaded the ANGLE source code. The build process will also build ANGLE to produce the required DLLs for the selected architecture.

Once you're set, run the SCons command similarly to the other platforms:

```
C:\godot>scons platform=uwp
```

22.8.3 Creating UWP export templates

To export using the editor you need to properly build package the templates. You need all three architectures with debug and release templates to be able to export.

Open the command prompt for one architecture and run SCons twice (once for each target):

```
C:\godot>scons platform=uwp target=release_debug  
C:\godot>scons platform=uwp target=release
```

Repeat for the other architectures.

In the end your `bin` folder will have the `.exe` binaries with a name like `godot.uwp.opt.debug.32.x86.exe` (with variations for each target/arch).

Copy one of these to `misc/dist/uwp_template` inside the Godot source folder and rename the binary to `godot.uwp.exe`. From the ANGLE source, under `winrt/10/src/Release_%arch%` (where `%arch%` can be Win32, x64 or ARM), get the `libEGL.dll` and the `libGLESv2.dll`, putting them along with the executable.

Add the files in the `uwp_template` folder to a ZIP. Rename the resulting Zip according to the target/architecture of the template:

```
uwp_x86_debug.zip  
uwp_x86_release.zip  
uwp_x64_debug.zip  
uwp_x64_release.zip  
uwp_arm_debug.zip  
uwp_arm_release.zip
```

Move those templates to the `templates` folder in Godot settings path. If you don't want to replace the templates, you can set the "Custom Package" property in the export window.

22.8.4 Running UWP apps with Visual Studio

If you want to debug the UWP port or simply run your apps without packaging and signing, you can deploy and launch them using Visual Studio. It might be the easiest way if you are testing on a device such as a Windows Phone or an Xbox One.

Within the ANGLE source folder, open `templates` and double-click the `install.bat` script file. This will install the Visual Studio project templates for ANGLE apps.

If you have not built Godot yet, open the `winrt/10/src/angle.sln` solution from the ANGLE source and build it to Release/Win32 target. You may also need to build it for ARM if you plan to run on a device. You can also use MSBuild if you're comfortable with the command line.

Create a new Windows App project using the "App for OpenGL ES (Windows Universal)" project template, which can be found under the `Visual C++/Windows/Universal` category.

This is a base project with the ANGLE dependencies already set up. However, by default it picks the debug version of the DLLs which usually have a very poor performance. So in the "Binaries" filter, click in each of the DLLs there and in the "Properties" window and change the relative path from `Debug_Win32` to `Release_Win32` (or `Release_ARM` for devices).

In the same "Binaries" filter, select "Add > Existing Item" and point to the Godot executable for UWP you have. In the "Properties" window, set "Content" to `True` so it's included in the project.

Right-click the `Package.appxmanifest` file and select “Open With... > XML (Text) Editor”. In the `Package/Applications/Application` element, replace the `Executable` attribute from `$targetnametoken$.exe` to `godot.uwp.exe` (or whatever your Godot executable is called). Also change the `EntryPoint` attribute to `GodotUWP.App`. This will ensure that the Godot executable is correctly called when the app starts.

Create a folder (*not* a filter) called `game` in your Visual Studio project folder and there you can put either a `data.pck` file or your Godot project files. After that, make sure to include it all with the “Add > Existing Item” command and set their “Content” property to `True` so they’re copied to the app.

To ease the workflow, you can open the “Solution Properties” and in the “Configuration” section untick the “Build” option for the app. You still have to build it at least once to generate some needed files, you can do so by right-clicking the project (*not* the solution) in the “Solution Explorer” and selecting “Build”.

Now you can just run the project and your app should open. You can use also the “Start Without Debugging” from the “Debug” menu (Ctrl+F5) to make it launch faster.

22.9 Compiling for the Web

22.9.1 Requirements

To compile export templates for the Web, the following is required:

- Emscripten 1.37.9+: If the version available per package manager is not recent enough, the best alternative is to install using the [Emscripten SDK](#) (Install in a path without spaces, i.e. not in `Program Files`)
- Python 2.7+ or Python 3.5+
- SCons build system

22.9.2 Building export templates

Start a terminal and set the environment variable `EMSCRIPTEN_ROOT` to the installation directory of Emscripten:

If you installed Emscripten via the Emscripten SDK, declare the variable with a path to the downloaded folder:

```
export EMSCRIPTEN_ROOT=~/emsdk/emscripten/master
```

If you installed Emscripten via package manager, the path can be retrieved with the `em-config` command:

```
export EMSCRIPTEN_ROOT=`em-config EMSCRIPTEN_ROOT`
```

On Windows you can set the environment variable in the system settings or in the command prompt:

```
set EMSCRIPTEN_ROOT="C:\emsdk\emscripten\master"
```

Now go to the root directory of the engine source code and instruct SCons to build the JavaScript platform. Specify `target` as either `release` for a release build or `release_debug` for a debug build:

```
scons platform=javascript tools=no target=release
scons platform=javascript tools=no target=release_debug
```

The engine will now be compiled to WebAssembly by Emscripten. If all goes well, the resulting file will be placed in the `bin` subdirectory. Its name is `godot.javascript.opt.zip` for release or `godot.javascript.opt.debug.zip` for debug.

Finally, rename the zip archive to `webassembly_release.zip` for the release template:

```
mv bin/godot.javascript.opt.zip bin/webassembly_release.zip
```

And webassembly_debug.zip for the debug template:

```
mv bin/godot.javascript.opt.debug.zip bin/webassembly_debug.zip
```

22.9.3 Building per asm.js translation or LLVM backend

WebAssembly can be compiled in two ways: The default is to first compile to asm.js, a highly optimizable subset of JavaScript, using Emscripten's *fastcomp* fork of LLVM. This code is then translated to WebAssembly using a tool called `asm2wasm`. Emscripten automatically takes care of both processes, we simply run SCons.

The other method uses LLVM's WebAssembly backend. This backend is not yet available in release versions of LLVM, only in development builds built with `LLVM_EXPERIMENTAL_TARGETS_TO_BUILD=WebAssembly`. Compiling with this backend outputs files in LLVM's `.s` format, which is translated into actual WebAssembly using a tool called `s2wasm`. Emscripten manages these processes as well, so we just invoke SCons.

In order to choose one of the two methods, the `LLVM_ROOT` variable in the Emscripten configuration file `~/.emsdk` is set. If it points to a directory containing binaries of Emscripten's *fastcomp* fork of clang, `asm2wasm` is used. This is the default in a normal Emscripten installation. Otherwise, LLVM binaries built with the WebAssembly backend will be expected and `s2wasm` is used. On Windows, make sure to escape backslashes of paths within this file as double backslashes `\\"` or use Unix-style paths with a single forward slash `/`.

22.10 Compiling with Mono

22.10.1 Requirements

- Mono 5.2.0+
- MSBuild
- pkg-config

22.10.2 Environment Variables

By default, SCons will try to find Mono in the Windows Registry on Windows or via `pkg-config` on other platforms. You can specify a different installation directory by using the following environment variables for the respective `bits` option: `MONO32_PREFIX` and `MONO64_PREFIX`.

The specified directory must contain the subdirectories `bin`, `include`, and `lib`.

22.10.3 Enable Mono Module

By default, the mono module is disabled for builds. To enable it you can pass the option `module_mono_enabled=yes` to your SCons command.

22.10.4 Generate The Glue

The glue sources are the wrapper functions that will be called by the managed side. In order to generate them, first, you must build Godot with the options `tools=yes` and `mono_glue=no`:

```
scons p=<platform> tools=yes module_mono_enabled=yes mono_glue=no
```

After the build finishes, you need to run the compiled executable with the parameter `--generate-mono-glue` followed by the path to an output directory. This path must be `modules/mono/glue` in the Godot directory.

```
godot --generate-mono-glue modules/mono/glue
```

This command will tell Godot to generate the file `modules/mono/glue/mono_glue.gen.cpp`. Once this file is generated, you can build Godot for all the desired targets without the need to repeat this process.

Notes

- **Do not** build normal binaries with `mono_glue=no`. This option disables C# scripting and therefore must only be used for the temporary binary that will be used to generate the glue. Godot should print a message at startup warning you about this.
- The glue sources must be regenerated every time the ClassDB API changes. If there is an API mismatch with the generated glue, Godot will print an error at startup.

22.10.5 Example (x11)

```
# Build temporary binary
scons p=x11 tools=yes module_mono_enabled=yes mono_glue=no
# Generate glue sources
bin/godot.x11.tools.64.mono --generate-mono-glue modules/mono/glue

### Build binaries normally
# Editor
scons p=x11 target=release_debug tools=yes module_mono_enabled=yes
# Export templates
scons p=x11 target=debug tools=no module_mono_enabled=yes
scons p=x11 target=release tools=no module_mono_enabled=yes
```

If everything went well, apart from the normal output, SCons should have also built the `GodotSharpTools.dll` assembly and copied it together with the mono runtime shared library to the `bin` subdirectory.

22.11 Batch building templates

Note: This page is outdated and needs to be updated or removed.

The following is almost the same script that we use to build all the export templates that go to the website. If you want to build or roll them yourself, this might be of use.

(note: Apple stuff is missing)

```
#This script is intended to run on Linux or macOS. Cygwin might work.

# if this flag is set, build is tagged as release in the version
# echo $IS_RELEASE_BUILD

#Need to set path to Emscripten
```

```
export EMSCRIPTEN_ROOT=/home/to/emscripten

#Build templates

#remove this stuff, will be created anew
rm -rf templates
mkdir -p templates

# Windows 32 Release and Debug

scons -j 4 p=windows target=release tools=no bits=32
cp bin/godot.windows.opt.32.exe templates/windows_32_release.exe
upx templates/windows_32_release.exe
scons -j 4 p=windows target=release_debug tools=no bits=32
cp bin/godot.windows.opt.debug.32.exe templates/windows_32_debug.exe
upx templates/windows_32_debug.exe

# Windows 64 Release and Debug (UPX does not support it yet)

scons -j 4 p=windows target=release tools=no bits=64
cp bin/godot.windows.opt.64.exe templates/windows_64_release.exe
x86_64-w64-mingw32-strip templates/windows_64_release.exe
scons -j 4 p=windows target=release_debug tools=no bits=64
cp bin/godot.windows.opt.debug.64.exe templates/windows_64_debug.exe
x86_64-w64-mingw32-strip templates/windows_64_debug.exe

# Linux 64 Release and Debug

scons -j 4 p=x11 target=release tools=no bits=64
cp bin/godot.x11.opt.64 templates/linux_x11_64_release
upx templates/linux_x11_64_release
scons -j 4 p=x11 target=release_debug tools=no bits=64
cp bin/godot.x11.opt.debug.64 templates/linux_x11_64_debug
upx templates/linux_x11_64_debug

# Linux 32 Release and Debug

scons -j 4 p=x11 target=release tools=no bits=32
cp bin/godot.x11.opt.32 templates/linux_x11_32_release
upx templates/linux_x11_32_release
scons -j 4 p=x11 target=release_debug tools=no bits=32
cp bin/godot.x11.opt.debug.32 templates/linux_x11_32_debug
upx templates/linux_x11_32_debug

# Server for 32 and 64 bits (always in debug)
scons -j 4 p=server target=release_debug tools=no bits=64
cp bin/godot_server.server.opt.debug.64 templates/linux_server_64
upx templates/linux_server_64
scons -j 4 p=server target=release_debug tools=no bits=32
cp bin/godot_server.server.opt.debug.32 templates/linux_server_32
upx templates/linux_server_32

# Android
**IMPORTANT REPLACE THIS BY ACTUAL VALUES**

export ANDROID_HOME=/home/to/android-sdk
```

```

export ANDROID_NDK_ROOT=/home/to/android-ndk

# git does not allow empty dirs, so create those
mkdir -p platform/android/java/libs/armeabi
mkdir -p platform/android/java/libs/x86

#Android Release

scons -j 4 p=android target=release
cp bin/libgodot.android.opt.so platform/android/java/libs/armeabi/libgodot_android.so
./gradlew build
cp platform/android/java/build/outputs/apk/java-release-unsigned.apk templates/
→ android_release.apk

#Android Debug

scons -j 4 p=android target=release_debug
cp bin/libgodot.android.opt.debug.so platform/android/java/libs/armeabi/libgodot_
→ android.so
./gradlew build
cp platform/android/java/build/outputs/apk/java-release-unsigned.apk templates/
→ android_debug.apk

# EMScripten

scons -j 4 p=javascript target=release
cp bin/godot.javascript.opt.html godot.html
cp bin/godot.javascript.opt.js godot.js
cp tools/html_fs/filesystem.js .
zip javascript_release.zip godot.html godot.js filesystem.js
mv javascript_release.zip templates/

scons -j 4 p=javascript target=release_debug
cp bin/godot.javascript.opt.debug.html godot.html
cp bin/godot.javascript.opt.debug.js godot.js
cp tools/html_fs/filesystem.js .
zip javascript_debug.zip godot.html godot.js filesystem.js
mv javascript_debug.zip templates/

# BlackBerry 10 (currently disabled)

#./path/to/bbndk/bbndk-env.sh
#scons -j 4 platform/bb10/godot_bb10_opt.qnx.armle target=release
#cp platform/bb10/godot_bb10_opt.qnx.armle platform/bb10/bar

#scons -j 4 platform/bb10/godot_bb10.qnx.armle target=release_debug
#cp platform/bb10/godot_bb10.qnx.armle platform/bb10/bar
#cd platform/bb10/bar
#zip -r bb10.zip *
#mv bb10.zip ../../templates
#cd ../../..

# BUILD ON MAC

[...]

# Build release executables with editor

```

```
mkdir -p release

scons -j 4 p=server target=release_debug bits=64
cp bin/godot_server.server.opt.tools.64 release/linux_server.64
upx release/linux_server.64

scons -j 4 p=x11 target=release_debug tools=yes bits=64
cp bin/godot.x11.opt.tools.64 release/godot_x11.64
# upx release/godot_x11.64 -- fails on some linux distros

scons -j 4 p=x11 target=release_debug tools=yes bits=32
cp bin/godot.x11.opt.tools.32 release/godot_x11.32

scons -j 4 p=windows target=release_debug tools=yes bits=64
cp bin/godot.windows.opt.tools.64.exe release/godot_win64.exe
x86_64-w64-mingw32-strip release/godot_win64.exe
#upx release/godot_win64.exe

scons -j 4 p=windows target=release_debug tools=yes bits=32
cp bin/godot.windows.opt.tools.32.exe release/godot_win32.exe
x86_64-w64-mingw32-strip release/godot_win32.exe
#upx release/godot_win32.exe

[...] # mac stuff

# Update classes.xml (used to generate doc)

cp doc/base/classes.xml .
release/linux_server.64 -doctool classes.xml

cd demos
rm -f godot_demos.zip
zip -r godot_demos *
cd ..

cd tools/export/blender25
zip -r bettercollada *
mv bettercollada.zip ../../..
cd ../../..
```

22.12 Packaging Godot

Starting with 2.0, Godot has features to make it easier to package it for application repositories.

22.12.1 Default behaviour

Per default, Godot stores all settings and installed templates in a per-user directory. First Godot checks the APPDATA environment variable. If it exists, the per-user directory is the “Godot” subdirectory of \$APPDATA. If APPDATA doesn’t exist, Godot checks the HOME environment variable. The per-user directory is then the “.godot” subdir of \$HOME.

This meets common operating system standards.

22.12.2 Global template path (Unix only)

The `unix_global_settings_path` build variable is meant for Unix/Linux distro packagers who want to package export templates together with godot. It allows to put the export templates on a hardcoded path.

To use it, pass the desired path via the scons `unix_global_settings_path` build variable when building the editor. The export templates then live at the “templates” subdirectory of the path specified.

Templates installed at the per-user location still override the system wide templates.

This option is only available on unix based platforms.

22.12.3 Self contained mode

The self contained mode can be used to package godot for distribution systems where godot doesn't live at a fixed location. If the godot editor finds a `.sc_` file in the directory the executable is located, godot will continue in “self contained mode”.

In self contained mode, all config files are located next to the executable in a directory called `editor_data`. Godot doesn't read or write to the per-user location anymore.

The contents of the `.sc_` file (when not empty) are read with the `ConfigFile` api (same format as `engine.cfg`, etc). So far it can contain a list of pre-loaded project in this format:

```
[init_projects]
list=[ "demos/2d/platformer", "demos/2d/isometric" ]
```

The paths are relative to the executable location, and will be added to the file `editor_settings.xml` when this is created for the first time.

CHAPTER 23

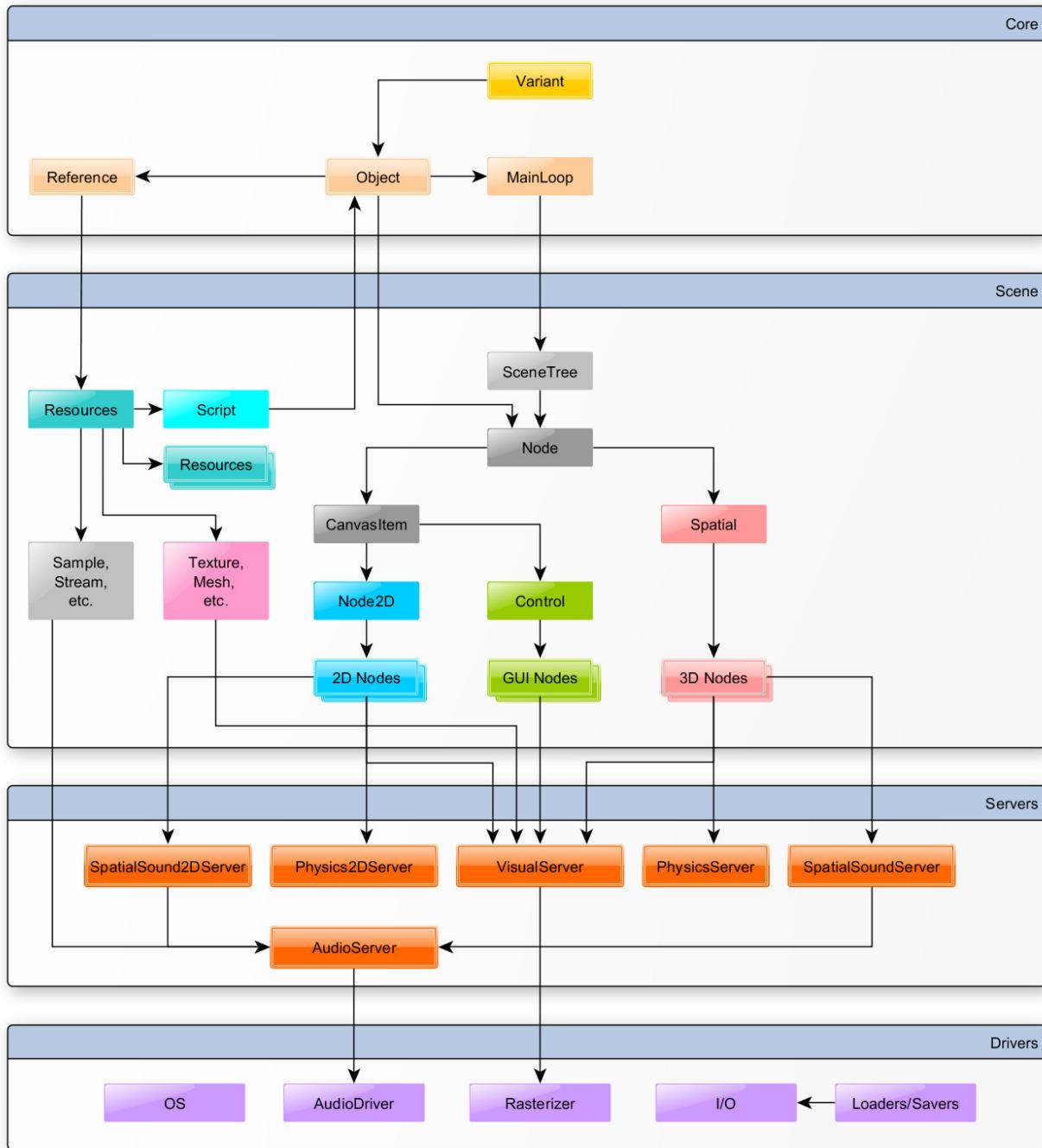
Engine development

23.1 Introduction to Godot development

This page is meant to introduce the global organization of Godot Engine's source code, and give useful tips for extending/fixing the engine on the C++ side.

23.1.1 Architecture diagram

The following diagram describes the architecture used by Godot, from the core components down to the abstracted drivers, via the scene structure and the servers.



23.1.2 Debugging the editor with gdb

If you are writing or correcting bugs affecting Godot Engine's editor, remember that the binary will by default run the project manager first, and then only run the editor in another process once you've selected a project. To launch a project directly, you need to run the editor by passing the `-e` argument to Godot Engine's binary from within your project's folder. Typically:

```
$ cd ~/myproject
$ gdb godot
```

```
> run -e
```

Or:

```
$ gdb godot
> run -e -path ~/myproject
```

23.2 Configuring an IDE

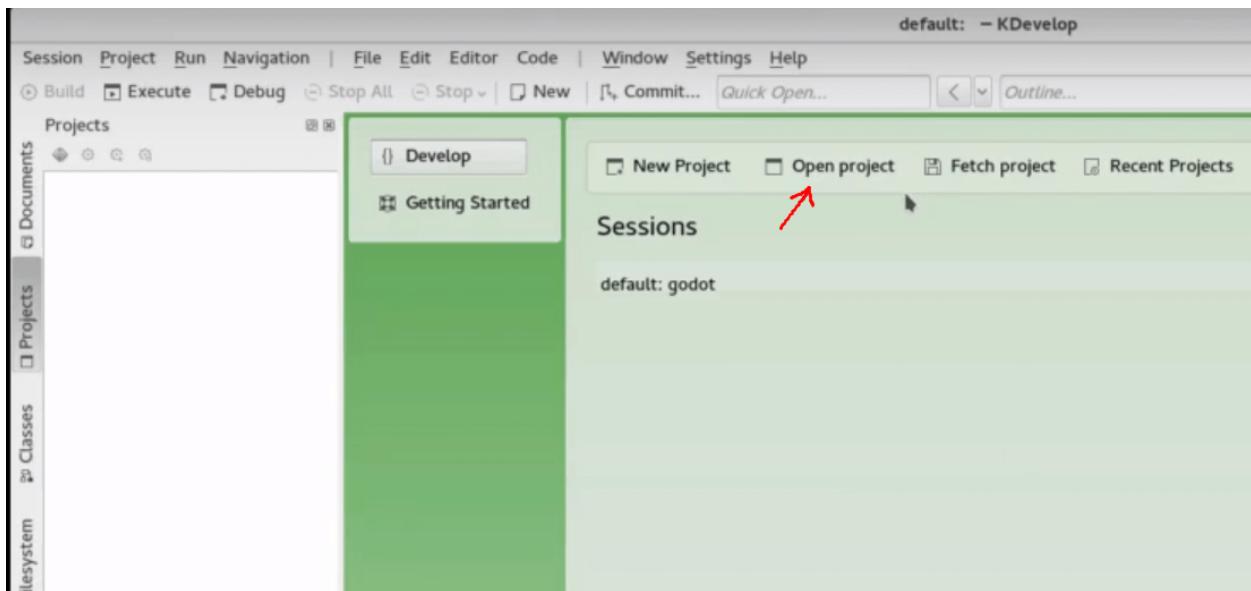
We assume that you already [cloned](#) and [compiled](#) Godot.

23.2.1 Kdevelop

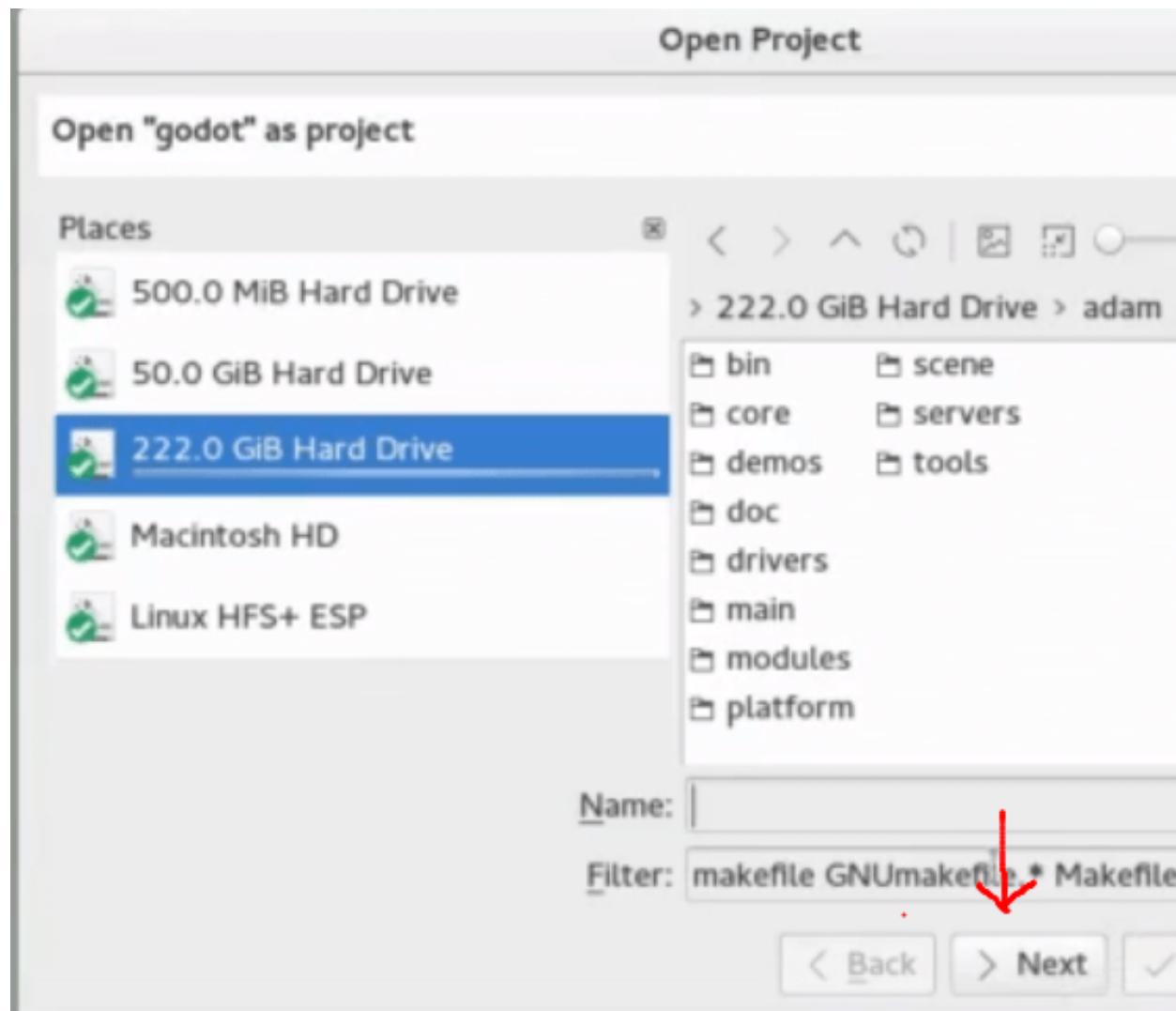
It is a free, open source IDE (Integrated Development Environment) for Linux, Solaris, FreeBSD, Mac OS X and other Unix flavors.

You can find a video tutorial [here](#). Or you may follow this text version tutorial.

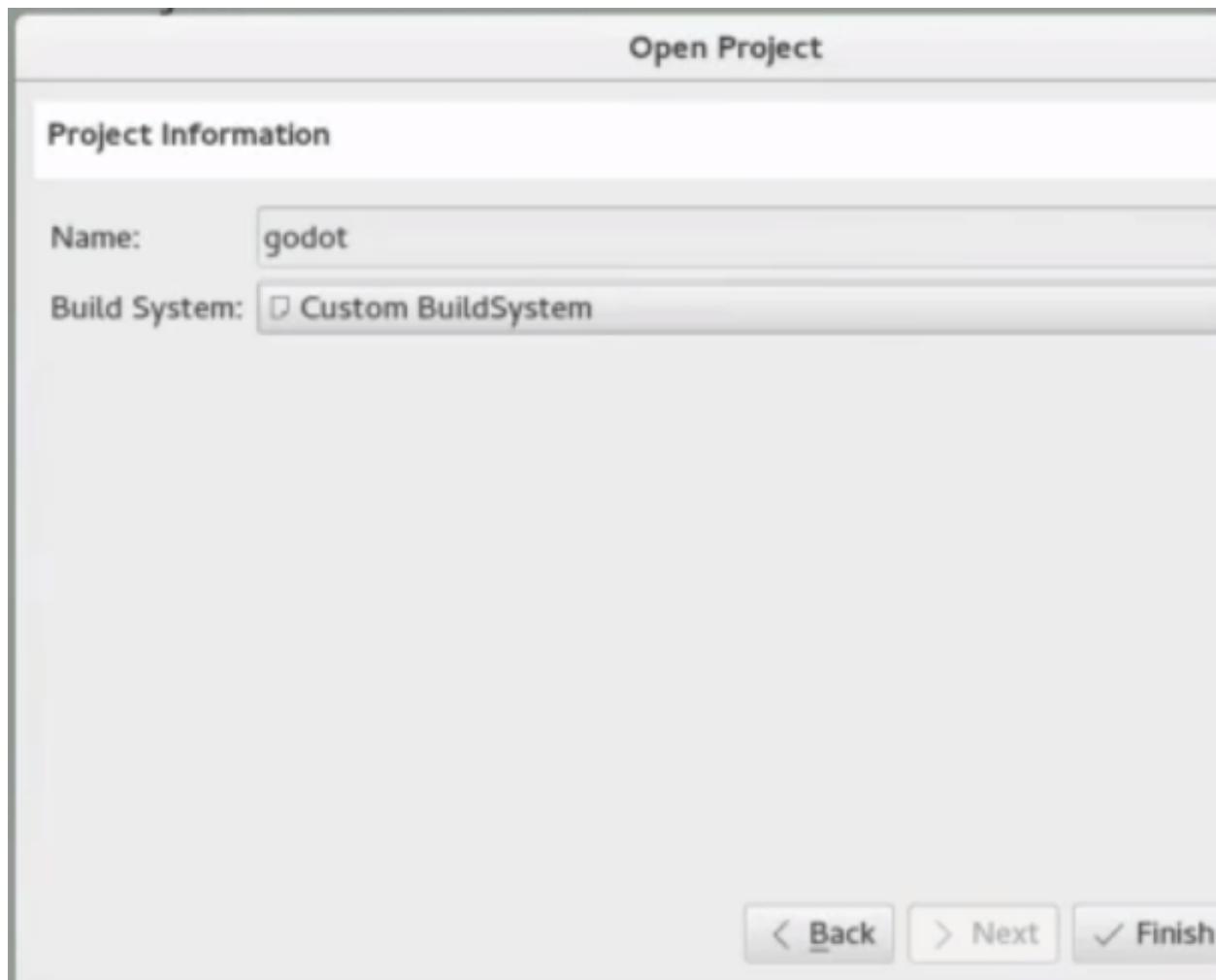
Start by opening Kdevelop and choosing “open project”.



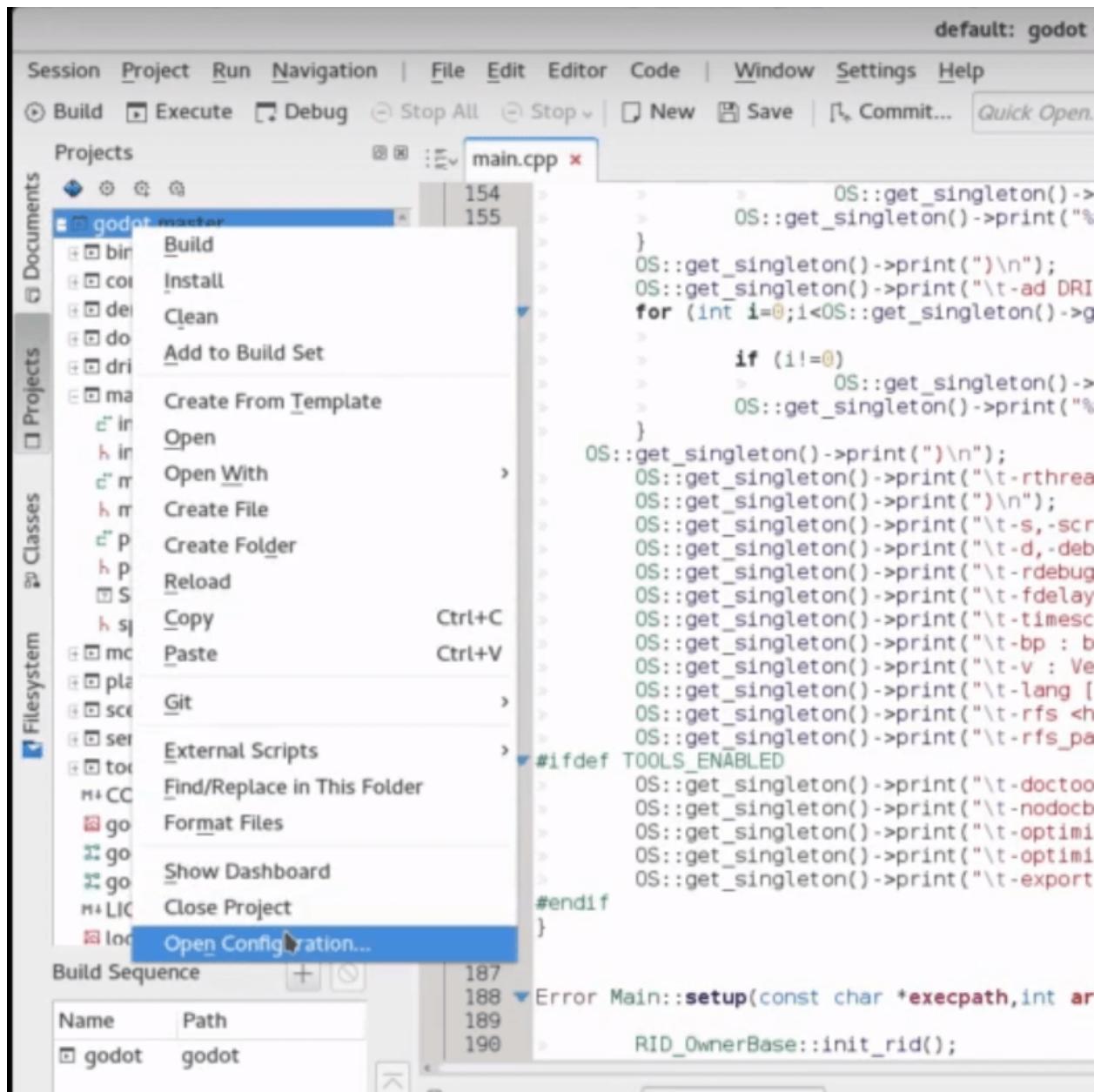
Choose the directory where you cloned Godot.



For the build system, choose “custom build system”.



Now that the project has been imported, open the project configuration.

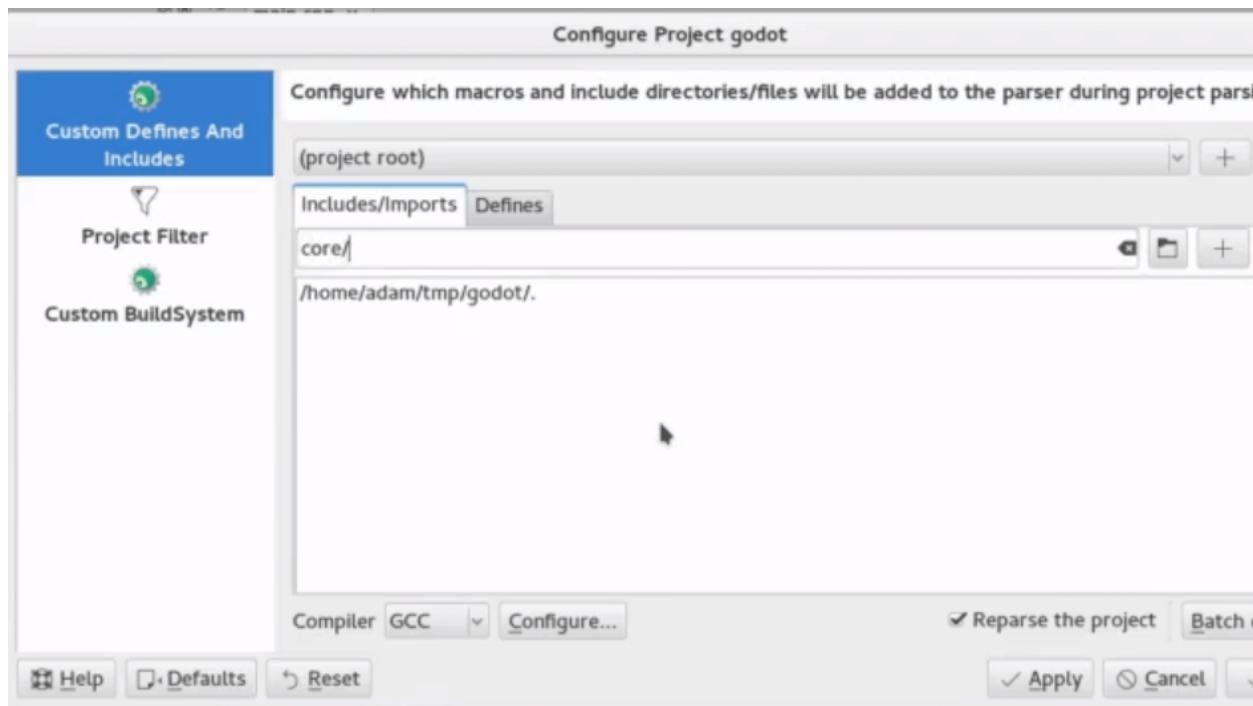


Add the following includes/imports:

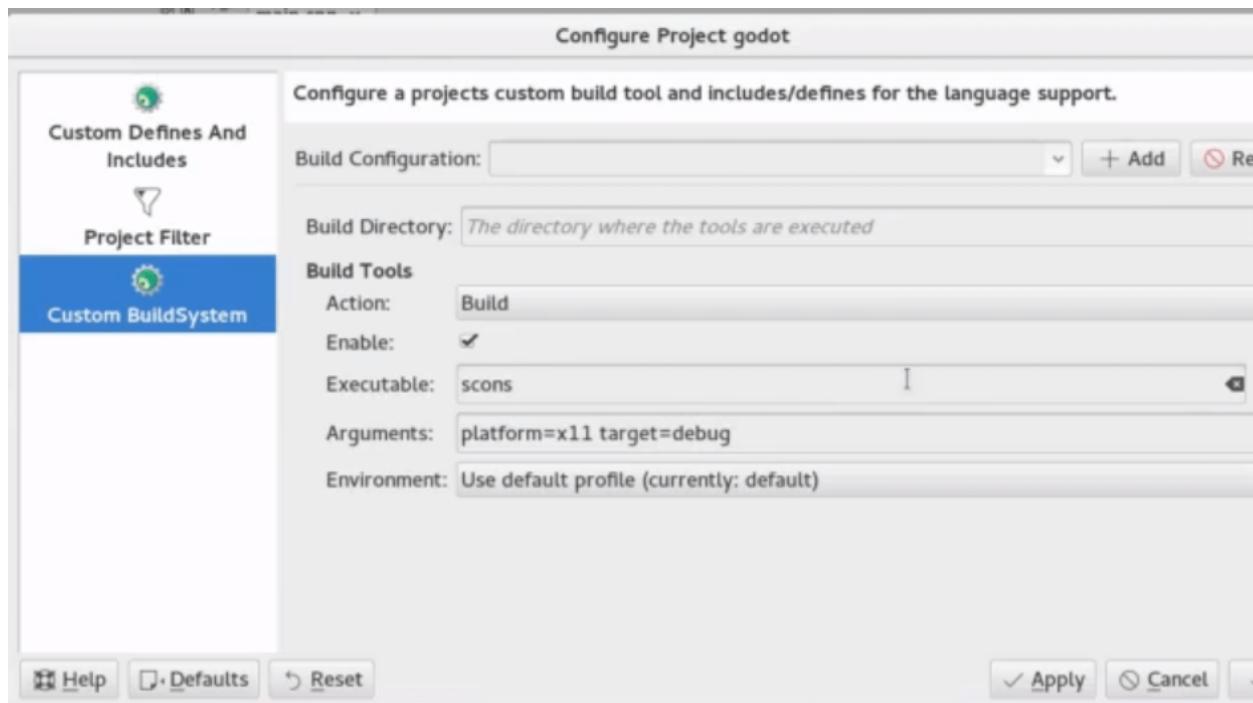
```

. // a dot to indicate the root of the Godot project
core/
core/os/
core/math/
tools/
drivers/
platform/x11/ // make that platform/osx/ if you're using OS X

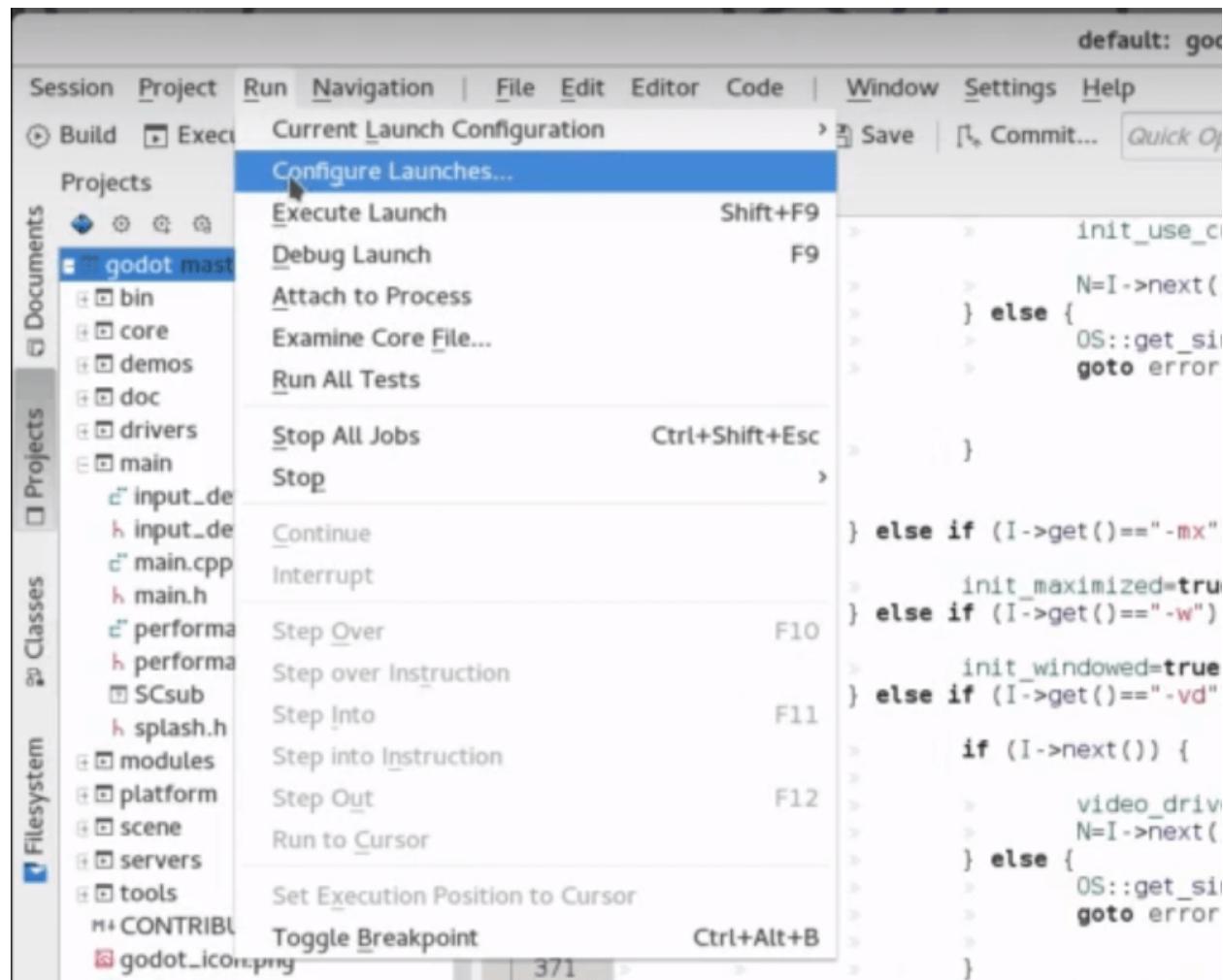
```



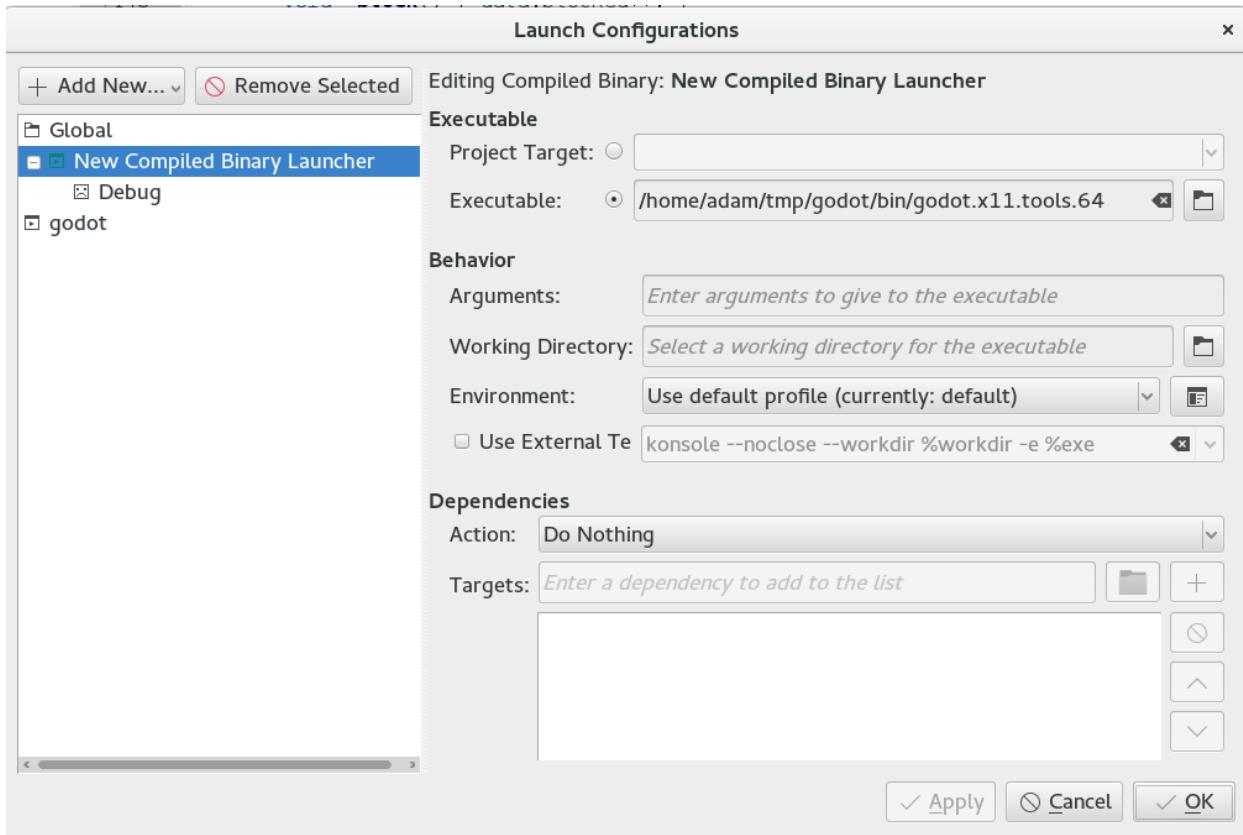
Apply the changes then switch to the “Custom Buildsystem” tab. Leave the build directory blank. Enable build tools and add `scons` as the executable and add `platform=x11 target=debug` (`platform=osx` if you’re on OS X).



Next we need to tell KDevelop where to find the binary. From the “run” menu, choose “Configure Launches”.



Click “Add new” if no launcher exists. Then add the path to your executable in the executable section. Your executable should be located in the `bin/` sub-directory and should be named something like `godot.x11.tools.64` (the name could be different depending on your platform and depending on your build options).



That's it! Now you should be good to go :)

23.2.2 Eclipse

TODO.

23.2.3 QtCreator

Importing the project

- Choose *New Project -> Import Project -> Import Existing Project*.
- Set the path to your Godot root directory and enter the project name.
- Here you can choose which folders and files will be visible to the project. C/C++ files are added automatically. Potentially useful additions: *.py for buildsystem files, *.java for Android development, *.mm for macOS. Click “Next”.
- Click *Finish*.
- Add a line containing `. to project_name.includes` to get working code completion.

Build and run

Build configuration:

- Click on *Projects* and open the *Build* tab.

- Delete the pre-defined `make` build step.
- Click *Add Build Step -> Custom Process Step*.
- Type `scons` in the *Command* field.
- Fill the *Arguments* field with your compilation options. (e.g.: `p=x11 target=debug -j 4`)

Run configuration:

- Open the *Run* tab.
- Point the *Executable* to your compiled Godot binary (e.g: `%{buildDir}\bin\godot.windows.tools.64.exe`)
- If you want to run a specific game or project, point *Working directory* to the game directory.
- If you want to run the editor, add `-e` to the *Command line arguments* field.

23.2.4 Xcode

Project Setup

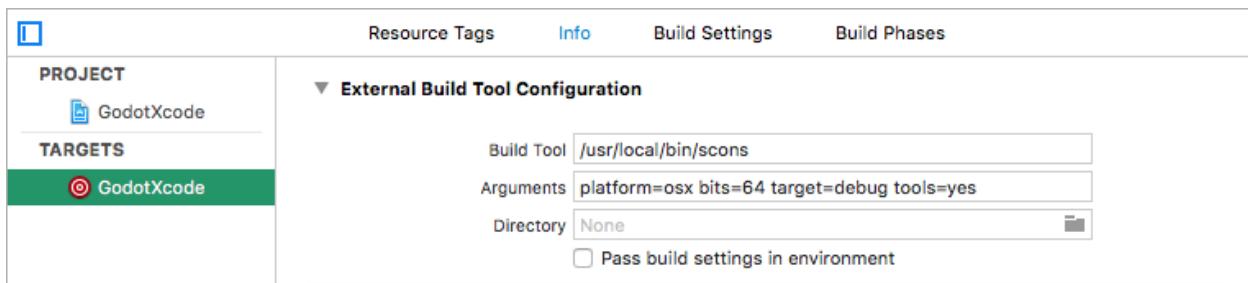
- Create an Xcode external build project anywhere

iOS						(highlighted)
watchOS						
tvOS						
OS X						
Other						
	External Build System					
	This template builds using an external build system.					

- Set the *Build tool* to the path to `scons`

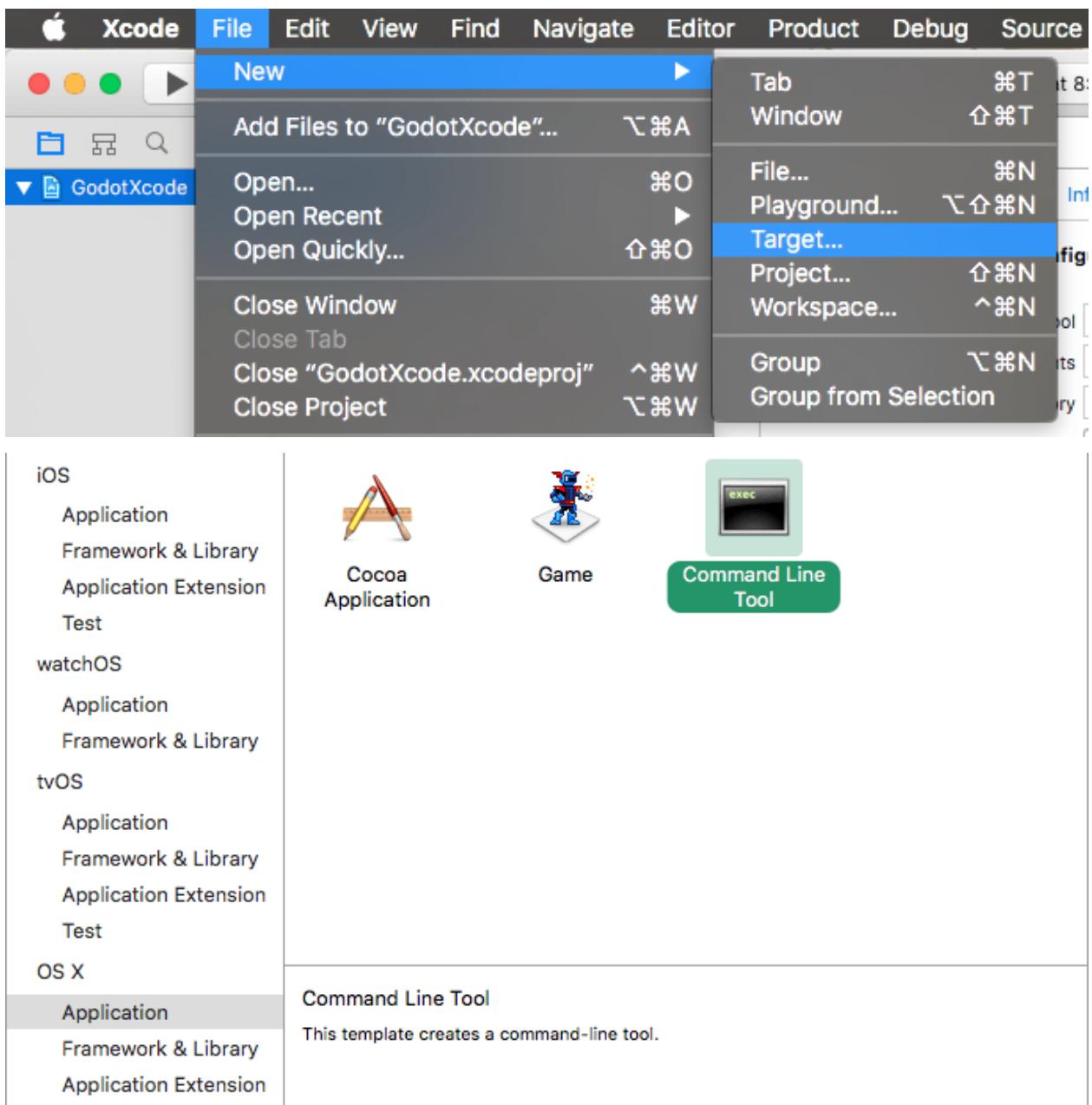
Modify Build Target's Xcode Info Tab:

- Set *Arguments* to something like: `platform=osx tools=yes bits=64 target=debug`
- Set *Directory* to the path to Godot's source folder. Keep it blank if project is already there.
- You may uncheck *Pass build settings in environment*



Add a Command Line Target:

- Go to Xcode File > New > Target... and add a new Xcode command line target



- Name it something so you know not to compile with this target

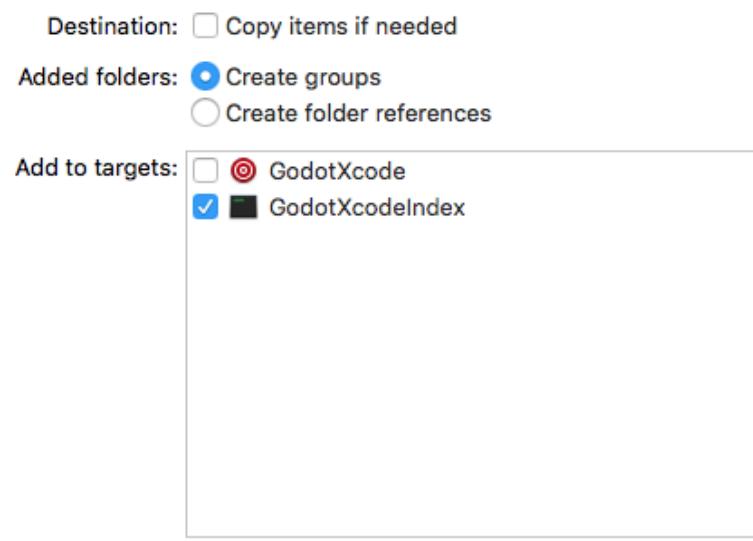
- e.g. GodotXcodeIndex
- Goto the newly created target's *Build Settings* tab and search for *Header Search Paths*
- Set *Header Search Paths* to an absolute path to Godot's source folder
- Make it recursive by adding two *'s to the end of the path
- e.g. /Users/me/repos/godot-source/**

Add Godot Source to the Project:

- Drag and drop godot source into project file browser.
- Uncheck *Create External Build System*



- Click Next
- Select *create groups*



- Check off only your command line target in the *Add to targets* section
- Click finish. Xcode will now index the files.
- Grab a cup of coffee... Maybe make something to eat, too
- You should have jump to definition, auto completion, and full syntax highlighting when it is done.

Scheme Setup

Edit Build Scheme of External Build Target:

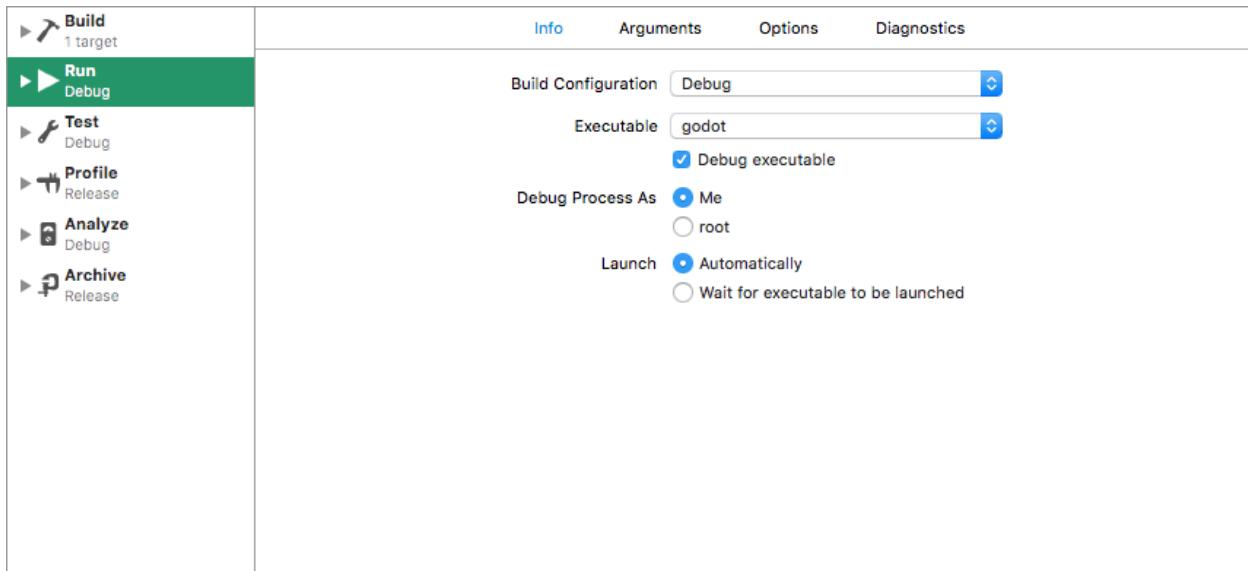
- Open scheme editor of external build target
- Expand the *Build* menu
- Goto *Post Actions*
- Add a new script run action, select your project in *Provide build settings from* as this allows you to use `${PROJECT_DIR}` variable.



- Write a script that gives the binary a name that Xcode will recognize
- e.g. `ln -f ${PROJECT_DIR}/godot/bin/godot.osx.tools.64 ${PROJECT_DIR}/godot/bin/godot`
- Build the external build target

Edit Run Scheme of External Build Target:

- Open the scheme editor again
- Click Run



- Set the *Executable* to the file you linked in your post build action script
- Check *Debug executable* if it isn't already
- You can go to *Arguments* tab and add an -e and a -path to a project to debug the editor not the project selection screen

Test it:

- Set a breakpoint in platform/osx/godot_main_osx.mm
- It should break at the point!

23.2.5 Other editors (vim, emacs, Atom...)

TODO.

23.3 Core types

Godot has a rich set of classes and templates that compose its core, and everything is built upon them.

This reference will try to list them in order for their better understanding.

23.3.1 Definitions

Godot uses the standard C98 datatypes, such as `uint8_t`, `uint32_t`, `int64_t`, etc. which are nowadays supported by every compiler. Reinventing the wheel for those is not fun, as it makes code more difficult to read.

In general, care is not taken to use the most efficient datatype for a given task unless using large structures or arrays. `int` is used through most of the code unless necessary. This is done because nowadays every device has at least a 32 bits bus and can do such operations in one cycle. It makes code more readable too.

For files or memory sizes, `size_t` is used, which is warranted to be 64 bits.

For Unicode characters, `CharType` instead of `wchar_t` is used, because many architectures have 4 bytes long `wchar_t`, where 2 bytes might be desired. However, by default, this has not been forced and `CharType` maps directly to `wchar_t`.

References:

- core/typedefs.h

23.3.2 Memory model

PC is a wonderful architecture. Computers often have gigabytes of RAM, terabytes of storage and gigahertz of CPU, and when an application needs more resources the OS will just swap out the inactive ones. Other architectures (like mobile or consoles) are in general more limited.

The most common memory model is the heap, where an application will request a region of memory, and the underlying OS will try to fit it somewhere and return it. This often works best and is very flexible, but over time and with abuse, this can lead to segmentation.

Segmentation slowly creates holes that are too small for most common allocations, so that memory is wasted. There is a lot of literature about heap and segmentation, so this topic will not be developed further here. Modern operating systems use paged memory, which helps mitigate the problem of segmentation but doesn't solve it.

However, in many studies and tests, it is shown that given enough memory, if the maximum allocation size is below a given threshold in proportion to the maximum heap size and proportion of memory intended to be unused, segmentation will not be a problem over time as it will remain constant. In other words, just leave 10-20% of your memory free and perform all small allocations and you are fine.

Godot ensures that all objects that can be allocated dynamically are small (less than a few kb at most). But what happens if an allocation is too large (like an image or mesh geometry or large array)? In this case Godot has the option to use a dynamic memory pool. This memory needs to be locked to be accessed, and if an allocation runs out of memory, the pool will be rearranged and compacted on demand. Depending on the need of the game, the programmer can configure the dynamic memory pool size.

23.3.3 Allocating memory

Godot has many tools for tracking memory usage in a game, especially during debug. Because of this, the regular C and C++ library calls should not be used. Instead, a few other ones are provided.

For C-style allocation, Godot provides a few macros:

```
memalloc()
memrealloc()
memfree()
```

These are equivalent to the usual malloc, realloc, free of the standard library.

For C++-style allocation, special macros are provided:

```
memnew( Class / Class(args) )
memdelete( instance )

memnew_arr( Class , amount )
memdelete_arr( pointer to array )
```

which are equivalent to new, delete, new[] and delete[].

memnew/memdelete also use a little C++ magic and notify Objects right after they are created, and right before they are deleted.

For dynamic memory, the DVector<> template is provided. Just use it like:

```
DVector<int>
```

DVector is just a standard vector class, it can be accessed using the [] operator, but that's probably slow for large amount of accesses (as it has to lock internally). A few helpers exist for this:

```
DVector<int>::Read r = dvector.read()
int someint = r[4]
```

and

```
DVector<int>::Write w = dvector.write()
w[4]=22;
```

respectively. These allow fast read/write from DVectors and keep it locked until they go out of scope.

References:

- [core/os/memory.h](#)
- [core/dvector.h](#)

23.3.4 Containers

Godot provides also a set of common containers:

- Vector
- List
- Set
- Map

They are very simple and aim to be as minimal as possible, as templates in C++ are often inlined and make the binary size much fatter, both in debug symbols and code. List, Set and Map can be iterated using pointers, like this:

```
for(List<int>::Element *E=somelist.front();E;E=E->next()) {
    print_line(E->get()); //print the element
}
```

The Vector<> class also has a few nice features:

- It does copy on write, so making copies of it is cheap as long as they are not modified.
- It supports multi-threading, by using atomic operations on the reference counter.

References:

- [core/vector.h](#)
- [core/list.h](#)
- [core/set.h](#)
- [core/map.h](#)

23.3.5 String

Godot also provides a String class. This class has a huge amount of features, full Unicode support in all the functions (like case operations) and utf8 parsing/extracting, as well as helpers for conversion and visualization.

References:

- [core/ustring.h](#)

23.3.6 StringName

StringNames are like a String, but they are unique. Creating a StringName from a string results in a unique internal pointer for all equal strings. StringNames are really useful for using strings as identifier, as comparing them is basically comparing a pointer.

Creation of a StringName (especially a new one) is slow, but comparison is fast.

References:

- [core/string_db.h](#)

23.3.7 Math types

There are several linear math types available in the core/math directory, they are basically just that.

References:

- [core/math](#)

23.3.8 NodePath

This is a special datatype used for storing paths in a scene tree and referencing them fast.

References:

- [core/path_db.h](#)

23.3.9 RID

RIDs are resource IDs. Servers use these to reference data stored in them. RIDs are opaque, meaning that the data they reference can't be accessed directly. RIDs are unique, even for different types of referenced data.

References:

- [core/rid.h](#)

23.4 Variant class

23.4.1 About

Variant is the most important datatype of Godot, it's the most important class in the engine. A Variant takes up only 20 bytes and can store almost any engine datatype inside of it. Variants are rarely used to hold information for long periods of time, instead they are used mainly for communication, editing, serialization and generally moving data around.

A Variant can:

- Store almost any datatype
- Perform operations between many variants (GDScript uses Variant as its atomic/native datatype).
- Be hashed, so it can be compared quickly to other variants
- Be used to convert safely between datatypes
- Be used to abstract calling methods and their arguments (Godot exports all its functions through variants)
- Be used to defer calls or move data between threads.
- Be serialized as binary and stored to disk, or transferred via network.
- Be serialized to text and use it for printing values and editable settings.
- Work as an exported property, so the editor can edit it universally.
- Be used for dictionaries, arrays, parsers, etc.

Basically, thanks to the Variant class, writing Godot itself was a much, much easier task, as it allows for highly dynamic constructs not common of C++ with little effort. Become a friend of Variant today.

References:

- [core/variant.h](#)

23.4.2 Containers: Dictionary and Array

Both are implemented using variants. A Dictionary can match any datatype used as key to any other datatype. An Array just holds an array of Variants. Of course, a Variant can also hold a Dictionary and an Array inside, making it even more flexible.

Modifications to a container will modify all references to it. A Mutex should be created to lock it if multi threaded access is desired.

Copy-on-write (COW) mode support for containers was dropped with Godot 3.0.

References:

- [core/dictionary.h](#)
- [core/array.h](#)

23.5 Object class

23.5.1 General definition

Object is the base class for almost everything. Most classes in Godot inherit directly or indirectly from it. Objects provide reflection and editable properties, and declaring them is a matter of using a single macro like this.

```
class CustomObject : public Object {

    GDCLASS(CustomObject, Object); // this is required to inherit
};
```

This makes Objects gain a lot of functionality, like for example

```
obj = memnew(CustomObject);
print_line("Object class: ", obj->get_class()); // print object class

obj2 = obj->cast_to<OtherClass>(); // converting between classes, this also works
// without RTTI enabled.
```

References:

- core/object.h

23.5.2 Registering an Object

ClassDB is a static class that holds the entire list of registered classes that inherit from Object, as well as dynamic bindings to all their methods properties and integer constants.

Classes are registered by calling:

```
ClassDB::register_class<MyCustomClass>()
```

Registering it will allow the class to be instanced by scripts, code, or creating them again when deserializing.

Registering as virtual is the same but it can't be instanced.

```
ClassDB::register_virtual_class<MyCustomClass>()
```

Object-derived classes can override the static function `static void _bind_methods()`. When one class is registered, this static function is called to register all the object methods, properties, constants, etc. It's only called once. If an Object derived class is instanced but has not been registered, it will be registered as virtual automatically.

Inside `_bind_methods`, there are a couple of things that can be done. Registering functions is one:

```
ClassDB::register_method(D_METHOD("methodname", "arg1name", "arg2name"), &
// MyCustomMethod);
```

Default values for arguments can be passed in reverse order:

```
ClassDB::register_method(D_METHOD("methodname", "arg1name", "arg2name"), &
// MyCustomType::method, DEFVAL(-1)); // default value for arg2name
```

D_METHOD is a macro that converts “methodname” to a StringName for more efficiency. Argument names are used for introspection, but when compiling on release, the macro ignores them, so the strings are unused and optimized away.

Check _bind_methods of Control or Object for more examples.

If just adding modules and functionality that is not expected to be documented as thoroughly, the D_METHOD () macro can safely be ignored and a string passing the name can be passed for brevity.

References:

- core/class_db.h

23.5.3 Constants

Classes often have enums such as:

```
enum SomeMode {  
    MODE_FIRST,  
    MODE_SECOND  
};
```

For these to work when binding to methods, the enum must be declared convertible to int, for this a macro is provided:

```
VARIANT_ENUM_CAST(MyClass::SomeMode); // now functions that take SomeMode can be  
// bound.
```

The constants can also be bound inside _bind_methods, by using:

```
BIND_CONSTANT(MODE_FIRST);  
BIND_CONSTANT(MODE_SECOND);
```

23.5.4 Properties (set/get)

Objects export properties, properties are useful for the following:

- Serializing and deserializing the object.
- Creating a list of editable values for the Object derived class.

Properties are usually defined by the PropertyInfo() class. Usually constructed as:

```
PropertyInfo(type, name, hint, hint_string, usage_flags)
```

For example:

```
PropertyInfo(Variant::INT, "amount", PROPERTY_HINT_RANGE, "0,49,1", PROPERTY_USAGE_  
EDITOR)
```

This is an integer property, named “amount”, hint is a range, range goes from 0 to 49 in steps of 1 (integers). It is only usable for the editor (edit value visually) but won’t be serialized.

Another example:

```
PropertyInfo(Variant::STRING, "modes", PROPERTY_HINT_ENUM, "Enabled,Disabled,Turbo")
```

This is a string property, can take any string but the editor will only allow the defined hint ones. Since no usage flags were specified, the default ones are PROPERTY_USAGE_STORAGE and PROPERTY_USAGE_EDITOR.

There are plenty of hints and usage flags available in object.h, give them a check.

Properties can also work like C# properties and be accessed from script using indexing, but this usage is generally discouraged, as using functions is preferred for legibility. Many properties are also bound with categories, such as “animation/frame” which also make indexing impossible unless using operator [].

From `_bind_methods()`, properties can be created and bound as long as set/get functions exist. Example:

```
ADD_PROPERTY(PropertyInfo(Variant::INT, "amount"), "set_amount", "get_amount")
```

This creates the property using the setter and the getter.

23.5.5 Binding properties using `_set/_get/_get_property_list`

An additional method of creating properties exists when more flexibility is desired (i.e. adding or removing properties on context).

The following functions can be overridden in an Object derived class, they are NOT virtual, DO NOT make them virtual, they are called for every override and the previous ones are not invalidated (multilevel call).

```
void _get_property_info(List< PropertyInfo > *r_props); // return list of properties
bool _get(const StringName &p_property, Variant &r_value) const; // return true if
//property was found
bool _set(const StringName &p_property, const Variant &p_value); // return true if
//property was found
```

This is also a little less efficient since `p_property` must be compared against the desired names in serial order.

23.5.6 Dynamic casting

Godot provides dynamic casting between Object-derived classes, for example:

```
void somefunc(Object *some_obj) {
    Button *button = some_obj->cast_to<Button>();
}
```

If cast fails, NULL is returned. This system uses RTTI, but it also works fine (although a bit slower) when RTTI is disabled. This is useful on platforms where a very small binary size is ideal, such as HTML5 or consoles (with low memory footprint).

23.5.7 Signals

Objects can have a set of signals defined (similar to Delegates in other languages). Connecting to them is rather easy:

```
obj->connect(<signal>, target_instance, target_method)
// for example:
obj->connect("enter_tree", this, "_node_entered_tree")
```

The method `_node_entered_tree` must be registered to the class using `ClassDB::register_method` (explained before).

Adding signals to a class is done in `_bind_methods`, using the `ADD_SIGNAL` macro, for example:

```
ADD_SIGNAL(MethodInfo("been_killed"))
```

23.5.8 References

Reference inherits from Object and holds a reference count. It is the base for reference counted object types. Declaring them must be done using Ref<> template. For example:

```
class MyReference: public Reference {
    GDCLASS(MyReference, Reference);
};

Ref<MyReference> myref(memnew(MyReference));
```

`myref` is reference counted. It will be freed when no more Ref<> templates point to it.

References:

- core/reference.h

23.5.9 Resources:

Resource inherits from Reference, so all resources are reference counted. Resources can optionally contain a path, which reference a file on disk. This can be set with `resource.set_path(path)`. This is normally done by the resource loader though. No two different resources can have the same path, attempt to do so will result in an error.

Resources without a path are fine too.

References:

- core/resource.h

23.5.10 Resource loading

Resources can be loaded with the ResourceLoader API, like this:

```
Ref<Resource> res = ResourceLoader::load("res://someresource.res")
```

If a reference to that resource has been loaded previously and is in memory, the resource loader will return that reference. This means that there can be only one resource loaded from a file referenced on disk at the same time.

- resourceinteractiveloader (TODO)

References:

- core/io/resource_loader.h

23.5.11 Resource saving

Saving a resource can be done with the resource saver API:

```
ResourceSaver::save("res://someresource.res", instance)
```

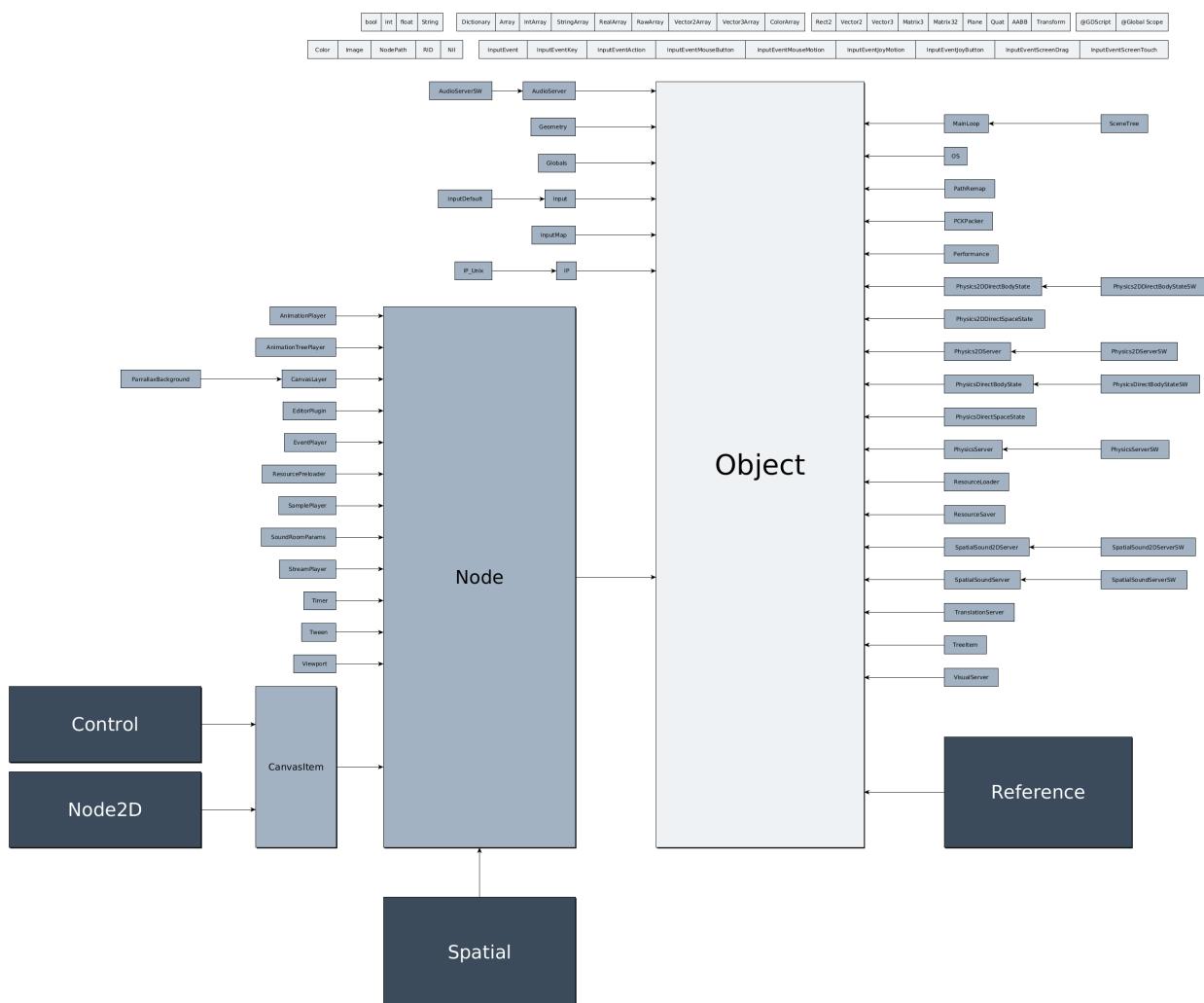
Instance will be saved. Sub resources that have a path to a file will be saved as a reference to that resource. Sub resources without a path will be bundled with the saved resource and assigned sub-IDs, like “`res://someresource.res::1`”. This also helps to cache them when loaded.

References:

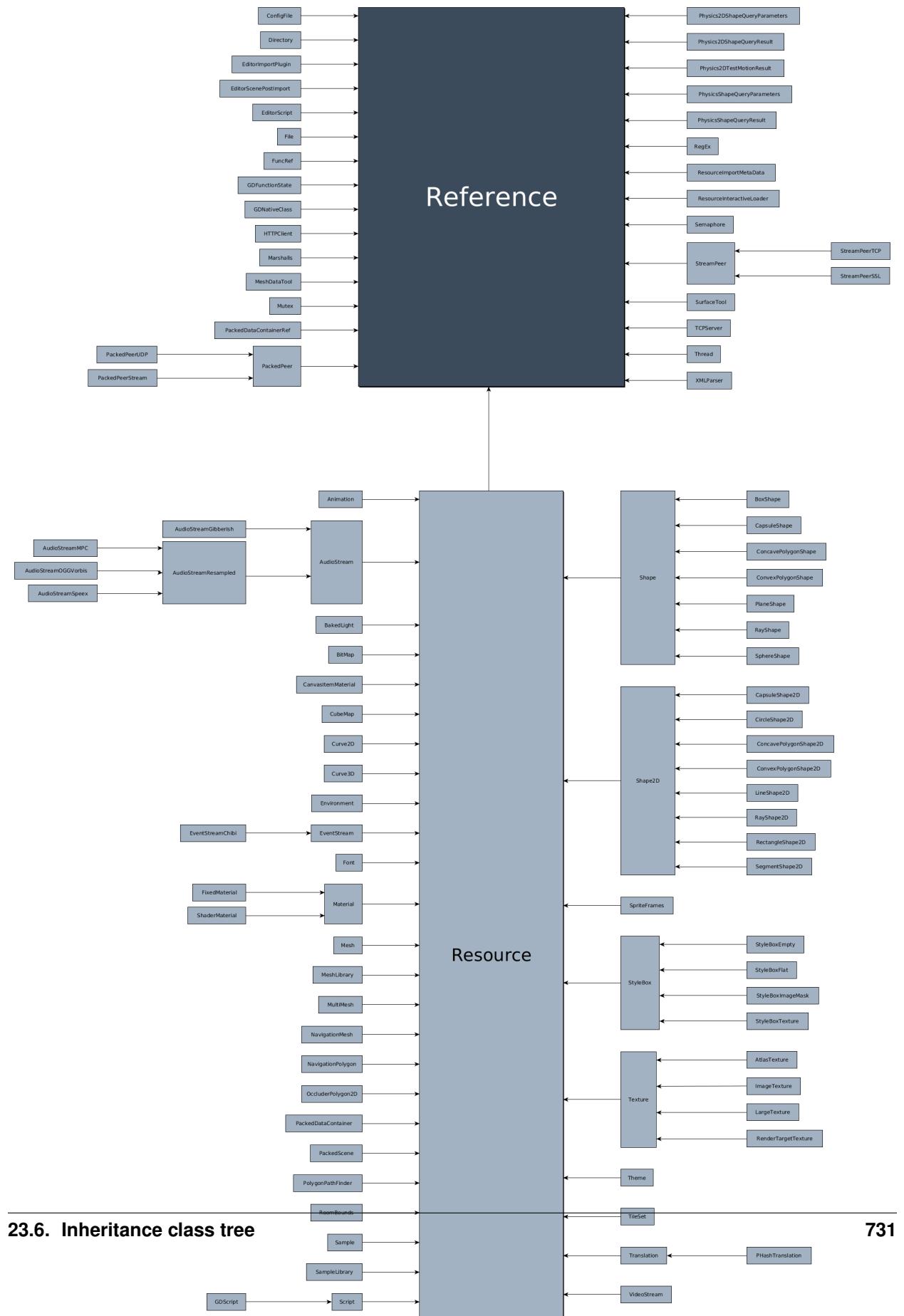
- core/io/resource_saver.h

23.6 Inheritance class tree

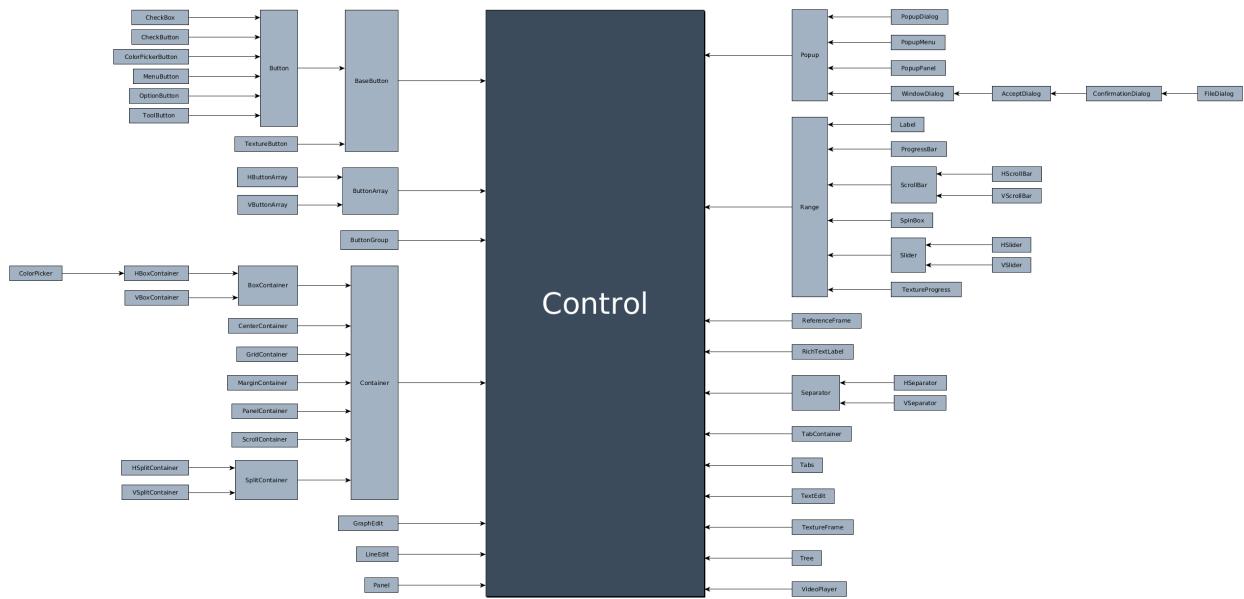
23.6.1 Object



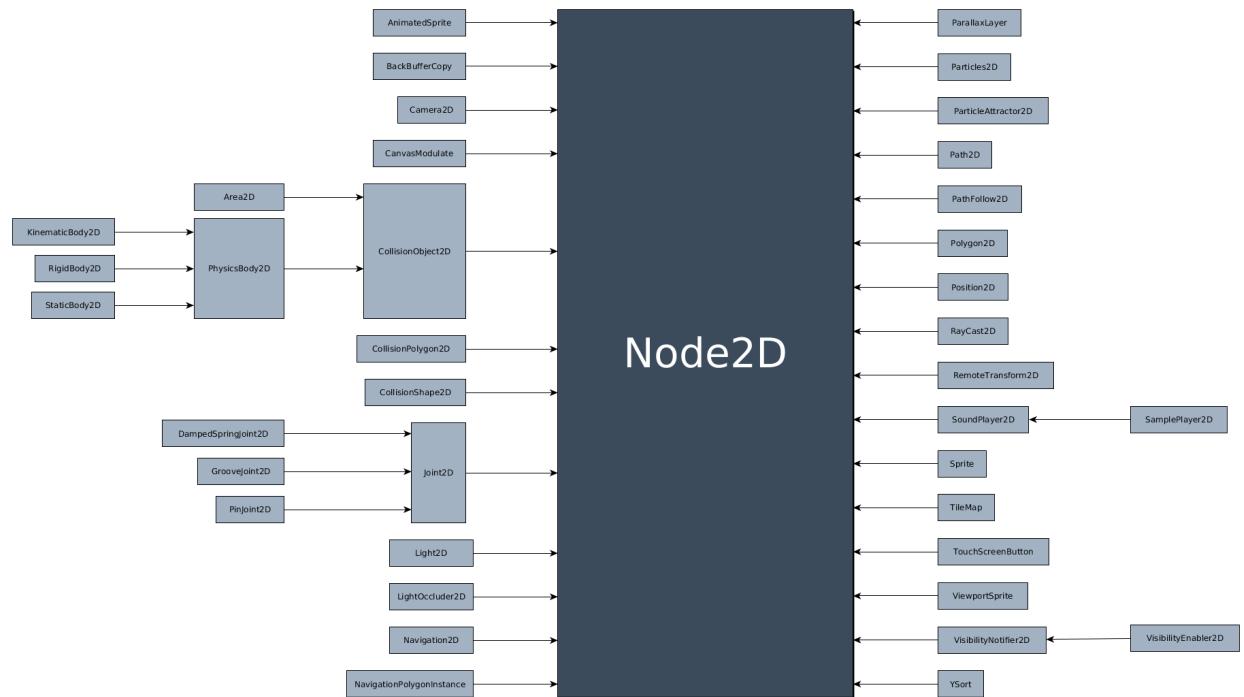
23.6.2 Reference



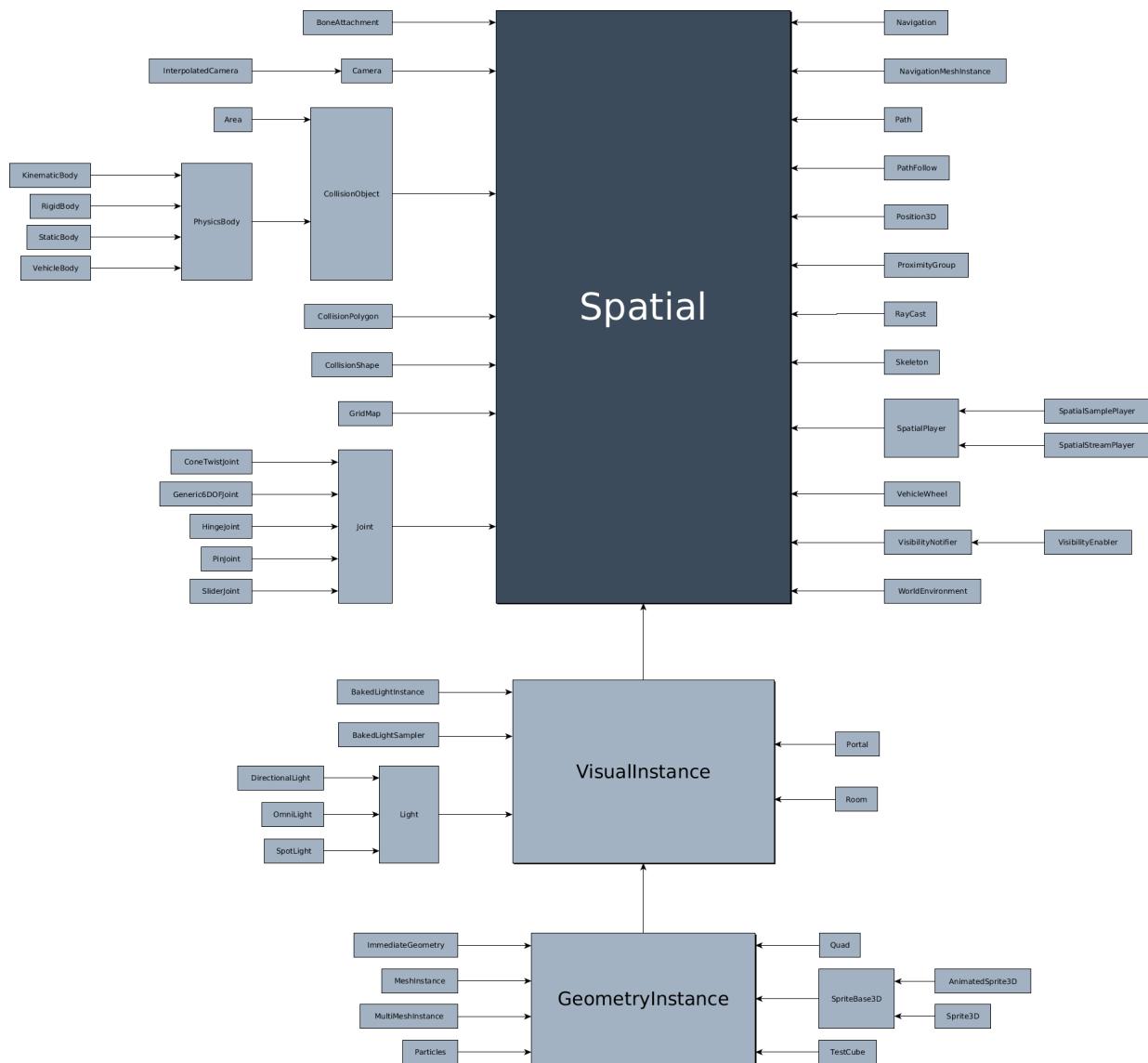
23.6.3 Control



23.6.4 Node2D



23.6.5 Spatial



Source files: `class_tree.zip`.

23.7 Custom modules in C++

23.7.1 Modules

Godot allows extending the engine in a modular way. New modules can be created and then enabled/disabled. This allows for adding new engine functionality at every level without modifying the core, which can be split for use and reuse in different modules.

Modules are located in the `modules/` subdirectory of the build system. By default, many different modules exist, such as GDScript (which, yes, is not part of the base engine), the Mono runtime, a regular expressions module, and

others. As many new modules as desired can be created and combined, and the SCons build system will take care of it transparently.

23.7.2 What for?

While it's recommended that most of a game is written in scripting (as it is an enormous time saver), it's perfectly possible to use C++ instead. Adding C++ modules can be useful in the following scenarios:

- Binding an external library to Godot (like PhysX, FMOD, etc).
- Optimize critical parts of a game.
- Adding new functionality to the engine and/or editor.
- Porting an existing game.
- Write a whole, new game in C++ because you can't live without C++.

23.7.3 Creating a new module

Before creating a module, make sure to download the source code of Godot and manage to compile it. There are tutorials in the documentation for this.

To create a new module, the first step is creating a directory inside `modules/`. If you want to maintain the module separately, you can checkout a different VCS into `modules` and use it.

The example module will be called “summator”, and is placed inside the Godot source tree (`C:\godot` refers to wherever the Godot sources are located):

```
C:\godot> cd modules
C:\godot\modules> mkdir summator
C:\godot\modules> cd summator
C:\godot\modules\summator>
```

Inside we will create a simple summator class:

```
/* summator.h */
#ifndef SUMMATOR_H
#define SUMMATOR_H

#include "reference.h"

class Summator : public Reference {
    GDCLASS(Summator, Reference);

    int count;

protected:
    static void _bind_methods();

public:
    void add(int value);
    void reset();
    int get_total() const;

    Summator();
};
```

```
#endif
```

And then the cpp file.

```
/* summator.cpp */

#include "summator.h"

void Summator::add(int value) {

    count += value;
}

void Summator::reset() {

    count = 0;
}

int Summator::get_total() const {

    return count;
}

void Summator::_bind_methods() {

    ClassDB::bind_method(D_METHOD("add", "value"), &Summator::add);
    ClassDB::bind_method(D_METHOD("reset"), &Summator::reset);
    ClassDB::bind_method(D_METHOD("get_total"), &Summator::get_total);
}

Summator::Summator() {
    count = 0;
}
```

Then, the new class needs to be registered somehow, so two more files need to be created:

```
register_types.h
register_types.cpp
```

With the following contents:

```
/* register_types.h */

void register_summator_types();
void unregister_summator_types();
/* yes, the word in the middle must be the same as the module folder name */
```

```
/* register_types.cpp */

#include "register_types.h"
#include "class_db.h"
#include "summator.h"

void register_summator_types() {

    ClassDB::register_class<Summator>();
}
```

```
void unregister_summator_types() {
    //nothing to do here
}
```

Next, we need to create a *SCsub* file so the build system compiles this module:

```
# SCsub
Import('env')

env.add_source_files(env.modules_sources, "*.cpp") # just add all cpp files to the
#build
```

With multiple sources, you can also add each file individually to a Python string list:

```
src_list = ["summator.cpp", "other.cpp", "etc.cpp"]
env.add_source_files(env.modules_sources, src_list)
```

This allows for powerful possibilities using Python to construct the file list using loops and logic statements. Look at some of the other modules that ship with Godot by default for examples.

To add include directories for the compiler to look at you can append it to the environment's paths:

```
env.Append(CPPPATH="mylib/include") # this is a relative path
env.Append(CPPPATH="#myotherlib/include") # this is an 'absolute' path
```

If you want to add custom compiler flags when building your module, you need to clone *env* first, so it won't add those flags to whole Godot build (which can cause errors). Example *SCsub* with custom flags:

```
# SCsub
Import('env')

module_env = env.Clone()
module_env.add_source_files(env.modules_sources, "*.cpp")
module_env.Append(CXXFLAGS=['-O2', '-std=c++11'])
```

And finally, the configuration file for the module, this is a simple python script that must be named *config.py*:

```
# config.py

def can_build(platform):
    return True

def configure(env):
    pass
```

The module is asked if it's ok to build for the specific platform (in this case, True means it will build for every platform).

And that's it. Hope it was not too complex! Your module should look like this:

```
godot/modules/summator/config.py
godot/modules/summator/summator.h
godot/modules/summator/summator.cpp
godot/modules/summator/register_types.h
godot/modules/summator/register_types.cpp
godot/modules/summator/SCsub
```

You can then zip it and share the module with everyone else. When building for every platform (instructions in the previous sections), your module will be included.

23.7.4 Using the module

Using your newly created module is very easy, from any script you can now do:

```
var s = Summator.new()
s.add(10)
s.add(20)
s.add(30)
print(s.get_total())
s.reset()
```

And the output will be 60.

23.7.5 Improving the build system for development

So far we defined a clean and simple SCsub that allows us to add the sources of our new module as part of the Godot binary.

This static approach is fine when we want to build a release version of our game given we want all the modules in a single binary.

However the trade-off is every single change means a full recompilation of the game. Even if SCons is able to detect and recompile only the file that have changed, finding such files and eventually linking the final binary is a long and costly part.

The solution to avoid such a cost is to build our own module as a shared library that will be dynamically loaded when starting our game's binary.

```
# SCsub
Import('env')

sources = [
    "register_types.cpp",
    "summator.cpp"
]

# First, create a custom env for the shared library.
module_env = env.Clone()
module_env.Append(CXXFLAGS='-fPIC') # Needed to compile shared library
# We don't want godot's dependencies to be injected into our shared library.
module_env['LIBS'] = []

# Now define the shared library. Note that by default it would be built
# into the module's folder, however it's better to output it into `bin`
# next to the godot binary.
shared_lib = module_env.SharedLibrary(target='#bin/summator', source=sources)

# Finally notify the main env it has our shared library as a new dependency.
# To do so, SCons wants the name of the lib with its custom suffixes
# (e.g. ".x11.tools.64") but without the final ".so".
# We pass this along with the directory of our library to the main env.
shared_lib_shim = shared_lib[0].name.rsplit('.', 1)[0]
env.Append(LIBS=[shared_lib_shim])
env.Append(LIBPATH=['#bin'])
```

Once compiled, we should end up with a bin directory containing both the godot* binary and our libsummator*.so. However given the .so is not in a standard directory (like /usr/lib), we have to help our binary find it during runtime with the LD_LIBRARY_PATH environ variable:

```
user@host:~/godot$ export LD_LIBRARY_PATH=$(pwd)/bin/
user@host:~/godot$ ./bin/godot*
```

note: Pay attention you have to `export` the environ variable otherwise you won't be able to play your project from within the editor.

On top of that, it would be nice to be able to select whether to compile our module as shared library (for development) or as a part of the godot binary (for release). To do that we can define a custom flag to be passed to SCons using the *ARGUMENT* command:

```
# SCsub
Import('env')

sources = [
    "register_types.cpp",
    "summator.cpp"
]

module_env = env.Clone()
module_env.Append(CXXFLAGS=['-O2', '-std=c++11'])

if ARGUMENTS.get('summator_shared', 'no') == 'yes':
    # Shared lib compilation
    module_env.Append(CXXFLAGS='-fPIC')
    module_env['LIBS'] = []
    shared_lib = module_env.SharedLibrary(target='#bin/summator', source=sources)
    shared_lib_shim = shared_lib[0].name.rsplit('.', 1)[0]
    env.Append(LIBS=[shared_lib_shim])
    env.Append(LIBPATH=['#bin'])
else:
    # Static compilation
    module_env.add_source_files(env.modules_sources, sources)
```

Now by default scons command will build our module as part of godot's binary and as a shared library when passing `summator_shared=yes`.

Finally you can even speedup build further by explicitly specifying your shared module as target in the scons command:

```
user@host:~/godot$ scons summator_shared=yes bin/summator.x11.tools.64.so
```

23.7.6 Writing custom documentation

Writing documentation may seem like a boring task, but it is highly recommended to document your newly created module in order to make it easier for users to benefit from it. Not to mention that the code you've written one year ago may become indistinguishable from the code that was written by someone else, so be kind to your future self!

There are several steps in order to setup custom docs for the module:

1. Make a new directory in the root of the module. The directory name can be anything, but we'll be using the `doc_classes` name throughout this section.
2. Append the following code snippet to `config.py`:

```

def get_doc_classes():
    return [
        "ClassName",
    ]

def get_doc_path():
    return "doc_classes"

```

The `get_doc_classes()` method is necessary for the build system to know which documentation classes of the module must be merged, since the module may contain several classes. Replace `ClassName` with the name of the class you want to write documentation for. If you need docs for more than one class, append those as well.

The `get_doc_path()` method is used by the build system to determine the location of the docs. In our case, they will be located in the `doc_classes` directory.

3. Run command:

```
godot --doctool <path>
```

This will dump the engine API reference to the given `<path>` in XML format. Notice that you'll need to configure your PATH to locate Godot's executable, and make sure that you have write access rights. If not, you might encounter an error similar to the following:

```

ERROR: Can't write doc file: docs/doc/classes/@GDScript.xml
At: editor/doc/doc_data.cpp:956

```

4. Get generated doc file from `godot/doc/classes/ClassName.xml`
5. Copy this file to `doc_classes`, optionally edit it, then compile the engine.

The build system will fetch the documentation files from the `doc_classes` directory and merge them with the base types. Once the compilation process is finished, the docs will become accessible within the engine's built-in documentation system.

In order to keep documentation up-to-date, all you'll have to do is simply modify one of the `ClassName.xml` files and recompile the engine from now on.

23.7.7 Summing up

As you see, it's really easy to develop Godot in C++. Just write your stuff normally and remember to:

- use `GDCLASS` macro for inheritance, so Godot can wrap it
- use `_bind_methods` to bind your functions to scripting, and to allow them to work as callbacks for signals.

But this is not all, depending what you do, you will be greeted with some surprises.

- If you inherit from `Node` (or any derived node type, such as `Sprite`), your new class will appear in the editor, in the inheritance tree in the “Add Node” dialog.
- If you inherit from `Resource`, it will appear in the resource list, and all the exposed properties can be serialized when saved/loaded.
- By this same logic, you can extend the Editor and almost any area of the engine.

23.8 Creating Android modules

23.8.1 Introduction

Making video games portable is all fine and dandy, until mobile gaming monetization shows up.

This area is complex, usually a mobile game that monetizes needs special connections to a server for stuff such as:

- Analytics
- In-app purchases
- Receipt validation
- Install tracking
- Ads
- Video ads
- Cross-promotion
- In-game soft & hard currencies
- Promo codes
- A/B testing
- Login
- Cloud saves
- Leaderboards and scores
- User support & feedback
- Posting to Facebook, Twitter, etc.
- Push notifications

Oh yeah, developing for mobile is a lot of work. On iOS, you can just write a C++ module and take advantage of the C++/ObjC intercommunication, so this is rather easy.

For C++ developers Java is a pain, the build system is severely bloated and interfacing it with C++ through JNI (Java Native Interface) is more pain than you don't want even for your worst enemy.

23.8.2 Maybe REST?

Most of these APIs allow communication via REST+JSON APIs. Godot has great support for HTTP, HTTPS and JSON, so consider this as an option that works in every platform. Only write the code once and you are set to go.

Popular engines that have half the share of apps published on mobile get special plugins written just for them. Godot does not have that luxury yet. So, if you write a REST implementation of a SDK for Godot, please share it with the community.

23.8.3 Android module

Writing an Android module is similar to *Custom modules in C++*, but needs a few more steps.

Make sure you are familiar with building your own *Android export templates*, as well as creating *Custom modules in C++*.

config.py

In the config.py for the module, some extra functions are provided for convenience. First, it's often wise to detect if android is being built and only enable building in this case:

```
def can_build(plat):
    return plat=="android"
```

If more than one platform can be built (typical if implementing the module also for iOS), check manually for Android in the configure functions:

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # android specific code
```

23.8.4 Java singleton

An android module will usually have a singleton class that will load it, this class inherits from Godot.SingletonBase. Resource identifiers for any additional resources you have provided for the module will be in the com.godot.game.R class, so you'll likely want to import it.

A singleton object template follows:

```
// package com.android.godot; // for 1.1
package org.godotengine.godot; // for 2.0

import com.godot.game.R;

public class MySingleton extends Godot.SingletonBase {

    public int myFunction(String p_str) {
        // a function to bind
    }

    static public Godot.SingletonBase initialize(Activity p_activity) {
        return new MySingleton(p_activity);
    }

    public MySingleton(Activity p_activity) {
        //register class name and functions to bind
        registerClass("MySingleton", new String[] {"myFunction"});

        // you might want to try initializing your singleton here, but android
        // threads are weird and this runs in another thread, so you usually have to
do
        activity.runOnUiThread(new Runnable() {
            public void run() {
                //useful way to get config info from engine.cfg
                String key = GodotLib.getGlobal("plugin/api_key");
                SDK.initializeHere();
            }
        });
    }
}
```

```
// forwarded callbacks you can reimplement, as SDKs often need them

protected void onActivityResult(int requestCode, int resultCode, Intent data)
{} // ←

protected void onMainPause() {}
protected void onMainResume() {}
protected void onMainDestroy() {}

protected void onGLDrawFrame(GL10 gl) {}
protected void onGLSurfaceChanged(GL10 gl, int width, int height) {} // ←
// singletons will always miss first onGLSurfaceChanged call

}
```

Calling back to Godot from Java is a little more difficult. The instance ID of the script must be known first, this is obtained by calling `get_instance_ID()` on the script. This returns an integer that can be passed to Java.

From Java, use the `calldeferred` function to communicate back with Godot. Java will most likely run in a separate thread, so calls are deferred:

```
GodotLib.calldeferred(<instanceid>, "<function>", new Object[]{param1,param2,etc});
```

Add this singleton to the build of the project by adding the following to config.py:

(Before Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_module_file("MySingleton.java")
        #env.android_module_file("MySingleton2.java") call again for more files
```

(After Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_add_java_dir("Directory that contain MySingleton.java")
```

23.8.5 AndroidManifest

Some SDKs need custom values in `AndroidManifest.xml`. Permissions can be edited from the godot exporter so there is no need to add those, but maybe other functionalities are needed.

Create the custom chunk of android manifest and put it inside the module, add it like this:

(Before Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_module_file("MySingleton.java")
        env.android_module_manifest("AndroidManifestChunk.xml")
```

(After Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_add_java_dir("Directory that contain MySingelton.java")
        env.android_add_to_manifest("AndroidManifestChunk.xml")
```

23.8.6 Resources

In order to provide additional resources with your module you have to add something like this:

```
def configure(env):
    if env['platform'] == 'android':
        # [...]
        env.android_add_res_dir("Directory that contains resource subdirectories,
        ↴(values, drawable, etc.)")
```

Now you can refer to those resources by their id (R.string.my_string, and the like) by importing the com.godot.game.R class in your Java code.

23.8.7 SDK library

So, finally it's time to add the SDK library. The library can come in two flavors, a JAR file or an Android project for ant. JAR is the easiest to integrate, just put it in the module directory and add it:

(Before Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_module_file("MySingleton.java")
        env.android_module_manifest("AndroidManifestChunk.xml")
        env.android_module_library("MyLibrary-3.1.jar")
```

(After Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"
```

```
def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_add_java_dir("Directory that contain MySingleton.java")
        env.android_add_to_manifest("AndroidManifestChunk.xml")
        env.android_add_dependency("compile files('something_local.jar')") # if you
        ↪have a jar, the path is relative to platform/android/java/gradlew, so it will start
        ↪with ../../modules/module_name/
        env.android_add_maven_repository("maven url") #add a maven url
        env.android_add_dependency("compile 'com.google.android.gms:play-services-
        ↪ads:8'") #get dependency from maven repository
```

23.8.8 SDK project

When this is an Android project, things usually get more complex. Copy the project folder inside the module directory and configure it:

```
c:\godot\modules\mymodule\sdk-1.2> android -p . -t 15
```

As of this writing, Godot uses minSdk 10 and target sdk 15. If this ever changes, it should be reflected in the manifest template: *AndroidManifest.xml.template* <<https://github.com/godotengine/godot/blob/master/platform/android/AndroidManifest.xml.template>>

Then, add the module folder to the project:

(Before Version 2.0)

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # will copy this to the java folder
        env.android_module_file("MySingleton.java")
        env.android_module_manifest("AndroidManifestChunk.xml")
        env.android_module_source("sdk-1.2", "")
```

(After Version 2.0)

23.8.9 Building

As you probably modify the contents of the module, and modify your .java inside the module, you need the module to be built with the rest of Godot, so compile android normally.

```
c:\godot> scons p=android
```

This will cause your module to be included, the .jar will be copied to the java folder, the .java will be copied to the sources folder, etc. Each time you modify the .java, scons must be called.

Afterwards, just continue the steps for compiling android *Compiling for Android*.

Using the module

To use the module from GDScript, first enable the singleton by adding the following line to engine.cfg (Godot Engine 2.0 and greater):

```
[android]
modules="org/godotengine/godot/MySingleton"
```

For Godot Engine 1.1 is

```
[android]
modules="com/android/godot/MySingleton"
```

More than one singleton module can be enabled by separating with commas:

```
[android]
modules="com/android/godot/MySingleton, com/android/godot/MyOtherSingleton"
```

Then just request the singleton Java object from Globals like this:

```
# in any file

var singleton = null

func _init():
    singleton = Globals.get_singleton("MySingleton")
    print(singleton.myFunction("Hello"))
```

23.8.10 Troubleshooting

(This section is a work in progress, report your problems here!)

Godot crashes upon load

Check adb logcat for possible problems, then:

- Make sure libgodot_android.so is in the libs/armeabi folder
- Check that the methods used in the Java singleton only use simple Java datatypes, more complex ones are not supported.

23.8.11 Future

Godot has an experimental Java API Wrapper that allows to use the entire Java API from GDScript.

It's simple to use and it's used like this:

```
class □JavaClassWrapper.wrap(<javaclass as text>)
```

This is most likely not functional yet, if you want to test it and help us make it work, contact us through the [developer mailing list](#).

CHAPTER 24

Community

24.1 Contributing

24.1.1 Ways to contribute

Godot Engine is a non-profit, community-driven free and open source project. Almost all (but our lead dev Juan, more on that below) developers are working *pro bono* on their free time, out of personal interest and for the love of creating a libre engine of exceptional quality.

This means that to thrive, Godot needs as many users as possible to get involved by contributing to the engine. There are many ways to contribute to such a big project, making it possible for everybody to bring something positive to the engine, regardless of their skill set:

- **Be part of the community.** The best way to contribute to Godot and help it become ever better is simply to use the engine and promote it by word-of-mouth, in the credits or splash screen of your games, blog posts, tutorials, videos, demos, gamedev or free software events, support on the Q&A, IRC, forums, Discord, etc. Participate! Being a user and advocate helps spread the word about our great engine, which has no marketing budget and can therefore only rely on its community to become more mainstream.
- **Make games.** It's no secret that, to convince new users and especially the industry at large that Godot is a relevant market player, we need great games made with Godot. We know that the engine has a lot of potential, both for 2D and 3D games, but given its young age we still lack big releases that will draw attention to Godot. So keep working on your awesome projects, each new game increases our credibility on the gamedev market!
- **Get involved in the engine's development.** This can be by contributing code via pull requests, testing the development snapshots or directly the git *master* branch, report bugs or suggest enhancements on the issue tracker, improve the official documentation (both the class reference and tutorials). The following sections will cover each of those "direct" ways of contributing to the engine.
- **Donate.** Godot is a non-profit project, but it can still benefit from user donations for many things. Apart from usual expenses such as hosting costs or promotion material on events, we also use donation money to acquire hardware when necessary (e.g. we used donation money to buy a Macbook Pro to implement Retina/HiDPI support and various other macOS-related features). Most importantly, we also used donation money to hire our lead developer Juan Linietsky, so that he can work full-time on the engine. Even with a low monthly wage, we

need a steady donation income to continue doing this, which has been very beneficial to the project so far. So if you want to donate some money to the project, check [our website](#) for details.

Contributing code

The possibility to study, use, modify and redistribute modifications of the engine's source code are the fundamental rights that Godot's license grant you, making it [free and open source software](#).

As such, everyone is entitled to modify [Godot's source code](#), and send those modifications back to the upstream project in the form of a patch (a text file describing the changes in a ready-to-apply manner) or - in the modern workflow that we use - via a so-called "pull request" (PR), i.e. a proposal to directly merge one or more git commits (patches) into the main development branch.

Contributing code changes upstream has two big advantages:

- Your own code will be reviewed and improved by other developers, and will be further maintained directly in the upstream project, so you won't have to reapply your own changes every time you move to a newer version. On the other hand it comes with a responsibility, as your changes have to be generic enough to be beneficial to all users, and not just your project; so in some cases it might still be relevant to keep your changes only for your own project, if they are too specific.
- The whole community will benefit from your work, and other contributors will behave the same way, contributing code that will be beneficial to you. At the time of this writing, more than 300 developers have contributed code changes to the engine!

To ensure good collaboration and overall quality, the Godot developers enforce some rules for code contributions, for example regarding the style to use in the C++ code (indentation, brackets, etc.) or the git and PR workflow.

See also:

Technical details about the PR workflow are outlined in a specific section, [*Pull request workflow*](#).

Details about the code style guidelines and the `clang-format` tool used to enforce them are outlined in [*Code style guidelines*](#).

Testing and reporting issues

Another great way of contributing to the engine is to test development releases or the development branch and to report issues. It is also helpful to report issues discovered in stable releases, so that they can be fixed in the development branch and in future maintenance releases.

Testing development versions

To help with the testing, you have several possibilities:

- Compile the engine from source yourself, following the instructions of the [*Compiling*](#) page for your platform.
- Test official pre-release binaries when they are announced (usually on the blog and other community platforms), such as alpha, beta and RC (release candidate) builds.
- Test "trusted" unofficial builds of the development branch; just ask community members for reliable providers. Whenever possible, it's best to use official binaries or to compile yourself though, to be sure about the provenance of your binaries.

As mentioned previously, it is also helpful to keep your eyes peeled for potential bugs that might still be present in the stable releases, especially when using some niche features of the engine which might get less testing by the developers.

Filing an issue on GitHub

Godot uses [GitHub's issue tracker](#) for bug reports and enhancement suggestions. You will need a GitHub account to be able to open a new issue there, and click on the “New issue” button.

When you report a bug, you should keep in mind that the process is very similar to an appointment with your doctor. You noticed *symptoms* that make you think that something might be wrong (the engine crashes, some features don't work as expected, etc.). It's the role of the bug triaging team and the developers to then help make the diagnosis of the issue you met, so that the actual cause of the bug can be identified and addressed.

You should therefore always ask yourself: what is relevant information to give so that other Godot contributors can understand the bug, identify it and hopefully fix it. Here are some of the most important infos that you should always provide:

- **Operating system.** Sometimes bugs are system-specific, i.e. they happen only on Windows, or only on Linux, etc. That's particularly relevant for all bugs related to OS interfaces, such as file management, input, window management, audio, etc.
- **Hardware.** Sometimes bugs are hardware-specific, i.e. they happen only on certain processors, graphic cards, etc. If you are able to, it can be helpful to include information on your hardware.
- **Godot version.** This is a must have. Some issues might be relevant in the current stable release, but fixed in the development branch, or the other way around. You might also be using an obsolete version of Godot and experiencing a known issue fixed in a later version, so knowing this from the start helps to speed up the diagnosis.
- **How to reproduce the bug.** In the majority of cases, bugs are reproducible, i.e. it is possible to trigger them reliably by following some steps. Please always describe those steps as clearly as possible, so that everyone can try to reproduce the issue and confirm it. Ideally, make a demo project that reproduces this issue out of the box, zip it and attach it to the issue (you can do this by drag and drop). Even if you think that the issue is trivial to reproduce, adding a minimal project that lets reproduce it is a big added value. You have to keep in mind that there are thousands of issues in the tracker, and developers can only dedicate little time to each issue.

When you click the “New issue” button, you should be presented with a text area prefilled with our issue template. Please try to follow it so that all issues are consistent and provide the required information.

Contributing to the documentation

There are two separate resources referred to as “documentation” in Godot:

- **The class reference.** This is the documentation for the complete Godot API as exposed to GDScript and the other scripting languages. It can be consulted offline, directly in Godot's code editor, or online at [Godot API](#). To contribute to the class reference, you have to edit the `doc/base/classes.xml` in Godot's git repository, and make a pull request. See [Contribute to the Class Reference](#) for more details.
- **The tutorials and engine documentation.** This is the part you are reading now, which is distributed in the HTML, PDF and EPUB formats. Its contents are generated from plain text files in the reStructured Text (rst) format, to which you can contribute via pull requests on the `godot-docs` GitHub repository. See [Documentation guidelines](#) for more details.

24.1.2 Pull request workflow

The so-called “PR workflow” used by Godot is common to many projects using Git, and should be familiar to veteran free software contributors. The idea is that only a small number (if any) commit directly to the *master* branch. Instead, contributors *fork* the project (i.e. create a copy of it, which they can modify as they wish), and then use the GitHub interface to request a *pull* from one of their fork's branch to one branch of the original (often named *upstream*) repository.

The resulting *pull request* (PR) can then be reviewed by other contributors, which might approve it, reject it, or most often request that modifications be done. Once approved, the PR can then be merged by one of the core developers, and its commit(s) will become part of the target branch (usually the *master* branch).

We will go together through an example to show the typical workflow and associated Git commands. But first, let's have a quick look at the organisation of Godot's Git repository.

Git source repository

The repository on GitHub is a Git code repository together with an embedded issue tracker and PR system.

Note: If you are contributing to the documentation, its repository can be found [here](#).

The Git version control system is the tool used to keep track of successive edits to the source code - to contribute efficiently to Godot, learning the basics of the Git command line is *highly* recommended. There exist some graphical interfaces for Git, but they usually encourage users to take bad habits regarding the Git and PR workflow, and we therefore recommend not to use them. In particular, we advise not to use Github's online editor for code contributions (It's tolerated for documentation changes) as it enforces one commit per file and per modification, which quickly leads to PRs with an unreadable Git history (especially after peer review).

See also:

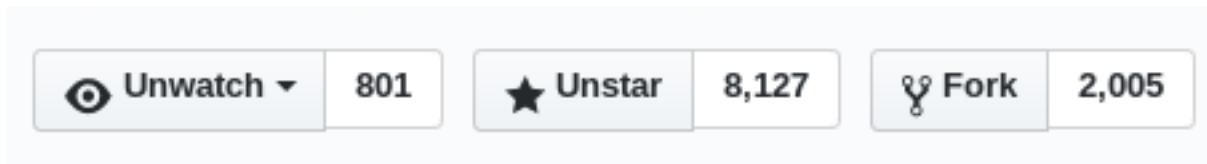
The first sections of Git's "Book" are a good introduction to the tool's philosophy and the various commands you need to master in your daily workflow. You can read them online on the [Git SCM](#) website.

The branches on the Git repository are organized as follows:

- The *master* branch is where the development of the next major version (3.0, 3.1, 4.0, etc.) occurs. As a development branch, it can be unstable and is not meant for use in production. This is where PRs should be done in priority.
- The stable branches are named after their version, e.g. *2.0* and *2.1*. They are used to backport bugfixes and enhancements from the *master* branch to the currently maintained stable release (e.g. *2.0.1* or *2.1.3*). As a rule of thumb, the last stable branch is maintained until the next major version (e.g. the *2.0* branch was maintained until the release of Godot *2.1*). If you want to make PRs against a maintained stable branch, you will have to check if your changes are also relevant for the *master* branch.
- There might be feature branches at time, usually meant to be merged into the *master* branch at some time.

Forking and cloning

The first step is to *fork* the [godotengine/godot](#) repository on GitHub. To do so, you will need to have a GitHub account and to be logged in. In the top right corner of the repository's GitHub page, you should see the "Fork" button as shown below:



Click it, and after a while you should be redirected to your own fork of the Godot repo, with your GitHub username as namespace:

 GitHub, Inc. (US) | <https://github.com/akien-mga/godot>

You can then *clone* your fork, i.e. create a local copy of the online repository (in Git speak, the *origin remote*). If you haven't already, download Git from [its website](#) if you're using Windows or Mac, if you're using Linux install it through your package manager.

Note: if you are on Windows open Git Bash to type commands. Mac and linux users can use their respective terminals.

To clone your fork from GitHub, use the following command:

```
$ git clone https://github.com/USERNAME/godot
```

Note: In our examples, the “\$” character denotes the command line prompt on typical UNIX shells. It is not part of the command and should not be typed.

After a little while, you should have a `godot` directory in your current working directory. Move into it using the `cd` command:

```
$ cd godot
```

We will start by setting up a reference of the original repository that we forked:

```
$ git remote add upstream https://github.com/godotengine/godot
$ git fetch upstream
```

This will create a reference named *upstream* pointing to the original godotengine/godot repository. This will be useful when you want to pull new commits from its *master* branch to update your fork. You have another *remote* reference named *origin*, which points to your fork.

You only need to do the above steps once, as long as you keep that local `godot` folder (which you can move around if you want, the relevant metadata is hidden in its `.git` subfolder).

Note: *Branch it, pull it, code it, stage it, commit, push it, rebase it... technologic.*

This bad take on Daft Punk's *Technologic* shows the general conception Git beginners have of its workflow: lots of strange commands to learn by copy and paste, hoping they will work as expected. And that's actually not a bad way to learn, as long as you're curious and don't hesitate to question your search engine when lost, so we will give you the basic commands to know when working in Git.

In the following, we will assume that you want to implement a feature in Godot's project manager, which is coded in the `editor/project_manager.cpp` file.

Branching

By default, the `git clone` should have put you on the *master* branch of your fork (*origin*). To start your own feature development, we will create a feature branch:

```
// Create the branch based on the current branch (master)
$ git branch better-project-manager
```

```
// Change the current branch to the new one  
$ git checkout better-project-manager
```

This command is equivalent:

```
// Change the current branch to a new named one, based on the current branch  
$ git checkout -b better-project-manager
```

If you want to go back to the *master* branch, you'd use:

```
$ git checkout master
```

You can see which branch you are currently on with the `git branch` command:

```
$ git branch  
 2.1  
* better-project-manager  
  master
```

Updating your branch

This would not be needed the first time, just after you forked the upstream repository. However, the next time you want to work on something, you will notice that your fork's *master* is several commits behind the upstream *master* branch: pull requests from other contributors would have been merged in the meantime.

To ensure there won't be conflicts between the feature you develop and the current upstream *master* branch, you will have to update your branch by *pulling* the upstream branch.

```
$ git pull upstream master
```

However, if you had local commits, this method will create a so-called “merge commit”, and you will soon hear from fellow contributors that those are not wanted in PRs. Then how to update the branch without creating a merge commit? You will have to use the `--rebase` option, so that your local commits are replayed on top of the updated upstream *master* branch. It will effectively modify the Git history of your branch, but that is for the greater good.

Then command that you should (almost) always use is there:

```
$ git pull --rebase upstream master
```

Making changes

You would then do your changes to our example's `editor/project_manager.cpp` file with your usual development environment (text editor, IDE, etc.).

By default, those changes are *unstaged*. The staging area is a layer between your working directory (where you make your modifications) and the local git repository (the commits and all the metadata in the `.git` folder). To bring changes from the working directory to the git repository, you need to *stage* them with the `git add` command, and then to commit them with the `git commit` command.

There are various commands you should know to review your current work, before staging it, while it is staged, and after it has been committed.

- `git diff` will show you the current unstaged changes, i.e. the differences between your working directory and the staging area.

- `git checkout -- <files>` will undo the unstaged changes to the given files.
- `git add <files>` will *stage* the changes on the listed files.
- `git diff --staged` will show the current staged changes, i.e. the differences between the staging area and the last commit.
- `git reset HEAD <files>` will *unstage* changes to the listed files.
- `git status` will show you what are the currently staged and unstaged modifications.
- `git commit` will commit the staged files. It will open a text editor (you can define the one you want to use with the `GIT_EDITOR` environment variable or the `core.editor` setting in your Git config) to let you write a commit log. You can use `git commit -m "Cool commit log"` to write the log directly.
- `git log` will show you the last commits of your current branch. If you did local commits, they should be shown at the top.
- `git show` will show you the changes of the last commit. You can also specify a commit hash to see the changes for that commit.

That's a lot to memorise! Don't worry, just check this cheat sheet when you need to make changes, and learn by doing.

Here's how the shell history could look like on our example:

```
// It's nice to know where you're starting from
$ git log

// Do changes to the project manager with the nano text editor
$ nano editor/project_manager.cpp

// Find an unrelated bug in Control and fix it
$ nano scene/gui/control.cpp

// Review changes
$ git status
$ git diff

// We'll do two commits for our unrelated changes,
// starting by the Control changes necessary for the PM enhancements
$ git add scene/gui/control.cpp
$ git commit -m "Fix handling of margins in Control"

// Check we did good
$ git log
$ git show
$ git status

// Make our second commit
$ git add editor/project_manager.cpp
$ git commit -m "Add a pretty banner to the project manager"
$ git log
```

With this, we should have two new commits in our *better-project-manager* branch which were not in the *master* branch. They are still only local though, the remote fork does not know about them, nor does the upstream repo.

Pushing changes to a remote

That's where `git push` will come into play. In Git, a commit is always done in the local repository (unlike Subversion where a commit will modify the remote repository directly). You need to *push* the new commits to a remote

branch to share them with the world. The syntax for this is:

```
$ git push <remote> <local branch>[:<remote branch>]
```

The part about the remote branch can be omitted if you want it to have the same name as the local branch, which is our case in this example, so we will do:

```
$ git push origin better-project-manager
```

Git will ask you for your username and password, and the changes will be sent to your remote. If you check the fork's page on GitHub, you should see a new branch with your added commits.

Issuing a pull request

When you load your fork's branch on GitHub, you should see a line saying "This branch is 2 commits ahead of godotengine:master." (and potentially some commits behind, if your *master* branch was out of sync with the upstream *master* branch).



On that line, there is a "Pull request" link. Clicking it will open a form that will let you issue a pull request on the godotengine/godot upstream repository. It should show you your two commits, and state "Able to merge". If not (e.g. it has way more commits, or says there are merge conflicts), don't create the PR, something went wrong. Go to IRC and ask for support :)

Use an explicit title for the PR and put the necessary details in the comment area. You can drag and drop screenshots, gifs or zipped projects if relevant, to showcase what your work implements. Click "Create a pull request", and tadaa!

Modifying a pull request

While it is reviewed by other contributors, you will often need to make changes to your yet-unmerged PR, either because contributors requested them, or because you found issues yourself while testing.

The good news is that you can modify a pull request simply by acting on the branch you made the pull request from. You can e.g. make a new commit on that branch, push it to your fork, and the PR will be updated automatically:

```
// Check out your branch again if you had changed in the meantime
$ git checkout better-project-manager

// Fix a mistake
$ nano editor/project_manager.cpp
$ git add editor/project_manager.cpp
$ git commit -m "Fix a typo in the banner's title"
$ git push origin better-project-manager
```

That should do the trick, but...

Mastering the PR workflow: the rebase

On the situation outlined above, your fellow contributors who are particularly pedantic regarding the Git history might ask you to *rebase* your branch to *squash* or *meld* the last two commits together (i.e. the two related to the project manager), as the second commit basically fixes an issue in the first one.

Once the PR is merged, it is not relevant for a changelog reader that the PR author made mistakes; instead, we want to keep only commits that bring from one working state to another working state.

To squash those two commits together, we will have to *rewrite history*. Right, we have that power. You may read that it's a bad practice, and it's true when it comes to branches of the upstream repo. But in your fork, you can do whatever you want, and everything is allowed to get neat PRs :)

We will use the *interactive rebase* `git rebase -i` to do this. This command takes a commit hash as argument, and will let you modify all commits between that commit hash and the last one of the branch, the so-called `HEAD`. In our example, we want to act on the last two commits, so we will do:

```
// The HEAD~X syntax means X commits before HEAD
$ git rebase -i HEAD~2
```

This will open a text editor with:

```
pick 1b4aad7 Add a pretty banner to the project manager
pick e07077e Fix a typo in the banner's title
```

The editor will also show instructions regarding how you can act on those commits. In particular, it should tell you that “pick” means to use that commit (do nothing), and that “squash” and “fixup” can be used to *meld* the commit in its parent commit. The difference between “squash” and “fixup” is that “fixup” will discard the commit log from the squashed commit. In our example, we are not interested in keeping the log of the “Fix a typo” commit, so we use:

```
pick 1b4aad7 Add a pretty banner to the project manager
fixup e07077e Fix a typo in the banner's title
```

Upon saving and quitting the editor, the rebase will occur. The second commit will be melded into the first one, and `git log` and `git show` should now confirm that you have only one commit with the changes from both previous commits.

Note: You could have avoided this rebase by using `git commit --amend` when fixing the typo. This command will write the staged changes directly into the *last* commit (`HEAD`), instead of creating a new commit like we did in this example. So it is equivalent to what we did with a new commit and then a rebase to mark it as “fixup”.

But! You rewrote the history, and now your local and remote branches have diverged. Indeed, commit `1b4aad7` in the above example will have changed, and therefore got a new commit hash. If you try to push to your remote branch, it will raise an error:

```
$ git push origin better-project-manager
To https://github.com/akien-mga/godot
 ! [rejected]          better-project-manager -> better-project-manager (non-fast-
 ↵forward)
error: failed to push some refs to 'https://akien-mga@github.com/akien-mga/godot'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart.
```

This is a sane behaviour, Git will not let you push changes that would override remote content. But that's actually what we want to do here, so we will have to *force* it:

```
$ git push --force origin better-project-manager
```

And tadaa! Git will happily *replace* your remote branch with what you had locally (so make sure that's what you wanted, using `git log`). This will also update the PR accordingly.

Deleting a Git Branch

After your pull request gets merged there's one last thing you should do, delete your Git branch for the PR. There won't be issues if you don't delete your branch, but it's good practice to do so. You'll need to do this twice, once for the local branch and another for the remote branch on GitHub.

To delete our better project manager branch locally use this command:

```
$ git branch -d better-project-manager
```

Alternatively, if the branch hadn't been merged yet and we wanted to delete it anyway, instead of `-d` you would use `-D`.

Next, to delete the remote branch on GitHub use this command:

```
$ git push origin -d better-project-manager
```

24.1.3 Code style guidelines

When contributing to Godot's source code, you will be expected to follow the style guidelines outlined below. Some of them are checked via the Continuous Integration process and reviewers will ask you to fix potential issues, so best setup your system as outlined below to ensure all your commits follow the guidelines.

C++ and Objective-C

There are no written guidelines, but the code style agreed upon by the developers is enforced via the [clang-format](#) code beautifier, which takes care for you of all our conventions. To name a few:

- Indentation and alignment are both tab based (respectively one and two tabs)
- One space around math and assignments operators as well as after commas
- Pointer and reference operators are affixed to the variable identifier, not to the type name

The rules used by clang-format are outlined in the [.clang-format](#) file of the Godot repository.

As long as you ensure that your style matches the surrounding code and that you're not introducing trailing whitespace or space-based indentation, you should be fine. If you plan to contribute regularly however, we strongly advise that you setup clang-format locally to check and automatically fix all your commits.

Warning: Godot's code style should *not* be applied to thirdparty code, i.e. that is included in Godot's source tree but was not written specifically for our project. Such code usually comes from different upstream projects with their own style guides (or lack thereof), and don't want to introduce differences that would make syncing with upstream repositories harder.

Thirdparty code is usually included in the `thirdparty/` folder and can thus easily be excluded from formatting scripts. For the rare cases where a thirdparty code snippet needs to be included directly within a Godot file, you can use `/* clang-format off */` and `/* clang-format on */` to tell clang-format to ignore a chunk of code.

Using clang-format locally

First of all, you will need to install clang-format. As of now, you need to use [clang-format 5.x](#) to be compatible with Godot's format. The upcoming 6.x branch has not been tested yet and may cause inconsistencies; the previous 3.x

branch is incompatible with the style definitions and will error out.

Installation

Here's how to install clang-format:

- Linux: It will usually be available out-of-the-box with the clang toolchain packaged by your distribution. If your distro version is not the required one, you can download a pre-compiled version from the [LLVM website](#), or if you are on a Debian derivative, use the [upstream repos](#).
- macOS and Windows: You can download precompiled binaries from the [LLVM website](#). You may need to add the path to the binary's folder to your system's PATH environment variable to be able to call `clang-format` out of the box.

You then have different possibilities to apply clang-format to your changes:

Manual usage

You can apply clang-format manually one or more files with the following command:

```
clang-format -i <path/to/file(s)>
```

- `-i` means that the changes should be written directly to the file (by default clang-format would only output the fixed version to the terminal).
- The path can point to several files, either one after the other or using wildcards like in a typical Unix shell. Be careful when globbing so that you don't run clang-format on compiled objects (.o and .a files) that are in Godot's tree. So better use `core/*.{cpp,h}` than `core/*`.

Pre-commit hook

For ease of use, we provide a pre-commit hook for Git that will run clang-format automatically on all your commits to check them, and let you apply its changes in the final commit.

This "hook" is a script which can be found in `misc/hooks`, refer to that folder's `README.md` for installation instructions.

If your clang-format is not in the PATH, you may have to edit the `pre-commit-clang-format` to point to the correct binary for it to work. The hook was tested on Linux and macOS, but should also work in the Git Shell on Windows.

IDE plugin

Most IDEs or code editors have beautifier plugins that can be configured to run clang-format automatically, for example each time you save a file.

Here is a non-exhaustive list of beautifier plugins for some IDEs:

- Qt Creator: [Beautifier plugin](#)
- Visual Studio Code: [Clang-Format](#)
- Visual Studio: [ClangFormat](#)
- vim: [vim-clang-format](#)

(Pull requests welcome to extend this list with tested plugins.)

Java

For Godot's Java code (mostly in `platform/android`), there is currently no style guide, so for now try to stay consistent with the existing code.

Once a style is decided upon, it could also be enforced via clang-format.

Python

Godot's SCons buildsystem is written in Python, and various scripts included in the source tree are also using Python.

For those, we follow the [PEP-8 style guide](#), this is however not as strongly enforced as for the C++ code. If you are so inclined, you can check and format your Python changes using [autopep8](#).

24.1.4 Bug triage guidelines

This page describes the typical workflow of the bug triage team aka bugsquad when handling issues and pull requests on Godot's [GitHub](#) repository. It is bound to evolve together with the bugsquad, so do not hesitate to propose modifications to the following guidelines.

Issues management

GitHub proposes various features to manage issues:

- Set one or several labels from a predefined list
- Set one milestone from a predefined list
- Keep track of the issue in the project dashboard
- Define one contributor as “assignee” among the Godot engine organization members

As the Godot engine organization on GitHub currently has a restricted number of contributors, we do not use assignees extensively for now. All contributors are welcome to take on any issue, if relevant after mentioning it on the issue ticket and/or discussing the best way to resolve it with other developers.

For the time being we do not use the project dashboard feature either.

As far as possible, we try to assign labels (and milestones, when relevant) to both issues and pull requests.

Labels

The following labels are currently defined in the Godot repository:

Categories:

- *Archived*: either a duplicate of another issue, or invalid. Such an issue would also be closed.
- *Bug*: describes something that is not working properly.
- *Confirmed*: has been confirmed by at least one other contributor than the bug reporter (typically for *Bug* reports). The purpose of this label is to let developers know which issues are still reproducible when they want to select what to work on. It is therefore a good practice to add in a comment on what platform and what version or commit of Godot the issue could be reproduced; if a developer looks at the issue one year later, the *Confirmed* label may not be relevant anymore.
- *Discussion*: the issue is not consensual and needs further discussion to define what exactly should be done to address the topic.

- *Documentation*: issue related to the documentation. Mainly to request enhancements in the API documentation. Issues related to the ReadTheDocs documentation should be filed on the [godot-docs](#) repository.
- *Enhancement*: describes a proposed enhancement to an existing functionality.
- *Feature proposal*: describes a wish for a new feature to be implemented.
- *Junior job*: the issue is *assumed* to be an easy one to fix, which makes it a great fit for junior contributors who need to become familiar with the code base.
- *Needs rebase*: the issue need a git rebase to be merged.
- *Needs testing*: the issue/pull request could not be completely tested and thus need further testing. This can mean that it needs to be tested on different hardware/software configurations or even that the steps to reproduce are not certain.
- *PR welcome / hero wanted!*: Contributions for issues with these labels are especially welcome. Note that this **doesn't** mean you can't work on issues without these labels.
- *Tracker*: issue used to track other issues (like all issues related to the plugin system).
- *Usability*: issues that directly impact user usability.

The categories are used for general triage of the issues. They can be combined in some way when relevant, e.g. an issue can be labelled *Enhancement* and *Usability* at the same time if it's an issue to improve usability. Or *Feature proposal* and *Discussion* if it's a non-consensual feature request, or one that is not precise enough to be worked on.

Topics:

- *Assetlib*: relates to issues with the asset library.
- *Audio*: relates to the audio features (low and high level).
- *Buildsystem*: relates to building issues, either linked to the SCons buildsystem or to compiler peculiarities.
- *Core*: anything related to the core engine. It might be further split later on as it's a pretty big topic.
- *Drivers*: relates to issues with the drivers used by the engine.
- *Editor*: relates to issues in the editor (mainly UI).
- *GDNative*: relates to the GDNative module.
- *GDScript*: relates to GDScript.
- *Mono*: relates to the C# / Mono bindings.
- *Network*: relates to networking.
- *Physics*: relates to the physics engine (2D/3D).
- *Plugin*: relates to problems encountered while writing plugins.
- *Porting*: relates to some specific platforms.
- *Rendering*: relates to the 2D and 3D rendering engines.
- *VisualScript*: relates to issues with the visual scripting language.

Issues would typically correspond to only one topic, though it's not unthinkable to see issues that fit two bills. The general idea is that there will be specialized contributors teams behind all topics, so they can focus on the issues labelled with their team's topic.

Platforms:

Android, HTML5, iOS, Linux, OS X, Windows, UWP

By default, it is assumed that a given issue applies to all platforms. If one of the platform labels is used, it is then exclusive and the previous assumption doesn't stand anymore (so if it's a bug on e.g. Android and Linux exclusively, select those two platforms).

Milestones

Milestones correspond to planned future versions of Godot for which there is an existing roadmap. Issues that fit in the said roadmap should be filed under the corresponding milestone; if they don't correspond to any current roadmap, they should be left without milestone. As a rule of thumb, an issue corresponds to a given milestone if it concerns a feature that is new in the milestone, or a critical bug that can't be accepted in any future stable release, or anything that Juan wants to work on right now :)

Contributors are free to pick issues regardless of their assigned milestone; if a fix is proposed for a bug that was not deemed urgent and thus without milestone, it would likely still be very welcome.

24.1.5 Documentation guidelines

This page describes the rules to follow if you want to contribute to Godot Engine by writing or reviewing documentation, or by translating existing documentation.

How to contribute

Creating or modifying documentation pages is mainly done via the [godot-docs GitHub repository](#). The HTML (or PDF and EPUB) documentation is generated from the .rst files (reStructuredText markup language) in that repository. Modifying those pages in a pull request and getting it merged will trigger a rebuild of the online documentation.

See also:

For details on Git usage and the pull request workflow, please refer to the [Pull request workflow](#) page. Most of what it describes regarding the main godotengine/godot repository is also valid for the docs repository.

The README.md file contains all the information you need to get you started, please read it. In particular, it contains some tips and tricks and links to reference documentation about the reStructuredText markup language.

Warning: If you want to edit the **API reference**, please note that it should *not* be done in the godot-docs repository. Instead, you should edit the `doc/classes/*` XML files of Godot's main repository. These files are then later used to generate the in-editor documentation as well as the API reference of the online docs.

What makes good documentation?

Documentation should be well written in plain English, using well-formed sentences and various levels of sections and subsections. It should be clear and objective.

We differentiate tutorial pages from other documentation pages by these definitions:

- Tutorial: a page aiming at explaining how to use one or more concepts in the editor or scripts in order to achieve a specific goal with a learning purpose (e.g. “Making a simple 2d Pong game”, “Applying forces to an object”).
- Documentation: a page describing precisely one and only one concept at a time, if possible exhaustively (e.g. the list of methods of the Sprite class, or an overview of the input management in Godot).

You are free to write the kind of documentation you wish, as long as you respect the following rules (and the ones on the repo). In particular, you can contribute tutorials in the “Community” section of the docs, where they could be merged relatively easily, improved by other contributors and eventually moved to an “official” section if relevant.

Titles

Always begin pages with their title and a Sphinx reference name:

```
... _doc_insert_your_title_here:  
  
Insert your title here  
=====
```

The reference allows to link to this page using the :ref: format, e.g. :ref:`doc_insert_your_title_here` would link to the above example page (note the lack of leading underscore in the reference).

Also, avoid American CamelCase titles: title’s first word should begin with a capitalized letter, and every following word should not. Thus, this is a good example:

- Insert your title here

And this is a bad example:

- Insert Your Title Here

Only project, people and node class names should have capitalized first letter.

Translating existing pages

For the moment, only English documentation is provided. We want to propose localized documentation in the future, but this is not a priority for now. Indeed, since the English documentation is still evolving a lot as the community improves it and make it more professional, we prefer that translators do not spend too much time on it, as their translated documentation might quickly grow out of sync with the original.

When the English documentation is ready for translations, we will provide tools to simplify the work of translators by tracking changes to the English docs that they should translate on their end.

License

This documentation and every page it contains is published under the terms of the Creative Commons Attribution 3.0 license (CC-BY-3.0), with attribution to “Juan Linietsky, Ariel Manzur and the Godot community”.

By contributing to the documentation on the GitHub repository, you agree that your changes are distributed under this license.

24.1.6 Docs writing guidelines

The Godot community is rich and international. Users come from all around the world. Some of them are young, and many aren’t native English speakers. That’s why we must all write using a clear and a common language. For the class reference, the goal is to make it easy to read for everyone and precise.

In summary, always try to:

1. Use the direct voice
2. Use precise action verbs

3. Avoid verbs that end in -ing
4. Remove unnecessary adverbs and adjectives.
5. Ban these 8 words: obvious, simple, basic, easy, actual, just, clear, and however
6. Use explicit references
7. Use 's to show possession
8. Use the Oxford comma

There are 3 rules to describe classes:

1. Give an overview of the node in the brief description
2. Mention what methods return if it's useful
3. Use "if true" to describe booleans

Note: A technical writer's job is to pack as much information as possible into the smallest and clearest sentences possible. These guidelines will help you work towards that goal.

7 rules for a clear english

Use the direct voice

Use the direct voice when possible. Take the classes, methods, and constants you describe as the subject. It's natural to write using the passive voice, but it's harder to read and produces longer sentences.

Passive:

```
The man **was bitten** by the dog.
```

Active:

```
The dog bit the man.
```

Don't use the passive voice:

```
void edit_set_pivot ( Vector2 pivot )
[...] This method **is implemented** only in some nodes that inherit Node2D.
```

Do use the node's name as a noun:

```
void edit_set_pivot ( Vector2 pivot )
[...] Only some Node2Ds **implement** this method.
```

Use precise action verbs

Favor precise yet common verbs over generic ones like make, set, and any expression you can replace with a single word.

Don't repeat the method's name. It already states it sets the pivot value to a new one:

```
void edit_set_pivot ( Vector2 pivot )
Set the pivot position of the 2D node to [code]pivot[/code] value. [...]
```

Do explain what's the consequence of this “set”: use precise verbs like place, position, rotate, fade, etc.

```
void edit_set_pivot ( Vector2 pivot )
Position the node's pivot to the [code]pivot[/code] value. [...]
```

Avoid verbs that end in -ing

The progressive forms describe continuous actions. E.g. “is calling”, “is moving”.

Don't use the progressive form for instant changes.

```
Vector2 move ( Vector2 rel_vec )
Move the body in the given direction, **stopping** if there is an obstacle. [...]
```

Do use simple present, preterit or future.

```
Vector2 move ( Vector2 rel_vec )
Moves the body in the vector's direction. The body **stops** if it collides with an
↪obstacle. [...]
```

You may use the progressive tense to describe actions that are continuous in time. Anything like animation or coroutines.

Tip: Verbs can turn into adjectival nouns with -ing. This is not a conjugation, so you may use them: the remaining movement, the missing file, etc.

Remove unnecessary adverbs and adjectives

Write as few adjectives and adverbs as possible. Only use them if they add key information to the description.

Don't use redundant or meaningless adverbs. Words that lengthen the documentation but don't add any information:

```
**Basically** a big texture [...]
```

Do write short sentences in a simple, descriptive language:

```
A big texture [...]
```

Ban these 8 words

Don't ever use these 8 banned words:

1. obvious
2. simple
3. basic
4. easy
5. actual
6. just
7. clear

8. however (some uses)

Game creation and programming aren't simple, and nothing's easy to someone learning to use the API for the first time. Other words in the list, like just or actual won't add any info to the sentence. Don't use corresponding adverbs either: obviously, simply, basically, easily, actually, clearly.

Don't example. The banned words lengthen the description and take attention away from the most important info:

```
**TextureRect**  
Control frame that **simply** draws an assigned texture. It can stretch or not. It's↳  
↳ a **simple** way to **just** show an image in a UI.
```

Do remove them:

```
**TextureRect**  
[Control] node that displays a texture. The texture can stretch to the node's↳  
↳ bounding box or stay in the center. Useful to display sprites in your UIs.
```

“Simple” never helps. Remember, for other users, anything could be complex or frustrate them. There's nothing like a good old *it's simple* to make you cringe. Here's the old brief description, the first sentence on the Timer node's page:

```
**Timer**  
A **simple** Timer node.
```

Do explain what the node does instead:

```
**Timer**  
Calls a function of your choice after a certain duration.
```

Don't use “basic”, it is too vague:

```
**Vector3**  
Vector class, which performs **basic** 3D vector math operations.
```

Do use the brief description to offer an overview of the node:

```
**Vector3**  
Provides essential math functions to manipulate 3D vectors: cross product, normalize,↳  
↳ rotate, etc.
```

Use explicit references

Favor explicit references over implicit ones.

Don't use words like “the former”, “the latter”, etc. They're not the most common in English, and they require you to check the reference.

```
[code]w[/code] and [code]h[/code] define right and bottom margins. The **latter** two↳  
↳ resize the texture so it fits in the defined margin.
```

Do repeat words. They remove all ambiguity:

```
[code]w[/code] and [code]h[/code] define right and bottom margins. **[code]w[/code]↳  
↳ and [code]h[/code]** resize the texture so it fits the margin.
```

If you need to repeat the same variable name 3 or 4 times, you probably need to rephrase your description.

Use 's to show possession

Avoid “The milk **of** the cow”. It feels unnatural in English. Write “The cow’s milk” instead.

Don’t write “of the X”:

```
The region **of the AtlasTexture that is** used.
```

Do use 's. It lets you put the main subject at the start of the sentence, and keep it short:

```
The **AtlasTexture's** used region.
```

Use the Oxford comma to enumerate anything

From the Oxford dictionary:

The ‘Oxford comma’ is an optional comma before the word ‘and’ at the end of a list: *We sell books, videos, and magazines.*

[...] Not all writers and publishers use it, but it can clarify the meaning of a sentence when the items in a list are not single words: *These items are available in black and white, red and yellow, and blue and green.*

Don’t leave the last element of a list without a comma:

```
Create a KinematicBody2D node, a CollisionShape2D node and a sprite node.
```

Do add a comma before *and* or *or*, for the last element of a list with more than two elements.

```
Create a KinematicBody2D node, a CollisionShape2D node, and a sprite node.
```

How to write methods and classes

Give an overview of the node in the brief description

The brief description is the reference’s most important sentence. It’s the user’s first contact with a node:

1. It’s the only description in the “Create New Node” dialog.
2. It’s at the top of every page in the reference

The brief description should explain the node’s role and its functionality, in up to 200 characters.

Don’t write tiny and vague summaries:

```
**Node2D**
Base node for 2D system.
```

Do give an overview of the node’s functionality:

```
**Node2D**
2D game object, parent of all 2D related nodes. Has a position, rotation, scale and z-index.
```

Use the node’s full description to provide more information, and a code example, if possible.

Mention what methods return if it's useful

Some methods return important values. Describe them at the end of the description, ideally on a new line. No need to mention the return values for any method whose name starts with set or get.

Don't use the passive voice:

```
Vector2 move ( Vector2 rel_vec )
[...] The returned vector is how much movement was remaining before being stopped.
```

Do always use “Returns”.

```
Vector2 move ( Vector2 rel_vec )
[...] Returns the remaining movement before the body was stopped.
```

Notice the exception to the “direct voice” rule: with the move method, an external collider can influence the method and the body that calls move. In this case, you can use the passive voice.

Use “if true” to describe booleans

For boolean member variables, always use `if true` and/or `if false`, to stay explicit. Controls whether or not may be ambiguous and won’t work for every member variable.

Also surround boolean values, variable names and methods with `[code]/[code]`.

Do start with “if true”:

```
Timer.autostart
If [code]true[/code] the timer will automatically start when it enters the scene tree.
↳ Default value: [code]false[/code].
```

Use `[code]` around arguments

In the class reference, always surround arguments with `[code]/[code]`. In the documentation and in Godot, it will display like `this`. When you edit XML files in the Godot repository, replace existing arguments written like ‘this’ or ‘this’ with `[code]this[/code]`.

Common vocabulary to use in godot’s docs

The developers chose some specific words to refer to areas of the interface. They’re used in the sources, in the documentation, and you should always use them instead of synonyms, so the users know what you’re talking about.

In the top left corner of the editor lie the `main` menus. In the center, the buttons change the workspace. And together the buttons in the top right are the `playtest` buttons. The area in the center, that displays the 2D or the 3D space, is the `viewport`. At its top, you find a list of tools inside the toolbar.

The tabs or dockable panels on either side of the viewport are `docks`. You have the `FileSystem` dock, the `Scene` dock that contains your scene tree, the `Import` dock, the `Node` dock, and the `Inspector` or `Inspector` dock. With the default layout you may call the tabbed docks tabs: the `Scene` tab, the `Node` tab...

The Animation, Debugger, etc. at the bottom of the viewport are `panels`. Together they make up the bottom panels.

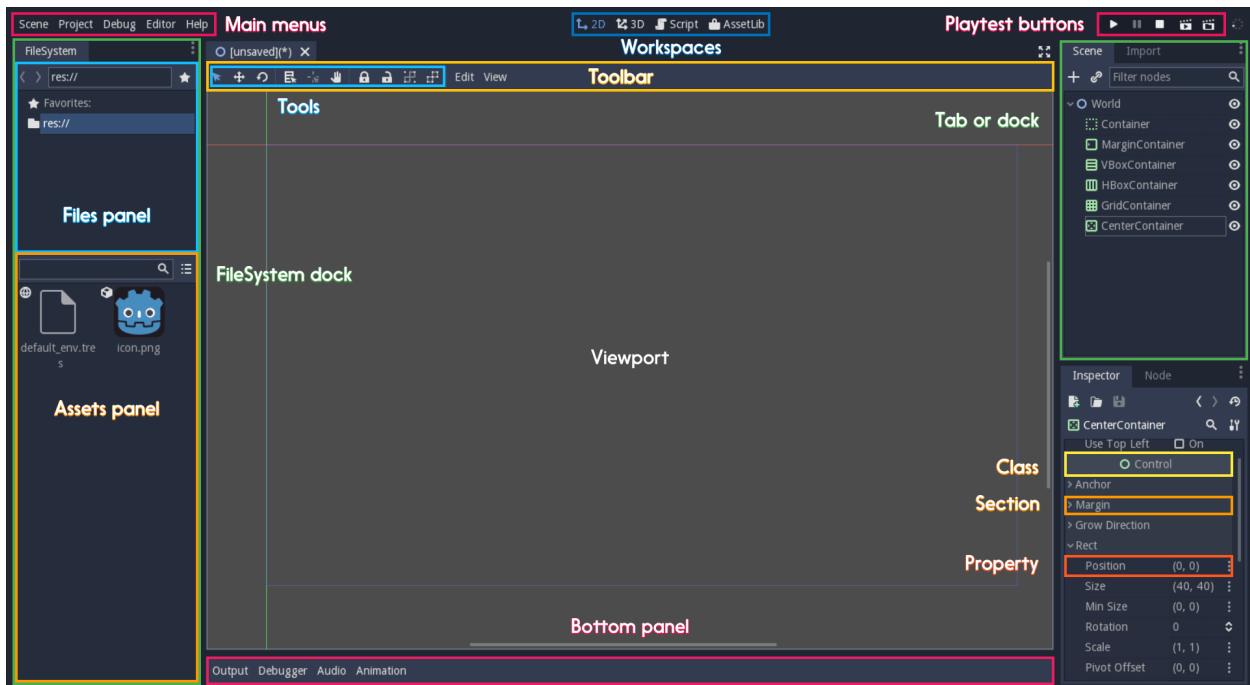


Fig. 24.1: Overview of the interface and common vocabulary

Foldable areas of the Inspector are **sections**. The node's parent class names, which you can't fold, are **Classes** e.g. the `KinematicBody2D` class. And individual lines with key-value pairs are **properties**. E.g. `position` or `modulate color` are both properties.

Image Contribution guidelines

A significant part of the documentation is images, and there are several important guidelines to follow.

First, you should always be using the default editor theme and text when taking screenshots.

For 3D screenshots use 4xMSAA, enable anisotropic filtering on the projects textures, and set the anisotropic filter quality to 16x in Project Settings

Screenshot size should not exceed 1920x1080.

When you need to highlight an area of the editor to show something, like a button or option, use a 2 pixel thick outline without a bevel.

Before you add or replace any images in the documentation, they should be run through a png compressor to save size. The built in lossless compressor in programs like Krita or Photoshop should be done. However you should also use a lossy one, such as [pngquant](#) where almost no image quality is lost during compression.

24.1.7 Contribute to the Class Reference

Godot ships with many nodes and singletons to help you develop your games in GDscript. Each is a class, documented in the *class reference*. This reference is essential for anyone learning the engine: it is available both online and in the engine.

But it's incomplete. Many methods, variables and signals lack descriptions. Others changed with recent releases and need updates. The developers can't write the entire reference on their own. Godot needs you, all of us, to contribute.

Important: we use a collaborative document to track who's working on what class. Always notify other writers about what you are working on. You'll find the instructions in the doc.

Note: This guide is available as a [Youtube video](#).

How to contribute

The class reference lies in the following XML files, in Godot's GitHub repository: [doc/classes/](#).

There are 5 steps to update the class reference (full guide below):

1. Fork Godot's repository
2. Clone your fork on your computer
3. Edit the class file in `doc/classes/` to write documentation
4. Commit your changes and push them to your fork
5. Make a pull request on the Godot repository

Warning: always use these XML files to edit the API reference. Do not edit the generated .rst files *in the online documentation*, hosted in the [godot-docs](#) repository.

Get started with GitHub

If you're new to git and GitHub, this guide will help you get started. You'll learn to:

- Fork and clone Godot's repository
- Keep your fork up to date with other contributors
- Create a pull request so your improvements end in the official docs

Note: If you're new to git, the version-control system Godot uses, go through [GitHub's interactive guide](#). You'll learn some essential vocabulary and get a sense for the tool.

Fork Godot

Fork the Godot Engine into a GitHub repository of your own.

Clone the repository on your computer:

```
git clone https://github.com/your_name/godot.git
```

Create a new branch to make your changes. It makes it a lot easier to sync your improvements with other docs writers, and it's easier to cleanup your repository clean if you have any issues with git.

```
git checkout -b your-new-branch-name
```

The new branch is the same as your master branch, until you start to write API docs. In the `doc/` folder, you'll find the class reference.

How to keep your local clone up-to-date

Other writers contribute to Godot's documentation. Your local repository will fall behind it, and you'll have to synchronize it. Especially if other contributors update the class reference while you work on it.

First add an `upstream` git *remote* to work with. Remotes are links to online repositories you can download new files from.

```
git remote add upstream https://github.com/godotengine/godot
```

You can check the list of all remote servers with:

```
git remote -v
```

You should have 2: `origin`, your fork on github, that git adds by default, and `upstream`, that you just added:

```
origin  https://github.com/your_name/godot.git (fetch)
origin  https://github.com/your_name/godot.git (push)
upstream      https://github.com/godotengine/godot.git (fetch)
upstream      https://github.com/godotengine/godot.git (push)
```

Each time you want to sync your branch to the state of the upstream repository, enter:

```
git pull --rebase upstream master
```

This command will first `fetch`, or download the latest version of the Godot repository. Then, it will reapply your local changes on top.

If you made changes you don't want to keep in your local branch, use the following commands instead:

```
git fetch upstream
git reset --hard upstream master
```

Warning: The above command will reset your branch to the state of the `upstream master` branch. It will discard all local changes. Make sure to only run this *before* you make important changes.

Another option is to delete the branch you're working on, synchronize the master branch with the Godot repository, and create a brand new branch:

```
git checkout master
git branch -d your-new-branch-name
git pull --rebase upstream master
git checkout -b your-new-branch-name
```

If you're feeling lost by now, come to our [IRC channels](#) and ask for help. Experienced git users will give you a hand.

Updating the documentation template

When classes are modified in the source code, the documentation template might become outdated. To make sure that you are editing an up-to-date version, you first need to compile Godot (you can follow the [Introduction to the buildsystem](#) page), and then run the following command (assuming 64-bit Linux):

```
./bin/godot.x11.tools.64 --doctool .
```

The xml files in `doc/classes` should then be up-to-date with current Godot Engine features. You can then check what changed using the `git diff` command. If there are changes to other classes than the one you are planning to document, please commit those changes first before starting to edit the template:

```
git add doc/classes/*.xml  
git commit -m "Sync classes reference template with current code base"
```

You are now ready to edit this file to add stuff.

Note: If this has been done recently by another contributor, you don't forcefully need to go through these steps (unless you know that the class you plan to edit *has* been modified recently).

Push and request a pull of your changes

Once your modifications are finished, push your changes on your GitHub repository:

```
git add doc/classes/<edited_file>.xml  
git commit -m "Explain your modifications."  
git push
```

When it's done, you can ask for a Pull Request via the GitHub UI of your Godot fork.

Warning: Although you can edit files on GitHub, it's not recommended. As hundreds of contributors work on Godot, the git history must stay clean. Each commit should bundle all related improvements you make to the class reference, a new feature, bug fixes... When you edit from GitHub, it will create a new branch and a Pull Request every time you want to save it. If a few days pass before your changes get a review, you won't be able to update to the latest version of the repository cleanly. Also, it's harder to keep clean indents from GitHub. And they're very important in the docs.

TL;DR: If you don't know what you're doing exactly, do not edit files from GitHub.

How to edit class XML

Edit the file for your chosen class in `doc/classes/` to update the class reference. The folder contains an XML file for each class. The XML lists the constants and methods you'll find in the class reference. Godot generates and updates the XML automatically.

Edit it using your favourite text editor. If you use a code editor, make sure that it doesn't change the indent style: tabs for the XML, and 4 spaces inside BBcode-style blocks. More on that below.

How to write the class reference

Each class has a brief and a long description. The brief description is always at the top of the page, while the full description lies below the list of methods, variables and constants. Methods, member variables, constants and signals are in separate categories or XML nodes. For each, learn how they work in Godot's source code, and fill their `<description>`.

Our job is to add the missing text between these marks:

- `<description></description>`
- `<brief_description></brief_description>`
- `<constant></constant>`
- `<method></method>`
- `<member></member>`

- <signal></signal>

Write in a clear and simple language. Always follow the *writing guidelines* to keep your descriptions short and easy to read. **Do not leave empty lines** in the descriptions: each line in the XML file will result in a new paragraph.

Here's how a class looks like in XML:

```
<class name="Node2D" inherits="CanvasItem" category="Core">
    <brief_description>
        Base node for 2D system.
    </brief_description>
    <description>
        Base node for 2D system. Node2D contains a position, rotation and scale, which is used to position and animate. It can alternatively be used with a custom 2D transform ([Matrix32]). A tree of Node2Ds allows complex hierarchies for animation and positioning.
    </description>
    <methods>
        <method name="set_pos">
            <argument index="0" name="pos" type="Vector2">
            </argument>
            <description>
                Set the position of the 2d node.
            </description>
        </method>
        [...]
        <method name="edit_set_pivot">
            <argument index="0" name="arg0" type="Vector2">
            </argument>
            <description>
            </description>
        </method>
    </methods>
    <members>
        <member name="global_position" type="Vector2" setter="set_global_position" getter"get_global_position" brief="">
        </member>
        [...]
        <member name="z_as_relative" type="bool" setter="set_z_as_relative" getter="is_z_relative" brief="">
        </member>
    </members>
    <constants>
    </constants>
</class>
```

Use a code editor like Vim, Atom, Code, Notepad++ or anything similar to edit the file quickly. Use the search function to find classes fast.

Improve formatting with BBcode style tags

Godot's class reference supports BBcode-like tags. They add nice formatting to the text. Here's the list of available tags:

Tag	Effect	Usage	Result
[Class]	Link a class	Move the [Sprite].	Move the <i>Sprite</i> .
[method methodname]	Link to a method in this class	Call [method hide].	See <i>hide</i> .
[method Class.methodname]	Link to another class's method	Call [method Spatial.hide].	See <i>hide</i> .
[member membername]	Link to a member in this class	Get [member scale].	Get <i>scale</i> .
[member Class.membername]	Link to another class's member	Get [member Node2D.scale].	Get <i>scale</i> .
[signal signalname]	Link to a signal in this class	Emit [signal renamed].	Emit <i>renamed</i> .
[signal Class.signalname]	Link to another class's signal	Emit [signal Node.renamed].	Emit <i>renamed</i> .
[b] [/b]	Bold	Some [b]bold[/b] text.	Some bold text.
[i] [/i]	Italic	Some [i]italic[/i] text.	Some <i>italic</i> text.
[code] [/code]	Monospace	Some [code]monospace[/code] text.	Some monospace text.
[codeblock] [/codeblock]	Multiline preformatted block	<i>See below</i> .	<i>See below</i> .

Use [codeblock] for pre-formatted code blocks. Inside [codeblock], always use spaces for indentation (the parser will delete tabs). Example:

```
[codeblock]
func _ready():
    var sprite = get_node("Sprite")
    print(sprite.get_pos())
[/codeblock]
```

Will display as:

```
func _ready():
    var sprite = get_node("Sprite")
    print(sprite.get_pos())
```

I don't know what this method does!

No problem. Leave it behind, and list the methods you skipped when you request a pull of your changes. Another writer will take care of it.

You can still have a look at the methods' implementation in Godot's source code on GitHub. Also, if you have doubts, feel free to ask on the [Q&A website](#) and on IRC (freenode, #godotengine).

Localization

Before we translate the documentation, we need to complete and proof-read it in English. We'll work on localization when we get past 90% completion.

24.2 Channels

So, where is the Godot community and where can you ask questions and get help?

Note that some of these channels are run and moderated by members of the Godot community or third parties.

A brief overview over these channels is also available on the [website](#).

24.2.1 Q & A

- Official Godot Questions & Answers

24.2.2 IRC on Freenode

- General: #godotengine
- Engine development: #godotengine-devel
- Documentation: #godotengine-doc
- GDNative: #godotengine-gdnative
- Website/PR: #godotengine-atelier
- IRC logs

24.2.3 Other chats

- Matrix (IRC compatible)
- Discord

24.2.4 Social networks

- GitHub
- Facebook group
- Twitter (also, #godotengine)
- Reddit
- YouTube
- Steam

24.2.5 Forum

- Forum ([godotdevelopers.org](#))

24.3 Tutorials

This is a list of third-party tutorials created by the community that may be of interest.

24.3.1 Video tutorials

- [GDQuest](#)
- [KidsCanCode](#)

24.4 Resources

This is a list of third-party resources created by the community that may be of interest.

24.4.1 General

- [awesome-godot](#): A curated list of resources by Calinou
- [Zeef Godot Engine](#): A curated directory of resources by Andre Schmitz

CHAPTER 25

Godot API

25.1 @C#

Category: Core

25.1.1 Brief Description

25.2 @GDScript

Category: Core

25.2.1 Brief Description

Built-in GDScript functions.

25.2.2 Member Functions

<i>Color</i>	<i>Color8 (int r8, int g8, int b8, int a8)</i>
<i>Color</i>	<i>ColorN (String name, float alpha)</i>
<i>float</i>	<i>abs (float s)</i>
<i>float</i>	<i>acos (float s)</i>
<i>float</i>	<i>asin (float s)</i>
<i>void</i>	<i>assert (bool condition)</i>
<i>float</i>	<i>atan (float s)</i>
<i>float</i>	<i>atan2 (float x, float y)</i>
<i>Variant</i>	<i>bytes2var (PoolByteArray bytes)</i>
<i>Vector2</i>	<i>cartesian2polar (float x, float y)</i>

Continued on next page

Table 25.1 – continued from previous page

<i>float</i>	<i>ceil (float s)</i>
<i>String</i>	<i>char (int ascii)</i>
<i>float</i>	<i>clamp (float value, float min, float max)</i>
<i>Object</i>	<i>convert (Variant what, int type)</i>
<i>float</i>	<i>cos (float s)</i>
<i>float</i>	<i>cosh (float s)</i>
<i>float</i>	<i>db2linear (float db)</i>
<i>float</i>	<i>decimals (float step)</i>
<i>float</i>	<i>decime (float value, float amount, float step)</i>
<i>float</i>	<i>deg2rad (float deg)</i>
<i>Object</i>	<i>dici2inst (Dictionary dict)</i>
<i>float</i>	<i>ease (float s, float curve)</i>
<i>float</i>	<i>exp (float s)</i>
<i>float</i>	<i>floor (float s)</i>
<i>float</i>	<i>fmod (float x, float y)</i>
<i>float</i>	<i>fposmod (float x, float y)</i>
<i>Object</i>	<i>funcref (Object instance, String funcname)</i>
<i>int</i>	<i>hash (Variant var)</i>
<i>Dictionary</i>	<i>inst2dict (Object inst)</i>
<i>Object</i>	<i>instance_from_id (int instance_id)</i>
<i>float</i>	<i>inverse_lerp (float from, float to, float weight)</i>
<i>bool</i>	<i>is_inf (float s)</i>
<i>bool</i>	<i>is_nan (float s)</i>
<i>int</i>	<i>len (Variant var)</i>
<i>float</i>	<i>lerp (float from, float to, float weight)</i>
<i>float</i>	<i>linear2db (float nrg)</i>
<i>Object</i>	<i>load (String path)</i>
<i>float</i>	<i>log (float s)</i>
<i>float</i>	<i>max (float a, float b)</i>
<i>float</i>	<i>min (float a, float b)</i>
<i>int</i>	<i>nearest_po2 (int value)</i>
<i>Variant</i>	<i>parse_json (String json)</i>
<i>Vector2</i>	<i>polar2cartesian (float r, float th)</i>
<i>float</i>	<i>pow (float x, float y)</i>
<i>Resource</i>	<i>preload (String path)</i>
<i>void</i>	<i>print () vararg</i>
<i>void</i>	<i>print_stack ()</i>
<i>void</i>	<i>printer () vararg</i>
<i>void</i>	<i>printraw () vararg</i>
<i>void</i>	<i>prints () vararg</i>
<i>void</i>	<i>printt () vararg</i>
<i>float</i>	<i>rad2deg (float rad)</i>
<i>float</i>	<i>rand_range (float from, float to)</i>
<i>Array</i>	<i>rand_seed (int seed)</i>
<i>float</i>	<i>randf ()</i>
<i>int</i>	<i>randi ()</i>
<i>void</i>	<i>randomize ()</i>
<i>Array</i>	<i>range () vararg</i>
<i>float</i>	<i>range_lerp (float value, float istart, float istop, float ostart, float ostop)</i>
<i>float</i>	<i>round (float s)</i>

Continued on next page

Table 25.1 – continued from previous page

void	<i>seed (int seed)</i>
float	<i>sign (float s)</i>
float	<i>sin (float s)</i>
float	<i>sinh (float s)</i>
float	<i>sqrt (float s)</i>
float	<i>stepify (float s, float step)</i>
String	<i>str () vararg</i>
Variant	<i>str2var (String string)</i>
float	<i>tan (float s)</i>
float	<i>tanh (float s)</i>
String	<i>to_json (Variant var)</i>
bool	<i>type_exists (String type)</i>
int	<i>typeof (Variant what)</i>
String	<i>validate_json (String json)</i>
PoolByteArray	<i>var2bytes (Variant var)</i>
String	<i>var2str (Variant var)</i>
Object	<i>weakref (Object obj)</i>
float	<i>wrapf (float value, float min, float max)</i>
int	<i>wrapi (int value, int min, int max)</i>
GDScriptFunctionState	<i>yield (Object object, String signal)</i>

25.2.3 Numeric Constants

- **PI = 3.141593** — Constant that represents how many times the diameter of a circle fits around its perimeter.
- **TAU = 6.283185** — The circle constant, the circumference of the unit circle.
- **INF = inf** — A positive infinity. (For negative infinity, use -INF).
- **NAN = nan** — Macro constant that expands to an expression of type float that represents a NaN.

The NaN values are used to identify undefined or non-representable values for floating-point elements, such as the square root of negative numbers or the result of 0/0.

25.2.4 Description

This contains the list of built-in gdscript functions. Mostly math functions and other utilities. Everything else is expanded by objects.

25.2.5 Member Function Description

- **Color Color8 (int r8, int g8, int b8, int a8)**

Returns a 32 bit color with red, green, blue and alpha channels. Each channel has 8 bits of information ranging from 0 to 255.

r8 red channel

g8 green channel

b8 blue channel

a8 alpha channel

```
red = Color8(255, 0, 0)
```

- **Color ColorN (String name, float alpha)**

Returns a color according to the standardised name with alpha ranging from 0 to 1.

```
red = ColorN("red", 1)
```

Supported color names:

"aliceblue", "antiquewhite", "aqua", "aquamarine", "azure", "beige", "bisque", "black", "blanchedalmond", "blue", "blueviolet", "brown", "burlywood", "cadetblue", "chartreuse", "chocolate", "coral", "cornflower", "cornsilk", "crimson", "cyan", "darkblue", "darkcyan", "darkgoldenrod", "darkgray", "darkgreen", "darkkhaki", "darkmagenta", "darkolivegreen", "darkorange", "darkorchid", "darkred", "darksalmon", "darkseagreen", "darkslateblue", "darkslategray", "darkturquoise", "darkviolet", "deeppink", "deepskyblue", "dimgray", "dodgerblue", "firebrick", "floralwhite", "forestgreen", "fuchsia", "gainsboro", "ghostwhite", "gold", "goldenrod", "gray", "webgray", "green", "webgreen", "greenyellow", "honeydew", "hotpink", "indianred", "indigo", "ivory", "khaki", "lavender", "lavenderblush", "lawngreen", "lemonchiffon", "lightblue", "lightcoral", "lightcyan", "lightgoldenrod", "lightgray", "lightgreen", "lightpink", "lightsalmon", "lightseagreen", "lightskyblue", "lightslategray", "lightsteelblue", "lightyellow", "lime", "limegreen", "linen", "magenta", "maroon", "webmaroon", "mediumaquamarine", "mediumblue", "mediumorchid", "mediumpurple", "mediumseagreen", "mediumslateblue", "mediumspringgreen", "mediumturquoise", "mediumvioletred", "midnightblue", "mintcream", "mistyrose", "moccasin", "navajowhite", "navyblue", "oldlace", "olive", "olivedrab", "orange", "orangered", "orchid", "palegoldenrod", "palegreen", "paleturquoise", "palevioletred", "papayawhip", "peachpuff", "peru", "pink", "plum", "powderblue", "purple", "webpurple", "rebeccapurple", "red", "rosybrown", "royalblue", "saddlebrown", "salmon", "sandybrown", "seagreen", "seashell", "sienna", "silver", "skyblue", "slateblue", "slategray", "snow", "springgreen", "steelblue", "tan", "teal", "thistle", "tomato", "turquoise", "violet", "wheat", "white", "whitesmoke", "yellow", "yellowgreen".

- **float abs (float s)**

Returns the absolute value of parameter s (i.e. unsigned value, works for integer and float).

```
# a is 1
a = abs(-1)
```

- **float acos (float s)**

Returns the arc cosine of s in radians. Use to get the angle of cosine s.

```
# c is 0.523599 or 30 degrees if converted with rad2deg(s)
c = acos(0.866025)
```

- **float asin (float s)**

Returns the arc sine of s in radians. Use to get the angle of sine s.

```
# s is 0.523599 or 30 degrees if converted with rad2deg(s)
s = asin(0.5)
```

- **void assert (bool condition)**

Assert that the condition is true. If the condition is false a fatal error is generated and the program is halted. Useful for debugging to make sure a value is always true.

```
# Speed should always be between 0 and 20
speed = -10
assert(speed < 20) # Is true and program continues
assert(speed >= 0) # Is false and program stops
assert(speed >= 0 && speed < 20) # Or combined
```

- **float atan (float s)**

Returns the arc tangent of `s` in radians. Use it to get the angle from an angle's tangent in trigonometry: `atan(tan(angle)) == angle`.

The method cannot know in which quadrant the angle should fall. See `atan2` if you always want an exact angle.

```
a = atan(0.5) # a is 0.463648
```

- **float atan2 (float x, float y)**

Returns the arc tangent of `y/x` in radians. Use to get the angle of tangent `y/x`. To compute the value, the method takes into account the sign of both arguments in order to determine the quadrant.

```
a = atan(0,-1) # a is 3.141593
```

- **Variant bytes2var (PoolByteArray bytes)**

Decodes a byte array back to a value.

- **Vector2 cartesian2polar (float x, float y)**

Converts a 2D point expressed in the cartesian coordinate system (x and y axis) to the polar coordinate system (a distance from the origin and an angle).

- **float ceil (float s)**

Rounds `s` upward, returning the smallest integral value that is not less than `s`.

```
i = ceil(1.45) # i is 2
i = ceil(1.001) # i is 2
```

- **String char (int ascii)**

Returns a character as a String of the given ASCII code.

```
# a is 'A'
a = char(65)
# a is 'a'
a = char(65+32)
```

- **float clamp (float value, float min, float max)**

Clamps `value` and returns a value not less than `min` and not more than `max`.

```
speed = 1000
# a is 20
a = clamp(speed, 1, 20)

speed = -10
# a is 1
a = clamp(speed, 1, 20)
```

- **Object convert (Variant what, int type)**

Converts from a type to another in the best way possible. The `type` parameter uses the enum `TYPE_*` in `@GlobalScope`.

```
a = Vector2(1, 0)
# prints 1
print(a.length())
a = convert(a, TYPE_STRING)
```

```
# prints 6
# (1, 0) is 6 characters
print(a.length())
```

- *float cos (float s)*

Returns the cosine of angle s in radians.

```
# prints 1 and -1
print(cos(PI\*2))
print(cos(PI))
```

- *float cosh (float s)*

Returns the hyperbolic cosine of s in radians.

```
# prints 1.543081
print(cosh(1))
```

- *float db2linear (float db)*

Converts from decibels to linear energy (audio).

- *float decimals (float step)*

Returns the position of the first non-zero digit, after the decimal point.

```
# n is 2
n = decimals(0.035)
```

- *float dectime (float value, float amount, float step)*

Returns the result of value decreased by step * amount.

```
# a = 59
a = dectime(60, 10, 0.1))
```

- *float deg2rad (float deg)*

Returns degrees converted to radians.

```
# r is 3.141593
r = deg2rad(180)
```

- *Object dict2inst (Dictionary dict)*

Converts a previously converted instance to a dictionary, back into an instance. Useful for deserializing.

- *float ease (float s, float curve)*

Easing function, based on exponent. 0 is constant, 1 is linear, 0 to 1 is ease-in, 1+ is ease out. Negative values are in-out/out in.

- *float exp (float s)*

Raises the Euler's constant e to the power of s and returns it. e has an approximate value of 2.71828.

```
a = exp(2) # approximately 7.39
```

- *float floor (float s)*

Rounds s to the closest smaller integer and returns it.

```
# a is 2
a = floor(2.99)
# a is -3
a = floor(-2.99)
```

- **float fmod (float x, float y)**

Returns the floating-point remainder of x/y.

```
# remainder is 1.5
var remainder = fmod(7, 5.5)
```

- **float fposmod (float x, float y)**

Returns the floating-point remainder of x/y that wraps equally in positive and negative.

```
var i = -10;
while i < 0:
    prints(i, fposmod(i, 10))
    i += 1
```

Produces:

```
-10 10
-9 1
-8 2
-7 3
-6 4
-5 5
-4 6
-3 7
-2 8
-1 9
```

- **Object funcref (Object instance, String funcname)**

Returns a reference to the specified function `funcname` in the `instance` node. As functions aren't first-class objects in GDscript, use `funcref` to store a `FuncRef` in a variable and call it later.

```
func foo():
    return("bar")

a = funcref(self, "foo")
print(a.call_func()) # prints bar
```

- **int hash (Variant var)**

Returns the integer hash of the variable passed.

```
print(hash("a")) # prints 177670
```

- **Dictionary inst2dict (Object inst)**

Returns the passed instance converted to a dictionary (useful for serializing).

```
var foo = "bar"
func _ready():
    var d = inst2dict(self)
    print(d.keys())
    print(d.values())
```

Prints out:

```
[@subpath, @path, foo]
[, res://test.gd, bar]
```

- *Object instance_from_id (int instance_id)*

Returns the Object that corresponds to `instance_id`. All Objects have a unique instance ID.

```
var foo = "bar"
func _ready():
    var id = get_instance_id()
    var inst = instance_from_id(id)
    print(inst.foo) # prints bar
```

- *float inverse_lerp (float from, float to, float weight)*

Returns a normalized value considering the given range.

```
inverse_lerp(3, 5, 4) # returns 0.5
```

- *bool is_inf (float s)*

Returns True/False whether `s` is an infinity value (either positive infinity or negative infinity).

- *bool is_nan (float s)*

Returns True/False whether `s` is a NaN (Not-A-Number) value.

- *int len (Variant var)*

Returns length of Variant `var`. Length is the character count of String, element count of Array, size of Dictionary, etc.

Note: Generates a fatal error if Variant can not provide a length.

```
a = [1, 2, 3, 4]
len(a) # returns 4
```

- *float lerp (float from, float to, float weight)*

Linearly interpolates between two values by a normalized value.

```
lerp(1, 3, 0.5) # returns 2
```

- *float linear2db (float nrg)*

Converts from linear energy to decibels (audio).

- *Object load (String path)*

Loads a resource from the filesystem located at `path`. Note: resource paths can be obtained by right clicking on a resource in the Assets Panel and choosing “Copy Path”.

```
# load a scene called main located in the root of the project directory
var main = load("res://main.tscn")
```

- *float log (float s)*

Natural logarithm. The amount of time needed to reach a certain level of continuous growth. Note: This is not the same as the log function on your calculator which is a base 10 logarithm.

```
log(10) # returns 2.302585
```

- *float max (float a, float b)*

Returns the maximum of two values.

```
max(1,2) # returns 2
max(-3.99, -4) # returns -3.99
```

- *float min (float a, float b)*

Returns the minimum of two values.

```
min(1,2) # returns 1
min(-3.99, -4) # returns -4
```

- *int nearest_po2 (int value)*

Returns the nearest larger power of 2 for integer value.

```
nearest_po2(3) # returns 4
nearest_po2(4) # returns 4
nearest_po2(5) # returns 8
```

- *Variant parse_json (String json)*

Parse JSON text to a Variant (use *typeof* to check if it is what you expect).

Be aware that the JSON specification does not define integer or float types, but only a number type. Therefore, parsing a JSON text will convert all numerical values to *float* types.

```
p = parse_json('["a", "b", "c"]')
if typeof(p) == TYPE_ARRAY:
    print(p[0]) # prints a
else:
    print("unexpected results")
```

- *Vector2 polar2cartesian (float r, float th)*

Converts a 2D point expressed in the polar coordinate system (a distance from the origin *r* and an angle *th*) to the cartesian coordinate system (*x* and *y* axis).

- *float pow (float x, float y)*

Returns the result of *x* raised to the power of *y*.

```
pow(2,5) # returns 32
```

- *Resource preload (String path)*

Returns a resource from the filesystem that is loaded during script parsing. Note: resource paths can be obtained by right clicking on a resource in the Assets Panel and choosing “Copy Path”.

```
# load a scene called main located in the root of the project directory
var main = preload("res://main.tscn")
```

- *void print () vararg*

Converts one or more arguments to strings in the best way possible and prints them to the console.

```
a = [1,2,3]
print("a", "b", a) # prints ab[1, 2, 3]
```

- **void print_stack ()**

Prints a stack track at code location, only works when running with debugger turned on.

Output in the console would look something like this:

```
Frame 0 - res://test.gd:16 in function '_process'
```

- **void printerr () vararg**

Prints one or more arguments to strings in the best way possible to standard error line.

```
printerr("prints to stderr")
```

- **void printraw () vararg**

Prints one or more arguments to strings in the best way possible to console. No newline is added at the end.

```
printraw("A")
printraw("B")
# prints AB
```

- **void prints () vararg**

Prints one or more arguments to the console with a space between each argument.

```
prints("A", "B", "C") # prints A B C
```

- **void printt () vararg**

Prints one or more arguments to the console with a tab between each argument.

```
printt("A", "B", "C") # prints A      B      C
```

- **float rad2deg (float rad)**

Converts from radians to degrees.

```
rad2deg(0.523599) # returns 30
```

- **float rand_range (float from, float to)**

Random range, any floating point value between `from` and `to`.

```
prints(rand_range(0, 1), rand_range(0, 1)) # prints 0.135591 0.405263
```

- **Array rand_seed (int seed)**

Random from seed: pass a `seed`, and an array with both number and new seed is returned. “Seed” here refers to the internal state of the pseudo random number generator. The internal state of the current implementation is 64 bits.

- **float randf ()**

Returns a random floating point value between 0 and 1.

```
randf() # returns 0.375671
```

- **int randi ()**

Returns a random 32 bit integer. Use remainder to obtain a random value between 0 and N (where N is smaller than $2^{32} - 1$).

```
randi() % 20      # returns random number between 0 and 19
randi() % 100     # returns random number between 0 and 99
randi() % 100 + 1 # returns random number between 1 and 100
```

- void **randomize ()**

Randomizes the seed (or the internal state) of the random number generator. Current implementation reseeds using a number based on time.

```
func _ready():
    randomize()
```

- **Array range () vararg**

Returns an array with the given range. Range can be 1 argument N (0 to N-1), two arguments (initial, final-1) or three arguments (initial, final-1, increment).

```
for i in range(4):
    print(i)
for i in range(2, 5):
    print(i)
for i in range(0, 6, 2):
    print(i)
```

Output:

```
0
1
2
3

2
3
4

0
2
4
```

- float **range_lerp (float value, float istart, float istop, float ostart, float ostop)**

Maps a value from range [istart, istop] to [ostart, ostop].

```
range_lerp(75, 0, 100, -1, 1) # returns 0.5
```

- float **round (float s)**

Returns the integral value that is nearest to s, with halfway cases rounded away from zero.

```
round(2.6) # returns 3
```

- void **seed (int seed)**

Sets seed for the random number generator.

```
my_seed = "Godot Rocks"
seed(my_seed.hash())
```

- **float sign (float s)**

Returns the sign of s: -1 or 1. Returns 0 if s is 0.

```
sign(-6) # returns -1
sign(0)  # returns 0
sign(6)  # returns 1
```

- **float sin (float s)**

Returns the sine of angle s in radians.

```
sin(0.523599) # returns 0.5
```

- **float sinh (float s)**

Returns the hyperbolic sine of s.

```
a = log(2.0) # returns 0.693147
sinh(a) # returns 0.75
```

- **float sqrt (float s)**

Returns the square root of s.

```
sqrt(9) # returns 3
```

- **float stepify (float s, float step)**

Snaps float value s to a given step.

- **String str () vararg**

Converts one or more arguments to string in the best way possible.

```
var a = [10, 20, 30]
var b = str(a);
len(a) # returns 3
len(b) # returns 12
```

- **Variant str2var (String string)**

Converts a formatted string that was returned by var2str to the original value.

```
a = '{ "a": 1, "b": 2 }'
b = str2var(a)
print(b['a']) # prints 1
```

- **float tan (float s)**

Returns the tangent of angle s in radians.

```
tan( deg2rad(45) ) # returns 1
```

- **float tanh (float s)**

Returns the hyperbolic tangent of s.

```
a = log(2.0) # returns 0.693147
tanh(a)      # returns 0.6
```

- **String to_json (Variant var)**

Converts a Variant `var` to JSON text and return the result. Useful for serializing data to store or send over the network.

```
a = { 'a': 1, 'b': 2 }
b = to_json(a)
print(b) # {"a":1, "b":2}
```

- `bool type_exists (String type)`

Returns whether the given class exists in *ClassDB*.

```
type_exists("Sprite") # returns true
type_exists("Variant") # returns false
```

- `int typeof (Variant what)`

Returns the internal type of the given Variant object, using the `TYPE_*` enum in `@GlobalScope`.

```
p = parse_json('["a", "b", "c"]')
if typeof(p) == TYPE_ARRAY:
    print(p[0]) # prints a
else:
    print("unexpected results")
```

- `String validate_json (String json)`

Checks that `json` is valid JSON data. Returns empty string if valid. Returns error message if not valid.

```
j = to_json([1, 2, 3])
v = validate_json(j)
if not v:
    print("valid")
else:
    prints("invalid", v)
```

- `PoolByteArray var2bytes (Variant var)`

Encodes a variable value to a byte array.

- `String var2str (Variant var)`

Converts a Variant `var` to a formatted string that can later be parsed using `str2var`.

```
a = { 'a': 1, 'b': 2 }
print(var2str(a))
```

prints

```
{
  "a": 1,
  "b": 2
}
```

- `Object weakref (Object obj)`

Returns a weak reference to an object.

A weak reference to an object is not enough to keep the object alive: when the only remaining references to a referent are weak references, garbage collection is free to destroy the referent and reuse its memory for something else. However, until the object is actually destroyed the weak reference may return the object even if there are no strong references to it.

- `float wrapf (float value, float min, float max)`

Wraps float value between min and max.

Usable for creating loop-alike behavior or infinite surfaces.

```
# a is 0.5
a = wrapf(10.5, 0.0, 10.0)
```

```
# a is 9.5
a = wrapf(-0.5, 0.0, 10.0)
```

```
# infinite loop between 0.0 and 0.99
f = wrapf(f + 0.1, 0.0, 1.0)
```

- *int* **wrapi** (*int* value, *int* min, *int* max)

Wraps integer value between min and max.

Usable for creating loop-alike behavior or infinite surfaces.

```
# a is 0
a = wrapi(10, 0, 10)
```

```
# a is 9
a = wrapi(-1, 0, 10)
```

```
# infinite loop between 0 and 9
frame = wrapi(frame + 1, 0, 10)
```

- *GDScriptFunctionState* **yield** (*Object* object, *String* signal)

Stops the function execution and returns the current state. Call *GDScriptFunctionState.resume* on the state to resume execution. This invalidates the state.

Returns anything that was passed to the resume function call. If passed an object and a signal, the execution is resumed when the object's signal is emitted.

25.3 @GlobalScope

Category: Core

25.3.1 Brief Description

Global scope constants and variables.

25.3.2 Member Variables

- *ARVRServer* **ARVRServer** - *ARVRServer* singleton
- *AudioServer* **AudioServer** - *AudioServer* singleton
- *ClassDB* **ClassDB** - *ClassDB* singleton
- *Engine* **Engine** - *Engine* singleton

- *Geometry* **Geometry** - *Geometry* singleton
- *GodotSharp* **GodotSharp** - *GodotSharp* singleton
- *IP* **IP** - *IP* singleton
- *Input* **Input** - *Input* singleton
- *InputMap* **InputMap** - *InputMap* singleton
- *JSON* **JSON** - *JSON* singleton
- *JavaScript* **JavaScript** - *JavaScript* singleton
- *Reference* **Marshalls** - *Marshalls* singleton
- *OS* **OS** - *OS* singleton
- *Performance* **Performance** - *Performance* singleton
- *Physics2DServer* **Physics2DServer** - *Physics2DServer* singleton
- *PhysicsServer* **PhysicsServer** - *PhysicsServer* singleton
- *ProjectSettings* **ProjectSettings** - *ProjectSettings* singleton
- *ResourceLoader* **ResourceLoader** - *ResourceLoader* singleton
- *ResourceSaver* **ResourceSaver** - *ResourceSaver* singleton
- *TranslationServer* **TranslationServer** - *TranslationServer* singleton
- *VisualScriptEditor* **VisualScriptEditor** - *VisualScriptEditor* singleton
- *VisualServer* **VisualServer** - *VisualServer* singleton

25.3.3 Numeric Constants

- **SPKEY = 16777216** — Scancodes with this bit applied are non printable.

25.3.4 Enums

enum **Variant.Operator**

- **OP_EQUAL = 0**
- **OP_NOT_EQUAL = 1**
- **OP_LESS = 2**
- **OP_LESS_EQUAL = 3**
- **OP_GREATER = 4**
- **OP_GREATER_EQUAL = 5**
- **OP_ADD = 6**
- **OP_SUBTRACT = 7**
- **OP_MULTIPLY = 8**
- **OP_DIVIDE = 9**
- **OP_NEGATE = 10**

- **OP_POSITIVE = 11**
- **OP_MODULE = 12**
- **OP_STRING_CONCAT = 13**
- **OP_SHIFT_LEFT = 14**
- **OP_SHIFT_RIGHT = 15**
- **OP_BIT_AND = 16**
- **OP_BIT_OR = 17**
- **OP_BIT_XOR = 18**
- **OP_BIT_NEGATE = 19**
- **OP_AND = 20**
- **OP_OR = 21**
- **OP_XOR = 22**
- **OP_NOT = 23**
- **OP_IN = 24**
- **OP_MAX = 25**

enum MethodFlags

- **METHOD_FLAG_NORMAL = 1** — Flag for normal method
- **METHOD_FLAG_EDITOR = 2** — Flag for editor method
- **METHOD_FLAG_NOSCRIPT = 4**
- **METHOD_FLAG_CONST = 8** — Flag for constant method
- **METHOD_FLAG_REVERSE = 16**
- **METHOD_FLAG_VIRTUAL = 32** — Flag for virtual method
- **METHOD_FLAG_FROM_SCRIPT = 64** — Flag for method from script
- **METHOD_FLAGS_DEFAULT = 1** — Default method flags

enum Orientation

- **VERTICAL = 1** — General vertical alignment, used usually for *Separator*, *ScrollBar*, *Slider*, etc.
- **HORIZONTAL = 0** — General horizontal alignment, used usually for *Separator*, *ScrollBar*, *Slider*, etc.

enum PropertyUsageFlags

- **PROPERTY_USAGE_STORAGE = 1** — Property will be used as storage (default).
- **PROPERTY_USAGE_EDITOR = 2** — Property will be visible in editor (default).
- **PROPERTY_USAGE_NETWORK = 4**
- **PROPERTY_USAGE_EDITOR_HELPER = 8**
- **PROPERTY_USAGE_CHECKABLE = 16**
- **PROPERTY_USAGE_CHECKED = 32**
- **PROPERTY_USAGE INTERNATIONALIZED = 64**
- **PROPERTY_USAGE_GROUP = 128**

- **PROPERTY_USAGE_CATEGORY = 256**
- **PROPERTY_USAGE_STORE_IF_NONZERO = 512**
- **PROPERTY_USAGE_STORE_IF_NONONE = 1024**
- **PROPERTY_USAGE_NO_INSTANCE_STATE = 2048**
- **PROPERTY_USAGE_RESTART_IF_CHANGED = 4096**
- **PROPERTY_USAGE_SCRIPT_VARIABLE = 8192**
- **PROPERTY_USAGE_DEFAULT = 7** — Default usage (storage and editor).
- **PROPERTY_USAGE_DEFAULT_INTL = 71**
- **PROPERTY_USAGE_NOEDITOR = 5**

enum **JoystickList**

- **JOY_BUTTON_0 = 0** — Joypad Button 0
- **JOY_BUTTON_1 = 1** — Joypad Button 1
- **JOY_BUTTON_2 = 2** — Joypad Button 2
- **JOY_BUTTON_3 = 3** — Joypad Button 3
- **JOY_BUTTON_4 = 4** — Joypad Button 4
- **JOY_BUTTON_5 = 5** — Joypad Button 5
- **JOY_BUTTON_6 = 6** — Joypad Button 6
- **JOY_BUTTON_7 = 7** — Joypad Button 7
- **JOY_BUTTON_8 = 8** — Joypad Button 8
- **JOY_BUTTON_9 = 9** — Joypad Button 9
- **JOY_BUTTON_10 = 10** — Joypad Button 10
- **JOY_BUTTON_11 = 11** — Joypad Button 11
- **JOY_BUTTON_12 = 12** — Joypad Button 12
- **JOY_BUTTON_13 = 13** — Joypad Button 13
- **JOY_BUTTON_14 = 14** — Joypad Button 14
- **JOY_BUTTON_15 = 15** — Joypad Button 15
- **JOY_BUTTON_MAX = 16** — Joypad Button 16
- **JOY_SONY_CIRCLE = 1** — DUALSHOCK circle button
- **JOY_SONY_X = 0** — DUALSHOCK X button
- **JOY_SONY_SQUARE = 2** — DUALSHOCK square button
- **JOY_SONY_TRIANGLE = 3** — DUALSHOCK triangle button
- **JOY_XBOX_B = 1** — XBOX controller B button
- **JOY_XBOX_A = 0** — XBOX controller A button
- **JOY_XBOX_X = 2** — XBOX controller X button
- **JOY_XBOX_Y = 3** — XBOX controller Y button
- **JOY_DS_A = 1** — DualShock controller A button

- **JOY_DS_B = 0** — DualShock controller B button
- **JOY_DS_X = 3** — DualShock controller X button
- **JOY_DS_Y = 2** — DualShock controller Y button
- **JOY_SELECT = 10** — Joypad Button Select
- **JOY_START = 11** — Joypad Button Start
- **JOY_DPAD_UP = 12** — Joypad DPad Up
- **JOY_DPAD_DOWN = 13** — Joypad DPad Down
- **JOY_DPAD_LEFT = 14** — Joypad DPad Left
- **JOY_DPAD_RIGHT = 15** — Joypad DPad Right
- **JOY_L = 4** — Joypad Left Shoulder Button
- **JOY_L2 = 6** — Joypad Left Trigger
- **JOY_L3 = 8** — Joypad Left Stick Click
- **JOY_R = 5** — Joypad Right Shoulder Button
- **JOY_R2 = 7** — Joypad Right Trigger
- **JOY_R3 = 9** — Joypad Right Stick Click
- **JOY_AXIS_0 = 0** — Joypad Left Stick Horizontal Axis
- **JOY_AXIS_1 = 1** — Joypad Left Stick Vertical Axis
- **JOY_AXIS_2 = 2** — Joypad Right Stick Horizontal Axis
- **JOY_AXIS_3 = 3** — Joypad Right Stick Vertical Axis
- **JOY_AXIS_4 = 4**
- **JOY_AXIS_5 = 5**
- **JOY_AXIS_6 = 6** — Joypad Left Trigger Analog Axis
- **JOY_AXIS_7 = 7** — Joypad Right Trigger Analog Axis
- **JOY_AXIS_8 = 8**
- **JOY_AXIS_9 = 9**
- **JOY_AXIS_MAX = 10**
- **JOY_ANALOG_LX = 0** — Joypad Left Stick Horizontal Axis
- **JOY_ANALOG LY = 1** — Joypad Left Stick Vertical Axis
- **JOY_ANALOG_RX = 2** — Joypad Right Stick Horizontal Axis
- **JOY_ANALOG_RY = 3** — Joypad Right Stick Vertical Axis
- **JOY_ANALOG_L2 = 6** — Joypad Left Analog Trigger
- **JOY_ANALOG_R2 = 7** — Joypad Right Analog Trigger

enum Error

- **OK = 0** — Functions that return Error return OK when no error occurred. Most functions don't return errors and/or just print errors to STDOUT.
- **FAILED = 1** — Generic error.

- **ERR_UNAVAILABLE = 2** — Unavailable error
- **ERR_UNCONFIGURED = 3** — Unconfigured error
- **ERR_UNAUTHORIZED = 4** — Unauthorized error
- **ERR_PARAMETER_RANGE_ERROR = 5** — Parameter range error
- **ERR_OUT_OF_MEMORY = 6** — Out of memory (OOM) error
- **ERR_FILE_NOT_FOUND = 7** — File: Not found error
- **ERR_FILE_BAD_DRIVE = 8** — File: Bad drive error
- **ERR_FILE_BAD_PATH = 9** — File: Bad path error
- **ERR_FILE_NO_PERMISSION = 10** — File: No permission error
- **ERR_FILE_ALREADY_IN_USE = 11** — File: Already in use error
- **ERR_FILE_CANT_OPEN = 12** — File: Can't open error
- **ERR_FILE_CANT_WRITE = 13** — File: Can't write error
- **ERR_FILE_CANT_READ = 14** — File: Can't read error
- **ERR_FILE_UNRECOGNIZED = 15** — File: Unrecognized error
- **ERR_FILE_CORRUPT = 16** — File: Corrupt error
- **ERR_FILE_MISSING_DEPENDENCIES = 17** — File: Missing dependencies error
- **ERR_FILE_EOF = 18** — File: End of file (EOF) error
- **ERR_CANT_OPEN = 19** — Can't open error
- **ERR_CANT_CREATE = 20** — Can't create error
- **ERR_PARSE_ERROR = 43** — Parse error
- **ERR_QUERY_FAILED = 21** — Query failed error
- **ERR_ALREADY_IN_USE = 22** — Already in use error
- **ERR_LOCKED = 23** — Locked error
- **ERR_TIMEOUT = 24** — Timeout error
- **ERR_CANT_ACQUIRE_RESOURCE = 28** — Can't acquire resource error
- **ERR_INVALID_DATA = 30** — Invalid data error
- **ERR_INVALID_PARAMETER = 31** — Invalid parameter error
- **ERR_ALREADY_EXISTS = 32** — Already exists error
- **ERR_DOES_NOT_EXIST = 33** — Does not exist error
- **ERR_DATABASE_CANT_READ = 34** — Database: Read error
- **ERR_DATABASE_CANT_WRITE = 35** — Database: Write error
- **ERR_COMPILATION_FAILED = 36** — Compilation failed error
- **ERR_METHOD_NOT_FOUND = 37** — Method not found error
- **ERR_LINK_FAILED = 38** — Linking failed error
- **ERR_SCRIPT_FAILED = 39** — Script failed error
- **ERR_CYCLIC_LINK = 40** — Cycling link (import cycle) error

- **ERR_BUSY = 44** — Busy error
- **ERR_HELP = 46** — Help error
- **ERR_BUG = 47** — Bug error

enum **KeyModifierMask**

- **KEY_CODE_MASK = 33554431** — Key Code Mask
- **KEY_MODIFIER_MASK = -16777216** — Modifier Key Mask
- **KEY_MASK_SHIFT = 33554432** — Shift Key Mask
- **KEY_MASK_ALT = 67108864** — Alt Key Mask
- **KEY_MASK_META = 134217728** — Meta Key Mask
- **KEY_MASK_CTRL = 268435456** — CTRL Key Mask
- **KEY_MASK_CMD = 268435456** — CMD Key Mask
- **KEY_MASK_KPAD = 536870912** — Keypad Key Mask
- **KEY_MASK_GROUP_SWITCH = 1073741824** — Group Switch Key Mask

enum **HAlign**

- **HALIGN_LEFT = 0** — Horizontal left alignment, usually for text-derived classes.
- **HALIGN_CENTER = 1** — Horizontal center alignment, usually for text-derived classes.
- **HALIGN_RIGHT = 2** — Horizontal right alignment, usually for text-derived classes.

enum **VAlign**

- **VALIGN_TOP = 0** — Vertical top alignment, usually for text-derived classes.
- **VALIGN_CENTER = 1** — Vertical center alignment, usually for text-derived classes.
- **VALIGN_BOTTOM = 2** — Vertical bottom alignment, usually for text-derived classes.

enum **PropertyHint**

- **PROPERTY_HINT_NONE = 0** — No hint for edited property.
- **PROPERTY_HINT_RANGE = 1** — Hints that the string is a range, defined as “min,max” or “min,max,step”. This is valid for integers and floats.
- **PROPERTY_HINT_EXP_RANGE = 2** — Hints that the string is an exponential range, defined as “min,max” or “min,max,step”. This is valid for integers and floats.
- **PROPERTY_HINT_ENUM = 3** — Property hint for an enumerated value, like “Hello,Something,Else”. This is valid for integer, float and string properties.
- **PROPERTY_HINT_EXP_EASING = 4**
- **PROPERTY_HINT_LENGTH = 5**
- **PROPERTY_HINT_KEY_ACCEL = 7**
- **PROPERTY_HINT_FLAGS = 8** — Property hint for a bitmask description, for bits 0,1,2,3 and 5 the hint would be like “Bit0, Bit1, Bit2, Bit3,, Bit5”. Valid only for integers.
- **PROPERTY_HINT_LAYERS_2D_RENDER = 9**
- **PROPERTY_HINT_LAYERS_2D_PHYSICS = 10**
- **PROPERTY_HINT_LAYERS_3D_RENDER = 11**

- **PROPERTY_HINT_LAYERS_3D_PHYSICS = 12**
- **PROPERTY_HINT_FILE = 13** — String property is a file (so pop up a file dialog when edited). Hint string can be a set of wildcards like “*.doc”.
- **PROPERTY_HINT_DIR = 14** — String property is a directory (so pop up a file dialog when edited).
- **PROPERTY_HINT_GLOBAL_FILE = 15**
- **PROPERTY_HINT_GLOBAL_DIR = 16**
- **PROPERTY_HINT_RESOURCE_TYPE = 17** — String property is a resource, so open the resource popup menu when edited.
- **PROPERTY_HINT_MULTILINE_TEXT = 18**
- **PROPERTY_HINT_COLOR_NO_ALPHA = 19**
- **PROPERTY_HINT_IMAGE_COMPRESS_LOSSY = 20** — Hints that the image is compressed using lossy compression.
- **PROPERTY_HINT_IMAGE_COMPRESS_LOSSLESS = 21** — Hints that the image is compressed using lossless compression.

enum **Corner**

- **CORNER_TOP_LEFT = 0**
- **CORNER_TOP_RIGHT = 1**
- **CORNER_BOTTOM_RIGHT = 2**
- **CORNER_BOTTOM_LEFT = 3**

enum **KeyList**

- **KEY_ESCAPE = 16777217** — Escape Key
- **KEY_TAB = 16777218** — Tab Key
- **KEY_BACKTAB = 16777219** — Shift-Tab Key
- **KEY_BACKSPACE = 16777220** — Backspace Key
- **KEY_ENTER = 16777221** — Return Key (On Main Keyboard)
- **KEY_KP_ENTER = 16777222** — Enter Key (On Numpad)
- **KEY_INSERT = 16777223** — Insert Key
- **KEY_DELETE = 16777224** — Delete Key
- **KEY_PAUSE = 16777225** — Pause Key
- **KEY_PRINT = 16777226** — Printscreen Key
- **KEY_SYSREQ = 16777227** — System Request Key
- **KEY_CLEAR = 16777228** — Clear Key
- **KEY_HOME = 16777229** — Home Key
- **KEY_END = 16777230** — End Key
- **KEY_LEFT = 16777231** — Left Arrow Key
- **KEY_UP = 16777232** — Up Arrow Key
- **KEY_RIGHT = 16777233** — Right Arrow Key

- **KEY_DOWN = 16777234** — Down Arrow Key
- **KEY_PAGEUP = 16777235** — Pageup Key
- **KEY_PAGEDOWN = 16777236** — Pagedown Key
- **KEY_SHIFT = 16777237** — Shift Key
- **KEY_CONTROL = 16777238** — Control Key
- **KEY_META = 16777239** — Meta Key
- **KEY_ALT = 16777240** — Alt Key
- **KEY_CAPSLOCK = 16777241** — Capslock Key
- **KEY_NUMLOCK = 16777242** — Numlock Key
- **KEY_SCROLLLOCK = 16777243** — Scrolllock Key
- **KEY_F1 = 16777244** — F1 Key
- **KEY_F2 = 16777245** — F2 Key
- **KEY_F3 = 16777246** — F3 Key
- **KEY_F4 = 16777247** — F4 Key
- **KEY_F5 = 16777248** — F5 Key
- **KEY_F6 = 16777249** — F6 Key
- **KEY_F7 = 16777250** — F7 Key
- **KEY_F8 = 16777251** — F8 Key
- **KEY_F9 = 16777252** — F9 Key
- **KEY_F10 = 16777253** — F10 Key
- **KEY_F11 = 16777254** — F11 Key
- **KEY_F12 = 16777255** — F12 Key
- **KEY_F13 = 16777256** — F13 Key
- **KEY_F14 = 16777257** — F14 Key
- **KEY_F15 = 16777258** — F15 Key
- **KEY_F16 = 16777259** — F16 Key
- **KEY_KP_MULTIPLY = 16777345** — Multiply Key on Numpad
- **KEY_KP_DIVIDE = 16777346** — Divide Key on Numpad
- **KEY_KP_SUBTRACT = 16777347** — Subtract Key on Numpad
- **KEY_KP_PERIOD = 16777348** — Period Key on Numpad
- **KEY_KP_ADD = 16777349** — Add Key on Numpad
- **KEY_KP_0 = 16777350** — Number 0 on Numpad
- **KEY_KP_1 = 16777351** — Number 1 on Numpad
- **KEY_KP_2 = 16777352** — Number 2 on Numpad
- **KEY_KP_3 = 16777353** — Number 3 on Numpad
- **KEY_KP_4 = 16777354** — Number 4 on Numpad

- **KEY_KP_5 = 16777355** — Number 5 on Numpad
- **KEY_KP_6 = 16777356** — Number 6 on Numpad
- **KEY_KP_7 = 16777357** — Number 7 on Numpad
- **KEY_KP_8 = 16777358** — Number 8 on Numpad
- **KEY_KP_9 = 16777359** — Number 9 on Numpad
- **KEY_SUPER_L = 16777260** — Left Super Key (Windows Key)
- **KEY_SUPER_R = 16777261** — Right Super Key (Windows Key)
- **KEY_MENU = 16777262** — Context menu key
- **KEY_HYPER_L = 16777263** — Left Hyper Key
- **KEY_HYPER_R = 16777264** — Right Hyper Key
- **KEY_HELP = 16777265** — Help key
- **KEY_DIRECTION_L = 16777266** — Left Direction Key
- **KEY_DIRECTION_R = 16777267** — Right Direction Key
- **KEY_BACK = 16777280** — Back key
- **KEY_FORWARD = 16777281** — Forward key
- **KEY_STOP = 16777282** — Stop key
- **KEY_REFRESH = 16777283** — Refresh key
- **KEY_VOLUMEDOWN = 16777284** — Volume down key
- **KEY_VOLUMEMUTE = 16777285** — Mute volume key
- **KEY_VOLUMEUP = 16777286** — Volume up key
- **KEY_BASSBOOST = 16777287** — Bass Boost Key
- **KEY_BASSUP = 16777288** — Bass Up Key
- **KEY_BASSDOWN = 16777289** — Bass Down Key
- **KEY_TREBLEUP = 16777290** — Treble Up Key
- **KEY_TREBLEDOWN = 16777291** — Treble Down Key
- **KEY_MEDIAPLAY = 16777292** — Media play key
- **KEY_MEDIASTOP = 16777293** — Media stop key
- **KEY_MEDIAPREVIOUS = 16777294** — Previous song key
- **KEY_MEDIANEXT = 16777295** — Next song key
- **KEY_MEDIARECORD = 16777296** — Media record key
- **KEY_HOMEPAGE = 16777297** — Home page key
- **KEY_FAVORITES = 16777298** — Favorites key
- **KEY_SEARCH = 16777299** — Search key
- **KEY_STANDBY = 16777300** — Standby Key
- **KEY_OPENURL = 16777301** — Open URL / Launch Browser Key
- **KEY_LAUNCHMAIL = 16777302** — Launch Mail Key

- **KEY_LAUNCHMEDIA = 16777303** — Launch Media Key
- **KEY_LAUNCH0 = 16777304** — Launch Shortcut 0 Key
- **KEY_LAUNCH1 = 16777305** — Launch Shortcut 1 Key
- **KEY_LAUNCH2 = 16777306** — Launch Shortcut 2 Key
- **KEY_LAUNCH3 = 16777307** — Launch Shortcut 3 Key
- **KEY_LAUNCH4 = 16777308** — Launch Shortcut 4 Key
- **KEY_LAUNCH5 = 16777309** — Launch Shortcut 5 Key
- **KEY_LAUNCH6 = 16777310** — Launch Shortcut 6 Key
- **KEY_LAUNCH7 = 16777311** — Launch Shortcut 7 Key
- **KEY_LAUNCH8 = 16777312** — Launch Shortcut 8 Key
- **KEY_LAUNCH9 = 16777313** — Launch Shortcut 9 Key
- **KEY_LAUNCHA = 16777314** — Launch Shortcut A Key
- **KEY_LAUNCHB = 16777315** — Launch Shortcut B Key
- **KEY_LAUNCHC = 16777316** — Launch Shortcut C Key
- **KEY_LAUNCHD = 16777317** — Launch Shortcut D Key
- **KEY_LAUNCHE = 16777318** — Launch Shortcut E Key
- **KEY_LAUNCHF = 16777319** — Launch Shortcut F Key
- **KEY_UNKNOWN = 33554431** — Unknown Key
- **KEY_SPACE = 32** — Space Key
- **KEY_EXCLAM = 33** — ! key
- **KEY_QUOTEDBL = 34** — " key
- **KEY_NUMBERSIGN = 35** — # key
- **KEY_DOLLAR = 36** — \$ key
- **KEY_PERCENT = 37** — % key
- **KEY_AMPERSAND = 38** — & key
- **KEY_APOSTROPHE = 39** — ' key
- **KEY_PARENLEFT = 40** — (key
- **KEY_PARENRIGHT = 41** —) key
- **KEY_ASTERISK = 42** — * key
- **KEY_PLUS = 43** — + key
- **KEY_COMMA = 44** — , key
- **KEY_MINUS = 45** — - key
- **KEY_PERIOD = 46** — . key
- **KEY_SLASH = 47** — / key
- **KEY_0 = 48** — Number 0
- **KEY_1 = 49** — Number 1

- **KEY_2 = 50** — Number 2
- **KEY_3 = 51** — Number 3
- **KEY_4 = 52** — Number 4
- **KEY_5 = 53** — Number 5
- **KEY_6 = 54** — Number 6
- **KEY_7 = 55** — Number 7
- **KEY_8 = 56** — Number 8
- **KEY_9 = 57** — Number 9
- **KEY_COLON = 58** — : key
- **KEY_SEMICOLON = 59** — ; key
- **KEY_LESS = 60** — Lower than key
- **KEY_EQUAL = 61** — = key
- **KEY_GREATER = 62** — Greater than key
- **KEY_QUESTION = 63** — ? key
- **KEY_AT = 64** — @ key
- **KEY_A = 65** — A Key
- **KEY_B = 66** — B Key
- **KEY_C = 67** — C Key
- **KEY_D = 68** — D Key
- **KEY_E = 69** — E Key
- **KEY_F = 70** — F Key
- **KEY_G = 71** — G Key
- **KEY_H = 72** — H Key
- **KEY_I = 73** — I Key
- **KEY_J = 74** — J Key
- **KEY_K = 75** — K Key
- **KEY_L = 76** — L Key
- **KEY_M = 77** — M Key
- **KEY_N = 78** — N Key
- **KEY_O = 79** — O Key
- **KEY_P = 80** — P Key
- **KEY_Q = 81** — Q Key
- **KEY_R = 82** — R Key
- **KEY_S = 83** — S Key
- **KEY_T = 84** — T Key
- **KEY_U = 85** — U Key

- **KEY_V = 86** — V Key
- **KEY_W = 87** — W Key
- **KEY_X = 88** — X Key
- **KEY_Y = 89** — Y Key
- **KEY_Z = 90** — Z Key
- **KEY_BRACKETLEFT = 91** — [key
- **KEY_BACKSLASH = 92** — key
- **KEY_BRACKETRIGHT = 93** —] key
- **KEY_ASCIICIRCUM = 94** — ^ key
- **KEY_UNDERSCORE = 95** — _ key
- **KEY_QUOTELEFT = 96** — Left Quote Key
- **KEY_BRACELEFT = 123** — { key
- **KEY_BAR = 124** — | key
- **KEY_BRACERIGHT = 125** — } key
- **KEY_ASCIITILDE = 126** — ~ key
- **KEY_NOBREAKSPACE = 160**
- **KEY_EXCLAMDOWN = 161**
- **KEY_CENT = 162** — ¢ key
- **KEY_STERLING = 163**
- **KEY_CURRENCY = 164**
- **KEY_YEN = 165** — Yen Key
- **KEY_BROKENBAR = 166** — ¤ key
- **KEY_SECTION = 167** — § key
- **KEY_DIAERESIS = 168** — “ key
- **KEY_COPYRIGHT = 169** — © key
- **KEY_ORDFEMININE = 170**
- **KEY_GUILLEMOTLEFT = 171** — « key
- **KEY_NOTSIGN = 172** — » key
- **KEY_HYPHEN = 173** — - key
- **KEY_REGISTERED = 174** — ® key
- **KEY_MACRON = 175** — Macron Key
- **KEY_DEGREE = 176** — ° key
- **KEY_PLUSMINUS = 177** — ± key
- **KEY_TWOSUPERIOR = 178** — ² key
- **KEY_THREESUPERIOR = 179** — ³ key
- **KEY_ACUTE = 180** — ‘ key

- **KEY_MU = 181** — μ key
- **KEY_PARAGRAPH = 182** — Paragraph Key
- **KEY_PERIODCENTERED = 183** — · key
- **KEY_CEDILLA = 184** — ¬ key
- **KEY_ONESUPERIOR = 185** — ¹ key
- **KEY_MASCULINE = 186** — key
- **KEY_GUILLEMOTRIGHT = 187** — » key
- **KEY_ONEQUARTER = 188** — ^{1/4} key
- **KEY_ONEHALF = 189** — ^{1/2} key
- **KEY_THREEQUARTERS = 190** — ^{3/4} key
- **KEY_QUESTIONDOWN = 191** — ¸ key
- **KEY_AGRAVE = 192** — à key
- **KEY_AACUTE = 193** — á key
- **KEY_ACIRCUMFLEX = 194** — â key
- **KEY_ATILDE = 195** — ã key
- **KEY_ADIAERESIS = 196** — ä key
- **KEY_ARING = 197** — å key
- **KEY_AE = 198** — æ key
- **KEY_CCEDILLA = 199** — ç key
- **KEY_EGRAVE = 200** — è key
- **KEY_EACUTE = 201** — é key
- **KEY_ECIRCUMFLEX = 202** — ê key
- **KEY_EDIAERESIS = 203** — ë key
- **KEY_IGRAVE = 204** — ì key
- **KEY_IACUTE = 205** — í key
- **KEY_ICIRCUMFLEX = 206** — î key
- **KEY_IDIAERESIS = 207** — ë key
- **KEY_ETH = 208** — ð key
- **KEY_NTILDE = 209** — ñ key
- **KEY_OGRAVE = 210** — ò key
- **KEY_OACUTE = 211** — ó key
- **KEY_OCIRCUMFLEX = 212** — ô key
- **KEY_OTILDE = 213** — õ key
- **KEY_ODIAERESIS = 214** — ö key
- **KEY_MULTIPLY = 215** — × key
- **KEY_OOBLIQUE = 216** — ø key

- **KEY_UGRAVE = 217** — ù key
- **KEY_UACUTE = 218** — ú key
- **KEY_UCIRCUMFLEX = 219** — û key
- **KEY_UDIAERESIS = 220** — ü key
- **KEY_YACUTE = 221** — ý key
- **KEY_THORN = 222** — þ key
- **KEY_SSHARP = 223** — ß key
- **KEY_DIVISION = 247** — ÷ key
- **KEY_YDIAERESIS = 255** — ѕ key

enum **Variant.Type**

- **TYPE_NIL = 0** — Variable is of type `nil` (only applied for null).
- **TYPE_BOOL = 1** — Variable is of type `bool`.
- **TYPE_INT = 2** — Variable is of type `int`.
- **TYPE_REAL = 3** — Variable is of type `float/real`.
- **TYPE_STRING = 4** — Variable is of type `String`.
- **TYPE_VECTOR2 = 5** — Variable is of type `Vector2`.
- **TYPE_RECT2 = 6** — Variable is of type `Rect2`.
- **TYPE_VECTOR3 = 7** — Variable is of type `Vector3`.
- **TYPE_TRANSFORM2D = 8** — Variable is of type `Transform2D`.
- **TYPE_PLANE = 9** — Variable is of type `Plane`.
- **TYPE_QUAT = 10** — Variable is of type `Quat`.
- **TYPE_AABB = 11** — Variable is of type `AABB`.
- **TYPE_BASIS = 12** — Variable is of type `Basis`.
- **TYPE_TRANSFORM = 13** — Variable is of type `Transform`.
- **TYPE_COLOR = 14** — Variable is of type `Color`.
- **TYPE_NODE_PATH = 15** — Variable is of type `NodePath`.
- **TYPE RID = 16** — Variable is of type `RID`.
- **TYPE_OBJECT = 17** — Variable is of type `Object`.
- **TYPE_DICTIONARY = 18** — Variable is of type `Dictionary`.
- **TYPE_ARRAY = 19** — Variable is of type `Array`.
- **TYPE_RAW_ARRAY = 20** — Variable is of type `PoolByteArray`.
- **TYPE_INT_ARRAY = 21** — Variable is of type `PoolIntArray`.
- **TYPE_REAL_ARRAY = 22** — Variable is of type `PoolRealArray`.
- **TYPE_STRING_ARRAY = 23** — Variable is of type `PoolStringArray`.
- **TYPE_VECTOR2_ARRAY = 24** — Variable is of type `PoolVector2Array`.
- **TYPE_VECTOR3_ARRAY = 25** — Variable is of type `PoolVector3Array`.

- **TYPE_COLOR_ARRAY = 26** — Variable is of type *PoolColorArray*.
- **TYPE_MAX = 27** — Marker for end of type constants.

enum **Margin**

- **MARGIN_LEFT = 0** — Left margin, used usually for *Control* or *StyleBox* derived classes.
- **MARGIN_TOP = 1** — Top margin, used usually for *Control* or *StyleBox* derived classes.
- **MARGIN_RIGHT = 2** — Right margin, used usually for *Control* or *StyleBox* derived classes.
- **MARGIN_BOTTOM = 3** — Bottom margin, used usually for *Control* or *StyleBox* derived classes.

enum **ButtonList**

- **BUTTON_LEFT = 1** — Left Mouse Button
- **BUTTON_RIGHT = 2** — Right Mouse Button
- **BUTTON_MIDDLE = 3** — Middle Mouse Button
- **BUTTON_WHEEL_UP = 4** — Mouse wheel up
- **BUTTON_WHEEL_DOWN = 5** — Mouse wheel down
- **BUTTON_WHEEL_LEFT = 6** — Mouse wheel left button
- **BUTTON_WHEEL_RIGHT = 7** — Mouse wheel right button
- **BUTTON_MASK_LEFT = 1** — Left Mouse Button Mask
- **BUTTON_MASK_RIGHT = 2** — Right Mouse Button Mask
- **BUTTON_MASK_MIDDLE = 4** — Middle Mouse Button Mask

25.3.5 Description

Global scope constants and variables. This is all that resides in the globals, constants regarding error codes, scancodes, property hints, etc. It's not much.

Singletons are also documented here, since they can be accessed from anywhere.

25.4 @NativeScript

Category: Core

25.4.1 Brief Description

25.5 @VisualScript

Category: Core

25.5.1 Brief Description

Built-in visual script functions.

25.5.2 Description

A list of built-in visual script functions, see *VisualScriptBuiltinFunc* and *VisualScript*.

25.6 AABB

Category: Built-In Types

25.6.1 Brief Description

Axis-Aligned Bounding Box.

25.6.2 Member Functions

<i>AABB</i>	<i>AABB (Vector3 position, Vector3 size)</i>
<i>bool</i>	<i>encloses (AABB with)</i>
<i>AABB</i>	<i>expand (Vector3 to_point)</i>
<i>float</i>	<i>get_area ()</i>
<i>Vector3</i>	<i>get_endpoint (int idx)</i>
<i>Vector3</i>	<i>get_longest_axis ()</i>
<i>int</i>	<i>get_longest_axis_index ()</i>
<i>float</i>	<i>get_longest_axis_size ()</i>
<i>Vector3</i>	<i>get_shortest_axis ()</i>
<i>int</i>	<i>get_shortest_axis_index ()</i>
<i>float</i>	<i>get_shortest_axis_size ()</i>
<i>Vector3</i>	<i>get_support (Vector3 dir)</i>
<i>AABB</i>	<i>grow (float by)</i>
<i>bool</i>	<i>has_no_area ()</i>
<i>bool</i>	<i>has_no_surface ()</i>
<i>bool</i>	<i>has_point (Vector3 point)</i>
<i>AABB</i>	<i>intersection (AABB with)</i>
<i>bool</i>	<i>intersects (AABB with)</i>
<i>bool</i>	<i>intersects_plane (Plane plane)</i>
<i>bool</i>	<i>intersects_segment (Vector3 from, Vector3 to)</i>
<i>AABB</i>	<i>merge (AABB with)</i>

25.6.3 Member Variables

- *Vector3 end* - Ending corner.
- *Vector3 position* - Beginning corner.
- *Vector3 size* - Size from position to end.

25.6.4 Description

AABB consists of a position, a size, and several utility functions. It is typically used for fast overlap tests.

25.6.5 Member Function Description

- **AABB AABB** (*Vector3* position, *Vector3* size)

Optional constructor, accepts position and size.

- *bool encloses* (*AABB* with)

Returns `true` if this *AABB* completely encloses another one.

- **AABB expand** (*Vector3* to_point)

Returns this *AABB* expanded to include a given point.

- *float get_area* ()

Gets the area of the *AABB*.

- *Vector3 get_endpoint* (*int idx*)

Gets the position of the 8 endpoints of the *AABB* in space.

- *Vector3 get_longest_axis* ()

Returns the normalized longest axis of the *AABB*.

- *int get_longest_axis_index* ()

Returns the index of the longest axis of the *AABB* (according to *Vector3::AXIS** enum).

- *float get_longest_axis_size* ()

Returns the scalar length of the longest axis of the *AABB*.

- *Vector3 get_shortest_axis* ()

Returns the normalized shortest axis of the *AABB*.

- *int get_shortest_axis_index* ()

Returns the index of the shortest axis of the *AABB* (according to *Vector3::AXIS** enum).

- *float get_shortest_axis_size* ()

Returns the scalar length of the shortest axis of the *AABB*.

- *Vector3 get_support* (*Vector3 dir*)

Returns the support point in a given direction. This is useful for collision detection algorithms.

- **AABB grow** (*float by*)

Returns a copy of the *AABB* grown a given amount of units towards all the sides.

- *bool has_no_area* ()

Returns `true` if the *AABB* is flat or empty.

- *bool has_no_surface* ()

Returns `true` if the *AABB* is empty.

- *bool has_point* (*Vector3 point*)

Returns `true` if the *AABB* contains a point.

- **AABB intersection** (*AABB* with)

Returns the intersection between two *AABB*. An empty *AABB* (size 0,0,0) is returned on failure.

- *bool intersects* (*AABB* with)

Returns `true` if the AABB overlaps with another.

- `bool intersects_plane (Plane plane)`

Returns `true` if the AABB is on both sides of a plane.

- `bool intersects_segment (Vector3 from, Vector3 to)`

Returns `true` if the AABB intersects the line segment between `from` and `to`.

- `AABB merge (AABB with)`

Returns a larger AABB that contains this AABB and `with`.

25.7 AcceptDialog

Inherits: `WindowDialog < Popup < Control < CanvasItem < Node < Object`

Inherited By: `ConfirmationDialog`

Category: Core

25.7.1 Brief Description

Base dialog for user notification.

25.7.2 Member Functions

<code>Button</code>	<code>add_button (String text, bool right=false, String action="")</code>
<code>Button</code>	<code>add_cancel (String name)</code>
<code>Label</code>	<code>get_label ()</code>
<code>Button</code>	<code>get_ok ()</code>
<code>void</code>	<code>register_text_enter (Node line_edit)</code>

25.7.3 Signals

- `confirmed ()`

Emitted when the dialog is accepted.

- `custom_action (String action)`

Emitted when a custom button is pressed. See `add_button`.

25.7.4 Member Variables

- `bool dialog_hide_on_ok` - If `true` the dialog is hidden when accepted. Default value: `true`.
- `String dialog_text` - The text displayed by this dialog.

25.7.5 Description

This dialog is useful for small notifications to the user about an event. It can only be accepted or closed, with the same result.

25.7.6 Member Function Description

- *Button add_button (String text, bool right=false, String action="")*

Adds a button with label *text* and a custom *action* to the dialog and returns the created button. *action* will be passed to the custom_action signal when pressed.

If *true*, *right* will place the button to the right of any sibling buttons. Default value: *false*.

- *Button add_cancel (String name)*

Adds a button with label *name* and a cancel action to the dialog and returns the created button.

- *Label get_label ()*

Return the label used for built-in text.

- *Button get_ok ()*

Return the OK Button.

- *void register_text_enter (Node line_edit)*

Registers a *LineEdit* in the dialog. When the enter key is pressed, the dialog will be accepted.

25.8 AnimatedSprite

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.8.1 Brief Description

Sprite node that can use multiple textures for animation.

25.8.2 Member Functions

<i>bool</i>	<i>is_playing () const</i>
<i>void</i>	<i>play (String anim="")</i>
<i>void</i>	<i>stop ()</i>

25.8.3 Signals

- *animation_finished ()*

Emitted when the animation is finished (when it plays the last frame). If the animation is looping, this signal is emitted every time the last frame is drawn.

- *frame_changed ()*

Emitted when *frame* changed.

25.8.4 Member Variables

- *String animation* - The current animation from the `frames` resource. If this value changes, the `frame` counter is reset.
- *bool centered* - If `true` texture will be centered. Default value: `true`.
- *bool flip_h* - If `true` texture is flipped horizontally. Default value: `false`.
- *bool flip_v* - If `true` texture is flipped vertically. Default value: `false`.
- *int frame* - The displayed animation frame's index.
- *SpriteFrames frames* - The `SpriteFrames` resource containing the animation(s).
- *Vector2 offset* - The texture's drawing offset.
- *bool playing* - If `true` the *animation* is currently playing.

25.8.5 Description

Animations are created using a `SpriteFrames` resource, which can be configured in the editor via the `SpriteFrames` panel.

25.8.6 Member Function Description

- *bool is_playing () const*

Return true if an animation is currently being played.

- *void play (String anim="")*

Play the animation set in parameter. If no parameter is provided, the current animation is played.

- *void stop ()*

Stop the current animation (does not reset the frame counter).

25.9 AnimatedSprite3D

Inherits: `SpriteBase3D < GeometryInstance < VisualInstance < Spatial < Node < Object`

Category: Core

25.9.1 Brief Description

2D sprite node in 3D world, that can use multiple 2D textures for animation.

25.9.2 Member Functions

<i>bool</i>	<i>is_playing () const</i>
<i>void</i>	<i>play (String anim="")</i>
<i>void</i>	<i>stop ()</i>

25.9.3 Signals

- **frame_changed ()**

Emitted when *frame* changed.

25.9.4 Member Variables

- *String animation* - The current animation from the *frames* resource. If this value changes, the *frame* counter is reset.
- *int frame* - The displayed animation frame's index.
- *SpriteFrames frames* - The *SpriteFrames* resource containing the animation(s).
- *bool playing* - If *true* the *animation* is currently playing.

25.9.5 Description

Animations are created using a *SpriteFrames* resource, which can be configured in the editor via the *SpriteFrames* panel.

25.9.6 Member Function Description

- *bool is_playing () const*

Return *true* if an animation is currently being played.

- *void play (String anim="")*

Play the animation set in parameter. If no parameter is provided, the current animation is played.

- *void stop ()*

Stop the current animation (does not reset the frame counter).

25.10 Animation

Inherits: *Resource < Reference < Object*

Category: Core

25.10.1 Brief Description

Contains data used to animate everything in the engine.

25.10.2 Member Functions

<i>int</i>	<i>add_track (int type, int at_position=-1)</i>
<i>void</i>	<i>clear ()</i>
<i>void</i>	<i>copy_track (int track, Animation to_animation)</i>
<i>int</i>	<i>find_track (NodePath path) const</i>
<i>int</i>	<i>get_track_count () const</i>
<i>PoolIntArray</i>	<i>method_track_get_key_indices (int idx, float time_sec, float delta) const</i>
<i>String</i>	<i>method_track_get_name (int idx, int key_idx) const</i>
<i>Array</i>	<i>method_track_get_params (int idx, int key_idx) const</i>
<i>void</i>	<i>remove_track (int idx)</i>
<i>int</i>	<i>track_find_key (int idx, float time, bool exact=false) const</i>
<i>bool</i>	<i>track_get_interpolation_loop_wrap (int idx) const</i>
<i>int</i>	<i>track_get_interpolation_type (int idx) const</i>
<i>int</i>	<i>track_get_key_count (int idx) const</i>
<i>float</i>	<i>track_get_key_time (int idx, int key_idx) const</i>
<i>float</i>	<i>track_get_key_transition (int idx, int key_idx) const</i>
<i>Variant</i>	<i>track_get_key_value (int idx, int key_idx) const</i>
<i>NodePath</i>	<i>track_get_path (int idx) const</i>
<i>int</i>	<i>track_get_type (int idx) const</i>
<i>void</i>	<i>track_insert_key (int idx, float time, Variant key, float transition=1)</i>
<i>bool</i>	<i>track_is_enabled (int idx) const</i>
<i>bool</i>	<i>track_is_imported (int idx) const</i>
<i>void</i>	<i>track_move_down (int idx)</i>
<i>void</i>	<i>track_move_up (int idx)</i>
<i>void</i>	<i>track_remove_key (int idx, int key_idx)</i>
<i>void</i>	<i>track_remove_key_at_position (int idx, float position)</i>
<i>void</i>	<i>track_set_enabled (int idx, bool enabled)</i>
<i>void</i>	<i>track_set_imported (int idx, bool imported)</i>
<i>void</i>	<i>track_set_interpolation_loop_wrap (int idx, bool interpolation)</i>
<i>void</i>	<i>track_set_interpolation_type (int idx, int interpolation)</i>
<i>void</i>	<i>track_set_key_transition (int idx, int key_idx, float transition)</i>
<i>void</i>	<i>track_set_key_value (int idx, int key, Variant value)</i>
<i>void</i>	<i>track_set_path (int idx, NodePath path)</i>
<i>int</i>	<i>transform_track_insert_key (int idx, float time, Vector3 location, Quat rotation, Vector3 scale)</i>
<i>Array</i>	<i>transform_track_interpolate (int idx, float time_sec) const</i>
<i>PoolIntArray</i>	<i>value_track_get_key_indices (int idx, float time_sec, float delta) const</i>
<i>int</i>	<i>value_track_get_update_mode (int idx) const</i>
<i>void</i>	<i>value_track_set_update_mode (int idx, int mode)</i>

25.10.3 Member Variables

- ***float length*** - The total length of the animation (in seconds). Note that length is not delimited by the last key, as this one may be before or after the end to ensure correct interpolation and looping.
- ***bool loop*** - A flag indicating that the animation must loop. This is used for correct interpolation of animation cycles, and for hinting the player that it must restart the animation.
- ***float step*** - The animation step value.

25.10.4 Enums

enum UpdateMode

- **UPDATE_CONTINUOUS = 0** — Update between keyframes.
- **UPDATE_DISCRETE = 1** — Update at the keyframes and hold the value.
- **UPDATE_TRIGGER = 2** — Update at the keyframes.

enum InterpolationType

- **INTERPOLATION_NEAREST = 0** — No interpolation (nearest value).
- **INTERPOLATION_LINEAR = 1** — Linear interpolation.
- **INTERPOLATION_CUBIC = 2** — Cubic interpolation.

enum TrackType

- **TYPE_VALUE = 0** — Value tracks set values in node properties, but only those which can be Interpolated.
- **TYPE_TRANSFORM = 1** — Transform tracks are used to change node local transforms or skeleton pose bones. Transitions are Interpolated.
- **TYPE_METHOD = 2** — Method tracks call functions with given arguments per key.

25.10.5 Description

An Animation resource contains data used to animate everything in the engine. Animations are divided into tracks, and each track must be linked to a node. The state of that node can be changed through time, by adding timed keys (events) to the track.

Animations are just data containers, and must be added to nodes such as an *AnimationPlayer* or *AnimationTreePlayer* to be played back.

25.10.6 Member Function Description

- `int add_track (int type, int at_position=-1)`

Add a track to the Animation. The track type must be specified as any of the values in the `TYPE_*` enumeration.

- `void clear ()`

Clear the animation (clear all tracks and reset all).

- `void copy_track (int track, Animation to_animation)`

Adds a new track that is a copy of the given track from `to_animation`.

- `int find_track (NodePath path) const`

Return the index of the specified track. If the track is not found, return -1.

- `int get_track_count () const`

Return the amount of tracks in the animation.

- `PoolIntArray method_track_get_key_indices (int idx, float time_sec, float delta) const`

Return all the key indices of a method track, given a position and delta time.

- `String method_track_get_name (int idx, int key_idx) const`

Return the method name of a method track.

- **Array** **method_track_get_params** (*int idx, int key_idx*) const

Return the arguments values to be called on a method track for a given key in a given track.

- **void** **remove_track** (*int idx*)

Remove a track by specifying the track index.

- **int** **track_find_key** (*int idx, float time, bool exact=false*) const

Find the key index by time in a given track. Optionally, only find it if the exact time is given.

- **bool** **track_get_interpolation_loop_wrap** (*int idx*) const

Returns `true` if the track at `idx` wraps the interpolation loop. Default value: `true`.

- **int** **track_get_interpolation_type** (*int idx*) const

Return the interpolation type of a given track, from the `INTERPOLATION_*` enum.

- **int** **track_get_key_count** (*int idx*) const

Return the amount of keys in a given track.

- **float** **track_get_key_time** (*int idx, int key_idx*) const

Return the time at which the key is located.

- **float** **track_get_key_transition** (*int idx, int key_idx*) const

Return the transition curve (easing) for a specific key (see built-in math function “ease”).

- **Variant** **track_get_key_value** (*int idx, int key_idx*) const

Return the value of a given key in a given track.

- **NodePath** **track_get_path** (*int idx*) const

Get the path of a track. for more information on the path format, see `track_set_path`

- **int** **track_get_type** (*int idx*) const

Get the type of a track.

- **void** **track_insert_key** (*int idx, float time, Variant key, float transition=1*)

Insert a generic key in a given track.

- **bool** **track_is_enabled** (*int idx*) const

Returns `true` if the track at index `idx` is enabled.

- **bool** **track_is_imported** (*int idx*) const

Return true if the given track is imported. Else, return false.

- **void** **track_move_down** (*int idx*)

Move a track down.

- **void** **track_move_up** (*int idx*)

Move a track up.

- **void** **track_remove_key** (*int idx, int key_idx*)

Remove a key by index in a given track.

- **void** **track_remove_key_at_position** (*int idx, float position*)

Remove a key by position (seconds) in a given track.

- `void track_set_enabled (int idx, bool enabled)`

Enables/disables the given track. Tracks are enabled by default.

- `void track_set_imported (int idx, bool imported)`

Set the given track as imported or not.

- `void track_set_interpolation_loop_wrap (int idx, bool interpolation)`

If `true` the track at `idx` wraps the interpolation loop.

- `void track_set_interpolation_type (int idx, int interpolation)`

Set the interpolation type of a given track, from the `INTERPOLATION_*` enum.

- `void track_set_key_transition (int idx, int key_idx, float transition)`

Set the transition curve (easing) for a specific key (see built-in math function “ease”).

- `void track_set_key_value (int idx, int key, Variant value)`

Set the value of an existing key.

- `void track_set_path (int idx, NodePath path)`

Set the path of a track. Paths must be valid scene-tree paths to a node, and must be specified starting from the parent node of the node that will reproduce the animation. Tracks that control properties or bones must append their name after the path, separated by “:”. Example: “character/skeleton:ankle” or “character/mesh:transform/local”

- `int transform_track_insert_key (int idx, float time, Vector3 location, Quat rotation, Vector3 scale)`

Insert a transform key for a transform track.

- `Array transform_track_interpolate (int idx, float time_sec) const`

Return the interpolated value of a transform track at a given time (in seconds). An array consisting of 3 elements: position (`Vector3`), rotation (`Quat`) and scale (`Vector3`).

- `PoolIntArray value_track_get_key_indices (int idx, float time_sec, float delta) const`

Return all the key indices of a value track, given a position and delta time.

- `int value_track_get_update_mode (int idx) const`

Return the update mode of a value track.

- `void value_track_set_update_mode (int idx, int mode)`

Set the update mode (`UPDATE_*`) of a value track.

25.11 AnimationPlayer

Inherits: `Node < Object`

Category: Core

25.11.1 Brief Description

Container and player of *Animation* resources.

25.11.2 Member Functions

<i>int</i>	<i>add_animation (String name, Animation animation)</i>
<i>void</i>	<i>advance (float delta)</i>
<i>String</i>	<i>animation_get_next (String anim_from) const</i>
<i>void</i>	<i>animation_set_next (String anim_from, String anim_to)</i>
<i>void</i>	<i>clear_caches ()</i>
<i>void</i>	<i>clear_queue ()</i>
<i>String</i>	<i>find_animation (Animation animation) const</i>
<i>Animation</i>	<i>get_animation (String name) const</i>
<i>PoolStringArray</i>	<i>get_animation_list () const</i>
<i>float</i>	<i>get_blend_time (String anim_from, String anim_to) const</i>
<i>bool</i>	<i>has_animation (String name) const</i>
<i>bool</i>	<i>is_playing () const</i>
<i>void</i>	<i>play (String name="" , float custom_blend=-1, float custom_speed=1.0, bool from_end=false)</i>
<i>void</i>	<i>play_backwards (String name="" , float custom_blend=-1)</i>
<i>void</i>	<i>queue (String name)</i>
<i>void</i>	<i>remove_animation (String name)</i>
<i>void</i>	<i>rename_animation (String name, String newname)</i>
<i>void</i>	<i>seek (float seconds, bool update=false)</i>
<i>void</i>	<i>set_blend_time (String anim_from, String anim_to, float sec)</i>
<i>void</i>	<i>stop (bool reset=true)</i>

25.11.3 Signals

- **animation_changed** (*String old_name, String new_name*)

If the currently being played animation changes, this signal will notify of such change.

- **animation_finished** (*String anim_name*)

Notifies when an animation finished playing.

- **animation_started** (*String anim_name*)

Notifies when an animation starts playing.

25.11.4 Member Variables

- *String assigned_animation* - If playing, the current animation; otherwise, the animation last played. When set, would change the animation, but would not play it unless currently playing. See also *current_animation*.
- *String autoplay* - The name of the animation to play when the scene loads. Default value: "".
- *String current_animation* - The name of the current animation, "" if not playing anything. When being set, does not restart the animation. See also *play*. Default value: "".
- *float current_animation_length* - The length (in seconds) of the currently being played animation.
- *float current_animation_position* - The position (in seconds) of the currently playing animation.
- *bool playback_active* - If true, updates animations in response to process-related notifications. Default value: true.

- *float playback_default_blend_time* - The default time in which to blend animations. Ranges from 0 to 4096 with 0.01 precision. Default value: 0.
- *AnimationProcessMode playback_process_mode* - The process notification in which to update animations. Default value: enum ANIMATION_PROCESS_IDLE.
- *float playback_speed* - The speed scaling ratio. For instance, if this value is 1 then the animation plays at normal speed. If it's 0.5 then it plays at half speed. If it's 2 then it plays at double speed. Default value: 1.
- *NodePath root_node* - The node from which node path references will travel. Default value: "...".

25.11.5 Enums

enum AnimationProcessMode

- **ANIMATION_PROCESS_PHYSICS = 0** — Process animation during the physics process. This is especially useful when animating physics bodies.
- **ANIMATION_PROCESS_IDLE = 1** — Process animation during the idle process.

25.11.6 Description

An animation player is used for general purpose playback of *Animation* resources. It contains a dictionary of animations (referenced by name) and custom blend times between their transitions. Additionally, animations can be played and blended in different channels.

25.11.7 Member Function Description

- *int add_animation (String name, Animation animation)*

Adds `animation` to the player accessible with the key `name`.

- *void advance (float delta)*

Shifts position in the animation timeline. Delta is the time in seconds to shift.

- *String animation_get_next (String anim_from) const*

Returns the name of the next animation in the queue.

- *void animation_set_next (String anim_from, String anim_to)*

Triggers the `anim_to` animation when the `anim_from` animation completes.

- *void clear_caches ()*

`AnimationPlayer` caches animated nodes. It may not notice if a node disappears, so `clear_caches` forces it to update the cache again.

- *void clear_queue ()*

Clears all queued, unplayed animations.

- *String find_animation (Animation animation) const*

Returns the name of `animation` or empty string if not found.

- *Animation get_animation (String name) const*

Returns the `Animation` with key `name` or `null` if not found.

- `PoolStringArray get_animation_list() const`

Returns the list of stored animation names.

- `float get_blend_time(String anim_from, String anim_to) const`

Get the blend time (in seconds) between two animations, referenced by their names.

- `bool has_animation(String name) const`

Returns `true` if the `AnimationPlayer` stores an `Animation` with key `name`.

- `bool is_playing() const`

Returns `true` if playing an animation.

- `void play(String name="", float custom_blend=-1, float custom_speed=1.0, bool from_end=false)`

Play the animation with key `name`. Custom speed and blend times can be set. If custom speed is negative (-1), '`from_end`' being true can play the animation backwards.

- `void play_backwards(String name="", float custom_blend=-1)`

Play the animation with key `name` in reverse.

- `void queue(String name)`

Queue an animation for playback once the current one is done.

- `void remove_animation(String name)`

Remove the animation with key `name`.

- `void rename_animation(String name, String newname)`

Rename an existing animation with key `name` to `newname`.

- `void seek(float seconds, bool update=false)`

Seek the animation to the `seconds` point in time (in seconds). If `update` is `true`, the animation updates too, otherwise it updates at process time.

- `void set_blend_time(String anim_from, String anim_to, float sec)`

Specify a blend time (in seconds) between two animations, referenced by their names.

- `void stop(bool reset=true)`

Stop the currently playing animation. If `reset` is `true`, the anim position is reset to 0.

25.12 AnimationTreePlayer

Inherits: `Node < Object`

Category: Core

25.12.1 Brief Description

Animation Player that uses a node graph for blending Animations.

25.12.2 Member Functions

void	<i>add_node</i> (int type, String id)
void	<i>advance</i> (float delta)
<i>Animation</i>	<i>animation_node_get_animation</i> (String id) const
<i>String</i>	<i>animation_node_get_master_animation</i> (String id) const
void	<i>animation_node_set_animation</i> (String id, <i>Animation</i> animation)
void	<i>animation_node_set_filter_path</i> (String id, <i>NodePath</i> path, bool enable)
void	<i>animation_node_set_master_animation</i> (String id, String source)
<i>bool</i>	<i>are_nodes_connected</i> (String id, String dst_id, int dst_input_idx) const
<i>float</i>	<i>blend2_node_get_amount</i> (String id) const
void	<i>blend2_node_set_amount</i> (String id, float blend)
void	<i>blend2_node_set_filter_path</i> (String id, <i>NodePath</i> path, bool enable)
<i>float</i>	<i>blend3_node_get_amount</i> (String id) const
void	<i>blend3_node_set_amount</i> (String id, float blend)
<i>Vector2</i>	<i>blend4_node_get_amount</i> (String id) const
void	<i>blend4_node_set_amount</i> (String id, <i>Vector2</i> blend)
<i>int</i>	<i>connect_nodes</i> (String id, String dst_id, int dst_input_idx)
void	<i>disconnect_nodes</i> (String id, int dst_input_idx)
<i>PoolStringArray</i>	<i>get_node_list</i> ()
<i>float</i>	<i>mix_node_get_amount</i> (String id) const
void	<i>mix_node_set_amount</i> (String id, float ratio)
<i>bool</i>	<i>node_exists</i> (String node) const
<i>int</i>	<i>node_get_input_count</i> (String id) const
<i>String</i>	<i>node_get_input_source</i> (String id, int idx) const
<i>Vector2</i>	<i>node_get_position</i> (String id) const
<i>int</i>	<i>node_get_type</i> (String id) const
<i>int</i>	<i>node_rename</i> (String node, String new_name)
void	<i>node_set_position</i> (String id, <i>Vector2</i> screen_position)
<i>float</i>	<i>oneshot_node_get_autorestart_delay</i> (String id) const
<i>float</i>	<i>oneshot_node_get_autorestart_random_delay</i> (String id) const
<i>float</i>	<i>oneshot_node_get_fadein_time</i> (String id) const
<i>float</i>	<i>oneshot_node_get_fadeout_time</i> (String id) const
<i>bool</i>	<i>oneshot_node_has_autorestart</i> (String id) const
<i>bool</i>	<i>oneshot_node_is_active</i> (String id) const
void	<i>oneshot_node_set_autorestart</i> (String id, bool enable)
void	<i>oneshot_node_set_autorestart_delay</i> (String id, float delay_sec)
void	<i>oneshot_node_set_autorestart_random_delay</i> (String id, float rand_sec)
void	<i>oneshot_node_set_fadein_time</i> (String id, float time_sec)
void	<i>oneshot_node_set_fadeout_time</i> (String id, float time_sec)
void	<i>oneshot_node_set_filter_path</i> (String id, <i>NodePath</i> path, bool enable)
void	<i>oneshot_node_start</i> (String id)
void	<i>oneshot_node_stop</i> (String id)
void	<i>recompute_caches</i> ()
void	<i>remove_node</i> (String id)
void	<i>reset</i> ()
<i>float</i>	<i>timescale_node_get_scale</i> (String id) const
void	<i>timescale_node_set_scale</i> (String id, float scale)
void	<i>timeseek_node_seek</i> (String id, float seconds)
void	<i>transition_node_delete_input</i> (String id, int input_idx)

Continued on next page

Table 25.3 – continued from previous page

<i>int</i>	<i>transition_node_get_current (String id) const</i>
<i>int</i>	<i>transition_node_get_input_count (String id) const</i>
<i>float</i>	<i>transition_node_get_xfade_time (String id) const</i>
<i>bool</i>	<i>transition_node_has_input_auto_advance (String id, int input_idx) const</i>
<i>void</i>	<i>transition_node_set_current (String id, int input_idx)</i>
<i>void</i>	<i>transition_node_set_input_auto_advance (String id, int input_idx, bool enable)</i>
<i>void</i>	<i>transition_node_set_input_count (String id, int count)</i>
<i>void</i>	<i>transition_node_set_xfade_time (String id, float time_sec)</i>

25.12.3 Member Variables

- *bool active* - If true the AnimationTreePlayer is able to play animations. Default value: false.
- *NodePath base_path* - The node from which to relatively access other nodes. Default value: "...".

It accesses the Bones, so it should point to the same Node the AnimationPlayer would point its Root Node at.

- *NodePath master_player* - The path to the AnimationPlayer from which this AnimationTreePlayer binds animations to animation nodes.

Once set, Animation nodes can be added to the AnimationTreePlayer.

- *AnimationProcessMode playback_process_mode* - The thread in which to update animations. Default value: enum ANIMATION_PROCESS_IDLE.

25.12.4 Enums

enum AnimationProcessMode

- **ANIMATION_PROCESS_PHYSICS = 0** — Process animation during the physics process. This is especially useful when animating physics bodies.
- **ANIMATION_PROCESS_IDLE = 1** — Process animation during the idle process.

enum NodeType

- **NODE_OUTPUT = 0** — Output node.
- **NODE_ANIMATION = 1** — Animation node.
- **NODE_ONESHOT = 2** — OneShot node.
- **NODE_MIX = 3** — Mix node.
- **NODE_BLEND2 = 4** — Blend2 node.
- **NODE_BLEND3 = 5** — Blend3 node.
- **NODE_BLEND4 = 6** — Blend4 node.
- **NODE_TIMESCALE = 7** — TimeScale node.
- **NODE_TIMESEEK = 8** — TimeSeek node.

- **NODE_TRANSITION = 9** — Transition node.

25.12.5 Description

A node graph tool for blending multiple animations bound to an *AnimationPlayer*. Especially useful for animating characters or other skeleton-based rigs. It can combine several animations to form a desired pose.

It takes *Animations* from an *AnimationPlayer* node and mixes them depending on the graph.

25.12.6 Member Function Description

- `void add_node (int type, String id)`

Adds a `type` node to the graph with name `id`.

- `void advance (float delta)`

Shifts position in the animation timeline. Delta is the time in seconds to shift.

- `Animation animation_node_get_animation (String id) const`

Returns the *AnimationPlayer*'s *Animation* bound to the *AnimationTreePlayer*'s animation node with name `id`.

- `String animation_node_get_master_animation (String id) const`

Returns the name of the *master_player*'s *Animation* bound to this animation node.

- `void animation_node_set_animation (String id, Animation animation)`

Binds a new *Animation* from the *master_player* to the *AnimationTreePlayer*'s animation node with name `id`.

- `void animation_node_set_filter_path (String id, NodePath path, bool enable)`

If `enable` is `true`, the animation node with ID `id` turns off the track modifying the property at `path`. The modified node's children continue to animate.

- `void animation_node_set_master_animation (String id, String source)`

Binds the *Animation* named `source` from *master_player* to the animation node `id`. Recalculates caches.

- `bool are_nodes_connected (String id, String dst_id, int dst_input_idx) const`

Returns whether node `id` and `dst_id` are connected at the specified slot.

- `float blend2_node_get_amount (String id) const`

Returns the blend amount of a Blend2 node given its name.

- `void blend2_node_set_amount (String id, float blend)`

Sets the blend amount of a Blend2 node given its name and value.

A Blend2 Node blends two animations with the amount between 0 and 1.

At 0, Output is input a.

Towards 1, the influence of a gets lessened, the influence of b gets raised.

At 1, Output is input b.

- `void blend2_node_set_filter_path (String id, NodePath path, bool enable)`

If `enable` is `true`, the blend2 node with ID `id` turns off the track modifying the property at `path`. The modified node's children continue to animate.

- `float blend3_node_get_amount (String id) const`

Returns the blend amount of a Blend3 node given its name.

- `void blend3_node_set_amount (String id, float blend)`

Sets the blend amount of a Blend3 node given its name and value.

A Blend3 Node blends three animations with the amount between -1 and 1.

At -1, Output is input b-.

From -1 to 0, the influence of b- gets lessened, the influence of a gets raised and the influence of b+ is 0.

At 0, Output is input a.

From 0 to 1, the influence of a gets lessened, the influence of b+ gets raised and the influence of b+ is 0.

At 1, Output is input b+.

- `Vector2 blend4_node_get_amount (String id) const`

Returns the blend amount of a Blend4 node given its name.

- `void blend4_node_set_amount (String id, Vector2 blend)`

Sets the blend amount of a Blend4 node given its name and value.

A Blend4 Node blends two pairs of animations.

The two pairs are blended like blend2 and then added together.

- `int connect_nodes (String id, String dst_id, int dst_input_idx)`

Connects node `id` to `dst_id` at the specified input slot.

- `void disconnect_nodes (String id, int dst_input_idx)`

Disconnects nodes connected to `id` at the specified input slot.

- `PoolStringArray get_node_list ()`

Returns a `PoolStringArray` containing the name of all nodes.

- `float mix_node_get_amount (String id) const`

Returns mix amount of a Mix node given its name.

- `void mix_node_set_amount (String id, float ratio)`

Sets mix amount of a Mix node given its name and value.

A Mix node adds input b to input a by a the amount given by ratio.

- `bool node_exists (String node) const`

Check if a node exists (by name).

- `int node_get_input_count (String id) const`

Return the input count for a given node. Different types of nodes have different amount of inputs.

- `String node_get_input_source (String id, int idx) const`

Return the input source for a given node input.

- `Vector2 node_get_position (String id) const`

Returns position of a node in the graph given its name.

- `int node_get_type (String id) const`

Get the node type, will return from `NODE_*` enum.

- *int node_rename (String node, String new_name)*

Rename a node in the graph.

- *void node_set_position (String id, Vector2 screen_position)*

Sets position of a node in the graph given its name and position.

- *float oneshot_node_get_autorestart_delay (String id) const*

Returns autostart delay of a OneShot node given its name.

- *float oneshot_node_get_autorestart_random_delay (String id) const*

Returns autostart random delay of a OneShot node given its name.

- *float oneshot_node_get_fadein_time (String id) const*

Returns fade in time of a OneShot node given its name.

- *float oneshot_node_get_fadeout_time (String id) const*

Returns fade out time of a OneShot node given its name.

- *bool oneshot_node_has_autorestart (String id) const*

Returns whether a OneShot node will auto restart given its name.

- *bool oneshot_node_is_active (String id) const*

Returns whether a OneShot node is active given its name.

- *void oneshot_node_set_autorestart (String id, bool enable)*

Sets autorestart property of a OneShot node given its name and value.

- *void oneshot_node_set_autorestart_delay (String id, float delay_sec)*

Sets autorestart delay of a OneShot node given its name and value in seconds.

- *void oneshot_node_set_autorestart_random_delay (String id, float rand_sec)*

Sets autorestart random delay of a OneShot node given its name and value in seconds.

- *void oneshot_node_set_fadein_time (String id, float time_sec)*

Sets fade in time of a OneShot node given its name and value in seconds.

- *void oneshot_node_set_fadeout_time (String id, float time_sec)*

Sets fade out time of a OneShot node given its name and value in seconds.

- *void oneshot_node_set_filter_path (String id, NodePath path, bool enable)*

If `enable` is `true`, the oneshot node with ID `id` turns off the track modifying the property at `path`. The modified node's children continue to animate.

- *void oneshot_node_start (String id)*

Starts a OneShot node given its name.

- *void oneshot_node_stop (String id)*

Stops the OneShot node with name `id`.

- *void recompute_caches ()*

Manually recalculates the cache of track information generated from animation nodes. Needed when external sources modify the animation nodes' state.

- *void remove_node (String id)*

Removes the animation node with name `id`.

- `void reset()`

Resets this AnimationTreePlayer.

- `float timescale_node_get_scale(String id) const`

Returns time scale value of the TimeScale node with name `id`.

- `void timescale_node_set_scale(String id, float scale)`

Sets the time scale of the TimeScale node with name `id` to `scale`.

The timescale node is used to speed *Animations* up if the scale is above 1 or slow them down if it is below 1.

If applied after a blend or mix, affects all input animations to that blend or mix.

- `void timeseek_node_seek(String id, float seconds)`

Sets the time seek value of the TimeSeek node with name `id` to `seconds`

This functions as a seek in the *Animation* or the blend or mix of *Animations* input in it.

- `void transition_node_delete_input(String id, int input_idx)`

Deletes the input at `input_idx` for the transition node with name `id`.

- `int transition_node_get_current(String id) const`

Returns the index of the currently evaluated input for the transition node with name `id`.

- `int transition_node_get_input_count(String id) const`

Returns the number of inputs for the transition node with name `id`. You can add inputs by rightclicking on the transition node.

- `float transition_node_get_xfade_time(String id) const`

Returns the cross fade time for the transition node with name `id`.

- `bool transition_node_has_input_auto_advance(String id, int input_idx) const`

Returns `true` if the input at `input_idx` on transition node with name `id` is set to automatically advance to the next input upon completion.

- `void transition_node_set_current(String id, int input_idx)`

The transition node with name `id` sets its current input at `input_idx`.

- `void transition_node_set_input_auto_advance(String id, int input_idx, bool enable)`

The transition node with name `id` advances to its next input automatically when the input at `input_idx` completes.

- `void transition_node_set_input_count(String id, int count)`

Resizes the number of inputs available for the transition node with name `id`.

- `void transition_node_set_xfade_time(String id, float time_sec)`

The transition node with name `id` sets its cross fade time to `time_sec`.

25.13 Area

Inherits: *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

25.13.1 Brief Description

General purpose area node for detection and 3D physics influence.

25.13.2 Member Functions

<i>bool</i>	<i>get_collision_layer_bit (int bit) const</i>
<i>bool</i>	<i>get_collision_mask_bit (int bit) const</i>
<i>Array</i>	<i>get_overlapping_areas () const</i>
<i>Array</i>	<i>get_overlapping_bodies () const</i>
<i>bool</i>	<i>overlaps_area (Node area) const</i>
<i>bool</i>	<i>overlaps_body (Node body) const</i>
<i>void</i>	<i>set_collision_layer_bit (int bit, bool value)</i>
<i>void</i>	<i>set_collision_mask_bit (int bit, bool value)</i>

25.13.3 Signals

- **area_entered** (*Object* area)

Emitted when another area enters.

- **area_exited** (*Object* area)

Emitted when another area exits.

- **area_shape_entered** (*int* area_id, *Object* area, *int* area_shape, *int* self_shape)

Emitted when another area enters, reporting which areas overlapped.

- **area_shape_exited** (*int* area_id, *Object* area, *int* area_shape, *int* self_shape)

Emitted when another area exits, reporting which areas were overlapping.

- **body_entered** (*Object* body)

Emitted when a *PhysicsBody* object enters.

- **body_exited** (*Object* body)

Emitted when a *PhysicsBody2D* object exits.

- **body_shape_entered** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody2D* object enters, reporting which shapes overlapped.

- **body_shape_exited** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody2D* object exits, reporting which shapes were overlapping.

25.13.4 Member Variables

- **float angular_damp** - The rate at which objects stop spinning in this area. Represents the angular velocity lost per second. Values range from 0 (no damping) to 1 (full damping).
- **String audio_bus_name** - The name of the area's audio bus.
- **bool audio_bus_override** - If `true` the area's audio bus overrides the default audio bus. Default value: `false`.

- *int collision_layer* - The area's physics layer(s). Collidable objects can exist in any of 32 different layers. A contact is detected if object A is in any of the layers that object B scans, or object B is in any layers that object A scans. See also `collision_mask`.
- *int collision_mask* - The physics layers this area scans to determine collision detection.
- *float gravity* - The area's gravity intensity (ranges from -1024 to 1024). This value multiplies the gravity vector. This is useful to alter the force of gravity without altering its direction.
- *float gravity_distance_scale* - The falloff factor for point gravity. The greater the value, the faster gravity decreases with distance.
- *bool gravity_point* - If `true` gravity is calculated from a point (set via `gravity_vec`). Also see `space_override`. Default value: `false`.
- *Vector3 gravity_vec* - The area's gravity vector (not normalized). If gravity is a point (see `is_gravity_a_point`), this will be the point of attraction.
- *float linear_damp* - The rate at which objects stop moving in this area. Represents the linear velocity lost per second. Values range from 0 (no damping) to 1 (full damping).
- *bool monitorable* - If `true` other monitoring areas can detect this area. Default value: `true`.
- *bool monitoring* - If `true` the area detects bodies or areas entering and exiting it. Default value: `true`.
- *float priority* - The area's priority. Higher priority areas are processed first. Default value: 0.
- *float reverb_bus_amount* - The degree to which this area applies reverb to its associated audio. Ranges from 0 to 1 with 0.1 precision.
- *bool reverb_bus_enable* - If `true` the area applies reverb to its associated audio.
- *String reverb_bus_name* - The reverb bus name to use for this area's associated audio.
- *float reverb_bus_uniformity* - The degree to which this area's reverb is a uniform effect. Ranges from 0 to 1 with 0.1 precision.
- *SpaceOverride space_override* - Override mode for gravity and damping calculations within this area. See the `SPACE_OVERRIDE_*` constants for values.

25.13.5 Enums

enum SpaceOverride

- **SPACE_OVERRIDE_DISABLED = 0** — This area does not affect gravity/damping.
- **SPACE_OVERRIDE_COMBINE = 1** — This area adds its gravity/damping values to whatever has been calculated so far (in priority order).
- **SPACE_OVERRIDE_COMBINE_REPLACE = 2** — This area adds its gravity/damping values to whatever has been calculated so far (in priority order), ignoring any lower priority areas.
- **SPACE_OVERRIDE_REPLACE = 3** — This area replaces any gravity/damping, even the defaults, ignoring any lower priority areas.
- **SPACE_OVERRIDE_REPLACE_COMBINE = 4** — This area replaces any gravity/damping calculated so far (in priority order), but keeps calculating the rest of the areas.

25.13.6 Description

3D area that detects *CollisionObject* nodes overlapping, entering, or exiting. Can also alter or override local physics parameters (gravity, damping).

25.13.7 Member Function Description

- `bool get_collision_layer_bit (int bit) const`

Returns an individual bit on the layer mask.

- `bool get_collision_mask_bit (int bit) const`

Returns an individual bit on the collision mask.

- `Array get_overlapping_areas () const`

Returns a list of intersecting Areas. For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

- `Array get_overlapping_bodies () const`

Returns a list of intersecting *PhysicsBodys*. For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

- `bool overlaps_area (Node area) const`

If true the given area overlaps the Area. Note that the result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

- `bool overlaps_body (Node body) const`

If true the given body overlaps the Area. Note that the result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

- `void set_collision_layer_bit (int bit, bool value)`

Set/clear individual bits on the layer mask. This simplifies editing this “Area[code]’s layers.

- `void set_collision_mask_bit (int bit, bool value)`

Set/clear individual bits on the collision mask. This simplifies editing which Area layers this Area scans.

25.14 Area2D

Inherits: *CollisionObject2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.14.1 Brief Description

2D area for detection and 2D physics influence.

25.14.2 Member Functions

<i>bool</i>	<code>get_collision_layer_bit (int bit) const</code>
<i>bool</i>	<code>get_collision_mask_bit (int bit) const</code>
<i>Array</i>	<code>get_overlapping_areas () const</code>
<i>Array</i>	<code>get_overlapping_bodies () const</code>
<i>bool</i>	<code>overlaps_area (Node area) const</code>
<i>bool</i>	<code>overlaps_body (Node body) const</code>
<i>void</i>	<code>set_collision_layer_bit (int bit, bool value)</code>
<i>void</i>	<code>set_collision_mask_bit (int bit, bool value)</code>

25.14.3 Signals

- **area_entered** (*Object* area)

Emitted when another area enters.

- **area_exited** (*Object* area)

Emitted when another area exits.

- **area_shape_entered** (*int* area_id, *Object* area, *int* area_shape, *int* self_shape)

Emitted when another area enters, reporting which shapes overlapped.

- **area_shape_exited** (*int* area_id, *Object* area, *int* area_shape, *int* self_shape)

Emitted when another area exits, reporting which shapes were overlapping.

- **body_entered** (*Object* body)

Emitted when a *PhysicsBody2D* object enters.

- **body_exited** (*Object* body)

Emitted when a *PhysicsBody2D* object exits.

- **body_shape_entered** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody2D* object enters, reporting which shapes overlapped.

- **body_shape_exited** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody2D* object exits, reporting which shapes were overlapping.

25.14.4 Member Variables

- **float angular_damp** - The rate at which objects stop spinning in this area. Represents the angular velocity lost per second. Values range from 0 (no damping) to 1 (full damping).
- **String audio_bus_name** - The name of the area's audio bus.
- **bool audio_bus_override** - If `true` the area's audio bus overrides the default audio bus. Default value: `false`.
- **int collision_layer** - The area's physics layer(s). Collidable objects can exist in any of 32 different layers. A contact is detected if object A is in any of the layers that object B scans, or object B is in any layers that object A scans. See also `collision_mask`.
- **int collision_mask** - The physics layers this area scans to determine collision detection.

- *float gravity* - The area's gravity intensity (ranges from -1024 to 1024). This value multiplies the gravity vector. This is useful to alter the force of gravity without altering its direction.
- *float gravity_distance_scale* - The falloff factor for point gravity. The greater the value, the faster gravity decreases with distance.
- *bool gravity_point* - If true gravity is calculated from a point (set via `gravity_vec`). Also see `space_override`. Default value: false.
- *Vector2 gravity_vec* - The area's gravity vector (not normalized). If gravity is a point (see `is_gravity_a_point`), this will be the point of attraction.
- *float linear_damp* - The rate at which objects stop moving in this area. Represents the linear velocity lost per second. Values range from 0 (no damping) to 1 (full damping).
- *bool monitorable* - If true other monitoring areas can detect this area. Default value: true.
- *bool monitoring* - If true the area detects bodies or areas entering and exiting it. Default value: true.
- *float priority* - The area's priority. Higher priority areas are processed first. Default value: 0.
- *SpaceOverride space_override* - Override mode for gravity and damping calculations within this area. See the `SPACE_OVERRIDE_*` constants for values.

25.14.5 Enums

enum SpaceOverride

- **SPACE_OVERRIDE_DISABLED = 0** — This area does not affect gravity/damping.
- **SPACE_OVERRIDE_COMBINE = 1** — This area adds its gravity/damping values to whatever has been calculated so far (in priority order).
- **SPACE_OVERRIDE_COMBINE_REPLACE = 2** — This area adds its gravity/damping values to whatever has been calculated so far (in priority order), ignoring any lower priority areas.
- **SPACE_OVERRIDE_REPLACE = 3** — This area replaces any gravity/damping, even the defaults, ignoring any lower priority areas.
- **SPACE_OVERRIDE_REPLACE_COMBINE = 4** — This area replaces any gravity/damping calculated so far (in priority order), but keeps calculating the rest of the areas.

25.14.6 Description

2D area that detects `CollisionObject2D` nodes overlapping, entering, or exiting. Can also alter or override local physics parameters (gravity, damping).

25.14.7 Member Function Description

- *bool get_collision_layer_bit (int bit) const*

Return an individual bit on the layer mask. Describes whether other areas will collide with this one on the given layer.

- *bool get_collision_mask_bit (int bit) const*

Return an individual bit on the collision mask. Describes whether this area will collide with others on the given layer.

- *Array get_overlapping_areas () const*

Returns a list of intersecting *Area2Ds*. For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

- `Array get_overlapping_bodies () const`

Returns a list of intersecting *PhysicsBody2Ds*. For performance reasons (collisions are all processed at the same time) this list is modified once during the physics step, not immediately after objects are moved. Consider using signals instead.

- `bool overlaps_area (Node area) const`

If `true` the given area overlaps the *Area2D*. Note that the result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

- `bool overlaps_body (Node body) const`

If `true` the given body overlaps the *Area2D*. Note that the result of this test is not immediate after moving objects. For performance, list of overlaps is updated once per frame and before the physics step. Consider using signals instead.

- `void set_collision_layer_bit (int bit, bool value)`

Set/clear individual bits on the layer mask. This makes getting an area in/out of only one layer easier.

- `void set_collision_mask_bit (int bit, bool value)`

Set/clear individual bits on the collision mask. This makes selecting the areas scanned easier.

25.15 Array

Category: Built-In Types

25.15.1 Brief Description

Generic array datatype.

25.15.2 Member Functions

<code>Array</code>	<code>Array (PoolColorArray from)</code>
<code>Array</code>	<code>Array (PoolVector3Array from)</code>
<code>Array</code>	<code>Array (PoolVector2Array from)</code>
<code>Array</code>	<code>Array (PoolStringArray from)</code>
<code>Array</code>	<code>Array (PoolRealArray from)</code>
<code>Array</code>	<code>Array (PoolIntArray from)</code>
<code>Array</code>	<code>Array (PoolByteArray from)</code>
<code>void</code>	<code>append (var value)</code>
<code>var</code>	<code>back ()</code>
<code>int</code>	<code>bsearch (var value, bool before=True)</code>
<code>int</code>	<code>bsearch_custom (var value, Object obj, String func, bool before=True)</code>
<code>void</code>	<code>clear ()</code>
<code>int</code>	<code>count (var value)</code>
<code>Array</code>	<code>duplicate ()</code>
<code>bool</code>	<code>empty ()</code>
<code>void</code>	<code>erase (var value)</code>

Continued on next page

Table 25.4 – continued from previous page

<i>int</i>	<i>find</i> (var what, <i>int</i> from=0)
<i>int</i>	<i>find_last</i> (var value)
<i>var</i>	<i>front</i> ()
<i>bool</i>	<i>has</i> (var value)
<i>int</i>	<i>hash</i> ()
<i>void</i>	<i>insert</i> (<i>int</i> position, var value)
<i>void</i>	<i>invert</i> ()
<i>var</i>	<i>pop_back</i> ()
<i>var</i>	<i>pop_front</i> ()
<i>void</i>	<i>push_back</i> (var value)
<i>void</i>	<i>push_front</i> (var value)
<i>void</i>	<i>remove</i> (<i>int</i> position)
<i>void</i>	<i>resize</i> (<i>int</i> size)
<i>int</i>	<i>rfind</i> (var what, <i>int</i> from=-1)
<i>int</i>	<i>size</i> ()
<i>void</i>	<i>sort</i> ()
<i>void</i>	<i>sort_custom</i> (<i>Object</i> obj, <i>String</i> func)

25.15.3 Description

Generic array, contains several elements of any type, accessible by numerical index starting at 0. Negative indices can be used to count from the right, like in Python. Arrays are always passed by reference.

25.15.4 Member Function Description

- **Array Array** (*PoolColorArray* from)

Construct an array from a *PoolColorArray*.

- **Array Array** (*PoolVector3Array* from)

Construct an array from a *PoolVector3Array*.

- **Array Array** (*PoolVector2Array* from)

Construct an array from a *PoolVector2Array*.

- **Array Array** (*PoolStringArray* from)

Construct an array from a *PoolStringArray*.

- **Array Array** (*PoolRealArray* from)

Construct an array from a *PoolRealArray*.

- **Array Array** (*PoolIntArray* from)

Construct an array from a *PoolIntArray*.

- **Array Array** (*PoolByteArray* from)

Construct an array from a *PoolByteArray*.

- **void append** (var value)

Append an element at the end of the array (alias of *push_back*).

- **var back** ()

Returns the last element of the array if the array is not empty (size>0).

- *int bsearch (var value, bool before=True)*

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search. Optionally, a before specifier can be passed. If false, the returned index comes after all existing entries of the value in the array. Note that calling bsearch on an unsorted array results in unexpected behavior.

- *int bsearch_custom (var value, Object obj, String func, bool before=True)*

Finds the index of an existing value (or the insertion index that maintains sorting order, if the value is not yet present in the array) using binary search and a custom comparison method. Optionally, a before specifier can be passed. If false, the returned index comes after all existing entries of the value in the array. The custom method receives two arguments (an element from the array and the value searched for) and must return true if the first argument is less than the second, and return false otherwise. Note that calling bsearch on an unsorted array results in unexpected behavior.

- *void clear ()*

Clear the array (resize to 0).

- *int count (var value)*

Return the amount of times an element is in the array.

- *Array duplicate ()*

Returns a copy of this Array.

- *bool empty ()*

Return true if the array is empty (size==0).

- *void erase (var value)*

Remove the first occurrence of a value from the array.

- *int find (var what, int from=0)*

Searches the array for a value and returns its index or -1 if not found. Optionally, the initial search index can be passed.

- *int find_last (var value)*

Searches the array in reverse order for a value and returns its index or -1 if not found.

- *var front ()*

Returns the first element of the array if the array is not empty (size>0).

- *bool has (var value)*

Return true if the array contains given value.

```
[ "inside", 7 ].has("inside") == true
[ "inside", 7 ].has("outside") == false
[ "inside", 7 ].has(7) == true
[ "inside", 7 ].has("7") == false
```

- *int hash ()*

Return a hashed integer value representing the array contents.

- *void insert (int position, var value)*

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (pos==size()).

- *void invert ()*

Reverse the order of the elements in the array (so first element will now be the last) and return reference to the array.

- *var pop_back ()*

Remove the last element of the array.

- var **pop_front** ()

Remove the first element of the array.

- void **push_back** (var value)

Append an element at the end of the array.

- void **push_front** (var value)

Add an element at the beginning of the array.

- void **remove** (int position)

Remove an element from the array by index.

- void **resize** (int size)

Resize the array to contain a different number of elements. If the array size is smaller, elements are cleared, if bigger, new elements are Null.

- int **rfind** (var what, int from=-1)

Searches the array in reverse order. Optionally, a start search index can be passed. If negative, the start index is considered relative to the end of the array.

- int **size** ()

Return the amount of elements in the array.

- void **sort** ()

Sort the array using natural order and return reference to the array.

- void **sort_custom** (Object obj, String func)

Sort the array using a custom method and return reference to the array. The arguments are an object that holds the method and the name of such method. The custom method receives two arguments (a pair of elements from the array) and must return true if the first argument is less than the second, and return false otherwise. Note: you cannot randomize the return value as the heapsort algorithm expects a deterministic result. Doing so will result in unexpected behavior.

```
class MyCustomSorter:
    static func sort(a, b):
        if a[0] < b[0]:
            return true
        return false

var my_items = [[5, "Potato"], [9, "Rice"], [4, "Tomato"]]
my_items.sort_custom(MyCustomSorter, "sort")
```

25.16 ArrayMesh

Inherits: Mesh < Resource < Reference < Object

Category: Core

25.16.1 Brief Description

25.16.2 Member Functions

void	<code>add_blend_shape (String name)</code>
void	<code>add_surface_from_arrays (int primitive, Array arrays, Array blend_shapes=[], int compress_flags=97792)</code>
void	<code>center_geometry ()</code>
void	<code>clear_blend_shapes ()</code>
int	<code>get_blend_shape_count () const</code>
<i>String</i>	<code>get_blend_shape_name (int index) const</code>
int	<code>get_surface_count () const</code>
void	<code>regen_normalmaps ()</code>
int	<code>surface_get_array_index_len (int surf_idx) const</code>
int	<code>surface_get_array_len (int surf_idx) const</code>
<i>Array</i>	<code>surface_get_arrays (int surf_idx) const</code>
<i>Array</i>	<code>surface_get_blend_shape_arrays (int surf_idx) const</code>
int	<code>surface_get_format (int surf_idx) const</code>
<i>Material</i>	<code>surface_get_material (int surf_idx) const</code>
<i>String</i>	<code>surface_get_name (int surf_idx) const</code>
int	<code>surface_get_primitive_type (int surf_idx) const</code>
void	<code>surface_remove (int surf_idx)</code>
void	<code>surface_set_material (int surf_idx, Material material)</code>
void	<code>surface_set_name (int surf_idx, String name)</code>
void	<code>surface_update_region (int surf_idx, int offset, PoolByteArray data)</code>

25.16.3 Member Variables

- *BlendShapeMode* **blend_shape_mode**
- *AABB* **custom_aabb**

25.16.4 Numeric Constants

- **NO_INDEX_ARRAY = -1** — Default value used for index_array_len when no indices are present.
- **ARRAY_WEIGHTS_SIZE = 4** — Amount of weights/bone indices per vertex (always 4).

25.16.5 Enums

enum **ArrayFormat**

- **ARRAY_FORMAT_VERTEX = 1** — Array format will include vertices (mandatory).
- **ARRAY_FORMAT_NORMAL = 2** — Array format will include normals
- **ARRAY_FORMAT_TANGENT = 4** — Array format will include tangents

- **ARRAY_FORMAT_COLOR = 8** — Array format will include a color array.
- **ARRAY_FORMAT_TEX_UV = 16** — Array format will include UVs.
- **ARRAY_FORMAT_TEX_UV2 = 32** — Array format will include another set of UVs.
- **ARRAY_FORMAT_BONES = 64** — Array format will include bone indices.
- **ARRAY_FORMAT_WEIGHTS = 128** — Array format will include bone weights.
- **ARRAY_FORMAT_INDEX = 256** — Index array will be used.

enum **ArrayType**

- **ARRAY_VERTEX = 0** — Vertex array (array of *Vector3* vertices).
- **ARRAY_NORMAL = 1** — Normal array (array of *Vector3* normals).
- **ARRAY_TANGENT = 2** — Tangent array, array of groups of 4 floats. first 3 floats determine the tangent, and the last the binormal direction as -1 or 1.
- **ARRAY_COLOR = 3** — Vertex array (array of *Color* colors).
- **ARRAY_TEX_UV = 4** — UV array (array of *Vector3* UVs or float array of groups of 2 floats (u,v)).
- **ARRAY_TEX_UV2 = 5** — Second UV array (array of *Vector3* UVs or float array of groups of 2 floats (u,v)).
- **ARRAY_BONES = 6** — Array of bone indices, as a float array. Each element in groups of 4 floats.
- **ARRAY_WEIGHTS = 7** — Array of bone weights, as a float array. Each element in groups of 4 floats.
- **ARRAY_INDEX = 8** — Array of integers, used as indices referencing vertices. No index can be beyond the vertex array size.
- **ARRAY_MAX = 9**

25.16.6 Member Function Description

- void **add_blend_shape** (*String* name)
- void **add_surface_from_arrays** (*int* primitive, *Array* arrays, *Array* blend_shapes=[], *int* compress_flags=97792)

Create a new surface (*get_surface_count* that will become *surf_idx* for this).

Surfaces are created to be rendered using a “primitive”, which may be PRIMITIVE_POINTS, PRIMITIVE_LINES, PRIMITIVE_LINE_STRIP, PRIMITIVE_LINE_LOOP, PRIMITIVE_TRIANGLES, PRIMITIVE_TRIANGLE_STRIP, PRIMITIVE_TRIANGLE_FAN. (As a note, when using indices, it is recommended to only use just points, lines or triangles).

- void **center_geometry** ()
- void **clear_blend_shapes** ()
- *int* **get_blend_shape_count** () const
- *String* **get_blend_shape_name** (*int* index) const
- *int* **get_surface_count** () const

Return the amount of surfaces that the *ArrayMesh* holds.

- void **regen_normalmaps** ()
- *int* **surface_get_array_index_len** (*int* surf_idx) const

Return the length in indices of the index array in the requested surface (see *add_surface*).

- `int surface_get_array_len (int surf_idx) const`

Return the length in vertices of the vertex array in the requested surface (see `add_surface`).

- `Array surface_get_arrays (int surf_idx) const`
- `Array surface_get_blend_shape_arrays (int surf_idx) const`
- `int surface_get_format (int surf_idx) const`

Return the format mask of the requested surface (see `add_surface`).

- `Material surface_get_material (int surf_idx) const`

Return a *Material* in a given surface. Surface is rendered using this material.

- `String surface_get_name (int surf_idx) const`
- `int surface_get_primitive_type (int surf_idx) const`

Return the primitive type of the requested surface (see `add_surface`).

- `void surface_remove (int surf_idx)`

Remove a surface at position `surf_idx`, shifting greater surfaces one `surf_idx` slot down.

- `void surface_set_material (int surf_idx, Material material)`
- `void surface_set_name (int surf_idx, String name)`

Set a *Material* for a given surface. Surface will be rendered using this material.

- `void surface_update_region (int surf_idx, int offset, PoolByteArray data)`

25.17 ARVRAnchor

Inherits: *Spatial < Node < Object*

Category: Core

25.17.1 Brief Description

Anchor point in AR Space

25.17.2 Member Functions

<code>String</code>	<code>get_anchor_name () const</code>
<code>bool</code>	<code>get_is_active () const</code>
<code>Plane</code>	<code>get_plane () const</code>
<code>Vector3</code>	<code>get_size () const</code>

25.17.3 Member Variables

- *int anchor_id* - The anchor's id. You can set this before the anchor itself exists. The first anchor gets an id of 1, the second an id of 2, etc. When anchors get removed, the engine can then assign the corresponding id to new anchors. The most common situation where anchors 'disappear' is when the AR server identifies that two anchors represent different parts of the same plane and merges them.

25.17.4 Description

The ARVR Anchor point is a spatial node that maps a real world location identified by the AR platform to a position within the game world. For example, as long as plane detection in ARKit is on, ARKit will identify and update the position of planes (tables, floors, etc) and create anchors for them.

This node is mapped to one of the anchors through its unique id. When you receive a signal that a new anchor is available you should add this node to your scene for that anchor. You can predefine nodes and set the id and the nodes will simply remain on 0,0,0 until a plane is recognised.

Keep in mind that as long as plane detection is enable the size, placing and orientation of an anchor will be updates as the detection logic learns more about the real world out there especially if only part of the surface is in view.

25.17.5 Member Function Description

- *String get_anchor_name () const*

Returns the name given to this anchor.

- *bool get_is_active () const*

Returns true if the anchor is being tracked and false if no anchor with this id is currently known.

- *Plane get_plane () const*

Returns a plane aligned with our anchor, handy for intersection testing

- *Vector3 get_size () const*

Returns the estimated size of the plane that was detected. Say when the anchor relates to a table in the real world, this is the estimated size of the surface of that table.

25.18 ARVRCamera

Inherits: *Camera < Spatial < Node < Object*

Category: Core

25.18.1 Brief Description

A camera node with a few overrules for AR/VR applied such as location tracking.

25.18.2 Description

This is a helper spatial node for our camera, note that if stereoscopic rendering is applicable (VR-HMD) most of the camera properties are ignored as the HMD information overrides them. The only properties that can be trusted are the near and far planes.

The position and orientation of this node is automatically updated by the ARVR Server to represent the location of the HMD if such tracking is available and can thus be used by game logic. Note that in contrast to the ARVR Controller the render thread has access to the most up to date tracking data of the HMD and the location of the ARVRCamera can lag a few milliseconds behind what is used for rendering as a result.

25.19 ARVRController

Inherits: *Spatial < Node < Object*

Category: Core

25.19.1 Brief Description

A spatial node representing a spatially tracked controller.

25.19.2 Member Functions

<i>String</i>	<code>get_controller_name () const</code>
<i>int</i>	<code>get_hand () const</code>
<i>bool</i>	<code>get_is_active () const</code>
<i>float</i>	<code>get_joystick_axis (int axis) const</code>
<i>int</i>	<code>get_joystick_id () const</code>
<i>int</i>	<code>is_button_pressed (int button) const</code>

25.19.3 Signals

- **button_pressed (int button)**

Emitted when a button on this controller is pressed.

- **button_release (int button)**

Emitted when a button on this controller is released.

25.19.4 Member Variables

- *int controller_id* - The controller's id.

A controller id of 0 is unbound and will always result in an inactive node. Controller id 1 is reserved for the first controller that identifies itself as the left hand controller and id 2 is reserved for the first controller that identifies itself as the right hand controller.

For any other controller that the *ARVRServer* detects we continue with controller id 3.

When a controller is turned off, its slot is freed. This ensures controllers will keep the same id even when controllers with lower ids are turned off.

- *float rumble* - The degree to which the tracker rumbles. Ranges from 0.0 to 1.0 with precision .01. If changed, updates *ARVRPositionalTracker.rumble* accordingly.

25.19.5 Description

This is a helper spatial node that is linked to the tracking of controllers. It also offers several handy pass throughs to the state of buttons and such on the controllers.

Controllers are linked by their id. You can create controller nodes before the controllers are available. Say your game always uses two controllers (one for each hand) you can predefine the controllers with id 1 and 2 and they will become active as soon as the controllers are identified. If you expect additional controllers to be used you should react to the signals and add ARVRController nodes to your scene.

The position of the controller node is automatically updated by the ARVR Server. This makes this node ideal to add child nodes to visualise the controller.

25.19.6 Member Function Description

- `String get_controller_name () const`

If active, returns the name of the associated controller if provided by the AR/VR SDK used.

- `int get_hand () const`

Returns the hand holding this controller, if known. See TRACKER_* constants in *ARVRPositionalTracker*.

- `bool get_is_active () const`

Returns `true` if the bound controller is active. ARVR systems attempt to track active controllers.

- `float get_joystick_axis (int axis) const`

Returns the value of the given axis for things like triggers, touchpads, etc. that are embedded into the controller.

- `int get_joystick_id () const`

Returns the ID of the joystick object bound to this. Every controller tracked by the ARVR Server that has buttons and axis will also be registered as a joystick within Godot. This means that all the normal joystick tracking and input mapping will work for buttons and axis found on the AR/VR controllers. This ID is purely offered as information so you can link up the controller with its joystick entry.

- `int is_button_pressed (int button) const`

Returns `true` if the button at index `button` is pressed.

25.20 ARVRInterface

Inherits: Reference < Object

Inherited By: ARVRInterfaceGDNative, MobileVRInterface

Category: Core

25.20.1 Brief Description

Base class for ARVR interface implementation.

25.20.2 Member Functions

<i>int</i>	<code>get_capabilities () const</code>
<i>String</i>	<code>get_name () const</code>
<i>Vector2</i>	<code>get_render_targetsize ()</code>
<i>int</i>	<code>get_tracking_status () const</code>
<i>bool</i>	<code>initialize ()</code>
<i>bool</i>	<code>is_stereo ()</code>
<i>void</i>	<code>uninitialize ()</code>

25.20.3 Member Variables

- `bool ar_is_anchor_detection_enabled` - On an AR interface, is our anchor detection enabled?
- `bool interface_is_initialized` - Has this interface been initialized?
- `bool interface_is_primary` - Is this our primary interface?

25.20.4 Enums

enum Eyes

- **EYE_MONO = 0** — Mono output, this is mostly used internally when retrieving positioning information for our camera node or when stereoscopic rendering is not supported.
- **EYE_LEFT = 1** — Left eye output, this is mostly used internally when rendering the image for the left eye and obtaining positioning and projection information.
- **EYE_RIGHT = 2** — Right eye output, this is mostly used internally when rendering the image for the right eye and obtaining positioning and projection information.

enum Tracking_status

- **ARVR_NORMAL_TRACKING = 0** — Tracking is behaving as expected.
- **ARVR_EXCESSIVE_MOTION = 1** — Tracking is hindered by excessive motion, player is moving faster than tracking can keep up.
- **ARVR_INSUFFICIENT_FEATURES = 2** — Tracking is hindered by insufficient features, it's too dark (for camera based tracking), player is blocked, etc.
- **ARVR_UNKNOWN_TRACKING = 3** — We don't know the status of the tracking or this interface does not provide feedback.
- **ARVR_NOT_TRACKING = 4** — Tracking is not functional (camera not plugged in or obscured, lighthouses turned off, etc.)

enum Capabilities

- **ARVR_NONE = 0** — No ARVR capabilities.
- **ARVR_MONO = 1** — This interface can work with normal rendering output (non-HMD based AR).
- **ARVR_STEREO = 2** — This interface supports stereoscopic rendering.
- **ARVR_AR = 4** — This interface supports AR (video background and real world tracking).

- **ARVR_EXTERNAL = 8** — This interface outputs to an external device, if the main viewport is used the on screen output is an unmodified buffer of either the left or right eye (stretched if the viewport size is not changed to the same aspect ratio of `get_render_targetsize`). Using a separate viewport node frees up the main viewport for other purposes.

25.20.5 Description

This class needs to be implemented to make an AR or VR platform available to Godot and these should be implemented as C++ modules or GDNative modules (note that for GDNative the subclass `ARVRScriptInterface` should be used). Part of the interface is exposed to GDScript so you can detect, enable and configure an AR or VR platform.

Interfaces should be written in such a way that simply enabling them will give us a working setup. You can query the available interfaces through `ARVRServer`.

25.20.6 Member Function Description

- `int get_capabilities () const`

Returns a combination of flags providing information about the capabilities of this interface.

- `String get_name () const`

Returns the name of this interface (OpenVR, OpenHMD, ARKit, etc).

- `Vector2 get_render_targetsize ()`

Returns the resolution at which we should render our intermediate results before things like lens distortion are applied by the VR platform.

- `int get_tracking_status () const`

If supported, returns the status of our tracking. This will allow you to provide feedback to the user whether there are issues with positional tracking.

- `bool initialize ()`

Call this to initialize this interface. The first interface that is initialized is identified as the primary interface and it will be used for rendering output.

After initializing the interface you want to use you then need to enable the AR/VR mode of a viewport and rendering should commence.

Note that you must enable the AR/VR mode on the main viewport for any device that uses the main output of Godot such as for mobile VR.

If you do this for a platform that handles its own output (such as OpenVR) Godot will show just one eye without distortion on screen. Alternatively you can add a separate viewport node to your scene and enable AR/VR on that viewport and it will be used to output to the HMD leaving you free to do anything you like in the main window such as using a separate camera as a spectator camera or render out something completely different.

While currently not used you can activate additional interfaces, you may wish to do this if you want to track controllers from other platforms. However at this point in time only one interface can render to an HMD.

- `bool is_stereo ()`

Returns true if the current output of this interface is in stereo.

- `void uninitialized ()`

Turns the interface off.

25.21 ARVRInterfaceGDNative

Inherits: *ARVRInterface < Reference < Object*

Category: Core

25.21.1 Brief Description

GDNative wrapper for an ARVR interface

25.21.2 Description

This is a wrapper class for GDNative implementations of the ARVR interface. To use a GDNative ARVR interface simply instantiate this object and set your GDNative library containing the ARVR interface implementation.

25.22 ARVROrigin

Inherits: *Spatial < Node < Object*

Category: Core

25.22.1 Brief Description

Our origin point in AR/VR.

25.22.2 Member Variables

- `float world_scale` - Allows you to adjust the scale to your game's units. Most AR/VR platforms assume a scale of 1 game world unit = 1 meter in the real world.

Note that this method is a passthrough to the *ARVRServer* itself.

25.22.3 Description

This is a special node within the AR/VR system that maps the physical location of the center of our tracking space to the virtual location within our game world.

There should be only one of these nodes in your scene and you must have one. All the ARVRCamera, ARVRController and ARVRAuthority nodes should be direct children of this node for spatial tracking to work correctly.

It is the position of this node that you update when your character needs to move through your game world while we're not moving in the real world. Movement in the real world is always in relation to this origin point.

So say that your character is driving a car, the ARVROrigin node should be a child node of this car. If you implement a teleport system to move your character, you change the position of this node. Etc.

25.23 ARVRPositionalTracker

Inherits: *Object*

Category: Core

25.23.1 Brief Description

A tracked object

25.23.2 Member Functions

<i>int</i>	<code>get_hand()</code> const
<i>int</i>	<code>get_joy_id()</code> const
<i>String</i>	<code>get_name()</code> const
<i>Basis</i>	<code>get_orientation()</code> const
<i>Vector3</i>	<code>get_position()</code> const
<i>bool</i>	<code>get_tracks_orientation()</code> const
<i>bool</i>	<code>get_tracks_position()</code> const
<i>Transform</i>	<code>get_transform(bool adjust_by_reference_frame)</code> const
<i>int</i>	<code>get_type()</code> const

25.23.3 Member Variables

- *float rumble* - The degree to which the tracker rumbles. Ranges from 0.0 to 1.0 with precision .01.

25.23.4 Enums

enum **TrackerHand**

- **TRACKER_HAND_UNKNOWN = 0** — The hand this tracker is held in is unknown or not applicable.
- **TRACKER_LEFT_HAND = 1** — This tracker is the left hand controller.
- **TRACKER_RIGHT_HAND = 2** — This tracker is the right hand controller.

25.23.5 Description

An instance of this object represents a device that is tracked such as a controller or anchor point. HMDs aren't represented here as they are fully handled internally.

As controllers are turned on and the AR/VR interface detects them instances of this object are automatically added to this list of active tracking objects accessible through the ARVRSERVER

The ARVRCController and ARVRAncor both consume objects of this type and should be the objects you use in game. The positional trackers are just the under the hood objects that make this all work and are mostly exposed so GDNative based interfaces can interact with them.

25.23.6 Member Function Description

- `int get_hand() const`

Returns the hand holding this tracker, if known. See TRACKER_* constants.

- `int get_joy_id() const`

If this is a controller that is being tracked the controller will also be represented by a joystick entry with this id.

- `String get_name() const`

Returns the controller or anchor point's name if available.

- `Basis get_orientation() const`

Returns the controller's orientation matrix.

- `Vector3 get_position() const`

Returns the world-space controller position.

- `bool get_tracks_orientation() const`

Returns true if this device tracks orientation.

- `bool get_tracks_position() const`

Returns true if this device tracks position.

- `Transform get_transform(bool adjust_by_reference_frame) const`

Returns the transform combining this device's orientation and position.

- `int get_type() const`

Returns the tracker's type.

25.24 ARVRServer

Inherits: *Object*

Category: Core

25.24.1 Brief Description

This is our AR/VR Server.

25.24.2 Member Functions

void	<code>center_on_hmd(int rotation_mode, bool keep_height)</code>
<i>ARVRInterface</i>	<code>find_interface(String name) const</code>
<i>ARVRInterface</i>	<code>get_interface(int idx) const</code>
<i>int</i>	<code>get_interface_count() const</code>
<i>Array</i>	<code>get_interfaces() const</code>
<i>Transform</i>	<code>get_reference_frame() const</code>
<i>ARVRPositionalTracker</i>	<code>get_tracker(int idx) const</code>
<i>int</i>	<code>get_tracker_count() const</code>
void	<code>set_primary_interface(ARVRInterface interface)</code>

25.24.3 Signals

- **interface_added** (*String interface_name*)

Signal send when a new interface has been added.

- **interface_removed** (*String interface_name*)

Signal send when an interface is removed.

- **tracker_added** (*String tracker_name, int type, int id*)

Signal send when a new tracker has been added. If you don't use a fixed number of controllers or if you're using ARVRArchors for an AR solution it is important to react to this signal and add the appropriate ARVRController or ARVRAuthor node related to this new tracker.

- **tracker_removed** (*String tracker_name, int type, int id*)

Signal send when a tracker is removed, you should remove any ARVRController or ARVRAuthor points if applicable. This is not mandatory, the nodes simply become inactive and will be made active again when a new tracker becomes available (i.e. a new controller is switched on that takes the place of the previous one).

25.24.4 Member Variables

- *float world_scale* - Allows you to adjust the scale to your game's units. Most AR/VR platforms assume a scale of 1 game world unit = 1 meter in the real world.

25.24.5 Enums

enum RotationMode

- **RESET_FULL_ROTATION = 0** — Fully reset the orientation of the HMD. Regardless of what direction the user is looking to in the real world. The user will look dead ahead in the virtual world.
- **RESET_BUT_KEEP_TILT = 1** — Resets the orientation but keeps the tilt of the device. So if we're looking down, we keep looking down but heading will be reset.
- **DONT_RESET_ROTATION = 2** — Does not reset the orientation of the HMD, only the position of the player gets centered.

enum TrackerType

- **TRACKER_CONTROLLER = 1** — Our tracker tracks the location of a controller.
- **TRACKER_BASESTATION = 2** — Our tracker tracks the location of a base station.
- **TRACKER_ANCHOR = 4** — Our tracker tracks the location and size of an AR anchor.
- **TRACKER_ANY_KNOWN = 127** — Used internally to filter trackers of any known type.
- **TRACKER_UNKNOWN = 128** — Used internally if we haven't set the tracker type yet.
- **TRACKER_ANY = 255** — Used internally to select all trackers.

25.24.6 Description

The AR/VR Server is the heart of our AR/VR solution and handles all the processing.

25.24.7 Member Function Description

- void **center_on_hmd** (*int rotation_mode, bool keep_height*)

This is a really important function to understand correctly. AR and VR platforms all handle positioning slightly differently.

For platforms that do not offer spatial tracking our origin point (0,0,0) is the location of our HMD but you have little control over the direction the player is facing in the real world.

For platforms that do offer spatial tracking our origin point depends very much on the system. For OpenVR our origin point is usually the center of the tracking space, on the ground. For other platforms its often the location of the tracking camera.

This method allows you to center our tracker on the location of the HMD, it will take the current location of the HMD and use that to adjust all our tracking data in essence realigning the real world to your players current position in your game world.

For this method to produce usable results tracking information should be available and this often takes a few frames after starting your game.

You should call this method after a few seconds have passed, when the user requests a realignment of the display holding a designated button on a controller for a short period of time, and when implementing a teleport mechanism.

- *ARVRInterface* **find_interface** (*String name*) const

Find an interface by its name. Say that you're making a game that uses specific capabilities of an AR/VR platform you can find the interface for that platform by name and initialize it.

- *ARVRInterface* **get_interface** (*int idx*) const

Get the interface registered at a given index in our list of interfaces.

- *int* **get_interface_count** () const

Get the number of interfaces currently registered with the AR/VR server. If you're game supports multiple AR/VR platforms you can look through the available interface and either present the user with a selection or simply try an initialize each interface and use the first one that returns true.

- *Array* **get_interfaces** () const

Returns a list of available interfaces with both id and name of the interface.

- *Transform* **get_reference_frame** () const

Gets our reference frame transform, mostly used internally and exposed for GDNative build interfaces.

- *ARVRPositionalTracker* **get_tracker** (*int idx*) const

Get the positional tracker at the given ID.

- *int* **get_tracker_count** () const

Get the number of trackers currently registered.

- void **set_primary_interface** (*ARVRInterface interface*)

Changes the primary interface to the specified interface. Again mostly exposed for GDNative interfaces.

25.25 AStar

Inherits: Reference < Object

Category: Core

25.25.1 Brief Description

AStar class representation that uses vectors as edges.

25.25.2 Member Functions

<i>float</i>	<i>_compute_cost (int from_id, int to_id) virtual</i>
<i>float</i>	<i>_estimate_cost (int from_id, int to_id) virtual</i>
<i>void</i>	<i>add_point (int id, Vector3 position, float weight_scale=1.0)</i>
<i>bool</i>	<i>are_points_connected (int id, int to_id) const</i>
<i>void</i>	<i>clear ()</i>
<i>void</i>	<i>connect_points (int id, int to_id, bool bidirectional=true)</i>
<i>void</i>	<i>disconnect_points (int id, int to_id)</i>
<i>int</i>	<i>get_available_point_id () const</i>
<i>int</i>	<i>get_closest_point (Vector3 to_position) const</i>
<i>Vector3</i>	<i>get_closest_position_in_segment (Vector3 to_position) const</i>
<i>PoolIntArray</i>	<i>get_id_path (int from_id, int to_id)</i>
<i>PoolIntArray</i>	<i>get_point_connections (int id)</i>
<i>PoolVector3Array</i>	<i>get_point_path (int from_id, int to_id)</i>
<i>Vector3</i>	<i>get_point_position (int id) const</i>
<i>float</i>	<i>get_point_weight_scale (int id) const</i>
<i>Array</i>	<i>get_points ()</i>
<i>bool</i>	<i>has_point (int id) const</i>
<i>void</i>	<i>remove_point (int id)</i>
<i>void</i>	<i>set_point_position (int id, Vector3 position)</i>
<i>void</i>	<i>set_point_weight_scale (int id, float weight_scale)</i>

25.25.3 Description

A* (A star) is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently directed path between multiple points. It enjoys widespread use due to its performance and accuracy. Godot's A* implementation make use of vectors as points.

You must add points manually with *AStar:add_point* and create segments manually with *AStar:connect_points*. So you can test if there is a path between two points with the *AStar:are_points_connected* function, get the list of existing ids in the found path with *AStar:get_id_path*, or the points list with *AStar:get_point_path*.

25.25.4 Member Function Description

- *float _compute_cost (int from_id, int to_id) virtual*

Called when computing the cost between two connected points.

- *float _estimate_cost (int from_id, int to_id) virtual*

Called when estimating the cost between a point and the path's ending point.

- *void add_point (int id, Vector3 position, float weight_scale=1.0)*

Adds a new point at the given position with the given identifier. The algorithm prefers points with lower *weight_scale* to form a path. The *id* must be 0 or larger, and the *weight_scale* must be 1 or larger.

```
var as = AStar.new()

as.add_point(1, Vector3(1, 0, 0), 4) # Adds the point (1, 0, 0) with weight_scale=4 and
# id=1
```

If there already exists a point for the given id, its position and weight scale are updated to the given values.

- *bool are_points_connected (int id, int to_id) const*

Returns whether there is a connection/segment between the given points.

- *void clear ()*

Clears all the points and segments.

- *void connect_points (int id, int to_id, bool bidirectional=true)*

Creates a segment between the given points.

```
var as = AStar.new()

as.add_point(1, Vector3(1, 1, 0))
as.add_point(2, Vector3(0, 5, 0))

as.connect_points(1, 2, false) # If bidirectional=false it's only possible to go from
# point 1 to point 2
# and not from point 2 to point 1.
```

- *void disconnect_points (int id, int to_id)*

Deletes the segment between the given points.

- *int get_available_point_id () const*

Returns the next available point id with no point associated to it.

- *int get_closest_point (Vector3 to_position) const*

Returns the id of the closest point to `to_position`. Returns -1 if there are no points in the points pool.

- *Vector3 get_closest_position_in_segment (Vector3 to_position) const*

Returns the closest position to `to_position` that resides inside a segment between two connected points.

```
var as = AStar.new()

as.add_point(1, Vector3(0, 0, 0))
as.add_point(2, Vector3(0, 5, 0))

as.connect_points(1, 2)

var res = as.get_closest_position_in_segment(Vector3(3, 3, 0)) # returns (0, 3, 0)
```

The result is in the segment that goes from y=0 to y=5. It's the closest position in the segment to the given point.

- *PoolIntArray get_id_path (int from_id, int to_id)*

Returns an array with the ids of the points that form the path found by AStar between the given points. The array is ordered from the starting point to the ending point of the path.

```
var as = AStar.new()

as.add_point(1, Vector3(0, 0, 0))
```

```

as.add_point(2, Vector3(0,1,0), 1) # default weight is 1
as.add_point(3, Vector3(1,1,0))
as.add_point(4, Vector3(2,0,0))

as.connect_points(1, 2, false)
as.connect_points(2, 3, false)
as.connect_points(4, 3, false)
as.connect_points(1, 4, false)
as.connect_points(5, 4, false)

var res = as.get_id_path(1, 3) # returns [1, 2, 3]

```

If you change the 2nd point's weight to 3, then the result will be [1, 4, 3] instead, because now even though the distance is longer, it's "easier" to get through point 4 than through point 2.

- *PoolIntArray* **get_point_connections** (*int id*)

Returns an array with the ids of the points that form the connect with the given point.

```

var as = AStar.new()

as.add_point(1, Vector3(0,0,0))
as.add_point(2, Vector3(0,1,0))
as.add_point(3, Vector3(1,1,0))
as.add_point(4, Vector3(2,0,0))

as.connect_points(1, 2, true)
as.connect_points(1, 3, true)

var neighbors = as.get_point_connections(1) # returns [2, 3]

```

- *PoolVector3Array* **get_point_path** (*int from_id, int to_id*)

Returns an array with the points that are in the path found by AStar between the given points. The array is ordered from the starting point to the ending point of the path.

- *Vector3* **get_point_position** (*int id*) const

Returns the position of the point associated with the given id.

- *float* **get_point_weight_scale** (*int id*) const

Returns the weight scale of the point associated with the given id.

- *Array* **get_points** ()

Returns an array of all points.

- *bool* **has_point** (*int id*) const

Returns whether a point associated with the given id exists.

- *void* **remove_point** (*int id*)

Removes the point associated with the given id from the points pool.

- *void* **set_point_position** (*int id, Vector3 position*)

Sets the position for the point with the given id.

- *void* **set_point_weight_scale** (*int id, float weight_scale*)

Sets the `weight_scale` for the point with the given id.

25.26 AtlasTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.26.1 Brief Description

Packs multiple small textures in a single, bigger one. Helps to optimize video memory costs and render calls.

25.26.2 Member Variables

- *Texture atlas* - The texture that contains the atlas. Can be any *Texture* subtype.
- *bool filter_clip* - If `true` clips the area outside of the region to avoid bleeding of the surrounding texture pixels.
- *Rect2 margin* - The margin around the region. The *Rect2*'s ‘size’ parameter ('w' and 'h' in the editor) resizes the texture so it fits within the margin.
- *Rect2 region* - The *AtlasTexture*'s used region.

25.26.3 Description

Texture resource aimed at managing big textures files that pack multiple smaller textures. Consists of a *Texture*, a margin that defines the border width,

and a region that defines the actual area of the *AtlasTexture*.

25.27 AudioBusLayout

Inherits: *Resource < Reference < Object*

Category: Core

25.27.1 Brief Description

Stores information about the audiobusses.

25.27.2 Description

Stores position, muting, solo, bypass, effects, effect position, volume, and the connections between busses. See *AudioServer* for usage.

25.28 AudioEffect

Inherits: *Resource < Reference < Object*

Inherited By: *AudioEffectCompressor, AudioEffectLimiter, AudioEffectAmplify, AudioEffectFilter, AudioEffectDistortion, AudioEffectDelay, AudioEffectStereoEnhance, AudioEffectReverb, AudioEffectPanner, AudioEffectEQ, AudioEffectPitchShift, AudioEffectChorus, AudioEffectPhaser*

Category: Core

25.28.1 Brief Description

Audio Effect For Audio.

25.28.2 Description

Base resource for audio bus. Applies an audio effect on the bus that the resource is applied on.

25.29 AudioEffectAmplify

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.29.1 Brief Description

Adds a Amplify audio effect to an Audio bus.

Increases or decreases the volume of the selected audio bus.

25.29.2 Member Variables

- *float volume_db* - Amount of amplification. Positive values make the sound louder, negative values make it quieter. Value can range from -80 to 24. Default value: 0.

25.29.3 Description

Increases or decreases the volume being routed through the audio bus.

25.30 AudioEffectBandLimitFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.30.1 Brief Description

Adds a band limit filter to the Audio Bus.

25.30.2 Description

Limits the frequencies in a range around the cutoff_hz and allows frequencies outside of this range to pass.

25.31 AudioEffectBandPassFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.31.1 Brief Description

Adds a band pass filter to the Audio Bus.

25.31.2 Description

Attenuates the frequencies inside of a range around the cutoff_hz and cuts frequencies outside of this band.

25.32 AudioEffectChorus

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.32.1 Brief Description

Adds a chorus audio effect.

25.32.2 Member Variables

- *float dry* - The effect's raw signal.
- *float voice/1/cutoff_hz* - The voice's cutoff frequency.
- *float voice/1/delay_ms* - The voice's signal delay.
- *float voice/1/depth_ms* - The voice filter's depth.
- *float voice/1/level_db* - The voice's volume.
- *float voice/1/pan* - The voice's pan level.
- *float voice/1/rate_hz* - The voice's filter rate.
- *float voice/2/cutoff_hz* - The voice's cutoff frequency.

- *float voice/2/delay_ms* - The voice's signal delay.
- *float voice/2/depth_ms* - The voice filter's depth.
- *float voice/2/level_db* - The voice's volume.
- *float voice/2/pan* - The voice's pan level.
- *float voice/2/rate_hz* - The voice's filter rate.
- *float voice/3/cutoff_hz* - The voice's cutoff frequency.
- *float voice/3/delay_ms* - The voice's signal delay.
- *float voice/3/depth_ms* - The voice filter's depth.
- *float voice/3/level_db* - The voice's volume.
- *float voice/3/pan* - The voice's pan level.
- *float voice/3/rate_hz* - The voice's filter rate.
- *float voice/4/cutoff_hz* - The voice's cutoff frequency.
- *float voice/4/delay_ms* - The voice's signal delay.
- *float voice/4/depth_ms* - The voice filter's depth.
- *float voice/4/level_db* - The voice's volume.
- *float voice/4/pan* - The voice's pan level.
- *float voice/4/rate_hz* - The voice's filter rate.
- *int voice_count* - The amount of voices in the effect.
- *float wet* - The effect's processed signal.

25.32.3 Description

Adds a chorus audio effect. The effect applies a filter with voices to duplicate the audio source and manipulate it through the filter.

25.33 AudioEffectCompressor

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.33.1 Brief Description

Adds a Compressor audio effect to an Audio bus.

Reduces sounds that exceed a certain threshold level, smooths out the dynamics and increases the overall volume.

25.33.2 Member Variables

- *float attack_us* - Compressor's reaction time when the signal exceeds the threshold. Value can range from 20 to 2000. Default value: 20ms.
- *float gain* - Gain applied to the output signal.
- *float mix* - Balance between original signal and effect signal. Value can range from 0 (totally dry) to 1 (totally wet). Default value: 1.
- *float ratio* - Amount of compression applied to the audio once it passes the threshold level. The higher the ratio the more the loud parts of the audio will be compressed. Value can range from 1 to 48. Default value: 4.
- *float release_ms* - Compressor's delay time to stop reducing the signal after the signal level falls below the threshold. Value can range from 20 to 2000. Default value: 250ms.
- *String sidechain* - Reduce the sound level using another audio bus for threshold detection.
- *float threshold* - The level above which compression is applied to the audio. Value can range from -60 to 0. Default value: 0.

25.33.3 Description

Dynamic range compressor reduces the level of the sound when the amplitude goes over a certain threshold in Decibels. One of the main uses of a compressor is to increase the dynamic range by clipping as little as possible (when sound goes over 0dB).

Compressor has many uses in the mix:

- In the Master bus to compress the whole output (Although a *AudioEffectLimiter* is probably better)
- In voice channels to ensure they sound as balanced as possible.
- Sidechained. Sidechained, which can reduce the sound level sidechained with another audio bus for threshold detection.. This technique is very common in video game mixing to download the level of Music/SFX while voices are being heard.
- Accentuates transients by using a wider attack, making effects sound more punchy.

25.34 AudioEffectDelay

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.34.1 Brief Description

Adds a Delay audio effect to an Audio bus. Plays input signal back after a period of time.

Two tap delay and feedback options.

25.34.2 Member Variables

- *float dry* - Output percent of original sound. At 0, only delayed sounds are output. Value can range from 0 to 1. Default value: 1.
- *bool feedback/active* - If true feedback is enabled. Default value: false.
- *float feedback/delay_ms* - Feedback delay time in milliseconds. Default value: 340.
- *float feedback/level_db* - Sound level for tap1. Default value: -6 dB.
- *float feedback/lowpass* - Low-pass filter for feedback. Frequencies below the Low Cut value are filtered out of the source signal. Default value: 16000.
- *bool tap1/active* - If true, tap1 will be enabled. Default value: true.
- *float tap1/delay_ms* - Tap1 delay time in milliseconds. Default value: 250ms.
- *float tap1/level_db* - Sound level for tap1. Default value: -6 dB.
- *float tap1/pan* - Pan position for tap1. Value can range from -1 (fully left) to 1 (fully right). Default value: 0.2.
- *bool tap2/active* - If true, tap2 will be enabled. Default value: true.
- *float tap2/delay_ms* - Tap2 delay time in milliseconds. Default value: 500ms.
- *float tap2/level_db* - Sound level for tap2. Default value: -12 dB.
- *float tap2/pan* - Pan position for tap2. Value can range from -1 (fully left) to 1 (fully right). Default value: -0.4.

25.34.3 Description

Plays input signal back after a period of time. The delayed signal may be played back multiple times to create the sound of a repeating, decaying echo. Delay effects range from a subtle echo effect to a pronounced blending of previous sounds with new sounds.

25.35 AudioEffectDistortion

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.35.1 Brief Description

Adds a Distortion audio effect to an Audio bus.

Modify the sound to make it dirty.

25.35.2 Member Variables

- *float drive* - Distortion power. Value can range from 0 to 1. Default value: 0.

- *float keep_hf_hz* - High-pass filter. Frequencies higher than this value will not be affected by the distortion. Value can range from 1 to 20000. Default value: 16000.
- *Mode mode* - Distortion type. Default value: MODE_CLIP.
- *float post_gain* - Increases or decreases the volume after the effect. Value can range from -80 to 24. Default value: 0.
- *float pre_gain* - Increases or decreases the volume before the effect. Value can range from -60 to 60. Default value: 0.

25.35.3 Enums

enum Mode

- **MODE_CLIP = 0** — Digital distortion effect which cuts off peaks at the top and bottom of the waveform.
- **MODE_ATAN = 1**
- **MODE_LOFI = 2** — Low-resolution digital distortion effect. You can use it to emulate the sound of early digital audio devices.
- **MODE_OVERDRIVE = 3** — Emulates the warm distortion produced by a field effect transistor, which is commonly used in solid-state musical instrument amplifiers.
- **MODE_WAVESHAPE = 4** — Waveshaper distortions are used mainly by electronic musicians to achieve an extra-abrasive sound.

25.35.4 Description

Modify the sound and make it dirty. Different types are available : clip, tan, lofi (bit crushing), overdrive, or waveshape. By distorting the waveform the frequency content change, which will often make the sound “crunchy” or “abrasive”. For games, it can simulate sound coming from some saturated device or speaker very efficiently.

25.36 AudioEffectEQ

Inherits: *AudioEffect < Resource < Reference < Object*

Inherited By: *AudioEffectEQ6, AudioEffectEQ21, AudioEffectEQ10*

Category: Core

25.36.1 Brief Description

Base class for audio equalizers. Gives you control over frequencies.

Use it to create a custom equalizer if *AudioEffectEQ6*, *AudioEffectEQ10* or *AudioEffectEQ21* don't fit your needs.

25.36.2 Member Functions

<i>int</i>	<code>get_band_count () const</code>
<i>float</i>	<code>get_band_gain_db (int band_idx) const</code>
<i>void</i>	<code>set_band_gain_db (int band_idx, float volume_db)</code>

25.36.3 Description

AudioEffectEQ gives you control over frequencies. Use it to compensate for existing deficiencies in audio. AudioEffectEQ are very useful on the Master Bus to completely master a mix and give it character. They are also very useful when a game is run on a mobile device, to adjust the mix to that kind of speakers (it can be added but disabled when headphones are plugged).

25.36.4 Member Function Description

- *int get_band_count () const*

Returns the number of bands of the equalizer.

- *float get_band_gain_db (int band_idx) const*

Returns the band's gain at the specified index, in dB.

- *void set_band_gain_db (int band_idx, float volume_db)*

Sets band's gain at the specified index, in dB.

25.37 AudioEffectEQ10

Inherits: `AudioEffectEQ < AudioEffect < Resource < Reference < Object`

Category: Core

25.37.1 Brief Description

Adds a 10-band equalizer audio effect to an Audio bus. Gives you control over frequencies from 31 Hz to 16000 Hz.

Each frequency can be modulated between -60/+24 dB.

25.37.2 Description

Frequency bands :

Band 1 : 31 Hz

Band 2 : 62 Hz

Band 3 : 125 Hz

Band 4 : 250 Hz

Band 5 : 500 Hz

Band 6 : 1000 Hz

Band 7 : 2000 Hz

Band 8 : 4000 Hz

Band 9 : 8000 Hz

Band 10 : 16000 Hz

See also *AudioEffectEQ*, *AudioEffectEQ6*, *AudioEffectEQ21*.

25.38 AudioEffectEQ21

Inherits: *AudioEffectEQ < AudioEffect < Resource < Reference < Object*

Category: Core

25.38.1 Brief Description

Adds a 21-band equalizer audio effect to an Audio bus. Gives you control over frequencies from 22 Hz to 22000 Hz.

Each frequency can be modulated between -60/+24 dB.

25.38.2 Description

Frequency bands :

Band 1 : 22 Hz

Band 2 : 32 Hz

Band 3 : 44 Hz

Band 4 : 63 Hz

Band 5 : 90 Hz

Band 6 : 125 Hz

Band 7 : 175 Hz

Band 8 : 250 Hz

Band 9 : 350 Hz

Band 10 : 500 Hz

Band 11 : 700 Hz

Band 12 : 1000 Hz

Band 13 : 1400 Hz

Band 14 : 2000 Hz

Band 15 : 2800 Hz

Band 16 : 4000 Hz

Band 17 : 5600 Hz

Band 18 : 8000 Hz

Band 19 : 11000 Hz

Band 20 : 16000 Hz

Band 21 : 22000 Hz

See also *AudioEffectEQ*, *AudioEffectEQ6*, *AudioEffectEQ10*.

25.39 AudioEffectEQ6

Inherits: *AudioEffectEQ < AudioEffect < Resource < Reference < Object*

Category: Core

25.39.1 Brief Description

Adds a 6-band equalizer audio effect to an Audio bus. Gives you control over frequencies from 32 Hz to 10000 Hz.

Each frequency can be modulated between -60/+24 dB.

25.39.2 Description

Frequency bands :

Band 1 : 32 Hz

Band 2 : 100 Hz

Band 3 : 320 Hz

Band 4 : 1000 Hz

Band 5 : 3200 Hz

Band 6 : 10000 Hz

See also *AudioEffectEQ*, *AudioEffectEQ10*, *AudioEffectEQ21*.

25.40 AudioEffectFilter

Inherits: *AudioEffect < Resource < Reference < Object*

Inherited By: *AudioEffectHighPassFilter*, *AudioEffectBandLimitFilter*, *AudioEffectLowShelfFilter*, *AudioEffectHighShelfFilter*, *AudioEffectBandPassFilter*, *AudioEffectNotchFilter*, *AudioEffectLowPassFilter*

Category: Core

25.40.1 Brief Description

Adds a filter to the Audio Bus.

25.40.2 Member Variables

- *float cutoff_hz* - Threshold frequency for the filter.
- *FilterDB db*
- *float gain* - Gain amount of the frequencies after the filter.
- *float resonance* - Amount of boost in the overtones near the cutoff frequency.

25.40.3 Enums

enum **FilterDB**

- **FILTER_6DB = 0**
- **FILTER_12DB = 1**
- **FILTER_18DB = 2**
- **FILTER_24DB = 3**

25.40.4 Description

Allows frequencies other than the *cutoff_hz* to pass.

25.41 AudioEffectHighPassFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.41.1 Brief Description

Adds a high pass filter to the Audio Bus.

25.41.2 Description

Cuts frequencies lower than the *cutoff_hz* and allows higher frequencies to pass.

25.42 AudioEffectHighShelfFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.42.1 Brief Description

25.43 AudioEffectLimiter

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.43.1 Brief Description

Adds a soft clip Limiter audio effect to an Audio bus.

25.43.2 Member Variables

- *float ceiling_db* - The waveform's maximum allowed value. Value can range from -20 to -0.1. Default value: -0.1dB.
- *float soft_clip_db* - Applies a gain to the limited waves. Value can range from 0 to 6. Default value: 2dB.
- *float soft_clip_ratio*
- *float threshold_db* - Threshold from which the limiter begins to be active. Value can range from -30 to 0. Default value: 0dB.

25.43.3 Description

A limiter is similar to a compressor, but it's less flexible and designed to disallow sound going over a given dB threshold. Adding one in the Master Bus is always recommended to reduce the effects of clipping.

Soft clipping starts to reduce the peaks a little below the threshold level and progressively increases its effect as the input level increases such that the threshold is never exceeded.

25.44 AudioEffectLowPassFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.44.1 Brief Description

Adds a low pass filter to the Audio Bus.

25.44.2 Description

Cuts frequencies higher than the cutoff_hz and allows lower frequencies to pass.

25.45 AudioEffectLowShelfFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.45.1 Brief Description

25.46 AudioEffectNotchFilter

Inherits: *AudioEffectFilter < AudioEffect < Resource < Reference < Object*

Category: Core

25.46.1 Brief Description

Adds a notch filter to the Audio Bus.

25.46.2 Description

Attenuates frequencies in a narrow band around the cutoff_hz and cuts frequencies outside of this range.

25.47 AudioEffectPanner

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.47.1 Brief Description

Adds a Panner audio effect to an Audio bus. Pans sound left or right.

25.47.2 Member Variables

- *float pan* - Pan position. Value can range from -1 (fully left) to 1 (fully right).

25.47.3 Description

Determines how much of an audio signal is sent to the left and right buses.

25.48 AudioEffectPhaser

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.48.1 Brief Description

Adds a Phaser audio effect to an Audio bus.

Combines the original signal with a copy that is slightly out of phase with the original.

25.48.2 Member Variables

- *float depth* - Governs how high the filter frequencies sweep. Low value will primarily affect bass frequencies. High value can sweep high into the treble. Value can range from 0.1 to 4. Default value: 1.
- *float feedback* - Output percent of modified sound. Value can range from 0.1 to 0.9. Default value: 0.7.
- *float range_max_hz* - Determines the maximum frequency affected by the LFO modulations. Value can range from 10 to 10000. Default value: 1600hz.
- *float range_min_hz* - Determines the minimum frequency affected by the LFO modulations. Value can range from 10 to 10000. Default value: 440hz.
- *float rate_hz* - Adjusts the rate at which the effect sweeps up and down across the frequency range.

25.48.3 Description

Combines phase-shifted signals with the original signal. The movement of the phase-shifted signals is controlled using a Low Frequency Oscillator.

25.49 AudioEffectPitchShift

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.49.1 Brief Description

Adds a Pitch shift audio effect to an Audio bus.

Raises or lowers the pitch of original sound.

25.49.2 Member Variables

- *float pitch_scale* - Pitch value. Can range from 0 (-1 octave) to 16 (+16 octaves).

25.49.3 Description

Allows modulation of pitch independently of tempo. All frequencies can be increased/decreased with minimal effect on transients.

25.50 AudioEffectReverb

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.50.1 Brief Description

Adds a Reverb audio effect to an Audio bus.

Simulates the sound of acoustic environments such as rooms, concert halls, caverns, or an open spaces.

25.50.2 Member Variables

- *float damping* - Widens or narrows the stereo image of the reverb tail. 1 means fully widens. Value can range from 0 to 1. Default value: 1.
- *float dry* - Output percent of original sound. At 0, only modified sound is outputted. Value can range from 0 to 1. Default value: 1.
- *float hipass* - High-pass filter passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than the cutoff frequency. Value can range from 0 to 1. Default value: 0.
- *float predelay_feedback* - Output percent of predelay. Value can range from 0 to 1. Default value: 1.
- *float predelay_msec* - Time between the original signal and the early reflections of the reverb signal. Default value: 150ms.
- *float room_size* - Dimensions of simulated room. Bigger means more echoes. Value can range from 0 to 1. Default value: 0 . 8.
- *float spread* - Defines how reflective the imaginary room's walls are. Value can range from 0 to 1. Default value: 1.
- *float wet* - Output percent of modified sound. At 0, only original sound is outputted. Value can range from 0 to 1. Default value: 0 . 5.

25.50.3 Description

Simulates rooms of different sizes. Its parameters can be adjusted to simulate the sound of a specific room.

25.51 AudioEffectStereoEnhance

Inherits: *AudioEffect < Resource < Reference < Object*

Category: Core

25.51.1 Brief Description

25.51.2 Member Variables

- *float pan_pullout*
- *float surround*
- *float time_pullout_ms*

25.52 AudioServer

Inherits: *Object*

Category: Core

25.52.1 Brief Description

Server interface for low level audio access.

25.52.2 Member Functions

void	<i>add_bus (int at_position=-1)</i>
void	<i>add_bus_effect (int bus_idx, AudioEffect effect, int at_position=-1)</i>
<i>AudioBusLayout</i>	<i>generate_bus_layout () const</i>
<i>int</i>	<i>get_bus_count () const</i>
<i>AudioEffect</i>	<i>get_bus_effect (int bus_idx, int effect_idx)</i>
<i>int</i>	<i>get_bus_effect_count (int bus_idx)</i>
<i>int</i>	<i>get_bus_index (String bus_name) const</i>
<i>String</i>	<i>get_bus_name (int bus_idx) const</i>
<i>float</i>	<i>get_bus_peak_volume_left_db (int bus_idx, int channel) const</i>
<i>float</i>	<i>get_bus_peak_volume_right_db (int bus_idx, int channel) const</i>
<i>String</i>	<i>get_bus_send (int bus_idx) const</i>
<i>float</i>	<i>get_bus_volume_db (int bus_idx) const</i>
<i>float</i>	<i>get_mix_rate () const</i>
<i>int</i>	<i>get_speaker_mode () const</i>
<i>bool</i>	<i>is_bus_bypassing_effects (int bus_idx) const</i>
<i>bool</i>	<i>is_bus_effect_enabled (int bus_idx, int effect_idx) const</i>
<i>bool</i>	<i>is_bus_mute (int bus_idx) const</i>
<i>bool</i>	<i>is_bus_solo (int bus_idx) const</i>
void	<i>lock ()</i>
void	<i>move_bus (int index, int to_index)</i>
void	<i>remove_bus (int index)</i>
void	<i>remove_bus_effect (int bus_idx, int effect_idx)</i>
void	<i>set_bus_bypass_effects (int bus_idx, bool enable)</i>
void	<i>set_bus_count (int amount)</i>
void	<i>set_bus_effect_enabled (int bus_idx, int effect_idx, bool enabled)</i>
void	<i>set_bus_layout (AudioBusLayout bus_layout)</i>

Continued on next page

Table 25.5 – continued from previous page

void	<code>set_bus_mute (int bus_idx, bool enable)</code>
void	<code>set_bus_name (int bus_idx, String name)</code>
void	<code>set_bus_send (int bus_idx, String send)</code>
void	<code>set_bus_solo (int bus_idx, bool enable)</code>
void	<code>set_bus_volume_db (int bus_idx, float volume_db)</code>
void	<code>swap_bus_effects (int bus_idx, int effect_idx, int by_effect_idx)</code>
void	<code>unlock ()</code>

25.52.3 Signals

- **bus_layout_changed ()**

Emitted when the *AudioBusLayout* changes.

25.52.4 Enums

enum SpeakerMode

- **SPEAKER_MODE_STEREO = 0** — Two or fewer speakers are detected.
- **SPEAKER_SURROUND_51 = 2** — A 5.1 channel surround setup detected.
- **SPEAKER_SURROUND_71 = 3** — A 7.1 channel surround setup detected.

25.52.5 Description

AudioServer is a low level server interface for audio access. It is in charge of creating sample data (playable audio) as well as its playback via a voice interface.

25.52.6 Member Function Description

- void **add_bus (int at_position=-1)**

Adds a bus at *at_position*.

- void **add_bus_effect (int bus_idx, AudioEffect effect, int at_position=-1)**

Adds an *AudioEffect* effect to the bus *bus_idx* at *at_position*.

- *AudioBusLayout* **generate_bus_layout () const**

Generates an *AudioBusLayout* using the available busses and effects.

- *int* **get_bus_count () const**

Returns the number of available busses.

- *AudioEffect* **get_bus_effect (int bus_idx, int effect_idx)**

Returns the *AudioEffect* at position *effect_idx* in bus *bus_idx*.

- *int* **get_bus_effect_count (int bus_idx)**

Returns the number of effects on the bus at *bus_idx*.

- *int* **get_bus_index (String bus_name) const**

Returns the index of the bus with the name `bus_name`.

- `String get_bus_name (int bus_idx) const`

Returns the name of the bus with the index `bus_idx`.

- `float get_bus_peak_volume_left_db (int bus_idx, int channel) const`

Returns the peak volume of the left speaker at bus index `bus_idx` and channel index `channel`.

- `float get_bus_peak_volume_right_db (int bus_idx, int channel) const`

Returns the peak volume of the right speaker at bus index `bus_idx` and channel index `channel`.

- `String get_bus_send (int bus_idx) const`

Returns the name of the bus that the bus at index `bus_idx` sends to.

- `float get_bus_volume_db (int bus_idx) const`

Returns the volume of the bus at index `bus_idx` in dB.

- `float get_mix_rate () const`

Returns the sample rate at the output of the audioserver.

- `int get_speaker_mode () const`

Returns the speaker configuration.

- `bool is_bus_bypassing_effects (int bus_idx) const`

If true the bus at index `bus_idx` is bypassing effects.

- `bool is_bus_effect_enabled (int bus_idx, int effect_idx) const`

If true the effect at index `effect_idx` on the bus at index `bus_idx` is enabled.

- `bool is_bus_mute (int bus_idx) const`

If true the bus at index `bus_idx` is muted.

- `bool is_bus_solo (int bus_idx) const`

If true the bus at index `bus_idx` is in solo mode.

- `void lock ()`

Locks the audio drivers mainloop. Remember to unlock it afterwards.

- `void move_bus (int index, int to_index)`

Moves the bus from index `index` to index `to_index`.

- `void remove_bus (int index)`

Removes the bus at index `index`.

- `void remove_bus_effect (int bus_idx, int effect_idx)`

Removes the effect at index `effect_idx` from the bus at index `bus_idx`.

- `void set_bus_bypass_effects (int bus_idx, bool enable)`

If true the bus at index `bus_idx` is bypassing effects.

- `void set_bus_count (int amount)`

Adds and removes busses to make the number of busses match `amount`.

- `void set_bus_effect_enabled (int bus_idx, int effect_idx, bool enabled)`

If true the effect at index `effect_idx` on the bus at index `bus_idx` is enabled.

- `void set_bus_layout (AudioBusLayout bus_layout)`

Overwrites the currently used `AudioBusLayout`.

- `void set_bus_mute (int bus_idx, bool enable)`

If true the bus at index `bus_idx` is muted.

- `void set_bus_name (int bus_idx, String name)`

Sets the name of the bus at index `bus_idx` to `name`.

- `void set_bus_send (int bus_idx, String send)`

Connects the output of the bus at `bus_idx` to the bus named `send`.

- `void set_bus_solo (int bus_idx, bool enable)`

If true the bus at index `bus_idx` is in solo mode.

- `void set_bus_volume_db (int bus_idx, float volume_db)`

Sets the volume of the bus at index `bus_idx` to `volume_db`.

- `void swap_bus_effects (int bus_idx, int effect_idx, int by_effect_idx)`

Swaps the position of two effects in bus `bus_idx`.

- `void unlock ()`

Unlocks the audioplayer's main loop. After locking it always unlock it.

25.53 AudioStream

Inherits: `Resource < Reference < Object`

Inherited By: `AudioStreamSample, AudioStreamRandomPitch, AudioStreamOGGVorbis`

Category: Core

25.53.1 Brief Description

Base class for audio streams.

25.53.2 Description

Base class for audio streams. Audio streams are used for music playback, or other types of streamed sounds that don't fit or require more flexibility than a Sample.

25.54 AudioStreamOGGVorbis

Inherits: `AudioStream < Resource < Reference < Object`

Category: Core

25.54.1 Brief Description

OGG Vorbis audio stream driver.

25.54.2 Member Variables

- *bool loop*
- *float loop_offset*

25.54.3 Description

OGG Vorbis audio stream driver.

25.55 AudioStreamPlayback

Inherits: *Reference < Object*

Category: Core

25.55.1 Brief Description

Meta class for playing back audio.

25.55.2 Description

Can play, loop, pause a scroll through Audio. See *AudioStream* and *AudioStreamOGGVorbis* for usage.

25.56 AudioStreamPlayer

Inherits: *Node < Object*

Category: Core

25.56.1 Brief Description

Plays back audio.

25.56.2 Member Functions

<i>float</i>	<i>get_playback_position ()</i>
<i>void</i>	<i>play (float from_position=0.0)</i>
<i>void</i>	<i>seek (float to_position)</i>
<i>void</i>	<i>stop ()</i>

25.56.3 Signals

- **finished ()**

Emitted when the audio stops playing.

25.56.4 Member Variables

- *bool autoplay* - If true audio plays when added to scene tree. Default value: false.
- *String bus* - Bus on which this audio is playing.
- *MixTarget mix_target* - If the audio configuration has more than two speakers, this sets the target channels. See MIX_TARGET__* constants.
- *bool playing* - If true audio is playing.
- *AudioStream stream* - The *AudioStream* object to be played.
- *float volume_db* - Volume of sound, in dB.

25.56.5 Enums

enum MixTarget

- **MIX_TARGET_STEREO = 0** — The audio will be played only on the first channel.
- **MIX_TARGET_SURROUND = 1** — The audio will be played on all surround channels.
- **MIX_TARGET_CENTER = 2** — The audio will be played on the second channel, which is usually the center.

25.56.6 Description

Plays background audio.

25.56.7 Member Function Description

- *float get_playback_position ()*

Returns the position in the *AudioStream*.

- *void play (float from_position=0.0)*

Plays the audio from the given position ‘from_position’, in seconds.

- *void seek (float to_position)*

Sets the position from which audio will be played, in seconds.

- *void stop ()*

Stops the audio.

25.57 AudioStreamPlayer2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.57.1 Brief Description

Plays audio in 2D.

25.57.2 Member Functions

<i>float</i>	<i>get_playback_position ()</i>
<i>void</i>	<i>play (float from_position=0.0)</i>
<i>void</i>	<i>seek (float to_position)</i>
<i>void</i>	<i>stop ()</i>

25.57.3 Signals

- **finished ()**

Emitted when the audio stops playing.

25.57.4 Member Variables

- *int area_mask* - Areas in which this sound plays.
- *float attenuation* - Dampens audio over distance with this as an exponent.
- *bool autoplay* - If `true` audio plays when added to scene tree. Default value: `false`.
- *String bus* - Bus on which this audio is playing.
- *float max_distance* - Maximum distance from which audio is still hearable.
- *bool playing* - If `true` audio is playing.
- *AudioStream stream* - The *AudioStream* object to be played.
- *float volume_db* - Base volume without dampening.

25.57.5 Description

Plays audio that dampens with distance from screen center.

25.57.6 Member Function Description

- `float get_playback_position ()`

Returns the position in the *AudioStream*.

- `void play (float from_position=0.0)`

Plays the audio from the given position ‘from_position’, in seconds.

- `void seek (float to_position)`

Sets the position from which audio will be played, in seconds.

- `void stop ()`

Stops the audio.

25.58 AudioStreamPlayer3D

Inherits: *Spatial < Node < Object*

Category: Core

25.58.1 Brief Description

Plays 3D sound in 3D space

25.58.2 Member Functions

<code>float</code>	<code>get_playback_position ()</code>
<code>void</code>	<code>play (float from_position=0.0)</code>
<code>void</code>	<code>seek (float to_position)</code>
<code>void</code>	<code>stop ()</code>

25.58.3 Signals

- `finished ()`

Fires when the audio stops playing.

25.58.4 Member Variables

- `int area_mask` - Areas in which this sound plays.
- `float attenuation_filter_cutoff_hz` - Dampens audio above this frequency, in Hz.
- `float attenuation_filter_db` - Amount how much the filter affects the loudness, in dB.
- `AttenuationModel attenuation_model` - Decides if audio should get quieter with distance linearly, quadratically or logarithmically.
- `bool autoplay` - If `true` audio plays when added to scene tree. Default value: `false`.

- *String bus* - Bus on which this audio is playing.
- *DopplerTracking doppler_tracking* - Decides in which step the Doppler effect should be calculated.
- *float emission_angle_degrees* - The angle in which the audio reaches cameras undampened.
- *bool emission_angle_enabled* - If true the audio should be damped according to the direction of the sound.
- *float emission_angle_filter_attenuation_db* - dampens audio if camera is outside of ‘emission_angle_degrees’ and ‘emission_angle_enabled’ is set by this factor, in dB.
- *float max_db* - Sets the absolute maximum of the soundlevel, in dB.
- *float max_distance* - Sets the distance from which the ‘out_of_range_mode’ takes effect. Has no effect if set to 0.
- *OutOfRangeMode out_of_range_mode* - Decides if audio should pause when source is outside of ‘max_distance’ range.
- *bool playing* - If true, audio is playing.
- *AudioStream stream* - The *AudioStream* object to be played.
- *float unit_db* - Base sound level unaffected by dampening, in dB.
- *float unit_size* - Factor for the attenuation effect.

25.58.5 Enums

enum DopplerTracking

- **DOPPLER_TRACKING_DISABLED = 0** — Disables doppler tracking.
- **DOPPLER_TRACKING_IDLE_STEP = 1** — Executes doppler tracking in idle step.
- **DOPPLER_TRACKING_PHYSICS_STEP = 2** — Executes doppler tracking in physics step.

enum OutOfRangeMode

- **OUT_OF_RANGE_MIX = 0** — Mix this audio in, even when it’s out of range.
- **OUT_OF_RANGE_PAUSE = 1** — Pause this audio when it gets out of range.

enum AttenuationModel

- **ATTENUATION_INVERSE_DISTANCE = 0** — Linear dampening of loudness according to distance.
- **ATTENUATION_INVERSE_SQUARE_DISTANCE = 1** — Squared dampening of loudness according to distance.
- **ATTENUATION_LOGARITHMIC = 2** — Logarithmic dampening of loudness according to distance.

25.58.6 Description

Plays a sound effect with directed sound effects, dampens with distance if needed, generates effect of hearable position in space.

25.58.7 Member Function Description

- *float get_playback_position ()*

Returns the position in the *AudioStream*.

- **void play (float from_position=0.0)**

Plays the audio from the given position ‘from_position’, in seconds.

- **void seek (float to_position)**

Sets the position from which audio will be played, in seconds.

- **void stop ()**

Stops the audio.

25.59 AudioStreamRandomPitch

Inherits: *AudioStream < Resource < Reference < Object*

Category: Core

25.59.1 Brief Description

Plays audio with random pitch tweaking.

25.59.2 Member Variables

- *AudioStream audio_stream* - The current *AudioStream*.
- *float random_pitch* - The intensity of random pitch variation.

25.59.3 Description

Randomly varies pitch on each start.

25.60 AudioStreamSample

Inherits: *AudioStream < Resource < Reference < Object*

Category: Core

25.60.1 Brief Description

Plays audio.

25.60.2 Member Variables

- *Format format* - Audio format. See FORMAT_* constants for values.
- *int loop_begin* - Loop start in bytes.
- *int loop_end* - Loop end in bytes.
- *LoopMode loop_mode* - Loop mode. See LOOP_* constants for values.
- *int mix_rate* - The sample rate for mixing this audio.
- *bool stereo* - If true, audio is stereo. Default value: false.

25.60.3 Enums

enum LoopMode

- **LOOP_DISABLED = 0** — Audio does not loop.
- **LOOP_FORWARD = 1** — Audio loops the data between loop_begin and loop_end playing forward only.
- **LOOP_PING_PONG = 2** — Audio loops the data between loop_begin and loop_end playing back and forth.

enum Format

- **FORMAT_8_BITS = 0** — Audio codec 8 bit.
- **FORMAT_16_BITS = 1** — Audio codec 16 bit.
- **FORMAT_IMA_ADPCM = 2** — Audio codec IMA ADPCM.

25.60.4 Description

Plays audio, can loop.

25.61 BackBufferCopy

Inherits: Node2D < CanvasItem < Node < Object

Category: Core

25.61.1 Brief Description

Copies a region of the screen (or the whole screen) to a buffer so it can be accessed with the texscreen() shader instruction.

25.61.2 Member Variables

- *CopyMode copy_mode* - Buffer mode. See COPY_MODE__* constants.
- *Rect2 rect* - The area covered by the BackBufferCopy. Only used if `copy_mode` is COPY_MODE_RECT.

25.61.3 Enums

enum CopyMode

- **COPY_MODE_DISABLED = 0** — Disables the buffering mode. This means the BackBufferCopy node will directly use the portion of screen it covers.
- **COPY_MODE_RECT = 1** — BackBufferCopy buffers a rectangular region.
- **COPY_MODE_VIEWPORT = 2** — BackBufferCopy buffers the entire screen.

25.61.4 Description

Node for back-buffering the currently displayed screen. The region defined in the BackBufferCopy node is bufferized with the content of the screen it covers, or the entire screen according to the copy mode set. Accessing this buffer is done with the `texscreen()` shader instruction.

25.62 BakedLightmap

Inherits: *VisualInstance < Spatial < Node < Object*

Category: Core

25.62.1 Brief Description

25.62.2 Member Functions

<i>int</i>	<i>bake (Node from_node=null, bool create_visual_debug=false)</i>
<i>void</i>	<i>debug_bake ()</i>

25.62.3 Member Variables

- *float bake_cell_size*
- *float bake_energy*
- *Vector3 bake_extents*
- *bool bake_hdr*
- *BakeMode bake_mode*

- *float* **bake_propagation**
- *BakeQuality* **bake_quality**
- *float* **capture_cell_size**
- *String* **image_path**
- *BakedLightmapData* **light_data**

25.62.4 Enums

enum **BakeQuality**

- **BAKE_QUALITY_LOW** = 0
- **BAKE_QUALITY_MEDIUM** = 1
- **BAKE_QUALITY_HIGH** = 2

enum **BakeError**

- **BAKE_ERROR_OK** = 0
- **BAKE_ERROR_NO_SAVE_PATH** = 1
- **BAKE_ERROR_NO_MESHES** = 2
- **BAKE_ERROR_CANT_CREATE_IMAGE** = 3
- **BAKE_ERROR_USER_ABORTED** = 4

enum **BakeMode**

- **BAKE_MODE_CONE_TRACE** = 0
- **BAKE_MODE_RAY_TRACE** = 1

25.62.5 Member Function Description

- *int* **bake** (*Node* from_node=null, *bool* create_visual_debug=false)
- *void* **debug_bake** ()

25.63 BakedLightmapData

Inherits: *Resource < Reference < Object*

Category: Core

25.63.1 Brief Description

25.63.2 Member Functions

void	<i>add_user</i> (<i>NodePath</i> path, <i>Texture</i> lightmap, <i>int</i> instance)
void	<i>clear_users</i> ()
<i>int</i>	<i>get_user_count</i> () const
<i>Texture</i>	<i>get_user_lightmap</i> (<i>int</i> user_idx) const
<i>NodePath</i>	<i>get_user_path</i> (<i>int</i> user_idx) const

25.63.3 Member Variables

- *AABB bounds*
- *Transform cell_space_transform*
- *int cell_subdiv*
- *float energy*
- *PoolByteArray octree*

25.63.4 Member Function Description

- void **add_user** (*NodePath* path, *Texture* lightmap, *int* instance)
- void **clear_users** ()
- *int get_user_count* () const
- *Texture get_user_lightmap* (*int* user_idx) const
- *NodePath get_user_path* (*int* user_idx) const

25.64 BaseButton

Inherits: *Control < CanvasItem < Node < Object*

Inherited By: *LinkButton, TextureButton, Button*

Category: Core

25.64.1 Brief Description

Base class for different kinds of buttons.

25.64.2 Member Functions

void	<code>_pressed ()</code> virtual
void	<code>_toggled (bool button_pressed)</code> virtual
int	<code>get_draw_mode ()</code> const
bool	<code>is_hovered ()</code> const

25.64.3 Signals

- `button_down ()`

Emitted when the button starts being held down.

- `button_up ()`

Emitted when the button stops being held down.

- `pressed ()`

This signal is emitted every time the button is toggled or pressed (i.e. activated, so on `button_down` if “Click on press” is active and on `button_up` otherwise).

- `toggled (bool button_pressed)`

This signal is emitted when the button was just toggled between pressed and normal states (only if `toggle_mode` is active). The new state is contained in the `pressed` argument.

25.64.4 Member Variables

- *ActionMode* `action_mode` - Determines when the button is considered clicked, one of the ACTION_MODE_* constants.
- `bool disabled` - If `true` the button is in disabled state and can't be clicked or toggled.
- *FocusMode* `enabled_focus_mode` - Focus access mode to use when switching between enabled/disabled (see `Control.set_focus_mode` and `disabled`).
- *ButtonGroup* `group` - `ButtonGroup` associated to the button.
- `bool pressed` - If `true` the button's state is pressed. Means the button is pressed down or toggled (if `toggle_mode` is active).
- *ShortCut* `shortcut` - Shortcut associated to the button.
- `bool toggle_mode` - If `true` the button is in toggle mode. Makes the button flip state between pressed and unpressed each time its area is clicked.

25.64.5 Enums

enum ActionMode

- `ACTION_MODE_BUTTON_PRESS = 0` — Require just a press to consider the button clicked.
- `ACTION_MODE_BUTTON_RELEASE = 1` — Require a press and a subsequent release before considering the button clicked.

enum DrawMode

- **DRAW_NORMAL = 0** — The normal state (i.e. not pressed, not hovered, not toggled and enabled) of buttons.
- **DRAW_PRESSED = 1** — The state of buttons are pressed.
- **DRAW_HOVER = 2** — The state of buttons are hovered.
- **DRAW_DISABLED = 3** — The state of buttons are disabled.

25.64.6 Description

BaseButton is the abstract base class for buttons, so it shouldn't be used directly (it doesn't display anything). Other types of buttons inherit from it.

25.64.7 Member Function Description

- `void _pressed() virtual`

Called when button is pressed.

- `void _toggled(bool button_pressed) virtual`

Called when button is toggled (only if `toggle_mode` is active).

- `int get_draw_mode() const`

Return the visual state used to draw the button. This is useful mainly when implementing your own draw code by either overriding `_draw()` or connecting to "draw" signal. The visual state of the button is defined by the `DRAW_*` enum.

- `bool is_hovered() const`

Return true if mouse entered the button before it exit.

25.65 Basis

Category: Built-In Types

25.65.1 Brief Description

3x3 matrix datatype.

25.65.2 Member Functions

<i>Basis</i>	<code>Basis (Quat from)</code>
<i>Basis</i>	<code>Basis (Vector3 from)</code>
<i>Basis</i>	<code>Basis (Vector3 axis, float phi)</code>
<i>Basis</i>	<code>Basis (Vector3 x_axis, Vector3 y_axis, Vector3 z_axis)</code>
<i>float</i>	<code>determinant ()</code>
<i>Vector3</i>	<code>get_euler ()</code>
<i>int</i>	<code>get_orthogonal_index ()</code>
<i>Vector3</i>	<code>get_scale ()</code>
<i>Basis</i>	<code>inverse ()</code>
<i>Basis</i>	<code>orthonormalized ()</code>
<i>Basis</i>	<code>rotated (Vector3 axis, float phi)</code>
<i>Basis</i>	<code>scaled (Vector3 scale)</code>
<i>float</i>	<code>tdotx (Vector3 with)</code>
<i>float</i>	<code>tdoty (Vector3 with)</code>
<i>float</i>	<code>tdotz (Vector3 with)</code>
<i>Basis</i>	<code>transposed ()</code>
<i>Vector3</i>	<code>xform (Vector3 v)</code>
<i>Vector3</i>	<code>xform_inv (Vector3 v)</code>

25.65.3 Member Variables

- *Vector3 x* - The basis matrix's x vector.
- *Vector3 y* - The basis matrix's y vector.
- *Vector3 z* - The basis matrix's z vector.

25.65.4 Description

3x3 matrix used for 3D rotation and scale. Contains 3 vector fields x,y and z as its columns, which can be interpreted as the local basis vectors of a transformation. Can also be accessed as array of 3D vectors. These vectors are orthogonal to each other, but are not necessarily normalized. Almost always used as orthogonal basis for a *Transform*.

For such use, it is composed of a scaling and a rotation matrix, in that order (M = R.S).

25.65.5 Member Function Description

- **Basis Basis (Quat from)**

Create a rotation matrix from the given quaternion.

- **Basis Basis (Vector3 from)**

Create a rotation matrix (in the YXZ convention: first Z, then X, and Y last) from the specified Euler angles, given in the vector format as (X-angle, Y-angle, Z-angle).

- **Basis Basis (Vector3 axis, float phi)**

Create a rotation matrix which rotates around the given axis by the specified angle, in radians. The axis must be a normalized vector.

- *Basis* **Basis** (*Vector3* x_axis, *Vector3* y_axis, *Vector3* z_axis)

Create a matrix from 3 axis vectors.

- *float* **determinant** ()

Return the determinant of the matrix.

- *Vector3* **get_euler** ()

Assuming that the matrix is a proper rotation matrix (orthonormal matrix with determinant +1), return Euler angles (in the YXZ convention: first Z, then X, and Y last). Returned vector contains the rotation angles in the format (X-angle, Y-angle, Z-angle).

- *int* **get_orthogonal_index** ()

This function considers a discretization of rotations into 24 points on unit sphere, lying along the vectors (x,y,z) with each component being either -1,0 or 1, and returns the index of the point best representing the orientation of the object. It is mainly used by the grid map editor. For further details, refer to Godot source code.

- *Vector3* **get_scale** ()

Assuming that the matrix is the combination of a rotation and scaling, return the absolute value of scaling factors along each axis.

- *Basis* **inverse** ()

Return the inverse of the matrix.

- *Basis* **orthonormalized** ()

Return the orthonormalized version of the matrix (useful to call from time to time to avoid rounding error for orthogonal matrices). This performs a Gram-Schmidt orthonormalization on the basis of the matrix.

- *Basis* **rotated** (*Vector3* axis, *float* phi)

Introduce an additional rotation around the given axis by phi (radians). Only relevant when the matrix is being used as a part of *Transform*. The axis must be a normalized vector.

- *Basis* **scaled** (*Vector3* scale)

Introduce an additional scaling specified by the given 3D scaling factor. Only relevant when the matrix is being used as a part of *Transform*.

- *float* **tdotx** (*Vector3* with)

Transposed dot product with the x axis of the matrix.

- *float* **tdoty** (*Vector3* with)

Transposed dot product with the y axis of the matrix.

- *float* **tdotz** (*Vector3* with)

Transposed dot product with the z axis of the matrix.

- *Basis* **transposed** ()

Return the transposed version of the matrix.

- *Vector3* **xform** (*Vector3* v)

Return a vector transformed (multiplied) by the matrix.

- *Vector3* **xform_inv** (*Vector3* v)

Return a vector transformed (multiplied) by the transposed matrix. Note that this results in a multiplication by the inverse of the matrix only if it represents a rotation-reflection.

25.66 BitMap

Inherits: *Resource < Reference < Object*

Category: Core

25.66.1 Brief Description

Boolean matrix.

25.66.2 Member Functions

void	<i>create (Vector2 size)</i>
void	<i>create_from_image_alpha (Image image)</i>
bool	<i>get_bit (Vector2 position) const</i>
Vector2	<i>get_size () const</i>
int	<i>get_true_bit_count () const</i>
void	<i>set_bit (Vector2 position, bool bit)</i>
void	<i>set_bit_rect (Rect2 p_rect, bool bit)</i>

25.66.3 Description

A two-dimensional array of boolean values, can be used to efficiently store a binary matrix (every matrix element takes only one bit) and query the values using natural cartesian coordinates.

25.66.4 Member Function Description

- void **create** (*Vector2 size*)

Creates a bitmap with the specified size, filled with false.

- void **create_from_image_alpha** (*Image image*)

Creates a bitmap that matches the given image dimensions, every element of the bitmap is set to false if the alpha value of the image at that position is 0, and true in other case.

- bool **get_bit** (*Vector2 position*) const

Returns bitmap's value at the specified position.

- Vector2 **get_size** () const

Returns bitmap's dimensions.

- int **get_true_bit_count** () const

Returns the amount of bitmap elements that are set to true.

- void **set_bit** (*Vector2 position, bool bit*)

Sets the bitmap's element at the specified position, to the specified value.

- void **set_bit_rect** (*Rect2 p_rect, bool bit*)

Sets a rectangular portion of the bitmap to the specified value.

25.67 BitmapFont

Inherits: *Font < Resource < Reference < Object*

Category: Core

25.67.1 Brief Description

Renders text using `*.fnt` fonts.

25.67.2 Member Functions

void	<code>add_char (int character, int texture, Rect2 rect, Vector2 align=Vector2(0, 0), float advance=-1)</code>
void	<code>add_kerning_pair (int char_a, int char_b, int kerning)</code>
void	<code>add_texture (Texture texture)</code>
void	<code>clear ()</code>
int	<code>create_from_fnt (String path)</code>
Vector2	<code>get_char_size (int char, int next=0) const</code>
int	<code>get_kerning_pair (int char_a, int char_b) const</code>
Texture	<code>get_texture (int idx) const</code>
int	<code>get_texture_count () const</code>

25.67.3 Member Variables

- *float ascent* - Ascent (number of pixels above the baseline).
- *bool distance_field* - If `true` distance field hint is enabled.
- *BitmapFont fallback* - The fallback font.
- *float height* - Total font height (ascent plus descent) in pixels.

25.67.4 Description

Renders text using `*.fnt` fonts containing texture atlases. Supports distance fields. For using vector font files like TTF directly, see *DynamicFont*.

25.67.5 Member Function Description

- void **add_char** (*int character, int texture, Rect2 rect, Vector2 align=Vector2(0, 0), float advance=-1*)

Adds a character to the font, where `character` is the unicode value, `texture` is the texture index, `rect` is the region in the texture (in pixels!), `align` is the (optional) alignment for the character and `advance` is the (optional) `advance`.

- void **add_kerning_pair** (*int char_a, int char_b, int kerning*)

Adds a kerning pair to the `BitmapFont` as a difference. Kerning pairs are special cases where a typeface advance is determined by the next character.

- void **add_texture** (*Texture texture*)

Adds a texture to the `BitmapFont`.

- void **clear** ()

Clears all the font data and settings.

- int **create_from_fnt** (*String path*)

Creates a `BitmapFont` from the `*.fnt` file at *path*.

- *Vector2 get_char_size* (*int char, int next=0*) const

Returns the size of a character, optionally taking kerning into account if the next character is provided.

- *int get_kerning_pair* (*int char_a, int char_b*) const

Returns a kerning pair as a difference.

- *Texture get_texture* (*int idx*) const

Returns the font atlas texture at index *idx*.

- *int get_texture_count* () const

Returns the number of textures in the `BitmapFont` atlas.

25.68 BoneAttachment

Inherits: `Spatial < Node < Object`

Category: Core

25.68.1 Brief Description

A node that will attach to a bone.

25.68.2 Member Variables

- *String bone_name* - The name of the attached bone.

25.68.3 Description

This node must be the child of a `Skeleton` node. You can then select a bone for this node to attach to. The `BoneAttachment` node will copy the transform of the selected bone.

25.69 bool

Category: Built-In Types

25.69.1 Brief Description

Boolean built-in type

25.69.2 Member Functions

<i>bool</i>	<code>bool (int from)</code>
<i>bool</i>	<code>bool (float from)</code>
<i>bool</i>	<code>bool (String from)</code>

25.69.3 Description

Boolean built-in type.

25.69.4 Member Function Description

- `bool bool (int from)`

Cast an *int* value to a boolean value, this method will return true if called with an integer value different to 0 and false in other case.

- `bool bool (float from)`

Cast a *float* value to a boolean value, this method will return true if called with a floating point value different to 0 and false in other case.

- `bool bool (String from)`

Cast a *String* value to a boolean value, this method will return true if called with a non empty string and false in other case. Examples: `bool ('False')` returns true, `bool ('')`. returns false

25.70 BoxContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Inherited By: *VBoxContainer, HBoxContainer, ColorPicker*

Category: Core

25.70.1 Brief Description

Base class for box containers.

25.70.2 Member Functions

<code>void</code>	<code>add_spacer (bool begin)</code>
-------------------	--

25.70.3 Member Variables

- *AlignMode alignment* - The alignment of the container's children (must be one of ALIGN_BEGIN, ALIGN_CENTER, or ALIGN_END).

25.70.4 Enums

enum AlignMode

- **ALIGN_BEGIN = 0** — Aligns children with the beginning of the container.
- **ALIGN_CENTER = 1** — Aligns children with the center of the container.
- **ALIGN_END = 2** — Aligns children with the end of the container.

25.70.5 Description

Arranges child controls vertically or horizontally, and rearranges the controls automatically when their minimum size changes.

25.70.6 Member Function Description

- void **add_spacer (bool begin)**

Adds a control to the box as a spacer. If `true`, *begin* will insert the spacer control in front of other children.

25.71 BoxShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.71.1 Brief Description

Box shape resource.

25.71.2 Member Variables

- *Vector3 extents* - The shape's half extents.

25.71.3 Description

3D box shape that can be a child of a *PhysicsBody* or *Area*.

25.72 BulletPhysicsDirectBodyState

Inherits: *PhysicsDirectBodyState < Object*

Category: Core

25.72.1 Brief Description

25.73 BulletPhysicsServer

Inherits: *PhysicsServer < Object*

Category: Core

25.73.1 Brief Description

25.74 Button

Inherits: *BaseButton < Control < CanvasItem < Node < Object*

Inherited By: *OptionButton, ColorPickerButton, CheckButton, MenuButton, ToolButton, CheckBox*

Category: Core

25.74.1 Brief Description

Standard themed Button.

25.74.2 Member Variables

- *TextAlign align* - Text alignment policy for the button's text, use one of the ALIGN_* constants.
- *bool clip_text* - When this property is enabled, text that is too large to fit the button is clipped, when disabled the Button will always be wide enough to hold the text. This property is disabled by default.
- *bool flat* - Flat buttons don't display decoration.
- *Texture icon* - Button's icon, if text is present the icon will be placed before the text.
- *String text* - The button's text that will be displayed inside the button's area.

25.74.3 Enums

enum TextAlign

- **ALIGN_LEFT = 0** — Align the text to the left.
- **ALIGN_CENTER = 1** — Align the text to the center.
- **ALIGN_RIGHT = 2** — Align the text to the right.

25.74.4 Description

Button is the standard themed button. It can contain text and an icon, and will display them according to the current *Theme*.

25.75 ButtonGroup

Inherits: *Resource < Reference < Object*

Category: Core

25.75.1 Brief Description

Group of Buttons.

25.75.2 Member Functions

<i>BaseButton</i>	<code>get_pressed_button ()</code>
-------------------	------------------------------------

25.75.3 Description

Group of *Button*. All direct and indirect children buttons become radios. Only one allows being pressed.

25.75.4 Member Function Description

- *BaseButton get_pressed_button ()*

Return the pressed button.

25.76 Camera

Inherits: *Spatial < Node < Object*

Inherited By: *InterpolatedCamera, ARVRCamera*

Category: Core

25.76.1 Brief Description

Camera node, displays from a point of view.

25.76.2 Member Functions

void	<i>clear_current ()</i>
<i>Transform</i>	<i>get_camera_transform () const</i>
<i>bool</i>	<i>is_position_behind (Vector3 world_point) const</i>
void	<i>make_current ()</i>
<i>Vector3</i>	<i>project_local_ray_normal (Vector2 screen_point) const</i>
<i>Vector3</i>	<i>project_position (Vector2 screen_point) const</i>
<i>Vector3</i>	<i>project_ray_normal (Vector2 screen_point) const</i>
<i>Vector3</i>	<i>project_ray_origin (Vector2 screen_point) const</i>
void	<i>set_orthogonal (float size, float z_near, float z_far)</i>
void	<i>set_perspective (float fov, float z_near, float z_far)</i>
<i>Vector2</i>	<i>unproject_position (Vector3 world_point) const</i>

25.76.3 Member Variables

- *int cull_mask* - The culling mask that describes which 3D render layers are rendered by this camera.
- *bool current* - If `true` the ancestor *Viewport* is currently using this Camera. Default value: `false`.
- *DopplerTracking doppler_tracking* - If not `DOPPLER_TRACKING_DISABLED` this Camera will simulate the Doppler effect for objects changed in particular `_process` methods. Default value: `DOPPLER_TRACKING_DISABLED`.
- *Environment environment* - Set the *Environment* to use for this Camera.
- *float far* - The distance to the far culling boundary for this Camera relative to its local z-axis.
- *float fov* - The camera's field of view angle (in degrees). Only applicable in perspective mode. Since `keep_aspect` locks one axis, `fov` sets the other axis' field of view angle.
- *float h_offset* - The horizontal (X) offset of the Camera viewport.
- *KeepAspect keep_aspect* - The axis to lock during `fov/size` adjustments.
- *float near* - The distance to the near culling boundary for this Camera relative to its local z-axis.
- *Projection projection* - The camera's projection mode. In `PROJECTION_PERSPECTIVE` mode, objects' z-distance from the camera's local space scales their perceived size.
- *float size* - The camera's size measured as 1/2 the width or height. Only applicable in orthogonal mode. Since `keep_aspect` locks on axis, `size` sets the other axis' size length.
- *float v_offset* - The horizontal (Y) offset of the Camera viewport.

25.76.4 Enums

enum **DopplerTracking**

- **DOPPLER_TRACKING_DISABLED = 0** — Disable Doppler effect simulation (default).
- **DOPPLER_TRACKING_IDLE_STEP = 1** — Simulate Doppler effect by tracking positions of objects that are changed in `_process`. Changes in the relative velocity of this Camera compared to those objects affect how Audio is perceived (changing the Audio's pitch shift).

- **DOPPLER_TRACKING_PHYSICS_STEP = 2** — Simulate Doppler effect by tracking positions of objects that are changed in `_physics_process`. Changes in the relative velocity of this Camera compared to those objects affect how Audio is perceived (changing the Audio's pitch shift).

enum **Projection**

- **PROJECTION_PERSPECTIVE = 0** — Perspective Projection (object's size on the screen becomes smaller when far away).
- **PROJECTION_ORTHOGONAL = 1** — Orthogonal Projection (objects remain the same size on the screen no matter how far away they are).

enum **KeepAspect**

- **KEEP_WIDTH = 0** — Preserves the horizontal aspect ratio.
- **KEEP_HEIGHT = 1** — Preserves the vertical aspect ratio.

25.76.5 Description

Camera is a special node that displays what is visible from its current location. Cameras register themselves in the nearest *Viewport* node (when ascending the tree). Only one camera can be active per viewport. If no viewport is available ascending the tree, the Camera will register in the global viewport. In other words, a Camera just provides 3D display capabilities to a *Viewport*, and, without one, a scene registered in that *Viewport* (or higher viewports) can't be displayed.

25.76.6 Member Function Description

- **void clear_current ()**

If this is the current Camera, remove it from being current. If it is inside the node tree, request to make the next Camera current, if any.

- ***Transform* get_camera_transform () const**

Get the camera transform. Subclassed cameras (such as CharacterCamera) may provide different transforms than the *Node* transform.

- ***bool* is_position_behind (*Vector3* world_point) const**

Returns `true` if the given position is behind the Camera.

- **void make_current ()**

Make this camera the current Camera for the *Viewport* (see class description). If the Camera Node is outside the scene tree, it will attempt to become current once it's added.

- ***Vector3* project_local_ray_normal (*Vector2* screen_point) const**

Returns a normal vector from the screen point location directed along the camera. Orthogonal cameras are normalized. Perspective cameras account for perspective, screen width/height, etc.

- ***Vector3* project_position (*Vector2* screen_point) const**

Returns how a 2D coordinate in the Viewport rectangle maps to a 3D point in worldspace.

- ***Vector3* project_ray_normal (*Vector2* screen_point) const**

Returns a normal vector in worldspace, that is the result of projecting a point on the *Viewport* rectangle by the camera projection. This is useful for casting rays in the form of (origin, normal) for object intersection or picking.

- ***Vector3* project_ray_origin (*Vector2* screen_point) const**

Returns a 3D position in worldspace, that is the result of projecting a point on the *Viewport* rectangle by the camera projection. This is useful for casting rays in the form of (origin, normal) for object intersection or picking.

- void **set_orthogonal** (*float size, float z_near, float z_far*)

Set the camera projection to orthogonal mode, by specifying a width and the *near* and *far* clip planes in worldspace units. (As a hint, 2D games often use this projection, with values specified in pixels)

- void **set_perspective** (*float fov, float z_near, float z_far*)

Set the camera projection to perspective mode, by specifying a *FOV* Y angle in degrees (FOV means Field of View), and the *near* and *far* clip planes in worldspace units.

- *Vector2 unproject_position (Vector3 world_point) const*

Returns how a 3D point in worldspace maps to a 2D coordinate in the *Viewport* rectangle.

25.77 Camera2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.77.1 Brief Description

Camera node for 2D scenes.

25.77.2 Member Functions

void	<i>align ()</i>
void	<i>clear_current ()</i>
void	<i>force_update_scroll ()</i>
<i>Vector2</i>	<i>get_camera_position () const</i>
<i>Vector2</i>	<i>get_camera_screen_center () const</i>
void	<i>make_current ()</i>
void	<i>reset_smoothing ()</i>

25.77.3 Member Variables

- *AnchorMode anchor_mode* - The Camera2D's anchor point. See `ANCHOR_MODE_*` constants.
- *bool current* - If `true` the camera is the active camera for the current scene. Only one camera can be current, so setting a different camera `current` will disable this one.
- *Node custom_viewport* - The custom *Viewport* node attached to the Camera2D. If null or not a *Viewport*, uses the default viewport instead.
- *float drag_margin_bottom* - Bottom margin needed to drag the camera. A value of 1 makes the camera move only when reaching the edge of the screen.
- *bool drag_margin_h_enabled* - If `true` the camera only moves when reaching the horizontal drag margins. If `false` the camera moves horizontally regardless of margins. Default value: `true`.

- *float drag_margin_left* - Left margin needed to drag the camera. A value of 1 makes the camera move only when reaching the edge of the screen.
- *float drag_margin_right* - Right margin needed to drag the camera. A value of 1 makes the camera move only when reaching the edge of the screen.
- *float drag_margin_top* - Top margin needed to drag the camera. A value of 1 makes the camera move only when reaching the edge of the screen.
- *bool drag_margin_v_enabled* - If `true` the camera only moves when reaching the vertical drag margins. If `false` the camera moves vertically regardless of margins. Default value: `true`.
- *bool editor_draw_drag_margin* - If `true` draws the camera's drag margin rectangle in the editor. Default value: `false`
- *bool editor_draw_limits* - If `true` draws the camera's limits rectangle in the editor. Default value: `true`
- *bool editor_draw_screen* - If `true` draws the camera's screen rectangle in the editor. Default value: `false`
- *int limit_bottom* - Bottom scroll limit in pixels. The camera stops moving when reaching this value.
- *int limit_left* - Left scroll limit in pixels. The camera stops moving when reaching this value.
- *int limit_right* - Right scroll limit in pixels. The camera stops moving when reaching this value.
- *bool limit_smoothed* - If `true` the camera smoothly stops when reaches its limits. Default value: `false`
- *int limit_top* - Top scroll limit in pixels. The camera stops moving when reaching this value.
- *Vector2 offset* - The camera's offset, useful for looking around or camera shake animations.
- *float offset_h* - The horizontal offset of the camera, relative to the drag margins. Default value: 0
- *float offset_v* - The vertical offset of the camera, relative to the drag margins. Default value: 0
- *bool rotating* - If `true` the camera rotates with the target. Default value: `false`
- *bool smoothing_enabled* - If `true` the camera smoothly moves towards the target at *smoothing_speed*. Default value: `false`
- *float smoothing_speed* - Speed in pixels per second of the camera's smoothing effect when *smoothing_enabled* is `true`
- *Vector2 zoom* - The camera's zoom relative to the viewport. Values larger than `Vector2(1, 1)` zoom out and smaller values zoom in. For an example, use `Vector2(0.5, 0.5)` for a 2x zoom in, and `Vector2(4, 4)` for a 4x zoom out.

25.77.4 Enums

enum AnchorMode

- **ANCHOR_MODE_FIXED_TOP_LEFT = 0** — The camera's position is fixed so that the top-left corner is always at the origin.
- **ANCHOR_MODE_DRAG_CENTER = 1** — The camera's position takes into account vertical/horizontal offsets and the screen size.

25.77.5 Description

Camera node for 2D scenes. It forces the screen (current layer) to scroll following this node. This makes it easier (and faster) to program scrollable scenes than manually changing the position of *CanvasItem* based nodes.

This node is intended to be a simple helper to get things going quickly and it may happen often that more functionality is desired to change how the camera works. To make your own custom camera node, simply inherit from *Node2D* and change the transform of the canvas by calling `get_viewport().set_canvas_transform(m)` in *Viewport*.

25.77.6 Member Function Description

- void **align()**

Align the camera to the tracked node

- void **clear_current()**

Removes any *Camera2D* from the ancestor *Viewport*'s internal currently-assigned camera.

- void **force_update_scroll()**

Force the camera to update scroll immediately.

- *Vector2* **get_camera_position()** const

Return the camera position.

- *Vector2* **get_camera_screen_center()** const

Returns the location of the *Camera2D*'s screen-center, relative to the origin.

- void **make_current()**

Make this the current 2D camera for the scene (viewport and layer), in case there's many cameras in the scene.

- void **reset_smoothing()**

Set the camera's position immediately to its current smoothing destination.

This has no effect if smoothing is disabled.

25.78 CanvasItem

Inherits: *Node < Object*

Inherited By: *Node2D, Control*

Category: Core

25.78.1 Brief Description

Base class of anything 2D.

25.78.2 Member Functions

| void | **_draw()** virtual

Continued on next page

Table 25.6 – continued from previous page

float	<code>draw_char (Font font, Vector2 position, String char, String next, Color modulate=Color(1, 1, 1, 1))</code>
void	<code>draw_circle (Vector2 position, float radius, Color color)</code>
void	<code>draw_colored_polygon (PoolVector2Array points, Color color, PoolVector2Array uvs=PoolVector2Array(), Texture texture=null)</code>
void	<code>draw_line (Vector2 from, Vector2 to, Color color, float width=1.0, bool antialiased=false)</code>
void	<code>draw_multiline (PoolVector2Array points, Color color, float width=1.0, bool antialiased=false)</code>
void	<code>draw_multiline_colors (PoolVector2Array points, PoolColorArray colors, float width=1.0, bool antialiased=false)</code>
void	<code>draw_polygon (PoolVector2Array points, PoolColorArray colors, PoolVector2Array uvs=PoolVector2Array(), Texture texture=null)</code>
void	<code>draw_polyline (PoolVector2Array points, Color color, float width=1.0, bool antialiased=false)</code>
void	<code>draw_polyline_colors (PoolVector2Array points, PoolColorArray colors, float width=1.0, bool antialiased=false)</code>
void	<code>draw_primitive (PoolVector2Array points, PoolColorArray colors, PoolVector2Array uvs, Texture texture=null, float width=1.0)</code>
void	<code>draw_rect (Rect2 rect, Color color, bool filled=true)</code>
void	<code>draw_set_transform (Vector2 position, float rotation, Vector2 scale)</code>
void	<code>draw_set_transform_matrix (Transform2D xform)</code>
void	<code>draw_string (Font font, Vector2 position, String text, Color modulate=Color(1, 1, 1, 1), int clip_w=-1)</code>
void	<code>draw_style_box (StyleBox style_box, Rect2 rect)</code>
void	<code>draw_texture (Texture texture, Vector2 position, Color modulate=Color(1, 1, 1, 1), Texture normal_map=null)</code>
void	<code>draw_texture_rect (Texture texture, Rect2 rect, bool tile, Color modulate=Color(1, 1, 1, 1), bool transpose=false, Texture normal_map=null)</code>
void	<code>draw_texture_rect_region (Texture texture, Rect2 rect, Rect2 src_rect, Color modulate=Color(1, 1, 1, 1), bool transpose=false)</code>
RID	<code>get_canvas () const</code>
RID	<code>get_canvas_item () const</code>
Transform2D	<code>get_canvas_transform () const</code>
Vector2	<code>get_global_mouse_position () const</code>
Transform2D	<code>get_global_transform () const</code>
Transform2D	<code>get_global_transform_with_canvas () const</code>
Vector2	<code>get_local_mouse_position () const</code>
Transform2D	<code>get_transform () const</code>
Rect2	<code>get_viewport_rect () const</code>
Transform2D	<code>get_viewport_transform () const</code>
World2D	<code>get_world_2d () const</code>
void	<code>hide ()</code>
bool	<code>is_local_transform_notification_enabled () const</code>
bool	<code>is_set_as_toplevel () const</code>
bool	<code>is_transform_notification_enabled () const</code>
bool	<code>is_visible_in_tree () const</code>
Vector2	<code>make_canvas_position_local (Vector2 screen_point) const</code>
InputEvent	<code>make_input_local (InputEvent event) const</code>
void	<code>set_as_toplevel (bool enable)</code>
void	<code>set_notify_local_transform (bool enable)</code>
void	<code>set_notify_transform (bool enable)</code>
void	<code>show ()</code>
void	<code>update ()</code>

25.78.3 Signals

- **draw ()**

Emitted when the CanvasItem must redraw. This can only be connected realtime, as deferred will not allow drawing.

- **hide ()**

Emitted when becoming hidden.

- **item_rect_changed ()**

Emitted when the item rect has changed.

- **visibility_changed ()**

Emitted when the visibility (hidden/visible) changes.

25.78.4 Member Variables

- **int light_mask** - The rendering layers in which this `CanvasItem` responds to `Light2D` nodes. Default value: 1.
- **Material material** - The material applied to textures on this `CanvasItem`. Default value: `null`.
- **Color modulate** - The color applied to textures on this `CanvasItem`. Default value: `Color(1, 1, 1, 1)` (opaque “white”).
- **Color self_modulate** - The color applied to textures on this `CanvasItem`. This is not inherited by children `CanvasItems`. Default value: `Color(1, 1, 1, 1)` (opaque “white”)..
- **bool show_behind_parent** - If true the object draws behind its parent. Default value: `false`.
- **bool show_on_top** - If true the object draws on top of its parent. Default value: `true`.
- **bool use_parent_material** - If true the parent `CanvasItem`’s *material* property is used as this one’s material. Default value: `false`.
- **bool visible** - If true this `CanvasItem` is drawn. Default value: `true`.

25.78.5 Numeric Constants

- **NOTIFICATION_TRANSFORM_CHANGED = 29** — Canvas item transform has changed. Only received if requested.
- **NOTIFICATION_DRAW = 30** — `CanvasItem` is requested to draw.
- **NOTIFICATION_VISIBILITY_CHANGED = 31** — Canvas item visibility has changed.
- **NOTIFICATION_ENTER_CANVAS = 32** — Canvas item has entered the canvas.
- **NOTIFICATION_EXIT_CANVAS = 33** — Canvas item has exited the canvas.

25.78.6 Enums

enum BlendMode

- **BLEND_MODE_MIX = 0** — Mix blending mode. Colors are assumed to be independent of the alpha (opacity) value.
- **BLEND_MODE_ADD = 1** — Additive blending mode.
- **BLEND_MODE_SUB = 2** — Subtractive blending mode.
- **BLEND_MODE_MUL = 3** — Multiplicative blending mode.
- **BLEND_MODE_PREMULT_ALPHA = 4** — Mix blending mode. Colors are assumed to be premultiplied by the alpha (opacity) value.

25.78.7 Description

Base class of anything 2D. Canvas items are laid out in a tree and children inherit and extend the transform of their parent. CanvasItem is extended by *Control*, for anything GUI related, and by *Node2D* for anything 2D engine related.

Any CanvasItem can draw. For this, the “update” function must be called, then NOTIFICATION_DRAW will be received on idle time to request redraw. Because of this, canvas items don’t need to be redraw on every frame, improving the performance significantly. Several functions for drawing on the CanvasItem are provided (see `draw_*` functions). They can only be used inside the notification, signal or `_draw()` overrides function, though.

Canvas items are drawn in tree order. By default, children are on top of their parents so a root CanvasItem will be drawn behind everything (this can be changed per item though).

Canvas items can also be hidden (hiding also their subtree). They provide many means for changing standard parameters such as opacity (for it and the subtree) and self opacity, blend mode.

Ultimately, a transform notification can be requested, which will notify the node that its global position changed in case the parent tree changed.

25.78.8 Member Function Description

- `void _draw() virtual`

Called (if exists) to draw the canvas item.

- `float draw_char(Font font, Vector2 position, String char, String next, Color modulate=Color(1, 1, 1, 1))`

Draws a string character using a custom font. Returns the advance, depending on the char width and kerning with an optional next char.

- `void draw_circle(Vector2 position, float radius, Color color)`

Draws a colored circle.

- `void draw_colored_polygon(PoolVector2Array points, Color color, PoolVector2Array uvs=PoolVector2Array(), Texture texture=null, Texture normal_map=null, bool antialiased=false)`

Draws a colored polygon of any amount of points, convex or concave.

- `void draw_line(Vector2 from, Vector2 to, Color color, float width=1.0, bool antialiased=false)`

Draws a line from a 2D point to another, with a given color and width. It can be optionally antialiased.

- `void draw_multiline(PoolVector2Array points, Color color, float width=1.0, bool antialiased=false)`

Draws multiple, parallel lines with a uniform `color` and `width` and optional antialiasing.

- `void draw_multiline_colors(PoolVector2Array points, PoolColorArray colors, float width=1.0, bool antialiased=false)`

Draws multiple, parallel lines with a uniform `width`, segment-by-segment coloring, and optional antialiasing. Colors assigned to line segments match by index between `points` and `colors`.

- `void draw_polygon(PoolVector2Array points, PoolColorArray colors, PoolVector2Array uvs=PoolVector2Array(), Texture texture=null, Texture normal_map=null, bool antialiased=false)`

Draws a polygon of any amount of points, convex or concave.

- `void draw_polyline(PoolVector2Array points, Color color, float width=1.0, bool antialiased=false)`

Draws interconnected line segments with a uniform `color` and `width` and optional antialiasing.

- `void draw_polyline_colors(PoolVector2Array points, PoolColorArray colors, float width=1.0, bool antialiased=false)`

Draws interconnected line segments with a uniform width, segment-by-segment coloring, and optional antialiasing. Colors assigned to line segments match by index between points and colors.

- void **draw_primitive** (*PoolVector2Array* points, *PoolColorArray* colors, *PoolVector2Array* uvs, *Texture* texture=null, *float* width=1.0, *Texture* normal_map=null)

Draws a custom primitive, 1 point for a point, 2 points for a line, 3 points for a triangle and 4 points for a quad.

- void **draw_rect** (*Rect2* rect, *Color* color, *bool* filled=true)

Draws a colored rectangle.

- void **draw_set_transform** (*Vector2* position, *float* rotation, *Vector2* scale)

Sets a custom transform for drawing via components. Anything drawn afterwards will be transformed by this.

- void **draw_set_transform_matrix** (*Transform2D* xform)

Sets a custom transform for drawing via matrix. Anything drawn afterwards will be transformed by this.

- void **draw_string** (*Font* font, *Vector2* position, *String* text, *Color* modulate=Color(1, 1, 1, 1), *int* clip_w=-1)

Draws a string using a custom font.

- void **draw_style_box** (*StyleBox* style_box, *Rect2* rect)

Draws a styled rectangle.

- void **draw_texture** (*Texture* texture, *Vector2* position, *Color* modulate=Color(1, 1, 1, 1), *Texture* normal_map=null)

Draws a texture at a given position.

- void **draw_texture_rect** (*Texture* texture, *Rect2* rect, *bool* tile, *Color* modulate=Color(1, 1, 1, 1), *bool* transpose=false, *Texture* normal_map=null)

Draws a textured rectangle at a given position, optionally modulated by a color. Transpose swaps the x and y coordinates when reading the texture.

- void **draw_texture_rect_region** (*Texture* texture, *Rect2* rect, *Rect2* src_rect, *Color* modulate=Color(1, 1, 1, 1), *bool* transpose=false, *Texture* normal_map=null, *bool* clip_uv=true)

Draws a textured rectangle region at a given position, optionally modulated by a color. Transpose swaps the x and y coordinates when reading the texture.

- *RID* **get_canvas** () const

Return the *RID* of the *World2D* canvas where this item is in.

- *RID* **get_canvas_item** () const

Return the canvas item *RID* used by *VisualServer* for this item.

- *Transform2D* **get_canvas_transform** () const

Get the transform matrix of this item's canvas.

- *Vector2* **get_global_mouse_position** () const

Get the global position of the mouse.

- *Transform2D* **get_global_transform** () const

Get the global transform matrix of this item.

- *Transform2D* **get_global_transform_with_canvas** () const

Get the global transform matrix of this item in relation to the canvas.

- `Vector2 get_local_mouse_position() const`

Get the mouse position relative to this item's position.

- `Transform2D get_transform() const`

Get the transform matrix of this item.

- `Rect2 get_viewport_rect() const`

Get the viewport's boundaries as a `Rect2`.

- `Transform2D get_viewport_transform() const`

Get this item's transform in relation to the viewport.

- `World2D get_world_2d() const`

Get the `World2D` where this item is in.

- `void hide()`

Hide the `CanvasItem` currently visible.

- `bool is_local_transform_notification_enabled() const`

Returns `true` if local transform notifications are communicated to children.

- `bool is_set_as_toplevel() const`

Return if set as toplevel. See `set_as_toplevel`.

- `bool is_transform_notification_enabled() const`

Returns `true` if global transform notifications are communicated to children.

- `bool is_visible_in_tree() const`

Returns `true` if the node is present in the `SceneTree`, its `visible` property is `true` and its inherited visibility is also `true`.

- `Vector2 make_canvas_position_local(Vector2 screen_point) const`

Assigns `screen_point` as this node's new local transform.

- `InputEvent make_input_local(InputEvent event) const`

Transformations issued by `event`'s inputs are applied in local space instead of global space.

- `void set_as_toplevel(bool enable)`

Sets as top level. This means that it will not inherit transform from parent canvas items.

- `void set_notify_local_transform(bool enable)`

If `enable` is `true`, children will be updated with local transform data.

- `void set_notify_transform(bool enable)`

If `enable` is `true`, children will be updated with global transform data.

- `void show()`

Show the `CanvasItem` currently hidden.

- `void update()`

Queue the `CanvasItem` for update. `NOTIFICATION_DRAW` will be called on idle time to request redraw.

25.79 CanvasItemMaterial

Inherits: *Material < Resource < Reference < Object*

Category: Core

25.79.1 Brief Description

A material for *CanvasItems*.

25.79.2 Member Variables

- *BlendMode blend_mode* - The manner in which a material's rendering is applied to underlying textures.
- *LightMode light_mode* - The manner in which material reacts to lighting.

25.79.3 Enums

enum LightMode

- **LIGHT_MODE_NORMAL = 0** — Render the material using both light and non-light sensitive material properties.
- **LIGHT_MODE_UNSHADED = 1** — Render the material as if there were no light.
- **LIGHT_MODE_LIGHT_ONLY = 2** — Render the material as if there were only light.

enum BlendMode

- **BLEND_MODE_MIX = 0** — Mix blending mode. Colors are assumed to be independent of the alpha (opacity) value.
- **BLEND_MODE_ADD = 1** — Additive blending mode.
- **BLEND_MODE_SUB = 2** — Subtractive blending mode.
- **BLEND_MODE_MUL = 3** — Multiplicative blending mode.
- **BLEND_MODE_PREMULT_ALPHA = 4** — Mix blending mode. Colors are assumed to be premultiplied by the alpha (opacity) value.

25.79.4 Description

CanvasItemMaterials provide a means of modifying the textures associated with a *CanvasItem*. They specialize in describing blend and lighting behaviors for textures. Use a *ShaderMaterial* to more fully customize a material's interactions with a *CanvasItem*.

25.80 CanvasLayer

Inherits: *Node < Object*

Inherited By: *ParallaxBackground*

Category: Core

25.80.1 Brief Description

Canvas drawing layer.

25.80.2 Member Functions

<i>World2D</i>	<code>get_world_2d () const</code>
----------------	------------------------------------

25.80.3 Member Variables

- *Node custom_viewport* - The custom *Viewport* node assigned to the *CanvasLayer*. If null, uses the default *viewport* instead.
- *int layer* - Layer index for draw order. Lower values are drawn first. Default value: 1.
- *Vector2 offset* - The layer's base offset.
- *float rotation* - The layer's rotation in radians.
- *float rotation_degrees* - The layer's rotation in degrees.
- *Vector2 scale* - The layer's scale.
- *Transform2D transform* - The layer's transform.

25.80.4 Description

Canvas drawing layer. *CanvasItem* nodes that are direct or indirect children of a *CanvasLayer* will be drawn in that layer. The layer is a numeric index that defines the draw order. The default 2D scene renders with index 0, so a *CanvasLayer* with index -1 will be drawn below, and one with index 1 will be drawn above. This is very useful for HUDs (in layer 1+ or above), or backgrounds (in layer -1 or below).

25.80.5 Member Function Description

- *World2D get_world_2d () const*

Return the *World2D* used by this layer.

25.81 CanvasModulate

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.81.1 Brief Description

Tint the entire canvas.

25.81.2 Member Variables

- *Color color* - The tint color to apply.

25.81.3 Description

`CanvasModulate` tints the canvas elements using its assigned `color`.

25.82 CapsuleMesh

Inherits: *PrimitiveMesh < Mesh < Resource < Reference < Object*

Category: Core

25.82.1 Brief Description

Class representing a capsule-shaped *PrimitiveMesh*.

25.82.2 Member Variables

- *float mid_height* - Height of the capsule mesh from the center point. Defaults to 1.0.
- *int radial_segments* - Number of radial segments on the capsule mesh. Defaults to 64.
- *float radius* - Radius of the capsule mesh. Defaults to 1.0.
- *int rings* - Number of rings along the height of the capsule. Defaults to 8.

25.82.3 Description

Class representing a capsule-shaped *PrimitiveMesh*.

25.83 CapsuleShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.83.1 Brief Description

Capsule shape for collisions.

25.83.2 Member Variables

- *float height* - The capsule's height.
- *float radius* - The capsule's radius.

25.83.3 Description

Capsule shape for collisions.

25.84 CapsuleShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.84.1 Brief Description

Capsule shape for 2D collisions.

25.84.2 Member Variables

- *float height* - The capsule's height.
- *float radius* - The capsules's radius.

25.84.3 Description

Capsule shape for 2D collisions.

25.85 CenterContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Category: Core

25.85.1 Brief Description

Keeps children controls centered.

25.85.2 Member Variables

- *bool use_top_left* - If true centers children relative to the CenterContainer's top left corner. Default value: false.

25.85.3 Description

CenterContainer Keeps children controls centered. This container keeps all children to their minimum size, in the center.

25.86 CheckBox

Inherits: *Button < BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.86.1 Brief Description

Binary choice user interface widget

25.86.2 Description

A checkbox allows the user to make a binary choice (choosing only one of two possible options), for example Answer 'yes' or 'no'.

25.87 CheckButton

Inherits: *Button < BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.87.1 Brief Description

Checkable button.

25.87.2 Description

CheckButton is a toggle button displayed as a check field.

25.88 CircleShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.88.1 Brief Description

Circular shape for 2D collisions.

25.88.2 Member Variables

- *float radius* - The circle's radius.

25.88.3 Description

Circular shape for 2D collisions. This shape is useful for modeling balls or small characters and its collision detection with everything else is very fast.

25.89 ClassDB

Inherits: *Object*

Category: Core

25.89.1 Brief Description

Class information repository.

25.89.2 Member Functions

<i>bool</i>	<code>can_instance (String class) const</code>
<i>bool</i>	<code>class_exists (String class) const</code>
<i>String</i>	<code>class_get_category (String class) const</code>
<i>int</i>	<code>class_get_integer_constant (String class, String name) const</code>
<i>PoolStringArray</i>	<code>class_get_integer_constant_list (String class, bool no_inheritance=false) const</code>
<i>Array</i>	<code>class_get_method_list (String class, bool no_inheritance=false) const</code>
<i>Variant</i>	<code>class_get_property (Object object, String property) const</code>
<i>Array</i>	<code>class_get_property_list (String class, bool no_inheritance=false) const</code>
<i>Dictionary</i>	<code>class_get_signal (String class, String signal) const</code>
<i>Array</i>	<code>class_get_signal_list (String class, bool no_inheritance=false) const</code>
<i>bool</i>	<code>class_has_integer_constant (String class, String name) const</code>
<i>bool</i>	<code>class_has_method (String class, String method, bool no_inheritance=false) const</code>
<i>bool</i>	<code>class_has_signal (String class, String signal) const</code>
<i>int</i>	<code>class_set_property (Object object, String property, Variant value) const</code>
<i>PoolStringArray</i>	<code>get_class_list () const</code>
<i>PoolStringArray</i>	<code>get_inheritors_from_class (String class) const</code>
<i>String</i>	<code>get_parent_class (String class) const</code>
<i>Variant</i>	<code>instance (String class) const</code>
<i>bool</i>	<code>is_class_enabled (String class) const</code>
<i>bool</i>	<code>is_parent_class (String class, String inherits) const</code>

25.89.3 Description

Provides access to metadata stored for every available class.

25.89.4 Member Function Description

- `bool can_instance (String class) const`

Returns true if you can instance objects from the specified ‘class’, false in other case.

- `bool class_exists (String class) const`

Returns whether the specified ‘class’ is available or not.

- `String class_get_category (String class) const`

Returns a category associated with the class for use in documentation and the Asset Library. Debug mode required.

- `int class_get_integer_constant (String class, String name) const`

Returns the value of the integer constant ‘name’ of ‘class’ or its ancestry. Always returns 0 when the constant could not be found.

- `PoolStringArray class_get_integer_constant_list (String class, bool no_inheritance=false) const`

Returns an array with the names all the integer constants of ‘class’ or its ancestry.

- `Array class_get_method_list (String class, bool no_inheritance=false) const`

Returns an array with all the methods of ‘class’ or its ancestry if ‘no_inheritance’ is false. Every element of the array is a *Dictionary* with the following keys: args, default_args, flags, id, name, return: (class_name, hint, hint_string, name, type, usage).

- *Variant* **class_get_property** (*Object* object, *String* property) const

Returns the value of ‘property’ of ‘class’ or its ancestry.

- *Array* **class_get_property_list** (*String* class, *bool* no_inheritance=false) const

Returns an array with all the properties of ‘class’ or its ancestry if ‘no_inheritance’ is false.

- *Dictionary* **class_get_signal** (*String* class, *String* signal) const

Returns the ‘signal’ data of ‘class’ or its ancestry. The returned value is a *Dictionary* with the following keys: args, default_args, flags, id, name, return: (class_name, hint, hint_string, name, type, usage).

- *Array* **class_get_signal_list** (*String* class, *bool* no_inheritance=false) const

Returns an array with all the signals of ‘class’ or its ancestry if ‘no_inheritance’ is false. Every element of the array is a *Dictionary* as described in `class_get_signal`.

- *bool* **class_has_integer_constant** (*String* class, *String* name) const

Return whether ‘class’ or its ancestry has an integer constant called ‘name’ or not.

- *bool* **class_has_method** (*String* class, *String* method, *bool* no_inheritance=false) const

Return whether ‘class’ (or its ancestry if ‘no_inheritance’ is false) has a method called ‘method’ or not.

- *bool* **class_has_signal** (*String* class, *String* signal) const

Return whether ‘class’ or its ancestry has a signal called ‘signal’ or not.

- *int* **class_set_property** (*Object* object, *String* property, *Variant* value) const

Sets ‘property’ value of ‘class’ to ‘value’.

- *PoolStringArray* **get_class_list** () const

Returns the names of all the classes available.

- *PoolStringArray* **get_inheritors_from_class** (*String* class) const

Returns the names of all the classes that directly or indirectly inherit from ‘class’.

- *String* **get_parent_class** (*String* class) const

Returns the parent class of ‘class’.

- *Variant* **instance** (*String* class) const

Creates an instance of ‘class’.

- *bool* **is_class_enabled** (*String* class) const

Returns whether this class is enabled or not.

- *bool* **is_parent_class** (*String* class, *String* inherits) const

Returns whether ‘inherits’ is an ancestor of ‘class’ or not.

25.90 CollisionObject

Inherits: *Spatial < Node < Object*

Inherited By: *PhysicsBody, Area*

Category: Core

25.90.1 Brief Description

Base node for collision objects.

25.90.2 Member Functions

void	<code>_input_event (Object camera, InputEvent event, Vector3 click_position, Vector3 click_normal, int shape_idx) virtual</code>
int	<code>create_shape_owner (Object owner)</code>
RID	<code>get_rid () const</code>
Array	<code>get_shape_owners ()</code>
bool	<code>is_shape_owner_disabled (int owner_id) const</code>
void	<code>remove_shape_owner (int owner_id)</code>
int	<code>shape_find_owner (int shape_index) const</code>
void	<code>shape_owner_add_shape (int owner_id, Shape shape)</code>
void	<code>shape_owner_clear_shapes (int owner_id)</code>
Object	<code>shape_owner_get_owner (int owner_id) const</code>
Shape	<code>shape_owner_get_shape (int owner_id, int shape_id) const</code>
int	<code>shape_owner_get_shape_count (int owner_id) const</code>
int	<code>shape_owner_get_shape_index (int owner_id, int shape_id) const</code>
Transform	<code>shape_owner_get_transform (int owner_id) const</code>
void	<code>shape_owner_remove_shape (int owner_id, int shape_id)</code>
void	<code>shape_owner_set_disabled (int owner_id, bool disabled)</code>
void	<code>shape_owner_set_transform (int owner_id, Transform transform)</code>

25.90.3 Signals

- **input_event** (*Object camera, Object event, Vector3 click_position, Vector3 click_normal, int shape_idx*)

Emitted when `_input_event` receives an event. See its description for details.

- **mouse_entered** ()

Emitted when the mouse pointer enters any of this object's shapes.

- **mouse_exited** ()

Emitted when the mouse pointer exits all this object's shapes.

25.90.4 Member Variables

- **bool input_capture_on_drag** - If true the `CollisionObject` will continue to receive input events as the mouse is dragged across its shapes. Default value: false.
- **bool input_ray_pickable** - If true the `CollisionObject`'s shapes will respond to `RayCasts`. Default value: true.

25.90.5 Description

`CollisionObject` is the base class for physics objects. It can hold any number of collision *Shapes*. Each shape must be assigned to a *shape owner*. The `CollisionObject` can have any number of shape owners. Shape owners are not nodes and do not appear in the editor, but are accessible through code using the `shape_owner__*` methods.

25.90.6 Member Function Description

- `void _input_event (Object camera, InputEvent event, Vector3 click_position, Vector3 click_normal, int shape_idx) virtual`

Accepts unhandled *InputEvents*. `click_position` is the clicked location in world space and `click_normal` is the normal vector extending from the clicked surface of the *Shape* at `shape_idx`. Connect to the `input_event` signal to easily pick up these events.

- `int create_shape_owner (Object owner)`

Creates a new shape owner for the given object. Returns `owner_id` of the new owner for future reference.

- `RID get_rid () const`

Returns the object's *RID*.

- `Array get_shape_owners ()`

Returns an *Array* of `owner_id` identifiers. You can use these ids in other methods that take `owner_id` as an argument.

- `bool is_shape_owner_disabled (int owner_id) const`

If true the shape owner and its shapes are disabled.

- `void remove_shape_owner (int owner_id)`

Removes the given shape owner.

- `int shape_find_owner (int shape_index) const`

Returns the `owner_id` of the given shape.

- `void shape_owner_add_shape (int owner_id, Shape shape)`

Adds a *Shape* to the shape owner.

- `void shape_owner_clear_shapes (int owner_id)`

Removes all shapes from the shape owner.

- `Object shape_owner_get_owner (int owner_id) const`

Returns the parent object of the given shape owner.

- `Shape shape_owner_get_shape (int owner_id, int shape_id) const`

Returns the *Shape* with the given id from the given shape owner.

- `int shape_owner_get_shape_count (int owner_id) const`

Returns the number of shapes the given shape owner contains.

- `int shape_owner_get_shape_index (int owner_id, int shape_id) const`

Returns the child index of the *Shape* with the given id from the given shape owner.

- `Transform shape_owner_get_transform (int owner_id) const`

Returns the shape owner's *Transform*.

- void **shape_owner_remove_shape** (*int owner_id, int shape_id*)

Removes a shape from the given shape owner.

- void **shape_owner_set_disabled** (*int owner_id, bool disabled*)

If `true` disables the given shape owner.

- void **shape_owner_set_transform** (*int owner_id, Transform transform*)

Sets the *Transform* of the given shape owner.

25.91 CollisionObject2D

Inherits: *Node2D < CanvasItem < Node < Object*

Inherited By: *Area2D, PhysicsBody2D*

Category: Core

25.91.1 Brief Description

Base node for 2D collision objects.

25.91.2 Member Functions

void	<code>_input_event (Object viewport, InputEvent event, int shape_idx) virtual</code>
<i>int</i>	<code>create_shape_owner (Object owner)</code>
<i>RID</i>	<code>get_rid () const</code>
<i>Array</i>	<code>get_shape_owners ()</code>
<i>bool</i>	<code>is_shape_owner_disabled (int owner_id) const</code>
<i>bool</i>	<code>is_shape_owner_one_way_collision_enabled (int owner_id) const</code>
void	<code>remove_shape_owner (int owner_id)</code>
<i>int</i>	<code>shape_find_owner (int shape_index) const</code>
void	<code>shape_owner_add_shape (int owner_id, Shape2D shape)</code>
void	<code>shape_owner_clear_shapes (int owner_id)</code>
<i>Object</i>	<code>shape_owner_get_owner (int owner_id) const</code>
<i>Shape2D</i>	<code>shape_owner_get_shape (int owner_id, int shape_id) const</code>
<i>int</i>	<code>shape_owner_get_shape_count (int owner_id) const</code>
<i>int</i>	<code>shape_owner_get_shape_index (int owner_id, int shape_id) const</code>
<i>Transform2D</i>	<code>shape_owner_get_transform (int owner_id) const</code>
void	<code>shape_owner_remove_shape (int owner_id, int shape_id)</code>
void	<code>shape_owner_set_disabled (int owner_id, bool disabled)</code>
void	<code>shape_owner_set_one_way_collision (int owner_id, bool enable)</code>
void	<code>shape_owner_set_transform (int owner_id, Transform2D transform)</code>

25.91.3 Signals

- **input_event** (*Object viewport, Object event, int shape_idx*)

Emitted when an input event occurs and `input_pickable` is `true`. See `_input_event` for details.

- **mouse_entered ()**

Emitted when the mouse pointer enters any of this object's shapes.

- **mouse_exited ()**

Emitted when the mouse pointer exits all this object's shapes.

25.91.4 Member Variables

- *bool input_pickable* - If true this object is pickable. A pickable object can detect the mouse pointer entering/leaving, and if the mouse is inside it, report input events.

25.91.5 Description

`CollisionObject2D` is the base class for 2D physics objects. It can hold any number of 2D collision *Shape2Ds*. Each shape must be assigned to a *shape owner*. The `CollisionObject2D` can have any number of shape owners. Shape owners are not nodes and do not appear in the editor, but are accessible through code using the `shape_owner_*` methods.

25.91.6 Member Function Description

- **void _input_event (Object viewport, InputEvent event, int shape_idx) virtual**

Accepts unhandled *InputEvents*. `shape_idx` is the child index of the clicked *Shape2D*. Connect to the `input_event` signal to easily pick up these events.

- **int create_shape_owner (Object owner)**

Creates a new shape owner for the given object. Returns `owner_id` of the new owner for future reference.

- **RID get_rid () const**

Returns the object's *RID*.

- **Array get_shape_owners ()**

Returns an *Array* of `owner_id` identifiers. You can use these ids in other methods that take `owner_id` as an argument.

- **bool is_shape_owner_disabled (int owner_id) const**

If true the shape owner and its shapes are disabled.

- **bool is_shape_owner_one_way_collision_enabled (int owner_id) const**

Returns true if collisions for the shape owner originating from this `CollisionObject2D` will not be reported to collided with `CollisionObject2Ds`.

- **void remove_shape_owner (int owner_id)**

Removes the given shape owner.

- **int shape_find_owner (int shape_index) const**

Returns the `owner_id` of the given shape.

- **void shape_owner_add_shape (int owner_id, Shape2D shape)**

Adds a *Shape2D* to the shape owner.

- `void shape_owner_clear_shapes (int owner_id)`

Removes all shapes from the shape owner.

- `Object shape_owner_get_owner (int owner_id) const`

Returns the parent object of the given shape owner.

- `Shape2D shape_owner_get_shape (int owner_id, int shape_id) const`

Returns the *Shape2D* with the given id from the given shape owner.

- `int shape_owner_get_shape_count (int owner_id) const`

Returns the number of shapes the given shape owner contains.

- `int shape_owner_get_shape_index (int owner_id, int shape_id) const`

Returns the child index of the *Shape2D* with the given id from the given shape owner.

- `Transform2D shape_owner_get_transform (int owner_id) const`

Returns the shape owner's *Transform2D*.

- `void shape_owner_remove_shape (int owner_id, int shape_id)`

Removes a shape from the given shape owner.

- `void shape_owner_set_disabled (int owner_id, bool disabled)`

If `true` disables the given shape owner.

- `void shape_owner_set_one_way_collision (int owner_id, bool enable)`

If `enable` is `true`, collisions for the shape owner originating from this *CollisionObject2D* will not be reported to collided with *CollisionObject2Ds*.

- `void shape_owner_set_transform (int owner_id, Transform2D transform)`

Sets the *Transform2D* of the given shape owner.

25.92 CollisionPolygon

Inherits: *Spatial < Node < Object*

Category: Core

25.92.1 Brief Description

Editor-only class for defining a collision polygon in 3D space.

25.92.2 Member Variables

- `float depth` - Length that the resulting collision extends in either direction perpendicular to its polygon.
- `bool disabled` - If `true`, no collision will be produced.
- `PoolVector2Array polygon` - Array of vertices which define the polygon.

25.92.3 Description

Allows editing a collision polygon's vertices on a selected plane. Can also set a depth perpendicular to that plane. This class is only available in the editor. It will not appear in the scene tree at runtime. Creates a *Shape* for gameplay. Properties modified during gameplay will have no effect.

25.93 CollisionPolygon2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.93.1 Brief Description

Defines a 2D collision polygon.

25.93.2 Member Variables

- *BuildMode build_mode* - Collision build mode. Use one of the `BUILD_*` constants. Default value: `BUILD_SOLIDS`.
- *bool disabled* - If `true` no collisions will be detected.
- *bool one_way_collision* - If `true` only edges that face up, relative to `CollisionPolygon2D`'s rotation, will collide with other objects.
- *PoolVector2Array polygon* - The polygon's list of vertices. The final point will be connected to the first.

25.93.3 Enums

enum BuildMode

- **BUILD_SOLIDS = 0** — Collisions will include the polygon and its contained area.
- **BUILD_SEGMENTS = 1** — Collisions will only include the polygon edges.

25.93.4 Description

Provides a 2D collision polygon to a *CollisionObject2D* parent. Polygon can be drawn in the editor or specified by a list of vertices.

25.94 CollisionShape

Inherits: *Spatial < Node < Object*

Category: Core

25.94.1 Brief Description

Node that represents collision shape data in 3D space.

25.94.2 Member Functions

void	<i>make_convex_from_brothers ()</i>
void	<i>resource_changed (Resource resource)</i>

25.94.3 Member Variables

- *bool disabled* - A disabled collision shape has no effect in the world.
- *Shape shape* - The actual shape owned by this collision shape.

25.94.4 Description

Editor facility for creating and editing collision shapes in 3D space. You can use this node to represent all sorts of collision shapes, for example, add this to an *Area* to give it a detection shape, or add it to a *PhysicsBody* to give create solid object. **IMPORTANT:** this is an Editor-only helper to create shapes, use `get_shape` to get the actual shape.

25.94.5 Member Function Description

- void **make_convex_from_brothers ()**

Sets the collision shape's shape to the addition of all its convexed *MeshInstance* siblings geometry.

- void **resource_changed (Resource resource)**

If this method exists within a script it will be called whenever the shape resource has been modified.

25.95 CollisionShape2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.95.1 Brief Description

Node that represents collision shape data in 2D space.

25.95.2 Member Variables

- *bool disabled* - A disabled collision shape has no effect in the world.

- *bool one_way_collision* - Sets whether this collision shape should only detect collision on one side (top or bottom).
- *Shape2D shape* - The actual shape owned by this collision shape.

25.95.3 Description

Editor facility for creating and editing collision shapes in 2D space. You can use this node to represent all sorts of collision shapes, for example, add this to an *Area2D* to give it a detection shape, or add it to a *PhysicsBody2D* to give create solid object. **IMPORTANT:** this is an Editor-only helper to create shapes, use `get_shape` to get the actual shape.

25.96 Color

Category: Built-In Types

25.96.1 Brief Description

Color in RGBA format with some support for ARGB format.

25.96.2 Member Functions

<i>Color</i>	<code>Color (float r, float g, float b, float a)</code>
<i>Color</i>	<code>Color (float r, float g, float b)</code>
<i>Color</i>	<code>Color (int from)</code>
<i>Color</i>	<code>Color (String from)</code>
<i>Color</i>	<code>blend (Color over)</code>
<i>Color</i>	<code>contrasted ()</code>
<i>Color</i>	<code>darkened (float amount)</code>
<i>float</i>	<code>gray ()</code>
<i>Color</i>	<code>inverted ()</code>
<i>Color</i>	<code>lightened (float amount)</code>
<i>Color</i>	<code>linear_interpolate (Color b, float t)</code>
<i>int</i>	<code>to_argb32 ()</code>
<i>String</i>	<code>to_html (bool with_alpha=True)</code>
<i>int</i>	<code>to_rgba32 ()</code>

25.96.3 Member Variables

- *float a* - Alpha (0 to 1)
- *int a8* - Alpha (0 to 255)
- *float b* - Blue (0 to 1)
- *int b8* - Blue (0 to 255)
- *float g* - Green (0 to 1)

- *int g8* - Green (0 to 255)
- *float h* - Hue (0 to 1)
- *float r* - Red (0 to 1)
- *int r8* - Red (0 to 255)
- *float s* - Saturation (0 to 1)
- *float v* - Value (0 to 1)

25.96.4 Description

A color is represented as red, green and blue (r,g,b) components. Additionally, “a” represents the alpha component, often used for transparency. Values are in floating point and usually range from 0 to 1. Some methods (such as `set_modulate(color)`) may accept values > 1.

You can also create a color from standardised color names with `@GDScript.ColorN`.

25.96.5 Member Function Description

- *Color Color (float r, float g, float b, float a)*

Constructs a color from an RGBA profile using values between 0 and 1 (float).

```
var c = Color(0.2, 1.0, .7, .8) # a color of an RGBA(51, 255, 178, 204)
```

- *Color Color (float r, float g, float b)*

Constructs a color from an RGB profile using values between 0 and 1 (float). Alpha will always be 1.

```
var c = Color(0.2, 1.0, .7) # a color of an RGBA(51, 255, 178, 255)
```

- *Color Color (int from)*

Constructs a color from a 32-bit integer (each byte represents a component of the RGBA profile).

```
var c = Color(274) # a color of an RGBA(0, 0, 1, 18)
```

- *Color Color (String from)*

Constructs a color from an HTML hexadecimal color string in ARGB or RGB format. See also `@GDScript.ColorN`.

The following string formats are supported:

"#ff00ff00" - ARGB format with '#'

"ff00ff00" - ARGB format

"#ff00ff" - RGB format with '#'

"ff00ff" - RGB format

```
# The following code creates the same color of an RGBA(178, 217, 10, 255)
var c1 = Color("#fb2d90a") # ARGB format with '#'
var c2 = Color("ffb2d90a") # ARGB format
var c3 = Color("#b2d90a") # RGB format with '#'
var c4 = Color("b2d90a") # RGB format
```

- *Color blend (Color over)*

Returns a new color resulting from blending this color over another color. If the color is opaque, the result would also be opaque. The other color could then take a range of values with different alpha values.

```
var bg = Color(0.0, 1.0, 0.0, 0.5) # Green with alpha of 50%
var fg = Color(1.0, 0.0, 0.0, .5) # Red with alpha of 50%
var blendedColor = bg.blend(fg) # Brown with alpha of 75%
```

- **Color contrasted ()**

Returns the most contrasting color.

```
var c = Color(.3, .4, .9)
var contrastedColor = c.contrasted() # a color of an RGBA(204, 229, 102, 255)
```

- **Color darkened (float amount)**

Returns a new color resulting from making this color darker by the specified percentage (0-1).

```
var green = Color(0.0, 1.0, 0.0)
var darkgreen = green.darkened(0.2) # 20% darker than regular green
```

- **float gray ()**

Returns the color's grayscale.

The gray is calculated by $(r + g + b) / 3$.

```
var c = Color(0.2, 0.45, 0.82)
var gray = c.gray() # a value of 0.466667
```

- **Color inverted ()**

Returns the inverted color (1-r, 1-g, 1-b, 1-a).

```
var c = Color(.3, .4, .9)
var invertedColor = c.inverted() # a color of an RGBA(178, 153, 26, 255)
```

- **Color lightened (float amount)**

Returns a new color resulting from making this color lighter by the specified percentage (0-1).

```
var green = Color(0.0, 1.0, 0.0)
var lightgreen = green.lightened(0.2) # 20% lighter than regular green
```

- **Color linear_interpolate (Color b, float t)**

Returns the color of the linear interpolation with another color. The value t is between 0 and 1 (float).

```
var c1 = Color(1.0, 0.0, 0.0)
var c2 = Color(0.0, 1.0, 0.0)
var li_c = c1.linear_interpolate(c2, 0.5) # a color of an RGBA(128, 128, 0, 255)
```

- **int to_argb32 ()**

Returns the color's 32-bit integer in ARGB format (each byte represents a component of the ARGB profile). More compatible with DirectX.

```
var c = Color(1, .5, .2)
print(str(c.to_32())) # prints 4294934323
```

- **String to_html (bool with_alpha=True)**

Returns the color's HTML hexadecimal color string in ARGB format (ex: `ff34f822`).

Optionally flag ‘`false`’ to not include alpha in hexadecimal string.

```
var c = Color(1, 1, 1, .5)
var s1 = c.to_html() # Results "7fffffff"
var s2 = c.to_html(false) # Results 'ffffff'
```

- `int to_rgba32 ()`

Returns the color's 32-bit integer in ARGB format (each byte represents a component of the ARGB profile).

```
var c = Color(1, .5, .2)
print(str(c.to_32())) # prints 4294934323
```

This is same as :ref:`to_argb32<class_Color_to_argb32>` but may be changed later to support RGBA format instead.

25.97 ColorPicker

Inherits: `BoxContainer < Container < Control < CanvasItem < Node < Object`

Category: Core

25.97.1 Brief Description

Color picker control.

25.97.2 Member Functions

void	<code>add_preset (Color color)</code>
------	---

25.97.3 Signals

- `color_changed (Color color)`

Emitted when the color is changed.

25.97.4 Member Variables

- `Color color` - The currently selected color.
- `bool edit_alpha` - If `true`, shows an alpha channel slider (transparency).
- `bool raw_mode` - If `true`, allows the color R, G, B component values to go beyond 1.0, which can be used for certain special operations that require it (like tinting without darkening or rendering sprites in HDR).

25.97.5 Description

This is a simple color picker *Control*. It's useful for selecting a color from an RGB/RGBA colorspace.

25.97.6 Member Function Description

- void **add_preset** (*Color* color)

Adds the current selected to color to a list of colors (presets), the presets will be displayed in the color picker and the user will be able to select them, notice that the presets list is only for this color picker.

25.98 ColorPickerButton

Inherits: *Button < BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.98.1 Brief Description

Button that pops out a *ColorPicker*

25.98.2 Member Functions

<i>ColorPicker</i>	<code>get_picker () const</code>
<i>PopupPanel</i>	<code>get_popup () const</code>

25.98.3 Signals

- **color_changed** (*Color* color)

Emitted when the color changes.

25.98.4 Member Variables

- *Color* **color** - The currently selected color.
- *bool* **edit_alpha** - If `true` the alpha channel in the displayed *ColorPicker* will be visible. Default value: `true`.

25.98.5 Description

Encapsulates a *ColorPicker* making it accessible by pressing a button, pressing the button will toggle the *ColorPicker* visibility

25.98.6 Member Function Description

- *ColorPicker* **get_picker** () const

Returns the *ColorPicker* that this *ColorPickerButton* toggles.

- *PopupPanel* **get_popup** () const

Returns the control's *PopupPanel* which allows you to connect to Popup Signals. This allows you to handle events when the ColorPicker is shown or hidden.

25.99 ColorRect

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.99.1 Brief Description

Colored rect for canvas.

25.99.2 Member Variables

- **Color color** - The color to fill the ColorRect.

```
$ColorRect.color = Color(1, 0, 0, 1) # Set ColorRect node's color to red
```

25.99.3 Description

An object that is represented on the canvas as a rect with color. *Color* is used to set or get color info for the rect.

25.100 ConcavePolygonShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.100.1 Brief Description

Concave polygon shape.

25.100.2 Member Functions

<i>PoolVector3Array</i>	<i>get_faces () const</i>
void	<i>set_faces (PoolVector3Array faces)</i>

25.100.3 Description

Concave polygon shape resource, which can be set into a *PhysicsBody* or area. This shape is created by feeding a list of triangles.

25.100.4 Member Function Description

- `PoolVector3Array get_faces () const`

Return the faces (an array of triangles).

- `void set_faces (PoolVector3Array faces)`

Set the faces (an array of triangles).

25.101 ConcavePolygonShape2D

Inherits: `Shape2D < Resource < Reference < Object`

Category: Core

25.101.1 Brief Description

Concave polygon 2D shape resource for physics.

25.101.2 Member Variables

- `PoolVector2Array segments` - The array of points that make up the ConcavePolygonShape2D's line segments.

25.101.3 Description

Concave polygon 2D shape resource for physics. It is made out of segments and is very optimal for complex polygonal concave collisions. It is really not advised to use for `RigidBody2D` nodes. A `CollisionPolygon2D` in convex decomposition mode (solids) or several convex objects are advised for that instead. Otherwise, a concave polygon 2D shape is better for static collisions.

The main difference between a `ConvexPolygonShape2D` and a `ConcavePolygonShape2D` is that a concave polygon assumes it is concave and uses a more complex method of collision detection, and a convex one forces itself to be convex in order to speed up collision detection.

25.102 ConeTwistJoint

Inherits: `Joint < Spatial < Node < Object`

Category: Core

25.102.1 Brief Description

A twist joint between two 3D bodies

25.102.2 Member Variables

- *float bias* - The speed with which the swing or twist will take place.

The higher, the faster.

- *float relaxation* - Defines, how fast the swing- and twist-speed-difference on both sides gets synced.
- *float softness* - The ease with which the joint starts to twist. If it's too low, it takes more force to start twisting the joint.
- *float swing_span* - Swing is rotation from side to side, around the axis perpendicular to the twist axis.

The swing span defines, how much rotation will not get corrected allong the swing axis.

Could be defined as looseness in the `ConeTwistJoint`.

If below 0.05, this behaviour is locked. Default value: $\text{PI}/4$.

- *float twist_span* - Twist is the rotation around the twist axis, this value defined how far the joint can twist.

Twist is locked if below 0.05.

25.102.3 Enums

enum Param

- **PARAM_SWING_SPAN = 0** — Swing is rotation from side to side, around the axis perpendicular to the twist axis.

The swing span defines, how much rotation will not get corrected allong the swing axis.

Could be defined as looseness in the `ConeTwistJoint`.

If below 0.05, this behaviour is locked. Default value: $\text{PI}/4$. - **PARAM_TWIST_SPAN = 1** — Twist is the rotation around the twist axis, this value defined how far the joint can twist.

Twist is locked if below 0.05. - **PARAM_BIAS = 2** — The speed with which the swing or twist will take place.

The higher, the faster. - **PARAM_SOFTNESS = 3** — The ease with which the joint starts to twist. If it's too low, it takes more force to start twisting the joint. - **PARAM_RELAXATION = 4** — Defines, how fast the swing- and twist-speed-difference on both sides gets synced. - **PARAM_MAX = 5** — End flag of `PARAM_*` constants, used internally.

25.102.4 Description

The joint can rotate the bodies across an axis defined by the local x-axes of the *Joint*.

The twist axis is initiated as the x-axis of the *Joint*.

Once the Bodies swing, the twist axis is calculated as the middle of the x-axes of the Joint in the local space of the two Bodies.

25.103 ConfigFile

Inherits: Reference < Object

Category: Core

25.103.1 Brief Description

Helper class to handle INI-style files.

25.103.2 Member Functions

void	<i>erase_section (String section)</i>
<i>PoolStringArray</i>	<i>get_section_keys (String section) const</i>
<i>PoolStringArray</i>	<i>get_sections () const</i>
<i>Variant</i>	<i>get_value (String section, String key, Variant default=null) const</i>
<i>bool</i>	<i>has_section (String section) const</i>
<i>bool</i>	<i>has_section_key (String section, String key) const</i>
<i>int</i>	<i>load (String path)</i>
<i>int</i>	<i>save (String path)</i>
void	<i>set_value (String section, String key, Variant value)</i>

25.103.3 Description

This helper class can be used to store *Variant* values on the filesystem using INI-style formatting. The stored values are identified by a section and a key:

```
[section]
some_key=42
string_example="Hello World!"
a_vector=Vector3( 1, 0, 2 )
```

The stored data can be saved to or parsed from a file, though ConfigFile objects can also be used directly without accessing the filesystem.

The following example shows how to parse an INI-style file from the system, read its contents and store new values in it:

```
var config = ConfigFile.new()
var err = config.load("user://settings.cfg")
if err == OK: # if not, something went wrong with the file loading
    # Look for the display/width pair, and default to 1024 if missing
    var screen_width = get_value("display", "width", 1024)
    # Store a variable if and only if it hasn't been defined yet
    if not config.has_section_key("audio", "mute"):
        config.set_value("audio", "mute", false)
    # Save the changes by overwriting the previous file
    config.save("user://settings.cfg")
```

25.103.4 Member Function Description

- `void erase_section (String section)`

Deletes the specified section along with all the key-value pairs inside.

- `PoolStringArray get_section_keys (String section) const`

Returns an array of all defined key identifiers in the specified section.

- `PoolStringArray get_sections () const`

Returns an array of all defined section identifiers.

- `Variant get_value (String section, String key, Variant default=null) const`

Returns the current value for the specified section and key. If the section and/or the key do not exist, the method returns the value of the optional `default` argument, or `null` if it is omitted.

- `bool has_section (String section) const`

Returns `true` if the specified section exists.

- `bool has_section_key (String section, String key) const`

Returns `true` if the specified section-key pair exists.

- `int load (String path)`

Loads the config file specified as a parameter. The file's contents are parsed and loaded in the `ConfigFile` object which the method was called on. Returns one of the `OK`, `FAILED` or `ERR__*` constants listed in `@GlobalScope`. If the load was successful, the return value is `OK`.

- `int save (String path)`

Saves the contents of the `ConfigFile` object to the file specified as a parameter. The output file uses an INI-style structure. Returns one of the `OK`, `FAILED` or `ERR__*` constants listed in `@GlobalScope`. If the load was successful, the return value is `OK`.

- `void set_value (String section, String key, Variant value)`

Assigns a value to the specified key of the the specified section. If the section and/or the key do not exist, they are created. Passing a `null` value deletes the specified key if it exists, and deletes the section if it ends up empty once the key has been removed.

25.104 ConfirmationDialog

Inherits: `AcceptDialog < WindowDialog < Popup < Control < CanvasItem < Node < Object`

Inherited By: `EditorFileDialog, FileDialog`

Category: Core

25.104.1 Brief Description

Dialog for confirmation of actions.

25.104.2 Member Functions

<i>Button</i>	<code>get_cancel()</code>
---------------	---------------------------

25.104.3 Description

Dialog for confirmation of actions. This dialog inherits from *AcceptDialog*, but has by default an OK and Cancel button (in host OS order).

25.104.4 Member Function Description

- *Button get_cancel()*

Return the cancel button.

25.105 Container

Inherits: *Control < CanvasItem < Node < Object*

Inherited By: *PanelContainer, GridContainer, ScrollContainer, ViewportContainer, MarginContainer, CenterContainer, GraphNode, SplitContainer, BoxContainer*

Category: Core

25.105.1 Brief Description

Base node for containers.

25.105.2 Member Functions

<code>void</code>	<code>fit_child_in_rect (Control child, Rect2 rect)</code>
<code>void</code>	<code>queue_sort()</code>

25.105.3 Signals

- *sort_children()*

Emitted when sorting the children is needed.

25.105.4 Numeric Constants

- **NOTIFICATION_SORT_CHILDREN = 50** — Notification for when sorting the children, it must be obeyed immediately.

25.105.5 Description

Base node for containers. A `Container` contains other controls and automatically arranges them in a certain way.

A Control can inherit this to create custom container classes.

25.105.6 Member Function Description

- void `fit_child_in_rect (Control child, Rect2 rect)`

Fit a child control in a given rect. This is mainly a helper for creating custom container classes.

- void `queue_sort ()`

Queue resort of the contained children. This is called automatically anyway, but can be called upon request.

25.106 Control

Inherits: `CanvasItem < Node < Object`

Inherited By: `TextureRect, ColorRect, Label, Tabs, GraphEdit, VideoPlayer, NinePatchRect, LineEdit, Container, TextEdit, BaseButton, Popup, Tree, Separator, ReferenceRect, Panel, TabContainer, Range, RichTextLabel, ItemList`

Category: Core

25.106.1 Brief Description

All User Interface nodes inherit from Control. Features anchors and margins to adapt its position and size to its parent.

25.106.2 Member Functions

<code>Vector2</code>	<code>_get_minimum_size () virtual</code>
<code>void</code>	<code>_gui_input (InputEvent event) virtual</code>
<code>void</code>	<code>accept_event ()</code>
<code>void</code>	<code>add_color_override (String name, Color color)</code>
<code>void</code>	<code>add_constant_override (String name, int constant)</code>
<code>void</code>	<code>add_font_override (String name, Font font)</code>
<code>void</code>	<code>add_icon_override (String name, Texture texture)</code>
<code>void</code>	<code>add_shader_override (String name, Shader shader)</code>
<code>void</code>	<code>add_stylebox_override (String name, StyleBox stylebox)</code>
<code>bool</code>	<code>can_drop_data (Vector2 position, Variant data) virtual</code>
<code>void</code>	<code>drop_data (Vector2 position, Variant data) virtual</code>
<code>void</code>	<code>force_drag (Variant data, Control preview)</code>
<code>Vector2</code>	<code>get_begin () const</code>
<code>Color</code>	<code>get_color (String name, String type="") const</code>
<code>Vector2</code>	<code>get_combined_minimum_size () const</code>
<code>int</code>	<code>get_constant (String name, String type="") const</code>
<code>int</code>	<code>get_cursor_shape (Vector2 position=Vector2(0, 0)) const</code>
<code>Object</code>	<code>get_drag_data (Vector2 position) virtual</code>
<code>Vector2</code>	<code>get_end () const</code>

Continued on next page

Table 25.7 – continued from previous page

<i>Control</i>	<code>get_focus_owner () const</code>
<i>Font</i>	<code>get_font (String name, String type="") const</code>
<i>Rect2</i>	<code>get_global_rect () const</code>
<i>Texture</i>	<code>get_icon (String name, String type="") const</code>
<i>Vector2</i>	<code>get_minimum_size () const</code>
<i>Vector2</i>	<code>get_parent_area_size () const</code>
<i>Control</i>	<code>get_parent_control () const</code>
<i>Rect2</i>	<code>get_rect () const</code>
<i>float</i>	<code>get_rotation () const</code>
<i>StyleBox</i>	<code>get_stylebox (String name, String type="") const</code>
<i>String</i>	<code>get_tooltip (Vector2 at_position=Vector2(0, 0)) const</code>
<i>void</i>	<code>grab_click_focus ()</code>
<i>void</i>	<code>grab_focus ()</code>
<i>bool</i>	<code>has_color (String name, String type="") const</code>
<i>bool</i>	<code>has_color_override (String name) const</code>
<i>bool</i>	<code>has_constant (String name, String type="") const</code>
<i>bool</i>	<code>has_constant_override (String name) const</code>
<i>bool</i>	<code>has_focus () const</code>
<i>bool</i>	<code>has_font (String name, String type="") const</code>
<i>bool</i>	<code>has_font_override (String name) const</code>
<i>bool</i>	<code>has_icon (String name, String type="") const</code>
<i>bool</i>	<code>has_icon_override (String name) const</code>
<i>bool</i>	<code>has_point (Vector2 point) virtual</code>
<i>bool</i>	<code>has_shader_override (String name) const</code>
<i>bool</i>	<code>has_stylebox (String name, String type="") const</code>
<i>bool</i>	<code>has_stylebox_override (String name) const</code>
<i>void</i>	<code>minimum_size_changed ()</code>
<i>void</i>	<code>release_focus ()</code>
<i>void</i>	<code>set_anchor (int margin, float anchor, bool keep_margin=false, bool push_opposite_anchor=true)</code>
<i>void</i>	<code>set_anchor_and_margin (int margin, float offset, bool push_opposite_anchor=false)</code>
<i>void</i>	<code>set_anchors_and_margins_preset (int preset, int resize_mode=0, int margin=0)</code>
<i>void</i>	<code>set_anchors_preset (int preset, bool keep_margin=false)</code>
<i>void</i>	<code>set_begin (Vector2 position)</code>
<i>void</i>	<code>set_drag_forwarding (Control target)</code>
<i>void</i>	<code>set_drag_preview (Control control)</code>
<i>void</i>	<code>set_end (Vector2 position)</code>
<i>void</i>	<code>set_margins_preset (int preset, int resize_mode=0, int margin=0)</code>
<i>void</i>	<code>set_rotation (float radians)</code>
<i>void</i>	<code>show_modal (bool exclusive=false)</code>
<i>void</i>	<code>warp_mouse (Vector2 to_position)</code>

25.106.3 Signals

- **focus_entered ()**

Emitted when the node gains keyboard focus.

- **focus_exited ()**

Emitted when the node loses keyboard focus.

- **gui_input (Object ev)**

Emitted when the node receives an *InputEvent*.

- **minimum_size_changed ()**

Emitted when the node's minimum size changes.

- **modal_closed ()**

Emitted when a modal `Control` is closed. See `show_modal`.

- **mouse_entered ()**

Emitted when the mouse enters the control's `Rect` area, provided its `mouse_filter` lets the event reach it.

- **mouse_exited ()**

Emitted when the mouse leaves the control's `Rect` area, provided its `mouse_filter` lets the event reach it.

- **resized ()**

Emitted when the control changes size.

- **size_flags_changed ()**

Emitted when one of the size flags changes. See `size_flags_horizontal` and `size_flags_vertical`.

25.106.4 Member Variables

- *float anchor_bottom* - Anchors the bottom edge of the node to the origin, the center, or the end of its parent container. It changes how the bottom margin updates when the node moves or changes size. Use one of the `ANCHOR__*` constants. Default value: `ANCHOR_BEGIN`.
- *float anchor_left* - Anchors the left edge of the node to the origin, the center or the end of its parent container. It changes how the left margin updates when the node moves or changes size. Use one of the `ANCHOR__*` constants. Default value: `ANCHOR_BEGIN`.
- *float anchor_right* - Anchors the right edge of the node to the origin, the center or the end of its parent container. It changes how the right margin updates when the node moves or changes size. Use one of the `ANCHOR__*` constants. Default value: `ANCHOR_BEGIN`.
- *FocusMode focus_mode* - The focus access mode for the control (None, Click or All). Only one `Control` can be focused at the same time, and it will receive keyboard signals.
- *NodePath focus_neighbour_bottom* - Tells Godot which node it should give keyboard focus to if the user presses Tab, the down arrow on the keyboard, or down on a gamepad. The node must be a `Control`. If this property is not set, Godot will give focus to the closest `Control` to the bottom of this one.

If the user presses Tab, Godot will give focus to the closest node to the right first, then to the bottom. If the user presses Shift+Tab, Godot will look to the left of the node, then above it.

- *NodePath focus_neighbour_left* - Tells Godot which node it should give keyboard focus to if the user presses Shift+Tab, the left arrow on the keyboard or left on a gamepad. The node must be a `Control`. If this property is not set, Godot will give focus to the closest `Control` to the left of this one.
- *NodePath focus_neighbour_right* - Tells Godot which node it should give keyboard focus to if the user presses Tab, the right arrow on the keyboard or right on a gamepad. The node must be a `Control`. If this property is not set, Godot will give focus to the closest `Control` to the bottom of this one.

- *NodePath* **focus_neighbour_top** - Tells Godot which node it should give keyboard focus to if the user presses Shift+Tab, the top arrow on the keyboard or top on a gamepad. The node must be a `Control`. If this property is not set, Godot will give focus to the closest `Control` to the bottom of this one.
- *NodePath* **focus_next**
- *NodePath* **focus_previous**
- *GrowDirection* **grow_horizontal**
- *GrowDirection* **grow_vertical**
- *String* **hint_tooltip** - Changes the tooltip text. The tooltip appears when the user's mouse cursor stays idle over this control for a few moments.
- *float* **margin_bottom** - Distance between the node's bottom edge and its parent container, based on *anchor_bottom*.

Margins are often controlled by one or multiple parent *Container* nodes. Margins update automatically when you move or resize the node.

- *float* **margin_left** - Distance between the node's left edge and its parent container, based on *anchor_left*.
- *float* **margin_right** - Distance between the node's right edge and its parent container, based on *anchor_right*.
- *float* **margin_top** - Distance between the node's top edge and its parent container, based on *anchor_top*.
- *CursorShape* **mouse_default_cursor_shape** - The default cursor shape for this control. Useful for Godot plugins and applications or games that use the system's mouse cursors.
- *MouseFilter* **mouse_filter** - Controls whether the control will be able to receive mouse button input events through *_gui_input* and how these events should be handled. Use one of the `MOUSE_FILTER__*` constants. See the constants to learn what each does.
- *bool* **rect_clip_content**
- *Vector2* **rect_global_position** - The node's global position, relative to the world (usually to the top-left corner of the window).
- *Vector2* **rect_min_size** - The minimum size of the node's bounding rectangle. If you set it to a value greater than (0, 0), the node's bounding rectangle will always have at least this size, even if its content is smaller. If it's set to (0, 0), the node sizes automatically to fit its content, be it a texture or child nodes.
- *Vector2* **rect_pivot_offset** - By default, the node's pivot is its top-left corner. When you change its *rect_scale*, it will scale around this pivot. Set this property to *rect_size* / 2 to center the pivot in the node's rectangle.
- *Vector2* **rect_position** - The node's position, relative to its parent. It corresponds to the rectangle's top-left corner. The property is not affected by *rect_pivot_offset*.
- *float* **rect_rotation** - The node's rotation around its pivot, in degrees. See *rect_pivot_offset* to change the pivot's position.
- *Vector2* **rect_scale** - The node's scale, relative to its *rect_size*. Change this property to scale the node around its *rect_pivot_offset*.
- *Vector2* **rect_size** - The size of the node's bounding rectangle, in pixels. *Container* nodes update this property automatically.
- *int* **size_flags_horizontal** - Tells the parent *Container* nodes how they should resize and place the node on the X axis. Use one of the `SIZE__*` constants to change the flags. See the constants to learn what each does.

- *float size_flags_stretch_ratio* - If the node and at least one of its neighbours uses the SIZE_EXPAND size flag, the parent *Container* will let it take more or less space depending on this property. If this node has a stretch ratio of 2 and its neighbour a ratio of 1, this node will take two thirds of the available space.
- *int size_flags_vertical* - Tells the parent *Container* nodes how they should resize and place the node on the Y axis. Use one of the SIZE_* constants to change the flags. See the constants to learn what each does.
- *Theme theme* - Changing this property replaces the current *Theme* resource this node and all its Control children use.

25.106.5 Numeric Constants

- **NOTIFICATION_RESIZED = 40** — Sent when the node changes size. Use *rect_size* to get the new size.
- **NOTIFICATION_MOUSE_ENTER = 41** — Sent when the mouse pointer enters the node's Rect area.
- **NOTIFICATION_MOUSE_EXIT = 42** — Sent when the mouse pointer exits the node's Rect area.
- **NOTIFICATION_FOCUS_ENTER = 43** — Sent when the node grabs focus.
- **NOTIFICATION_FOCUS_EXIT = 44** — Sent when the node loses focus.
- **NOTIFICATION_THEME_CHANGED = 45** — Sent when the node's *theme* changes, right before Godot redraws the Control. Happens when you call one of the add_*_override
- **NOTIFICATION_MODAL_CLOSE = 46** — Sent when an open modal dialog closes. See *show_modal*.

25.106.6 Enums

enum **SizeFlags**

- **SIZE_FILL = 1** — Tells the parent *Container* to expand the bounds of this node to fill all the available space without pushing any other node. Use with *size_flags_horizontal* and *size_flags_vertical*.
- **SIZE_EXPAND = 2** — Tells the parent *Container* to let this node take all the available space on the axis you flag. If multiple neighboring nodes are set to expand, they'll share the space based on their stretch ratio. See *size_flags_stretch_ratio*. Use with *size_flags_horizontal* and *size_flags_vertical*.
- **SIZE_EXPAND_FILL = 3** — Sets the node's size flags to both fill and expand. See the 2 constants above for more information.
- **SIZE_SHRINK_CENTER = 4** — Tells the parent *Container* to center the node in itself. It centers the Control based on its bounding box, so it doesn't work with the fill or expand size flags. Use with *size_flags_horizontal* and *size_flags_vertical*.
- **SIZE_SHRINK_END = 8** — Tells the parent *Container* to align the node with its end, either the bottom or the right edge. It doesn't work with the fill or expand size flags. Use with *size_flags_horizontal* and *size_flags_vertical*.

enum **CursorShape**

- **CURSOR_ARROW = 0** — Show the system's arrow mouse cursor when the user hovers the node. Use with *set_default_cursor_shape*.
- **CURSOR_IBEAM = 1** — Show the system's I-beam mouse cursor when the user hovers the node. The I-beam pointer has a shape similar to "I". It tells the user they can highlight or insert text.
- **CURSOR_POINTING_HAND = 2** — Show the system's pointing hand mouse cursor when the user hovers the node.

- **CURSOR_CROSS = 3** — Show the system's cross mouse cursor when the user hovers the node.
- **CURSOR_WAIT = 4** — Show the system's wait mouse cursor, often an hourglass, when the user hovers the node.
- **CURSOR_BUSY = 5** — Show the system's busy mouse cursor when the user hovers the node. Often an hourglass.
- **CURSOR_DRAG = 6** — Show the system's drag mouse cursor, often a closed fist or a cross symbol, when the user hovers the node. It tells the user they're currently dragging an item, like a node in the Scene dock.
- **CURSOR_CAN_DROP = 7** — Show the system's drop mouse cursor when the user hovers the node. It can be an open hand. It tells the user they can drop an item they're currently grabbing, like a node in the Scene dock.
- **CURSOR_FORBIDDEN = 8** — Show the system's forbidden mouse cursor when the user hovers the node. Often a crossed circle.
- **CURSOR_VSIZE = 9** — Show the system's vertical resize mouse cursor when the user hovers the node. A double headed vertical arrow. It tells the user they can resize the window or the panel vertically.
- **CURSOR_HSIZE = 10** — Show the system's horizontal resize mouse cursor when the user hovers the node. A double headed horizontal arrow. It tells the user they can resize the window or the panel horizontally.
- **CURSOR_BDIAGSIZE = 11** — Show the system's window resize mouse cursor when the user hovers the node. The cursor is a double headed arrow that goes from the bottom left to the top right. It tells the user they can resize the window or the panel both horizontally and vertically.
- **CURSOR_FDIAGSIZE = 12** — Show the system's window resize mouse cursor when the user hovers the node. The cursor is a double headed arrow that goes from the top left to the bottom right, the opposite of CURSOR_BDIAGSIZE. It tells the user they can resize the window or the panel both horizontally and vertically.
- **CURSOR_MOVE = 13** — Show the system's move mouse cursor when the user hovers the node. It shows 2 double-headed arrows at a 90 degree angle. It tells the user they can move a UI element freely.
- **CURSOR_VSPLIT = 14** — Show the system's vertical split mouse cursor when the user hovers the node. On Windows, it's the same as CURSOR_VSIZE.
- **CURSOR_HSPLIT = 15** — Show the system's horizontal split mouse cursor when the user hovers the node. On Windows, it's the same as CURSOR_HSIZE.
- **CURSOR_HELP = 16** — Show the system's help mouse cursor when the user hovers the node, a question mark.

enum FocusMode

- **FOCUS_NONE = 0** — The node cannot grab focus. Use with `set_focus_mode`.
- **FOCUS_CLICK = 1** — The node can only grab focus on mouse clicks. Use with `set_focus_mode`.
- **FOCUS_ALL = 2** — The node can grab focus on mouse click or using the arrows and the Tab keys on the keyboard. Use with `set_focus_mode`.

enum GrowDirection

- **GROW_DIRECTION_BEGIN = 0**
- **GROW_DIRECTION_END = 1**

enum LayoutPresetMode

- **PRESET_MODE_MINSIZE = 0**
- **PRESET_MODE_KEEP_WIDTH = 1**
- **PRESET_MODE_KEEP_HEIGHT = 2**

- **PRESET_MODE_KEEP_SIZE = 3**

enum **LayoutPreset**

- **PRESET_TOP_LEFT = 0** — Snap all 4 anchors to the top-left of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_TOP_RIGHT = 1** — Snap all 4 anchors to the top-right of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_BOTTOM_LEFT = 2** — Snap all 4 anchors to the bottom-left of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_BOTTOM_RIGHT = 3** — Snap all 4 anchors to the bottom-right of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_CENTER_LEFT = 4** — Snap all 4 anchors to the center of the left edge of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_CENTER_TOP = 5** — Snap all 4 anchors to the center of the top edge of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_CENTER_RIGHT = 6** — Snap all 4 anchors to the center of the right edge of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_CENTER_BOTTOM = 7** — Snap all 4 anchors to the center of the bottom edge of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_CENTER = 8** — Snap all 4 anchors to the center of the parent container's bounds. Use with *set_anchors_preset*.
- **PRESET_LEFT_WIDE = 9** — Snap all 4 anchors to the left edge of the parent container. The left margin becomes relative to the left edge and the top margin relative to the top left corner of the node's parent. Use with *set_anchors_preset*.
- **PRESET_TOP_WIDE = 10** — Snap all 4 anchors to the top edge of the parent container. The left margin becomes relative to the top left corner, the top margin relative to the top edge, and the right margin relative to the top right corner of the node's parent. Use with *set_anchors_preset*.
- **PRESET_RIGHT_WIDE = 11** — Snap all 4 anchors to the right edge of the parent container. The right margin becomes relative to the right edge and the top margin relative to the top right corner of the node's parent. Use with *set_anchors_preset*.
- **PRESET_BOTTOM_WIDE = 12** — Snap all 4 anchors to the bottom edge of the parent container. The left margin becomes relative to the bottom left corner, the bottom margin relative to the bottom edge, and the right margin relative to the bottom right corner of the node's parent. Use with *set_anchors_preset*.
- **PRESET_VCENTER_WIDE = 13** — Snap all 4 anchors to a vertical line that cuts the parent container in half. Use with *set_anchors_preset*.
- **PRESET_HCENTER_WIDE = 14** — Snap all 4 anchors to a horizontal line that cuts the parent container in half. Use with *set_anchors_preset*.
- **PRESET_WIDE = 15** — Snap all 4 anchors to the respective corners of the parent container. Set all 4 margins to 0 after you applied this preset and the `Control` will fit its parent container. Use with *set_anchors_preset*.

enum **MouseFilter**

- **MOUSE_FILTER_STOP = 0** — The control will receive mouse button input events through `_gui_input` if clicked on. These events are automatically marked as handled and they will not propagate further to other controls.
- **MOUSE_FILTER_PASS = 1** — The control will receive mouse button input events through `_gui_input` if clicked on. If this control does not handle the event, the parent control (if any) will be considered for a mouse

click, and so on until there is no more parent control to potentially handle it. Even if no control handled it at all, the event will still be handled automatically.

- **MOUSE_FILTER_IGNORE = 2** — The control will not receive mouse button input events through `_gui_input` and will not block other controls from receiving these events. These events will also not be handled automatically.

enum Anchor

- **ANCHOR_BEGIN = 0** — Snaps one of the 4 anchor's sides to the origin of the node's `Rect`, in the top left. Use it with one of the `anchor__*` member variables, like `anchor_left`. To change all 4 anchors at once, use `set_anchors_preset`.
- **ANCHOR_END = 1** — Snaps one of the 4 anchor's sides to the end of the node's `Rect`, in the bottom right. Use it with one of the `anchor__*` member variables, like `anchor_left`. To change all 4 anchors at once, use `set_anchors_preset`.

25.106.7 Description

Base class for all User Interface or *UI* related nodes. `Control` features a bounding rectangle that defines its extents, an anchor position relative to its parent and margins that represent an offset to the anchor. The margins update automatically when the node, any of its parents, or the screen size change.

For more information on Godot's UI system, anchors, margins, and containers, see the related tutorials in the manual. To build flexible UIs, you'll need a mix of UI elements that inherit from `Control` and `Container` nodes.

User Interface nodes and input

Godot sends input events to the scene's root node first, by calling `Node._input`. `Node._input` forwards the event down the node tree to the nodes under the mouse cursor, or on keyboard focus. To do so, it calls `MainLoop._input_event`. Call `accept_event` so no other node receives the event. Once you accepted an input, it becomes handled so `Node._unhandled_input` will not process it.

Only one `Control` node can be in keyboard focus. Only the node in focus will receive keyboard events. To get the focus, call `set_focus_mode`. `Control` nodes lose focus when another node grabs it, or if you hide the node in focus.

Call `set_ignore_mouse` to tell a `Control` node to ignore mouse or touch events. You'll need it if you place an icon on top of a button.

Theme resources change the `Control`'s appearance. If you change the *Theme* on a `Control` node, it affects all of its children. To override some of the theme's parameters, call one of the `add__*_override` methods, like `add_font_override`. You can override the theme with the inspector.

25.106.8 Member Function Description

- `Vector2 _get_minimum_size () virtual`

Returns the minimum size this `Control` can shrink to. The node can never be smaller than this minimum size.

- `void _gui_input (InputEvent event) virtual`

The node's parent forwards input events to this method. Use it to process and accept inputs on UI elements. See `accept_event`.

Replaces Godot 2's `_input_event`.

- `void accept_event ()`

Marks an input event as handled. Once you accept an input event, it stops propagating, even to nodes listening to `Node._unhandled_input` or `Node._unhandled_key_input`.

- void **add_color_override** (*String name, Color color*)

Overrides the color in the theme resource the node uses.

- void **add_constant_override** (*String name, int constant*)

Overrides an integer constant in the *Theme* resource the node uses. If the constant is invalid, Godot clears the override. See `Theme.INVALID_CONSTANT` for more information.

- void **add_font_override** (*String name, Font font*)

Overrides the name font in the theme resource the node uses. If `font` is empty, Godot clears the override.

- void **add_icon_override** (*String name, Texture texture*)

Overrides the name icon in the theme resource the node uses. If `icon` is empty, Godot clears the override.

- void **add_shader_override** (*String name, Shader shader*)

Overrides the name shader in the theme resource the node uses. If `shader` is empty, Godot clears the override.

- void **add_stylebox_override** (*String name, StyleBox stylebox*)

Overrides the name Stylebox in the theme resource the node uses. If `stylebox` is empty, Godot clears the override.

- *bool can_drop_data* (*Vector2 position, Variant data*) virtual

- void **drop_data** (*Vector2 position, Variant data*) virtual

- void **force_drag** (*Variant data, Control preview*)

- *Vector2 get_begin* () const

- *Color get_color* (*String name, String type=""*) const

- *Vector2 get_combined_minimum_size* () const

- *int get_constant* (*String name, String type=""*) const

- *int get_cursor_shape* (*Vector2 position=Vector2(0, 0)*) const

Returns the mouse cursor shape the control displays on mouse hover, one of the `CURSOR_*` constants.

- *Object get_drag_data* (*Vector2 position*) virtual

- *Vector2 get_end* () const

Returns `MARGIN_LEFT` and `MARGIN_TOP` at the same time. This is a helper (see `set_margin`).

- *Control get_focus_owner* () const

Return which control is owning the keyboard focus, or null if no one.

- *Font get_font* (*String name, String type=""*) const

- *Rect2 get_global_rect* () const

Return position and size of the Control, relative to the top-left corner of the *window* Control. This is a helper (see `get_global_position`, `get_size`).

- *Texture get_icon* (*String name, String type=""*) const

- *Vector2 get_minimum_size* () const

Return the minimum size this Control can shrink to. A control will never be displayed or resized smaller than its minimum size.

- *Vector2 get_parent_area_size* () const

- *Control get_parent_control* () const

- `Rect2 get_rect() const`

Return position and size of the Control, relative to the top-left corner of the parent Control. This is a helper (see `get_position`, `get_size`).

- `float get_rotation() const`

Return the rotation (in radians)

- `StyleBox get_stylebox(String name, String type="") const`
- `String get_tooltip(Vector2 at_position=Vector2(0, 0)) const`

Return the tooltip, which will appear when the cursor is resting over this control.

- `void grab_click_focus()`
- `void grab_focus()`

Steal the focus from another control and become the focused control (see `set_focus_mode`).

- `bool has_color(String name, String type="") const`
- `bool has_color_override(String name) const`
- `bool has_constant(String name, String type="") const`
- `bool has_constant_override(String name) const`
- `bool has_focus()` const

Return whether the Control is the current focused control (see `set_focus_mode`).

- `bool has_font(String name, String type="") const`
- `bool has_font_override(String name) const`
- `bool has_icon(String name, String type="") const`
- `bool has_icon_override(String name) const`
- `bool has_point(Vector2 point) virtual`
- `bool has_shader_override(String name) const`
- `bool has_stylebox(String name, String type="") const`
- `bool has_stylebox_override(String name) const`
- `void minimum_size_changed()`
- `void release_focus()`

Give up the focus, no other control will be able to receive keyboard input.

- `void set_anchor(int margin, float anchor, bool keep_margin=false, bool push_opposite_anchor=true)`
- `void set_anchor_and_margin(int margin, float anchor, float offset, bool push_opposite_anchor=false)`
- `void set_anchors_and_margins_preset(int preset, int resize_mode=0, int margin=0)`
- `void set_anchors_preset(int preset, bool keep_margin=false)`
- `void set_begin(Vector2 position)`

Sets `MARGIN_LEFT` and `MARGIN_TOP` at the same time. This is a helper (see `set_margin`).

- `void set_drag_forwarding(Control target)`
- `void set_drag_preview(Control control)`

- void **set_end** (*Vector2* position)

Sets MARGIN_RIGHT and MARGIN_BOTTOM at the same time. This is a helper (see `set_margin`).

- void **set_margins_preset** (*int* preset, *int* resize_mode=0, *int* margin=0)
- void **set_rotation** (*float* radians)

Set the rotation (in radians).

- void **show_modal** (*bool* exclusive=false)

Display a Control as modal. Control must be a subwindow. Modal controls capture the input signals until closed or the area outside them is accessed. When a modal control loses focus, or the ESC key is pressed, they automatically hide. Modal controls are used extensively for popup dialogs and menus.

- void **warp_mouse** (*Vector2* to_position)

25.107 ConvexPolygonShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

25.107.1 Brief Description

Convex polygon shape for 3D physics.

25.107.2 Member Variables

- *PoolVector3Array* **points** - The list of 3D points forming the convex polygon shape.

25.107.3 Description

Convex polygon shape resource, which can be added to a *PhysicsBody* or area.

25.108 ConvexPolygonShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

25.108.1 Brief Description

Convex Polygon Shape for 2D physics.

25.108.2 Member Functions

void	set_point_cloud (<i>PoolVector2Array</i> point_cloud)
------	--

25.108.3 Member Variables

- *PoolVector2Array points* - The polygon's list of vertices. Can be in either clockwise or counterclockwise order.

25.108.4 Description

Convex Polygon Shape for 2D physics. A convex polygon, whatever its shape, is internally decomposed into as many convex polygons as needed to ensure all collision checks against it are always done on convex polygons (which are faster to check).

The main difference between a `ConvexPolygonShape2D` and a `ConcavePolygonShape2D` is that a concave polygon assumes it is concave and uses a more complex method of collision detection, and a convex one forces itself to be convex in order to speed up collision detection.

25.108.5 Member Function Description

- void `set_point_cloud (PoolVector2Array point_cloud)`

Currently, this method does nothing.

25.109 CSharpScript

Inherits: `Script < Resource < Reference < Object`

Category: Core

25.109.1 Brief Description

25.109.2 Member Functions

<code>Object</code>	<code>new () vararg</code>
---------------------	----------------------------

25.109.3 Member Function Description

- `Object new () vararg`

25.110 CubeMap

Inherits: `Resource < Reference < Object`

Category: Core

25.110.1 Brief Description

A CubeMap is a 6 sided 3D texture.

25.110.2 Member Functions

<i>int</i>	<i>get_height () const</i>
<i>Image</i>	<i>get_side (int side) const</i>
<i>int</i>	<i>get_width () const</i>
<i>void</i>	<i>set_side (int side, Image image)</i>

25.110.3 Member Variables

- *int flags* - The render flags for the CubeMap. See the FLAG_* constants for details.
- *float lossy_storage_quality* - The lossy storage quality of the CubeMap if the storage mode is set to STORAGE_COMPRESS_LOSSY.
- *Storage storage_mode* - The CubeMap's storage mode. See STORAGE_* constants.

25.110.4 Enums

enum Flags

- **FLAG_MIPMAPS = 1** — Generate mipmaps, to enable smooth zooming out of the texture.
- **FLAG_REPEAT = 2** — Repeat (instead of clamp to edge).
- **FLAG_FILTER = 4** — Turn on magnifying filter, to enable smooth zooming in of the texture.
- **FLAGS_DEFAULT = 7** — Default flags. Generate mipmaps, repeat, and filter are enabled.

enum Storage

- **STORAGE_RAW = 0** — Store the CubeMap without any compression.
- **STORAGE_COMPRESS_LOSSY = 1** — Store the CubeMap with strong compression that reduces image quality.
- **STORAGE_COMPRESS_LOSSLESS = 2** — Store the CubeMap with moderate compression that doesn't reduce image quality.

enum Side

- **SIDE_LEFT = 0** — Identifier for the left face of the CubeMap.
- **SIDE_RIGHT = 1** — Identifier for the right face of the CubeMap.
- **SIDE_BOTTOM = 2** — Identifier for the bottom face of the CubeMap.
- **SIDE_TOP = 3** — Identifier for the top face of the CubeMap.
- **SIDE_FRONT = 4** — Identifier for the front face of the CubeMap.
- **SIDE_BACK = 5** — Identifier for the back face of the CubeMap.

25.110.5 Description

A 6-sided 3D texture typically used for faking reflections. It can be used to make an object look as if it's reflecting its surroundings. This usually delivers much better performance than other reflection methods.

25.110.6 Member Function Description

- `int get_height() const`

Returns the CubeMap's height.

- `Image get_side(int side) const`

Returns an *Image* for a side of the CubeMap using one of the `SIDE__*` constants or an integer 0-5.

- `int get_width() const`

Returns the CubeMap's width.

- `void set_side(int side, Image image)`

Sets an *Image* for a side of the CubeMap using one of the `SIDE__*` constants or an integer 0-5.

25.111 CubeMesh

Inherits: `PrimitiveMesh < Mesh < Resource < Reference < Object`

Category: Core

25.111.1 Brief Description

Generate an axis-aligned cuboid *PrimitiveMesh*.

25.111.2 Member Variables

- `Vector3 size` - Size of the cuboid mesh. Defaults to (2, 2, 2).
- `int subdivide_depth` - Number of extra edge loops inserted along the z-axis. Defaults to 0.
- `int subdivide_height` - Number of extra edge loops inserted along the y-axis. Defaults to 0.
- `int subdivide_width` - Number of extra edge loops inserted along the x-axis. Defaults to 0.

25.111.3 Description

Generate an axis-aligned cuboid *PrimitiveMesh*.

25.112 Curve

Inherits: `Resource < Reference < Object`

Category: Core

25.112.1 Brief Description

A mathematic curve.

25.112.2 Member Functions

<i>int</i>	<code>add_point (Vector2 position, float left_tangent=0, float right_tangent=0, int left_mode=0, int right_mode=0)</code>
<i>void</i>	<code>bake ()</code>
<i>void</i>	<code>clean_dupes ()</code>
<i>void</i>	<code>clear_points ()</code>
<i>int</i>	<code>get_point_left_mode (int index) const</code>
<i>float</i>	<code>get_point_left_tangent (int index) const</code>
<i>Vector2</i>	<code>get_point_position (int index) const</code>
<i>int</i>	<code>get_point_right_mode (int index) const</code>
<i>float</i>	<code>get_point_right_tangent (int index) const</code>
<i>float</i>	<code>interpolate (float offset) const</code>
<i>float</i>	<code>interpolate_baked (float offset)</code>
<i>void</i>	<code>remove_point (int index)</code>
<i>void</i>	<code>set_point_left_mode (int index, int mode)</code>
<i>void</i>	<code>set_point_left_tangent (int index, float tangent)</code>
<i>int</i>	<code>set_point_offset (int index, float offset)</code>
<i>void</i>	<code>set_point_right_mode (int index, int mode)</code>
<i>void</i>	<code>set_point_right_tangent (int index, float tangent)</code>
<i>void</i>	<code>set_point_value (int index, float y)</code>

25.112.3 Signals

- **range_changed ()**

Emitted when *max_value* or *min_value* is changed.

25.112.4 Member Variables

- *int bake_resolution* - The number of points to include in the baked (i.e. cached) curve data.
- *float max_value* - The maximum value the curve can reach. Default value: 1.
- *float min_value* - The minimum value the curve can reach. Default value: 0.

25.112.5 Enums

enum TangentMode

- **TANGENT_FREE = 0** — The tangent on this side of the point is user-defined.
- **TANGENT_LINEAR = 1** — The curve calculates the tangent on this side of the point as the slope halfway towards the adjacent point.
- **TANGENT_MODE_COUNT = 2** — The total number of available tangent modes.

25.112.6 Description

A curve that can be saved and re-used for other objects. By default it ranges between 0 and 1 on the y-axis and positions points relative to the 0.5 y-position.

25.112.7 Member Function Description

- `int add_point (Vector2 position, float left_tangent=0, float right_tangent=0, int left_mode=0, int right_mode=0)`

Adds a point to the curve. For each side, if the `*_mode` is TANGENT_LINEAR, the `*_tangent` angle (in degrees) uses the slope of the curve halfway to the adjacent point. Allows custom assignments to the `*_tangent` angle if `*_mode` is set to TANGENT_FREE.

- `void bake ()`

Recomputes the baked cache of points for the curve.

- `void clean_dups ()`

Removes points that are closer than CMP_EPSILON (0.00001) units to their neighbor on the curve.

- `void clear_points ()`

Removes all points from the curve.

- `int get_point_left_mode (int index) const`

Returns the left TangentMode for the point at `index`.

- `float get_point_left_tangent (int index) const`

Returns the left tangent angle (in degrees) for the point at `index`.

- `Vector2 get_point_position (int index) const`

Returns the curve coordinates for the point at `index`.

- `int get_point_right_mode (int index) const`

Returns the right TangentMode for the point at `index`.

- `float get_point_right_tangent (int index) const`

Returns the right tangent angle (in degrees) for the point at `index`.

- `float interpolate (float offset) const`

Returns the y value for the point that would exist at x-position `offset` along the curve.

- `float interpolate_baked (float offset)`

Returns the y value for the point that would exist at x-position `offset` along the curve using the baked cache. Bakes the curve's points if not already baked.

- `void remove_point (int index)`

Removes the point at `index` from the curve.

- `void set_point_left_mode (int index, int mode)`

Sets the left TangentMode for the point at `index` to `mode`.

- `void set_point_left_tangent (int index, float tangent)`

Sets the left tangent angle for the point at `index` to `tangent`.

- `int set_point_offset (int index, float offset)`

Sets the offset from 0..5

- `void set_point_right_mode (int index, int mode)`

Sets the right TangentMode for the point at `index` to `mode`.

- `void set_point_right_tangent (int index, float tangent)`

Sets the right tangent angle for the point at `index` to `tangent`.

- `void set_point_value (int index, float y)`

Assigns the vertical position `y` to the point at `index`.

25.113 Curve2D

Inherits: *Resource < Reference < Object*

Category: Core

25.113.1 Brief Description

Describes a Bezier curve in 2D space.

25.113.2 Member Functions

<code>void</code>	<code>add_point (Vector2 position, Vector2 in=Vector2(0, 0), Vector2 out=Vector2(0, 0), int at_position=-1)</code>
<code>void</code>	<code>clear_points ()</code>
<code>float</code>	<code>get_baked_length () const</code>
<code>PoolVector<Vector2></code>	<code>get_baked_points () const</code>
<code>int</code>	<code>get_point_count () const</code>
<code>Vector2</code>	<code>get_point_in (int idx) const</code>
<code>Vector2</code>	<code>get_point_out (int idx) const</code>
<code>Vector2</code>	<code>get_point_position (int idx) const</code>
<code>Vector2</code>	<code>interpolate (int idx, float t) const</code>
<code>Vector2</code>	<code>interpolate_baked (float offset, bool cubic=false) const</code>
<code>Vector2</code>	<code>interpolatef (float fofs) const</code>
<code>void</code>	<code>remove_point (int idx)</code>
<code>void</code>	<code>set_point_in (int idx, Vector2 position)</code>
<code>void</code>	<code>set_point_out (int idx, Vector2 position)</code>
<code>void</code>	<code>set_point_position (int idx, Vector2 position)</code>
<code>PoolVector<Vector2></code>	<code>tessellate (int max_stages=5, float tolerance_degrees=4) const</code>

25.113.3 Member Variables

- *float bake_interval* - The distance in pixels between two adjacent cached points. Changing it forces the cache to be recomputed the next time the *get_baked_points* or *get_baked_length* function is called. The smaller the distance, the more points in the cache and the more memory it will consume, so use with care.

25.113.4 Description

This class describes a Bezier curve in 2D space. It is mainly used to give a shape to a *Path2D*, but can be manually sampled for other purposes.

It keeps a cache of precalculated points along the curve, to speed further calculations up.

25.113.5 Member Function Description

- void **add_point** (*Vector2 position, Vector2 in=Vector2(0, 0), Vector2 out=Vector2(0, 0), int at_position=-1*)

Adds a point to a curve, at “position”, with control points “in” and “out”.

If “at_position” is given, the point is inserted before the point number “at_position”, moving that point (and every point after) after the inserted point. If “at_position” is not given, or is an illegal value (*at_position <0* or *at_position >= get_point_count*), the point will be appended at the end of the point list.

- void **clear_points** ()

Removes all points from the curve.

- *float get_baked_length () const*

Returns the total length of the curve, based on the cached points. Given enough density (see *set_bake_interval*), it should be approximate enough.

- *PoolVector2Array get_baked_points () const*

Returns the cache of points as a *PoolVector2Array*.

- *int get_point_count () const*

Returns the number of points describing the curve.

- *Vector2 get_point_in (int idx) const*

Returns the position of the control point leading to the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0).

- *Vector2 get_point_out (int idx) const*

Returns the position of the control point leading out of the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0).

- *Vector2 get_point_position (int idx) const*

Returns the position of the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0).

- *Vector2 interpolate (int idx, float t) const*

Returns the position between the vertex “idx” and the vertex “idx”+1, where “t” controls if the point is the first vertex (*t = 0.0*), the last vertex (*t = 1.0*), or in between. Values of “t” outside the range (*0.0 >= t <=1*) give strange, but predictable results.

If “idx” is out of bounds it is truncated to the first or last vertex, and “t” is ignored. If the curve has no points, the function sends an error to the console, and returns (0, 0).

- *Vector2 interpolate_baked (float offset, bool cubic=false) const*

Returns a point within the curve at position “offset”, where “offset” is measured as a pixel distance along the curve.

To do that, it finds the two cached points where the “offset” lies between, then interpolates the values. This interpolation is cubic if “cubic” is set to true, or linear if set to false.

Cubic interpolation tends to follow the curves better, but linear is faster (and often, precise enough).

- `Vector2 interpolatef (float fofs) const`

Returns the position at the vertex “fofs”. It calls *interpolate* using the integer part of fofs as “idx”, and its fractional part as “t”.

- `void remove_point (int idx)`

Deletes the point “idx” from the curve. Sends an error to the console if “idx” is out of bounds.

- `void set_point_in (int idx, Vector2 position)`

Sets the position of the control point leading to the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `void set_point_out (int idx, Vector2 position)`

Sets the position of the control point leading out of the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `void set_point_position (int idx, Vector2 position)`

Sets the position for the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `PoolVector2Array tessellate (int max_stages=5, float tolerance_degrees=4) const`

Returns a list of points along the curve, with a curvature controlled point density. That is, the curvier parts will have more points than the straighter parts.

This approximation makes straight segments between each point, then subdivides those segments until the resulting shape is similar enough.

“max_stages” controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

“tolerance_degrees” controls how many degrees the midpoint of a segment may deviate from the real curve, before the segment has to be subdivided.

25.114 Curve3D

Inherits: *Resource < Reference < Object*

Category: Core

25.114.1 Brief Description

Describes a Bezier curve in 3D space.

25.114.2 Member Functions

void	<code>add_point (Vector3 position, Vector3 in=Vector3(0, 0, 0), Vector3 out=Vector3(0, 0, 0), int at_position=-1)</code>
void	<code>clear_points ()</code>
float	<code>get_baked_length () const</code>
<i>PoolVec-</i> <i>tor3Array</i>	<code>get_baked_points () const</code>
<i>PoolRealArray</i>	<code>get_baked_tilts () const</code>
int	<code>get_point_count () const</code>
<i>Vector3</i>	<code>get_point_in (int idx) const</code>
<i>Vector3</i>	<code>get_point_out (int idx) const</code>
<i>Vector3</i>	<code>get_point_position (int idx) const</code>
float	<code>get_point_tilt (int idx) const</code>
<i>Vector3</i>	<code>interpolate (int idx, float t) const</code>
<i>Vector3</i>	<code>interpolate_baked (float offset, bool cubic=false) const</code>
<i>Vector3</i>	<code>interpolatef (float fofs) const</code>
void	<code>remove_point (int idx)</code>
void	<code>set_point_in (int idx, Vector3 position)</code>
void	<code>set_point_out (int idx, Vector3 position)</code>
void	<code>set_point_position (int idx, Vector3 position)</code>
void	<code>set_point_tilt (int idx, float tilt)</code>
<i>PoolVec-</i> <i>tor3Array</i>	<code>tessellate (int max_stages=5, float tolerance_degrees=4) const</code>

25.114.3 Member Variables

- `float bake_interval` - The distance in meters between two adjacent cached points. Changing it forces the cache to be recomputed the next time the `get_baked_points` or `get_baked_length` function is called. The smaller the distance, the more points in the cache and the more memory it will consume, so use with care.

25.114.4 Description

This class describes a Bezier curve in 3D space. It is mainly used to give a shape to a *Path*, but can be manually sampled for other purposes.

It keeps a cache of precalculated points along the curve, to speed further calculations up.

25.114.5 Member Function Description

- void `add_point (Vector3 position, Vector3 in=Vector3(0, 0, 0), Vector3 out=Vector3(0, 0, 0), int at_position=-1)`

Adds a point to a curve, at “position”, with control points “in” and “out”.

If “at_position” is given, the point is inserted before the point number “at_position”, moving that point (and every point after) after the inserted point. If “at_position” is not given, or is an illegal value (`at_position < 0` or `at_position >= get_point_count`), the point will be appended at the end of the point list.

- void `clear_points ()`

Removes all points from the curve.

- `float get_baked_length () const`

Returns the total length of the curve, based on the cached points. Given enough density (see `set_bake_interval`), it should be approximate enough.

- `PoolVector3Array get_baked_points () const`

Returns the cache of points as a `PoolVector3Array`.

- `PoolRealArray get_baked_tilts () const`

Returns the cache of tilts as a `RealArray`.

- `int get_point_count () const`

Returns the number of points describing the curve.

- `Vector3 get_point_in (int idx) const`

Returns the position of the control point leading to the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0, 0).

- `Vector3 get_point_out (int idx) const`

Returns the position of the control point leading out of the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0, 0).

- `Vector3 get_point_position (int idx) const`

Returns the position of the vertex “idx”. If the index is out of bounds, the function sends an error to the console, and returns (0, 0, 0).

- `float get_point_tilt (int idx) const`

Returns the tilt angle in radians for the point “idx”. If the index is out of bounds, the function sends an error to the console, and returns 0.

- `Vector3 interpolate (int idx, float t) const`

Returns the position between the vertex “idx” and the vertex “idx”+1, where “t” controls if the point is the first vertex ($t = 0.0$), the last vertex ($t = 1.0$), or in between. Values of “t” outside the range $(0.0 \geq t \leq 1)$ give strange, but predictable results.

If “idx” is out of bounds it is truncated to the first or last vertex, and “t” is ignored. If the curve has no points, the function sends an error to the console, and returns (0, 0, 0).

- `Vector3 interpolate_baked (float offset, bool cubic=false) const`

Returns a point within the curve at position “offset”, where “offset” is measured as a distance in 3D units along the curve.

To do that, it finds the two cached points where the “offset” lies between, then interpolates the values. This interpolation is cubic if “cubic” is set to true, or linear if set to false.

Cubic interpolation tends to follow the curves better, but linear is faster (and often, precise enough).

- `Vector3 interpolatef (float fofs) const`

Returns the position at the vertex “fofs”. It calls `interpolate` using the integer part of fofs as “idx”, and its fractional part as “t”.

- `void remove_point (int idx)`

Deletes the point “idx” from the curve. Sends an error to the console if “idx” is out of bounds.

- `void set_point_in (int idx, Vector3 position)`

Sets the position of the control point leading to the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `void set_point_out (int idx, Vector3 position)`

Sets the position of the control point leading out of the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `void set_point_position (int idx, Vector3 position)`

Sets the position for the vertex “idx”. If the index is out of bounds, the function sends an error to the console.

- `void set_point_tilt (int idx, float tilt)`

Sets the tilt angle in radians for the point “idx”. If the index is out of bounds, the function sends an error to the console.

The tilt controls the rotation along the look-at axis an object traveling the path would have. In the case of a curve controlling a *PathFollow*, this tilt is an offset over the natural tilt the *PathFollow* calculates.

- `PoolVector3Array tessellate (int max_stages=5, float tolerance_degrees=4) const`

Returns a list of points along the curve, with a curvature controlled point density. That is, the curvier parts will have more points than the straighter parts.

This approximation makes straight segments between each point, then subdivides those segments until the resulting shape is similar enough.

“max_stages” controls how many subdivisions a curve segment may face before it is considered approximate enough. Each subdivision splits the segment in half, so the default 5 stages may mean up to 32 subdivisions per curve segment. Increase with care!

“tolerance_degrees” controls how many degrees the midpoint of a segment may deviate from the real curve, before the segment has to be subdivided.

25.115 CurveTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.115.1 Brief Description

A texture that shows a curve.

25.115.2 Member Variables

- ***Curve curve*** - The *curve* rendered onto the texture.
- ***int width*** - The width of the texture.

25.115.3 Description

Renders a given *Curve* provided to it. Simplifies the task of drawing curves and/or saving them as image files.

25.116 CylinderMesh

Inherits: *PrimitiveMesh < Mesh < Resource < Reference < Object*

Category: Core

25.116.1 Brief Description

Class representing a cylindrical *PrimitiveMesh*.

25.116.2 Member Variables

- *float bottom_radius* - Bottom radius of the cylinder. Defaults to 1.0.
- *float height* - Full height of the cylinder. Defaults to 2.0.
- *int radial_segments* - Number of radial segments on the cylinder. Defaults to 64.
- *int rings* - Number of edge rings along the height of the cylinder. Defaults to 4.
- *float top_radius* - Top radius of the cylinder. Defaults to 1.0.

25.116.3 Description

Class representing a cylindrical *PrimitiveMesh*.

25.117 DampedSpringJoint2D

Inherits: *Joint2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.117.1 Brief Description

Damped spring constraint for 2D physics.

25.117.2 Member Variables

- *float damping* - The spring joint's damping ratio. A value between 0 and 1. When the two bodies move into different directions the system tries to align them to the spring axis again. A high damping value forces the attached bodies to align faster. Default value: 1
- *float length* - The spring joint's maximum length. The two attached bodies cannot stretch it past this value. Default value: 50
- *float rest_length* - When the bodies attached to the spring joint move they stretch or squash it. The joint always tries to resize towards this length. Default value: 0

- *float stiffness* - The higher the value, the less the bodies attached to the joint will deform it. The joint applies an opposing force to the bodies, the product of the stiffness multiplied by the size difference from its resting length. Default value: 20

25.117.3 Description

Damped spring constraint for 2D physics. This resembles a spring joint that always wants to go back to a given length.

25.118 Dictionary

Category: Built-In Types

25.118.1 Brief Description

Dictionary type.

25.118.2 Member Functions

void	<i>clear ()</i>
<i>Dictionary</i>	<i>duplicate ()</i>
<i>bool</i>	<i>empty ()</i>
void	<i>erase (var key)</i>
<i>bool</i>	<i>has (var key)</i>
<i>bool</i>	<i>has_all (Array keys)</i>
<i>int</i>	<i>hash ()</i>
<i>Array</i>	<i>keys ()</i>
<i>int</i>	<i>size ()</i>
<i>Array</i>	<i>values ()</i>

25.118.3 Description

Dictionary type. Associative container which contains values referenced by unique keys. Dictionaries are always passed by reference.

25.118.4 Member Function Description

- void **clear ()**

Clear the dictionary, removing all key/value pairs.

- *Dictionary* **duplicate ()**
- *bool* **empty ()**

Return true if the dictionary is empty.

- void **erase (var key)**

Erase a dictionary key/value pair by key.

- *bool* **has** (var key)

Return true if the dictionary has a given key.

- *bool* **has_all** (Array keys)

Return true if the dictionary has all of the keys in the given array.

- *int* **hash** ()

Return a hashed integer value representing the dictionary contents.

- *Array* **keys** ()

Return the list of keys in the Dictionary.

- *int* **size** ()

Return the size of the dictionary (in pairs).

- *Array* **values** ()

Return the list of values in the Dictionary.

25.119 DirectionalLight

Inherits: *Light < VisualInstance < Spatial < Node < Object*

Category: Core

25.119.1 Brief Description

Directional Light, such as the Sun or the Moon.

25.119.2 Member Variables

- *float* **directional_shadow_bias_split_scale**
- *bool* **directional_shadow_blend_splits**
- *ShadowDepthRange* **directional_shadow_depth_range**
- *float* **directional_shadow_max_distance**
- *ShadowMode* **directional_shadow_mode**
- *float* **directional_shadow_normal_bias**
- *float* **directional_shadow_split_1**
- *float* **directional_shadow_split_2**
- *float* **directional_shadow_split_3**

25.119.3 Enums

enum **ShadowDepthRange**

- **SHADOW_DEPTH_RANGE_STABLE = 0**
- **SHADOW_DEPTH_RANGE_OPTIMIZED = 1**

enum **ShadowMode**

- **SHADOW_ORTHOGONAL = 0**
- **SHADOW_PARALLEL_2_SPLITS = 1**
- **SHADOW_PARALLEL_4_SPLITS = 2**

25.119.4 Description

A DirectionalLight is a type of *Light* node that emits light constantly in one direction (the negative z axis of the node). It is used lights with strong intensity that are located far away from the scene to model sunlight or moonlight. The worldspace location of the DirectionalLight transform (origin) is ignored, only the basis is used to determine light direction.

25.120 Directory

Inherits: *Reference < Object*

Category: Core

25.120.1 Brief Description

Type used to handle the filesystem.

25.120.2 Member Functions

<code>int</code>	<code>change_dir (String todir)</code>
<code>int</code>	<code>copy (String from, String to)</code>
<code>bool</code>	<code>current_is_dir () const</code>
<code>bool</code>	<code>dir_exists (String path)</code>
<code>bool</code>	<code>file_exists (String path)</code>
<code>String</code>	<code>get_current_dir ()</code>
<code>int</code>	<code>get_current_drive ()</code>
<code>String</code>	<code>get_drive (int idx)</code>
<code>int</code>	<code>get_drive_count ()</code>
<code>String</code>	<code>get_next ()</code>
<code>int</code>	<code>get_space_left ()</code>
<code>int</code>	<code>list_dir_begin (bool skip_navigational=false, bool skip_hidden=false)</code>
<code>void</code>	<code>list_dir_end ()</code>
<code>int</code>	<code>make_dir (String path)</code>
<code>int</code>	<code>make_dir_recursive (String path)</code>
<code>int</code>	<code>open (String path)</code>
<code>int</code>	<code>remove (String path)</code>
<code>int</code>	<code>rename (String from, String to)</code>

25.120.3 Description

Directory type. It is used to manage directories and their content (not restricted to the project folder).

Here is an example on how to iterate through the files of a directory:

```
func dir_contents(path):
    var dir = Directory.new()
    if dir.open(path) == OK:
        dir.list_dir_begin()
        var file_name = dir.get_next()
        while (file_name != ""):
            if dir.current_is_dir():
                print("Found directory: " + file_name)
            else:
                print("Found file: " + file_name)
            file_name = dir.get_next()
    else:
        print("An error occurred when trying to access the path.")
```

25.120.4 Member Function Description

- `int change_dir (String todir)`

Change the currently opened directory to the one passed as an argument. The argument can be relative to the current directory (e.g. newdir or `./newdir`), or an absolute path (e.g. `/tmp/newdir` or `res://somedir/newdir`).

The method returns one of the error code constants defined in `@GlobalScope` (OK or `ERR_*`).

- `int copy (String from, String to)`

Copy the `from` file to the `to` destination. Both arguments should be paths to files, either relative or absolute. If the destination file exists and is not access-protected, it will be overwritten.

Returns one of the error code constants defined in `@GlobalScope` (OK, FAILED or ERR_*)�.

- `bool current_is_dir () const`

Return whether the current item processed with the last `get_next` call is a directory (. and .. are considered directories).

- `bool dir_exists (String path)`

Return whether the target directory exists. The argument can be relative to the current directory, or an absolute path.

- `bool file_exists (String path)`

Return whether the target file exists. The argument can be relative to the current directory, or an absolute path.

- `String get_current_dir ()`

Return the absolute path to the currently opened directory (e.g. `res://folder` or `C:\tmp\folder`).

- `int get_current_drive ()`

Returns the currently opened directory's drive index. See `get_drive` to convert returned index to the name of the drive.

- `String get_drive (int idx)`

On Windows, return the name of the drive (partition) passed as an argument (e.g. C:). On other platforms, or if the requested drive does not exist, the method returns an empty String.

- `int get_drive_count ()`

On Windows, return the number of drives (partitions) mounted on the current filesystem. On other platforms, the method returns 0.

- `String get_next ()`

Return the next element (file or directory) in the current directory (including . and .., unless `skip_navigation` was given to `list_dir_begin`).

The name of the file or directory is returned (and not its full path). Once the stream has been fully processed, the method returns an empty String and closes the stream automatically (i.e. `list_dir_end` would not be mandatory in such a case).

- `int get_space_left ()`

On Unix desktop systems, return the available space on the current directory's disk. On other platforms, this information is not available and the method returns 0 or -1.

- `int list_dir_begin (bool skip_navigation=false, bool skip_hidden=false)`

Initialise the stream used to list all files and directories using the `get_next` function, closing the current opened stream if needed. Once the stream has been processed, it should typically be closed with `list_dir_end`.

If you pass `skip_navigation`, then . and .. would be filtered out.

If you pass `skip_hidden`, then hidden files would be filtered out.

- `void list_dir_end ()`

Close the current stream opened with `list_dir_begin` (whether it has been fully processed with `get_next` or not does not matter).

- `int make_dir (String path)`

Create a directory. The argument can be relative to the current directory, or an absolute path. The target directory should be placed in an already existing directory (to create the full path recursively, see `make_dir_recursive`).

The method returns one of the error code constants defined in `@GlobalScope` (OK, FAILED or ERR_*).

- `int make_dir_recursive (String path)`

Create a target directory and all necessary intermediate directories in its path, by calling `make_dir` recursively. The argument can be relative to the current directory, or an absolute path.

Return one of the error code constants defined in `@GlobalScope` (OK, FAILED or ERR_*).

- `int open (String path)`

Open an existing directory of the filesystem. The `path` argument can be within the project tree (`res://folder`), the user directory (`user://folder`) or an absolute path of the user filesystem (e.g. `/tmp/folder` or `C:\tmp\folder`).

The method returns one of the error code constants defined in `@GlobalScope` (OK or ERR_*).

- `int remove (String path)`

Delete the target file or an empty directory. The argument can be relative to the current directory, or an absolute path. If the target directory is not empty, the operation will fail.

Return one of the error code constants defined in `@GlobalScope` (OK or FAILED).

- `int rename (String from, String to)`

Rename (move) the `from` file to the `to` destination. Both arguments should be paths to files, either relative or absolute. If the destination file exists and is not access-protected, it will be overwritten.

Return one of the error code constants defined in `@GlobalScope` (OK or FAILED).

25.121 DynamicFont

Inherits: `Font < Resource < Reference < Object`

Category: Core

25.121.1 Brief Description

`DynamicFont` renders vector font files at runtime.

25.121.2 Member Functions

<code>void</code>	<code>add_fallback (DynamicFontData data)</code>
<code>DynamicFontData</code>	<code>get_fallback (int idx) const</code>
<code>int</code>	<code>get_fallback_count () const</code>
<code>void</code>	<code>remove_fallback (int idx)</code>
<code>void</code>	<code>set_fallback (int idx, DynamicFontData data)</code>

25.121.3 Member Variables

- `int extra_spacing_bottom` - Extra spacing at the bottom in pixels.
- `int extra_spacing_char` - Extra character spacing in pixels.
- `int extra_spacing_space` - Extra space spacing in pixels.

- *int extra_spacing_top* - Extra spacing at the top in pixels.
- *DynamicFontData font_data* - The font data.
- *int size* - The font size.
- *bool use_filter* - If true filtering is used.
- *bool use_mipmaps* - If true mipmapping is used.

25.121.4 Enums

enum SpacingType

- **SPACING_TOP = 0** — Spacing at the top.
- **SPACING_BOTTOM = 1** — Spacing at the bottom.
- **SPACING_CHAR = 2** — Character spacing.
- **SPACING_SPACE = 3** — Space spacing.

25.121.5 Description

DynamicFont renders vector font files (such as TTF or OTF) dynamically at runtime instead of using a prerendered texture atlas like *BitmapFont*. This trades the faster loading time of *BitmapFonts* for the ability to change font parameters like size and spacing during runtime. *DynamicFontData* is used for referencing the font file paths.

25.121.6 Member Function Description

- `void add_fallback (DynamicFontData data)`

Adds a fallback font.

- `DynamicFontData get_fallback (int idx) const`

Returns the fallback font at index `idx`.

- `int get_fallback_count () const`

Returns the number of fallback fonts.

- `void remove_fallback (int idx)`

Removes the fallback font at index `idx`.

- `void set_fallback (int idx, DynamicFontData data)`

Sets the fallback font at index `idx`.

25.122 DynamicFontData

Inherits: *Resource < Reference < Object*

Category: Core

25.122.1 Brief Description

Used with *DynamicFont* to describe the location of a font file.

25.122.2 Member Variables

- **String font_path** - The path to the vector font file.

25.122.3 Description

Used with *DynamicFont* to describe the location of a vector font file for dynamic rendering at runtime.

25.123 EditorExportPlugin

Inherits: Reference < Object

Category: Core

25.123.1 Brief Description

25.123.2 Member Functions

void	<code>_export_begin (PoolStringArray features, bool is_debug, String path, int flags)</code> virtual
void	<code>_export_file (String path, String type, PoolStringArray features)</code> virtual
void	<code>add_file (String path, PoolByteArray file, bool remap)</code>
void	<code>add_ios_bundle_file (String path)</code>
void	<code>add_ios_cpp_code (String code)</code>
void	<code>add_ios_framework (String path)</code>
void	<code>add_ios_linker_flags (String flags)</code>
void	<code>add_ios_plist_content (String plist_content)</code>
void	<code>add_shared_object (String path, PoolStringArray tags)</code>
void	<code>skip ()</code>

25.123.3 Member Function Description

- void `_export_begin (PoolStringArray features, bool is_debug, String path, int flags)` virtual
- void `_export_file (String path, String type, PoolStringArray features)` virtual
- void `add_file (String path, PoolByteArray file, bool remap)`
- void `add_ios_bundle_file (String path)`
- void `add_ios_cpp_code (String code)`
- void `add_ios_framework (String path)`
- void `add_ios_linker_flags (String flags)`
- void `add_ios_plist_content (String plist_content)`

- void **add_shared_object** (*String path, PoolStringArray tags*)
- void **skip** ()

25.124 EditorFileDialog

Inherits: ConfirmationDialog < AcceptDialog < WindowDialog < Popup < Control < CanvasItem < Node < Object

Category: Core

25.124.1 Brief Description

25.124.2 Member Functions

void	<i>add_filter (String filter)</i>
void	<i>clear_filters ()</i>
VBoxContainer	<i>get_vbox ()</i>
void	<i>invalidate ()</i>

25.124.3 Signals

- **dir_selected** (*String dir*)

Emitted when a directory is selected.

- **file_selected** (*String path*)

Emitted when a file is selected.

- **files_selected** (*PoolStringArray paths*)

Emitted when multiple files are selected.

25.124.4 Member Variables

- *Access access* - The location from which the user may select a file, including `res://`, `user://`, and the local file system.
- *String current_dir* - The currently occupied directory.
- *String current_file* - The currently selected file.
- *String current_path* - The file system path in the address bar.
- *bool disable_overwrite_warning* - If `true` the `EditorFileDialog` will not warn the user before overwriting files.
- *DisplayMode display_mode* - The view format in which the `EditorFileDialog` displays resources to the user.
- *Mode mode* - The purpose of the `EditorFileDialog`. Changes allowed behaviors.
- *bool show_hidden_files* - If `true` hidden files and directories will be visible in the `EditorFileDialog`.

25.124.5 Enums

enum Access

- **ACCESS_RESOURCES = 0** — The EditorFileDialog can only view `res://` directory contents.
- **ACCESS_USERDATA = 1** — The EditorFileDialog can only view `user://` directory contents.
- **ACCESS_FILESYSTEM = 2** — The EditorFileDialog can view the entire local file system.

enum DisplayMode

- **DISPLAY_THUMBNAILS = 0** — The EditorFileDialog displays resources as thumbnails.
- **DISPLAY_LIST = 1** — The EditorFileDialog displays resources as a list of filenames.

enum Mode

- **MODE_OPEN_FILE = 0** — The EditorFileDialog can select only one file. Accepting the window will open the file.
- **MODE_OPEN_FILES = 1** — The EditorFileDialog can select multiple files. Accepting the window will open all files.
- **MODE_OPEN_DIR = 2** — The EditorFileDialog can select only one directory. Accepting the window will open the directory.
- **MODE_OPEN_ANY = 3** — The EditorFileDialog can select a file or directory. Accepting the window will open it.
- **MODE_SAVE_FILE = 4** — The EditorFileDialog can select only one file. Accepting the window will save the file.

25.124.6 Member Function Description

- `void add_filter (String filter)`

Adds a comma-delimited file extension filter option to the EditorFileDialog with an optional semi-colon-delimited label.

Example: “*.tscn, *.scn; Scenes”, results in filter text “Scenes (*.tscn, *.scn)”.

- `void clear_filters ()`

Removes all filters except for “All Files (*)”.

- `VBoxContainer get_vbox ()`

Returns the VBoxContainer used to display the file system.

- `void invalidate ()`

Notify the EditorFileDialog that its view of the data is no longer accurate. Updates the view contents on next view update.

25.125 EditorFileSystem

Inherits: `Node < Object`

Category: Core

25.125.1 Brief Description

Resource filesystem, as the editor sees it.

25.125.2 Member Functions

<i>String</i>	<i>get_file_type (String path) const</i>
<i>EditorFileSystemDirectory</i>	<i>get_filesystem ()</i>
<i>EditorFileSystemDirectory</i>	<i>get_filesystem_path (String path)</i>
<i>float</i>	<i>get_scanning_progress () const</i>
<i>bool</i>	<i>is_scanning () const</i>
<i>void</i>	<i>scan ()</i>
<i>void</i>	<i>scan_sources ()</i>
<i>void</i>	<i>update_file (String path)</i>

25.125.3 Signals

- **filesystem_changed ()**

Emitted if the filesystem changed.

- **resources_reimported (PoolStringArray resources)**

Remitted if a resource is reimported.

- **sources_changed (bool exist)**

Emitted if the source of any imported file changed.

25.125.4 Description

This object holds information of all resources in the filesystem, their types, etc.

25.125.5 Member Function Description

- **String get_file_type (String path) const**

Get the type of the file, given the full path.

- **EditorFileSystemDirectory get_filesystem ()**

Get the root directory object.

- **EditorFileSystemDirectory get_filesystem_path (String path)**

Returns a view into the filesystem at path.

- **float get_scanning_progress () const**

Return the scan progress for 0 to 1 if the FS is being scanned.

- **bool is_scanning () const**

Return true if the filesystem is being scanned.

- **void scan ()**

Scan the filesystem for changes.

- `void scan_sources()`

Check if the source of any imported resource changed.

- `void update_file(String path)`

Update a file information. Call this if an external program (not Godot) modified the file.

25.126 EditorFileSystemDirectory

Inherits: *Object*

Category: Core

25.126.1 Brief Description

A directory for the resource filesystem.

25.126.2 Member Functions

<code>int</code>	<code>find_dir_index(String name) const</code>
<code>int</code>	<code>find_file_index(String name) const</code>
<code>String</code>	<code>get_file(int idx) const</code>
<code>int</code>	<code>get_file_count() const</code>
<code>bool</code>	<code>get_file_import_is_valid(int idx) const</code>
<code>String</code>	<code>get_file_path(int idx) const</code>
<code>String</code>	<code>get_file_type(int idx) const</code>
<code>String</code>	<code>get_name()</code>
<code>EditorFileSystemDirectory</code>	<code>get_parent()</code>
<code>String</code>	<code>get_path() const</code>
<code>EditorFileSystemDirectory</code>	<code>get_subdir(int idx)</code>
<code>int</code>	<code>get_subdir_count() const</code>

25.126.3 Description

A more generalized, low-level variation of the directory concept.

25.126.4 Member Function Description

- `int find_dir_index(String name) const`

Returns the index of the directory with name `name` or `-1` if not found.

- `int find_file_index(String name) const`

Returns the index of the file with name `name` or `-1` if not found.

- `String get_file(int idx) const`

Returns the name of the file at index `idx`.

- `int get_file_count () const`

Returns the number of files in this directory.

- `bool get_file_import_is_valid (int idx) const`

Returns `true` if the file at index `idx` imported properly.

- `String get_file_path (int idx) const`

Returns the path to the file at index `idx`.

- `String get_file_type (int idx) const`

Returns the file extension of the file at index `idx`.

- `String get_name ()`

Returns the name of this directory.

- `EditorFileSystemDirectory get_parent ()`

Returns the parent directory for this directory or null if called on a directory at `res://` or `user://`.

- `String get_path () const`

Returns the path to this directory.

- `EditorFileSystemDirectory get_subdir (int idx)`

Returns the subdirectory at index `idx`.

- `int get_subdir_count () const`

Returns the number of subdirectories in this directory.

25.127 EditorImportPlugin

Inherits: *Reference < Object*

Category: Core

25.127.1 Brief Description

Registers a custom resource importer in the editor. Use the class to parse any file and import it as a new resource type.

25.127.2 Member Functions

<code>Ar-ray</code>	<code>get_import_options (int preset) virtual</code>
<code>String</code>	<code>get_importer_name () virtual</code>
<code>bool</code>	<code>get_option_visibility (String option, Dictionary options) virtual</code>
<code>int</code>	<code>get_preset_count () virtual</code>
<code>String</code>	<code>get_preset_name (int preset) virtual</code>
<code>Ar-ray</code>	<code>get_recognized_extensions () virtual</code>
<code>String</code>	<code>get_resource_type () virtual</code>
<code>String</code>	<code>get_save_extension () virtual</code>
<code>String</code>	<code>get_visible_name () virtual</code>
<code>int</code>	<code>import (String source_file, String save_path, Dictionary options, Array r_platform_variants, Array r_gen_files) virtual</code>

25.127.3 Description

EditorImportPlugins provide a way to extend the editor's resource import functionality. Use them to import resources from custom files or to provide alternatives to the editor's existing importers. Register your *EditorPlugin* with *EditorPlugin.add_import_plugin*.

EditorImportPlugins work by associating with specific file extensions and a resource type. See *get_recognized_extension* and *get_resource_type*). They may optionally specify some import presets that affect the import process. EditorImportPlugins are responsible for creating the resources and saving them in the `.import` directory.

Below is an example EditorImportPlugin that imports a *Mesh* from a file with the extension “.special” or “.spec”:

```
tool
extends EditorImportPlugin

func get_importer_name():
    return "my.special.plugin"

func get_visible_name():
    return "Special Mesh Importer"

func get_recognized_extensions():
    return ["special", "spec"]

func get_save_extension():
    return "mesh"

func get_resource_type():
    return "Mesh"

func get_preset_count():
    return 1

func get_preset_name(i):
    return "Default"

func get_import_options(i):
    return [{"name": "my_option", "default_value": false}]
```

```
func load(src, dst, opts, r_platform_variants, r_gen_files):
    var file = File.new()
    if file.open(src, File.READ) != OK:
        return FAILED

    var mesh = Mesh.new()

    var save = dst + "." + get_save_extension()
    ResourceSaver.save(file, mesh)
    return OK
```

25.127.4 Member Function Description

- *Array* **get_import_options** (*int* preset) *virtual*

Get the options and default values for the preset at this index. Returns an Array of Dictionaries with the following keys: “name”, “default_value”, “property_hint” (optional), “hint_string” (optional), “usage” (optional).

- *String* **get_importer_name** () *virtual*

Get the unique name of the importer.

- *bool* **get_option_visibility** (*String* option, *Dictionary* options) *virtual*
- *int* **get_preset_count** () *virtual*

Get the number of initial presets defined by the plugin. Use *get_import_options* to get the default options for the preset and *get_preset_name* to get the name of the preset.

- *String* **get_preset_name** (*int* preset) *virtual*

Get the name of the options preset at this index.

- *Array* **get_recognized_extensions** () *virtual*

Get the list of file extensions to associate with this loader (case insensitive). e.g. “obj”.

- *String* **get_resource_type** () *virtual*

Get the godot resource type associated with this loader. e.g. “Mesh” or “Animation”.

- *String* **get_save_extension** () *virtual*

Get the extension used to save this resource in the `.import` directory.

- *String* **get_visible_name** () *virtual*

Get the name to display in the import window.

- *int* **import** (*String* source_file, *String* save_path, *Dictionary* options, *Array* r_platform_variants, *Array* r_gen_files) *virtual*

25.128 EditorInterface

Inherits: *Node < Object*

Category: Core

25.128.1 Brief Description

Editor interface and main components.

25.128.2 Member Functions

void	<code>edit_resource (Resource resource)</code>
<i>Control</i>	<code>get_base_control ()</code>
<i>Node</i>	<code>get_edited_scene_root ()</code>
<i>EditorSettings</i>	<code>get_editor_settings ()</code>
<i>Control</i>	<code>get_editor_viewport ()</code>
<i>Array</i>	<code>get_open_scenes () const</code>
<i>EditorFileSystem</i>	<code>get_resource_filesystem ()</code>
<i>EditorResourcePreview</i>	<code>get_resource_previewer ()</code>
<i>ScriptEditor</i>	<code>get_script_editor ()</code>
<i>String</i>	<code>get_selected_path () const</code>
<i>EditorSelection</i>	<code>get_selection ()</code>
void	<code>inspect_object (Object object, String for_property="")</code>
<i>Array</i>	<code>make_mesh_previews (Array meshes, int preview_size)</code>
void	<code>open_scene_from_path (String scene_filepath)</code>
void	<code>reload_scene_from_path (String scene_filepath)</code>
<i>int</i>	<code>save_scene ()</code>
void	<code>save_scene_as (String path, bool with_preview=true)</code>
void	<code>select_file (String p_file)</code>

25.128.3 Description

Editor interface. Allows saving and (re-)loading scenes, rendering mesh previews, inspecting and editing resources and objects and provides access to *EditorSettings*, *EditorFileSystem*, *EditorResourcePreviewer*, *ScriptEditor*, the editor viewport, as well as information about scenes. Also see *EditorPlugin* and *EditorScript*.

25.128.4 Member Function Description

- void **edit_resource** (*Resource* resource)

Edits the given *Resource*.

- *Control* **get_base_control** ()

Returns the base *Control*.

- *Node* **get_edited_scene_root** ()

Returns the edited scene's root *Node*.

- *EditorSettings* **get_editor_settings** ()

Returns the *EditorSettings*.

- *Control* **get_editor_viewport** ()

Returns the editor *Viewport*.

- *Array* **get_open_scenes** () const

Returns an *Array* of the currently opened scenes.

- *EditorFileSystem* **get_resource_filesystem ()**

Returns the *EditorFileSystem*.

- *EditorResourcePreview* **get_resource_previewer ()**

Returns the *EditorResourcePreviewer*.

- *ScriptEditor* **get_script_editor ()**

Returns the *ScriptEditor*.

- *String* **get_selected_path () const**

- *EditorSelection* **get_selection ()**

Returns the *EditorSelection*.

- **void inspect_object (Object object, String for_property="")**

Shows the given property on the given *object* in the Editor's Inspector dock.

- *Array* **make_mesh_previews (Array meshes, int preview_size)**

Returns mesh previews rendered at the given size as an *Array* of *Textures*.

- **void open_scene_from_path (String scene_filepath)**

Opens the scene at the given path.

- **void reload_scene_from_path (String scene_filepath)**

Reloads the scene at the given path.

- **int save_scene ()**

Saves the scene. Returns either OK or ERR_CANT_CREATE. See @GlobalScope constants.

- **void save_scene_as (String path, bool with_preview=true)**

Saves the scene as a file at *path*.

- **void select_file (String p_file)**

25.129 EditorPlugin

Inherits: *Node < Object*

Category: Core

25.129.1 Brief Description

Used by the editor to extend its functionality.

25.129.2 Member Functions

<i>ToolButton</i>	<i>add_control_to_bottom_panel (Control control, String title)</i>
<i>void</i>	<i>add_control_to_container (int container, Control control)</i>

Continued on next page

Table 25.8 – continued from previous page

void	<i>add_control_to_dock (int slot, Control control)</i>
void	<i>add_custom_type (String type, String base, Script script, Texture icon)</i>
void	<i>add_export_plugin (EditorExportPlugin exporter)</i>
void	<i>add_import_plugin (EditorImportPlugin importer)</i>
void	<i>add_scene_import_plugin (EditorSceneImporter scene_importer)</i>
void	<i>add_tool_submenu_item (String name, Object submenu)</i>
void	<i>apply_changes () virtual</i>
void	<i>clear () virtual</i>
<i>EditorSpatialGizmo</i>	<i>create_spatial_gizmo (Spatial for_spatial) virtual</i>
void	<i>edit (Object object) virtual</i>
bool	<i>forward_canvas_gui_input (InputEvent event) virtual</i>
void	<i>forward_draw_over_viewport (Control overlay) virtual</i>
void	<i>forward_force_draw_over_viewport (Control overlay) virtual</i>
bool	<i>forward_spatial_gui_input (Camera camera, InputEvent event) virtual</i>
<i>PoolStringArray</i>	<i>get_breakpoints () virtual</i>
<i>EditorInterface</i>	<i>get_editor_interface ()</i>
<i>Object</i>	<i>get_plugin_icon () virtual</i>
<i>String</i>	<i>get_plugin_name () virtual</i>
<i>Dictionary</i>	<i>get_state () virtual</i>
<i>UndoRedo</i>	<i>get_undo_redo ()</i>
void	<i>get_window_layout (ConfigFile layout) virtual</i>
bool	<i>handles (Object object) virtual</i>
bool	<i>has_main_screen () virtual</i>
void	<i>hide_bottom_panel ()</i>
void	<i>make_bottom_panel_item_visible (Control item)</i>
void	<i>make_visible (bool visible) virtual</i>
void	<i>queue_save_layout () const</i>
void	<i>remove_control_from_bottom_panel (Control control)</i>
void	<i>remove_control_from_docks (Control control)</i>
void	<i>remove_custom_type (String type)</i>
void	<i>remove_export_plugin (EditorExportPlugin exporter)</i>
void	<i>remove_import_plugin (EditorImportPlugin importer)</i>
void	<i>remove_scene_import_plugin (EditorSceneImporter scene_importer)</i>
void	<i>save_external_data () virtual</i>
void	<i>set_force_draw_over_forwarding_enabled ()</i>
void	<i>set_input_event_forwarding_always_enabled ()</i>
void	<i>set_state (Dictionary state) virtual</i>
void	<i>set_window_layout (ConfigFile layout) virtual</i>
int	<i>update_overlays () const</i>

25.129.3 Signals

- **main_screen_changed** (*String screen_name*)

Emitted when user change main screen view (2D, 3D, Script, AssetLib). Works also with screens which are defined by plugins.

- **scene_changed** (*Object scene_root*)

Emitted when user change scene. The argument is a root node of freshly opened scene.

- **scene_closed** (*String filepath*)

Emitted when user close scene. The argument is file path to a closed scene.

25.129.4 Enums

enum **DockSlot**

- **DOCK_SLOT_LEFT_UL = 0**
- **DOCK_SLOT_LEFT_BL = 1**
- **DOCK_SLOT_LEFT_UR = 2**
- **DOCK_SLOT_LEFT_BR = 3**
- **DOCK_SLOT_RIGHT_UL = 4**
- **DOCK_SLOT_RIGHT_BL = 5**
- **DOCK_SLOT_RIGHT_UR = 6**
- **DOCK_SLOT_RIGHT_BR = 7**
- **DOCK_SLOT_MAX = 8**

enum **CustomControlContainer**

- **CONTAINER_TOOLBAR = 0**
- **CONTAINER_SPATIAL_EDITOR_MENU = 1**
- **CONTAINER_SPATIAL_EDITOR_SIDE = 2**
- **CONTAINER_SPATIAL_EDITOR_BOTTOM = 3**
- **CONTAINER_CANVAS_EDITOR_MENU = 4**
- **CONTAINER_CANVAS_EDITOR_SIDE = 5**
- **CONTAINER_CANVAS_EDITOR_BOTTOM = 6**
- **CONTAINER_PROPERTY_EDITOR_BOTTOM = 7**

25.129.5 Description

Plugins are used by the editor to extend functionality. The most common types of plugins are those which edit a given node or resource type, import plugins and export plugins.

25.129.6 Member Function Description

- *ToolButton* **add_control_to_bottom_panel** (*Control* control, *String* title)

Add a control to the bottom panel (together with Output, Debug, Animation, etc). Returns a reference to the button added. It's up to you to hide/show the button when needed. If your plugin is being removed, also make sure to remove your control by calling *remove_control_from_bottom_panel*.

- **void add_control_to_container** (*int* container, *Control* control)

Add a custom control to a container (see *CONTAINER_** enum). There are many locations where custom controls can be added in the editor UI.

Please remember that you have to manage the visibility of your custom controls yourself (and likely hide it after adding it).

If your plugin is being removed, also make sure to remove your custom controls too.

- void **add_control_to_dock** (*int slot, Control control*)

Add the control to a specific dock slot (see DOCK_* enum for options).

If the dock is repositioned and as long as the plugin is active, the editor will save the dock position on further sessions.

If your plugin is being removed, also make sure to remove your control by calling *remove_control_from_docks*.

- void **add_custom_type** (*String type, String base, Script script, Texture icon*)

Add a custom type, which will appear in the list of nodes or resources. An icon can be optionally passed.

When given node or resource is selected, the base type will be instanced (ie, “Spatial”, “Control”, “Resource”), then the script will be loaded and set to this object.

You can use the *EditorPlugin.handles* to check if your custom object is being edited by checking the script or using ‘is’ keyword.

During run-time, this will be a simple object with a script so this function does not need to be called then.

- void **add_export_plugin** (*EditorExportPlugin exporter*)
- void **add_import_plugin** (*EditorImportPlugin importer*)
- void **add_scene_import_plugin** (*EditorSceneImporter scene_importer*)
- void **add_tool_submenu_item** (*String name, Object submenu*)
- void **apply_changes** () virtual

This method is called when the editor is about to save the project, switch to another tab, etc. It asks the plugin to apply any pending state changes to ensure consistency.

This is used, for example, in shader editors to let the plugin know that it must apply the shader code being written by the user to the object.

- void **clear** () virtual

Clear all the state and reset the object being edited to zero. This ensures your plugin does not keep editing a currently existing node, or a node from the wrong scene.

- *EditorSpatialGizmo* **create_spatial_gizmo** (*Spatial for_spatial*) virtual

This is used for plugins that create gizmos used by the spatial editor. Just check that the node passed in the “for_spatial” argument matches your plugin.

- void **edit** (*Object object*) virtual

This function is used for plugins that edit specific object types (nodes or resources). It requests the editor to edit the given object.

- *bool* **forward_canvas_gui_input** (*InputEvent event*) virtual
- void **forward_draw_over_viewport** (*Control overlay*) virtual
- void **forward_force_draw_over_viewport** (*Control overlay*) virtual
- *bool* **forward_spatial_gui_input** (*Camera camera, InputEvent event*) virtual

Implement this function if you are interested in 3D view screen input events. It will be called only if currently selected node is handled by your plugin.

If you would like to always get those input events then additionally use *set_input_forwarding_always_enabled*.

- *PoolStringArray* **get_breakpoints** () virtual

This is for editors that edit script based objects. You can return a list of breakpoints in the format (script:line), for example: res://path_to_script.gd:25

- *EditorInterface* **get_editor_interface()**
- *Object* **get_plugin_icon()** virtual
- *String* **get_plugin_name()** virtual
- *Dictionary* **get_state()** virtual

Get the state of your plugin editor. This is used when saving the scene (so state is kept when opening it again) and for switching tabs (so state can be restored when the tab returns).

- *UndoRedo* **get_undo_redo()**

Get the undo/redo object. Most actions in the editor can be undoable, so use this object to make sure this happens when it's worth it.

- void **get_window_layout(ConfigFile layout)** virtual

Get the GUI layout of the plugin. This is used to save the project's editor layout when the *EditorPlugin.queue_save_layout* is called or the editor layout was changed (For example changing the position of a dock).

- bool **handles(Object object)** virtual

Implement this function if your plugin edits a specific type of object (Resource or Node). If you return true, then you will get the functions *EditorPlugin.edit* and *EditorPlugin.make_visible* called when the editor requests them.

- bool **has_main_screen()** virtual

Return true if this is a main screen editor plugin (it goes in the main screen selector together with 2D, 3D, Script).

- void **hide_bottom_panel()**
- void **make_bottom_panel_item_visible(Control item)**
- void **make_visible(bool visible)** virtual

This function will be called when the editor is requested to become visible. It is used for plugins that edit a specific object type.

Remember that you have to manage the visibility of all your editor controls manually.

- void **queue_save_layout()** const

Queue save the project's editor layout.

- void **remove_control_from_bottom_panel(Control control)**

Remove the control from the bottom panel. Don't forget to call this if you added one, so the editor can remove it cleanly.

- void **remove_control_from_docks(Control control)**

Remove the control from the dock. Don't forget to call this if you added one, so the editor can save the layout and remove it cleanly.

- void **remove_custom_type(String type)**

Remove a custom type added by *EditorPlugin.add_custom_type*

- void **remove_export_plugin(EditorExportPlugin exporter)**
- void **remove_import_plugin(EditorImportPlugin importer)**
- void **remove_scene_import_plugin(EditorSceneImporter scene_importer)**
- void **save_external_data()** virtual

This method is called after the editor saves the project or when it's closed. It asks the plugin to save edited external scenes/resources.

- void **set_force_draw_over_forwarding_enabled** ()
- void **set_input_event_forwarding_always_enabled** ()

Use this method if you always want to receive inputs from 3D view screen inside *forward_spatial_gui_input*. It might be especially usable if your plugin will want to use raycast in the scene.

- void **set_state** (*Dictionary state*) virtual

Restore the state saved by *EditorPlugin.get_state*.

- void **set_window_layout** (*ConfigFile layout*) virtual

Restore the plugin GUI layout saved by *EditorPlugin.get_window_layout*.

- int **update_overlays** () const

25.130 EditorResourceConversionPlugin

Inherits: *Reference < Object*

Category: Core

25.130.1 Brief Description

25.130.2 Member Functions

<i>Resource</i>	_convert (<i>Resource resource</i>) virtual
<i>String</i>	_converts_to () virtual

25.130.3 Member Function Description

- *Resource* **_convert** (*Resource resource*) virtual
- *String* **_converts_to** () virtual

25.131 EditorResourcePreview

Inherits: *Node < Object*

Category: Core

25.131.1 Brief Description

Helper to generate previews of resources or files.

25.131.2 Member Functions

void	<code>add_preview_generator (EditorResourcePreviewGenerator generator)</code>
void	<code>check_for_invalidation (String path)</code>
void	<code>queue_edited_resource_preview (Resource resource, Object receiver, String receiver_func, Variant userdata)</code>
void	<code>queue_resource_preview (String path, Object receiver, String receiver_func, Variant userdata)</code>
void	<code>remove_preview_generator (EditorResourcePreviewGenerator generator)</code>

25.131.3 Signals

- **preview_invalidated** (*String path*)

If a preview was invalidated (changed) this signal will emit (using the path of the preview)

25.131.4 Description

This object is used to generate previews for resources of files.

25.131.5 Member Function Description

- void **add_preview_generator** (*EditorResourcePreviewGenerator generator*)

Create an own, custom preview generator.

- void **check_for_invalidation** (*String path*)

Check if the resource changed, if so it will be invalidated and the corresponding signal emitted.

- void **queue_edited_resource_preview** (*Resource resource, Object receiver, String receiver_func, Variant userdata*)

Queue a resource being edited for preview (using an instance). Once the preview is ready, your receiver.receiver_func will be called either containing the preview texture or an empty texture (if no preview was possible). Callback must have the format: (path,texture,userdata). Userdata can be anything.

- void **queue_resource_preview** (*String path, Object receiver, String receiver_func, Variant userdata*)

Queue a resource file for preview (using a path). Once the preview is ready, your receiver.receiver_func will be called either containing the preview texture or an empty texture (if no preview was possible). Callback must have the format: (path,texture,userdata). Userdata can be anything.

- void **remove_preview_generator** (*EditorResourcePreviewGenerator generator*)

Remove a custom preview generator.

25.132 EditorResourcePreviewGenerator

Inherits: Reference < Object

Category: Core

25.132.1 Brief Description

Custom generator of previews.

25.132.2 Member Functions

<i>Texture</i>	<i>generate (Resource from) virtual</i>
<i>Texture</i>	<i>generate_from_path (String path) virtual</i>
<i>bool</i>	<i>handles (String type) virtual</i>

25.132.3 Description

Custom code to generate previews. Please check “file_dialog/thumbnail_size” in EditorSettings to find out the right size to do previews at.

25.132.4 Member Function Description

- *Texture generate (Resource from) virtual*

Generate a preview from a given resource. This must be always implemented.

Returning an empty texture is an OK way to fail and let another generator take care.

Care must be taken because this function is always called from a thread (not the main thread).

- *Texture generate_from_path (String path) virtual*

Generate a preview directly from a path, implementing this is optional, as default code will load and call generate()

Returning an empty texture is an OK way to fail and let another generator take care.

Care must be taken because this function is always called from a thread (not the main thread).

- *bool handles (String type) virtual*

Return if your generator supports this resource type.

25.133 EditorSceneImporter

Inherits: *Reference < Object*

Category: Core

25.133.1 Brief Description

25.133.2 Member Functions

<i>Array</i>	<code>_get_extensions () virtual</code>
<i>int</i>	<code>_get_import_flags () virtual</code>
<i>Animation</i>	<code>_import_animation (String path, int flags, int bake_fps) virtual</code>
<i>Node</i>	<code>_import_scene (String path, int flags, int bake_fps) virtual</code>
<i>Animation</i>	<code>import_animation_from_other_importer (String path, int flags, int bake_fps)</code>
<i>Node</i>	<code>import_scene_from_other_importer (String path, int flags, int bake_fps)</code>

25.133.3 Numeric Constants

- **IMPORT_SCENE = 1**
- **IMPORT_ANIMATION = 2**
- **IMPORT_ANIMATION_DETECT_LOOP = 4**
- **IMPORT_ANIMATION_OPTIMIZE = 8**
- **IMPORT_ANIMATION_FORCE_ALL_TRACKS_IN_ALL_CLIPS = 16**
- **IMPORT_ANIMATION_KEEP_VALUE_TRACKS = 32**
- **IMPORT_GENERATE_TANGENT_ARRAYS = 256**
- **IMPORT_FAIL_ON_MISSING_DEPENDENCIES = 512**
- **IMPORT_MATERIALS_IN_INSTANCES = 1024**
- **IMPORT_USE_COMPRESSION = 2048**

25.133.4 Member Function Description

- *Array* `_get_extensions () virtual`
- *int* `_get_import_flags () virtual`
- *Animation* `_import_animation (String path, int flags, int bake_fps) virtual`
- *Node* `_import_scene (String path, int flags, int bake_fps) virtual`
- *Animation* `import_animation_from_other_importer (String path, int flags, int bake_fps)`
- *Node* `import_scene_from_other_importer (String path, int flags, int bake_fps)`

25.134 EditorScenePostImport

Inherits: *Reference < Object*

Category: Core

25.134.1 Brief Description

25.134.2 Member Functions

void	<i>post_import (Object scene) virtual</i>
------	---

25.134.3 Member Function Description

- void **post_import (Object scene) virtual**

25.135 EditorScript

Inherits: Reference < Object

Category: Core

25.135.1 Brief Description

Base script that can be used to add extension functions to the editor.

25.135.2 Member Functions

void	<i>_run () virtual</i>
void	<i>add_root_node (Node node)</i>
<i>EditorInterface</i>	<i>get_editor_interface ()</i>
<i>Node</i>	<i>get_scene ()</i>

25.135.3 Description

Scripts extending this class and implementing its `_run()` method can be executed from the Script Editor's `File -> Run` menu option (or by pressing `CTRL+Shift+X`) while the editor is running. This is useful for adding custom in-editor functionality to Godot. For more complex additions, consider using `EditorPlugins` instead. Note that extending scripts need to have `tool` mode enabled.

Example script:

```
tool
extends EditorScript

func _run():
    print("Hello from the Godot Editor!")
```

Note that the script is run in the Editor context, which means the output is visible in the console window started with the Editor (STDOUT) instead of the usual Godot *Output* dock.

25.135.4 Member Function Description

- `void _run () virtual`

This method is executed by the Editor when `File -> Run` is used.

- `void add_root_node (Node node)`

Adds `node` as a child of the root node in the editor context.

WARNING: The implementation of this method is currently disabled.

- `EditorInterface get_editor_interface ()`

Returns the `EditorInterface` singleton instance.

- `Node get_scene ()`

Returns the Editor's currently active scene.

25.136 EditorSelection

Inherits: `Object`

Category: Core

25.136.1 Brief Description

Manages the SceneTree selection in the editor.

25.136.2 Member Functions

<code>void</code>	<code>add_node (Node node)</code>
<code>void</code>	<code>clear ()</code>
<code>Array</code>	<code>get_selected_nodes ()</code>
<code>Array</code>	<code>get_transformable_selected_nodes ()</code>
<code>void</code>	<code>remove_node (Node node)</code>

25.136.3 Signals

- `selection_changed ()`

Emitted when the selection changes.

25.136.4 Description

This object manages the SceneTree selection in the editor.

25.136.5 Member Function Description

- void **add_node** (*Node* node)

Add a node to the selection.

- void **clear** ()

Clear the selection.

- *Array* **get_selected_nodes** ()

Get the list of selected nodes.

- *Array* **get_transformable_selected_nodes** ()

Get the list of selected nodes, optimized for transform operations (ie, moving them, rotating, etc). This list avoids situations where a node is selected and also child/grandchild.

- void **remove_node** (*Node* node)

Remove a node from the selection.

25.137 EditorSettings

Inherits: *Resource* < *Reference* < *Object*

Category: Core

25.137.1 Brief Description

Object that holds the project-independent editor settings.

25.137.2 Member Functions

void	<i>add_property_info</i> (<i>Dictionary</i> info)
void	<i>erase</i> (<i>String</i> property)
<i>PoolStringArray</i>	<i>get_favorite_dirs</i> () const
<i>String</i>	<i>get_project_settings_dir</i> () const
<i>PoolStringArray</i>	<i>get_recent_dirs</i> () const
<i>Variant</i>	<i>get_setting</i> (<i>String</i> name) const
<i>String</i>	<i>get_settings_dir</i> () const
<i>bool</i>	<i>has_setting</i> (<i>String</i> name) const
<i>bool</i>	<i>property_can_revert</i> (<i>String</i> name)
<i>Variant</i>	<i>property_get_revert</i> (<i>String</i> name)
void	<i>set_favorite_dirs</i> (<i>PoolStringArray</i> dirs)
void	<i>set_initial_value</i> (<i>String</i> name, <i>Variant</i> value, <i>bool</i> update_current)
void	<i>set_recent_dirs</i> (<i>PoolStringArray</i> dirs)
void	<i>set_setting</i> (<i>String</i> name, <i>Variant</i> value)

25.137.3 Signals

- **settings_changed** ()

25.137.4 Description

Object that holds the project-independent editor settings. These settings are generally visible in the Editor Settings menu.

Accessing the settings is done by using the regular *Object* API, such as:

```
settings.set(prop,value)
settings.get(prop)
list_of_settings = settings.get_property_list()
```

25.137.5 Member Function Description

- `void add_property_info (Dictionary info)`

Add a custom property info to a property. The dictionary must contain: name:String), and optionally hint:int), hint_string:String.

Example:

```
editor_settings.set("category/property_name", 0)

var property_info = {
    "name": "category/property_name",
    "type": TYPE_INT,
    "hint": PROPERTY_HINT_ENUM,
    "hint_string": "one,two,three"
}

editor_settings.add_property_info(property_info)
```

- `void erase (String property)`

Erase a given setting (pass full property path).

- `PoolStringArray get_favorite_dirs () const`

Get the list of favorite directories for this project.

- `String get_project_settings_dir () const`

Get the specific project settings path. Projects all have a unique sub-directory inside the settings path where project specific settings are saved.

- `PoolStringArray get_recent_dirs () const`

Get the list of recently visited folders in the file dialog for this project.

- `Variant get_setting (String name) const`

- `String get_settings_dir () const`

Get the global settings path for the engine. Inside this path you can find some standard paths such as:

settings/tmp - used for temporary storage of files

settings/templates - where export templates are located

- `bool has_setting (String name) const`
- `bool property_can_revert (String name)`
- `Variant property_get_revert (String name)`

- void **set_favorite_dirs** (*PoolStringArray dirs*)

Set the list of favorite directories for this project.

- void **set_initial_value** (*String name, Variant value, bool update_current*)
- void **set_recent_dirs** (*PoolStringArray dirs*)

Set the list of recently visited folders in the file dialog for this project.

- void **set_setting** (*String name, Variant value*)

25.138 EditorSpatialGizmo

Inherits: *SpatialGizmo < Reference < Object*

Category: Core

25.138.1 Brief Description

Custom gizmo for editing Spatial objects.

25.138.2 Member Functions

void	<i>add_collision_segments</i> (<i>PoolVector3Array segments</i>)
void	<i>add_collision_triangles</i> (<i>TriangleMesh triangles, AABB bounds</i>)
void	<i>add_handles</i> (<i>PoolVector3Array handles, bool billboard=false, bool secondary=false</i>)
void	<i>add_lines</i> (<i>PoolVector3Array lines, Material material, bool billboard=false</i>)
void	<i>add_mesh</i> (<i>ArrayMesh mesh, bool billboard=false, RID skeleton</i>)
void	<i>add_unscaled_billboard</i> (<i>Material material, float default_scale=1</i>)
void	<i>clear</i> ()
void	<i>commit_handle</i> (<i>int index, Variant restore, bool cancel=false</i>) virtual
<i>String</i>	<i>get_handle_name</i> (<i>int index</i>) virtual
<i>Variant</i>	<i>get_handle_value</i> (<i>int index</i>) virtual
void	<i>redraw</i> () virtual
void	<i>set_handle</i> (<i>int index, Camera camera, Vector2 point</i>) virtual
void	<i>set_spatial_node</i> (<i>Node node</i>)

25.138.3 Description

Custom gizmo that is used for providing custom visualization and editing (handles) for 3D Spatial objects. These are created by *EditorPlugin.create_spatial_gizmo*.

25.138.4 Member Function Description

- void **add_collision_segments** (*PoolVector3Array segments*)
- void **add_collision_triangles** (*TriangleMesh triangles, AABB bounds*)

Add collision triangles to the gizmo for picking. A *TriangleMesh* can be generated from a regular *Mesh* too. Call this function during *redraw*.

- void **add_handles** (*PoolVector3Array* handles, *bool* billboard=false, *bool* secondary=false)

Add a list of handles (points) which can be used to deform the object being edited.

There are virtual functions which will be called upon editing of these handles. Call this function during *redraw*.

- void **add_lines** (*PoolVector3Array* lines, *Material* material, *bool* billboard=false)

Add lines to the gizmo (as sets of 2 points), with a given material. The lines are used for visualizing the gizmo. Call this function during *redraw*.

- void **add_mesh** (*ArrayMesh* mesh, *bool* billboard=false, *RID* skeleton)
- void **add_unscaled_billboard** (*Material* material, *float* default_scale=1)

Add an unscaled billboard for visualization. Call this function during *redraw*.

- void **clear** ()
- void **commit_handle** (*int* index, *Variant* restore, *bool* cancel=false) virtual

Commit a handle being edited (handles must have been previously added by *add_handles*).

If the cancel parameter is true, an option to restore the edited value to the original is provided.

- *String* **get_handle_name** (*int* index) virtual

Get the name of an edited handle (handles must have been previously added by *add_handles*).

Handles can be named for reference to the user when editing.

- *Variant* **get_handle_value** (*int* index) virtual

Get actual value of a handle. This value can be anything and used for eventually undoing the motion when calling *commit_handle*

- void **redraw** () virtual

This function is called when the Spatial this gizmo refers to changes (the *Spatial.update_gizmo* is called).

- void **set_handle** (*int* index, *Camera* camera, *Vector2* point) virtual

This function is used when the user drags a gizmo handle (previously added with *add_handles*) in screen coordinates.

The *Camera* is also provided so screen coordinates can be converted to raycasts.

- void **set_spatial_node** (*Node* node)

25.139 EncodedObjectAsID

Inherits: *Reference* < *Object*

Category: Core

25.139.1 Brief Description

25.139.2 Member Functions

<i>int</i>	<i>get_object_id</i> () const
<i>void</i>	<i>set_object_id</i> (<i>int</i> id)

25.139.3 Member Function Description

- `int get_object_id() const`
- `void set_object_id(int id)`

25.140 Engine

Inherits: *Object*

Category: Core

25.140.1 Brief Description

Access to basic engine properties.

25.140.2 Member Functions

<code>int</code>	<code>get_frames_drawn()</code>
<code>float</code>	<code>get_frames_per_second() const</code>
<code>MainLoop</code>	<code>get_main_loop() const</code>
<code>Object</code>	<code>get_singleton(String name) const</code>
<code>Dictionary</code>	<code>get_version_info() const</code>
<code>bool</code>	<code>has_singleton(String name) const</code>
<code>bool</code>	<code>is_in_physics_frame() const</code>

25.140.3 Member Variables

- `bool editor_hint` - If `true`, it is running inside the editor. Useful for tool scripts.
- `int iterations_per_second` - The number of fixed iterations per second (for fixed process and physics).
- `int target_fps` - The desired frames per second. If the hardware cannot keep up, this setting may not be respected. Defaults to 0, which indicates no limit.
- `float time_scale` - Controls how fast or slow the in-game clock ticks versus the real life one. It defaults to 1.0. A value of 2.0 means the game moves twice as fast as real life, whilst a value of 0.5 means the game moves at half the regular speed.

25.140.4 Description

The Engine class allows you to query and modify the game's run-time parameters, such as frames per second, time scale, and others.

25.140.5 Member Function Description

- `int get_frames_drawn ()`

Returns the total number of frames drawn.

- `float get_frames_per_second () const`

Returns the frames per second of the running game.

- `MainLoop get_main_loop () const`

Returns the main loop object (see `MainLoop` and `SceneTree`).

- `Object get_singleton (String name) const`

- `Dictionary get_version_info () const`

Returns the current engine version information in a Dictionary.

“major” - Holds the major version number as a String

“minor” - Holds the minor version number as a String

“patch” - Holds the patch version number as a String

“status” - Holds the status (e.g. “beta”, “rc1”, “rc2”, … “stable”) as a String

“build” - Holds the build name (e.g. “custom-build”) as a String

“string” - major + minor + patch + status + build in a single String

- `bool has_singleton (String name) const`

- `bool is_in_physics_frame () const`

Returns `true` if the game is inside the fixed process and physics phase of the game loop.

25.141 Environment

Inherits: `Resource < Reference < Object`

Category: Core

25.141.1 Brief Description

Resource for environment nodes (like `WorldEnvironment`) that define multiple rendering options.

25.141.2 Member Variables

- `float adjustment_brightness` - Global brightness value of the rendered scene (default value is 1).
- `Texture adjustment_color_correction` - Applies the provided `Texture` resource to affect the global color aspect of the rendered scene.
- `float adjustment_contrast` - Global contrast value of the rendered scene (default value is 1).
- `bool adjustment_enabled` - Enables the `adjustment_*` options provided by this resource. If false, adjustments modifications will have no effect on the rendered scene.

- *float adjustment_saturation* - Global color saturation value of the rendered scene (default value is 1).
- *Color ambient_light_color* - *Color* of the ambient light.
- *float ambient_light_energy* - Energy of the ambient light. The higher the value, the stronger the light.
- *float ambient_light_sky_contribution* - Defines the amount of light that the sky brings on the scene. A value of 0 means that the sky's light emission has no effect on the scene illumination, thus all ambient illumination is provided by the ambient light. On the contrary, a value of 1 means that all the light that affects the scene is provided by the sky, thus the ambient light parameter has no effect on the scene.
- *bool auto_exposure_enabled* - Enables the tonemapping auto exposure mode of the scene renderer. If activated, the renderer will automatically determine the exposure setting to adapt to the illumination of the scene and the observed light.
- *float auto_exposure_max_luma* - Maximum luminance value for the auto exposure.
- *float auto_exposure_min_luma* - Minimum luminance value for the auto exposure.
- *float auto_exposure_scale* - Scale of the auto exposure effect. Affects the intensity of auto exposure.
- *float auto_exposure_speed* - Speed of the auto exposure effect. Affects the time needed for the camera to perform auto exposure.
- *int background_canvas_max_layer* - Maximum layer id (if using Layer background mode).
- *Color background_color* - Color displayed for clear areas of the scene (if using Custom color or Color+Sky background modes).
- *float background_energy* - Power of light emitted by the background.
- *BGMode background_mode* - Defines the mode of background.
- *Sky background_sky* - *Sky* resource defined as background.
- *float background_sky_custom_fov* - *Sky* resource's custom field of view.
- *float dof_blur_far_amount* - Amount of far blur.
- *float dof_blur_far_distance* - Distance from the camera where the far blur effect affects the rendering.
- *bool dof_blur_far_enabled* - Enables the far blur effect.
- *DOFBlurQuality dof_blur_far_quality* - Quality of the far blur quality.
- *float dof_blur_far_transition* - Transition between no-blur area and far blur.
- *float dof_blur_near_amount* - Amount of near blur.
- *float dof_blur_near_distance* - Distance from the camera where the near blur effect affects the rendering.
- *bool dof_blur_near_enabled* - Enables the near blur effect.
- *DOFBlurQuality dof_blur_near_quality* - Quality of the near blur quality.
- *float dof_blur_near_transition* - Transition between near blur and no-blur area.
- *Color fog_color* - Fog's *Color*.
- *float fog_depth_begin* - Fog's depth starting distance from the camera.
- *float fog_depth_curve* - Value defining the fog depth intensity.
- *bool fog_depth_enabled* - Enables the fog depth.
- *bool fog_enabled* - Enables the fog. Needs fog_height_enabled and/or for_depth_enabled to actually display fog.
- *float fog_height_curve* - Value defining the fog height intensity.

- *bool fog_height_enabled* - Enables the fog height.
- *float fog_height_max* - Maximum height of fog.
- *float fog_height_min* - Minimum height of fog.
- *float fog_sun_amount* - Amount of sun that affects the fog rendering.
- *Color fog_sun_color* - Sun Color.
- *float fog_transmit_curve* - Amount of light that the fog transmits.
- *bool fog_transmit_enabled* - Enables fog's light transmission. If enabled, lets reflections light to be transmitted by the fog.
- *bool glow_bicubic_upscale*
- *GlowBlendMode glow_blend_mode* - Glow blending mode.
- *float glow_bloom* - Bloom value (global glow).
- *bool glow_enabled* - Enables glow rendering.
- *float glow_hdr_scale* - Bleed scale of the HDR glow.
- *float glow_hdr_threshold* - Bleed threshold of the HDR glow.
- *float glow_intensity* - Glow intensity.
- *bool glow_levels/1* - First level of glow (most local).
- *bool glow_levels/2* - Second level of glow.
- *bool glow_levels/3* - Third level of glow.
- *bool glow_levels/4* - Fourth level of glow.
- *bool glow_levels/5* - Fifth level of glow.
- *bool glow_levels/6* - Sixth level of glow.
- *bool glow_levels/7* - Seventh level of glow (most global).
- *float glow_strength* - Glow strength.
- *float ss_reflections_depth_tolerance*
- *bool ss_reflections_enabled*
- *float ss_reflections_fade_in*
- *float ss_reflections_fade_out*
- *int ss_reflections_max_steps*
- *bool ss_reflections_roughness*
- *float ssao_bias*
- *SSAOBlur ssao_blur*
- *Color ssao_color*
- *float ssao_edge_sharpness*
- *bool ssao_enabled*
- *float ssao_intensity*
- *float ssao_intensity2*

- *float ssao_light_affect*
- *SSAOQuality ssao_quality*
- *float ssao_radius*
- *float ssao_radius2*
- *float tonemap_exposure* - Default exposure for tonemap.
- *ToneMapper tonemap_mode* - Tonemapping mode.
- *float tonemap_white* - White reference value for tonemap.

25.141.3 Enums

enum **BGMode**

- **BG_KEEP = 5** — Keep on screen every pixel drawn in the background.
- **BG_CLEAR_COLOR = 0** — Clear the background using the project's clear color.
- **BG_COLOR = 1** — Clear the background using a custom clear color.
- **BG_SKY = 2** — Display a user-defined sky in the background.
- **BG_COLOR_SKY = 3** — Clear the background using a custom clear color and allows defining a sky for shading and reflection.
- **BG_CANVAS = 4** — Display a *CanvasLayer* in the background.
- **BG_MAX = 6** — Helper constant keeping track of the enum's size, has no direct usage in API calls.

enum **DOFBlurQuality**

- **DOF_BLUR_QUALITY_LOW = 0** — Low depth-of-field blur quality.
- **DOF_BLUR_QUALITY_MEDIUM = 1** — Medium depth-of-field blur quality.
- **DOF_BLUR_QUALITY_HIGH = 2** — High depth-of-field blur quality.

enum **GlowBlendMode**

- **GLOW_BLEND_MODE_ADDITIVE = 0** — Additive glow blending mode. Mostly used for particles, glows (bloom), lens flare, bright sources.
- **GLOW_BLEND_MODE_SCREEN = 1** — Screen glow blending mode. Increases brightness, used frequently with bloom.
- **GLOW_BLEND_MODE_SOFTLIGHT = 2** — Softlight glow blending mode. Modifies contrast, exposes shadows and highlights, vivid bloom.
- **GLOW_BLEND_MODE_REPLACE = 3** — Replace glow blending mode. Replaces all pixels' color by the glow value.

enum **ToneMapper**

- **TONE_MAPPER_LINEAR = 0** — Linear tonemapper operator. Reads the linear data and performs an exposure adjustment.
- **TONE_MAPPER_REINHARDT = 1** — Reinhardt tonemapper operator. Performs a variation on rendered pixels' colors by this formula: $\text{color} = \text{color} / (1 + \text{color})$.
- **TONE_MAPPER_FILMIC = 2** — Filmic tonemapper operator.

- **TONE_MAPPER_ACES = 3** — Academy Color Encoding System tonemapper operator.

enum SSAOBlur

- **SSAO_BLUR_DISABLED = 0**
- **SSAO_BLUR_1x1 = 1**
- **SSAO_BLUR_2x2 = 2**
- **SSAO_BLUR_3x3 = 3**

enum SSAOQuality

- **SSAO_QUALITY_LOW = 0**
- **SSAO_QUALITY_MEDIUM = 1**
- **SSAO_QUALITY_HIGH = 2**

25.141.4 Description

Resource for environment nodes (like *WorldEnvironment*) that define multiple environment operations (such as background *Sky* or *Color*, ambient light, fog, depth-of-field...). These parameters affect the final render of the scene. The order of these operations is:

- DOF Blur
- Motion Blur
- Bloom
- Tonemap (auto exposure)
- Adjustments

25.142 File

Inherits: *Reference < Object*

Category: Core

25.142.1 Brief Description

Type to handle file reading and writing operations.

25.142.2 Member Functions

<code>void</code>	<code>close ()</code>
<code>bool</code>	<code>eof_reached () const</code>
<code>bool</code>	<code>file_exists (String path) const</code>
<code>int</code>	<code>get_16 () const</code>
<code>int</code>	<code>get_32 () const</code>
<code>int</code>	<code>get_64 () const</code>
<code>int</code>	<code>get_8 () const</code>

Continued on next page

Table 25.9 – continued from previous page

<i>String</i>	<code>get_as_text() const</code>
<i>PoolByteArray</i>	<code>get_buffer(int len) const</code>
<i>PoolStringArray</i>	<code>get_csv_line(String delim=",") const</code>
<i>float</i>	<code>get_double() const</code>
<i>int</i>	<code>get_error() const</code>
<i>float</i>	<code>get_float() const</code>
<i>int</i>	<code>get_len() const</code>
<i>String</i>	<code>get_line() const</code>
<i>String</i>	<code>get_md5(String path) const</code>
<i>int</i>	<code>get_modified_time(String file) const</code>
<i>String</i>	<code>get_pascal_string()</code>
<i>int</i>	<code>get_position() const</code>
<i>float</i>	<code>get_real() const</code>
<i>String</i>	<code>get_sha256(String path) const</code>
<i>Variant</i>	<code>get_var() const</code>
<i>bool</i>	<code>is_open() const</code>
<i>int</i>	<code>open(String path, int flags)</code>
<i>int</i>	<code>open_compressed(String path, int mode_flags, int compression_mode=0)</code>
<i>int</i>	<code>open_encrypted(String path, int mode_flags, PoolByteArray key)</code>
<i>int</i>	<code>open_encrypted_with_pass(String path, int mode_flags, String pass)</code>
<i>void</i>	<code>seek(int position)</code>
<i>void</i>	<code>seek_end(int position=0)</code>
<i>void</i>	<code>store_16(int value)</code>
<i>void</i>	<code>store_32(int value)</code>
<i>void</i>	<code>store_64(int value)</code>
<i>void</i>	<code>store_8(int value)</code>
<i>void</i>	<code>store_buffer(PoolByteArray buffer)</code>
<i>void</i>	<code>store_double(float value)</code>
<i>void</i>	<code>store_float(float value)</code>
<i>void</i>	<code>store_line(String line)</code>
<i>void</i>	<code>store_pascal_string(String string)</code>
<i>void</i>	<code>store_real(float value)</code>
<i>void</i>	<code>store_string(String string)</code>
<i>void</i>	<code>store_var(Variant value)</code>

25.142.3 Member Variables

- *bool endian_swap* - If `true` the file's endianness is swapped. Use this if you're dealing with files written in big endian machines.

Note that this is about the file format, not CPU type. This is always reseted to `false` whenever you open the file.

25.142.4 Enums

enum CompressionMode

- **COMPRESSION_FASTLZ = 0** — Uses the FastLZ compression method.
- **COMPRESSION_DEFLATE = 1** — Uses the Deflate compression method.

- **COMPRESSION_ZSTD = 2** — Uses the Zstd compression method.
- **COMPRESSION_GZIP = 3** — Uses the gzip compression method.

enum **ModeFlags**

- **READ = 1** — Opens the file for read operations.
- **WRITE = 2** — Opens the file for write operations. Create it if the file does not exist and truncate if it exists.
- **READ_WRITE = 3** — Opens the file for read and write operations. Does not truncate the file.
- **WRITE_READ = 7** — Opens the file for read and write operations. Create it if the file does not exist and truncate if it exists.

25.142.5 Description

File type. This is used to permanently store data into the user device's file system and to read from it. This can be used to store game save data or player configuration files, for example.

Here's a sample on how to write and read from a file:

```
func save(content):
    var file = File.new()
    file.open("user://save_game.dat", file.WRITE)
    file.store_string(content)
    file.close()

func load():
    var file = File.new()
    file.open("user://save_game.dat", file.READ)
    var content = file.get_as_text()
    file.close()
    return content
```

25.142.6 Member Function Description

- void **close ()**

Closes the currently opened file.

- *bool* **eof_reached () const**

Returns `true` if the file cursor has reached the end of the file.

- *bool* **file_exists (String path) const**

Returns `true` if the file exists in the given path.

- *int* **get_16 () const**

Returns the next 16 bits from the file as an integer.

- *int* **get_32 () const**

Returns the next 32 bits from the file as an integer.

- *int* **get_64 () const**

Returns the next 64 bits from the file as an integer.

- *int* **get_8 () const**

Returns the next 8 bits from the file as an integer.

- `String get_as_text() const`

Returns the whole file as a *String*.

- `PoolByteArray get_buffer(int len) const`

Returns next `len` bytes of the file as a *PoolByteArray*.

- `PoolStringArray get_csv_line(String delim=",") const`

Returns the next value of the file in CSV (Comma Separated Values) format. You can pass a different delimiter to use other than the default “,” (comma).

- `float get_double() const`

Returns the next 64 bits from the file as a floating point number.

- `int get_error() const`

Returns the last error that happened when trying to perform operations. Compare with the `ERR_FILE_*` constants from `@GlobalScope`.

- `float get_float() const`

Returns the next 32 bits from the file as a floating point number.

- `int get_len() const`

Returns the size of the file in bytes.

- `String get_line() const`

Returns the next line of the file as a *String*.

- `String get_md5(String path) const`

Returns an MD5 *String* representing the file at the given path or an empty *String* on failure.

- `int get_modified_time(String file) const`

Returns the last time the `file` was modified in unix timestamp format or returns a *String* “ERROR IN `file`”. This unix timestamp can be converted to datetime by using `OS.get_datetime_from_unix_time`.

- `String get_pascal_string()`

Returns a *String* saved in Pascal format from the file.

- `int get_position() const`

Returns the file cursor’s position.

- `float get_real() const`

Returns the next bits from the file as a floating point number.

- `String get_sha256(String path) const`

Returns a SHA-256 *String* representing the file at the given path or an empty *String* on failure.

- `Variant get_var() const`

Returns the next *Variant* value from the file.

- `bool is_open() const`

Returns `true` if the file is currently opened.

- `int open(String path, int flags)`

Opens the file for writing or reading, depending on the flags.

- `int open_compressed (String path, int mode_flags, int compression_mode=0)`

Opens a compressed file for reading or writing. Use COMPRESSION_* constants to set `compression_mode`.

- `int open_encrypted (String path, int mode_flags, PoolByteArray key)`

Opens an encrypted file in write or read mode. You need to pass a binary key to encrypt/decrypt it.

- `int open_encrypted_with_pass (String path, int mode_flags, String pass)`

Opens an encrypted file in write or read mode. You need to pass a password to encrypt/decrypt it.

- `void seek (int position)`

Change the file reading/writing cursor to the specified position (in bytes from the beginning of the file).

- `void seek_end (int position=0)`

Changes the file reading/writing cursor to the specified position (in bytes from the end of the file). Note that this is an offset, so you should use negative numbers or the cursor will be at the end of the file.

- `void store_16 (int value)`

Stores an integer as 16 bits in the file.

- `void store_32 (int value)`

Stores an integer as 32 bits in the file.

- `void store_64 (int value)`

Stores an integer as 64 bits in the file.

- `void store_8 (int value)`

Stores an integer as 8 bits in the file.

- `void store_buffer (PoolByteArray buffer)`

Stores the given array of bytes in the file.

- `void store_double (float value)`

Stores a floating point number as 64 bits in the file.

- `void store_float (float value)`

Stores a floating point number as 32 bits in the file.

- `void store_line (String line)`

Stores the given `String` as a line in the file.

- `void store_pascal_string (String string)`

Stores the given `String` as a line in the file in Pascal format (i.e. also store the length of the string).

- `void store_real (float value)`

Stores a floating point number in the file.

- `void store_string (String string)`

Stores the given `String` in the file.

- `void store_var (Variant value)`

Stores any `Variant` value in the file.

25.143 FileDialog

Inherits: *ConfirmationDialog < AcceptDialog < WindowDialog < Popup < Control < CanvasItem < Node < Object*

Category: Core

25.143.1 Brief Description

Dialog for selecting files or directories in the filesystem.

25.143.2 Member Functions

void	<i>add_filter (String filter)</i>
void	<i>clear_filters ()</i>
void	<i>deselect_items ()</i>
VBoxContainer	<i>get_vbox ()</i>
void	<i>invalidate ()</i>

25.143.3 Signals

- **dir_selected (String dir)**

Event emitted when the user selects a directory.

- **file_selected (String path)**

Event emitted when the user selects a file (double clicks it or presses the OK button).

- **files_selected (PoolStringArray paths)**

Event emitted when the user selects multiple files.

25.143.4 Member Variables

- **Access access**
- **String current_dir** - The current working directory of the file dialog.
- **String current_file** - The currently selected file of the file dialog.
- **String current_path** - The currently selected file path of the file dialog.
- **PoolStringArray filters**
- **Mode mode**
- **bool mode_overrides_title** - If true, changing the mode property will set the window title accordingly (e. g. setting mode to MODE_OPEN_FILE will change the window title to “Open a File”).
- **bool show_hidden_files**

25.143.5 Enums

enum Access

- **ACCESS_RESOURCES = 0** — The dialog allows the selection of file and directory.
- **ACCESS_USERDATA = 1** — The dialog allows access files under *Resource* path(`res://`) .
- **ACCESS_FILESYSTEM = 2** — The dialog allows access files in whole file system.

enum Mode

- **MODE_OPEN_FILE = 0** — The dialog allows the selection of one, and only one file.
- **MODE_OPEN_FILES = 1** — The dialog allows the selection of multiple files.
- **MODE_OPEN_DIR = 2** — The dialog functions as a folder selector, disallowing the selection of any file.
- **MODE_OPEN_ANY = 3** — The dialog allows the selection of a file or a directory.
- **MODE_SAVE_FILE = 4** — The dialog will warn when a file exists.

25.143.6 Description

FileDialog is a preset dialog used to choose files and directories in the filesystem. It supports filter masks.

25.143.7 Member Function Description

- **void add_filter (String filter)**

Add a custom filter. Filter format is: “mask ; description”, example (C++): `dialog->add_filter(“*.png ; PNG Images”);`

- **void clear_filters ()**

Clear all the added filters in the dialog.

- **void deselect_items ()**
- **VBoxContainer get_vbox ()**

Return the vertical box container of the dialog, custom controls can be added to it.

- **void invalidate ()**

Invalidate and update the current dialog content list.

25.144 float

Category: Built-In Types

25.144.1 Brief Description

Float built-in type

25.144.2 Member Functions

<i>float</i>	<code>float (bool from)</code>
<i>float</i>	<code>float (int from)</code>
<i>float</i>	<code>float (String from)</code>

25.144.3 Description

Float built-in type.

25.144.4 Member Function Description

- `float float (bool from)`

Cast a `bool` value to a floating point value, `float (true)` will be equals to 1.0 and `float (false)` will be equals to 0.0.

- `float float (int from)`

Cast an `int` value to a floating point value, `float (1)` will be equals to 1.0.

- `float float (String from)`

Cast a `String` value to a floating point value. This method accepts float value strings like “ ‘1.23’ ” and exponential notation strings for its parameter so calling “ `float(‘1e3’)` ” will return 1000.0 and calling “ `float(‘1e-3’)` ” will return -0.001.

25.145 Font

Inherits: *Resource < Reference < Object*

Inherited By: *DynamicFont, BitmapFont*

Category: Core

25.145.1 Brief Description

Internationalized font and text drawing support.

25.145.2 Member Functions

void	<code>draw (RID canvas_item, Vector2 position, String string, Color modulate=Color(1, 1, 1, 1), int clip_w=-1) const</code>
float	<code>draw_char (RID canvas_item, Vector2 position, int char, int next=-1, Color modulate=Color(1, 1, 1, 1)) const</code>
float	<code>get_ascent () const</code>
float	<code>get_descent () const</code>
float	<code>get_height () const</code>
Vector2	<code>get_string_size (String string) const</code>
bool	<code>is_distance_field_hint () const</code>
void	<code>update_changes ()</code>

25.145.3 Description

Font contains a unicode compatible character set, as well as the ability to draw it with variable width, ascent, descent and kerning. For creating fonts from TTF files (or other font formats), see the editor support for fonts. TODO check wikipedia for graph of ascent/baseline/descent/height/etc.

25.145.4 Member Function Description

- void `draw (RID canvas_item, Vector2 position, String string, Color modulate=Color(1, 1, 1, 1), int clip_w=-1) const`

Draw “string” into a canvas item using the font at a given position, with “modulate” color, and optionally clipping the width. “position” specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis.

- float `draw_char (RID canvas_item, Vector2 position, int char, int next=-1, Color modulate=Color(1, 1, 1, 1)) const`

Draw character “char” into a canvas item using the font at a given position, with “modulate” color, and optionally kerning if “next” is passed. clipping the width. “position” specifies the baseline, not the top. To draw from the top, *ascent* must be added to the Y axis. The width used by the character is returned, making this function useful for drawing strings character by character.

- float `get_ascent () const`

Return the font ascent (number of pixels above the baseline).

- float `get_descent () const`

Return the font descent (number of pixels below the baseline).

- float `get_height () const`

Return the total font height (ascent plus descent) in pixels.

- Vector2 `get_string_size (String string) const`

Return the size of a string, taking kerning and advance into account.

- bool `is_distance_field_hint () const`

- void `update_changes ()`

After editing a font (changing size, ascent, char rects, etc.). Call this function to propagate changes to controls that might use it.

25.146 FuncRef

Inherits: *Reference < Object*

Category: Core

25.146.1 Brief Description

Reference to a function in an object.

25.146.2 Member Functions

<i>Variant</i>	<code>call_func () vararg</code>
<code>void</code>	<code>set_function (String name)</code>
<code>void</code>	<code>set_instance (Object instance)</code>

25.146.3 Description

In GDScript, functions are not *first-class objects*. This means it is impossible to store them directly as variables, return them from another function, or pass them as arguments.

However, by creating a `FuncRef` using the `@GDScript.funcref` function, a reference to a function in a given object can be created, passed around and called.

25.146.4 Member Function Description

- *Variant* `call_func () vararg`

Calls the referenced function previously set by `set_function` or `@GDScript.funcref`.

- `void set_function (String name)`

The name of the referenced function to call on the object, without parentheses or any parameters.

- `void set_instance (Object instance)`

The object containing the referenced function. This object must be of a type actually inheriting from `Object`, not a built-in type such as `int`, `Vector2` or `Dictionary`.

25.147 GDNative

Inherits: *Reference < Object*

Category: Core

25.147.1 Brief Description

25.147.2 Member Functions

<i>Variant</i>	<i>call_native (String calling_type, String procedure_name, Array arguments)</i>
<i>bool</i>	<i>initialize ()</i>
<i>bool</i>	<i>terminate ()</i>

25.147.3 Member Variables

- *GDNativeLibrary library*

25.147.4 Member Function Description

- *Variant call_native (String calling_type, String procedure_name, Array arguments)*
- *bool initialize ()*
- *bool terminate ()*

25.148 GDNativeLibrary

Inherits: *Resource < Reference < Object*

Category: Core

25.148.1 Brief Description

25.148.2 Member Functions

<i>ConfigFile</i>	<i>get_config_file ()</i>
<i>PoolStringArray</i>	<i>get_current_dependencies () const</i>
<i>String</i>	<i>get_current_library_path () const</i>

25.148.3 Member Variables

- *bool load_once*
- *bool reloadable*
- *bool singleton*
- *String symbol_prefix*

25.148.4 Member Function Description

- *ConfigFile* **get_config_file ()**
- *PoolStringArray* **get_current_dependencies () const**
- *String* **get_current_library_path () const**

25.149 GDScript

Inherits: *Script < Resource < Reference < Object*

Category: Core

25.149.1 Brief Description

A script implemented in the GDScript programming language.

25.149.2 Member Functions

<i>PoolByteArray</i>	get_as_byte_code () const
<i>Object</i>	new () vararg

25.149.3 Description

A script implemented in the GDScript programming language. The script extends the functionality of all objects that instance it.

new creates a new instance of the script. *Object.set_script* extends an existing object, if that object's class matches one of the script's base classes.

25.149.4 Member Function Description

- *PoolByteArray* **get_as_byte_code () const**

Returns byte code for the script source code.

- *Object* **new () vararg**

Returns a new instance of the script.

For example:

```
var MyClass = load("myclass.gd")
var instance = MyClass.new()
assert(instance.get_script() == MyClass)
```

25.150 GDScriptFunctionState

Inherits: *Reference < Object*

Category: Core

25.150.1 Brief Description

State of a function call after yielding.

25.150.2 Member Functions

<i>bool</i>	<i>is_valid</i> (<i>bool extended_check=false</i>) const
<i>Variant</i>	<i>resume</i> (<i>Variant arg=null</i>)

25.150.3 Signals

- **completed** (*Nil* result)

25.150.4 Description

Calling `@GDScript.yield` within a function will cause that function to yield and return its current state as an object of this type. The yielded function call can then be resumed later by calling `resume` on this state object.

25.150.5 Member Function Description

- *bool is_valid* (*bool extended_check=false*) const

Check whether the function call may be resumed. This is not the case if the function state was already resumed.

If `extended_check` is enabled, it also checks if the associated script and object still exist. The extended check is done in debug mode as part of `GDScriptFunctionState.resume`, but you can use this if you know you may be trying to resume without knowing for sure the object and/or script have survived up to that point.

- *Variant resume* (*Variant arg=null*)

Resume execution of the yielded function call.

If handed an argument, return the argument from the `@GDScript.yield` call in the yielded function call. You can pass e.g. an `Array` to hand multiple arguments.

This function returns what the resumed function call returns, possibly another function state if yielded again.

25.151 GDScriptNativeClass

Inherits: *Reference < Object*

Category: Core

25.151.1 Brief Description

25.151.2 Member Functions

<i>Variant</i>	<i>new ()</i>
----------------	---------------

25.151.3 Member Function Description

- *Variant new ()*

25.152 Generic6DOFJoint

Inherits: *Joint < Spatial < Node < Object*

Category: Core

25.152.1 Brief Description

The generic 6 degrees of freedom joint can implement a variety of joint-types by locking certain axes' rotation or translation.

25.152.2 Member Variables

- *float angular_limit_x/damping* - The amount of rotational damping across the x-axis.

The lower, the longer an impulse from one side takes to travel to the other side.

- *bool angular_limit_x/enabled* - If true rotation across the x-axis is enabled.
- *float angular_limit_x/erp* - When rotating across x-axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.
- *float angular_limit_x/force_limit* - The maximum amount of force that can occur, when rotating around x-axis.
- *float angular_limit_x/lower_angle* - The minimum rotation in negative direction to break loose and rotate around the x-axis.
- *float angular_limit_x/restitution* - The amount of rotational restitution across the x-axis. The lower, the more restitution occurs.
- *float angular_limit_x/softness* - The speed of all rotations across the x-axis.
- *float angular_limit_x/upper_angle* - The minimum rotation in positive direction to break loose and rotate around the x-axis.
- *float angular_limit_y/damping* - The amount of rotational damping across the y-axis. The lower, the more dampening occurs.
- *bool angular_limit_y/enabled* - If true rotation across the y-axis is enabled.

- *float angular_limit_y/erp* - When rotating across y-axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.
- *float angular_limit_y/force_limit* - The maximum amount of force that can occur, when rotating around y-axis.
- *float angular_limit_y/lower_angle* - The minimum rotation in negative direction to break loose and rotate around the y-axis.
- *float angular_limit_y/restitution* - The amount of rotational restitution across the y-axis. The lower, the more restitution occurs.
- *float angular_limit_y/softness* - The speed of all rotations across the y-axis.
- *float angular_limit_y/upper_angle* - The minimum rotation in positive direction to break loose and rotate around the y-axis.
- *float angular_limit_z/damping* - The amount of rotational damping across the z-axis. The lower, the more dampening occurs.
- *bool angular_limit_z/enabled* - If true rotation across the z-axis is enabled.
- *float angular_limit_z/erp* - When rotating across z-axis, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.
- *float angular_limit_z/force_limit* - The maximum amount of force that can occur, when rotating around z-axis.
- *float angular_limit_z/lower_angle* - The minimum rotation in negative direction to break loose and rotate around the z-axis.
- *float angular_limit_z/restitution* - The amount of rotational restitution across the z-axis. The lower, the more restitution occurs.
- *float angular_limit_z/softness* - The speed of all rotations across the z-axis.
- *float angular_limit_z/upper_angle* - The minimum rotation in positive direction to break loose and rotate around the z-axis.
- *bool angular_motor_x/enabled* - If true a rotating motor at the x-axis is enabled.
- *float angular_motor_x/force_limit* - Maximum acceleration for the motor at the x-axis.
- *float angular_motor_x/target_velocity* - Target speed for the motor at the x-axis.
- *bool angular_motor_y/enabled* - If true a rotating motor at the y-axis is enabled.
- *float angular_motor_y/force_limit* - Maximum acceleration for the motor at the y-axis.
- *float angular_motor_y/target_velocity* - Target speed for the motor at the y-axis.
- *bool angular_motor_z/enabled* - If true a rotating motor at the z-axis is enabled.
- *float angular_motor_z/force_limit* - Maximum acceleration for the motor at the z-axis.
- *float angular_motor_z/target_velocity* - Target speed for the motor at the z-axis.
- *float linear_limit_x/damping* - The amount of damping that happens at the x-motion.
- *bool linear_limit_x/enabled* - If true the linear motion across the x-axis is enabled.
- *float linear_limit_x/lower_distance* - The minimum difference between the pivot points' x-axis.
- *float linear_limit_x/restitution* - The amount of restitution on the x-axis movement The lower, the more momentum gets lost.
- *float linear_limit_x/softness* - A factor applied to the movement across the x-axis The lower, the slower the movement.
- *float linear_limit_x/upper_distance* - The maximum difference between the pivot points' x-axis.

- *float linear_limit_y/damping* - The amount of damping that happens at the y-motion.
- *bool linear_limit_y/enabled* - If `true` the linear motion across the y-axis is enabled.
- *float linear_limit_y/lower_distance* - The minimum difference between the pivot points' y-axis.
- *float linear_limit_y/restitution* - The amount of restitution on the y-axis movement The lower, the more momentum gets lost.
- *float linear_limit_y/softness* - A factor applied to the movement across the y-axis The lower, the slower the movement.
- *float linear_limit_y/upper_distance* - The maximum difference between the pivot points' y-axis.
- *float linear_limit_z/damping* - The amount of damping that happens at the z-motion.
- *bool linear_limit_z/enabled* - If `true` the linear motion across the z-axis is enabled.
- *float linear_limit_z/lower_distance* - The minimum difference between the pivot points' z-axis.
- *float linear_limit_z/restitution* - The amount of restitution on the z-axis movement The lower, the more momentum gets lost.
- *float linear_limit_z/softness* - A factor applied to the movement across the z-axis The lower, the slower the movement.
- *float linear_limit_z/upper_distance* - The maximum difference between the pivot points' z-axis.

25.152.3 Enums

enum Flag

- **FLAG_ENABLE_LINEAR_LIMIT = 0** — If `set` there is linear motion possible within the given limits.
- **FLAG_ENABLE_ANGULAR_LIMIT = 1** — If `set` there is rotational motion possible.
- **FLAG_ENABLE_MOTOR = 2** — If `set` there is a rotational motor across these axes.
- **FLAG_MAX = 3** — End flag of FLAG_* constants, used internally.

enum Param

- **PARAM_LINEAR_LOWER_LIMIT = 0** — The minimum difference between the pivot points' axes.
- **PARAM_LINEAR_UPPER_LIMIT = 1** — The maximum difference between the pivot points' axes.
- **PARAM_LINEAR_LIMIT_SOFTNESS = 2** — A factor applied to the movement across the axes The lower, the slower the movement.
- **PARAM_LINEAR_RESTITUTION = 3** — The amount of restitution on the axes movement The lower, the more momentum gets lost.
- **PARAM_LINEAR_DAMPING = 4** — The amount of damping that happens at the linear motion across the axes.
- **PARAM_ANGULAR_LOWER_LIMIT = 5** — The minimum rotation in negative direction to break loose and rotate around the axes.
- **PARAM_ANGULAR_UPPER_LIMIT = 6** — The minimum rotation in positive direction to break loose and rotate around the axes.
- **PARAM_ANGULAR_LIMIT_SOFTNESS = 7** — The speed of all rotations across the axes.

- **PARAM_ANGULAR_DAMPING = 8** — The amount of rotational damping across the axes. The lower, the more dampening occurs.
- **PARAM_ANGULAR_RESTITUTION = 9** — The amount of rotational restitution across the axes. The lower, the more restitution occurs.
- **PARAM_ANGULAR_FORCE_LIMIT = 10** — The maximum amount of force that can occur, when rotating around the axes.
- **PARAM_ANGULAR_ERP = 11** — When rotating across the axes, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.
- **PARAM_ANGULAR_MOTOR_TARGET_VELOCITY = 12** — Target speed for the motor at the axes.
- **PARAM_ANGULAR_MOTOR_FORCE_LIMIT = 13** — Maximum acceleration for the motor at the axes.
- **PARAM_MAX = 14** — End flag of PARAM_* constants, used internally.

25.152.4 Description

The first 3 dof axes are linear axes, which represent translation of Bodies, and the latter 3 dof axes represent the angular motion. Each axis can be either locked, or limited.

25.153 Geometry

Inherits: *Object*

Category: Core

25.153.1 Brief Description

25.153.2 Member Functions

<code>Array</code>	<code>build_box_planes (Vector3 extents)</code>
<code>Array</code>	<code>build_capsule_planes (float radius, float height, int sides, int lats, int axis=2)</code>
<code>Array</code>	<code>build_cylinder_planes (float radius, float height, int sides, int axis=2)</code>
<code>PoolVector3Array</code>	<code>clip_polygon (PoolVector3Array points, Plane plane)</code>
<code>PoolVector2Array</code>	<code>convex_hull_2d (PoolVector2Array points)</code>
<code>Vector3</code>	<code>get_closest_point_to_segment (Vector3 point, Vector3 s1, Vector3 s2)</code>
<code>Vector2</code>	<code>get_closest_point_to_segment_2d (Vector2 point, Vector2 s1, Vector2 s2)</code>
<code>Vector3</code>	<code>get_closest_point_to_segment_uncapped (Vector3 point, Vector3 s1, Vector3 s2)</code>
<code>Vector2</code>	<code>get_closest_point_to_segment_uncapped_2d (Vector2 point, Vector2 s1, Vector2 s2)</code>
<code>PoolVector3Array</code>	<code>get_closest_points_between_segments (Vector3 p1, Vector3 p2, Vector3 q1, Vector3 q2)</code>
<code>PoolVector2Array</code>	<code>get_closest_points_between_segments_2d (Vector2 p1, Vector2 q1, Vector2 p2, Vector2 q2)</code>
<code>int</code>	<code>get_uv84_normal_bit (Vector3 normal)</code>
<code>Dictionary</code>	<code>make_atlas (PoolVector2Array sizes)</code>
<code>bool</code>	<code>point_is_inside_triangle (Vector2 point, Vector2 a, Vector2 b, Vector2 c) const</code>
<code>Variant</code>	<code>ray_intersects_triangle (Vector3 from, Vector3 dir, Vector3 a, Vector3 b, Vector3 c)</code>
<code>float</code>	<code>segment_intersects_circle (Vector2 segment_from, Vector2 segment_to, Vector2 circle_position, float circle_radius)</code>
<code>PoolVector3Array</code>	<code>segment_intersects_convex (Vector3 from, Vector3 to, Array planes)</code>
<code>PoolVector3Array</code>	<code>segment_intersects_cylinder (Vector3 from, Vector3 to, float height, float radius)</code>
<code>Variant</code>	<code>segment_intersects_segment_2d (Vector2 from_a, Vector2 to_a, Vector2 from_b, Vector2 to_b)</code>
<code>PoolVector3Array</code>	<code>segment_intersects_sphere (Vector3 from, Vector3 to, Vector3 sphere_position, float sphere_radius)</code>
<code>Variant</code>	<code>segment_intersects_triangle (Vector3 from, Vector3 to, Vector3 a, Vector3 b, Vector3 c)</code>
<code>PoolIntArray</code>	<code>triangulate_polygon (PoolVector2Array polygon)</code>

25.153.3 Member Function Description

- `Array build_box_planes (Vector3 extents)`

Returns an array with 6 *Planes* that describe the sides of a box centered at the origin. The box size is defined by `extents`, which represents one (positive) corner of the box (i.e. half its actual size).

- `Array build_capsule_planes (float radius, float height, int sides, int lats, int axis=2)`

Returns an array of *Planes* closely bounding a faceted capsule centered at the origin with radius `radius` and height `height`. The parameter `sides` defines how many planes will be generated for the side part of the capsule, whereas `lats` gives the number of latitudinal steps at the bottom and top of the capsule. The parameter `axis` describes the axis along which the capsule is oriented (0 for X, 1 for Y, 2 for Z).

- `Array build_cylinder_planes (float radius, float height, int sides, int axis=2)`

Returns an array of *Planes* closely bounding a faceted cylinder centered at the origin with radius `radius` and height `height`. The parameter `sides` defines how many planes will be generated for the round part of the cylinder. The parameter `axis` describes the axis along which the cylinder is oriented (0 for X, 1 for Y, 2 for Z).

- *PoolVector3Array* **clip_polygon** (*PoolVector3Array* points, *Plane* plane)

Clips the polygon defined by the points in `points` against the `plane` and returns the points of the clipped polygon.

- *PoolVector2Array* **convex_hull_2d** (*PoolVector2Array* points)

Given an array of `Vector2`s, returns the convex hull as a list of points in counter-clockwise order. The last point is the same as the first one.

- *Vector3* **get_closest_point_to_segment** (*Vector3* point, *Vector3* s1, *Vector3* s2)

Returns the 3d point on the 3d segment (`s1, s2`) that is closest to `point`. The returned point will always be inside the specified segment.

- *Vector2* **get_closest_point_to_segment_2d** (*Vector2* point, *Vector2* s1, *Vector2* s2)

Returns the 2d point on the 2d segment (`s1, s2`) that is closest to `point`. The returned point will always be inside the specified segment.

- *Vector3* **get_closest_point_to_segment_uncapped** (*Vector3* point, *Vector3* s1, *Vector3* s2)

Returns the 3d point on the 3d line defined by (`s1, s2`) that is closest to `point`. The returned point can be inside the segment (`s1, s2`) or outside of it, i.e. somewhere on the line extending from the segment.

- *Vector2* **get_closest_point_to_segment_uncapped_2d** (*Vector2* point, *Vector2* s1, *Vector2* s2)

Returns the 2d point on the 2d line defined by (`s1, s2`) that is closest to `point`. The returned point can be inside the segment (`s1, s2`) or outside of it, i.e. somewhere on the line extending from the segment.

- *PoolVector3Array* **get_closest_points_between_segments** (*Vector3* p1, *Vector3* p2, *Vector3* q1, *Vector3* q2)

Given the two 3d segments (`p1, p2`) and (`q1, q2`), finds those two points on the two segments that are closest to each other. Returns a `PoolVector3Array` that contains this point on (`p1, p2`) as well the accompanying point on (`q1, q2`).

- *PoolVector2Array* **get_closest_points_between_segments_2d** (*Vector2* p1, *Vector2* q1, *Vector2* p2, *Vector2* q2)

Given the two 2d segments (`p1, p2`) and (`q1, q2`), finds those two points on the two segments that are closest to each other. Returns a `PoolVector2Array` that contains this point on (`p1, p2`) as well the accompanying point on (`q1, q2`).

- *int* **get_uv84_normal_bit** (*Vector3* normal)

- *Dictionary* **make_atlas** (*PoolVector2Array* sizes)

Given an array of `Vector2`s representing tiles, builds an atlas. The returned dictionary has two keys: `points` is a vector of `Vector2` that specifies the positions of each tile, `size` contains the overall size of the whole atlas as `Vector2`.

- *bool* **point_is_inside_triangle** (*Vector2* point, *Vector2* a, *Vector2* b, *Vector2* c) const

Returns if `point` is inside the triangle specified by `a, b` and `c`.

- *Variant* **ray_intersects_triangle** (*Vector3* from, *Vector3* dir, *Vector3* a, *Vector3* b, *Vector3* c)

Tests if the 3d ray starting at `from` with the direction of `dir` intersects the triangle specified by `a, b` and `c`. If yes, returns the point of intersection as `Vector3`. If no intersection takes place, an empty `Variant` is returned.

- *float* **segment_intersects_circle** (*Vector2* segment_from, *Vector2* segment_to, *Vector2* circle_position, *float* circle_radius)

Given the 2d segment (`segment_from, segment_to`), returns the position on the segment (as a number between 0 and 1) at which the segment hits the circle that is located at position `circle_position` and has radius `circle_radius`. If the segment does not intersect the circle, -1 is returned (this is also the case if the line extending the segment would intersect the circle, but the segment does not).

- *PoolVector3Array* **segment_intersects_convex** (*Vector3* from, *Vector3* to, *Array* planes)

Given a convex hull defined through the *Planes* in the array `planes`, tests if the segment (`from`, `to`) intersects with that hull. If an intersection is found, returns a *PoolVector3Array* containing the point the intersection and the hull's normal. If no intersecion is found, an the returned array is empty.

- *PoolVector3Array* **segment_intersects_cylinder** (*Vector3* `from`, *Vector3* `to`, *float* `height`, *float* `radius`)

Checks if the segment (`from`, `to`) intersects the cylinder with height `height` that is centered at the origin and has radius `radius`. If no, returns an empty *PoolVector3Array*. If an intersection takes place, the returned array contains the point of intersection and the cylinder's normal at the point of intersection.

- *Variant* **segment_intersects_segment_2d** (*Vector2* `from_a`, *Vector2* `to_a`, *Vector2* `from_b`, *Vector2* `to_b`)

Checks if the two segments (`from_a`, `to_a`) and (`from_b`, `to_b`) intersect. If yes, return the point of intersection as *Vector2*. If no intersection takes place, returns an empty *Variant*.

- *PoolVector3Array* **segment_intersects_sphere** (*Vector3* `from`, *Vector3* `to`, *Vector3* `sphere_position`, *float* `sphere_radius`)

Checks if the segment (`from`, `to`) intersects the sphere that is located at `sphere_position` and has radius `sphere_radius`. If no, returns an empty *PoolVector3Array*. If yes, returns a *PoolVector3Array* containing the point of intersection and the sphere's normal at the point of intersection.

- *Variant* **segment_intersects_triangle** (*Vector3* `from`, *Vector3* `to`, *Vector3* `a`, *Vector3* `b`, *Vector3* `c`)

Tests if the segment (`from`, `to`) intersects the triangle `a`, `b`, `c`. If yes, returns the point of intersection as *Vector3*. If no intersection takes place, an empty *Variant* is returned.

- *PoolIntArray* **triangulate_polygon** (*PoolVector2Array* `polygon`)

Triangulates the polygon specified by the points in `polygon`. Returns a *PoolIntArray* where each triangle consists of three consecutive point indices into `polygon` (i.e. the returned array will have $n \times 3$ elements, with n being the number of found triangles). If the triangulation did not succeed, an empty *PoolIntArray* is returned.

25.154 GeometryInstance

Inherits: *VisualInstance* < *Spatial* < *Node* < *Object*

Inherited By: *MultiMeshInstance*, *MeshInstance*, *Particles*, *SpriteBase3D*, *ImmediateGeometry*

Category: Core

25.154.1 Brief Description

Base node for geometry based visual instances.

25.154.2 Member Variables

- *ShadowCastingSetting* **cast_shadow** - The selected shadow casting flag. See `SHADOW_CASTING_SETTING_*` constants for values.
- *float* **extra_cull_margin** - The extra distance added to the *GeometryInstance*'s bounding box (*AABB*) to increase its cull box.
- *float* **lod_max_distance** - The *GeometryInstance*'s max LOD distance.
- *float* **lod_max_hysteresis** - The *GeometryInstance*'s max LOD margin.

- *float lod_min_distance* - The GeometryInstance's min LOD distance.
- *float lod_min_hysteresis* - The GeometryInstance's min LOD margin.
- *Material material_override* - The material override for the whole geometry.

If there is a material in material_override, it will be used instead of any material set in any material slot of the mesh.

- *bool use_in_baked_light* - If true this GeometryInstance will be used when baking lights using a *GIProbe* and/or any other form of baked lighting.

25.154.3 Enums

enum Flags

- **FLAG_USE_BAKED_LIGHT = 0** — Will allow the GeometryInstance to be used when baking lights using a *GIProbe* and/or any other form of baked lighting.

Added documentation for GeometryInstance and VisualInstance - **FLAG_MAX = 1**

enum ShadowCastingSetting

- **SHADOW_CASTING_SETTING_OFF = 0** — Will not cast any shadows.
- **SHADOW_CASTING_SETTING_ON = 1** — Will cast shadows from all visible faces in the GeometryInstance.

Will take culling into account, so faces not being rendered will not be taken into account when shadow casting. - **SHADOW_CASTING_SETTING_DOUBLE_SIDED = 2** — Will cast shadows from all visible faces in the GeometryInstance.

Will not take culling into account, so all faces will be taken into account when shadow casting. - **SHADOW_CASTING_SETTING_SHADOWS_ONLY = 3** — Will only show the shadows casted from this object.

In other words: The actual mesh will not be visible, only the shadows casted from the mesh.

25.154.4 Description

Base node for geometry based visual instances. Shares some common functionality like visibility and custom materials.

25.155 GIProbe

Inherits: *VisualInstance < Spatial < Node < Object*

Category: Core

25.155.1 Brief Description

25.155.2 Member Functions

void	<i>bake (Node from_node=null, bool create_visual_debug=false)</i>
void	<i>debug_bake ()</i>

25.155.3 Member Variables

- *float bias*
- *bool compress*
- *GIProbeData data*
- *int dynamic_range*
- *float energy*
- *Vector3 extents*
- *bool interior*
- *float normal_bias*
- *float propagation*
- *Subdiv subdiv*

25.155.4 Enums

enum Subdiv

- **SUBDIV_64 = 0**
- **SUBDIV_128 = 1**
- **SUBDIV_256 = 2**
- **SUBDIV_512 = 3**
- **SUBDIV_MAX = 4**

25.155.5 Member Function Description

- void **bake (Node from_node=null, bool create_visual_debug=false)**
- void **debug_bake ()**

25.156 GIProbeData

Inherits: Resource < Reference < Object

Category: Core

25.156.1 Brief Description

25.156.2 Member Variables

- *float* **bias**
- **AABB bounds**
- *float* **cell_size**
- *bool* **compress**
- *PoolIntArray* **dynamic_data**
- *int* **dynamic_range**
- *float* **energy**
- *bool* **interior**
- *float* **normal_bias**
- *float* **propagation**
- *Transform* **to_cell_xform**

25.157 GodotSharp

Inherits: *Object*

Category: Core

25.157.1 Brief Description

25.157.2 Member Functions

<i>void</i>	<i>attach_thread ()</i>
<i>void</i>	<i>detach_thread ()</i>
<i>bool</i>	<i>is_domain_loaded ()</i>
<i>bool</i>	<i>is_finalizing_domain ()</i>

25.157.3 Member Function Description

- *void* **attach_thread ()**

Attaches the current thread to the mono runtime.

- *void* **detach_thread ()**

Detaches the current thread from the mono runtime.

- *bool* **is_domain_loaded ()**

Returns whether the scripts domain is loaded.

- *bool* **is_finalizing_domain ()**

Returns whether the scripts domain is being finalized.

25.158 Gradient

Inherits: *Resource < Reference < Object*

Category: Core

25.158.1 Brief Description

Color interpolator node.

25.158.2 Member Functions

void	<i>add_point (float offset, Color color)</i>
<i>Color</i>	<i>get_color (int point) const</i>
<i>float</i>	<i>get_offset (int point) const</i>
<i>int</i>	<i>get_point_count () const</i>
<i>Color</i>	<i>interpolate (float offset)</i>
void	<i>remove_point (int offset)</i>
void	<i>set_color (int point, Color color)</i>
void	<i>set_offset (int point, float offset)</i>

25.158.3 Member Variables

- *PoolColorArray colors* - Gradient's colors returned as a *PoolColorArray*.
- *PoolRealArray offsets* - Gradient's offsets returned as a *PoolRealArray*.

25.158.4 Description

Given a set of colors, this node will interpolate them in order, meaning, that if you have color 1, color 2 and color3, the ramp will interpolate (generate the colors between two colors) from color 1 to color 2 and from color 2 to color 3. Initially the ramp will have 2 colors (black and white), one (black) at ramp lower offset offset 0 and the other (white) at the ramp higher offset 1.

25.158.5 Member Function Description

- void **add_point** (*float offset, Color color*)

Adds the specified color to the end of the ramp, with the specified offset

- *Color get_color (int point) const*

Returns the color of the ramp color at index *point*

- *float get_offset (int point) const*

Returns the offset of the ramp color at index *point*

- `int get_point_count()` const

Returns the number of colors in the ramp

- `Color interpolate(float offset)`

Returns the interpolated color specified by *offset*

- `void remove_point(int offset)`

Removes the color at the index *offset*

- `void set_color(int point, Color color)`

Sets the color of the ramp color at index *point*

- `void set_offset(int point, float offset)`

Sets the offset for the ramp color at index *point*

25.159 GradientTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.159.1 Brief Description

Gradient filled texture.

25.159.2 Member Variables

- *Gradient gradient* - The *Gradient* that will be used to fill the texture.
- `int width` - The number of color samples that will be obtained from the *Gradient*.

25.159.3 Description

Uses a *Gradient* to fill the texture data, the gradient will be filled from left to right using colors obtained from the gradient, this means that the texture does not necessarily represent an exact copy of the gradient, but instead an interpolation of samples obtained from the gradient at fixed steps (see `set_width`).

25.160 GraphEdit

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.160.1 Brief Description

GraphEdit is an area capable of showing various GraphNodes. It manages connection events between them.

25.160.2 Member Functions

<i>int</i>	<i>connect_node</i> (<i>String</i> from, <i>int</i> from_port, <i>String</i> to, <i>int</i> to_port)
<i>void</i>	<i>disconnect_node</i> (<i>String</i> from, <i>int</i> from_port, <i>String</i> to, <i>int</i> to_port)
<i>Array</i>	<i>get_connection_list</i> () const
<i>bool</i>	<i>is_node_connected</i> (<i>String</i> from, <i>int</i> from_port, <i>String</i> to, <i>int</i> to_port)
<i>void</i>	<i>set_selected</i> (<i>Node</i> node)

25.160.3 Signals

- **_begin_node_move ()**

Signal sent at the beginning of a GraphNode movement.

- **_end_node_move ()**

Signal sent at the end of a GraphNode movement.

- **connection_request** (*String* from, *int* from_slot, *String* to, *int* to_slot)

Signal sent to the GraphEdit when the connection between ‘from_slot’ slot of ‘from’ GraphNode and ‘to_slot’ slot of ‘to’ GraphNode is attempted to be created.

- **connection_to_empty** (*String* from, *int* from_slot, *Vector2* release_position)

- **delete_nodes_request ()**

Signal sent when a GraphNode is attempted to be removed from the GraphEdit.

- **disconnection_request** (*String* from, *int* from_slot, *String* to, *int* to_slot)

Signal sent to the GraphEdit when the connection between ‘from_slot’ slot of ‘from’ GraphNode and ‘to_slot’ slot of ‘to’ GraphNode is attempted to be removed.

- **duplicate_nodes_request ()**

Signal sent when a GraphNode is attempted to be duplicated in the GraphEdit.

- **node_selected** (*Object* node)

- **popup_request** (*Vector2* p_position)

Signal sent when a popup is requested. Happens on right-clicking in the GraphEdit. ‘p_position’ is the position of the mouse pointer when the signal is sent.

- **scroll_offset_changed** (*Vector2* ofs)

25.160.4 Member Variables

- *bool* **right_disconnects** - If `true`, enables disconnection of existing connections in the GraphEdit by dragging the right end.
- *Vector2* **scroll_offset** - The scroll offset.
- *int* **snap_distance** - The snapping distance in pixels.
- *bool* **use_snap** - If `true`, enables snapping.
- *float* **zoom** - The current zoom value.

25.160.5 Description

GraphEdit manages the showing of GraphNodes it contains, as well as connections and disconnections between them. Signals are sent for each of these two events. Disconnection between GraphNodes slots is disabled by default.

It is greatly advised to enable low processor usage mode (see OS.set_low_processor_usage_mode) when using GraphEdits.

25.160.6 Member Function Description

- `int connect_node (String from, int from_port, String to, int to_port)`

Create a connection between ‘from_port’ slot of ‘from’ GraphNode and ‘to_port’ slot of ‘to’ GraphNode. If the connection already exists, no connection is created.

- `void disconnect_node (String from, int from_port, String to, int to_port)`

Remove the connection between ‘from_port’ slot of ‘from’ GraphNode and ‘to_port’ slot of ‘to’ GraphNode, if connection exists.

- `Array get_connection_list () const`

Return an Array containing the list of connections. A connection consists in a structure of the form {from_slot: 0, from: “GraphNode name 0”, to_slot: 1, to: “GraphNode name 1” }

- `bool is_node_connected (String from, int from_port, String to, int to_port)`

Return true if the ‘from_port’ slot of ‘from’ GraphNode is connected to the ‘to_port’ slot of ‘to’ GraphNode.

- `void set_selected (Node node)`

25.161 GraphNode

Inherits: Container < Control < CanvasItem < Node < Object

Category: Core

25.161.1 Brief Description

A GraphNode is a container with several input and output slots allowing connections between GraphNodes. Slots can have different, incompatible types.

25.161.2 Member Functions

void	<code>clear_all_slots()</code>
void	<code>clear_slot(int idx)</code>
<i>Color</i>	<code>get_connection_input_color(int idx)</code>
<i>int</i>	<code>get_connection_input_count()</code>
<i>Vector2</i>	<code>get_connection_input_position(int idx)</code>
<i>int</i>	<code>get_connection_input_type(int idx)</code>
<i>Color</i>	<code>get_connection_output_color(int idx)</code>
<i>int</i>	<code>get_connection_output_count()</code>
<i>Vector2</i>	<code>get_connection_output_position(int idx)</code>
<i>int</i>	<code>get_connection_output_type(int idx)</code>
<i>Color</i>	<code>get_slot_color_left(int idx) const</code>
<i>Color</i>	<code>get_slot_color_right(int idx) const</code>
<i>int</i>	<code>get_slot_type_left(int idx) const</code>
<i>int</i>	<code>get_slot_type_right(int idx) const</code>
<i>bool</i>	<code>is_slot_enabled_left(int idx) const</code>
<i>bool</i>	<code>is_slot_enabled_right(int idx) const</code>
void	<code>set_slot(int idx, bool enable_left, int type_left, Color color_left, bool enable_right, int type_right, Color color_right, Texture custom_left=null, Texture custom_right=null)</code>

25.161.3 Signals

- **close_request()**

Signal sent on closing the GraphNode.

- **dragged(Vector2 from, Vector2 to)**

Signal sent when the GraphNode is dragged.

- **offset_changed()**

Signal sent when the GraphNode is moved.

- **raise_request()**

Signal sent when the GraphNode is requested to be displayed over other ones. Happens on focusing (clicking into) the GraphNode.

- **resize_request(Vector2 new_minsize)**

25.161.4 Member Variables

- *bool comment*
- *Vector2 offset* - The offset of the GraphNode, relative to the scroll offset of the *GraphEdit*. Note that you cannot use position directly, as *GraphEdit* is a *Container*.
- *Overlay overlay*
- *bool resizable*

- *bool selected*
- *bool show_close*
- *String title*

25.161.5 Enums

enum Overlay

- **OVERLAY_DISABLED = 0**
- **OVERLAY_BREAKPOINT = 1**
- **OVERLAY_POSITION = 2**

25.161.6 Description

A GraphNode is a container defined by a title. It can have 1 or more input and output slots, which can be enabled (shown) or disabled (not shown) and have different (incompatible) types. Colors can also be assigned to slots. A tuple of input and output slots is defined for each GUI element included in the GraphNode. Input and output connections are left and right slots, but only enabled slots are counted as connections.

25.161.7 Member Function Description

- `void clear_all_slots()`

Disable all input and output slots of the GraphNode.

- `void clear_slot(int idx)`

Disable input and output slot whose index is ‘idx’.

- `Color get_connection_input_color(int idx)`

Return the color of the input connection ‘idx’.

- `int get_connection_input_count()`

Return the number of enabled input slots (connections) to the GraphNode.

- `Vector2 get_connection_input_position(int idx)`

Return the position of the input connection ‘idx’.

- `int get_connection_input_type(int idx)`

Return the type of the input connection ‘idx’.

- `Color get_connection_output_color(int idx)`

Return the color of the output connection ‘idx’.

- `int get_connection_output_count()`

Return the number of enabled output slots (connections) of the GraphNode.

- `Vector2 get_connection_output_position(int idx)`

Return the position of the output connection ‘idx’.

- `int get_connection_output_type (int idx)`

Return the type of the output connection ‘idx’.

- `Color get_slot_color_left (int idx) const`

Return the color set to ‘idx’ left (input) slot.

- `Color get_slot_color_right (int idx) const`

Return the color set to ‘idx’ right (output) slot.

- `int get_slot_type_left (int idx) const`

Return the (integer) type of left (input) ‘idx’ slot.

- `int get_slot_type_right (int idx) const`

Return the (integer) type of right (output) ‘idx’ slot.

- `bool is_slot_enabled_left (int idx) const`

Return true if left (input) slot ‘idx’ is enabled. False otherwise.

- `bool is_slot_enabled_right (int idx) const`

Return true if right (output) slot ‘idx’ is enabled. False otherwise.

- `void set_slot (int idx, bool enable_left, int type_left, Color color_left, bool enable_right, int type_right, Color color_right, Texture custom_left=null, Texture custom_right=null)`

25.162 GridContainer

Inherits: `Container < Control < CanvasItem < Node < Object`

Category: Core

25.162.1 Brief Description

Grid container used to arrange elements in a grid like layout

25.162.2 Member Variables

- `int columns` - The number of columns in the `GridContainer`. If modified, `GridContainer` reorders its children to accommodate the new layout.

25.162.3 Description

Grid container will arrange its children in a grid like structure, the grid columns are specified using the `set_columns` method and the number of rows will be equal to the number of children in the container divided by the number of columns, for example: if the container has 5 children, and 2 columns, there will be 3 rows in the container. Notice that grid layout will preserve the columns and rows for every size of the container.

25.163 GridMap

Inherits: *Spatial < Node < Object*

Category: Core

25.163.1 Brief Description

Node for 3D tile-based maps.

25.163.2 Member Functions

void	<i>clear ()</i>
void	<i>clear_baked_meshes ()</i>
<i>RID</i>	<i>get_bake_mesh_instance (int idx)</i>
<i>Array</i>	<i>get_bake_meshes ()</i>
<i>int</i>	<i>get_cell_item (int x, int y, int z) const</i>
<i>int</i>	<i>get_cell_item_orientation (int x, int y, int z) const</i>
<i>bool</i>	<i>get_collision_layer_bit (int bit) const</i>
<i>bool</i>	<i>get_collision_mask_bit (int bit) const</i>
<i>Array</i>	<i>get_meshes ()</i>
<i>Array</i>	<i>get_used_cells () const</i>
void	<i>make_baked_meshes (bool gen_lightmap_uv=false, float lightmap_uv_texel_size=0.1)</i>
<i>Vector3</i>	<i>map_to_world (int x, int y, int z) const</i>
void	<i>resource_changed (Resource resource)</i>
void	<i>set_cell_item (int x, int y, int z, int item, int orientation=0)</i>
void	<i>set_clip (bool enabled, bool clipabove=true, int floor=0, int axis=0)</i>
void	<i>set_collision_layer_bit (int bit, bool value)</i>
void	<i>set_collision_mask_bit (int bit, bool value)</i>
<i>Vector3</i>	<i>world_to_map (Vector3 pos) const</i>

25.163.3 Member Variables

- *bool cell_center_x* - If true grid items are centered on the X axis.
- *bool cell_center_y* - If true grid items are centered on the Y axis.
- *bool cell_center_z* - If true grid items are centered on the Z axis.
- *int cell_octant_size* - The size of each octant measured in number of cells. This applies to all three axis.
- *float cell_scale*
- *Vector3 cell_size* - The dimensions of the grid's cells.
- *int collision_layer*
- *int collision_mask*
- *MeshLibrary theme* - The assigned *MeshLibrary*.

25.163.4 Numeric Constants

- **INVALID_CELL_ITEM = -1** — Invalid cell item that can be used in `set_cell_item` to clear cells (or represent an empty cell in `get_cell_item`).

25.163.5 Description

GridMap lets you place meshes on a grid interactively. It works both from the editor and can help you create in-game level editors.

GridMaps use a *MeshLibrary* which contain a list of tiles: meshes with materials plus optional collisions and extra elements.

A GridMap contains a collection of cells. Each grid cell refers to a *MeshLibrary* item. All cells in the map have the same dimensions.

A GridMap is split into a sparse collection of octants for efficient rendering and physics processing. Every octant has the same dimensions and can contain several cells.

25.163.6 Member Function Description

- `void clear()`

Clear all cells.

- `void clear_baked_meshes()`
- `RID get_bake_mesh_instance(int idx)`
- `Array get_bake_meshes()`
- `int get_cell_item(int x, int y, int z) const`

The *MeshLibrary* item index located at the grid-based X, Y and Z coordinates. If the cell is empty, INVALID_CELL_ITEM will be returned.

- `int get_cell_item_orientation(int x, int y, int z) const`

The orientation of the cell at the grid-based X, Y and Z coordinates. -1 is retuned if the cell is empty.

- `bool get_collision_layer_bit(int bit) const`
- `bool get_collision_mask_bit(int bit) const`
- `Array get_meshes()`

Array of *Transform* and *Mesh* references corresponding to the non empty cells in the grid. The transforms are specified in world space.

- `Array get_used_cells() const`

Array of *Vector3* with the non empty cell coordinates in the grid map.

- `void make_baked_meshes(bool gen_lightmap_uv=false, float lightmap_uv_texel_size=0.1)`
- `Vector3 map_to_world(int x, int y, int z) const`
- `void resource_changed(Resource resource)`
- `void set_cell_item(int x, int y, int z, int item, int orientation=0)`

Set the mesh index for the cell referenced by its grid-based X, Y and Z coordinates.

A negative item index will clear the cell.

Optionally, the item's orientation can be passed.

- void `set_clip (bool enabled, bool clipabove=true, int floor=0, int axis=0)`
- void `set_collision_layer_bit (int bit, bool value)`
- void `set_collision_mask_bit (int bit, bool value)`
- `Vector3 world_to_map (Vector3 pos) const`

25.164 GrooveJoint2D

Inherits: *Joint2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.164.1 Brief Description

Groove constraint for 2D physics.

25.164.2 Member Variables

- `float initial_offset` - The body B's initial anchor position defined by the joint's origin and a local offset *initial_offset* along the joint's y axis (along the groove). Default value: 25
- `float length` - The groove's length. The groove is from the joint's origin towards *length* along the joint's local y axis. Default value: 50

25.164.3 Description

Groove constraint for 2D physics. This is useful for making a body “slide” through a segment placed in another.

25.165 HBoxContainer

Inherits: *BoxContainer < Container < Control < CanvasItem < Node < Object*

Category: Core

25.165.1 Brief Description

Horizontal box container.

25.165.2 Description

Horizontal box container. See *BoxContainer*.

25.166 HingeJoint

Inherits: *Joint < Spatial < Node < Object*

Category: Core

25.166.1 Brief Description

A hinge between two 3D bodies.

25.166.2 Member Variables

- *float angular_limit/bias* - The speed with which the rotation across the axis perpendicular to the hinge gets corrected.
- *bool angular_limit/enable* - If `true` the hinges maximum and minimum rotation, defined by *angular_limit/lower* and *angular_limit/upper* has effects.
- *float angular_limit/lower* - The minimum rotation. only active if *angular_limit/enable* is `true`.
- *float angular_limit/relaxation* - The lower this value, the more the rotation gets slowed down.
- *float angular_limit/softness*
- *float angular_limit/upper* - The maximum rotation. only active if *angular_limit/enable* is `true`.
- *bool motor/enable* - When activated, a motor turns the hinge.
- *float motor/max_impulse* - Maximum acceleration for the motor.
- *float motor/target_velocity* - Target speed for the motor.
- *float params/bias* - The speed with which the two bodies get pulled together when they move in different directions.

25.166.3 Enums

enum **Flag**

- **FLAG_USE_LIMIT = 0** — If `true` the hinges maximum and minimum rotation, defined by *angular_limit/lower* and *angular_limit/upper* has effects.
- **FLAG_ENABLE_MOTOR = 1** — When activated, a motor turns the hinge.
- **FLAG_MAX = 2** — End flag of **FLAG_*** constants, used internally.

enum **Param**

- **PARAM_BIAS = 0** — The speed with which the two bodies get pulled together when they move in different directions.
- **PARAM_LIMIT_UPPER = 1** — The maximum rotation. only active if *angular_limit/enable* is `true`.
- **PARAM_LIMIT_LOWER = 2** — The minimum rotation. only active if *angular_limit/enable* is `true`.
- **PARAM_LIMIT_BIAS = 3** — The speed with which the rotation across the axis perpendicular to the hinge gets corrected.

- **PARAM_LIMIT_SOFTNESS = 4**
- **PARAM_LIMIT_RELAXATION = 5** — The lower this value, the more the rotation gets slowed down.
- **PARAM_MOTOR_TARGET_VELOCITY = 6** — Target speed for the motor.
- **PARAM_MOTOR_MAX_IMPULSE = 7** — Maximum acceleration for the motor.
- **PARAM_MAX = 8** — End flag of PARAM_* constants, used internally.

25.166.4 Description

Normally uses the z-axis of body A as the hinge axis, another axis can be specified when adding it manually though.

25.167 HScrollBar

Inherits: *ScrollBar < Range < Control < CanvasItem < Node < Object*

Category: Core

25.167.1 Brief Description

Horizontal scroll bar.

25.167.2 Description

Horizontal scroll bar. See *ScrollBar*. This one goes from left (min) to right (max).

25.168 HSeparator

Inherits: *Separator < Control < CanvasItem < Node < Object*

Category: Core

25.168.1 Brief Description

Horizontal separator.

25.168.2 Description

Horizontal separator. See *Separator*. It is used to separate objects vertically, though (but it looks horizontal!).

25.169 HSlider

Inherits: *Slider < Range < Control < CanvasItem < Node < Object*

Category: Core

25.169.1 Brief Description

Horizontal slider.

25.169.2 Description

Horizontal slider. See *Slider*. This one goes from left (min) to right (max).

25.170 HSplitContainer

Inherits: *SplitContainer < Container < Control < CanvasItem < Node < Object*

Category: Core

25.170.1 Brief Description

Horizontal split container.

25.170.2 Description

Horizontal split container. See *SplitContainer*. This goes from left to right.

25.171 HTTPClient

Inherits: *Reference < Object*

Category: Core

25.171.1 Brief Description

Hyper-text transfer protocol client.

25.171.2 Member Functions

<code>void</code>	<code>close ()</code>
<code>int</code>	<code>connect_to_host (String host, int port=-1, bool use_ssl=false, bool verify_host=true)</code>
<code>int</code>	<code>get_response_body_length () const</code>
<code>int</code>	<code>get_response_code () const</code>
<code>PoolStringArray</code>	<code>get_response_headers ()</code>
<code>Dictionary</code>	<code>get_response_headers_as_dictionary ()</code>
<code>int</code>	<code>get_status () const</code>
<code>bool</code>	<code>has_response () const</code>
<code>bool</code>	<code>is_response_chunked () const</code>
<code>int</code>	<code>poll ()</code>
<code>String</code>	<code>query_string_from_dict (Dictionary fields)</code>
<code>PoolByteArray</code>	<code>read_response_body_chunk ()</code>
<code>int</code>	<code>request (int method, String url, PoolStringArray headers, String body="")</code>
<code>int</code>	<code>request_raw (int method, String url, PoolStringArray headers, PoolByteArray body)</code>
<code>void</code>	<code>set_read_chunk_size (int bytes)</code>

25.171.3 Member Variables

- `bool blocking_mode_enabled` - If `true`, execution will block until all data is read from the response.
- `StreamPeer connection` - The connection to use for this client.

25.171.4 Enums

enum Status

- **STATUS_DISCONNECTED = 0** — Status: Disconnected from the server.
- **STATUS_RESOLVING = 1** — Status: Currently resolving the hostname for the given URL into an IP.
- **STATUS_CANT_RESOLVE = 2** — Status: DNS failure: Can't resolve the hostname for the given URL.
- **STATUS_CONNECTING = 3** — Status: Currently connecting to server.
- **STATUS_CANT_CONNECT = 4** — Status: Can't connect to the server.
- **STATUS_CONNECTED = 5** — Status: Connection established.
- **STATUS_REQUESTING = 6** — Status: Currently sending request.
- **STATUS_BODY = 7** — Status: HTTP body received.
- **STATUS_CONNECTION_ERROR = 8** — Status: Error in HTTP connection.
- **STATUS_SSL_HANDSHAKE_ERROR = 9** — Status: Error in SSL handshake.

enum Method

- **METHOD_GET = 0** — HTTP GET method. The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

- **METHOD_HEAD = 1** — HTTP HEAD method. The HEAD method asks for a response identical to that of a GET request, but without the response body. This is useful to request metadata like HTTP headers or to check if a resource exists.
- **METHOD_POST = 2** — HTTP POST method. The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server. This is often used for forms and submitting data or uploading files.
- **METHOD_PUT = 3** — HTTP PUT method. The PUT method asks to replace all current representations of the target resource with the request payload. (You can think of POST as “create or update” and PUT as “update”, although many services tend to not make a clear distinction or change their meaning).
- **METHOD_DELETE = 4** — HTTP DELETE method. The DELETE method requests to delete the specified resource.
- **METHOD_OPTIONS = 5** — HTTP OPTIONS method. The OPTIONS method asks for a description of the communication options for the target resource. Rarely used.
- **METHOD_TRACE = 6** — HTTP TRACE method. The TRACE method performs a message loop-back test along the path to the target resource. Returns the entire HTTP request received in the response body. Rarely used.
- **METHOD_CONNECT = 7** — HTTP CONNECT method. The CONNECT method establishes a tunnel to the server identified by the target resource. Rarely used.
- **METHOD_PATCH = 8** — HTTP PATCH method. The PATCH method is used to apply partial modifications to a resource.
- **METHOD_MAX = 9** — Marker for end of METHOD__* enum. Not used.

enum **ResponseCode**

- **RESPONSE_CONTINUE = 100** — HTTP status code 100 Continue. Interim response that indicates everything so far is OK and that the client should continue with the request (or ignore this status if already finished).
- **RESPONSE_SWITCHING_PROTOCOLS = 101** — HTTP status code 101 Switching Protocol. Sent in response to an Upgrade request header by the client. Indicates the protocol the server is switching to.
- **RESPONSE_PROCESSING = 102** — HTTP status code 102 Processing (WebDAV). Indicates that the server has received and is processing the request, but no response is available yet.
- **RESPONSE_OK = 200** — HTTP status code 200 OK. The request has succeeded. Default response for successful requests. Meaning varies depending on the request. GET: The resource has been fetched and is transmitted in the message body. HEAD: The entity headers are in the message body. POST: The resource describing the result of the action is transmitted in the message body. TRACE: The message body contains the request message as received by the server.
- **RESPONSE_CREATED = 201** — HTTP status code 201 Created. The request has succeeded and a new resource has been created as a result of it. This is typically the response sent after a PUT request.
- **RESPONSE_ACCEPTED = 202** — HTTP status code 202 Accepted. The request has been received but not yet acted upon. It is non-committal, meaning that there is no way in HTTP to later send an asynchronous response indicating the outcome of processing the request. It is intended for cases where another process or server handles the request, or for batch processing.
- **RESPONSE_NON_AUTHORITATIVE_INFORMATION = 203** — HTTP status code 203 Non-Authoritative Information. This response code means returned meta-information set is not exact set as available from the origin server, but collected from a local or a third party copy. Except this condition, 200 OK response should be preferred instead of this response.

- **RESPONSE_NO_CONTENT = 204** — HTTP status code 204 No Content. There is no content to send for this request, but the headers may be useful. The user-agent may update its cached headers for this resource with the new ones.
- **RESPONSE_RESET_CONTENT = 205** — HTTP status code 205 Reset Content. The server has fulfilled the request and desires that the client resets the “document view” that caused the request to be sent to its original state as received from the origin server.
- **RESPONSE_PARTIAL_CONTENT = 206** — HTTP status code 206 Partial Content. This response code is used because of a range header sent by the client to separate download into multiple streams.
- **RESPONSE_MULTI_STATUS = 207** — HTTP status code 207 Multi-Status (WebDAV). A Multi-Status response conveys information about multiple resources in situations where multiple status codes might be appropriate.
- **RESPONSE_ALREADY_REPORTED = 208** — HTTP status code 208 Already Reported (WebDAV). Used inside a DAV: propstat response element to avoid enumerating the internal members of multiple bindings to the same collection repeatedly.
- **RESPONSE_IM_USED = 226** — HTTP status code 226 IM Used (WebDAV). The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.
- **RESPONSE_MULTIPLE_CHOICES = 300** — HTTP status code 300 Multiple Choice. The request has more than one possible responses and there is no standardized way to choose one of the responses. User-agent or user should choose one of them.
- **RESPONSE_MOVED_PERMANENTLY = 301** — HTTP status code 301 Moved Permanently. Redirection. This response code means the URI of requested resource has been changed. The new URI is usually included in the response.
- **RESPONSE_FOUND = 302** — HTTP status code 302 Found. Temporary redirection. This response code means the URI of requested resource has been changed temporarily. New changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.
- **RESPONSE_SEE_OTHER = 303** — HTTP status code 303 See Other. The server is redirecting the user agent to a different resource, as indicated by a URI in the Location header field, which is intended to provide an indirect response to the original request.
- **RESPONSE_NOT_MODIFIED = 304** — HTTP status code 304 Not Modified. A conditional GET or HEAD request has been received and would have resulted in a 200 OK response if it were not for the fact that the condition evaluated to false.
- **RESPONSE_USE_PROXY = 305** — HTTP status code 305 Use Proxy. Deprecated. Do not use.
- **RESPONSE_SWITCH_PROXY = 306** — HTTP status code 306 Switch Proxy. Deprecated. Do not use.
- **RESPONSE_TEMPORARY_REDIRECT = 307** — HTTP status code 307 Temporary Redirect. The target resource resides temporarily under a different URI and the user agent MUST NOT change the request method if it performs an automatic redirection to that URI.
- **RESPONSE_PERMANENT_REDIRECT = 308** — HTTP status code 308 Permanent Redirect. The target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed URIs.
- **RESPONSE_BAD_REQUEST = 400** — HTTP status code 400 Bad Request. The request was invalid. The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, invalid request contents, or deceptive request routing).

- **RESPONSE_UNAUTHORIZED = 401** — HTTP status code 401 Unauthorized. Credentials required. The request has not been applied because it lacks valid authentication credentials for the target resource.
- **RESPONSE_PAYMENT_REQUIRED = 402** — HTTP status code 402 Payment Required. This response code is reserved for future use. Initial aim for creating this code was using it for digital payment systems, however this is not currently used.
- **RESPONSE_FORBIDDEN = 403** — HTTP status code 403 Forbidden. The client does not have access rights to the content, i.e. they are unauthorized, so server is rejecting to give proper response. Unlike 401, the client's identity is known to the server.
- **RESPONSE_NOT_FOUND = 404** — HTTP status code 404 Not Found. The server can not find requested resource. Either the URL is not recognized or the endpoint is valid but the resource itself does not exist. May also be sent instead of 403 to hide existence of a resource if the client is not authorized.
- **RESPONSE_METHOD_NOT_ALLOWED = 405** — HTTP status code 405 Method Not Allowed. The request's HTTP method is known by the server but has been disabled and cannot be used. For example, an API may forbid DELETE-ing a resource. The two mandatory methods, GET and HEAD, must never be disabled and should not return this error code.
- **RESPONSE_NOT_ACCEPTABLE = 406** — HTTP status code 406 Not Acceptable. The target resource does not have a current representation that would be acceptable to the user agent, according to the proactive negotiation header fields received in the request. Used when negotiation content.
- **RESPONSE_PROXY_AUTHENTICATION_REQUIRED = 407** — HTTP status code 407 Proxy Authentication Required. Similar to 401 Unauthorized, but it indicates that the client needs to authenticate itself in order to use a proxy.
- **RESPONSE_REQUEST_TIMEOUT = 408** — HTTP status code 408 Request Timeout. The server did not receive a complete request message within the time that it was prepared to wait.
- **RESPONSE_CONFLICT = 409** — HTTP status code 409 Conflict. The request could not be completed due to a conflict with the current state of the target resource. This code is used in situations where the user might be able to resolve the conflict and resubmit the request.
- **RESPONSE_GONE = 410** — HTTP status code 410 Gone. The target resource is no longer available at the origin server and this condition is likely permanent.
- **RESPONSE_LENGTH_REQUIRED = 411** — HTTP status code 411 Length Required. The server refuses to accept the request without a defined Content-Length header.
- **RESPONSE_PRECONDITION_FAILED = 412** — HTTP status code 412 Precondition Failed. One or more conditions given in the request header fields evaluated to false when tested on the server.
- **RESPONSE_REQUEST_ENTITY_TOO_LARGE = 413** — HTTP status code 413 Entity Too Large. The server is refusing to process a request because the request payload is larger than the server is willing or able to process.
- **RESPONSE_REQUEST_URI_TOO_LONG = 414** — HTTP status code 414 Request-URI Too Long. The server is refusing to service the request because the request-target is longer than the server is willing to interpret.
- **RESPONSE_UNSUPPORTED_MEDIA_TYPE = 415** — HTTP status code 415 Unsupported Media Type. The origin server is refusing to service the request because the payload is in a format not supported by this method on the target resource.
- **RESPONSE_REQUESTED_RANGE_NOT_SATISFIABLE = 416** — HTTP status code 416 Requested Range Not Satisfiable. None of the ranges in the request's Range header field overlap the current extent of the selected resource or the set of ranges requested has been rejected due to invalid ranges or an excessive request of small or overlapping ranges.

- **RESPONSE_EXPECTATION_FAILED = 417** — HTTP status code 417 Expectation Failed. The expectation given in the request's Expect header field could not be met by at least one of the inbound servers.
- **RESPONSE_IM_A_TEAPOT = 418** — HTTP status code 418 I'm A Teapot. Any attempt to brew coffee with a teapot should result in the error code "418 I'm a teapot". The resulting entity body MAY be short and stout.
- **RESPONSE_MISDIRECTED_REQUEST = 421** — HTTP status code 421 Misdirected Request. The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.
- **RESPONSE_UNPROCESSABLE_ENTITY = 422** — HTTP status code 422 Unprocessable Entity (WebDAV). The server understands the content type of the request entity (hence a 415 Unsupported Media Type status code is inappropriate), and the syntax of the request entity is correct (thus a 400 Bad Request status code is inappropriate) but was unable to process the contained instructions.
- **RESPONSE_LOCKED = 423** — HTTP status code 423 Locked (WebDAV). The source or destination resource of a method is locked.
- **RESPONSE_FAILED_DEPENDENCY = 424** — HTTP status code 424 Failed Dependency (WebDAV). The method could not be performed on the resource because the requested action depended on another action and that action failed.
- **RESPONSE_UPGRADE_REQUIRED = 426** — HTTP status code 426 Upgrade Required. The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol.
- **RESPONSE_PRECONDITION_REQUIRED = 428** — HTTP status code 428 Precondition Required. The origin server requires the request to be conditional.
- **RESPONSE_TOO_MANY_REQUESTS = 429** — HTTP status code 429 Too Many Requests. The user has sent too many requests in a given amount of time (see "rate limiting"). Back off and increase time between requests or try again later.
- **RESPONSE_REQUEST_HEADER_FIELDS_TOO_LARGE = 431** — HTTP status code 431 Request Header Fields Too Large. The server is unwilling to process the request because its header fields are too large. The request MAY be resubmitted after reducing the size of the request header fields.
- **RESPONSE_UNAVAILABLE_FOR_LEGAL_REASONS = 451** — HTTP status code 451 Response Unavailable For Legal Reasons. The server is denying access to the resource as a consequence of a legal demand.
- **RESPONSE_INTERNAL_SERVER_ERROR = 500** — HTTP status code 500 Internal Server Error. The server encountered an unexpected condition that prevented it from fulfilling the request.
- **RESPONSE_NOT_IMPLEMENTED = 501** — HTTP status code 501 Not Implemented. The server does not support the functionality required to fulfill the request.
- **RESPONSE_BAD_GATEWAY = 502** — HTTP status code 502 Bad Gateway. The server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfill the request. Usually returned by load balancers or proxies.
- **RESPONSE_SERVICE_UNAVAILABLE = 503** — HTTP status code 503 Service Unavailable. The server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be alleviated after some delay. Try again later.
- **RESPONSE_GATEWAY_TIMEOUT = 504** — HTTP status code 504 Gateway Timeout. The server, while acting as a gateway or proxy, did not receive a timely response from an upstream server it needed to access in order to complete the request. Usually returned by load balancers or proxies.

- **RESPONSE_HTTP_VERSION_NOT_SUPPORTED = 505** — HTTP status code 505 HTTP Version Not Supported. The server does not support, or refuses to support, the major version of HTTP that was used in the request message.
- **RESPONSE_VARIANT_ALSO_NEGOTIATES = 506** — HTTP status code 506 Variant Also Negotiates. The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.
- **RESPONSE_INSUFFICIENT_STORAGE = 507** — HTTP status code 507 Insufficient Storage. The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.
- **RESPONSE_LOOP_DETECTED = 508** — HTTP status code 508 Loop Detected. The server terminated an operation because it encountered an infinite loop while processing a request with “Depth: infinity”. This status indicates that the entire operation failed.
- **RESPONSE_NOT_EXTENDED = 510** — HTTP status code 510 Not Extended. The policy for accessing the resource has not been met in the request. The server should send back all the information necessary for the client to issue an extended request.
- **RESPONSE_NETWORK_AUTH_REQUIRED = 511** — HTTP status code 511 Network Authentication Required. The client needs to authenticate to gain network access.

25.171.5 Description

Hyper-text transfer protocol client (sometimes called “User Agent”). Used to make HTTP requests to download web content, upload files and other data or to communicate with various services, among other use cases.

Note that this client only needs to connect to a host once (see `connect_to_host`) to send multiple requests. Because of this, methods that take URLs usually take just the part after the host instead of the full URL, as the client is already connected to a host. See `request` for a full example and to get started.

A `HTTPClient` should be reused between multiple requests or to connect to different hosts instead of creating one client per request. Supports SSL and SSL server certificate verification. HTTP status codes in the 2xx range indicate success, 3xx redirection (i.e. “try again, but over here”), 4xx something was wrong with the request, and 5xx something went wrong on the server’s side.

For more information on HTTP, see <https://developer.mozilla.org/en-US/docs/Web/HTTP> (or read RFC 2616 to get it straight from the source: <https://tools.ietf.org/html/rfc2616>).

25.171.6 Member Function Description

- `void close()`

Closes the current connection, allowing reuse of this `HTTPClient`.

- `int connect_to_host (String host, int port=-1, bool use_ssl=false, bool verify_host=true)`

Connect to a host. This needs to be done before any requests are sent.

The host should not have `http://` prepended but will strip the protocol identifier if provided.

If no port is specified (or `-1` is used), it is automatically set to 80 for HTTP and 443 for HTTPS (if `use_ssl` is enabled).

`verify_host` will check the SSL identity of the host if set to `true`.

- `int get_response_body_length () const`

Returns the response's body length.

- `int get_response_code () const`

Returns the response's HTTP status code.

- `PoolStringArray get_response_headers ()`

Returns the response headers.

- `Dictionary get_response_headers_as_dictionary ()`

Returns all response headers as dictionary where the case-sensitivity of the keys and values is kept like the server delivers it. A value is a simple String, this string can have more than one value where ";" is used as separator.

Structure: ("key": "value1; value2")

Example: (content-length:12), (Content-Type:application/json; charset=UTF-8)

- `int get_status () const`

Returns a STATUS_* enum constant. Need to call `poll` in order to get status updates.

- `bool has_response () const`

If true this `HTTPClient` has a response available.

- `bool is_response_chunked () const`

If true this `HTTPClient` has a response that is chunked.

- `int poll ()`

This needs to be called in order to have any request processed. Check results with `get_status`

- `String query_string_from_dict (Dictionary fields)`

Generates a GET/POST application/x-www-form-urlencoded style query string from a provided dictionary, e.g.:

```
var fields = {"username": "user", "password": "pass"}
String queryString = httpClient.query_string_from_dict(fields)
returns:= "username=user&password=pass"
```

- `PoolByteArray read_response_body_chunk ()`

Reads one chunk from the response.

- `int request (int method, String url, PoolStringArray headers, String body="")`

Sends a request to the connected host. The URL parameter is just the part after the host, so for `http://somehost.com/index.php`, it is `index.php`.

Headers are HTTP request headers. For available HTTP methods, see `METHOD_*`.

To create a POST request with query strings to push to the server, do:

```
var fields = {"username" : "user", "password" : "pass"}
var queryString = httpClient.query_string_from_dict(fields)
var headers = ["Content-Type: application/x-www-form-urlencoded", "Content-Length: " + str(queryString.length())]
var result = httpClient.request(httpClient.METHOD_POST, "index.php", headers, queryString)
```

- `int request_raw (int method, String url, PoolStringArray headers, PoolByteArray body)`

Sends a raw request to the connected host. The URL parameter is just the part after the host, so for `http://somehost.com/index.php`, it is `index.php`.

Headers are HTTP request headers. For available HTTP methods, see `METHOD__*`.

Sends the body data raw, as a byte array and does not encode it in any way.

- `void set_read_chunk_size (int bytes)`

Sets the size of the buffer used and maximum bytes to read per iteration. see `read_response_body_chunk`

25.172 HTTPRequest

Inherits: `Node < Object`

Category: Core

25.172.1 Brief Description

A node with the ability to send HTTP requests.

25.172.2 Member Functions

<code>void</code>	<code>cancel_request ()</code>
<code>int</code>	<code>get_body_size () const</code>
<code>int</code>	<code>get_downloaded_bytes () const</code>
<code>int</code>	<code>get_http_client_status () const</code>
<code>int</code>	<code>request (String url, PoolStringArray custom_headers=PoolStringArray(), bool ssl_validate_domain=true, int method=0, String request_data="")</code>

25.172.3 Signals

- `request_completed (int result, int response_code, PoolStringArray headers, PoolByteArray body)`

This signal is emitted upon request completion.

25.172.4 Member Variables

- `int body_size_limit` - Maximum allowed size for response bodies.
- `String download_file` - The file to download into. Will output any received file into it.
- `int max_redirects` - Maximum number of allowed redirects.
- `bool use_threads` - If `true` multithreading is used to improve performance.

25.172.5 Enums

enum **Result**

- **RESULT_SUCCESS = 0** — Request successful.
- **RESULT_CHUNKED_BODY_SIZE_MISMATCH = 1**
- **RESULT_CANT_CONNECT = 2** — Request failed while connecting.
- **RESULT_CANT_RESOLVE = 3** — Request failed while resolving.
- **RESULT_CONNECTION_ERROR = 4** — Request failed due to connection(read/write) error.
- **RESULT_SSL_HANDSHAKE_ERROR = 5** — Request failed on SSL handshake.
- **RESULT_NO_RESPONSE = 6** — Request does not have a response(yet).
- **RESULT_BODY_SIZE_LIMIT_EXCEEDED = 7** — Request exceeded its maximum size limit, see `set_body_size_limit`.
- **RESULT_REQUEST_FAILED = 8** — Request failed. (Unused)
- **RESULT_DOWNLOAD_FILE_CANT_OPEN = 9** — `HTTPRequest` couldn't open the download file.
- **RESULT_DOWNLOAD_FILE_WRITE_ERROR = 10** — `HTTPRequest` couldn't write to the download file.
- **RESULT_REDIRECT_LIMIT_REACHED = 11** — Request reached its maximum redirect limit, see `set_max_redirects`.

25.172.6 Description

A node with the ability to send HTTP requests. Uses *HTTPClient* internally.

Can be used to make HTTP requests, i.e. download or upload files or web content via HTTP.

25.172.7 Member Function Description

- void **cancel_request ()**

Cancels the current request.

- int **get_body_size () const**

Returns the response body length.

- int **get_downloaded_bytes () const**

Returns the amount of bytes this `HTTPRequest` downloaded.

- int **get_http_client_status () const**

Returns the current status of the underlying *HTTPClient*. See `STATUS_*` enum on *HTTPClient*.

- int **request (String url, PoolStringArray custom_headers=PoolStringArray(), bool ssl_validate_domain=true, int method=0, String request_data="")**

25.173 Image

Inherits: *Resource < Reference < Object*

Category: Core

25.173.1 Brief Description

Image datatype.

25.173.2 Member Functions

void	<i>blend_rect (Image src, Rect2 src_rect, Vector2 dst)</i>
void	<i>blend_rect_mask (Image src, Image mask, Rect2 src_rect, Vector2 dst)</i>
void	<i>blit_rect (Image src, Rect2 src_rect, Vector2 dst)</i>
void	<i>blit_rect_mask (Image src, Image mask, Rect2 src_rect, Vector2 dst)</i>
void	<i>clear_mipmaps ()</i>
<i>int</i>	<i>compress (int mode, int source, float lossy_quality)</i>
void	<i>convert (int format)</i>
void	<i>copy_from (Image src)</i>
void	<i>create (int width, int height, bool use_mipmaps, int format)</i>
void	<i>create_from_data (int width, int height, bool use_mipmaps, int format, PoolByteArray data)</i>
void	<i>crop (int width, int height)</i>
<i>int</i>	<i>decompress ()</i>
<i>int</i>	<i>detect_alpha () const</i>
void	<i>expand_x2_hq2x ()</i>
void	<i>fill (Color color)</i>
void	<i>fix_alpha_edges ()</i>
void	<i>flip_x ()</i>
void	<i>flip_y ()</i>
<i>int</i>	<i>generate_mipmaps ()</i>
<i>PoolByteArray</i>	<i>get_data () const</i>
<i>int</i>	<i>get_format () const</i>
<i>int</i>	<i>get_height () const</i>
<i>int</i>	<i>get_mipmap_offset (int mipmap) const</i>
<i>Color</i>	<i>get_pixel (int x, int y) const</i>
<i>Image</i>	<i>get_rect (Rect2 rect) const</i>
<i>Vector2</i>	<i>get_size () const</i>
<i>Rect2</i>	<i>get_used_rect () const</i>
<i>int</i>	<i>get_width () const</i>
<i>bool</i>	<i>has_mipmaps () const</i>
<i>bool</i>	<i>is_compressed () const</i>
<i>bool</i>	<i>is_empty () const</i>
<i>bool</i>	<i>is_invisible () const</i>
<i>int</i>	<i>load (String path)</i>
<i>int</i>	<i>load_jpg_from_buffer (PoolByteArray buffer)</i>
<i>int</i>	<i>load_png_from_buffer (PoolByteArray buffer)</i>
void	<i>lock ()</i>
void	<i>normalmap_to_xy ()</i>

Continued on next page

Table 25.10 – continued from previous page

void	<i>premultiply_alpha ()</i>
void	<i>resize (int width, int height, int interpolation=1)</i>
void	<i>resize_to_po2 (bool square=false)</i>
<i>int</i>	<i>save_png (String path) const</i>
void	<i>set_pixel (int x, int y, Color color)</i>
void	<i>shrink_x2 ()</i>
void	<i>srgb_to_linear ()</i>
void	<i>unlock ()</i>

25.173.3 Member Variables

- *Dictionary data* - Holds all of the image's color data in a given format. See `FORMAT__*` constants.

25.173.4 Enums

enum **CompressMode**

- **COMPRESS_S3TC = 0**
- **COMPRESS_PVRTC2 = 1**
- **COMPRESS_PVRTC4 = 2**
- **COMPRESS_ETC = 3**
- **COMPRESS_ETC2 = 4**

enum **Interpolation**

- **INTERPOLATE_NEAREST = 0**
- **INTERPOLATE_BILINEAR = 1**
- **INTERPOLATE_CUBIC = 2**

enum **AlphaMode**

- **ALPHA_NONE = 0**
- **ALPHA_BIT = 1**
- **ALPHA_BLEND = 2**

enum **CompressSource**

- **COMPRESS_SOURCE_GENERIC = 0**
- **COMPRESS_SOURCE_SRGB = 1**
- **COMPRESS_SOURCE_NORMAL = 2**

enum **Format**

- **FORMAT_L8 = 0**
- **FORMAT_LA8 = 1**
- **FORMAT_R8 = 2**

- **FORMAT_RG8** = 3
- **FORMAT_RGB8** = 4
- **FORMAT_RGBA8** = 5
- **FORMAT_RGBA4444** = 6
- **FORMAT_RGBA5551** = 7
- **FORMAT_RF** = 8
- **FORMAT_RGF** = 9
- **FORMAT_RGBF** = 10
- **FORMAT_RGBAF** = 11
- **FORMAT_RH** = 12
- **FORMAT_RGH** = 13
- **FORMAT_RGBH** = 14
- **FORMAT_RGBAH** = 15
- **FORMAT_RGBE9995** = 16
- **FORMAT_DXT1** = 17
- **FORMAT_DXT3** = 18
- **FORMAT_DXT5** = 19
- **FORMAT_RGTC_R** = 20
- **FORMAT_RGTC_RG** = 21
- **FORMAT_BPTC_RGBA** = 22
- **FORMAT_BPTC_RGBF** = 23
- **FORMAT_BPTC_RGBFU** = 24
- **FORMAT_PVRTC2** = 25
- **FORMAT_PVRTC2A** = 26
- **FORMAT_PVRTC4** = 27
- **FORMAT_PVRTC4A** = 28
- **FORMAT_ETC** = 29
- **FORMAT_ETC2_R11** = 30
- **FORMAT_ETC2_R11S** = 31
- **FORMAT_ETC2_RG11** = 32
- **FORMAT_ETC2_RG11S** = 33
- **FORMAT_ETC2_RGB8** = 34
- **FORMAT_ETC2_RGBA8** = 35
- **FORMAT_ETC2_RGB8A1** = 36
- **FORMAT_MAX** = 37

25.173.5 Description

Native image datatype. Contains image data, which can be converted to a *Texture*, and several functions to interact with it. The maximum width and height for an *Image* is 16384 pixels.

25.173.6 Member Function Description

- void **blend_rect** (*Image* src, *Rect2* src_rect, *Vector2* dst)

Alpha-blends src_rect from src image to this image at coordinates dst.

- void **blend_rect_mask** (*Image* src, *Image* mask, *Rect2* src_rect, *Vector2* dst)

Alpha-blends src_rect from src image to this image using mask image at coordinates dst. Alpha channels are required for both src and mask. dst pixels and src pixels will blend if the corresponding mask pixel's alpha value is not 0. src image and mask image **must** have the same size (width and height) but they can have different formats.

- void **blit_rect** (*Image* src, *Rect2* src_rect, *Vector2* dst)

Copies src_rect from src image to this image at coordinates dst.

- void **blit_rect_mask** (*Image* src, *Image* mask, *Rect2* src_rect, *Vector2* dst)

Blits src_rect area from src image to this image at the coordinates given by dst. src pixel is copied onto dst if the corresponding mask pixel's alpha value is not 0. src image and mask image **must** have the same size (width and height) but they can have different formats.

- void **clear_mipmaps** ()

Removes the image's mipmaps.

- int **compress** (int mode, int source, float lossy_quality)

Compresses the image to use less memory. Can not directly access pixel data while the image is compressed. Returns error if the chosen compression mode is not available. See COMPRESS_* constants.

- void **convert** (int format)

Converts the image's format. See FORMAT_* constants.

- void **copy_from** (*Image* src)

Copies src image to this image.

- void **create** (int width, int height, bool use_mipmaps, int format)

Creates an empty image of given size and format. See FORMAT_* constants. If use_mipmaps is true then generate mipmaps for this image. See the generate_mipmaps method.

- void **create_from_data** (int width, int height, bool use_mipmaps, int format, *PoolByteArray* data)

Creates a new image of given size and format. See FORMAT_* constants. Fills the image with the given raw data. If use_mipmaps is true then generate mipmaps for this image. See the generate_mipmaps method.

- void **crop** (int width, int height)

Crops the image to the given width and height. If the specified size is larger than the current size, the extra area is filled with black pixels.

- int **decompress** ()

Decompresses the image if it is compressed. Returns an error if decompress function is not available.

- int **detect_alpha** () const

Returns ALPHA_BLEND if the image has data for alpha values. Returns ALPHA_BIT if all the alpha values are below a certain threshold or the maximum value. Returns ALPHA_NONE if no data for alpha values is found.

- **void expand_x2_hq2x ()**

Stretches the image and enlarges it by a factor of 2. No interpolation is done.

- **void fill (Color color)**

Fills the image with a given *Color*.

- **void fix_alpha_edges ()**

Blends low-alpha pixels with nearby pixels.

- **void flip_x ()**

Flips the image horizontally.

- **void flip_y ()**

Flips the image vertically.

- **int generate_mipmaps ()**

Generates mipmaps for the image. Mipmaps are pre-calculated and lower resolution copies of the image. Mipmaps are automatically used if the image needs to be scaled down when rendered. This improves image quality and the performance of the rendering. Returns an error if the image is compressed, in a custom format or if the image's width/height is 0.

- **PoolByteArray get_data () const**

Returns the image's raw data.

- **int get_format () const**

Returns the image's raw data.

- **int get_height () const**

Returns the image's height.

- **int get_mipmap_offset (int mipmap) const**

Returns the offset where the image's mipmap with index `mipmap` is stored in the `data` dictionary.

- **Color get_pixel (int x, int y) const**

Returns the color of the pixel at `(x, y)` if the image is locked. If the image is unlocked it always returns a *Color* with the value `(0, 0, 0, 1.0)`.

- **Image get_rect (Rect2 rect) const**

Returns a new image that is a copy of the image's area specified with `rect`.

- **Vector2 get_size () const**

Returns the image's size (width and height).

- **Rect2 get_used_rect () const**

Returns a `Rect2` enclosing the visible portion of the image.

- **int get_width () const**

Returns the image's width.

- **bool has_mipmaps () const**

Returns `true` if the image has generated mipmaps.

- *bool is_compressed () const*

Returns `true` if the image is compressed.

- *bool is_empty () const*

Returns `true` if the image has no data.

- *bool is_invisible () const*

Returns `true` if all the image's pixels have an alpha value of 0. Returns `false` if any pixel has an alpha value higher than 0.

- *int load (String path)*

Loads an image from file `path`.

- *int load_jpg_from_buffer (PoolByteArray buffer)*

- *int load_png_from_buffer (PoolByteArray buffer)*

- *void lock ()*

Locks the data for writing access.

- *void normalmap_to_xy ()*

Converts the image's data to represent coordinates on a 3D plane. This is used when the image represents a normalmap. A normalmap can add lots of detail to a 3D surface without increasing the polygon count.

- *void premultiply_alpha ()*

Multiplies color values with alpha values. Resulting color values for a pixel are `(color * alpha) / 256`.

- *void resize (int width, int height, int interpolation=1)*

Resizes the image to the given `width` and `height`. New pixels are calculated using `interpolation`. See `interpolation` constants.

- *void resize_to_p02 (bool square=false)*

Resizes the image to the nearest power of 2 for the width and height. If `square` is `true` then set width and height to be the same.

- *int save_png (String path) const*

Saves the image as a PNG file to `path`.

- *void set_pixel (int x, int y, Color color)*

Sets the `Color` of the pixel at `(x, y)` if the image is locked. Example:

```
var img = Image.new()
img.create(img_width, img_height, false, Image.FORMAT_RGBA8)
img.lock()
img.set_pixel(x, y, color) # Works
img.unlock()
img.set_pixel(x, y, color) # Does not have an effect
```

- *void shrink_x2 ()*

Shrinks the image by a factor of 2.

- *void srgb_to_linear ()*

Converts the raw data from the sRGB colorspace to a linear scale.

- *void unlock ()*

Unlocks the data and prevents changes.

25.174 ImageTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.174.1 Brief Description

A *Texture* based on an *Image*.

25.174.2 Member Functions

void	<i>create</i> (<i>int width, int height, int format, int flags=7</i>)
void	<i>create_from_image</i> (<i>Image image, int flags=7</i>)
<i>int</i>	<i>get_format</i> () const
void	<i>load</i> (<i>String path</i>)
void	<i>set_data</i> (<i>Image image</i>)
void	<i>set_size_override</i> (<i>Vector2 size</i>)

25.174.3 Member Variables

- *float lossy_quality* - The storage quality for `ImageTexture.STORAGE_COMPRESS_LOSSY`.
- *Storage storage* - The storage type (raw, lossy, or compressed).

25.174.4 Enums

enum Storage

- **STORAGE_RAW = 0** — *Image* data is stored raw and unaltered.
- **STORAGE_COMPRESS_LOSSY = 1** — *Image* data is compressed with a lossy algorithm. You can set the storage quality with `set_lossy_storage_quality`.
- **STORAGE_COMPRESS_LOSSLESS = 2** — *Image* data is compressed with a lossless algorithm.

25.174.5 Description

A *Texture* based on an *Image*. Can be created from an *Image* with `create_from_image`.

25.174.6 Member Function Description

- void **create** (*int width, int height, int format, int flags=7*)

Create a new `ImageTexture` with “width” and “height”.

“format” one of `Image.FORMAT_*`.

“flags” one or more of `Texture.FLAG_*`.

- void **create_from_image** (*Image image, int flags=7*)

Create a new `ImageTexture` from an `Image` with “flags” from `Texture.FLAG_*`.

- *int get_format () const*

Return the format of the `ImageTexture`, one of `Image.FORMAT_*`.

- void **load** (*String path*)

Load an `ImageTexture`.

- void **set_data** (*Image image*)

Set the `Image` of this `ImageTexture`.

- void **set_size_override** (*Vector2 size*)

Resizes the `ImageTexture` to the specified dimensions.

25.175 ImmediateGeometry

Inherits: `GeometryInstance < VisualInstance < Spatial < Node < Object`

Category: Core

25.175.1 Brief Description

Draws simple geometry from code.

25.175.2 Member Functions

void	<code>add_sphere (int lats, int lons, float radius, bool add_uv=true)</code>
void	<code>add_vertex (Vector3 position)</code>
void	<code>begin (int primitive, Texture texture=null)</code>
void	<code>clear ()</code>
void	<code>end ()</code>
void	<code>set_color (Color color)</code>
void	<code>set_normal (Vector3 normal)</code>
void	<code>set_tangent (Plane tangent)</code>
void	<code>set_uv (Vector2 uv)</code>
void	<code>set_uv2 (Vector2 uv)</code>

25.175.3 Description

Draws simple geometry from code. Uses a drawing mode similar to OpenGL 1.x.

25.175.4 Member Function Description

- void **add_sphere** (*int* lats, *int* lons, *float* radius, *bool* add_uv=true)

Simple helper to draw a uvsphere, with given latitudes, longitude and radius.

- void **add_vertex** (*Vector3* position)

Adds a vertex with the currently set color/uv/etc.

- void **begin** (*int* primitive, *Texture* texture=null)

Begin drawing (And optionally pass a texture override). When done call end(). For more information on how this works, search for glBegin() glEnd() references.

For the type of primitive, use the *Mesh.PRIMITIVE_** enumerations.

- void **clear** ()

Clears everything that was drawn using begin/end.

- void **end** ()

Ends a drawing context and displays the results.

- void **set_color** (*Color* color)

The current drawing color.

- void **set_normal** (*Vector3* normal)

The next vertex's normal.

- void **set_tangent** (*Plane* tangent)

The next vertex's tangent (and binormal facing).

- void **set_uv** (*Vector2* uv)

The next vertex's UV.

- void **set_uv2** (*Vector2* uv)

The next vertex's second layer UV.

25.176 Input

Inherits: *Object*

Inherited By: *InputDefault*

Category: Core

25.176.1 Brief Description

A Singleton that deals with inputs.

25.176.2 Member Functions

void	<code>action_press (String action)</code>
void	<code>action_release (String action)</code>
void	<code>add_joy_mapping (String mapping, bool update_existing=false)</code>
<code>Vector3</code>	<code>get_accelerometer () const</code>
<code>Array</code>	<code>get_connected_joypads ()</code>
<code>Vector3</code>	<code>get_gravity () const</code>
<code>Vector3</code>	<code>get_gyroscope () const</code>
<code>float</code>	<code>get_joy_axis (int device, int axis) const</code>
<code>int</code>	<code>get_joy_axis_index_from_string (String axis)</code>
<code>String</code>	<code>get_joy_axis_string (int axis_index)</code>
<code>int</code>	<code>get_joy_button_index_from_string (String button)</code>
<code>String</code>	<code>get_joy_button_string (int button_index)</code>
<code>String</code>	<code>get_joy_guid (int device) const</code>
<code>String</code>	<code>get_joy_name (int device)</code>
<code>float</code>	<code>get_joy_vibration_duration (int device)</code>
<code>Vector2</code>	<code>get_joy_vibration_strength (int device)</code>
<code>Vector2</code>	<code>get_last_mouse_speed () const</code>
<code>Vector3</code>	<code>get_magnetometer () const</code>
<code>int</code>	<code>get_mouse_button_mask () const</code>
<code>int</code>	<code>get_mouse_mode () const</code>
<code>bool</code>	<code>is_action_just_pressed (String action) const</code>
<code>bool</code>	<code>is_action_just_released (String action) const</code>
<code>bool</code>	<code>is_action_pressed (String action) const</code>
<code>bool</code>	<code>is_joy_button_pressed (int device, int button) const</code>
<code>bool</code>	<code>is_joy_known (int device)</code>
<code>bool</code>	<code>is_key_pressed (int scancode) const</code>
<code>bool</code>	<code>is_mouse_button_pressed (int button) const</code>
<code>void</code>	<code>joy_connection_changed (int device, bool connected, String name, String guid)</code>
<code>void</code>	<code>parse_input_event (InputEvent event)</code>
<code>void</code>	<code>remove_joy_mapping (String guid)</code>
<code>void</code>	<code>set_custom_mouse_cursor (Resource image, int shape=0, Vector2 hotspot=Vector2(0, 0))</code>
<code>void</code>	<code>set_mouse_mode (int mode)</code>
<code>void</code>	<code>start_joy_vibration (int device, float weak_magnitude, float strong_magnitude, float duration=0)</code>
<code>void</code>	<code>stop_joy_vibration (int device)</code>
<code>void</code>	<code>warp_mouse_position (Vector2 to)</code>

25.176.3 Signals

- `joy_connection_changed (int index, bool connected)`

Emitted when a joypad device has been connected or disconnected

25.176.4 Enums

enum `MouseMode`

- `MOUSE_MODE_VISIBLE = 0` — Makes the mouse cursor visible if it is hidden.
- `MOUSE_MODE_HIDDEN = 1` — Makes the mouse cursor hidden if it is visible.

- **MOUSE_MODE_CAPTURED = 2** — Captures the mouse. The mouse will be hidden and unable to leave the game window. But it will still register movement and mouse button presses.
- **MOUSE_MODE_CONFINED = 3**

enum **CursorShape**

- **CURSOR_ARROW = 0**
- **CURSOR_IBEAM = 1**
- **CURSOR_POINTING_HAND = 2**
- **CURSOR_CROSS = 3**
- **CURSOR_WAIT = 4**
- **CURSOR_BUSY = 5**
- **CURSOR_DRAG = 6**
- **CURSOR_CAN_DROP = 7**
- **CURSOR_FORBIDDEN = 8**
- **CURSOR_VSIZE = 9**
- **CURSOR_HSIZE = 10**
- **CURSOR_BDIAGSIZE = 11**
- **CURSOR_FDIAGSIZE = 12**
- **CURSOR_MOVE = 13**
- **CURSOR_VSPLIT = 14**
- **CURSOR_HSPLIT = 15**
- **CURSOR_HELP = 16**

25.176.5 Description

A Singleton that deals with inputs. This includes key presses, mouse buttons and movement, joypads, and input actions. Actions and their events can be set in the Project Settings / Input Map tab. Or be set with *InputMap*.

25.176.6 Member Function Description

- void **action_press (String action)**

This will simulate pressing the specified action.

- void **action_release (String action)**

If the specified action is already pressed, this will release it.

- void **add_joy_mapping (String mapping, bool update_existing=false)**

Add a new mapping entry (in SDL2 format) to the mapping database. Optionally update already connected devices.

- **Vector3 get_accelerometer () const**

If the device has an accelerometer, this will return the movement.

- **Array get_connected_joypads ()**

Returns an *Array* containing the device IDs of all currently connected joypads.

- *Vector3* **get_gravity()** const
- *Vector3* **get_gyroscope()** const

If the device has a gyroscope, this will return the rate of rotation in rad/s around a device's x, y, and z axis.

- *float* **get_joy_axis(int device, int axis)** const

Returns the current value of the joypad axis at given index (see `JOY_*` constants in `@GlobalScope`)

- *int* **get_joy_axis_index_from_string(String axis)**
- *String* **get_joy_axis_string(int axis_index)**
- *int* **get_joy_button_index_from_string(String button)**
- *String* **get_joy_button_string(int button_index)**
- *String* **get_joy_guid(int device)** const

Returns a SDL2 compatible device guid on platforms that use gamepad remapping. Returns "Default Gamepad" otherwise.

- *String* **get_joy_name(int device)**

Returns the name of the joypad at the specified device index

- *float* **get_joy_vibration_duration(int device)**

Returns the duration of the current vibration effect in seconds.

- *Vector2* **get_joy_vibration_strength(int device)**

Returns the strength of the joypad vibration: x is the strength of the weak motor, and y is the strength of the strong motor.

- *Vector2* **get_last_mouse_speed()** const

Returns the mouse speed for the last time the cursor was moved, and this until the next frame where the mouse moves. This means that even if the mouse is not moving, this function will still return the value of the last motion.

- *Vector3* **get_magnetometer()** const

If the device has a magnetometer, this will return the magnetic field strength in micro-Tesla for all axes.

- *int* **get_mouse_button_mask()** const

Returns mouse buttons as a bitmask. If multiple mouse buttons are pressed at the same time the bits are added together.

- *int* **get_mouse_mode()** const

Return the mouse mode. See the constants for more information.

- *bool* **is_action_just_pressed(String action)** const

Returns true when you start pressing the action event.

- *bool* **is_action_just_released(String action)** const

Returns true when you stop pressing the action event.

- *bool* **is_action_pressed(String action)** const

Returns true if you are pressing the action event.

- *bool* **is_joy_button_pressed(int device, int button)** const

Returns true if you are pressing the joypad button. (see `JOY_*` constants in `@GlobalScope`)

- `bool is_joy_known (int device)`

Returns `true` if the system knows the specified device. This means that it sets all button and axis indices exactly as defined in the `JOY__*` constants (see `@GlobalScope`). Unknown joypads are not expected to match these constants, but you can still retrieve events from them.

- `bool is_key_pressed (int scancode) const`

Returns `true` if you are pressing the key. You can pass `KEY__*`, which are pre-defined constants listed in `@GlobalScope`.

- `bool is_mouse_button_pressed (int button) const`

Returns `true` if you are pressing the mouse button. You can pass `BUTTON__*`, which are pre-defined constants listed in `@GlobalScope`.

- `void joy_connection_changed (int device, bool connected, String name, String guid)`
- `void parse_input_event (InputEvent event)`
- `void remove_joy_mapping (String guid)`

Removes all mappings from the internal db that match the given uid.

- `void set_custom_mouse_cursor (Resource image, int shape=0, Vector2 hotspot=Vector2(0, 0))`

Set a custom mouse cursor image, which is only visible inside the game window. The hotspot can also be specified. See enum `CURSOR__*` for the list of shapes.

- `void set_mouse_mode (int mode)`

Set the mouse mode. See the constants for more information.

- `void start_joy_vibration (int device, float weak_magnitude, float strong_magnitude, float duration=0)`

Starts to vibrate the joypad. Joypads usually come with two rumble motors, a strong and a weak one. `weak_magnitude` is the strength of the weak motor (between 0 and 1) and `strong_magnitude` is the strength of the strong motor (between 0 and 1). `duration` is the duration of the effect in seconds (a duration of 0 will try to play the vibration indefinitely).

Note that not every hardware is compatible with long effect durations, it is recommended to restart an effect if in need to play it for more than a few seconds.

- `void stop_joy_vibration (int device)`

Stops the vibration of the joypad.

- `void warp_mouse_position (Vector2 to)`

Sets the mouse position to the specified vector.

25.177 InputDefault

Inherits: `Input < Object`

Category: Core

25.177.1 Brief Description

Default implementation of the `Input` class.

25.177.2 Description

Default implementation of the *Input* class, used internally by the editor and games for default input management.

25.178 InputEvent

Inherits: Resource < Reference < Object

Inherited By: InputEventScreenTouch, InputEventWithModifiers, InputEventScreenDrag, InputEventJoypadMotion, InputEventJoypadButton, InputEventAction

Category: Core

25.178.1 Brief Description

Generic input event

25.178.2 Member Functions

<code>bool</code>	<code>action_match (InputEvent event) const</code>
<code>String</code>	<code>as_text () const</code>
<code>bool</code>	<code>is_action (String action) const</code>
<code>bool</code>	<code>is_action_pressed (String action) const</code>
<code>bool</code>	<code>is_action_released (String action) const</code>
<code>bool</code>	<code>is_action_type () const</code>
<code>bool</code>	<code>is_echo () const</code>
<code>bool</code>	<code>is_pressed () const</code>
<code>bool</code>	<code>shortcut_match (InputEvent event) const</code>
<code>InputEvent</code>	<code>xformed_by (Transform2D xform, Vector2 local_ofs=Vector2(0, 0)) const</code>

25.178.3 Member Variables

- `int device` - The event's device ID.

25.178.4 Description

Base class of all sort of input event. See *Node._input*.

25.178.5 Member Function Description

- `bool action_match (InputEvent event) const`

Returns `true` if this event matches `event`.

- `String as_text () const`

Returns a `String` representation of the event.

- *bool is_action* (*String* action) const

Returns `true` if this input event matches a pre-defined action of any type.

- *bool is_action_pressed* (*String* action) const

Returns `true` if the given action is being pressed (and is not an echo event for KEY events). Not relevant for the event types MOUSE_MOTION, SCREEN_DRAG or NONE.

- *bool is_action_released* (*String* action) const

Returns `true` if the given action is released (i.e. not pressed). Not relevant for the event types MOUSE_MOTION, SCREEN_DRAG or NONE.

- *bool is_action_type* () const

Returns `true` if this input event's type is one of the `InputEvent` constants.

- *bool is_echo* () const

Returns `true` if this input event is an echo event (only for events of type KEY).

- *bool is_pressed* () const

Returns `true` if this input event is pressed. Not relevant for the event types MOUSE_MOTION, SCREEN_DRAG or NONE.

- *bool shortcut_match* (*InputEvent* event) const

- *InputEvent xformed_by* (*Transform2D* xform, *Vector2* local_ofs=*Vector2*(0, 0)) const

25.179 InputEventAction

Inherits: `InputEvent < Resource < Reference < Object`

Category: Core

25.179.1 Brief Description

Input event type for actions.

25.179.2 Member Variables

- *String action* - The action's name. Actions are accessed via this *String*.
- *bool pressed* - If `true` the action's state is pressed. If `false` the action's state is released.

25.179.3 Description

Contains a generic action which can be targeted from several type of inputs. Actions can be created from the project settings menu Project > Project Settings > Input Map. See `Node._input`.

25.180 InputEventGesture

Inherits: *InputEventWithModifiers < InputEvent < Resource < Reference < Object*

Inherited By: *InputEventPanGesture, InputEventMagnifyGesture*

Category: Core

25.180.1 Brief Description

25.180.2 Member Variables

- *Vector2 position*

25.181 InputEventJoypadButton

Inherits: *InputEvent < Resource < Reference < Object*

Category: Core

25.181.1 Brief Description

Input event for gamepad buttons.

25.181.2 Member Variables

- *int button_index* - Button identifier. One of the JOY_BUTTON__* constants from @global Scope.
- *bool pressed* - If true the button's state is pressed. If false the button's state is released.
- *float pressure* - Represents the pressure the user puts on the button with his finger, if the controller supports it. Ranges from 0 to 1.

25.181.3 Description

Input event type for gamepad buttons. For joysticks see *InputEventJoypadMotion*.

25.182 InputEventJoypadMotion

Inherits: *InputEvent < Resource < Reference < Object*

Category: Core

25.182.1 Brief Description

Input event type for gamepad joysticks and other motions. For buttons see *InputEventJoypadMotion*.

25.182.2 Member Variables

- *int axis* - Axis identifier. Use one of the JOY_AXIS_* constants in @global Scope.
- *float axis_value* - Current position of the joystick on the given axis. The value ranges from `-1.0` to `1.0`. A value of `0` means the axis is in its resting position.

25.182.3 Description

Stores information about joystick motions. One `InputEventJoypadMotion` represents one axis at a time.

25.183 InputEventKey

Inherits: `InputEventWithModifiers < InputEvent < Resource < Reference < Object`

Category: Core

25.183.1 Brief Description

Input event type for keyboard events.

25.183.2 Member Functions

<code>int</code>	<code>get_scancode_with_modifiers () const</code>
------------------	---

25.183.3 Member Variables

- *bool echo* - If `true` the key was already pressed before this event. It means the user is holding the key down.
- *bool pressed* - If `true` the key's state is pressed. If `false` the key's state is released.
- *int scancode* - Key scancode, one of the KEY_* constants in @global Scope.
- *int unicode* - Key unicode identifier when relevant.

25.183.4 Description

Stores key presses on the keyboard. Supports key presses, key releases and *echo* events.

25.183.5 Member Function Description

- `int get_scancode_with_modifiers () const`

25.184 InputEventMagnifyGesture

Inherits: *InputEventGesture < InputEventWithModifiers < InputEvent < Resource < Reference < Object*

Category: Core

25.184.1 Brief Description

25.184.2 Member Variables

- *float factor*

25.185 InputEventMouse

Inherits: *InputEventWithModifiers < InputEvent < Resource < Reference < Object*

Inherited By: *InputEventMouseMotion, InputEventMouseButton*

Category: Core

25.185.1 Brief Description

Base input event type for mouse events.

25.185.2 Member Variables

- *int button_mask* - Mouse button mask identifier, one of or a bitwise combination of the BUTTON_MASK_* constants in @GlobalScope.
- *Vector2 global_position* - Mouse position relative to the current *Viewport* when used in *Control._gui_input*, otherwise is at 0,0.
- *Vector2 position* - Mouse local position relative to the *Viewport*. If used in *Control._gui_input* the position is relative to the current *Control* which is under the mouse.

25.185.3 Description

Stores general mouse events information.

25.186 InputEventMouseButton

Inherits: *InputEventMouse < InputEventWithModifiers < InputEvent < Resource < Reference < Object*

Category: Core

25.186.1 Brief Description

Input event type for mouse button events.

25.186.2 Member Variables

- `int button_index` - Mouse button identifier, one of the `BUTTON_*` or `BUTTON_WHEEL_*` constants in `@GlobalScope`.
- `bool doubleclick` - If `true` the mouse button's state is a double-click. If `false` the mouse button's state is released.
- `float factor` - TO TALK in PR, reduz said : i think it's used for apple touch but i don't remember what it does
- `bool pressed` - If `true` the mouse button's state is pressed. If `false` the mouse button's state is released.

25.186.3 Description

Contains mouse click information. See `Node._input`.

25.187 InputEventMouseMotion

Inherits: `InputEventMouse < InputEventWithModifiers < InputEvent < Resource < Reference < Object`

Category: Core

25.187.1 Brief Description

Input event type for mouse motion events.

25.187.2 Member Variables

- `Vector2 relative` - Mouse position relative to the previous position (position at the last frame).
- `Vector2 speed` - Mouse speed.

25.187.3 Description

Contains mouse motion information. Supports relative, absolute positions and speed. See `Node._input`.

25.188 InputEventPanGesture

Inherits: `InputEventGesture < InputEventWithModifiers < InputEvent < Resource < Reference < Object`

Category: Core

25.188.1 Brief Description

25.188.2 Member Variables

- *Vector2* **delta**

25.189 InputEventScreenDrag

Inherits: *InputEvent < Resource < Reference < Object*

Category: Core

25.189.1 Brief Description

Input event type for screen drag events.

(only available on mobile devices)

25.189.2 Member Variables

- *int* **index** - Drag event index in the case of a multi-drag event.
- *Vector2* **position** - Drag position.
- *Vector2* **relative** - Drag position relative to its start position.
- *Vector2* **speed** - Drag speed.

25.189.3 Description

Contains screen drag information. See *Node._input*.

25.190 InputEventScreenTouch

Inherits: *InputEvent < Resource < Reference < Object*

Category: Core

25.190.1 Brief Description

Input event type for screen touch events.

(only available on mobile devices)

25.190.2 Member Variables

- *int index* - Touch index in the case of a multi-touch event. One index = one finger.
- *Vector2 position* - Touch position.
- *bool pressed* - If `true` the touch's state is pressed. If `false` the touch's state is released.

25.190.3 Description

Stores multi-touch press/release information. Supports touch press, touch release and *index* for multi-touch count and order.

25.191 InputEventWithModifiers

Inherits: *InputEvent < Resource < Reference < Object*

Inherited By: *InputEventKey, InputEventMouse, InputEventGesture*

Category: Core

25.191.1 Brief Description

Base class for keys events with modifiers.

25.191.2 Member Variables

- *bool alt* - State of the Alt modifier.
- *bool command* - State of the Command modifier.
- *bool control* - State of the Ctrl modifier.
- *bool meta* - State of the Meta modifier.
- *bool shift* - State of the Shift modifier.

25.191.3 Description

Contains keys events information with modifiers support like SHIFT or ALT. See `Node._input`.

25.192 InputMap

Inherits: *Object*

Category: Core

25.192.1 Brief Description

Singleton that manages *InputEventAction*.

25.192.2 Member Functions

<code>void</code>	<code>action_add_event (String action, InputEvent event)</code>
<code>void</code>	<code>action_erase_event (String action, InputEvent event)</code>
<code>bool</code>	<code>action_has_event (String action, InputEvent event)</code>
<code>void</code>	<code>add_action (String action)</code>
<code>void</code>	<code>erase_action (String action)</code>
<code>bool</code>	<code>event_is_action (InputEvent event, String action) const</code>
<code>Array</code>	<code>get_action_list (String action)</code>
<code>Array</code>	<code>get_actions ()</code>
<code>bool</code>	<code>has_action (String action) const</code>
<code>void</code>	<code>load_from_globals ()</code>

25.192.3 Description

Manages all *InputEventAction* which can be created/modified from the project settings menu Project > Project Settings > Input Map or in code with `add_action` and `action_add_event`. See `Node._input`.

25.192.4 Member Function Description

- `void action_add_event (String action, InputEvent event)`

Adds an *InputEvent* to an action. This *InputEvent* will trigger the action.

- `void action_erase_event (String action, InputEvent event)`

Removes an *InputEvent* from an action.

- `bool action_has_event (String action, InputEvent event)`

Returns true if an action has an *InputEvent* associated with it.

- `void add_action (String action)`

Adds an (empty) action to the `InputMap`. An *InputEvent* can then be added to this action with `action_add_event`.

- `void erase_action (String action)`

Removes an action from the `InputMap`.

- `bool event_is_action (InputEvent event, String action) const`

Returns true if the given event is part of an existing action. This method ignores keyboard modifiers if the given *InputEvent* is not pressed (for proper release detection). See `action_has_event` if you don't want this behavior.

- `Array get_action_list (String action)`

Returns an array of *InputEvents* associated with a given action.

- `Array get_actions ()`

Returns an array of all actions in the `InputMap`.

- `bool has_action (String action) const`

Returns `true` if the `InputMap` has a registered action with the given name.

- `void load_from_globals()`

Clears all `InputEventAction` in the `InputMap` and load it anew from `ProjectSettings`.

25.193 InstancePlaceholder

Inherits: `Node < Object`

Category: Core

25.193.1 Brief Description

Placeholder for the root `Node` of a `PackedScene`.

25.193.2 Member Functions

<code>String</code>	<code>get_instance_path()</code> const
<code>Dictionary</code>	<code>get_stored_values(bool with_order=false)</code>
<code>void</code>	<code>replace_by_instance(PackedScene custom_scene=null)</code>

25.193.3 Description

Turning on the option **Load As Placeholder** for an instanced scene in the editor causes it to be replaced by an `InstancePlaceholder` when running the game. This makes it possible to delay actually loading the scene until calling `replace_by_instance`. This is useful to avoid loading large scenes all at once by loading parts of it selectively.

The `InstancePlaceholder` does not have a transform. This causes any child nodes to be positioned relatively to the Viewport from point (0,0), rather than their parent as displayed in the editor. Replacing the placeholder with a scene with a transform will transform children relatively to their parent again.

25.193.4 Member Function Description

- `String get_instance_path()` const

Retrieve the path to the `PackedScene` resource file that is loaded by default when calling `replace_by_instance`.

- `Dictionary get_stored_values(bool with_order=false)`
- `void replace_by_instance(PackedScene custom_scene=null)`

Replace this placeholder by the scene handed as an argument, or the original scene if no argument is given. As for all resources, the scene is loaded only if it's not loaded already. By manually loading the scene beforehand, delays caused by this function can be avoided.

25.194 int

Category: Built-In Types

25.194.1 Brief Description

Integer built-in type.

25.194.2 Member Functions

<i>int</i>	<code>int (bool from)</code>
<i>int</i>	<code>int (float from)</code>
<i>int</i>	<code>int (String from)</code>

25.194.3 Description

Integer built-in type.

25.194.4 Member Function Description

- `int int (bool from)`

Cast a `bool` value to an integer value, `int (true)` will be equals to 1 and `int (false)` will be equals to 0.

- `int int (float from)`

Cast a `float` value to an integer value, this method simply removes the number fractions, so for example `int (2.7)` will be equals to 2, `int (.1)` will be equals to 0 and `int (-2.7)` will be equals to -2.

- `int int (String from)`

Cast a `String` value to an integer value, this method is an integer parser from a string, so calling this method with an invalid integer string will return 0, a valid string will be something like '`1.7`'. This method will ignore all non-number characters, so calling `int ('1e3')` will return 13.

25.195 InterpolatedCamera

Inherits: `Camera < Spatial < Node < Object`

Category: Core

25.195.1 Brief Description

25.195.2 Member Functions

<i>void</i>	<code>set_target (Object target)</code>
-------------	---

25.195.3 Member Variables

- `bool enabled`
- `float speed`

- *NodePath* target

25.195.4 Member Function Description

- void `set_target (Object target)`

25.196 IP

Inherits: *Object*

Inherited By: *IP_Unix*

Category: Core

25.196.1 Brief Description

Internet protocol (IP) support functions like DNS resolution.

25.196.2 Member Functions

void	<code>clear_cache (String hostname="")</code>
void	<code>erase_resolve_item (int id)</code>
Array	<code>get_local_addresses () const</code>
String	<code>get_resolve_item_address (int id) const</code>
int	<code>get_resolve_item_status (int id) const</code>
String	<code>resolve_hostname (String host, int ip_type=3)</code>
int	<code>resolve_hostname_queue_item (String host, int ip_type=3)</code>

25.196.3 Numeric Constants

- **RESOLVER_MAX_QUERIES = 32** — Maximum number of concurrent DNS resolver queries allowed, RESOLVER_INVALID_ID is returned if exceeded.
- **RESOLVER_INVALID_ID = -1** — Invalid ID constant. Returned if RESOLVER_MAX_QUERIES is exceeded.

25.196.4 Enums

enum **ResolverStatus**

- **RESOLVER_STATUS_NONE = 0** — DNS hostname resolver status: No status.
- **RESOLVER_STATUS_WAITING = 1** — DNS hostname resolver status: Waiting.
- **RESOLVER_STATUS_DONE = 2** — DNS hostname resolver status: Done.
- **RESOLVER_STATUS_ERROR = 3** — DNS hostname resolver status: Error.

enum **Type**

- **TYPE_NONE = 0** — Address type: None.
- **TYPE_IPV4 = 1** — Address type: Internet protocol version 4 (IPv4).
- **TYPE_IPV6 = 2** — Address type: Internet protocol version 6 (IPv6).
- **TYPE_ANY = 3** — Address type: Any.

25.196.5 Description

IP contains support functions for the Internet Protocol (IP). TCP/IP support is in different classes (see *StreamPeerTCP* and *TCP_Server*). IP provides DNS hostname resolution support, both blocking and threaded.

25.196.6 Member Function Description

- void **clear_cache** (*String* hostname="")

Removes all of a “hostname”‘s cached references. If no “hostname” is given then all cached IP addresses are removed.

- void **erase_resolve_item** (*int* id)

Removes a given item “id” from the queue. This should be used to free a queue after it has completed to enable more queries to happen.

- *Array* **get_local_addresses** () const

Returns all of the user’s current IPv4 and IPv6 addresses as an array.

- *String* **get_resolve_item_address** (*int* id) const

Returns a queued hostname’s IP address, given its queue “id”. Returns an empty string on error or if resolution hasn’t happened yet (see *get_resolve_item_status*).

- *int* **get_resolve_item_status** (*int* id) const

Returns a queued hostname’s status as a RESOLVER_STATUS_* constant, given its queue “id”.

- *String* **resolve_hostname** (*String* host, *int* ip_type=3)

Returns a given hostname’s IPv4 or IPv6 address when resolved (blocking-type method). The address type returned depends on the TYPE_* constant given as “ip_type”.

- *int* **resolve_hostname_queue_item** (*String* host, *int* ip_type=3)

Creates a queue item to resolve a hostname to an IPv4 or IPv6 address depending on the TYPE_* constant given as “ip_type”. Returns the queue ID if successful, or RESOLVER_INVALID_ID on error.

25.197 IP_Unix

Inherits: *IP* < *Object*

Category: Core

25.197.1 Brief Description

Unix IP support. See *IP*.

25.197.2 Description

Unix-specific implementation of IP support functions. See *IP*.

25.198 ItemList

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.198.1 Brief Description

Control that provides a list of selectable items (and/or icons) in a single column, or optionally in multiple columns.

25.198.2 Member Functions

void	<i>add_icon_item (Texture icon, bool selectable=true)</i>
void	<i>add_item (String text, Texture icon=null, bool selectable=true)</i>
void	<i>clear ()</i>
void	<i>ensure_current_is_visible ()</i>
int	<i>get_item_at_position (Vector2 position, bool exact=false) const</i>
int	<i>get_item_count () const</i>
Color	<i>get_item_custom_bg_color (int idx) const</i>
Texture	<i>get_item_icon (int idx) const</i>
Rect2	<i>get_item_icon_region (int idx) const</i>
Variant	<i>get_item_metadata (int idx) const</i>
String	<i>get_item_text (int idx) const</i>
String	<i>get_item_tooltip (int idx) const</i>
PoolIntArray	<i>get_selected_items ()</i>
VScrollBar	<i>get_v_scroll ()</i>
bool	<i>is_item_disabled (int idx) const</i>
bool	<i>is_item_selectable (int idx) const</i>
bool	<i>is_item_tooltip_enabled (int idx) const</i>
bool	<i>is_selected (int idx) const</i>
void	<i>remove_item (int idx)</i>
void	<i>select (int idx, bool single=true)</i>
void	<i>set_item_custom_bg_color (int idx, Color custom_bg_color)</i>
void	<i>set_item_disabled (int idx, bool disabled)</i>
void	<i>set_item_icon (int idx, Texture icon)</i>
void	<i>set_item_icon_region (int idx, Rect2 rect)</i>
void	<i>set_item_metadata (int idx, Variant metadata)</i>
void	<i>set_item_selectable (int idx, bool selectable)</i>
void	<i>set_item_text (int idx, String text)</i>
void	<i>set_item_tooltip (int idx, String tooltip)</i>
void	<i>set_item_tooltip_enabled (int idx, bool enable)</i>
void	<i>sort_items_by_text ()</i>
void	<i>unselect (int idx)</i>

25.198.3 Signals

- **item_activated** (*int index*)

Fired when specified list item is activated via double click or Enter.

- **item_rmb_selected** (*int index, Vector2 at_position*)

Fired when specified list item has been selected via right mouse clicking.

The click position is also provided to allow appropriate popup of context menus at the correct location.

- **item_selected** (*int index*)

Fired when specified item has been selected.

- **multi_selected** (*int index, bool selected*)

Fired when a multiple selection is altered on a list allowing multiple selection.

- **nothing_selected** ()

- **rmb_clicked** (*Vector2 at_position*)

25.198.4 Member Variables

- *bool* **allow_rmb_select**
- *bool* **auto_height**
- *int* **fixed_column_width**
- *Vector2* **fixed_icon_size**
- *IconMode* **icon_mode**
- *float* **icon_scale**
- *int* **max_columns**
- *int* **max_text_lines**
- *bool* **same_column_width**
- *SelectMode* **select_mode**

25.198.5 Enums

enum **IconMode**

- **ICON_MODE_TOP** = 0
- **ICON_MODE_LEFT** = 1

enum **SelectMode**

- **SELECT_SINGLE** = 0
- **SELECT_MULTI** = 1

25.198.6 Description

This control provides a selectable list of items that may be in a single (or multiple columns) with option of text, icons, or both text and icon. Tooltips are supported and may be different for every item in the list. Selectable items in the list may be selected or deselected and multiple selection may be enabled. Selection with right mouse button may also be enabled

to allow use of popup context menus. Items may also be ‘activated’ with a double click (or Enter key).

25.198.7 Member Function Description

- `void add_icon_item (Texture icon, bool selectable=true)`

Adds an item to the item list with no text, only an icon.

- `void add_item (String text, Texture icon=null, bool selectable=true)`

Adds an item to the item list with specified text. Specify an icon of null for a list item with no icon.

If selectable is true the list item will be selectable.

- `void clear ()`

Remove all items from the list.

- `void ensure_current_is_visible ()`

Ensure selection is visible, adjusting the scroll position as necessary.

- `int get_item_at_position (Vector2 position, bool exact=false) const`

Given a position within the control return the item (if any) at that point.

- `int get_item_count () const`

Return count of items currently in the item list.

- `Color get_item_custom_bg_color (int idx) const`

- `Texture get_item_icon (int idx) const`

- `Rect2 get_item_icon_region (int idx) const`

- `Variant get_item_metadata (int idx) const`

- `String get_item_text (int idx) const`

Return the text for specified item index.

- `String get_item_tooltip (int idx) const`

Return tooltip hint for specified item index.

- `PoolIntArray get_selected_items ()`

Returns the list of selected indexes.

- `VScrollBar get_v_scroll ()`

Returns the current vertical scroll bar for the List.

- `bool is_item_disabled (int idx) const`

Returns whether or not the item at the specified index is disabled

- `bool is_item_selectable (int idx) const`

Returns whether or not the item at the specified index is selectable.

- `bool is_item_tooltip_enabled (int idx) const`

Returns whether the tooltip is enabled for specified item index.

- `bool is_selected (int idx) const`

Returns whether or not item at the specified index is currently selected.

- `void remove_item (int idx)`

Remove item at specified index from the list.

- `void select (int idx, bool single=true)`

Select the item at the specified index.

Note: This method does not trigger the item selection signal.

- `void set_item_custom_bg_color (int idx, Color custom_bg_color)`
- `void set_item_disabled (int idx, bool disabled)`

Disable (or enable) item at specified index.

Disabled items are not be selectable and do not fire activation (Enter or double-click) signals.

- `void set_item_icon (int idx, Texture icon)`

Set (or replace) icon of the item at the specified index.

- `void set_item_icon_region (int idx, Rect2 rect)`
- `void set_item_metadata (int idx, Variant metadata)`

Sets a value (of any type) to be stored with the item at the specified index.

- `void set_item_selectable (int idx, bool selectable)`

Allow or disallow selection of the item at the specified index.

- `void set_item_text (int idx, String text)`

Sets text of item at specified index.

- `void set_item_tooltip (int idx, String tooltip)`

Sets tooltip hint for item at specified index.

- `void set_item_tooltip_enabled (int idx, bool enable)`

Sets whether the tooltip is enabled for specified item index.

- `void sort_items_by_text ()`

Sorts items in the list by their text.

- `void unselect (int idx)`

Ensure item at specified index is not selected.

25.199 JavaScript

Inherits: `Object`

Category: Core

25.199.1 Brief Description

Singleton that connects the engine with the browser's JavaScript context in HTML5 export.

25.199.2 Member Functions

<i>Variant</i>	<code>eval (String code, bool use_global_execution_context=false)</code>
----------------	--

25.199.3 Description

The JavaScript singleton is implemented only in HTML5 export. It's used to access the browser's JavaScript context. This allows interaction with embedding pages or calling third-party JavaScript APIs.

25.199.4 Member Function Description

- *Variant eval (String code, bool use_global_execution_context=false)*

Execute the string `code` as JavaScript code within the browser window. This is a call to the actual global JavaScript function `eval()`.

If `use_global_execution_context` is `true`, the code will be evaluated in the global execution context. Otherwise, it is evaluated in the execution context of a function within the engine's runtime environment.

25.200 Joint

Inherits: *Spatial < Node < Object*

Inherited By: *ConeTwistJoint, Generic6DOFJoint, SliderJoint, HingeJoint, PinJoint*

Category: Core

25.200.1 Brief Description

Base class for all 3D joints

25.200.2 Member Variables

- *bool collision/exclude_nodes* - If `true` the two bodies of the nodes are not able to collide with each other.
- *NodePath nodes/node_a* - The *Node*, the first side of the Joint attaches to.
- *NodePath nodes/node_b* - The *Node*, the second side of the Joint attaches to.
- *int solver/priority* - The order in which the solver is executed compared to the other Joints, the lower, the earlier.

25.200.3 Description

All 3D joints link two nodes, has a priority, and can decide if the two bodies of the nodes should be able to collide with each other

25.201 Joint2D

Inherits: *Node2D < CanvasItem < Node < Object*

Inherited By: *PinJoint2D, DampedSpringJoint2D, GrooveJoint2D*

Category: Core

25.201.1 Brief Description

Base node for all joint constraints in 2D physics.

25.201.2 Member Variables

- *float bias* - When *node_a* and *node_b* move in different directions the *bias* controls how fast the joint pulls them back to their original position. The lower the *bias* the more the two bodies can pull on the joint. Default value: 0
- *bool disable_collision* - If true *node_a* and *node_b* can collide. Default value: false.
- *NodePath node_a* - The first body attached to the joint. Must derive from *PhysicsBody2D*.
- *NodePath node_b* - The second body attached to the joint. Must derive from *PhysicsBody2D*.

25.201.3 Description

Base node for all joint constraints in 2D physics. Joints take 2 bodies and apply a custom constraint.

25.202 JSON

Inherits: *Object*

Category: Core

25.202.1 Brief Description

Helper class for parsing JSON data.

25.202.2 Member Functions

<i>JSONParseResult</i>	<i>parse (String json)</i>
<i>String</i>	<i>print (Variant value, String indent=""", bool sort_keys=false)</i>

25.202.3 Description

Helper class for parsing JSON data. For usage example, see *JSONParseResult*.

25.202.4 Member Function Description

- *JSONParseResult parse (String json)*

Parses a JSON encoded string and returns a *JSONParseResult* containing the result.

- *String print (Variant value, String indent=""", bool sort_keys=false)*

Converts a Variant var to JSON text and returns the result. Useful for serializing data to store or send over the network.

25.203 JSONParseResult

Inherits: Reference < Object

Category: Core

25.203.1 Brief Description

Data class wrapper for decoded JSON.

25.203.2 Member Variables

- Error **error** - The error type if JSON source was not successfully parsed. See *@GlobalScope* ERR_* constants.
- int **error_line** - The line number where the error occurred if JSON source was not successfully parsed.
- String **error_string** - The error message if JSON source was not successfully parsed. See *@GlobalScope* ERR_* constants.
- Variant **result** - A Variant containing the parsed JSON. Use `typeof()` to check if it is what you expect. For example, if JSON source starts with curly braces ({}) a *Dictionary* will be returned, if JSON source starts with braces ([]) an *Array* will be returned.

Be aware that the JSON specification does not define integer or float types, but only a number type. Therefore, parsing a JSON text will convert all numerical values to float types.

```
p = JSON.parse('["hello", "world", "!"]')
if typeof(p) == TYPE_ARRAY:
    print(p[0]) # prints 'hello'
else:
    print("unexpected results")
```

25.203.3 Description

Returned by `JSON.parse`, *JSONParseResult* contains decoded JSON or error information if JSON source not successfully parsed. You can check if JSON source was successfully parsed with `if json_result.error == 0`.

25.204 KinematicBody

Inherits: *PhysicsBody < CollisionObject < Spatial < Node < Object*

Category: Core

25.204.1 Brief Description

Kinematic body 3D node.

25.204.2 Member Functions

<i>Vector3</i>	<code>get_floor_velocity () const</code>
<i>Kinematic-Collision</i>	<code>get_slide_collision (int slide_idx)</code>
<i>int</i>	<code>get_slide_count () const</code>
<i>bool</i>	<code>is_on_ceiling () const</code>
<i>bool</i>	<code>is_on_floor () const</code>
<i>bool</i>	<code>is_on_wall () const</code>
<i>Kinematic-Collision</i>	<code>move_and_collide (Vector3 rel_vec)</code>
<i>Vector3</i>	<code>move_and_slide (Vector3 linear_velocity, Vector3 floor_normal=Vector3(0, 0, 0), float slope_stop_min_velocity=0.05, int max_slides=4, float floor_max_angle=0.785398)</code>
<i>bool</i>	<code>test_move (Transform from, Vector3 rel_vec)</code>

25.204.3 Member Variables

- *bool axis_lock_angular_x*
- *bool axis_lock_angular_y*
- *bool axis_lock_angular_z*
- *bool axis_lock_linear_x*
- *bool axis_lock_linear_y*
- *bool axis_lock_linear_z*
- *float collision/safe_margin* - If the body is at least this close to another body, this body will consider them to be colliding.

25.204.4 Description

Kinematic bodies are special types of bodies that are meant to be user-controlled. They are not affected by physics at all (to other types of bodies, such a character or a rigid body, these are the same as a static body). They have however, two main uses:

Simulated Motion: When these bodies are moved manually, either from code or from an AnimationPlayer (with process mode set to fixed), the physics will automatically compute an estimate of their linear and angular velocity.

This makes them very useful for moving platforms or other AnimationPlayer-controlled objects (like a door, a bridge that opens, etc).

Kinematic Characters: KinematicBody also has an API for moving objects (the `move_and_collide` and `move_and_slide` methods) while performing collision tests. This makes them really useful to implement characters that collide against a world, but that don't require advanced physics.

25.204.5 Member Function Description

- `Vector3 get_floor_velocity () const`

Returns the velocity of the floor. Only updates when calling `move_and_slide`.

- `KinematicCollision get_slide_collision (int slide_idx)`

Returns a `KinematicCollision`, which contains information about a collision that occurred during the last `move_and_slide` call. Since the body can collide several times in a single call to `move_and_slide`, you must specify the index of the collision in the range 0 to (`get_slide_count` - 1).

- `int get_slide_count () const`

Returns the number of times the body collided and changed direction during the last call to `move_and_slide`.

- `bool is_on_ceiling () const`

Returns `true` if the body is on the ceiling. Only updates when calling `move_and_slide`.

- `bool is_on_floor () const`

Returns `true` if the body is on the floor. Only updates when calling `move_and_slide`.

- `bool is_on_wall () const`

Returns `true` if the body is on a wall. Only updates when calling `move_and_slide`.

- `KinematicCollision move_and_collide (Vector3 rel_vec)`

Moves the body along the vector `rel_vec`. The body will stop if it collides. Returns a `KinematicCollision`, which contains information about the collision.

- `Vector3 move_and_slide (Vector3 linear_velocity, Vector3 floor_normal=Vector3(0, 0, 0), float slope_stop_min_velocity=0.05, int max_slides=4, float floor_max_angle=0.785398)`

Moves the body along a vector. If the body collides with another, it will slide along the other body rather than stop immediately. If the other body is a `KinematicBody` or `RigidBody`, it will also be affected by the motion of the other body. You can use this to make moving or rotating platforms, or to make nodes push other nodes.

`linear_velocity` is a value in pixels per second. Unlike in for example `move_and_collide`, you should *not* multiply it with `delta` — this is done by the method.

`floor_normal` is the up direction, used to determine what is a wall and what is a floor or a ceiling. If set to the default value of `Vector3(0, 0, 0)`, everything is considered a wall. This is useful for topdown games.

If the body is standing on a slope and the horizontal speed (relative to the floor's speed) goes below `slope_stop_min_velocity`, the body will stop completely. This prevents the body from sliding down slopes when you include gravity in `linear_velocity`. When set to lower values, the body will not be able to stand still on steep slopes.

If the body collides, it will change direction a maximum of `max_bounces` times before it stops.

`floor_max_angle` is the maximum angle (in radians) where a slope is still considered a floor (or a ceiling), rather than a wall. The default value equals 45 degrees.

Returns the movement that remained when the body stopped. To get more detailed information about collisions that occurred, use `get_slide_collision`.

- `bool test_move (Transform from, Vector3 rel_vec)`

Checks for collisions without moving the body. Virtually sets the node's position, scale and rotation to that of the given `Transform`, then tries to move the body along the vector `rel_vec`. Returns `true` if a collision would occur.

25.205 KinematicBody2D

Inherits: `PhysicsBody2D < CollisionObject2D < Node2D < CanvasItem < Node < Object`

Category: Core

25.205.1 Brief Description

Kinematic body 2D node.

25.205.2 Member Functions

<code>Vector2</code>	<code>get_floor_velocity () const</code>
<code>Kinematic-Collision2D</code>	<code>get_slide_collision (int slide_idx)</code>
<code>int</code>	<code>get_slide_count () const</code>
<code>bool</code>	<code>is_on_ceiling () const</code>
<code>bool</code>	<code>is_on_floor () const</code>
<code>bool</code>	<code>is_on_wall () const</code>
<code>Kinematic-Collision2D</code>	<code>move_and_collide (Vector2 rel_vec)</code>
<code>Vector2</code>	<code>move_and_slide (Vector2 linear_velocity, Vector2 floor_normal=Vector2(0, 0), float slope_stop_min_velocity=5, int max_bounces=4, float floor_max_angle=0.785398)</code>
<code>bool</code>	<code>test_move (Transform2D from, Vector2 rel_vec)</code>

25.205.3 Member Variables

- `float collision/safe_margin` - If the body is at least this close to another body, this body will consider them to be colliding.

25.205.4 Description

Kinematic bodies are special types of bodies that are meant to be user-controlled. They are not affected by physics at all (to other types of bodies, such a character or a rigid body, these are the same as a static body). They have however, two main uses:

Simulated Motion: When these bodies are moved manually, either from code or from an `AnimationPlayer` (with process mode set to fixed), the physics will automatically compute an estimate of their linear and angular velocity. This makes them very useful for moving platforms or other `AnimationPlayer`-controlled objects (like a door, a bridge that opens, etc).

Kinematic Characters: KinematicBody2D also has an API for moving objects (the *move_and_collide* and *move_and_slide* methods) while performing collision tests. This makes them really useful to implement characters that collide against a world, but that don't require advanced physics.

25.205.5 Member Function Description

- `Vector2 get_floor_velocity () const`

Returns the velocity of the floor. Only updates when calling *move_and_slide*.

- `KinematicCollision2D get_slide_collision (int slide_idx)`

Returns a *KinematicCollision2D*, which contains information about a collision that occurred during the last *move_and_slide* call. Since the body can collide several times in a single call to *move_and_slide*, you must specify the index of the collision in the range 0 to (*get_slide_count* - 1).

- `int get_slide_count () const`

Returns the number of times the body collided and changed direction during the last call to *move_and_slide*.

- `bool is_on_ceiling () const`

Returns true if the body is on the ceiling. Only updates when calling *move_and_slide*.

- `bool is_on_floor () const`

Returns true if the body is on the floor. Only updates when calling *move_and_slide*.

- `bool is_on_wall () const`

Returns true if the body is on a wall. Only updates when calling *move_and_slide*.

- `KinematicCollision2D move_and_collide (Vector2 rel_vec)`

Moves the body along the vector *rel_vec*. The body will stop if it collides. Returns a *KinematicCollision2D*, which contains information about the collision.

- `Vector2 move_and_slide (Vector2 linear_velocity, Vector2 floor_normal=Vector2(0, 0), float slope_stop_min_velocity=5, int max_bounces=4, float floor_max_angle=0.785398)`

Moves the body along a vector. If the body collides with another, it will slide along the other body rather than stop immediately. If the other body is a *KinematicBody2D* or *RigidBody2D*, it will also be affected by the motion of the other body. You can use this to make moving or rotating platforms, or to make nodes push other nodes.

linear_velocity is a value in pixels per second. Unlike in for example *move_and_collide*, you should *not* multiply it with *delta* — this is done by the method.

floor_normal is the up direction, used to determine what is a wall and what is a floor or a ceiling. If set to the default value of `Vector2(0, 0)`, everything is considered a wall. This is useful for topdown games.

If the body is standing on a slope and the horizontal speed (relative to the floor's speed) goes below *slope_stop_min_velocity*, the body will stop completely. This prevents the body from sliding down slopes when you include gravity in *linear_velocity*. When set to lower values, the body will not be able to stand still on steep slopes.

If the body collides, it will change direction a maximum of *max_bounces* times before it stops.

floor_max_angle is the maximum angle (in radians) where a slope is still considered a floor (or a ceiling), rather than a wall. The default value equals 45 degrees.

Returns the movement that remained when the body stopped. To get more detailed information about collisions that occurred, use *get_slide_collision*.

- `bool test_move (Transform2D from, Vector2 rel_vec)`

Checks for collisions without moving the body. Virtually sets the node's position, scale and rotation to that of the given `Transform2D`, then tries to move the body along the vector `rel_vec`. Returns `true` if a collision would occur.

25.206 KinematicCollision

Inherits: `Reference < Object`

Category: Core

25.206.1 Brief Description

Collision data for KinematicBody collisions.

25.206.2 Member Variables

- *Object collider* - The colliding body.
- *int collider_id* - The colliding body's unique *RID*.
- *Variant collider_metadata* - The colliding body's metadata. See *Object*.
- *Object collider_shape* - The colliding body's shape.
- *int collider_shape_index* - The colliding shape's index. See *CollisionObject*.
- *Vector3 collider_velocity* - The colliding object's velocity.
- *Object local_shape* - The moving object's colliding shape.
- *Vector3 normal* - The colliding body's shape's normal at the point of collision.
- *Vector3 position* - The point of collision.
- *Vector3 remainder* - The moving object's remaining movement vector.
- *Vector3 travel* - The distance the moving object traveled before collision.

25.206.3 Description

Contains collision data for KinematicBody collisions. When a *KinematicBody* is moved using *KinematicBody.move_and_collide*, it stops if it detects a collision with another body. If a collision is detected, a KinematicCollision object is returned.

This object contains information about the collision, including the colliding object, the remaining motion, and the collision position. This information can be used to calculate a collision response.

25.207 KinematicCollision2D

Inherits: `Reference < Object`

Category: Core

25.207.1 Brief Description

Collision data for KinematicBody2D collisions.

25.207.2 Member Variables

- *Object collider* - The colliding body.
- *int collider_id* - The colliding body's unique *RID*.
- *Variant collider_metadata* - The colliding body's metadata. See *Object*.
- *Object collider_shape* - The colliding body's shape.
- *int collider_shape_index* - The colliding shape's index. See *CollisionObject2D*.
- *Vector2 collider_velocity* - The colliding object's velocity.
- *Object local_shape* - The moving object's colliding shape.
- *Vector2 normal* - The colliding body's shape's normal at the point of collision.
- *Vector2 position* - The point of collision.
- *Vector2 remainder* - The moving object's remaining movement vector.
- *Vector2 travel* - The distance the moving object traveled before collision.

25.207.3 Description

Contains collision data for KinematicBody2D collisions. When a *KinematicBody2D* is moved using *KinematicBody2D.move_and_collide*, it stops if it detects a collision with another body. If a collision is detected, a *KinematicCollision2D* object is returned.

This object contains information about the collision, including the colliding object, the remaining motion, and the collision position. This information can be used to calculate a collision response.

25.208 Label

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.208.1 Brief Description

Displays plain text in a line or wrapped inside a rectangle. For formatted text, use *RichTextLabel*.

25.208.2 Member Functions

<i>int</i>	<i>get_line_count () const</i>
<i>int</i>	<i>get_line_height () const</i>
<i>int</i>	<i>get_total_character_count () const</i>
<i>int</i>	<i>get_visible_line_count () const</i>

25.208.3 Member Variables

- *Align align* - Controls the text's horizontal align. Supports left, center, right, and fill, or justify. Set it to one of the ALIGN__* constants.
- *bool autowrap* - If true, wraps the text inside the node's bounding rectangle. If you resize the node, it will change its height automatically to show all the text. Default: false.
- *bool clip_text* - If true, the Label only shows the text that fits inside its bounding rectangle. It also lets you scale the node down freely.
- *int lines_skipped* - The node ignores the first lines_skipped lines before it starts to display text.
- *int max_lines_visible* - Limits the lines of text the node shows on screen.
- *float percent_visible* - Limits the count of visible characters. If you set percent_visible to 50, only up to half of the text's characters will display on screen. Useful to animate the text in a dialog box.
- *String text* - The text to display on screen.
- *bool uppercase* - If true, all the text displays as UPPERCASE.
- *VAlign valign* - Controls the text's vertical align. Supports top, center, bottom, and fill. Set it to one of the VALIGN__* constants.
- *int visible_characters* - Restricts the number of characters to display. Set to -1 to disable.

25.208.4 Enums

enum Align

- **ALIGN_LEFT = 0** — Align rows to the left (default).
- **ALIGN_CENTER = 1** — Align rows centered.
- **ALIGN_RIGHT = 2** — Align rows to the right (default).
- **ALIGN_FILL = 3** — Expand row whitespaces to fit the width.

enum VAlign

- **VALIGN_TOP = 0** — Align the whole text to the top.
- **VALIGN_CENTER = 1** — Align the whole text to the center.
- **VALIGN_BOTTOM = 2** — Align the whole text to the bottom.
- **VALIGN_FILL = 3** — Align the whole text by spreading the rows.

25.208.5 Description

Label displays plain text on the screen. It gives you control over the horizontal and vertical alignment, and can wrap the text inside the node's bounding rectangle. It doesn't support bold, italics or other formatting. For that, use *RichTextLabel* instead.

Note that contrarily to most other *Controls*, Label's *Control.mouse_filter* defaults to MOUSE_FILTER_IGNORE (i.e. it doesn't react to mouse input events).

25.208.6 Member Function Description

- `int get_line_count () const`

Returns the amount of lines of text the Label has.

- `int get_line_height () const`

Returns the font size in pixels.

- `int get_total_character_count () const`

Returns the total length of the text.

- `int get_visible_line_count () const`

Returns the number of lines shown. Useful if the Label ‘s height cannot currently display all lines.

25.209 LargeTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.209.1 Brief Description

A Texture capable of storing many smaller Textures with offsets.

25.209.2 Member Functions

<code>int</code>	<code>add_piece (Vector2 ofs, Texture texture)</code>
<code>void</code>	<code>clear ()</code>
<code>int</code>	<code>get_piece_count () const</code>
<code>Vector2</code>	<code>get_piece_offset (int idx) const</code>
<code>Texture</code>	<code>get_piece_texture (int idx) const</code>
<code>void</code>	<code>set_piece_offset (int idx, Vector2 ofs)</code>
<code>void</code>	<code>set_piece_texture (int idx, Texture texture)</code>
<code>void</code>	<code>set_size (Vector2 size)</code>

25.209.3 Description

A Texture capable of storing many smaller Textures with offsets.

You can dynamically add pieces(*Texture*) to this LargeTexture using different offsets.

25.209.4 Member Function Description

- `int add_piece (Vector2 ofs, Texture texture)`

Add another *Texture* to this LargeTexture, starting on offset “ofs”.

- `void clear ()`

Clears the LargeTexture.

- `int get_piece_count () const`

Returns the number of pieces currently in this LargeTexture.

- `Vector2 get_piece_offset (int idx) const`

Returns the offset of the piece with index “idx”.

- `Texture get_piece_texture (int idx) const`

Returns the *Texture* of the piece with index “idx”.

- `void set_piece_offset (int idx, Vector2 ofs)`

Sets the offset of the piece with index “idx” to “ofs”.

- `void set_piece_texture (int idx, Texture texture)`

Sets the *Texture* of the piece with index “idx” to “texture”.

- `void set_size (Vector2 size)`

Sets the size of this LargeTexture.

25.210 Light

Inherits: *VisualInstance < Spatial < Node < Object*

Inherited By: *SpotLight, OmniLight, DirectionalLight*

Category: Core

25.210.1 Brief Description

Provides a base class for different kinds of light nodes.

25.210.2 Member Variables

- `bool editor_only`
- `BakeMode light_bake_mode`
- `Color light_color`
- `int light_cull_mask`
- `float light_energy`
- `float light_indirect_energy`
- `bool light_negative`
- `float light_specular`
- `float shadow_bias`
- `Color shadow_color`
- `float shadow_contact`

- *bool* **shadow_enabled**
- *bool* **shadow_reverse_cull_face**

25.210.3 Enums

enum **BakeMode**

- **BAKE_DISABLED = 0**
- **BAKE_INDIRECT = 1**
- **BAKE_ALL = 2**

enum **Param**

- **PARAM_ENERGY = 0**
- **PARAM_INDIRECT_ENERGY = 1**
- **PARAM_SPECULAR = 2**
- **PARAM_RANGE = 3**
- **PARAM_ATTENUATION = 4**
- **PARAM_SPOT_ANGLE = 5**
- **PARAM_SPOT_ATTENUATION = 6**
- **PARAM_CONTACT_SHADOW_SIZE = 7**
- **PARAM_SHADOW_MAX_DISTANCE = 8**
- **PARAM_SHADOW_SPLIT_1_OFFSET = 9**
- **PARAM_SHADOW_SPLIT_2_OFFSET = 10**
- **PARAM_SHADOW_SPLIT_3_OFFSET = 11**
- **PARAM_SHADOW_NORMAL_BIAS = 12**
- **PARAM_SHADOW_BIAS = 13**
- **PARAM_SHADOW_BIAS_SPLIT_SCALE = 14**
- **PARAM_MAX = 15**

25.210.4 Description

Light is the abstract base class for light nodes, so it shouldn't be used directly (It can't be instanced). Other types of light nodes inherit from it. Light contains the common variables and parameters used for lighting.

25.211 Light2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.211.1 Brief Description

Casts light in a 2D environment.

25.211.2 Member Variables

- *Color color* - The Light2D's *Color*.
- *bool editor_only* - If `true` Light2D will only appear when editing the scene. Default value: `false`.
- *bool enabled* - If `true` Light2D will emit light. Default value: `true`.
- *float energy* - The Light2D's energy value. The larger the value, the stronger the light.
- *Mode mode* - The Light2D's mode. See `MODE_*` constants for values.
- *Vector2 offset* - The offset of the Light2D's *texture*.
- *float range_height* - The height of the Light2D. Used with 2D normal mapping.
- *int range_item_cull_mask* - The layer mask. Only objects with a matching mask will be affected by the Light2D.
- *int range_layer_max* - Maximum layer value of objects that are affected by the Light2D. Default value: 0.
- *int range_layer_min* - Minimum layer value of objects that are affected by the Light2D. Default value: 0.
- *int range_z_max* - Maximum z value of objects that are affected by the Light2D. Default value: 1024.
- *int range_z_min* - Minimum z value of objects that are affected by the Light2D. Default value: -1024.
- *int shadow_buffer_size* - Shadow buffer size. Default value: 2048.
- *Color shadow_color* - *Color* of shadows cast by the Light2D.
- *bool shadow_enabled* - If `true` the Light2D will cast shadows. Default value: `false`.
- *ShadowFilter shadow_filter* - Shadow filter type. Use `SHADOW_FILTER_*` constants to set *shadow_filter*. Default value: `None`.
- *float shadow_filter_smooth* - Smoothing value for shadows.
- *float shadow_gradient_length* - Smooth shadow gradient length.
- *int shadow_item_cull_mask* - The shadow mask. Used with `LightOccluder2D` to cast shadows. Only occluders with a matching shadow mask will cast shadows.
- *Texture texture* - *Texture* used for the Light2D's appearance.
- *float texture_scale* - The *texture*'s scale factor.

25.211.3 Enums

enum Mode

- **MODE_ADD = 0** — Adds the value of pixels corresponding to the Light2D to the values of pixels under it. This is the common behaviour of a light.
- **MODE_SUB = 1** — Subtracts the value of pixels corresponding to the Light2D to the values of pixels under it, resulting in inverted light effect.

- **MODE_MIX = 2** — Mix the value of pixels corresponding to the Light2D to the values of pixels under it by linear interpolation.
- **MODE_MASK = 3** — The light texture of the Light2D is used as a mask, hiding or revealing parts of the screen underneath depending on the value of each pixel of the light (mask) texture.

enum ShadowFilter

- **SHADOW_FILTER_NONE = 0** — No filter applies to the shadow map. See *shadow_filter*.
- **SHADOW_FILTER_PCF3 = 1** — Percentage closer filtering (3 samples) applies to the shadow map. See *shadow_filter*.
- **SHADOW_FILTER_PCF5 = 2** — Percentage closer filtering (5 samples) applies to the shadow map. See *shadow_filter*.
- **SHADOW_FILTER_PCF7 = 3** — Percentage closer filtering (7 samples) applies to the shadow map. See *shadow_filter*.
- **SHADOW_FILTER_PCF9 = 4** — Percentage closer filtering (9 samples) applies to the shadow map. See *shadow_filter*.
- **SHADOW_FILTER_PCF13 = 5** — Percentage closer filtering (13 samples) applies to the shadow map. See *shadow_filter*.

25.211.4 Description

Casts light in a 2D environment. Light is defined by a (usually grayscale) texture, a color, an energy value, a mode (see constants), and various other parameters (range and shadows-related). Note that Light2D can be used as a mask.

25.212 LightOccluder2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.212.1 Brief Description

Occludes light cast by a Light2D, casting shadows.

25.212.2 Member Variables

- *int light_mask* - The LightOccluder2D's light mask. The LightOccluder2D will cast shadows only from Light2D(s) that have the same light mask(s).
- *OccluderPolygon2D occluder* - The *OccluderPolygon2D* used to compute the shadow.

25.212.3 Description

Occludes light cast by a Light2D, casting shadows. The LightOccluder2D must be provided with an *OccluderPolygon2D* in order for the shadow to be computed.

25.213 Line2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.213.1 Brief Description

A 2D line.

25.213.2 Member Functions

void	<i>add_point (Vector2 position)</i>
<i>int</i>	<i>get_point_count () const</i>
<i>Vector2</i>	<i>get_point_position (int i) const</i>
void	<i>remove_point (int i)</i>
void	<i>set_point_position (int i, Vector2 position)</i>

25.213.3 Member Variables

- *LineCapMode begin_cap_mode* - Controls the style of the line's first point. Use LINE_CAP_* constants. Default value: LINE_CAP_NONE.
- *Color default_color* - The line's color. Will not be used if a gradient is set.
- *LineCapMode end_cap_mode* - Controls the style of the line's last point. Use LINE_CAP_* constants. Default value: LINE_CAP_NONE.
- *Gradient gradient* - The gradient is drawn through the whole line from start to finish. The default color will not be used if a gradient is set.
- *LineJointMode joint_mode* - The style for the points between the start and the end.
- *PoolVector2Array points* - The points that form the lines. The line is drawn between every point set in this array.
- *int round_precision* - The smoothness of the rounded joints and caps. This is only used if a cap or joint is set as round.
- *float sharp_limit* - The direction difference in radians between vector points. This value is only used if joint mode is set to LINE_JOINT_SHARP.
- *Texture texture* - The texture used for the line's texture. Uses texture_mode for drawing style.
- *LineTextureMode texture_mode* - The style to render the texture on the line. Use LINE_TEXTURE_* constants. Default value: LINE_TEXTURE_NONE.
- *float width* - The line's width.

25.213.4 Enums

enum **LineCapMode**

- **LINE_CAP_NONE = 0** — Don't have a line cap.
- **LINE_CAP_BOX = 1** — Draws the line cap as a box.
- **LINE_CAP_ROUND = 2** — Draws the line cap as a circle.

enum LineTextureMode

- **LINE_TEXTURE_NONE = 0** — Takes the left pixels of the texture and renders it over the whole line.
- **LINE_TEXTURE_TILE = 1** — Tiles the texture over the line. The texture need to be imported with Repeat Enabled for it to work properly.

enum LineJointMode

- **LINE_JOINT_SHARP = 0** — The line's joints will be pointy. If sharp_limit is greater than the rotation of a joint, it becomes a bevel joint instead.
- **LINE_JOINT_BEVEL = 1** — The line's joints will be bevelled/chamfered.
- **LINE_JOINT_ROUND = 2** — The line's joints will be rounded.

25.213.5 Description

A line through several points in 2D space.

25.213.6 Member Function Description

- `void add_point (Vector2 position)`

Add a point at the `position`. Appends the point at the end of the line.

- `int get_point_count () const`

Returns the Line2D's amount of points.

- `Vector2 get_point_position (int i) const`

Returns point `i`'s position.

- `void remove_point (int i)`

Remove the point at index `i` from the line.

- `void set_point_position (int i, Vector2 position)`

Overwrites the position in point `i` with the supplied `position`.

25.214 LineEdit

Inherits: `Control < CanvasItem < Node < Object`

Category: Core

25.214.1 Brief Description

Control that provides single line string editing.

25.214.2 Member Functions

void	<i>append_at_cursor (String text)</i>
void	<i>clear ()</i>
void	<i>deselect ()</i>
<i>PopupMenu</i>	<i>get_menu () const</i>
void	<i>menu_option (int option)</i>
void	<i>select (int from=0, int to=-1)</i>
void	<i>select_all ()</i>

25.214.3 Signals

- **text_changed** (*String new_text*)

Emitted when the text changes.

- **text_entered** (*String new_text*)

Emitted when the user presses KEY_ENTER on the `LineEdit`.

25.214.4 Member Variables

- **Align align** - Text alignment as defined in the ALIGN_* enum.
- **bool caret_blink** - If true the caret (visual cursor) blinks.
- **float caret_blink_speed** - Duration (in seconds) of a caret's blinking cycle.
- **int caret_position** - The cursor's position inside the `LineEdit`. When set, the text may scroll to accommodate it.
- **bool context_menu_enabled** - If true the context menu will appear when right clicked.
- **bool editable** - If false existing text cannot be modified and new text cannot be added.
- **bool expand_to_text_length** - If true the `LineEdit` width will increase to stay longer than the *text*. It will **not** compress if the *text* is shortened.
- **FocusMode focus_mode** - Defines how the `LineEdit` can grab focus (Keyboard and mouse, only keyboard, or none). See enum `FocusMode` in *Control* for details.
- **int max_length** - Maximum amount of characters that can be entered inside the `LineEdit`. If 0, there is no limit.
- **float placeholder_alpha** - Opacity of the *placeholder_text*. From 0 to 1.
- **String placeholder_text** - Text shown when the `LineEdit` is empty. It is **not** the `LineEdit`'s default value (see *text*).
- **bool secret** - If true every character is shown as “*”.
- **String text** - String value of the `LineEdit`.

25.214.5 Enums

enum **Align**

- **ALIGN_LEFT = 0** — Aligns the text on the left hand side of the *LineEdit*.
- **ALIGN_CENTER = 1** — Centers the text in the middle of the *LineEdit*.
- **ALIGN_RIGHT = 2** — Aligns the text on the right hand side of the *LineEdit*.
- **ALIGN_FILL = 3** — Stretches whitespaces to fit the *LineEdit*'s width.

enum MenuItems

- **MENU_CUT = 0** — Cuts (Copies and clears) the selected text.
- **MENU_COPY = 1** — Copies the selected text.
- **MENU_PASTE = 2** — Pastes the clipboard text over the selected text (or at the cursor's position).
- **MENU_CLEAR = 3** — Erases the whole Linedit text.
- **MENU_SELECT_ALL = 4** — Selects the whole Linedit text.
- **MENU_UNDO = 5** — Undoes the previous action.
- **MENU_REDO = 6**
- **MENU_MAX = 7**

25.214.6 Description

LineEdit provides a single line string editor, used for text fields.

25.214.7 Member Function Description

- void **append_at_cursor (String text)**

Adds `text` after the cursor. If the resulting value is longer than `max_length`, nothing happens.

- void **clear ()**

Erases the *LineEdit* text.

- void **deselect ()**

Clears the current selection.

- *PopupMenu* **get_menu () const**

Returns the *PopupMenu* of this *LineEdit*. By default, this menu is displayed when right-clicking on the *LineEdit*.

- void **menu_option (int option)**

Executes a given action as defined in the `MENU_*` enum.

- void **select (int from=0, int to=-1)**

Selects characters inside *LineEdit* between `from` and `to`. By default `from` is at the beginning and `to` at the end.

```
text = "Welcome"
select()      # Welcome
select(4)      # ome
select(2, 5)  # lco
```

- void **select_all ()**

Selects the whole `String`.

25.215 LineShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.215.1 Brief Description

Line shape for 2D collisions.

25.215.2 Member Variables

- *float d* - The line's distance from the origin.
- *Vector2 normal* - The line's normal.

25.215.3 Description

Line shape for 2D collisions. It works like a 2D plane and will not allow any body to go to the negative side. Not recommended for rigid bodies, and usually not recommended for static bodies either because it forces checks against it on every frame.

25.216 LinkButton

Inherits: *BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.216.1 Brief Description

Simple button used to represent a link to some resource

25.216.2 Member Variables

- *String text*
- *UnderlineMode underline*

25.216.3 Enums

enum **UnderlineMode**

- **UNDERLINE_MODE_ALWAYS = 0** — The LinkButton will always show an underline at the bottom of its text

- **UNDERLINE_MODE_ON_HOVER = 1** — The LinkButton will show an underline at the bottom of its text when the mouse cursor is over it.
- **UNDERLINE_MODE_NEVER = 2** — The LinkButton will never show an underline at the bottom of its text.

25.216.4 Description

This kind of buttons are primarily used when the interaction with the button causes a context change (like linking to a web page).

25.217 Listener

Inherits: *Spatial < Node < Object*

Category: Core

25.217.1 Brief Description

25.217.2 Member Functions

void	<i>clear_current ()</i>
<i>Transform</i>	<i>get_listener_transform () const</i>
<i>bool</i>	<i>is_current () const</i>
void	<i>make_current ()</i>

25.217.3 Member Function Description

- void **clear_current ()**
- *Transform* **get_listener_transform () const**
- *bool* **is_current () const**
- void **make_current ()**

25.218 MainLoop

Inherits: *Object*

Inherited By: *SceneTree*

Category: Core

25.218.1 Brief Description

Main loop is the abstract main loop base class.

25.218.2 Member Functions

void	<code>_drop_files (PoolStringArray files, int screen)</code>	virtual
void	<code>_finalize ()</code>	virtual
void	<code>_idle (float delta)</code>	virtual
void	<code>_initialize ()</code>	virtual
void	<code>_input_event (InputEvent ev)</code>	virtual
void	<code>_input_text (String text)</code>	virtual
void	<code>_iteration (float delta)</code>	virtual
void	<code>finish ()</code>	
bool	<code>idle (float delta)</code>	
void	<code>init ()</code>	
void	<code>input_event (InputEvent ev)</code>	
void	<code>input_text (String text)</code>	
bool	<code>iteration (float delta)</code>	

25.218.3 Numeric Constants

- NOTIFICATION_WM_MOUSE_ENTER = 2
- NOTIFICATION_WM_MOUSE_EXIT = 3
- NOTIFICATION_WM_FOCUS_IN = 4
- NOTIFICATION_WM_FOCUS_OUT = 5
- NOTIFICATION_WM_QUIT_REQUEST = 6
- NOTIFICATION_WM_GO_BACK_REQUEST = 7
- NOTIFICATION_WM_UNFOCUS_REQUEST = 8
- NOTIFICATION_OS_MEMORY_WARNING = 9
- NOTIFICATION_TRANSLATION_CHANGED = 90
- NOTIFICATION_WM_ABOUT = 91

25.218.4 Description

Main loop is the abstract main loop base class. All other main loop classes are derived from it. Upon application start, a `MainLoop` has to be provided to OS, else the application will exit. This happens automatically (and a `SceneTree` is created), unless a main `Script` is supplied, which may or not create and return a `MainLoop`.

25.218.5 Member Function Description

- void `_drop_files (PoolStringArray files, int screen)` virtual
- void `_finalize ()` virtual

Called before the program exits.

- void `_idle (float delta)` virtual

Called each idle frame with time since last call as an only argument.

- void `_initialize ()` virtual

Called once during initialization.

- void **_input_event** (*InputEvent ev*) virtual
- void **_input_text** (*String text*) virtual
- void **_iteration** (*float delta*) virtual
- void **finish** ()
- *bool idle* (*float delta*)
- void **init** ()
- void **input_event** (*InputEvent ev*)
- void **input_text** (*String text*)
- *bool iteration* (*float delta*)

25.219 MarginContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Category: Core

25.219.1 Brief Description

Simple margin container.

25.219.2 Description

Simple margin container. Adds a left margin to anything contained.

25.220 Marshalls

Inherits: *Reference < Object*

Category: Core

25.220.1 Brief Description

Data transformation (marshalling) and encoding helpers.

25.220.2 Member Functions

<i>PoolByteArray</i>	<i>base64_to_raw</i> (<i>String base64_str</i>)
<i>String</i>	<i>base64_to_utf8</i> (<i>String base64_str</i>)
<i>Variant</i>	<i>base64_to_variant</i> (<i>String base64_str</i>)
<i>String</i>	<i>raw_to_base64</i> (<i>PoolByteArray array</i>)
<i>String</i>	<i>utf8_to_base64</i> (<i>String utf8_str</i>)
<i>String</i>	<i>variant_to_base64</i> (<i>Variant variant</i>)

25.220.3 Description

Provides data transformation and encoding utility functions.

25.220.4 Member Function Description

- *PoolByteArray* **base64_to_raw** (*String* base64_str)

Return *PoolByteArray* of a given base64 encoded String.

- *String* **base64_to_utf8** (*String* base64_str)

Return utf8 String of a given base64 encoded String.

- *Variant* **base64_to_variant** (*String* base64_str)

Return *Variant* of a given base64 encoded String.

- *String* **raw_to_base64** (*PoolByteArray* array)

Return base64 encoded String of a given *PoolByteArray*.

- *String* **utf8_to_base64** (*String* utf8_str)

Return base64 encoded String of a given utf8 String.

- *String* **variant_to_base64** (*Variant* variant)

Return base64 encoded String of a given *Variant*.

25.221 Material

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *ParticlesMaterial*, *ShaderMaterial*, *SpatialMaterial*, *CanvasItemMaterial*

Category: Core

25.221.1 Brief Description

Abstract base *Resource* for coloring and shading geometry.

25.221.2 Member Variables

- *Material* **next_pass**
- *int* **render_priority**

25.221.3 Numeric Constants

- **RENDER_PRIORITY_MAX** = 127
- **RENDER_PRIORITY_MIN** = -128

25.221.4 Description

Material is a base *Resource* used for coloring and shading geometry. All materials inherit from it and almost all *VisualInstance* derived nodes carry a Material. A few flags and parameters are shared between all material types and are configured here.

25.222 MenuButton

Inherits: *Button < BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.222.1 Brief Description

Special button that brings up a *PopupMenu* when clicked.

25.222.2 Member Functions

<i>PopupMenu</i>	<code>get_popup() const</code>
<code>void</code>	<code>set_disable_shortcuts(bool disabled)</code>

25.222.3 Signals

- `about_to_show()`

Emitted when *PopupMenu* of this MenuButton is about to show.

25.222.4 Description

Special button that brings up a *PopupMenu* when clicked. That's pretty much all it does, as it's just a helper class when building GUIs.

25.222.5 Member Function Description

- *PopupMenu* `get_popup() const`

Return the *PopupMenu* contained in this button.

- `void set_disable_shortcuts(bool disabled)`

25.223 Mesh

Inherits: *Resource < Reference < Object*

Inherited By: *ArrayMesh, PrimitiveMesh*

Category: Core

25.223.1 Brief Description

A *Resource* that contains vertex-array based geometry.

25.223.2 Member Functions

<i>Shape</i>	<i>create_convex_shape () const</i>
<i>Mesh</i>	<i>create_outline (float margin) const</i>
<i>Shape</i>	<i>create_trimesh_shape () const</i>
<i>TriangleMesh</i>	<i>generate_triangle_mesh () const</i>
<i>PoolVector3Array</i>	<i>get_faces () const</i>

25.223.3 Member Variables

- *Vector2 lightmap_size_hint*

25.223.4 Enums

enum **BlendShapeMode**

- **BLEND_SHAPE_MODE_NORMALIZED** = 0
- **BLEND_SHAPE_MODE_RELATIVE** = 1

enum **ArrayType**

- **ARRAY_VERTEX** = 0
- **ARRAY_NORMAL** = 1
- **ARRAY_TANGENT** = 2
- **ARRAY_COLOR** = 3
- **ARRAY_TEX_UV** = 4
- **ARRAY_TEX_UV2** = 5
- **ARRAY_BONES** = 6
- **ARRAY_WEIGHTS** = 7
- **ARRAY_INDEX** = 8
- **ARRAY_MAX** = 9

enum **ArrayFormat**

- **ARRAY_FORMAT_VERTEX** = 1
- **ARRAY_FORMAT_NORMAL** = 2
- **ARRAY_FORMAT_TANGENT** = 4
- **ARRAY_FORMAT_COLOR** = 8
- **ARRAY_FORMAT_TEX_UV** = 16

- **ARRAY_FORMAT_TEX_UV2** = 32
- **ARRAY_FORMAT_BONES** = 64
- **ARRAY_FORMAT_WEIGHTS** = 128
- **ARRAY_FORMAT_INDEX** = 256
- **ARRAY_COMPRESS_BASE** = 9
- **ARRAY_COMPRESS_VERTEX** = 512
- **ARRAY_COMPRESS_NORMAL** = 1024
- **ARRAY_COMPRESS_TANGENT** = 2048
- **ARRAY_COMPRESS_COLOR** = 4096
- **ARRAY_COMPRESS_TEX_UV** = 8192
- **ARRAY_COMPRESS_TEX_UV2** = 16384
- **ARRAY_COMPRESS_BONES** = 32768
- **ARRAY_COMPRESS_WEIGHTS** = 65536
- **ARRAY_COMPRESS_INDEX** = 131072
- **ARRAY_FLAG_USE_2D_VERTICES** = 262144
- **ARRAY_FLAG_USE_16_BIT_BONES** = 524288
- **ARRAY_COMPRESS_DEFAULT** = 97792

enum **PrimitiveType**

- **PRIMITIVE_POINTS** = 0 — Render array as points (one vertex equals one point).
- **PRIMITIVE_LINES** = 1 — Render array as lines (every two vertices a line is created).
- **PRIMITIVE_LINE_STRIP** = 2 — Render array as line strip.
- **PRIMITIVE_LINE_LOOP** = 3 — Render array as line loop (like line strip, but closed).
- **PRIMITIVE_TRIANGLES** = 4 — Render array as triangles (every three vertices a triangle is created).
- **PRIMITIVE_TRIANGLE_STRIP** = 5 — Render array as triangle strips.
- **PRIMITIVE_TRIANGLE_FAN** = 6 — Render array as triangle fans.

25.223.5 Description

Mesh is a type of *Resource* that contains vertex-array based geometry, divided in *surfaces*. Each surface contains a completely separate array and a material used to draw it. Design wise, a mesh with multiple surfaces is preferred to a single surface, because objects created in 3D editing software commonly contain multiple materials.

25.223.6 Member Function Description

- *Shape* **create_convex_shape** () const

Calculate a *ConvexPolygonShape* from the mesh.

- *Mesh* **create_outline** (float margin) const

Calculate an outline mesh at a defined offset (margin) from the original mesh. Note: Typically returns the vertices in reverse order (e.g. clockwise to anti-clockwise).

- `Shape create_trimesh_shape () const`

Calculate a *ConcavePolygonShape* from the mesh.

- `TriangleMesh generate_triangle_mesh () const`

Generate a *TriangleMesh* from the mesh.

- `PoolVector3Array get_faces () const`

Returns all the vertices that make up the faces of the mesh. Each three vertices represent one triangle.

25.224 MeshDataTool

Inherits: *Reference < Object*

Category: Core

25.224.1 Brief Description

25.224.2 Member Functions

<code>void</code>	<code>clear ()</code>
<code>int</code>	<code>commit_to_surface (ArrayMesh mesh)</code>
<code>int</code>	<code>create_from_surface (ArrayMesh mesh, int surface)</code>
<code>int</code>	<code>get_edge_count () const</code>
<code>PoolIntArray</code>	<code>get_edge_faces (int idx) const</code>
<code>Variant</code>	<code>get_edge_meta (int idx) const</code>
<code>int</code>	<code>get_edge_vertex (int idx, int vertex) const</code>
<code>int</code>	<code>get_face_count () const</code>
<code>int</code>	<code>get_face_edge (int idx, int edge) const</code>
<code>Variant</code>	<code>get_face_meta (int idx) const</code>
<code>Vector3</code>	<code>get_face_normal (int idx) const</code>
<code>int</code>	<code>get_face_vertex (int idx, int vertex) const</code>
<code>int</code>	<code>get_format () const</code>
<code>Material</code>	<code>get_material () const</code>
<code>Vector3</code>	<code>get_vertex (int idx) const</code>
<code>PoolIntArray</code>	<code>get_vertex_bones (int idx) const</code>
<code>Color</code>	<code>get_vertex_color (int idx) const</code>
<code>int</code>	<code>get_vertex_count () const</code>
<code>PoolIntArray</code>	<code>get_vertex_edges (int idx) const</code>
<code>PoolIntArray</code>	<code>get_vertex_faces (int idx) const</code>
<code>Variant</code>	<code>get_vertex_meta (int idx) const</code>
<code>Vector3</code>	<code>get_vertex_normal (int idx) const</code>
<code>Plane</code>	<code>get_vertex_tangent (int idx) const</code>
<code>Vector2</code>	<code>get_vertex_uv (int idx) const</code>
<code>Vector2</code>	<code>get_vertex_uv2 (int idx) const</code>
<code>PoolRealArray</code>	<code>get_vertex_weights (int idx) const</code>
<code>void</code>	<code>set_edge_meta (int idx, Variant meta)</code>
<code>void</code>	<code>set_face_meta (int idx, Variant meta)</code>
<code>void</code>	<code>set_material (Material material)</code>

Continued on next page

Table 25.13 – continued from previous page

void	<i>set_vertex</i> (int idx, <i>Vector3</i> vertex)
void	<i>set_vertex_bones</i> (int idx, <i>PoolIntArray</i> bones)
void	<i>set_vertex_color</i> (int idx, <i>Color</i> color)
void	<i>set_vertex_meta</i> (int idx, <i>Variant</i> meta)
void	<i>set_vertex_normal</i> (int idx, <i>Vector3</i> normal)
void	<i>set_vertex_tangent</i> (int idx, <i>Plane</i> tangent)
void	<i>set_vertex_uv</i> (int idx, <i>Vector2</i> uv)
void	<i>set_vertex_uv2</i> (int idx, <i>Vector2</i> uv2)
void	<i>set_vertex_weights</i> (int idx, <i>PoolRealArray</i> weights)

25.224.3 Member Function Description

- void **clear** ()
- *int commit_to_surface* (*ArrayMesh* mesh)
- *int create_from_surface* (*ArrayMesh* mesh, *int* surface)
- *int get_edge_count* () const
- *PoolIntArray get_edge_faces* (*int* idx) const
- *Variant get_edge_meta* (*int* idx) const
- *int get_edge_vertex* (*int* idx, *int* vertex) const
- *int get_face_count* () const
- *int get_face_edge* (*int* idx, *int* edge) const
- *Variant get_face_meta* (*int* idx) const
- *Vector3 get_face_normal* (*int* idx) const
- *int get_face_vertex* (*int* idx, *int* vertex) const
- *int get_format* () const
- *Material get_material* () const
- *Vector3 get_vertex* (*int* idx) const
- *PoolIntArray get_vertex_bones* (*int* idx) const
- *Color get_vertex_color* (*int* idx) const
- *int get_vertex_count* () const
- *PoolIntArray get_vertex_edges* (*int* idx) const
- *PoolIntArray get_vertex_faces* (*int* idx) const
- *Variant get_vertex_meta* (*int* idx) const
- *Vector3 get_vertex_normal* (*int* idx) const
- *Plane get_vertex_tangent* (*int* idx) const
- *Vector2 get_vertex_uv* (*int* idx) const
- *Vector2 get_vertex_uv2* (*int* idx) const
- *PoolRealArray get_vertex_weights* (*int* idx) const
- void **set_edge_meta** (*int* idx, *Variant* meta)

- void **set_face_meta** (int idx, Variant meta)
- void **set_material** (Material material)
- void **set_vertex** (int idx, Vector3 vertex)
- void **set_vertex_bones** (int idx, PoolIntArray bones)
- void **set_vertex_color** (int idx, Color color)
- void **set_vertex_meta** (int idx, Variant meta)
- void **set_vertex_normal** (int idx, Vector3 normal)
- void **set_vertex_tangent** (int idx, Plane tangent)
- void **set_vertex_uv** (int idx, Vector2 uv)
- void **set_vertex_uv2** (int idx, Vector2 uv2)
- void **set_vertex_weights** (int idx, PoolRealArray weights)

25.225 MeshInstance

Inherits: GeometryInstance < VisualInstance < Spatial < Node < Object

Category: Core

25.225.1 Brief Description

Node that instances meshes into a scenario.

25.225.2 Member Functions

void	<i>create_convex_collision</i> ()
void	<i>create_debug_tangents</i> ()
void	<i>create_trimesh_collision</i> ()
Material	<i>get_surface_material</i> (int surface) const
void	<i>set_surface_material</i> (int surface, Material material)

25.225.3 Member Variables

- **Mesh mesh** - The *Mesh* resource for the instance.
- **NodePath skeleton** - *NodePath* to the *Skeleton* associated with the instance.

25.225.4 Description

MeshInstance is a node that takes a *Mesh* resource and adds it to the current scenario by creating an instance of it. This is the class most often used to get 3D geometry rendered and can be used to instance a single *Mesh* in many places. This allows to reuse geometry and save on resources. When a *Mesh* has to be instanced more than thousands of times at close proximity, consider using a *MultiMesh* in a *MultiMeshInstance* instead.

25.225.5 Member Function Description

- void **create_convex_collision ()**

This helper creates a *StaticBody* child node with a *ConvexPolygonShape* collision shape calculated from the mesh geometry. It's mainly used for testing.

- void **create_debug_tangents ()**

This helper creates a *MeshInstance* child node with gizmos at every vertex calculated from the mesh geometry. It's mainly used for testing.

- void **create_trimesh_collision ()**

This helper creates a *StaticBody* child node with a *ConcavePolygonShape* collision shape calculated from the mesh geometry. It's mainly used for testing.

- *Material* **get_surface_material (int surface) const**

Returns the *Material* for a surface of the *Mesh* resource.

- void **set_surface_material (int surface, Material material)**

Sets the *Material* for a surface of the *Mesh* resource.

25.226 MeshLibrary

Inherits: *Resource < Reference < Object*

Category: Core

25.226.1 Brief Description

Library of meshes.

25.226.2 Member Functions

void	<i>clear ()</i>
void	<i>create_item (int id)</i>
<i>int</i>	<i>find_item_by_name (String name) const</i>
<i>PoolIntArray</i>	<i>get_item_list () const</i>
<i>Mesh</i>	<i>get_item_mesh (int id) const</i>
<i>String</i>	<i>get_item_name (int id) const</i>
<i>NavigationMesh</i>	<i>get_item_navmesh (int id) const</i>
<i>Texture</i>	<i>get_item_preview (int id) const</i>
<i>Array</i>	<i>get_item_shapes (int id) const</i>
<i>int</i>	<i>get_last_unused_item_id () const</i>
void	<i>remove_item (int id)</i>
void	<i>set_item_mesh (int id, Mesh mesh)</i>
void	<i>set_item_name (int id, String name)</i>
void	<i>set_item_navmesh (int id, NavigationMesh navmesh)</i>
void	<i>set_item_preview (int id, Texture texture)</i>
void	<i>set_item_shapes (int id, Array shapes)</i>

25.226.3 Description

Library of meshes. Contains a list of *Mesh* resources, each with name and ID. Useful for GridMap or painting Terrain.

25.226.4 Member Function Description

- void **clear()**

Clear the library.

- void **create_item(int id)**

Create a new item in the library, supplied an id.

- *int* **find_item_by_name(String name) const**
- *PoolIntArray* **get_item_list() const**

Return the list of items.

- *Mesh* **get_item_mesh(int id) const**

Return the mesh of the item.

- *String* **get_item_name(int id) const**

Return the name of the item.

- *NavigationMesh* **get_item_navmesh(int id) const**
- *Texture* **get_item_preview(int id) const**
- *Array* **get_item_shapes(int id) const**
- *int* **get_last_unused_item_id() const**

Get an unused id for a new item.

- void **remove_item(int id)**

Remove the item.

- void **set_item_mesh(int id, Mesh mesh)**

Set the mesh of the item.

- void **set_item_name(int id, String name)**

Set the name of the item.

- void **set_item_navmesh(int id, NavigationMesh navmesh)**
- void **set_item_preview(int id, Texture texture)**
- void **set_item_shapes(int id, Array shapes)**

25.227 MobileVRInterface

Inherits: *ARVRInterface* < *Reference* < *Object*

Category: Core

25.227.1 Brief Description

Generic mobile VR implementation

25.227.2 Member Variables

- *float display_to_lens* - The distance between the display and the lenses inside of the device in centimeters.
- *float display_width* - The width of the display in centimeters.
- *float iod* - The interocular distance, also known as the interpupillary distance. The distance between the pupils of the left and right eye.
- *float k1* - The k1 lens factor is one of the two constants that define the strength of the lens used and directly influences the lens distortion effect.
- *float k2* - The k2 lens factor, see k1.
- *float oversample* - The oversample setting. Because of the lens distortion we have to render our buffers at a higher resolution than the screen can natively handle. A value between 1.5 and 2.0 often provides good results but at the cost of performance.

25.227.3 Description

This is a generic mobile VR implementation where you need to provide details about the phone and HMD used. It does not rely on any existing framework. This is the most basic interface we have. For the best effect you do need a mobile phone with a gyroscope and accelerometer.

Note that even though there is no positional tracking the camera will assume the headset is at a height of 1.85 meters.

25.228 MultiMesh

Inherits: *Resource < Reference < Object*

Category: Core

25.228.1 Brief Description

Provides high performance mesh instancing.

25.228.2 Member Functions

<i>AABB</i>	<i>get_aabb () const</i>
<i>Color</i>	<i>get_instance_color (int instance) const</i>
<i>Transform</i>	<i>get_instance_transform (int instance) const</i>
<i>void</i>	<i>set_instance_color (int instance, Color color)</i>
<i>void</i>	<i>set_instance_transform (int instance, Transform transform)</i>

25.228.3 Member Variables

- *ColorFormat* **color_format**
- *int* **instance_count**
- *Mesh* **mesh**
- *TransformFormat* **transform_format**

25.228.4 Enums

enum **TransformFormat**

- **TRANSFORM_2D = 0**
- **TRANSFORM_3D = 1**

enum **ColorFormat**

- **COLOR_NONE = 0**
- **COLOR_8BIT = 1**
- **COLOR_FLOAT = 2**

25.228.5 Description

MultiMesh provides low level mesh instancing. If the amount of *Mesh* instances needed goes from hundreds to thousands (and most need to be visible at close proximity) creating such a large amount of *MeshInstance* nodes may affect performance by using too much CPU or video memory.

For this case a MultiMesh becomes very useful, as it can draw thousands of instances with little API overhead.

As a drawback, if the instances are too far away of each other, performance may be reduced as every single instance will always rendered (they are spatially indexed as one, for the whole object).

Since instances may have any behavior, the AABB used for visibility must be provided by the user.

25.228.6 Member Function Description

- *AABB* **get_aabb () const**

Return the visibility AABB.

- *Color* **get_instance_color (int instance) const**

Get the color of a specific instance.

- *Transform* **get_instance_transform (int instance) const**

Return the transform of a specific instance.

- *void* **set_instance_color (int instance, Color color)**

Set the color of a specific instance.

- *void* **set_instance_transform (int instance, Transform transform)**

Set the transform for a specific instance.

25.229 MultiMeshInstance

Inherits: *GeometryInstance < VisualInstance < Spatial < Node < Object*

Category: Core

25.229.1 Brief Description

Node that instances a *MultiMesh*.

25.229.2 Member Variables

- ***MultiMesh multimesh*** - The *MultiMesh* resource that will be used and shared among all instances of the *MultiMeshInstance*.

25.229.3 Description

MultiMeshInstance is a specialized node to instance *GeometryInstances* based on a *MultiMesh* resource.

This is useful to optimize the rendering of a high amount of instances of a given mesh (for example tree in a forest or grass strands).

25.230 Mutex

Inherits: *Reference < Object*

Category: Core

25.230.1 Brief Description

A synchronization Mutex.

25.230.2 Member Functions

void	<i>lock ()</i>
<i>int</i>	<i>try_lock ()</i>
void	<i>unlock ()</i>

25.230.3 Description

A synchronization Mutex. Element used to synchronize multiple *Threads*. Basically a binary *Semaphore*. Guarantees that only one thread can ever acquire this lock at a time. Can be used to protect a critical section. Be careful to avoid deadlocks.

25.230.4 Member Function Description

- `void lock()`

Lock this `Mutex`, blocks until it is unlocked by the current owner.

- `int try_lock()`

Try locking this `Mutex`, does not block. Returns OK on success, `ERR_BUSY` otherwise.

- `void unlock()`

Unlock this `Mutex`, leaving it to other threads.

25.231 NativeScript

Inherits: `Script < Resource < Reference < Object`

Category: Core

25.231.1 Brief Description

25.231.2 Member Functions

<code>Object</code>	<code>new() vararg</code>
---------------------	---------------------------

25.231.3 Member Variables

- `String class_name`
- `GDNativeLibrary library`

25.231.4 Member Function Description

- `Object new() vararg`

25.232 Navigation

Inherits: `Spatial < Node < Object`

Category: Core

25.232.1 Brief Description

A collection of `NavigationMesh` resources and methods used for pathfinding.

25.232.2 Member Functions

<code>Vector3</code>	<code>get_closest_point (Vector3 to_point)</code>
<code>Vector3</code>	<code>get_closest_point_normal (Vector3 to_point)</code>
<code>Object</code>	<code>get_closest_point_owner (Vector3 to_point)</code>
<code>Vector3</code>	<code>get_closest_point_to_segment (Vector3 start, Vector3 end, bool use_collision=false)</code>
<code>PoolVector3Array</code>	<code>get_simple_path (Vector3 start, Vector3 end, bool optimize=true)</code>
<code>int</code>	<code>navmesh_add (NavigationMesh mesh, Transform xform, Object owner=null)</code>
<code>void</code>	<code>navmesh_remove (int id)</code>
<code>void</code>	<code>navmesh_set_transform (int id, Transform xform)</code>

25.232.3 Member Variables

- `Vector3 up_vector` - Defines which direction is up. The default defines 0,1,0 as up which is the world up direction. To make this a ceiling use 0,-1,0 to define down as up.

25.232.4 Description

The Navigation node is used for basic or advanced navigation. By default it will automatically collect all child NavigationMesh resources, but they can also be added on the fly through scripting. It can be used for generating a simple path between two points or it can be used to ensure that a navigation agent is angled perfectly to the terrain it is navigating.

25.232.5 Member Function Description

- `Vector3 get_closest_point (Vector3 to_point)`

Returns the closest navigation point to the point passed.

- `Vector3 get_closest_point_normal (Vector3 to_point)`

Returns the surface normal of the navigation mesh at the point passed. For instance, if the point passed was at a 45 degree slope it would return something like (0.5,0.5,0). This is useful for rotating a navigation agent in accordance with the NavigationMesh.

- `Object get_closest_point_owner (Vector3 to_point)`

Returns the nearest NavigationMeshInstance to the point passed.

- `Vector3 get_closest_point_to_segment (Vector3 start, Vector3 end, bool use_collision=false)`

Returns the nearest point to the line segment passed. The third optional parameter takes collisions into account.

- `PoolVector3Array get_simple_path (Vector3 start, Vector3 end, bool optimize=true)`

Returns a path of points as a PoolVector3Array. If optimize is false the NavigationMesh agent properties will be taken into account, otherwise it will return the nearest path and ignore agent radius, height, etc.

- `int navmesh_add (NavigationMesh mesh, Transform xform, Object owner=null)`

Adds a NavigationMesh to the list of NavigationMesh's in this node. Returns an id. Its position, rotation and scale are associated with the Transform passed. The Node (or Object) that owns this node is an optional parameter.

- `void navmesh_remove (int id)`

Removes a NavigationMesh from the list of NavigationMesh's in this node.

- `void navmesh_set_transform (int id, Transform xform)`

Associates a NavigationMesh's id with a Transform. Its position, rotation and scale are based on the Transform passed.

25.233 Navigation2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.233.1 Brief Description

25.233.2 Member Functions

<code>Vector2</code>	<code>get_closest_point (Vector2 to_point)</code>
<code>Object</code>	<code>get_closest_point_owner (Vector2 to_point)</code>
<code>PoolVector2Array</code>	<code>get_simple_path (Vector2 start, Vector2 end, bool optimize=true)</code>
<code>int</code>	<code>navpoly_add (NavigationPolygon mesh, Transform2D xform, Object owner=null)</code>
<code>void</code>	<code>navpoly_remove (int id)</code>
<code>void</code>	<code>navpoly_set_transform (int id, Transform2D xform)</code>

25.233.3 Member Function Description

- `Vector2 get_closest_point (Vector2 to_point)`
- `Object get_closest_point_owner (Vector2 to_point)`
- `PoolVector2Array get_simple_path (Vector2 start, Vector2 end, bool optimize=true)`
- `int navpoly_add (NavigationPolygon mesh, Transform2D xform, Object owner=null)`
- `void navpoly_remove (int id)`
- `void navpoly_set_transform (int id, Transform2D xform)`

25.234 NavigationMesh

Inherits: *Resource < Reference < Object*

Category: Core

25.234.1 Brief Description

25.234.2 Member Functions

void	<i>add_polygon (PoolIntArray polygon)</i>
void	<i>clear_polygons ()</i>
void	<i>create_from_mesh (Mesh mesh)</i>
<i>PoolIntArray</i>	<i>get_polygon (int idx)</i>
<i>int</i>	<i>get_polygon_count () const</i>
<i>PoolVector3Array</i>	<i>get_vertices () const</i>
void	<i>set_vertices (PoolVector3Array vertices)</i>

25.234.3 Member Variables

- *float agent/height*
- *float agent/max_climb*
- *float agent/max_slope*
- *float agent/radius*
- *float cell/height*
- *float cell/size*
- *float detail/sample_distance*
- *float detail/sample_max_error*
- *float edge/max_error*
- *float edge/max_length*
- *bool filter/filter_walkable_low_height_spans*
- *bool filter/ledge_spans*
- *bool filter/low_hanging_obstacles*
- *float polygon/verts_per_poly*
- *float region/merge_size*
- *float region/min_size*
- *int sample_partition_type/sample_partition_type*

25.234.4 Numeric Constants

- **SAMPLE_PARTITION_WATERSHED = 0**
- **SAMPLE_PARTITION_MONOTONE = 1**
- **SAMPLE_PARTITION_LAYERS = 2**

25.234.5 Member Function Description

- void **add_polygon** (*PoolIntArray* polygon)
- void **clear_polygons** ()
- void **create_from_mesh** (*Mesh* mesh)
- *PoolIntArray* **get_polygon** (*int* idx)
- *int* **get_polygon_count** () const
- *PoolVector3Array* **get_vertices** () const
- void **set_vertices** (*PoolVector3Array* vertices)

25.235 NavigationMeshInstance

Inherits: *Spatial < Node < Object*

Category: Core

25.235.1 Brief Description

25.235.2 Member Variables

- *bool* **enabled**
- *NavigationMesh* **navmesh**

25.236 NavigationPolygon

Inherits: *Resource < Reference < Object*

Category: Core

25.236.1 Brief Description

25.236.2 Member Functions

void	<code>add_outline (PoolVector2Array outline)</code>
void	<code>add_outline_at_index (PoolVector2Array outline, int index)</code>
void	<code>add_polygon (PoolIntArray polygon)</code>
void	<code>clear_outlines ()</code>
void	<code>clear_polygons ()</code>
<i>PoolVector2Array</i>	<code>get_outline (int idx) const</code>
<i>int</i>	<code>get_outline_count () const</code>
<i>PoolIntArray</i>	<code>get_polygon (int idx)</code>
<i>int</i>	<code>get_polygon_count () const</code>
<i>PoolVector2Array</i>	<code>get_vertices () const</code>
void	<code>make_polygons_from_outlines ()</code>
void	<code>remove_outline (int idx)</code>
void	<code>set_outline (int idx, PoolVector2Array outline)</code>
void	<code>set_vertices (PoolVector2Array vertices)</code>

25.236.3 Member Function Description

- void **add_outline** (*PoolVector2Array* outline)
- void **add_outline_at_index** (*PoolVector2Array* outline, *int* index)
- void **add_polygon** (*PoolIntArray* polygon)
- void **clear_outlines** ()
- void **clear_polygons** ()
- *PoolVector2Array* **get_outline** (*int* idx) const
- *int* **get_outline_count** () const
- *PoolIntArray* **get_polygon** (*int* idx)
- *int* **get_polygon_count** () const
- *PoolVector2Array* **get_vertices** () const
- void **make_polygons_from_outlines** ()
- void **remove_outline** (*int* idx)
- void **set_outline** (*int* idx, *PoolVector2Array* outline)
- void **set_vertices** (*PoolVector2Array* vertices)

25.237 NavigationPolygonInstance

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

25.237.1 Brief Description

25.237.2 Member Variables

- *bool* **enabled**
- *NavigationPolygon* **navpoly**

25.238 NetworkedMultiplayerENet

Inherits: *NetworkedMultiplayerPeer < PacketPeer < Reference < Object*

Category: Core

25.238.1 Brief Description

PacketPeer implementation using the ENet library.

25.238.2 Member Functions

void	<i>close_connection ()</i>
<i>int</i>	<i>create_client (String ip, int port, int in_bandwidth=0, int out_bandwidth=0)</i>
<i>int</i>	<i>create_server (int port, int max_clients=32, int in_bandwidth=0, int out_bandwidth=0)</i>
void	<i>set_bind_ip (String ip)</i>

25.238.3 Member Variables

- *CompressionMode* **compression_mode**

25.238.4 Enums

enum **CompressionMode**

- **COMPRESS_NONE = 0**
- **COMPRESS_RANGE_CODER = 1**
- **COMPRESS_FASTLZ = 2**
- **COMPRESS_ZLIB = 3**
- **COMPRESS_ZSTD = 4**

25.238.5 Description

A connection (or a listening server) that should be passed to SceneTree.set_network_peer. Socket events can be handled by connecting to *SceneTree* signals.

25.238.6 Member Function Description

- void **close_connection ()**
- int **create_client (String ip, int port, int in_bandwidth=0, int out_bandwidth=0)**

Create client that connects to a server at address *ip* using specified *port*.

- int **create_server (int port, int max_clients=32, int in_bandwidth=0, int out_bandwidth=0)**

Create server that listens to connections via *port*.

- void **set_bind_ip (String ip)**

25.239 NetworkedMultiplayerPeer

Inherits: *PacketPeer < Reference < Object*

Inherited By: *NetworkedMultiplayerENet*

Category: Core

25.239.1 Brief Description

A high-level network interface to simplify multiplayer interactions.

25.239.2 Member Functions

int	<i>get_connection_status () const</i>
int	<i>get_packet_peer () const</i>
int	<i>get_unique_id () const</i>
void	<i>poll ()</i>
void	<i>set_target_peer (int id)</i>

25.239.3 Signals

- **connection_failed ()**

Emitted when a connection attempt fails.

- **connection_succeeded ()**

Emitted when a connection attempt succeeds.

- **peer_connected (int id)**

Emitted by the server when a client connects.

- **peer_disconnected (int id)**

Emitted by the server when a client disconnects.

- `server_disconnected ()`

Emitted by clients when the server disconnects.

25.239.4 Member Variables

- `bool refuse_new_connections` - If true this NetworkedMultiplayerPeer refuses new connections. Default value: false.
- `TransferMode transfer_mode` - The manner in which to send packets to the target_peer. See enum TransferMode.

25.239.5 Numeric Constants

- `TARGET_PEER_BROADCAST = 0` — Packets are sent to the server and then redistributed to other peers.
- `TARGET_PEER_SERVER = 1` — Packets are sent to the server alone.

25.239.6 Enums

enum `TransferMode`

- `TRANSFER_MODE_UNRELIABLE = 0` — Packets are sent via unordered UDP packets.
- `TRANSFER_MODE_UNRELIABLE_ORDERED = 1` — Packets are sent via ordered UDP packets.
- `TRANSFER_MODE_RELIABLE = 2` — Packets are sent via TCP packets.

enum `ConnectionStatus`

- `CONNECTION_DISCONNECTED = 0` — The ongoing connection disconnected.
- `CONNECTION_CONNECTING = 1` — A connection attempt is ongoing.
- `CONNECTION_CONNECTED = 2` — The connection attempt succeeded.

25.239.7 Description

Manages the connection to network peers. Assigns unique IDs to each client connected to the server.

25.239.8 Member Function Description

- `int get_connection_status () const`

Returns the current state of the connection. See enum ConnectionStatus.

- `int get_packet_peer () const`

Returns the ID of the NetworkedMultiplayerPeer who sent the most recent packet.

- `int get_unique_id () const`

Returns the ID of this NetworkedMultiplayerPeer.

- void **poll** ()

Waits up to 1 second to receive a new network event.

- void **set_target_peer** (*int id*)

The peer to which packets will be sent. Default value: 0.

25.240 Nil

Category: Built-In Types

25.240.1 Brief Description

25.240.2 Member Functions

void	Nil (<i>PoolColorArray</i> from)
void	Nil (<i>PoolVector3Array</i> from)
void	Nil (<i>PoolVector2Array</i> from)
void	Nil (<i>PoolStringArray</i> from)
void	Nil (<i>PoolRealArray</i> from)
void	Nil (<i>PoolIntArray</i> from)
void	Nil (<i>PoolByteArray</i> from)
void	Nil (<i>Array</i> from)
void	Nil (<i>Dictionary</i> from)
void	Nil (<i>Object</i> from)
void	Nil (<i>RID</i> from)
void	Nil (<i>NodePath</i> from)
void	Nil (<i>Color</i> from)
void	Nil (<i>Transform</i> from)
void	Nil (<i>Basis</i> from)
void	Nil (<i>AABB</i> from)
void	Nil (<i>Quat</i> from)
void	Nil (<i>Plane</i> from)
void	Nil (<i>Transform2D</i> from)
void	Nil (<i>Vector3</i> from)
void	Nil (<i>Rect2</i> from)
void	Nil (<i>Vector2</i> from)
void	Nil (<i>String</i> from)
void	Nil (<i>float</i> from)
void	Nil (<i>int</i> from)
void	Nil (<i>bool</i> from)

25.240.3 Member Function Description

- void **Nil** (*PoolColorArray* from)
- void **Nil** (*PoolVector3Array* from)
- void **Nil** (*PoolVector2Array* from)

- void **Nil** (*PoolStringArray* from)
- void **Nil** (*PoolRealArray* from)
- void **Nil** (*PoolIntArray* from)
- void **Nil** (*PoolByteArray* from)
- void **Nil** (*Array* from)
- void **Nil** (*Dictionary* from)
- void **Nil** (*Object* from)
- void **Nil** (*RID* from)
- void **Nil** (*NodePath* from)
- void **Nil** (*Color* from)
- void **Nil** (*Transform* from)
- void **Nil** (*Basis* from)
- void **Nil** (*AABB* from)
- void **Nil** (*Quat* from)
- void **Nil** (*Plane* from)
- void **Nil** (*Transform2D* from)
- void **Nil** (*Vector3* from)
- void **Nil** (*Rect2* from)
- void **Nil** (*Vector2* from)
- void **Nil** (*String* from)
- void **Nil** (*float* from)
- void **Nil** (*int* from)
- void **Nil** (*bool* from)

25.241 NinePatchRect

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

25.241.1 Brief Description

Scalable texture-based frame that tiles the texture's centers and sides, but keeps the corners' original size. Perfect for panels and dialog boxes.

25.241.2 Signals

- **texture_changed** ()

Fired when the node's texture changes.

25.241.3 Member Variables

- *AxisStretchMode* **axis_stretch_horizontal** - Doesn't do anything at the time of writing.
- *AxisStretchMode* **axis_stretch_vertical** - Doesn't do anything at the time of writing.
- *bool* **draw_center** - If `true`, draw the panel's center. Else, only draw the 9-slice's borders. Default value: `true`
- *int* **patch_margin_bottom** - The height of the 9-slice's bottom row. A margin of 16 means the 9-slice's bottom corners and side will have a height of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.
- *int* **patch_margin_left** - The height of the 9-slice's left column.
- *int* **patch_margin_right** - The height of the 9-slice's right column.
- *int* **patch_margin_top** - The height of the 9-slice's top row.
- *Rect2* **region_rect** - Rectangular region of the texture to sample from. If you're working with an atlas, use this property to define the area the 9-slice should use. All other properties are relative to this one.
- *Texture* **texture** - The node's texture resource.

25.241.4 Enums

enum AxisStretchMode

- **AXIS_STRETCH_MODE_STRETCH = 0** — Doesn't do anything at the time of writing. Default value for `axis_stretch_horizontal` and `axis_stretch_vertical`.
- **AXIS_STRETCH_MODE_TILE = 1** — Doesn't do anything at the time of writing.
- **AXIS_STRETCH_MODE_TILE_FIT = 2** — Doesn't do anything at the time of writing.

25.241.5 Description

Better known as 9-slice panels, NinePatchRect produces clean panels of any size, based on a small texture. To do so, it splits the texture in a 3 by 3 grid. When you scale the node, it tiles the texture's sides horizontally or vertically, the center on both axes but it doesn't scale or tile the corners.

25.242 Node

Inherits: *Object*

Inherited By: *Viewport, AudioStreamPlayer, EditorFileSystem, CanvasLayer, Spatial, AnimationPlayer, EditorPlugin, ResourcePreloader, AnimationTreePlayer, WorldEnvironment, InstancePlaceholder, HTTPRequest, EditorInterface, EditorResourcePreview, CanvasItem, Timer, Tween*

Category: Core

25.242.1 Brief Description

Base class for all *scene* objects.

25.242.2 Member Functions

void	<code>_enter_tree ()</code> virtual
void	<code>_exit_tree ()</code> virtual
void	<code>_input (InputEvent event)</code> virtual
void	<code>_physics_process (float delta)</code> virtual
void	<code>_process (float delta)</code> virtual
void	<code>_ready ()</code> virtual
void	<code>_unhandled_input (InputEvent event)</code> virtual
void	<code>_unhandled_key_input (InputEventKey event)</code> virtual
void	<code>add_child (Node node, bool legible_unique_name=false)</code>
void	<code>add_child_below_node (Node node, Node child_node, bool legible_unique_name=false)</code>
void	<code>add_to_group (String group, bool persistent=false)</code>
bool	<code>can_process ()</code> const
Node	<code>duplicate (int flags=15)</code> const
Node	<code>find_node (String mask, bool recursive=true, bool owned=true)</code> const
Node	<code>get_child (int idx)</code> const
int	<code>get_child_count ()</code> const
Array	<code>get_children ()</code> const
Array	<code>get_groups ()</code> const
int	<code>get_index ()</code> const
int	<code>get_network_master ()</code> const
Node	<code>get_node (NodePath path)</code> const
Array	<code>get_node_and_resource (NodePath path)</code>
Node	<code>get_parent ()</code> const
NodePath	<code>get_path ()</code> const
NodePath	<code>get_path_to (Node node)</code> const
float	<code>get_physics_process_delta_time ()</code> const
int	<code>get_position_in_parent ()</code> const
float	<code>get_process_delta_time ()</code> const
bool	<code>get_scene_instance_load_placeholder ()</code> const
SceneTree	<code>get_tree ()</code> const
Viewport	<code>get_viewport ()</code> const
bool	<code>has_node (NodePath path)</code> const
bool	<code>has_node_and_resource (NodePath path)</code> const
bool	<code>is_a_parent_of (Node node)</code> const
bool	<code>is_displayed_folded ()</code> const
bool	<code>is_greater_than (Node node)</code> const
bool	<code>is_in_group (String group)</code> const
bool	<code>is_inside_tree ()</code> const
bool	<code>is_network_master ()</code> const
bool	<code>is_physics_processing ()</code> const
bool	<code>is_physics_processing_internal ()</code> const
bool	<code>is_processing ()</code> const
bool	<code>is_processing_input ()</code> const
bool	<code>is_processing_internal ()</code> const

Continued on next page

Table 25.14 – continued from previous page

<code>bool</code>	<code>is_processingUnhandledInput() const</code>
<code>bool</code>	<code>is_processingUnhandledKeyInput() const</code>
<code>void</code>	<code>moveChild(Node child_node, int to_position)</code>
<code>void</code>	<code>printStrayNodes()</code>
<code>void</code>	<code>printTree()</code>
<code>void</code>	<code>propagateCall(String method, Array args=[], bool parentFirst=false)</code>
<code>void</code>	<code>propagateNotification(int what)</code>
<code>void</code>	<code>queueFree()</code>
<code>void</code>	<code>raise()</code>
<code>void</code>	<code>removeAndSkip()</code>
<code>void</code>	<code>removeChild(Node node)</code>
<code>void</code>	<code>removeFromGroup(String group)</code>
<code>void</code>	<code>replaceBy(Node node, bool keepData=false)</code>
<code>void</code>	<code>requestReady()</code>
<code>Variant</code>	<code>rpc(String method) vararg</code>
<code>void</code>	<code>rpcConfig(String method, int mode)</code>
<code>Variant</code>	<code>rpcId(int peer_id, String method) vararg</code>
<code>Variant</code>	<code>rpcUnreliable(String method) vararg</code>
<code>Variant</code>	<code>rpcUnreliableId(int peer_id, String method) vararg</code>
<code>void</code>	<code>rset(String property, Variant value)</code>
<code>void</code>	<code>rsetConfig(String property, int mode)</code>
<code>void</code>	<code>rsetId(int peer_id, String property, Variant value)</code>
<code>void</code>	<code>rsetUnreliable(String property, Variant value)</code>
<code>void</code>	<code>rsetUnreliableId(int peer_id, String property, Variant value)</code>
<code>void</code>	<code>setDisplayFolded(bool fold)</code>
<code>void</code>	<code>setNetworkMaster(int id, bool recursive=true)</code>
<code>void</code>	<code>setPhysicsProcess(bool enable)</code>
<code>void</code>	<code>setPhysicsProcessInternal(bool enable)</code>
<code>void</code>	<code>setProcess(bool enable)</code>
<code>void</code>	<code>setProcessInput(bool enable)</code>
<code>void</code>	<code>setProcessInternal(bool enable)</code>
<code>void</code>	<code>setProcessUnhandledInput(bool enable)</code>
<code>void</code>	<code>setProcessUnhandledKeyInput(bool enable)</code>
<code>void</code>	<code>setSceneInstanceLoadPlaceholder(bool loadPlaceholder)</code>

25.242.3 Signals

- **renamed()**

Emitted when the node is renamed.

- **tree_entered()**

Emitted when the node enters the tree.

- **tree_exited()**

Emitted after the node exits the tree and is no longer active.

- **tree_exiting()**

Emitted when the node is still active but about to exit the tree. This is the right place for de-initialization.

25.242.4 Member Variables

- *String filename* - When a scene is instanced from a file, its topmost node contains the filename from which it was loaded.
- *String name* - The name of the node. This name is unique among the siblings (other child nodes from the same parent).

When set to an existing name, the node will be automatically renamed

- *Node owner* - The node owner. A node can have any other node as owner (as long as it is a valid parent, grandparent, etc. ascending in the tree). When saving a node (using SceneSaver) all the nodes it owns will be saved with it. This allows for the creation of complex *SceneTrees*, with instancing and subinstancing.
- *PauseMode pause_mode*

25.242.5 Numeric Constants

- **NOTIFICATION_ENTER_TREE = 10** — Notification received when the node enters a *SceneTree*.
- **NOTIFICATION_EXIT_TREE = 11** — Notification received when the node is about to exit a *SceneTree*.
- **NOTIFICATION_MOVED_IN_PARENT = 12** — Notification received when the node is moved in the parent.
- **NOTIFICATION_READY = 13** — Notification received when the node is ready. See *_ready*.
- **NOTIFICATION_PAUSED = 14** — Notification received when the node is paused.
- **NOTIFICATION_UNPAUSED = 15** — Notification received when the node is unpause.
- **NOTIFICATION_PHYSICS_PROCESS = 16** — Notification received every frame when the physics process flag is set (see *set_physics_process*).
- **NOTIFICATION_PROCESS = 17** — Notification received every frame when the process flag is set (see *set_process*).
- **NOTIFICATION_PARENTED = 18** — Notification received when a node is set as a child of another node. Note that this doesn't mean that a node entered the Scene Tree.
- **NOTIFICATION_UNPARENTED = 19** — Notification received when a node is unparented (parent removed it from the list of children).
- **NOTIFICATION_INSTANCED = 20** — Notification received when the node is instanced.
- **NOTIFICATION_DRAG_BEGIN = 21** — Notification received when a drag begins.
- **NOTIFICATION_DRAG_END = 22** — Notification received when a drag ends.
- **NOTIFICATION_PATH_CHANGED = 23** — Notification received when the node's *NodePath* changed.
- **NOTIFICATION_TRANSLATION_CHANGED = 24**
- **NOTIFICATION_INTERNAL_PROCESS = 25**
- **NOTIFICATION_INTERNAL_PHYSICS_PROCESS = 26**

25.242.6 Enums

enum **PauseMode**

- **PAUSE_MODE_INHERIT = 0** — Inherits pause mode from parent. For root node, it is equivalent to **PAUSE_MODE_STOP**.
- **PAUSE_MODE_STOP = 1** — Stop processing when SceneTree is paused.
- **PAUSE_MODE_PROCESS = 2** — Continue to process regardless of SceneTree pause state.

enum **RPCMode**

- **RPC_MODE_DISABLED = 0**
- **RPC_MODE_REMOTE = 1** — Call a method remotely.
- **RPC_MODE_SYNC = 2** — Call a method both remotely and locally.
- **RPC_MODE_MASTER = 3** — Call a method if the Node is Master.
- **RPC_MODE_SLAVE = 4** — Call a method if the Node is Slave.

enum **DuplicateFlags**

- **DUPLICATE_SIGNALS = 1** — Duplicate the node's signals.
- **DUPLICATE_GROUPS = 2** — Duplicate the node's groups.
- **DUPLICATE_SCRIPTS = 4** — Duplicate the node's scripts.
- **DUPLICATE_USE_INSTANCING = 8** — Duplicate using instancing.

25.242.7 Description

Nodes are Godot's building blocks. They can be assigned as the child of another node, resulting in a tree arrangement. A given node can contain any number of nodes as children with the requirement that all siblings (direct children of a node) should have unique names.

A tree of nodes is called a *scene*. Scenes can be saved to the disk and then instanced into other scenes. This allows for very high flexibility in the architecture and data model of Godot projects. Nodes can also optionally be added to groups. This makes it possible to access a number of nodes from code (an "enemies" group, for example) to perform grouped actions.

Scene tree: The *SceneTree* contains the active tree of nodes. When a node is added to the scene tree, it receives the **NOTIFICATION_ENTER_TREE** notification and its *_enter_tree* callback is triggered. Child nodes are always added *after* their parent node, i.e. the *_enter_tree* callback of a parent node will be triggered before its child's.

Once all nodes have been added in the scene tree, they receive the **NOTIFICATION_READY** notification and their respective *_ready* callbacks are triggered. For groups of nodes, the *_ready* callback is called in reverse order, starting with the children and moving up to the parent nodes.

This means that when adding a node to the scene tree, the following order will be used for the callbacks: *_enter_tree* of the parent, *_enter_tree* of the children, *_ready* of the children and finally *_ready* of the parent (recursively for the entire scene tree).

Processing: Nodes can override the "process" state, so that they receive a callback on each frame requesting them to process (do something). Normal processing (callback *_process*, toggled with *set_process*) happens as fast as possible and is dependent on the frame rate, so the processing time *delta* is passed as an argument. Physics processing (callback *_physics_process*, toggled with *set_physics_process*) happens a fixed number of times per second (60 by default) and is useful for code related to the physics engine.

Nodes can also process input events. When present, the `_input` function will be called for each input that the program receives. In many cases, this can be overkill (unless used for simple projects), and the `_unhandled_input` function might be preferred; it is called when the input event was not handled by anyone else (typically, GUI *Control* nodes), ensuring that the node only receives the events that were meant for it.

To keep track of the scene hierarchy (especially when instancing scenes into other scenes), an “owner” can be set for the node with `set_owner`. This keeps track of who instanced what. This is mostly useful when writing editors and tools, though.

Finally, when a node is freed with `free` or `queue_free`, it will also free all its children.

Networking with nodes: After connecting to a server (or making one, see *NetworkedMultiplayerENet*) it is possible to use the built-in RPC (remote procedure call) system to communicate over the network. By calling `rpc` with a method name, it will be called locally and in all connected peers (peers = clients and the server that accepts connections), with behaviour varying depending on the network mode (`set_network_mode`) of the receiving peer. To identify which node receives the RPC call Godot will use its *NodePath* (make sure node names are the same on all peers).

25.242.8 Member Function Description

- `void _enter_tree () virtual`

Called when the node enters the *SceneTree* (e.g. upon instancing, scene changing, or after calling `add_child` in a script). If the node has children, its `_enter_tree` callback will be called first, and then that of the children.

Corresponds to the NOTIFICATION_ENTER_TREE notification in *Object._notification*.

- `void _exit_tree () virtual`

Called when the node is about to leave the *SceneTree* (e.g. upon freeing, scene changing, or after calling `remove_child` in a script). If the node has children, its `_exit_tree` callback will be called last, after all its children have left the tree.

Corresponds to the NOTIFICATION_EXIT_TREE notification in *Object._notification* and signal `tree_exiting`. To get notified when the node has already left the active tree, connect to the `tree_exited`

- `void _input (InputEvent event) virtual`

Called when there is an input event. The input event propagates through the node tree until a node consumes it.

It is only called if input processing is enabled, which is done automatically if this method is overridden, and can be toggled with `set_process_input`.

- `void _physics_process (float delta) virtual`

Called during the physics processing step of the main loop. Physics processing means that the frame rate is synced to the physics, i.e. the `delta` variable should be constant.

It is only called if physics processing is enabled, which is done automatically if this method is overridden, and can be toggled with `set_physics_process`.

Corresponds to the NOTIFICATION_PHYSICS_PROCESS notification in *Object._notification*.

- `void _process (float delta) virtual`

Called during the processing step of the main loop. Processing happens at every frame and as fast as possible, so the `delta` time since the previous frame is not constant.

It is only called if processing is enabled, which is done automatically if this method is overridden, and can be toggled with `set_process`.

Corresponds to the NOTIFICATION_PROCESS notification in *Object._notification*.

- `void _ready () virtual`

Called when the node is “ready”, i.e. when both the node and its children have entered the scene tree. If the node has children, their `_ready` callbacks get triggered first, and the parent node will receive the ready notification afterwards.

Corresponds to the NOTIFICATION_READY notification in *Object._notification*.

- `void _unhandled_input (InputEvent event) virtual`

Propagated to all nodes when the previous InputEvent is not consumed by any nodes.

It is only called if unhandled input processing is enabled, which is done automatically if this method is overridden, and can be toggled with `set_process_unhandled_input`.

- `void _unhandled_key_input (InputEventKey event) virtual`
- `void add_child (Node node, bool legible_unique_name=false)`

Adds a child node. Nodes can have any number of children, but every child must have a unique name. Child nodes are automatically deleted when the parent node is deleted, so an entire scene can be removed by deleting its topmost node.

Setting “legible_unique_name” `true` creates child nodes with human-readable names, based on the name of the node being instanced instead of its type.

- `void add_child_below_node (Node node, Node child_node, bool legible_unique_name=false)`

Adds a child node. The child is placed below the given node in the list of children.

Setting “legible_unique_name” `true` creates child nodes with human-readable names, based on the name of the node being instanced instead of its type.

- `void add_to_group (String group, bool persistent=false)`

Adds the node to a group. Groups are helpers to name and organize a subset of nodes, for example “enemies” or “collectables”. A node can be in any number of groups. Nodes can be assigned a group at any time, but will not be added until they are inside the scene tree (see `is_inside_tree`).

- `bool can_process () const`

Returns `true` if the node can process while the scene tree is paused (see `set_pause_mode`). Always returns `true` if the scene tree is not paused, and `false` if the node is not in the tree. **FIXME:** Why FAIL_COND?

- `Node duplicate (int flags=15) const`

Duplicates the node, returning a new node.

You can fine-tune the behavior using the `flags`. See DUPLICATE_* constants.

- `Node find_node (String mask, bool recursive=true, bool owned=true) const`

Finds a descendant of this node whose name matches `mask` as in `String.match` (i.e. case sensitive, but “*” matches zero or more characters and ‘?’ matches any single character except ‘.’). Note that it does not match against the full path, just against individual node names.

- `Node get_child (int idx) const`

Returns a child node by its index (see `get_child_count`). This method is often used for iterating all children of a node.

- `int get_child_count () const`

Returns the number of child nodes.

- `Array get_children () const`

Returns an array of references to node’s children.

- `Array get_groups () const`

Returns an array listing the groups that the node is a member of.

- `int get_index () const`

Returns the node's index, i.e. its position among the siblings of its parent.

- `int get_network_master () const`

Returns the peer ID of the network master for this node.

- `Node get_node (NodePath path) const`

Fetches a node. The `NodePath` can be either a relative path (from the current node) or an absolute path (in the scene tree) to a node. If the path does not exist, a `null` instance is returned and attempts to access it will result in an “Attempt to call <method> on a null instance.” error.

Note: fetching absolute paths only works when the node is inside the scene tree (see `is_inside_tree`).

Example: Assume your current node is Character and the following tree:

```
/root
/root/Character
/root/Character/Sword
/root/Character/Backpack/Dagger
/root/MyGame
/root/Swamp/Alligator
/root/Swamp/Mosquito
/root/Swamp/Goblin
```

Possible paths are:

```
get_node ("Sword")
get_node ("Backpack/Dagger")
get_node ("../Swamp/Alligator")
get_node ("/root/MyGame")
```

- `Array get_node_and_resource (NodePath path)`

- `Node get_parent () const`

Returns the parent node of the current node, or an empty `Node` if the node lacks a parent.

- `NodePath get_path () const`

Returns the absolute path of the current node. This only works if the current node is inside the scene tree (see `is_inside_tree`).

- `NodePath get_path_to (Node node) const`

Returns the relative path from the current node to the specified node in “node” argument. Both nodes must be in the same scene, or the function will fail.

- `float get_physics_process_delta_time () const`

Returns the time elapsed since the last physics-bound frame (see `_physics_process`). This is always a constant value in physics processing unless the frames per second is changed in OS.

- `int get_position_in_parent () const`

Returns the node's order in the scene tree branch. For example, if called on the first child node the position is 0.

- `float get_process_delta_time () const`

Returns the time elapsed (in seconds) since the last process callback. This value may vary from frame to frame.

- `bool get_scene_instance_load_placeholder () const`

- `SceneTree get_tree () const`

Returns the *SceneTree* that contains this node.

- *Viewport* **get_viewport()** const

Returns the node's *Viewport*.

- *bool* **has_node(NodePath path)** const

Returns `true` if the node that the *NodePath* points to exists.

- *bool* **has_node_and_resource(NodePath path)** const
- *bool* **is_a_parent_of(Node node)** const

Returns `true` if the given node is a direct or indirect child of the current node.

- *bool* **is_displayed_folded()** const

Returns `true` if the node is folded (collapsed) in the Scene dock.

- *bool* **is_greater_than(Node node)** const

Returns `true` if the given node occurs later in the scene hierarchy than the current node.

- *bool* **is_in_group(String group)** const

Returns `true` if this node is in the specified group.

- *bool* **is_inside_tree()** const

Returns `true` if this node is currently inside a *SceneTree*.

- *bool* **is_network_master()** const
- *bool* **is_physics_processing()** const

Returns `true` if physics processing is enabled (see *set_physics_process*).

- *bool* **is_physics_processing_internal()** const
- *bool* **is_processing()** const

Returns `true` if processing is enabled (see *set_process*).

- *bool* **is_processing_input()** const

Returns `true` if the node is processing input (see *set_process_input*).

- *bool* **is_processing_internal()** const
- *bool* **is_processingUnhandledInput()** const

Returns `true` if the node is processing unhandled input (see *set_process_unhandled_input*).

- *bool* **is_processingUnhandledKeyInput()** const

Returns `true` if the node is processing unhandled key input (see *set_process_unhandled_key_input*).

- *void* **move_child(Node child_node, int to_position)**

Moves a child node to a different position (order) amongst the other children. Since calls, signals, etc are performed by tree order, changing the order of children nodes may be useful.

- *void* **print_stray_nodes()**
- *void* **print_tree()**

Prints the scene to stdout. Used mainly for debugging purposes.

- *void* **propagate_call(String method, Array args=[], bool parent_first=false)**

Calls the given method (if present) with the arguments given in `args` on this node and recursively on all its children. If the `parent_first` argument is `true` then the method will be called on the current node first, then on all children. If it is `false` then the children will be called first.

- `void propagate_notification (int what)`

Notifies the current node and all its children recursively by calling `notification()` on all of them.

- `void queue_free ()`

Queues a node for deletion at the end of the current frame. When deleted, all of its child nodes will be deleted as well. This method ensures it's safe to delete the node, contrary to `Object.free`. Use `Object.is_queued_for_deletion` to check whether a node will be deleted at the end of the frame.

- `void raise ()`

Moves this node to the top of the array of nodes of the parent node. This is often useful in GUIs (*Control* nodes), because their order of drawing depends on their order in the tree.

- `void remove_and_skip ()`

Removes a node and sets all its children as children of the parent node (if it exists). All event subscriptions that pass by the removed node will be unsubscribed.

- `void remove_child (Node node)`

Removes a child node. The node is NOT deleted and must be deleted manually.

- `void remove_from_group (String group)`

Removes a node from a group.

- `void replace_by (Node node, bool keep_data=false)`

Replaces a node in a scene by the given one. Subscriptions that pass through this node will be lost.

- `void request_ready ()`

Requests that `_ready` be called again.

- `Variant rpc (String method) vararg`

Sends a remote procedure call request to all peers on the network (and locally), optionally sending additional data as arguments. Call request will be received by nodes with the same `NodePath`.

- `void rpc_config (String method, int mode)`

Changes the method's RPC mode (one of `RPC_MODE_*` constants).

- `Variant rpc_id (int peer_id, String method) vararg`

Sends a `rpc` to a specific peer identified by `peer_id`.

- `Variant rpc_unreliable (String method) vararg`

Sends a `rpc` using an unreliable protocol.

- `Variant rpc_unreliable_id (int peer_id, String method) vararg`

Sends a `rpc` to a specific peer identified by `peer_id` using an unreliable protocol.

- `void rset (String property, Variant value)`

Remotely changes property's value on other peers (and locally).

- `void rset_config (String property, int mode)`

Changes the property's RPC mode (one of `RPC_MODE_*` constants).

- `void rset_id (int peer_id, String property, Variant value)`

Remotely changes property's value on a specific peer identified by *peer_id*.

- `void rset_unreliable (String property, Variant value)`

Remotely changes property's value on other peers (and locally) using an unreliable protocol.

- `void rset_unreliable_id (int peer_id, String property, Variant value)`

Remotely changes property's value on a specific peer identified by *peer_id* using an unreliable protocol.

- `void set_display_folded (bool fold)`

Sets the folded state of the node in the Scene dock.

- `void set_network_master (int id, bool recursive=true)`

Sets the node network master to the peer with the given peer ID. The network master is the peer that has authority over it on the network. Inherited from the parent node by default, which ultimately defaults to peer ID 1 (the server).

- `void set_physics_process (bool enable)`

Enables or disables physics (i.e. fixed framerate) processing. When a node is being processed, it will receive a NOTIFICATION_PHYSICS_PROCESS at a fixed (usually 60 fps, see *OS* to change) interval (and the *_physics_process* callback will be called if exists). Enabled automatically if *_physics_process* is overridden. Any calls to this before *_ready* will be ignored.

- `void set_physics_process_internal (bool enable)`

- `void set_process (bool enable)`

Enables or disables processing. When a node is being processed, it will receive a NOTIFICATION_PROCESS on every drawn frame (and the *_process* callback will be called if exists). Enabled automatically if *_process* is overridden. Any calls to this before *_ready* will be ignored.

- `void set_process_input (bool enable)`

Enables or disables input processing. This is not required for GUI controls! Enabled automatically if *_input* is overridden. Any calls to this before *_ready* will be ignored.

- `void set_process_internal (bool enable)`

- `void set_processUnhandledInput (bool enable)`

Enables unhandled input processing. This is not required for GUI controls! It enables the node to receive all input that was not previously handled (usually by a *Control*). Enabled automatically if *_unhandled_input* is overridden. Any calls to this before *_ready* will be ignored.

- `void setProcessUnhandledKeyInput (bool enable)`

Enables unhandled key input processing. Enabled automatically if *_unhandled_key_input* is overridden. Any calls to this before *_ready* will be ignored.

- `void setSceneInstanceLoadPlaceholder (bool load_placeholder)`

25.243 Node2D

Inherits: *CanvasItem < Node < Object*

Inherited By: *RemoteTransform2D, Joint2D, VisibilityNotifier2D, Navigation2D, CollisionPolygon2D, TouchScreenButton, Particles2D, AnimatedSprite, RayCast2D, Light2D, Path2D, Line2D, AudioStreamPlayer2D, CanvasModulate, Sprite, CollisionShape2D, NavigationPolygonInstance, PathFollow2D, ParallaxLayer, Polygon2D, Position2D, LightOccluder2D, CollisionObject2D, BackBufferCopy, YSort, TileMap, Camera2D*

Category: Core

25.243.1 Brief Description

A 2D game object, parent of all 2D related nodes. Has a position, rotation, scale and Z-index.

25.243.2 Member Functions

void	<i>apply_scale (Vector2 ratio)</i>
float	<i>get_angle_to (Vector2 point) const</i>
Transform2D	<i>get_relative_transform_to_parent (Node parent) const</i>
void	<i>global_translate (Vector2 offset)</i>
void	<i>look_at (Vector2 point)</i>
void	<i>move_local_x (float delta, bool scaled=false)</i>
void	<i>move_local_y (float delta, bool scaled=false)</i>
void	<i>rotate (float radians)</i>
Vector2	<i>to_global (Vector2 local_point) const</i>
Vector2	<i>to_local (Vector2 global_point) const</i>
void	<i>translate (Vector2 offset)</i>

25.243.3 Member Variables

- **Vector2 global_position** - Global position.
- **float global_rotation** - Global rotation in radians.
- **float global_rotation_degrees** - Global rotation in degrees.
- **Vector2 global_scale** - Global scale.
- **Transform2D global_transform** - Global *Transform2D*.
- **Vector2 position** - Position, relative to the node's parent.
- **float rotation** - Rotation in radians, relative to the node's parent.
- **float rotation_degrees** - Rotation in degrees, relative to the node's parent.
- **Vector2 scale** - The node's scale. Unscaled value: (1, 1)
- **Transform2D transform** - Local *Transform2D*.
- **bool z_as_relative** - If true the node's Z-index is relative to its parent's Z-index. If this node's Z-index is 2 and its parent's effective Z-index is 3, then this node's effective Z-index will be 2 + 3 = 5.
- **int z_index** - Z-index. Controls the order in which the nodes render. A node with a higher Z-index will display in front of others.

25.243.4 Description

A 2D game object, with a position, rotation and scale. All 2D physics nodes and sprites inherit from Node2D. Use Node2D as a parent node to move, scale and rotate children in a 2D project. Also gives control on the node's render order.

25.243.5 Member Function Description

- `void apply_scale (Vector2 ratio)`

Multiplies the current scale by the ‘ratio’ vector.

- `float get_angle_to (Vector2 point) const`

Returns the angle between the node and the ‘point’ in radians.

- `Transform2D get_relative_transform_to_parent (Node parent) const`

Returns the *Transform2D* relative to this node’s parent.

- `void global_translate (Vector2 offset)`

Adds the ‘offset’ vector to the node’s global position.

- `void look_at (Vector2 point)`

Rotates the node so it points towards the ‘point’.

- `void move_local_x (float delta, bool scaled=false)`

Applies a local translation on the node’s X axis based on the *Node._process*’s `delta`. If `scaled` is false, normalizes the movement.

- `void move_local_y (float delta, bool scaled=false)`

Applies a local translation on the node’s Y axis based on the *Node._process*’s `delta`. If `scaled` is false, normalizes the movement.

- `void rotate (float radians)`

Applies a rotation to the node, in radians, starting from its current rotation.

- `Vector2 to_global (Vector2 local_point) const`

Converts a local point’s coordinates to global coordinates.

- `Vector2 to_local (Vector2 global_point) const`

Converts a global point’s coordinates to local coordinates.

- `void translate (Vector2 offset)`

Translates the node by the given `offset` in local coordinates.

25.244 NodePath

Category: Built-In Types

25.244.1 Brief Description

Pre-parsed scene tree path.

25.244.2 Member Functions

<i>NodePath</i>	<i>NodePath (String from)</i>
<i>NodePath</i>	<i>get_as_property_path ()</i>
<i>String</i>	<i>get_concatenated_subnames ()</i>
<i>String</i>	<i>get_name (int idx)</i>
<i>int</i>	<i>get_name_count ()</i>
<i>String</i>	<i>get_subname (int idx)</i>
<i>int</i>	<i>get_subname_count ()</i>
<i>bool</i>	<i>is_absolute ()</i>
<i>bool</i>	<i>is_empty ()</i>

25.244.3 Description

A pre-parsed relative or absolute path in a scene tree, for use with *Node.get_node* and similar functions. It can reference a node, a resource within a node, or a property of a node or resource. For instance, "Path2D/PathFollow2D/Sprite:texture:size" would refer to the size property of the texture resource on the node named "Sprite" which is a child of the other named nodes in the path. Note that if you want to get a resource, you must end the path with a colon, otherwise the last element will be used as a property name.

You will usually just pass a string to *Node.get_node* and it will be automatically converted, but you may occasionally want to parse a path ahead of time with *NodePath* or the literal syntax @ "path". Exporting a *NodePath* variable will give you a node selection widget in the properties panel of the editor, which can often be useful.

A *NodePath* is made up of a list of node names, a list of "subnode" (resource) names, and the name of a property in the final node or resource.

25.244.4 Member Function Description

- *NodePath NodePath (String from)*

Create a *NodePath* from a string, e.g. "Path2D/PathFollow2D/Sprite:texture:size". A path is absolute if it starts with a slash. Absolute paths are only valid in the global scene tree, not within individual scenes. In a relative path, ". ." and "... ." indicate the current node and its parent.

- *NodePath get_as_property_path ()*
- *String get_concatenated_subnames ()*
- *String get_name (int idx)*

Get the node name indicated by *idx* (0 to *get_name_count*)

- *int get_name_count ()*

Get the number of node names which make up the path.

- *String get_subname (int idx)*

Get the resource name indicated by *idx* (0 to *get_subname_count*)

- *int get_subname_count ()*

Get the number of resource names in the path.

- *bool is_absolute ()*

Return true if the node path is absolute (not relative).

- `bool is_empty()`

Return true if the node path is empty.

25.245 Object

Inherited By: Reference, Physics2DServer, MainLoop, Input, Node, Geometry, ARVRPositionalTracker, TreeItem, PhysicsDirectBodyState, JavaScript, ARVRServer, PhysicsDirectSpaceState, Engine, Physics2DDirectSpaceState, InputMap, UndoRedo, PhysicsServer, ProjectSettings, ResourceSaver, Performance, ResourceLoader, JSON, AudioServer, IP, ClassDB, VisualServer, OS, GodotSharp, EditorSelection, Physics2DDirectBodyState, VisualScriptEditor, TranslationServer, EditorFileSystemDirectory

Category: Core

25.245.1 Brief Description

Base class for all non built-in types.

25.245.2 Member Functions

void	<code>_get (String property) virtual</code>
Array	<code>_get_property_list () virtual</code>
void	<code>_init () virtual</code>
void	<code>_notification (int what) virtual</code>
bool	<code>_set (String property, Variant value) virtual</code>
void	<code>add_user_signal (String signal, Array arguments=[])</code>
Variant	<code>call (String method) vararg</code>
Variant	<code>call_deferred (String method) vararg</code>
Variant	<code>callv (String method, Array arg_array)</code>
bool	<code>can_translate_messages () const</code>
int	<code>connect (String signal, Object target, String method, Array binds=[], int flags=0)</code>
void	<code>disconnect (String signal, Object target, String method)</code>
Variant	<code>emit_signal (String signal) vararg</code>
void	<code>free ()</code>
Variant	<code>get (String property) const</code>
String	<code>get_class () const</code>
Array	<code>get_incoming_connections () const</code>
Variant	<code>get_indexed (NodePath property) const</code>
int	<code>get_instance_id () const</code>
Variant	<code>get_meta (String name) const</code>
PoolStringArray	<code>get_meta_list () const</code>
Array	<code>get_method_list () const</code>
Array	<code>get_property_list () const</code>
Reference	<code>get_script () const</code>
Array	<code>get_signal_connection_list (String signal) const</code>
Array	<code>get_signal_list () const</code>
bool	<code>has_meta (String name) const</code>
bool	<code>has_method (String method) const</code>
bool	<code>has_user_signal (String signal) const</code>

Continued on next page

Table 25.15 – continued from previous page

<code>bool</code>	<code>is_blocking_signals () const</code>
<code>bool</code>	<code>is_class (String type) const</code>
<code>bool</code>	<code>is_connected (String signal, Object target, String method) const</code>
<code>bool</code>	<code>is_queued_for_deletion () const</code>
<code>void</code>	<code>notification (int what, bool reversed=false)</code>
<code>void</code>	<code>property_list_changed_notify ()</code>
<code>void</code>	<code>set (String property, Variant value)</code>
<code>void</code>	<code>set_block_signals (bool enable)</code>
<code>void</code>	<code>set_indexed (NodePath property, Variant value)</code>
<code>void</code>	<code>set_message_translation (bool enable)</code>
<code>void</code>	<code>set_meta (String name, Variant value)</code>
<code>void</code>	<code>set_script (Reference script)</code>
<code>String</code>	<code>tr (String message) const</code>

25.245.3 Signals

- `script_changed ()`

25.245.4 Numeric Constants

- **NOTIFICATION_POSTINITIALIZE = 0** — Called right when the object is initialized. Not available in script.
- **NOTIFICATION_PREDELETE = 1** — Called before the object is about to be deleted.

25.245.5 Enums

enum ConnectFlags

- **CONNECT_DEFERRED = 1** — Connect a signal in deferred mode. This way, signal emissions are stored in a queue, then set on idle time.
- **CONNECT_PERSIST = 2** — Persisting connections are saved when the object is serialized to file.
- **CONNECT_ONESHOT = 4** — One shot connections disconnect themselves after emission.

25.245.6 Description

Base class for all non built-in types. Everything not a built-in type starts the inheritance chain from this class.

Objects do not manage memory, if inheriting from one the object will most likely have to be deleted manually (call the `free` function from the script or delete from C++).

Some derivatives add memory management, such as `Reference` (which keeps a reference count and deletes itself automatically when no longer referenced) and `Node`, which deletes the children tree when deleted.

Objects export properties, which are mainly useful for storage and editing, but not really so much in programming. Properties are exported in `_get_property_list` and handled in `_get` and `_set`. However, scripting languages and C++ have simpler means to export them.

Objects also receive notifications (`_notification`). Notifications are a simple way to notify the object about simple events, so they can all be handled together.

25.245.7 Member Function Description

- `void _get (String property) virtual`

Returns the given property. Returns `null` if the property does not exist.

- `Array _get_property_list () virtual`

Returns the object's property list as an `Array` of dictionaries. Dictionaries must contain: `name:String`, `type:int` (see `TYPE_*` enum in `@GlobalScope`) and optionally: `hint:int` (see `PROPERTY_HINT_*` in `@GlobalScope`), `hint_string:String`, `usage:int` (see `PROPERTY_USAGE_*` in `@GlobalScope`).

- `void _init () virtual`

The virtual method called upon initialization.

- `void _notification (int what) virtual`

Notify the object internally using an ID.

- `bool _set (String property, Variant value) virtual`

Sets a property. Returns `true` if the property exists.

- `void add_user_signal (String signal, Array arguments=[])`

Adds a user-defined signal. Arguments are optional, but can be added as an `Array` of dictionaries, each containing "name" and "type" (from `@GlobalScope TYPE_*`).

- `Variant call (String method) vararg`

Calls the method on the object and returns a result. Pass parameters as a comma separated list.

- `Variant call_deferred (String method) vararg`

Calls the method on the object during idle time and returns a result. Pass parameters as a comma separated list.

- `Variant callv (String method, Array arg_array)`

Calls the method on the object and returns a result. Pass parameters as an `Array`.

- `bool can_translate_messages () const`

Returns `true` if the object can translate strings.

- `int connect (String signal, Object target, String method, Array binds=[], int flags=0)`

Connects a signal to a method on a target object. Pass optional binds to the call. Use flags to set deferred or one shot connections. See `CONNECT__*` constants. A signal can only be connected once to a method. It will throw an error if already connected. To avoid this, first use `is_connected` to check for existing connections.

- `void disconnect (String signal, Object target, String method)`

Disconnects a signal from a method on the given target.

- `Variant emit_signal (String signal) vararg`

Emits the given signal.

- `void free ()`

Deletes the object from memory.

- `Variant get (String property) const`

Returns a `Variant` for a property.

- `String get_class () const`

Returns the object's class as a *String*.

- **Array** **get_incoming_connections** () const

Returns an *Array* of dictionaries with information about signals that are connected to the object.

Inside each *Dictionary* there are 3 fields:

- “source” is a reference to signal emitter.
- “signal_name” is name of connected signal.
- “method_name” is a name of method to which signal is connected.
- **Variant** **get_indexed** (*NodePath* property) const
- **int** **get_instance_id** () const

Returns the object's unique instance ID.

- **Variant** **get_meta** (*String* name) const

Returns the object's metadata for the given name.

- **PoolStringArray** **get_meta_list** () const

Returns the object's metadata as a *PoolStringArray*.

- **Array** **get_method_list** () const

Returns the object's methods and their signatures as an *Array*.

- **Array** **get_property_list** () const

Returns the list of properties as an *Array* of dictionaries. Dictionaries contain: name:*String*, type:*int* (see *TYPE_** enum in *@GlobalScope*) and optionally: hint:*int* (see *PROPERTY_HINT_** in *@GlobalScope*), hint_string:*String*, usage:*int* (see *PROPERTY_USAGE_** in *@GlobalScope*).

- **Reference** **get_script** () const

Returns the object's *Script* or *null* if one doesn't exist.

- **Array** **get_signal_connection_list** (*String* signal) const

Returns an *Array* of connections for the given signal.

- **Array** **get_signal_list** () const

Returns the list of signals as an *Array* of dictionaries.

- **bool** **has_meta** (*String* name) const

Returns *true* if a metadata is found with the given name.

- **bool** **has_method** (*String* method) const

Returns *true* if the object contains the given method.

- **bool** **has_user_signal** (*String* signal) const

Returns *true* if the given user-defined signal exists.

- **bool** **is_blocking_signals** () const

Returns *true* if signal emission blocking is enabled.

- **bool** **is_class** (*String* type) const

Returns *true* if the object inherits from the given type.

- **bool** **is_connected** (*String* signal, *Object* target, *String* method) const

Returns `true` if a connection exists for a given `signal`, `target`, and `method`.

- `bool is_queued_for_deletion () const`

Returns `true` if the `queue_free` method was called for the object.

- `void notification (int what, bool reversed=false)`

Notify the object of something.

- `void property_list_changed_notify ()`
- `void set (String property, Variant value)`

Set property into the object.

- `void set_block_signals (bool enable)`

If set to true, signal emission is blocked.

- `void set_indexed (NodePath property, Variant value)`
- `void set_message_translation (bool enable)`

Define whether the object can translate strings (with calls to `tr`). Default is true.

- `void set_meta (String name, Variant value)`

Set a metadata into the object. Metadata is serialized. Metadata can be *anything*.

- `void set_script (Reference script)`

Set a script into the object, scripts extend the object functionality.

- `String tr (String message) const`

Translate a message. Only works if message translation is enabled (which it is by default). See `set_message_translation`.

25.246 OccluderPolygon2D

Inherits: `Resource < Reference < Object`

Category: Core

25.246.1 Brief Description

Defines a 2D polygon for LightOccluder2D.

25.246.2 Member Variables

- `bool closed` - If `true` closes the polygon. A closed `OccluderPolygon2D` occludes the light coming from any direction. An opened `OccluderPolygon2D` occludes the light only at its outline's direction. Default value `true`.
- `CullMode cull_mode` - Set the direction of the occlusion culling when not `CULL_DISABLED`. Default value `DISABLED`.
- `PoolVector2Array polygon` - A `Vector2` array with the index for polygon's vertices positions.

25.246.3 Enums

enum **CullMode**

- **CULL_DISABLED = 0** — Culling mode for the occlusion. Disabled means no culling. See *cull_mode*.
- **CULL_CLOCKWISE = 1** — Culling mode for the occlusion. Sets the culling to be in clockwise direction. See *cull_mode*.
- **CULL_COUNTER_CLOCKWISE = 2** — Culling mode for the occlusion. Sets the culling to be in counter clockwise direction. See *cull_mode*.

25.246.4 Description

Editor facility that helps you draw a 2D polygon used as resource for *LightOccluder2D*.

25.247 OmniLight

Inherits: *Light < VisualInstance < Spatial < Node < Object*

Category: Core

25.247.1 Brief Description

OmniDirectional Light, such as a light bulb or a candle.

25.247.2 Member Variables

- *float omni_attenuation*
- *float omni_range*
- *ShadowDetail omni_shadow_detail*
- *ShadowMode omni_shadow_mode*

25.247.3 Enums

enum **ShadowDetail**

- **SHADOW_DETAIL_VERTICAL = 0**
- **SHADOW_DETAIL_HORIZONTAL = 1**

enum **ShadowMode**

- **SHADOW_DUAL_PARABOLOID = 0**
- **SHADOW_CUBE = 1**

25.247.4 Description

An OmniDirectional light is a type of *Light* node that emits lights in all directions. The light is attenuated through the distance and this attenuation can be configured by changing the energy, radius and attenuation parameters of *Light*.

25.248 OptionButton

Inherits: *Button < BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.248.1 Brief Description

Button control that provides selectable options when pressed.

25.248.2 Member Functions

<code>void</code>	<code>add_icon_item (Texture texture, String label, int id)</code>
<code>void</code>	<code>add_item (String label, int id=-1)</code>
<code>void</code>	<code>add_separator ()</code>
<code>void</code>	<code>clear ()</code>
<code>int</code>	<code>get_item_count () const</code>
<code>Texture</code>	<code>get_item_icon (int idx) const</code>
<code>int</code>	<code>get_item_id (int idx) const</code>
<code>Variant</code>	<code>get_item_metadata (int idx) const</code>
<code>String</code>	<code>get_item_text (int idx) const</code>
<code>PopupMenu</code>	<code>get_popup () const</code>
<code>int</code>	<code>get_selected_id () const</code>
<code>Variant</code>	<code>get_selected_metadata () const</code>
<code>bool</code>	<code>is_item_disabled (int idx) const</code>
<code>void</code>	<code>remove_item (int idx)</code>
<code>void</code>	<code>select (int idx)</code>
<code>void</code>	<code>set_item_disabled (int idx, bool disabled)</code>
<code>void</code>	<code>set_item_icon (int idx, Texture texture)</code>
<code>void</code>	<code>set_item_id (int idx, int id)</code>
<code>void</code>	<code>set_item_metadata (int idx, Variant metadata)</code>
<code>void</code>	<code>set_item_text (int idx, String text)</code>

25.248.3 Signals

- **item_selected (int ID)**

This signal is emitted when the current item was changed by the user. ID of the item selected is passed as argument (if no IDs were added, ID will be just the item index).

25.248.4 Member Variables

- *int selected*

25.248.5 Description

OptionButton is a type button that provides a selectable list of items when pressed. The item selected becomes the “current” item and is displayed as the button text.

25.248.6 Member Function Description

- void **add_icon_item** (*Texture texture, String label, int id*)

Add an item, with a “texture” icon, text “label” and (optionally) id. If no “id” is passed, “id” becomes the item index. New items are appended at the end.

- void **add_item** (*String label, int id=-1*)

Add an item, with text “label” and (optionally) id. If no “id” is passed, “id” becomes the item index. New items are appended at the end.

- void **add_separator** ()

Add a separator to the list of items. Separators help to group items. Separator also takes up an index and is appended at the end.

- void **clear** ()

Clear all the items in the OptionButton.

- *int get_item_count* () const

Return the amount of items in the OptionButton.

- *Texture get_item_icon* (*int idx*) const

Return the icon of the item at index “idx”.

- *int get_item_id* (*int idx*) const

Return the ID of the item at index “idx”.

- *Variant get_item_metadata* (*int idx*) const

- *String get_item_text* (*int idx*) const

Return the text of the item at index “idx”.

- *PopupMenu get_popup* () const

Return the *PopupMenu* contained in this button.

- *int get_selected_id* () const

- *Variant get_selected_metadata* () const

- *bool is_item_disabled* (*int idx*) const

- void **remove_item** (*int idx*)

- void **select** (*int idx*)

Select an item by index and make it the current item.

- void **set_item_disabled** (*int idx, bool disabled*)

- void **set_item_icon** (*int idx, Texture texture*)

Set the icon of an item at index “idx”.

- `void set_item_id (int idx, int id)`

Set the ID of an item at index “idx”.

- `void set_item_metadata (int idx, Variant metadata)`
- `void set_item_text (int idx, String text)`

Set the text of an item at index “idx”.

25.249 OS

Inherits: *Object*

Category: Core

25.249.1 Brief Description

Operating System functions.

25.249.2 Member Functions

<code>void</code>	<code>alert (String text, String title="Alert!")</code>
<code>bool</code>	<code>can_draw () const</code>
<code>bool</code>	<code>can_use_threads () const</code>
<code>void</code>	<code>delay_msec (int msec) const</code>
<code>void</code>	<code>delay_usec (int usec) const</code>
<code>void</code>	<code>dump_memory_to_file (String file)</code>
<code>void</code>	<code>dump_resources_to_file (String file)</code>
<code>int</code>	<code>execute (String path, PoolStringArray arguments, bool blocking, Array output=[])</code>
<code>int</code>	<code>find_scancode_from_string (String string) const</code>
<code>PoolStringArray</code>	<code>get_cmdline_args ()</code>
<code>Dictionary</code>	<code>get_date (bool utc=false) const</code>
<code>Dictionary</code>	<code>get_datetime (bool utc=false) const</code>
<code>Dictionary</code>	<code>get_datetime_from_unix_time (int unix_time_val) const</code>
<code>int</code>	<code>get_dynamic_memory_usage () const</code>
<code>String</code>	<code>get_environment (String environment) const</code>
<code>String</code>	<code>get_executable_path () const</code>
<code>String</code>	<code>get_latin_keyboard_variant () const</code>
<code>String</code>	<code>get_locale () const</code>
<code>String</code>	<code>get_model_name () const</code>
<code>String</code>	<code>get_name () const</code>
<code>int</code>	<code>get_power_percent_left ()</code>
<code>int</code>	<code>get_power_seconds_left ()</code>
<code>int</code>	<code>get_power_state ()</code>
<code>int</code>	<code>get_process_id () const</code>
<code>int</code>	<code>get_processor_count () const</code>
<code>String</code>	<code>get_scancode_string (int code) const</code>
<code>int</code>	<code>get_screen_count () const</code>

Continued on next page

Table 25.16 – continued from previous page

<i>int</i>	<code>get_screen_dpi (int screen=-1) const</code>
<i>Vector2</i>	<code>get_screen_position (int screen=-1) const</code>
<i>Vector2</i>	<code>get_screen_size (int screen=-1) const</code>
<i>int</i>	<code>get_splash_tick_msec () const</code>
<i>int</i>	<code>get_static_memory_peak_usage () const</code>
<i>int</i>	<code>get_static_memory_usage () const</code>
<i>String</i>	<code>get_system_dir (int dir) const</code>
<i>int</i>	<code>get_system_time_secs () const</code>
<i>int</i>	<code>get_ticks_msec () const</code>
<i>Dictionary</i>	<code>get_time (bool utc=false) const</code>
<i>Dictionary</i>	<code>get_time_zone_info () const</code>
<i>String</i>	<code>get_unique_id () const</code>
<i>int</i>	<code>get_unix_time () const</code>
<i>int</i>	<code>get_unix_time_from_datetime (Dictionary datetime) const</code>
<i>String</i>	<code>get_user_data_dir () const</code>
<i>int</i>	<code>get_virtual_keyboard_height ()</code>
<i>bool</i>	<code>has_environment (String environment) const</code>
<i>bool</i>	<code>has_feature (String tag_name) const</code>
<i>bool</i>	<code>has_touchscreen_ui_hint () const</code>
<i>bool</i>	<code>has_virtual_keyboard () const</code>
<i>void</i>	<code>hide_virtual_keyboard ()</code>
<i>bool</i>	<code>is_debug_build () const</code>
<i>bool</i>	<code>is_ok_left_and_cancel_right () const</code>
<i>bool</i>	<code>is_scancode_unicode (int code) const</code>
<i>bool</i>	<code>is_stdout_verbose () const</code>
<i>bool</i>	<code>is_userfs_persistent () const</code>
<i>int</i>	<code>kill (int pid)</code>
<i>bool</i>	<code>native_video_is_playing ()</code>
<i>void</i>	<code>native_video_pause ()</code>
<i>int</i>	<code>native_video_play (String path, float volume, String audio_track, String subtitle_track)</code>
<i>void</i>	<code>native_video_stop ()</code>
<i>void</i>	<code>native_video_unpause ()</code>
<i>void</i>	<code>print_all_resources (String tofile="")</code>
<i>void</i>	<code>print_all_textures_by_size ()</code>
<i>void</i>	<code>print_resources_by_type (PoolStringArray types)</code>
<i>void</i>	<code>print_resources_in_use (bool short=false)</code>
<i>void</i>	<code>request_attention ()</code>
<i>void</i>	<code>set_icon (Image icon)</code>
<i>void</i>	<code>set_ime_position (Vector2 position)</code>
<i>int</i>	<code>set_thread_name (String name)</code>
<i>void</i>	<code>set_use_file_access_save_and_swap (bool enabled)</code>
<i>void</i>	<code>set_window_title (String title)</code>
<i>int</i>	<code>shell_open (String uri)</code>
<i>void</i>	<code>show_virtual_keyboard (String existing_text="")</code>

25.249.3 Member Variables

- ***String clipboard*** - The clipboard from the host OS. Might be unavailable on some platforms.
- ***int current_screen*** - The current screen index (starting from 0).

- *int* **exit_code**
- *bool* **keep_screen_on**
- *bool* **low_processor_usage_mode**
- `_OS.ScreenOrientation screen_orientation` - The current screen orientation.
- *bool* **vsync_enabled**
- *bool* **window_borderless** - If `true`, removes the window frame.
- *bool* **window_fullscreen** - If `true`, the window is fullscreen.
- *bool* **window_maximized** - If `true`, the window is maximized.
- *bool* **window_minimized** - If `true`, the window is minimized.
- *Vector2* **window_position** - The window position relative to the screen, the origin is the top left corner, +Y axis goes to the bottom and +X axis goes to the right.
- *bool* **window_resizable** - If `true`, the window is resizable by the user.
- *Vector2* **window_size** - The size of the window (without counting window manager decorations).

25.249.4 Enums

enum **SystemDir**

- **SYSTEM_DIR_DESKTOP = 0**
- **SYSTEM_DIR_DCIM = 1**
- **SYSTEM_DIR_DOCUMENTS = 2**
- **SYSTEM_DIR_DOWNLOADS = 3**
- **SYSTEM_DIR_MOVIES = 4**
- **SYSTEM_DIR_MUSIC = 5**
- **SYSTEM_DIR_PICTURES = 6**
- **SYSTEM_DIR_RINGTONES = 7**

enum **ScreenOrientation**

- **SCREEN_ORIENTATION_LANDSCAPE = 0**
- **SCREEN_ORIENTATION_PORTRAIT = 1**
- **SCREEN_ORIENTATION_REVERSE_LANDSCAPE = 2**
- **SCREEN_ORIENTATION_REVERSE_PORTRAIT = 3**
- **SCREEN_ORIENTATION_SENSOR_LANDSCAPE = 4**
- **SCREEN_ORIENTATION_SENSOR_PORTRAIT = 5**
- **SCREEN_ORIENTATION_SENSOR = 6**

enum **PowerState**

- **POWERSTATE_UNKNOWN = 0**
- **POWERSTATE_ON_BATTERY = 1**

- **POWERSTATE_NO_BATTERY** = 2
- **POWERSTATE_CHARGING** = 3
- **POWERSTATE_CHARGED** = 4

enum **Weekday**

- **DAY_SUNDAY** = 0
- **DAY_MONDAY** = 1
- **DAY_TUESDAY** = 2
- **DAY_WEDNESDAY** = 3
- **DAY_THURSDAY** = 4
- **DAY_FRIDAY** = 5
- **DAY_SATURDAY** = 6

enum **Month**

- **MONTH_JANUARY** = 1
- **MONTH_FEBRUARY** = 2
- **MONTH_MARCH** = 3
- **MONTH_APRIIL** = 4
- **MONTH_MAY** = 5
- **MONTH_JUNE** = 6
- **MONTH_JULY** = 7
- **MONTH_AUGUST** = 8
- **MONTH_SEPTEMBER** = 9
- **MONTH_OCTOBER** = 10
- **MONTH_NOVEMBER** = 11
- **MONTH_DECEMBER** = 12

25.249.5 Description

Operating System functions. OS Wraps the most common functionality to communicate with the host Operating System, such as: mouse grabbing, mouse cursors, clipboard, video mode, date and time, timers, environment variables, execution of binaries, command line, etc.

25.249.6 Member Function Description

- void **alert** (*String* text, *String* title="Alert!")

Displays a modal dialog box utilizing the host OS.

- *bool* **can_draw** () const

Returns `true` if the host OS allows drawing.

- *bool* **can_use_threads** () const

Returns `true` if the current host platform is using multiple threads.

- `void delay_msec (int msec) const`

Delay executing of the current thread by given milliseconds.

- `void delay_usec (int usec) const`

Delay executing of the current thread by given microseconds.

- `void dump_memory_to_file (String file)`

Dumps the memory allocation ringlist to a file (only works in debug).

Entry format per line: “Address - Size - Description”

- `void dump_resources_to_file (String file)`

Dumps all used resources to file (only works in debug).

Entry format per line: “Resource Type : Resource Location”

At the end of the file is a statistic of all used Resource Types.

- `int execute (String path, PoolStringArray arguments, bool blocking, Array output=[])`

Execute the file at the given path, optionally blocking until it returns.

Platform path resolution will take place. The resolved file must exist and be executable.

Returns a process id.

For example:

```
var output = []
var pid = OS.execute('ls', [], true, output)
```

If you wish to access a shell built-in or perform a composite command, a platform specific shell can be invoked. For example:

```
var pid = OS.execute('CMD.exe', ['/C', 'cd %TEMP% && dir'], true, output)
```

- `int find_scancode_from_string (String string) const`

Returns the scancode of the given string (e.g. “Escape”)

- `PoolStringArray get_cmdline_args ()`

Returns the command line arguments passed to the engine.

- `Dictionary get_date (bool utc=false) const`

Returns current date as a dictionary of keys: year, month, day, weekday, dst (daylight savings time).

- `Dictionary get_datetime (bool utc=false) const`

Returns current datetime as a dictionary of keys: year, month, day, weekday, dst (daylight savings time), hour, minute, second.

- `Dictionary get_datetime_from_unix_time (int unix_time_val) const`

Get a dictionary of time values when given epoch time.

Dictionary Time values will be a union of values from `get_time` and `get_date` dictionaries (with the exception of dst = day light standard time, as it cannot be determined from epoch).

- `int get_dynamic_memory_usage () const`

Returns the total amount of dynamic memory used (only works in debug).

- *String* **get_environment** (*String* environment) const

Returns an environment variable.

- *String* **get_executable_path** () const

Returns the path to the current engine executable.

- *String* **get_latin_keyboard_variant** () const

Returns the current latin keyboard variant as a String.

Possible return values are: “QWERTY”, “AZERTY”, “QZERTY”, “DVORAK”, “NEO”, “COLEMAK” or “ERROR”.

- *String* **get_locale** () const

Returns the host OS locale.

- *String* **get_model_name** () const

Returns the model name of the current device.

- *String* **get_name** () const

Returns the name of the host OS. Possible values are: “Android”, “Haiku”, “iOS”, “HTML5”, “OSX”, “Server”, “Windows”, “UWP”, “X11”.

- *int* **get_power_percent_left** ()

Returns the amount of battery left in the device as a percentage.

- *int* **get_power_seconds_left** ()

Returns the time in seconds before the device runs out of battery.

- *int* **get_power_state** ()

Returns the current state of the device regarding battery and power. See `POWERSTATE__*` constants.

- *int* **get_process_id** () const

Returns the game process ID

- *int* **get_processor_count** () const

Returns the number of cores available in the host machine.

- *String* **get_scancode_string** (*int* code) const

Returns the given scancode as a string (e.g. Return values: “Escape”, “Shift+Escape”).

- *int* **get_screen_count** () const

Returns the number of displays attached to the host machine.

- *int* **get_screen_dpi** (*int* screen=-1) const

Returns the dots per inch density of the specified screen.

On Android Devices, the actual screen densities are grouped into six generalized densities:

ldpi - 120 dpi

mdpi - 160 dpi

hdpi - 240 dpi

xhdpi - 320 dpi

xxhdpi - 480 dpi

xxxhdpi - 640 dpi

- `Vector2 get_screen_position (int screen=-1) const`

Returns the position of the specified screen by index. If no screen index is provided, the current screen will be used.

- `Vector2 get_screen_size (int screen=-1) const`

Returns the dimensions in pixels of the specified screen.

- `int get_splash_tick_msec () const`
- `int get_static_memory_peak_usage () const`

Returns the max amount of static memory used (only works in debug).

- `int get_static_memory_usage () const`

Returns the amount of static memory being used by the program in bytes.

- `String get_system_dir (int dir) const`

Returns the actual path to commonly used folders across different platforms. Available locations are specified in OS.SystemDir.

- `int get_system_time_secs () const`

Returns the epoch time of the operating system in seconds.

- `int get_ticks_msec () const`

Returns the amount of time passed in milliseconds since the engine started.

- `Dictionary get_time (bool utc=false) const`

Returns current time as a dictionary of keys: hour, minute, second.

- `Dictionary get_time_zone_info () const`

Returns the current time zone as a dictionary with the keys: bias and name.

- `String get_unique_id () const`

Returns a string that is unique to the device. Currently only works on Android and iOS. Returns empty string on other platforms.

- `int get_unix_time () const`

Return the current unix timestamp.

- `int get_unix_time_from_datetime (Dictionary datetime) const`

Get an epoch time value from a dictionary of time values.

`datetime` must be populated with the following keys: year, month, day, hour, minute, second.

You can pass the output from `get_datetime_from_unix_time` directly into this function. Daylight savings time (dst), if present, is ignored.

- `String get_user_data_dir () const`

Returns the absolute directory path where user data is written (`user://`).

- `int get_virtual_keyboard_height ()`
- `bool has_environment (String environment) const`

Returns `true` if an environment variable exists.

- `bool has_feature (String tag_name) const`

- *bool has_touchscreen_ui_hint () const*

Returns `true` if the device has a touchscreen or emulates one.

- *bool has_virtual_keyboard () const*

Returns `true` if the platform has a virtual keyboard, `false` otherwise.

- *void hide_virtual_keyboard ()*

Hides the virtual keyboard if it is shown, does nothing otherwise.

- *bool is_debug_build () const*

Returns `true` if the build is a debug build.

Returns `true` when running in the editor.

Returns `false` if the build is a release build.

- *bool is_ok_left_and_cancel_right () const*

Returns `true` if the “Okay” button should appear on the left and “Cancel” on the right.

- *bool is_scancode_unicode (int code) const*

Returns `true` if the input code has a unicode character.

- *bool is_stdout_verbose () const*

Returns `true` if the engine was executed with `-v` (verbose stdout).

- *bool is_userfs_persistent () const*

If `true`, the `user://` file system is persistent, so that its state is the same after a player quits and starts the game again. Relevant to the HTML5 platform, where this persistence may be unavailable.

- *int kill (int pid)*

Kill a process ID (this method can be used to kill processes that were not spawned by the game).

- *bool native_video_is_playing ()*

Returns `true` if native video is playing.

- *void native_video_pause ()*

Pauses native video playback.

- *int native_video_play (String path, float volume, String audio_track, String subtitle_track)*

Plays native video from the specified path, at the given volume and with audio and subtitle tracks.

- *void native_video_stop ()*

Stops native video playback.

- *void native_video_unpause ()*

Resumes native video playback.

- *void print_all_resources (String tofile="")*

Shows all resources in the game. Optionally the list can be written to a file.

- *void print_all_textures_by_size ()*

Shows the list of loaded textures sorted by size in memory.

- *void print_resources_by_type (PoolStringArray types)*

Shows the number of resources loaded by the game of the given types.

- `void print_resources_in_use (bool short=false)`

Shows all resources currently used by the game.

- `void request_attention ()`

Request the user attention to the window. It'll flash the taskbar button on Windows or bounce the dock icon on OSX.

- `void set_icon (Image icon)`

Sets the game's icon.

- `void set_ime_position (Vector2 position)`

- `int set_thread_name (String name)`

Sets the name of the current thread.

- `void set_use_file_access_save_and_swap (bool enabled)`

Enables backup saves if `enabled` is `true`.

- `void set_window_title (String title)`

Sets the window title to the specified string.

- `int shell_open (String uri)`

Requests the OS to open a resource with the most appropriate program. For example.

`OS.shell_open ("C:\\\\Users\\\\name\\\\Downloads")` on Windows opens the file explorer at the downloads folders of the user.

`OS.shell_open ("http://godotengine.org")` opens the default web browser on the official Godot website.

- `void show_virtual_keyboard (String existing_text="")`

Shows the virtual keyboard if the platform has one. The `existing_text` parameter is useful for implementing your own LineEdit, as it tells the virtual keyboard what text has already been typed (the virtual keyboard uses it for auto-correct and predictions).

25.250 PackedDataContainer

Inherits: `Resource < Reference < Object`

Category: Core

25.250.1 Brief Description

25.250.2 Member Functions

<code>int</code>	<code>pack (Variant value)</code>
<code>int</code>	<code>size () const</code>

25.250.3 Member Variables

- *PoolByteArray __data__*

25.250.4 Member Function Description

- *int pack (Variant value)*
- *int size () const*

25.251 PackedDataContainerRef

Inherits: *Reference < Object*

Category: Core

25.251.1 Brief Description

25.251.2 Member Functions

<i>int</i>	<i>size () const</i>
------------	-----------------------

25.251.3 Member Function Description

- *int size () const*

25.252 PackedScene

Inherits: *Resource < Reference < Object*

Category: Core

25.252.1 Brief Description

An abstraction of a serialized scene.

25.252.2 Member Functions

<i>bool</i>	<i>can_instance () const</i>
<i>SceneState</i>	<i>get_state ()</i>
<i>Node</i>	<i>instance (int edit_state=0) const</i>
<i>int</i>	<i>pack (Node path)</i>

25.252.3 Member Variables

- *Dictionary _bundled* - A dictionary representation of the scene contents.

Available keys include “rnames” and “variants” for resources, “node_count”, “nodes”, “node_paths” for nodes, “editable_instances” for base scene children overrides, “conn_count” and “conns” for signal connections, and “version” for the format style of the PackedScene.

25.252.4 Enums

enum GenEditState

- **GEN_EDIT_STATE_DISABLED = 0** — If passed to *instance*, blocks edits to the scene state.
- **GEN_EDIT_STATE_INSTANCE = 1** — If passed to *instance*, provides local scene resources to the local scene. Requires tools compiled.
- **GEN_EDIT_STATE_MAIN = 2** — If passed to *instance*, provides local scene resources to the local scene. Only the main scene should receive the main edit state. Requires tools compiled.

25.252.5 Description

A simplified interface to a scene file. Provides access to operations and checks that can be performed on the scene resource itself.

TODO: explain ownership, and that node does not need to own itself

25.252.6 Member Function Description

- *bool can_instance () const*

Returns `true` if the scene file has nodes.

- *SceneState get_state ()*

Returns the `SceneState` representing the scene file contents.

- *Node instance (int edit_state=0) const*

Instantiates the scene’s node hierarchy. Triggers child scene instantiation(s). Triggers the enum `Object.NOTIFICATION_INSTANCED` notification on the root node.

- *int pack (Node path)*

Pack will ignore any sub-nodes not owned by given node. See `Node.set_owner`.

25.253 PacketPeer

Inherits: *Reference < Object*

Inherited By: *PacketPeerStream, PacketPeerUDP, NetworkedMultiplayerPeer*

Category: Core

25.253.1 Brief Description

Abstraction and base class for packet-based protocols.

25.253.2 Member Functions

<i>int</i>	<code>get_available_packet_count () const</code>
<i>PoolByteArray</i>	<code>get_packet ()</code>
<i>int</i>	<code>get_packet_error () const</code>
<i>Variant</i>	<code>get_var ()</code>
<i>int</i>	<code>put_packet (PoolByteArray buffer)</code>
<i>int</i>	<code>put_var (Variant var)</code>

25.253.3 Member Variables

- `bool allow_object_decoding`

25.253.4 Description

PacketPeer is an abstraction and base class for packet-based protocols (such as UDP). It provides an API for sending and receiving packets both as raw data or variables. This makes it easy to transfer data over a protocol, without having to encode data as low level bytes or having to worry about network ordering.

25.253.5 Member Function Description

- `int get_available_packet_count () const`

Return the number of packets currently available in the ring-buffer.

- `PoolByteArray get_packet ()`

Get a raw packet.

- `int get_packet_error () const`

Return the error state of the last packet received (via `get_packet` and `get_var`).

- `Variant get_var ()`

Get a Variant.

- `int put_packet (PoolByteArray buffer)`

Send a raw packet.

- `int put_var (Variant var)`

Send a Variant as a packet.

25.254 PacketPeerStream

Inherits: *PacketPeer < Reference < Object*

Category: Core

25.254.1 Brief Description

Wrapper to use a PacketPeer over a StreamPeer.

25.254.2 Member Variables

- *int input_buffer_max_size*
- *int output_buffer_max_size*
- *StreamPeer stream_peer* - The wrapped *StreamPeer* object.

25.254.3 Description

PacketStreamPeer provides a wrapper for working using packets over a stream. This allows for using packet based code with StreamPeers. PacketPeerStream implements a custom protocol over the StreamPeer, so the user should not read or write to the wrapped StreamPeer directly.

25.255 PacketPeerUDP

Inherits: *PacketPeer < Reference < Object*

Category: Core

25.255.1 Brief Description

UDP packet peer.

25.255.2 Member Functions

<i>void</i>	<i>close ()</i>
<i>String</i>	<i>get_packet_ip () const</i>
<i>int</i>	<i>get_packet_port () const</i>
<i>bool</i>	<i>is_listening () const</i>
<i>int</i>	<i>listen (int port, String bind_address="*", int recv_buf_size=65536)</i>
<i>int</i>	<i>set_dest_address (String host, int port)</i>
<i>int</i>	<i>wait ()</i>

25.255.3 Description

UDP packet peer. Can be used to send raw UDP packets as well as *Variants*.

25.255.4 Member Function Description

- `void close()`

Close the UDP socket the `PacketPeerUDP` is currently listening on.

- `String get_packet_ip() const`

Return the IP of the remote peer that sent the last packet(that was received with `get_packet` or `get_var`).

- `int get_packet_port() const`

Return the port of the remote peer that sent the last packet(that was received with `get_packet` or `get_var`).

- `bool is_listening() const`

Return whether this `PacketPeerUDP` is listening.

- `int listen(int port, String bind_address="*", int recv_buf_size=65536)`

Make this `PacketPeerUDP` listen on the “port” binding to “bind_address” with a buffer size “recv_buf_size”.

If “bind_address” is set as “*” (default), the peer will listen on all available addresses (both IPv4 and IPv6).

If “bind_address” is set as “0.0.0.0” (for IPv4) or “::” (for IPv6), the peer will listen on all available addresses matching that IP type.

If “bind_address” is set to any valid address (e.g. “192.168.1.101”, “::1”, etc), the peer will only listen on the interface with that addresses (or fail if no interface with the given address exists).

- `int set_dest_address(String host, int port)`

Set the destination address and port for sending packets and variables, a hostname will be resolved using if valid.

- `int wait()`

Wait for a packet to arrive on the listening port, see *listen*.

25.256 Panel

Inherits: `Control < CanvasItem < Node < Object`

Category: Core

25.256.1 Brief Description

Provides an opaque background for `Control` children.

25.256.2 Description

Panel is a `Control` that displays an opaque background. It’s commonly used as a parent and container for other types of `Control` nodes.

25.257 PanelContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Inherited By: *ScriptEditor*

Category: Core

25.257.1 Brief Description

Panel container type.

25.257.2 Description

Panel container type. This container fits controls inside of the delimited area of a stylebox. It's useful for giving controls an outline.

25.258 PanoramaSky

Inherits: *Sky < Resource < Reference < Object*

Category: Core

25.258.1 Brief Description

25.258.2 Member Variables

- *Texture panorama*

25.259 ParallaxBackground

Inherits: *CanvasLayer < Node < Object*

Category: Core

25.259.1 Brief Description

A node used to create a parallax scrolling background.

25.259.2 Member Variables

- *Vector2 scroll_base_offset* - Base position offset of all *ParallaxLayer* children.
- *Vector2 scroll_base_scale* - Base motion scale of all *ParallaxLayer* children.

- `bool scroll_ignore_camera_zoom` - If `true` elements in `ParallaxLayer` child aren't affected by the zoom level of the camera.
- `Vector2 scroll_limit_begin` - Top left limits for scrolling to begin. If the camera is outside of this limit the background will stop scrolling. Must be lower than `scroll_limit_end` to work.
- `Vector2 scroll_limit_end` - Right bottom limits for scrolling to end. If the camera is outside of this limit the background will stop scrolling. Must be higher than `scroll_limit_begin` to work.
- `Vector2 scroll_offset` - The `ParallaxBackground`'s scroll value. Calculated automatically when using a `Camera2D`, but can be used to manually manage scrolling when no camera is present.

25.259.3 Description

A `ParallaxBackground` uses one or more `ParallaxLayer` child nodes to create a parallax effect. Each `ParallaxLayer` can move at a different speed using `ParallaxLayer.motion_offset`. This creates an illusion of depth in a 2D game. If not used with a `Camera2D`, you must manually calculate the `scroll_offset`.

25.260 ParallaxLayer

Inherits: `Node2D < CanvasItem < Node < Object`

Category: Core

25.260.1 Brief Description

A parallax scrolling layer to be used with `ParallaxBackground`.

25.260.2 Member Variables

- `Vector2 motion_mirroring` - The `ParallaxLayer`'s `Texture` mirroring. Useful for creating an infinite scrolling background. If an axis is set to `0` the `Texture` will not be mirrored. Default value: `(0, 0)`.
- `Vector2 motion_offset` - The `ParallaxLayer`'s offset relative to the parent `ParallaxBackground`'s `ParallaxBackground.scroll_offset`.
- `Vector2 motion_scale` - Multiplies the `ParallaxLayer`'s motion. If an axis is set to `0` it will not scroll.

25.260.3 Description

A `ParallaxLayer` must be the child of a `ParallaxBackground` node. Each `ParallaxLayer` can be set to move at different speeds relative to the camera movement or the `ParallaxBackground.scroll_offset` value.

This node's children will be affected by its scroll offset.

25.261 Particles

Inherits: `GeometryInstance < VisualInstance < Spatial < Node < Object`

Category: Core

25.261.1 Brief Description

3D particle emitter.

25.261.2 Member Functions

AABB	<code>capture_aabb () const</code>
void	<code>restart ()</code>

25.261.3 Member Variables

- *int amount* - Number of particles to emit.
- *DrawOrder draw_order* - Particle draw order. Uses DRAW_ORDER_* values. Default value: DRAW_ORDER_INDEX.
- *Mesh draw_pass_1*
- *Mesh draw_pass_2*
- *Mesh draw_pass_3*
- *Mesh draw_pass_4*
- *int draw_passes*
- *bool emitting* - If true particles are being emitted. Default value: true.
- *float explosiveness* - Time ratio between each emission. If 0 particles are emitted continuously. If 1 all particles are emitted simultaneously. Default value: 0.
- *int fixed_fps*
- *bool fract_delta*
- *float lifetime* - Amount of time each particle will exist. Default value: 1.
- *bool local_coords* - If true particles use the parent node's coordinate space. If false they use global coordinates. Default value: true.
- *bool one_shot* - If true only amount particles will be emitted. Default value: false.
- *float preprocess*
- *Material process_material* - Material for processing particles. Can be a *ParticlesMaterial* or a *ShaderMaterial*.
- *float randomness* - Emission randomness ratio. Default value: 0.
- *float speed_scale* - Speed scaling ratio. Default value: 1.
- *AABB visibility_aabb*

25.261.4 Numeric Constants

- **MAX_DRAW_PASSES = 4**

25.261.5 Enums

enum DrawOrder

- **DRAW_ORDER_INDEX = 0** — Particles are drawn in the order emitted.
- **DRAW_ORDER_LIFETIME = 1** — Particles are drawn in order of remaining lifetime.
- **DRAW_ORDER_VIEW_DEPTH = 2** — Particles are drawn in order of depth.

25.261.6 Description

3D particle node used to create a variety of particle systems and effects. Particles features an emitter that generates some number of particles at a given rate.

Use the `process_material` property to add a *ParticlesMaterial* to configure particle appearance and behavior. Alternatively, you can add a *ShaderMaterial* which will be applied to all particles.

25.261.7 Member Function Description

- `AABB capture_aabb () const`
- `void restart ()`

25.262 Particles2D

Inherits: `Node2D < CanvasItem < Node < Object`

Category: Core

25.262.1 Brief Description

2D particle emitter.

25.262.2 Member Functions

<code>Rect2</code>	<code>capture_rect () const</code>
<code>void</code>	<code>restart ()</code>

25.262.3 Member Variables

- `int amount` - Number of particles emitted in one emission cycle.
- `DrawOrder draw_order` - Particle draw order. Uses `DRAW_ORDER_*` values. Default value: `DRAW_ORDER_INDEX`.
- `bool emitting` - If true particles are being emitted. Default value: `true`.

- *float explosiveness* - How rapidly particles in an emission cycle are emitted. If greater than 0, there will be a gap in emissions before the next cycle begins. Default value: 0.
- *int fixed_fps*
- *bool fract_delta*
- *int h_frames* - Number of horizontal frames in `texture`.
- *float lifetime* - Amount of time each particle will exist. Default value: 1.
- *bool local_coords* - If `true` particles use the parent node's coordinate space. If `false` they use global coordinates. Default value: `true`.
- *Texture normal_map*
- *bool one_shot* - If `true` only one emission cycle occurs. If set `true` during a cycle, emission will stop at the cycle's end. Default value: `false`.
- *float preprocess* - Particle system starts as if it had already run for this many seconds.
- *Material process_material* - *Material* for processing particles. Can be a *ParticlesMaterial* or a *ShaderMaterial*.
- *float randomness* - Emission lifetime randomness ratio. Default value: 0.
- *float speed_scale* - Particle system's running speed scaling ratio. Default value: 1.
- *Texture texture* - Particle texture. If `null` particles will be squares.
- *int v_frames* - Number of vertical frames in `texture`.
- *Rect2 visibility_rect* - Editor visibility helper.

25.262.4 Enums

enum DrawOrder

- **DRAW_ORDER_INDEX = 0** — Particles are drawn in the order emitted.
- **DRAW_ORDER_LIFETIME = 1** — Particles are drawn in order of remaining lifetime.

25.262.5 Description

2D particle node used to create a variety of particle systems and effects. `Particles2D` features an emitter that generates some number of particles at a given rate.

Use the `process_material` property to add a *ParticlesMaterial* to configure particle appearance and behavior. Alternatively, you can add a *ShaderMaterial* which will be applied to all particles.

25.262.6 Member Function Description

- *Rect2 capture_rect () const*
- *void restart ()*

25.263 ParticlesMaterial

Inherits: *Material < Resource < Reference < Object*

Category: Core

25.263.1 Brief Description

Particle properties for *Particles* and *Particles2D* nodes.

25.263.2 Member Variables

- *float angle* - Initial rotation applied to each particle.
- *Texture angle_curve* - Each particle's rotation will be animated along this *CurveTexture*.
- *float angle_random* - Rotation randomness ratio. Default value: 0.
- *float angular_velocity* - Initial angular velocity applied to each particle.
- *Texture angular_velocity_curve* - Each particle's angular velocity will vary along this *CurveTexture*.
- *float angular_velocity_random* - Angular velocity randomness ratio. Default value: 0.
- *bool anim_loop* - If `true` animation will loop. Default value: `false`.
- *float anim_offset* - Particle animation offset.
- *Texture anim_offset_curve* - Each particle's animation offset will vary along this *CurveTexture*.
- *float anim_offset_random* - Animation offset randomness ratio. Default value: 0.
- *float anim_speed* - Particle animation speed.
- *Texture anim_speed_curve* - Each particle's animation speed will vary along this *CurveTexture*.
- *float anim_speed_random* - Animation speed randomness ratio. Default value: 0.
- *Color color* - Each particle's initial color. If the Particle2D's `texture` is defined, it will be multiplied by this color.
- *Texture color_ramp* - Each particle's color will vary along this *GradientTexture*.
- *float damping* - The rate at which particles lose velocity.
- *Texture damping_curve* - Damping will vary along this *CurveTexture*.
- *float damping_random* - Damping randomness ratio. Default value: 0.
- *Vector3 emission_box_extents* - The box's extents if `emission_shape` is set to `EMISSION_SHAPE_BOX`.
- *Texture emission_color_texture*
- *Texture emission_normal_texture*
- *int emission_point_count* - The number of emission points if `emission_shape` is set to `EMISSION_SHAPE_POINTS` or `EMISSION_SHAPE_DIRECTED_POINTS`.
- *Texture emission_point_texture*
- *EmissionShape emission_shape* - Particles will be emitted inside this region. Use `EMISSION_SHAPE__*` constants for values. Default value: `EMISSION_SHAPE_POINT`.

- *float emission_sphere_radius* - The sphere's radius if emission_shape is set to EMISSION_SHAPE_SPHERE.
- *bool flag_align_y*
- *bool flag_disable_z* - If true particles will not move on the z axis. Default value: true for Particles2D, false for Particles.
- *bool flag_rotate_y*
- *float flatness*
- *Vector3 gravity* - Gravity applied to every particle. Default value: (0, 98, 0).
- *float hue_variation* - Initial hue variation applied to each particle.
- *Texture hue_variation_curve* - Each particle's hue will vary along this *CurveTexture*.
- *float hue_variation_random* - Hue variation randomness ratio. Default value: 0.
- *float initial_velocity* - Initial velocity for each particle.
- *float initial_velocity_random* - Initial velocity randomness ratio. Default value: 0.
- *float linear_accel* - Linear acceleration applied to each particle.
- *Texture linear_accel_curve* - Each particle's linear acceleration will vary along this *CurveTexture*.
- *float linear_accel_random* - Linear acceleration randomness ratio. Default value: 0.
- *float orbit_velocity* - Orbital velocity applied to each particle.
- *Texture orbit_velocity_curve* - Each particle's orbital velocity will vary along this *CurveTexture*.
- *float orbit_velocity_random* - Orbital velocity randomness ratio. Default value: 0.
- *float radial_accel* - Radial acceleration applied to each particle.
- *Texture radial_accel_curve* - Each particle's radial acceleration will vary along this *CurveTexture*.
- *float radial_accel_random* - Radial acceleration randomness ratio. Default value: 0.
- *float scale* - Initial scale applied to each particle.
- *Texture scale_curve* - Each particle's scale will vary along this *CurveTexture*.
- *float scale_random* - Scale randomness ratio. Default value: 0.
- *float spread* - Each particle's initial direction range from +spread to -spread degrees. Default value: 45.
- *float tangential_accel* - Tangential acceleration applied to each particle. Tangential acceleration is perpendicular to the particle's velocity.
- *Texture tangential_accel_curve* - Each particle's tangential acceleration will vary along this *CurveTexture*.
- *float tangential_accel_random* - Tangential acceleration randomness ratio. Default value: 0.
- *GradientTexture trail_color_modifier* - Trail particles' color will vary along this *GradientTexture*.
- *int trail_divisor* - Emitter will emit amount divided by trail_divisor particles. The remaining particles will be used as trail(s).
- *CurveTexture trail_size_modifier* - Trail particles' size will vary along this *CurveTexture*.

25.263.3 Enums

enum Flags

- **FLAG_ALIGN_Y_TO_VELOCITY = 0** — Use with set_flag to set *flag_align_y*.
- **FLAG_ROTATE_Y = 1** — Use with set_flag to set *flag_rotate_y*
- **FLAG_MAX = 4**

enum Parameter

- **PARAM_INITIAL_LINEAR_VELOCITY = 0** — Use with set_param, set_param_randomness, and set_param_texture to set initial velocity properties.
- **PARAM_ANGULAR_VELOCITY = 1** — Use with set_param, set_param_randomness, and set_param_texture to set angular velocity properties.
- **PARAM_ORBIT_VELOCITY = 2** — Use with set_param, set_param_randomness, and set_param_texture to set orbital_velocity properties.
- **PARAM_LINEAR_ACCEL = 3** — Use with set_param, set_param_randomness, and set_param_texture to set linear acceleration properties.
- **PARAM_RADIAL_ACCEL = 4** — Use with set_param, set_param_randomness, and set_param_texture to set radial acceleration properties.
- **PARAM_TANGENTIAL_ACCEL = 5** — Use with set_param, set_param_randomness, and set_param_texture to set tangential acceleration properties.
- **PARAM_DAMPING = 6** — Use with set_param, set_param_randomness, and set_param_texture to set damping properties.
- **PARAM_ANGLE = 7** — Use with set_param, set_param_randomness, and set_param_texture to set angle properties.
- **PARAM_SCALE = 8** — Use with set_param, set_param_randomness, and set_param_texture to set scale properties.
- **PARAM_HUE_VARIATION = 9** — Use with set_param, set_param_randomness, and set_param_texture to set hue_variation properties.
- **PARAM_ANIM_SPEED = 10** — Use with set_param, set_param_randomness, and set_param_texture to set animation speed properties.
- **PARAM_ANIM_OFFSET = 11** — Use with set_param, set_param_randomness, and set_param_texture to set animation offset properties.
- **PARAM_MAX = 12**

enum EmissionShape

- **EMISSION_SHAPE_POINT = 0** — All particles will be emitted from a single point.
- **EMISSION_SHAPE_SPHERE = 1** — Particles will be emitted in the volume of a sphere.
- **EMISSION_SHAPE_BOX = 2** — Particles will be emitted in the volume of a box.
- **EMISSION_SHAPE_POINTS = 3**
- **EMISSION_SHAPE_DIRECTED_POINTS = 4**

25.263.4 Description

ParticlesMaterial defines particle properties and behavior. It is used in the `process_material` of *Particles* and *Particles2D* emitter nodes.

Some of this material's properties are applied to each particle when emitted, while others can have a *CurveTexture* applied to vary values over the lifetime of the particle.

25.264 Path

Inherits: *Spatial < Node < Object*

Category: Core

25.264.1 Brief Description

Container for a *Curve3D*.

25.264.2 Member Variables

- *Curve3D curve*

25.264.3 Description

This class is a container/Node-ification of a *Curve3D*, so it can have *Spatial* properties and *Node* info.

25.265 Path2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.265.1 Brief Description

Contains a *Curve2D* path for *PathFollow2D* nodes to follow.

25.265.2 Member Variables

- *Curve2D curve* - A *Curve2D* describing the path.

25.265.3 Description

Can have *PathFollow2D* child-nodes moving along the *Curve2D*. See *PathFollow2D* for more information on this usage.

25.266 PathFollow

Inherits: *Spatial < Node < Object*

Category: Core

25.266.1 Brief Description

Point sampler for a *Path*.

25.266.2 Member Variables

- *bool cubic_interp* - If `true` the position between two cached points is interpolated cubically, and linearly otherwise.

The points along the *Curve3D* of the *Path* are precomputed before use, for faster calculations. The point at the requested offset is then calculated interpolating between two adjacent cached points. This may present a problem if the curve makes sharp turns, as the cached points may not follow the curve closely enough.

There are two answers to this problem: Either increase the number of cached points and increase memory consumption, or make a cubic interpolation between two points at the cost of (slightly) slower calculations.

- *float h_offset* - The node's offset along the curve.
- *bool loop* - If `true`, any offset outside the path's length will wrap around, instead of stopping at the ends. Use it for cyclic paths.
- *float offset* - The distance from the first vertex, measured in 3D units along the path. This sets this node's position to a point within the path.
- *RotationMode rotation_mode* - Allows or forbids rotation on one or more axes, depending on the constants being used.
- *float unit_offset* - The distance from the first vertex, considering 0.0 as the first vertex and 1.0 as the last. This is just another way of expressing the offset within the path, as the offset supplied is multiplied internally by the path's length.
- *float v_offset* - The node's offset perpendicular to the curve.

25.266.3 Enums

enum **RotationMode**

- **ROTATION_NONE = 0** — Forbids the PathFollow to rotate.
- **ROTATION_Y = 1** — Allows the PathFollow to rotate in the Y axis only.
- **ROTATION_XY = 2** — Allows the PathFollow to rotate in both the X, and Y axes.
- **ROTATION_XYZ = 3** — Allows the PathFollow to rotate in any axis.

25.266.4 Description

This node takes its parent *Path*, and returns the coordinates of a point within it, given a distance from the first vertex.

It is useful for making other nodes follow a path, without coding the movement pattern. For that, the nodes must be descendants of this node. Then, when setting an offset in this node, the descendant nodes will move accordingly.

25.267 PathFollow2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.267.1 Brief Description

Point sampler for a *Path2D*.

25.267.2 Member Variables

- *bool cubic_interp* - If `true` the position between two cached points is interpolated cubically, and linearly otherwise.

The points along the *Curve2D* of the *Path2D* are precomputed before use, for faster calculations. The point at the requested offset is then calculated interpolating between two adjacent cached points. This may present a problem if the curve makes sharp turns, as the cached points may not follow the curve closely enough.

There are two answers to this problem: Either increase the number of cached points and increase memory consumption, or make a cubic interpolation between two points at the cost of (slightly) slower calculations.

- *float h_offset* - The node's offset along the curve.
- *float lookahead*
- *bool loop* - If `true`, any offset outside the path's length will wrap around, instead of stopping at the ends. Use it for cyclic paths.
- *float offset* - The distance along the path in pixels.
- *bool rotate* - If `true`, this node rotates to follow the path, making its descendants rotate.
- *float unit_offset* - The distance along the path as a number in the range 0.0 (for the first vertex) to 1.0 (for the last). This is just another way of expressing the offset within the path, as the offset supplied is multiplied internally by the path's length.
- *float v_offset* - The node's offset perpendicular to the curve.

25.267.3 Description

This node takes its parent *Path2D*, and returns the coordinates of a point within it, given a distance from the first vertex.

It is useful for making other nodes follow a path, without coding the movement pattern. For that, the nodes must be descendants of this node. Then, when setting an offset in this node, the descendant nodes will move accordingly.

25.268 PCKPacker

Inherits: *Reference < Object*

Category: Core

25.268.1 Brief Description

25.268.2 Member Functions

<i>int</i>	<code>add_file (String pck_path, String source_path)</code>
<i>int</i>	<code>flush (bool verbose)</code>
<i>int</i>	<code>pck_start (String pck_name, int alignment)</code>

25.268.3 Member Function Description

- `int add_file (String pck_path, String source_path)`
- `int flush (bool verbose)`
- `int pck_start (String pck_name, int alignment)`

25.269 Performance

Inherits: *Object*

Category: Core

25.269.1 Brief Description

Exposes performance related data.

25.269.2 Member Functions

<i>float</i>	<code>get_monitor (int monitor) const</code>
--------------	--

25.269.3 Enums

enum **Monitor**

- **TIME_FPS = 0** — Frames per second.
- **TIME_PROCESS = 1** — Time it took to complete one frame.
- **TIME_PHYSICS_PROCESS = 2** — Time it took to complete one physics frame.
- **MEMORY_STATIC = 3** — Static memory currently used, in bytes. Not available in release builds.

- **MEMORY_DYNAMIC = 4** — Dynamic memory currently used, in bytes. Not available in release builds.
- **MEMORY_STATIC_MAX = 5** — Available static memory. Not available in release builds.
- **MEMORY_DYNAMIC_MAX = 6** — Available dynamic memory. Not available in release builds.
- **MEMORY_MESSAGE_BUFFER_MAX = 7** — Largest amount of memory the message queue buffer has used, in bytes. The message queue is used for deferred functions calls and notifications.
- **OBJECT_COUNT = 8** — Number of objects currently instanced (including nodes).
- **OBJECT_RESOURCE_COUNT = 9** — Number of resources currently used.
- **OBJECT_NODE_COUNT = 10** — Number of nodes currently instanced. This also includes the root node, as well as any nodes not in the scene tree.
- **RENDER_OBJECTS_IN_FRAME = 11** — 3D objects drawn per frame.
- **RENDER_VERTICES_IN_FRAME = 12** — Vertices drawn per frame. 3D only.
- **RENDER_MATERIAL_CHANGES_IN_FRAME = 13** — Material changes per frame. 3D only
- **RENDER_SHADER_CHANGES_IN_FRAME = 14** — Shader changes per frame. 3D only.
- **RENDER_SURFACE_CHANGES_IN_FRAME = 15** — Render surface changes per frame. 3D only.
- **RENDER_DRAW_CALLS_IN_FRAME = 16** — Draw calls per frame. 3D only.
- **RENDER_VIDEO_MEM_USED = 17** — Video memory used. Includes both texture and vertex memory.
- **RENDER_TEXTURE_MEM_USED = 18** — Texture memory used.
- **RENDER_VERTEX_MEM_USED = 19** — Vertex memory used.
- **RENDER_USAGE_VIDEO_MEM_TOTAL = 20**
- **PHYSICS_2D_ACTIVE_OBJECTS = 21** — Number of active *RigidBody2D* nodes in the game.
- **PHYSICS_2D_COLLISION_PAIRS = 22** — Number of collision pairs in the 2D physics engine.
- **PHYSICS_2D_ISLAND_COUNT = 23** — Number of islands in the 2D physics engine.
- **PHYSICS_3D_ACTIVE_OBJECTS = 24** — Number of active *RigidBody* and *VehicleBody* nodes in the game.
- **PHYSICS_3D_COLLISION_PAIRS = 25** — Number of collision pairs in the 3D physics engine.
- **PHYSICS_3D_ISLAND_COUNT = 26** — Number of islands in the 3D physics engine.
- **MONITOR_MAX = 27**

25.269.4 Description

This class provides access to a number of different monitors related to performance, such as memory usage, draw calls, and FPS. These are the same as the values displayed in the *Monitor* tab in the editor's *Debugger* panel. By using the `get_monitor` method of this class, you can access this data from your code. Note that a few of these monitors are only available in debug mode and will always return 0 when used in a release build.

Many of these monitors are not updated in real-time, so there may be a short delay between changes.

25.269.5 Member Function Description

- `float get_monitor (int monitor) const`

Returns the value of one of the available monitors. You should provide one of this class's constants as the argument, like this:

```
print(Performance.get_monitor(Performance.TIME_FPS)) # Prints the FPS to the console
```

25.270 PHashTranslation

Inherits: *Translation < Resource < Reference < Object*

Category: Core

25.270.1 Brief Description

Optimized translation.

25.270.2 Member Functions

void	<i>generate (Translation from)</i>
------	--------------------------------------

25.270.3 Description

Optimized translation. Uses real-time compressed translations, which results in very small dictionaries.

25.270.4 Member Function Description

- void **generate** (*Translation from*)

25.271 Physics2DDirectBodyState

Inherits: *Object*

Inherited By: *Physics2DDirectBodyStateSW*

Category: Core

25.271.1 Brief Description

Direct access object to a physics body in the *Physics2DServer*.

25.271.2 Member Functions

<i>RID</i>	<code>get_contact.collider (int contact_idx) const</code>
<i>int</i>	<code>get_contact.collider_id (int contact_idx) const</code>
<i>Object</i>	<code>get_contact.collider_object (int contact_idx) const</code>
<i>Vector2</i>	<code>get_contact.collider_position (int contact_idx) const</code>
<i>int</i>	<code>get_contact.collider_shape (int contact_idx) const</code>
<i>Variant</i>	<code>get_contact.collider_shape_metadata (int contact_idx) const</code>
<i>Vector2</i>	<code>get_contact.collider_velocity_at_position (int contact_idx) const</code>
<i>int</i>	<code>get_contact_count () const</code>
<i>Vector2</i>	<code>get_contact_local_normal (int contact_idx) const</code>
<i>Vector2</i>	<code>get_contact_local_position (int contact_idx) const</code>
<i>int</i>	<code>get_contact_local_shape (int contact_idx) const</code>
<i>Physics2DDirectSpaceState</i>	<code>get_space_state ()</code>
<i>void</i>	<code>integrate_forces ()</code>

25.271.3 Member Variables

- *float angular_velocity* - The angular velocity of the body.
- *float inverse_inertia* - The inverse of the inertia of the body.
- *float inverse_mass* - The inverse of the mass of the body.
- *Vector2 linear_velocity* - The linear velocity of the body.
- *bool sleeping* - `true` if this body is currently sleeping (not active).
- *float step* - The timestep (delta) used for the simulation.
- *float total_angular_damp* - The rate at which the body stops rotating, if there are not any other forces moving it.
- *Vector2 total_gravity* - The total gravity vector being currently applied to this body.
- *float total_linear_damp* - The rate at which the body stops moving, if there are not any other forces moving it.
- *Transform2D transform* - The transformation matrix of the body.

25.271.4 Description

Direct access object to a physics body in the *Physics2DServer*. This object is passed via the direct state callback of rigid/character bodies, and is intended for changing the direct state of that body.

25.271.5 Member Function Description

- *RID get_contact.collider (int contact_idx) const*

Return the *RID* of the collider.

- *int get_contact.collider_id (int contact_idx) const*

Return the object id of the collider.

- *Object get_contact.collider_object (int contact_idx) const*

Return the collider object, this depends on how it was created (will return a scene node if such was used to create it).

- `Vector2 get_contact.collider_position (int contact_idx) const`

Return the contact position in the collider.

- `int get_contact.collider_shape (int contact_idx) const`

Return the collider shape index.

- `Variant get_contact.collider_shape_metadata (int contact_idx) const`

Return the metadata of the collided shape. This metadata is different from `Object.get_meta`, and is set with `Physics2DServer.shape_set_data`.

- `Vector2 get_contact.collider_velocity_at_position (int contact_idx) const`

Return the linear velocity vector at contact point of the collider.

- `int get_contact_count () const`

Return the amount of contacts this body has with other bodies. Note that by default this returns 0 unless bodies are configured to log contacts.

- `Vector2 get_contact_local_normal (int contact_idx) const`

Return the local normal (of this body) of the contact point.

- `Vector2 get_contact_local_position (int contact_idx) const`

Return the local position (of this body) of the contact point.

- `int get_contact_local_shape (int contact_idx) const`

Return the local shape index of the collision.

- `Physics2DDirectSpaceState get_space_state ()`

Return the current state of space, useful for queries.

- `void integrate_forces ()`

Call the built-in force integration code.

25.272 Physics2DDirectBodyStateSW

Inherits: `Physics2DDirectBodyState < Object`

Category: Core

25.272.1 Brief Description

Software implementation of `Physics2DDirectBodyState`.

25.272.2 Description

Software implementation of `Physics2DDirectBodyState`. This object exposes no new methods or properties and should not be used, as `Physics2DDirectBodyState` selects the best implementation available.

25.273 Physics2DDirectSpaceState

Inherits: *Object*

Category: Core

25.273.1 Brief Description

Direct access object to a space in the *Physics2DServer*.

25.273.2 Member Functions

Array	<code>cast_motion (Physics2DShapeQueryParameters shape)</code>
Array	<code>collide_shape (Physics2DShapeQueryParameters shape, int max_results=32)</code>
Dictionary	<code>get_rest_info (Physics2DShapeQueryParameters shape)</code>
Array	<code>intersect_point (Vector2 point, int max_results=32, Array exclude=[], int collision_layer=2147483647)</code>
Dictionary	<code>intersect_ray (Vector2 from, Vector2 to, Array exclude=[], int collision_layer=2147483647)</code>
Array	<code>intersect_shape (Physics2DShapeQueryParameters shape, int max_results=32)</code>

25.273.3 Description

Direct access object to a space in the *Physics2DServer*. It's used mainly to do queries against objects and areas residing in a given space.

25.273.4 Member Function Description

- **Array `cast_motion (Physics2DShapeQueryParameters shape)`**

Checks how far the shape can travel toward a point. Note that both the shape and the motion are supplied through a *Physics2DShapeQueryParameters* object. The method will return an array with two floats between 0 and 1, both representing a fraction of motion. The first is how far the shape can move without triggering a collision, and the second is the point at which a collision will occur. If no collision is detected, the returned array will be 1, 1.

If the shape can not move, the array will be empty (`dir.empty ()==true`).

- **Array `collide_shape (Physics2DShapeQueryParameters shape, int max_results=32)`**

Checks the intersections of a shape, given through a *Physics2DShapeQueryParameters* object, against the space. The resulting array contains a list of points where the shape intersects another. Like with *intersect_shape*, the number of returned results can be limited to save processing time.

- **Dictionary `get_rest_info (Physics2DShapeQueryParameters shape)`**

Checks the intersections of a shape, given through a *Physics2DShapeQueryParameters* object, against the space. If it collides with more than one shape, the nearest one is selected. Note that this method does not take into account the *motion* property of the object. The returned object is a dictionary containing the following fields:

`collider_id`: The colliding object's ID.

`linear_velocity`: The colliding object's velocity *Vector2*. If the object is an *Area2D*, the result is (0, 0).

`metadata`: The intersecting shape's metadata. This metadata is different from `Object.get_meta`, and is set with `Physics2DServer.shape_set_data`.

`normal`: The object's surface normal at the intersection point.

`point`: The intersection point.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

If the shape did not intersect anything, then an empty dictionary (`dir.empty() == true`) is returned instead.

- `Array intersect_point (Vector2 point, int max_results=32, Array exclude=[], int collision_layer=2147483647)`

Checks whether a point is inside any shape. The shapes the point is inside of are returned in an array containing dictionaries with the following fields:

`collider`: The colliding object.

`collider_id`: The colliding object's ID.

`metadata`: The intersecting shape's metadata. This metadata is different from `Object.get_meta`, and is set with `Physics2DServer.shape_set_data`.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

Additionally, the method can take an array of objects or *RIDs* that are to be excluded from collisions, or a bitmask representing the physics layers to check in.

- `Dictionary intersect_ray (Vector2 from, Vector2 to, Array exclude=[], int collision_layer=2147483647)`

Intersects a ray in a given space. The returned object is a dictionary with the following fields:

`collider`: The colliding object.

`collider_id`: The colliding object's ID.

`metadata`: The intersecting shape's metadata. This metadata is different from `Object.get_meta`, and is set with `Physics2DServer.shape_set_data`.

`normal`: The object's surface normal at the intersection point.

`position`: The intersection point.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

If the ray did not intersect anything, then an empty dictionary (`dir.empty() == true`) is returned instead.

Additionally, the method can take an array of objects or *RIDs* that are to be excluded from collisions, or a bitmask representing the physics layers to check in.

- `Array intersect_shape (Physics2DShapeQueryParameters shape, int max_results=32)`

Checks the intersections of a shape, given through a `Physics2DShapeQueryParameters` object, against the space. Note that this method does not take into account the `motion` property of the object. The intersected shapes are returned in an array containing dictionaries with the following fields:

`collider`: The colliding object.

`collider_id`: The colliding object's ID.

`metadata`: The intersecting shape's metadata. This metadata is different from `Object.get_meta`, and is set with `Physics2DServer.shape_set_data`.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

The number of intersections can be limited with the second parameter, to reduce the processing time.

25.274 Physics2DServer

Inherits: `Object`

Inherited By: `Physics2DServerSW`

Category: Core

25.274.1 Brief Description

Physics 2D Server.

25.274.2 Member Functions

void	<code>area_add_shape (RID area, RID shape, Transform2D transform=Transform2D(1, 0, 0, 1, 0, 0))</code>
void	<code>area_attach_object_instance_id (RID area, int id)</code>
void	<code>area_clear_shapes (RID area)</code>
<i>RID</i>	<code>area_create ()</code>
<i>int</i>	<code>area_get_object_instance_id (RID area) const</code>
<i>Variant</i>	<code>area_get_param (RID area, int param) const</code>
<i>RID</i>	<code>area_get_shape (RID area, int shape_idx) const</code>
<i>int</i>	<code>area_get_shape_count (RID area) const</code>
<i>Transform2D</i>	<code>area_get_shape_transform (RID area, int shape_idx) const</code>
<i>RID</i>	<code>area_get_space (RID area) const</code>
<i>int</i>	<code>area_get_space_override_mode (RID area) const</code>
<i>Transform2D</i>	<code>area_get_transform (RID area) const</code>
void	<code>area_remove_shape (RID area, int shape_idx)</code>
void	<code>area_set_collision_layer (RID area, int layer)</code>
void	<code>area_set_collision_mask (RID area, int mask)</code>
void	<code>area_set_monitor_callback (RID area, Object receiver, String method)</code>
void	<code>area_set_param (RID area, int param, Variant value)</code>
void	<code>area_set_shape (RID area, int shape_idx, RID shape)</code>
void	<code>area_set_shape_disabled (RID area, int shape_idx, bool disable)</code>
void	<code>area_set_shape_transform (RID area, int shape_idx, Transform2D transform)</code>
void	<code>area_set_space (RID area, RID space)</code>
void	<code>area_set_space_override_mode (RID area, int mode)</code>
void	<code>area_set_transform (RID area, Transform2D transform)</code>
void	<code>body_add_collision_exception (RID body, RID excepted_body)</code>
void	<code>body_add_force (RID body, Vector2 offset, Vector2 force)</code>
void	<code>body_add_shape (RID body, RID shape, Transform2D transform=Transform2D(1, 0, 0, 1, 0, 0))</code>
void	<code>body_apply_impulse (RID body, Vector2 position, Vector2 impulse)</code>

Continued on next page

Table 25.17 – continued from previous page

void	<i>body_attach_object_instance_id</i> (<i>RID body</i> , <i>int id</i>)
void	<i>body_clear_shapes</i> (<i>RID body</i>)
<i>RID</i>	<i>body_create</i> ()
<i>int</i>	<i>body_get_collision_layer</i> (<i>RID body</i>) const
<i>int</i>	<i>body_get_collision_mask</i> (<i>RID body</i>) const
<i>int</i>	<i>body_get_continuous_collision_detection_mode</i> (<i>RID body</i>) const
<i>Physics2DDirectBodyState</i>	<i>body_get_direct_state</i> (<i>RID body</i>)
<i>int</i>	<i>body_get_max_contacts_reported</i> (<i>RID body</i>) const
<i>int</i>	<i>body_get_mode</i> (<i>RID body</i>) const
<i>int</i>	<i>body_get_object_instance_id</i> (<i>RID body</i>) const
<i>float</i>	<i>body_get_param</i> (<i>RID body</i> , <i>int param</i>) const
<i>RID</i>	<i>body_get_shape</i> (<i>RID body</i> , <i>int shape_idx</i>) const
<i>int</i>	<i>body_get_shape_count</i> (<i>RID body</i>) const
<i>Variant</i>	<i>body_get_shape_metadata</i> (<i>RID body</i> , <i>int shape_idx</i>) const
<i>Transform2D</i>	<i>body_get_shape_transform</i> (<i>RID body</i> , <i>int shape_idx</i>) const
<i>RID</i>	<i>body_get_space</i> (<i>RID body</i>) const
<i>Variant</i>	<i>body_get_state</i> (<i>RID body</i> , <i>int state</i>) const
<i>bool</i>	<i>body_is_omitting_force_integration</i> (<i>RID body</i>) const
void	<i>body_remove_collision_exception</i> (<i>RID body</i> , <i>RID excepted_body</i>)
void	<i>body_remove_shape</i> (<i>RID body</i> , <i>int shape_idx</i>)
void	<i>body_set_axis_velocity</i> (<i>RID body</i> , <i>Vector2 axis_velocity</i>)
void	<i>body_set_collision_layer</i> (<i>RID body</i> , <i>int layer</i>)
void	<i>body_set_collision_mask</i> (<i>RID body</i> , <i>int mask</i>)
void	<i>body_set_continuous_collision_detection_mode</i> (<i>RID body</i> , <i>int mode</i>)
void	<i>body_set_force_integration_callback</i> (<i>RID body</i> , <i>Object receiver</i> , <i>String method</i> , <i>Variant userdata=null</i>)
void	<i>body_set_max_contacts_reported</i> (<i>RID body</i> , <i>int amount</i>)
void	<i>body_set_mode</i> (<i>RID body</i> , <i>int mode</i>)
void	<i>body_set OMIT force_integration</i> (<i>RID body</i> , <i>bool enable</i>)
void	<i>body_set_param</i> (<i>RID body</i> , <i>int param</i> , <i>float value</i>)
void	<i>body_set_shape</i> (<i>RID body</i> , <i>int shape_idx</i> , <i>RID shape</i>)
void	<i>body_set_shape_as_one_way_collision</i> (<i>RID body</i> , <i>int shape_idx</i> , <i>bool enable</i>)
void	<i>body_set_shape_disabled</i> (<i>RID body</i> , <i>int shape_idx</i> , <i>bool disable</i>)
void	<i>body_set_shape_metadata</i> (<i>RID body</i> , <i>int shape_idx</i> , <i>Variant metadata</i>)
void	<i>body_set_shape_transform</i> (<i>RID body</i> , <i>int shape_idx</i> , <i>Transform2D transform</i>)
void	<i>body_set_space</i> (<i>RID body</i> , <i>RID space</i>)
void	<i>body_set_state</i> (<i>RID body</i> , <i>int state</i> , <i>Variant value</i>)
<i>bool</i>	<i>body_test_motion</i> (<i>RID body</i> , <i>Transform2D from</i> , <i>Vector2 motion</i> , <i>float margin=0.08</i> , <i>Physics2DTestMode mode</i>)
<i>RID</i>	<i>capsule_shape_create</i> ()
<i>RID</i>	<i>circle_shape_create</i> ()
<i>RID</i>	<i>concave_polygon_shape_create</i> ()
<i>RID</i>	<i>convex_polygon_shape_create</i> ()
<i>RID</i>	<i>damped_spring_joint_create</i> (<i>Vector2 anchor_a</i> , <i>Vector2 anchor_b</i> , <i>RID body_a</i> , <i>RID body_b</i>)
<i>float</i>	<i>damped_string_joint_get_param</i> (<i>RID joint</i> , <i>int param</i>) const
void	<i>damped_string_joint_set_param</i> (<i>RID joint</i> , <i>int param</i> , <i>float value</i>)
void	<i>free_rid</i> (<i>RID rid</i>)
<i>int</i>	<i>get_process_info</i> (<i>int process_info</i>)
<i>RID</i>	<i>groove_joint_create</i> (<i>Vector2 groove1_a</i> , <i>Vector2 groove2_a</i> , <i>Vector2 anchor_b</i> , <i>RID body_a</i> , <i>RID body_b</i>)
<i>float</i>	<i>joint_get_param</i> (<i>RID joint</i> , <i>int param</i>) const
<i>int</i>	<i>joint_get_type</i> (<i>RID joint</i>) const
void	<i>joint_set_param</i> (<i>RID joint</i> , <i>int param</i> , <i>float value</i>)

Continued on next page

Table 25.17 – continued from previous page

<i>RID</i>	<i>line_shape_create ()</i>
<i>RID</i>	<i>pin_joint_create (Vector2 anchor, RID body_a, RID body_b)</i>
<i>RID</i>	<i>ray_shape_create ()</i>
<i>RID</i>	<i>rectangle_shape_create ()</i>
<i>RID</i>	<i>segment_shape_create ()</i>
void	<i>set_active (bool active)</i>
<i>Variant</i>	<i>shape_get_data (RID shape) const</i>
<i>int</i>	<i>shape_get_type (RID shape) const</i>
void	<i>shape_set_data (RID shape, Variant data)</i>
<i>RID</i>	<i>space_create ()</i>
<i>Physics2DDirectSpaceState</i>	<i>space_get_direct_state (RID space)</i>
<i>float</i>	<i>space_get_param (RID space, int param) const</i>
<i>bool</i>	<i>space_is_active (RID space) const</i>
void	<i>space_set_active (RID space, bool active)</i>
void	<i>space_set_param (RID space, int param, float value)</i>

25.274.3 Enums

enum **CCDMode**

- **CCD_MODE_DISABLED = 0** — Disables continuous collision detection. This is the fastest way to detect body collisions, but can miss small, fast-moving objects.
- **CCD_MODE_CAST_RAY = 1** — Enables continuous collision detection by raycasting. It is faster than shapecasting, but less precise.
- **CCD_MODE_CAST_SHAPE = 2** — Enables continuous collision detection by shapecasting. It is the slowest CCD method, and the most precise.

enum **BodyState**

- **BODY_STATE_TRANSFORM = 0** — Constant to set/get the current transform matrix of the body.
- **BODY_STATE_LINEAR_VELOCITY = 1** — Constant to set/get the current linear velocity of the body.
- **BODY_STATE_ANGULAR_VELOCITY = 2** — Constant to set/get the current angular velocity of the body.
- **BODY_STATE_SLEEPING = 3** — Constant to sleep/wake up a body, or to get whether it is sleeping.
- **BODY_STATE_CAN_SLEEP = 4** — Constant to set/get whether the body can sleep.

enum **ProcessInfo**

- **INFO_ACTIVE_OBJECTS = 0** — Constant to get the number of objects that are not sleeping.
- **INFO_COLLISION_PAIRS = 1** — Constant to get the number of possible collisions.
- **INFO_ISLAND_COUNT = 2** — Constant to get the number of space regions where a collision could occur.

enum **JointParam**

- **JOINT_PARAM_BIAS = 0**
- **JOINT_PARAM_MAX_BIAS = 1**
- **JOINT_PARAM_MAX_FORCE = 2**

enum **ShapeType**

- **SHAPE_LINE = 0** — This is the constant for creating line shapes. A line shape is an infinite line with an origin point, and a normal. Thus, it can be used for front/behind checks.
- **SHAPE_RAY = 1**
- **SHAPE_SEGMENT = 2** — This is the constant for creating segment shapes. A segment shape is a line from a point A to a point B. It can be checked for intersections.
- **SHAPE_CIRCLE = 3** — This is the constant for creating circle shapes. A circle shape only has a radius. It can be used for intersections and inside/outside checks.
- **SHAPE_RECTANGLE = 4** — This is the constant for creating rectangle shapes. A rectangle shape is defined by a width and a height. It can be used for intersections and inside/outside checks.
- **SHAPE_CAPSULE = 5** — This is the constant for creating capsule shapes. A capsule shape is defined by a radius and a length. It can be used for intersections and inside/outside checks.
- **SHAPE_CONVEX_POLYGON = 6** — This is the constant for creating convex polygon shapes. A polygon is defined by a list of points. It can be used for intersections and inside/outside checks. Unlike the method CollisionPolygon2D.set_polygon, polygons modified with *shape_set_data* do not verify that the points supplied form is a convex polygon.
- **SHAPE_CONCAVE_POLYGON = 7** — This is the constant for creating concave polygon shapes. A polygon is defined by a list of points. It can be used for intersections checks, but not for inside/outside checks.
- **SHAPE_CUSTOM = 8** — This constant is used internally by the engine. Any attempt to create this kind of shape results in an error.

enum AreaParameter

- **AREA_PARAM_GRAVITY = 0** — Constant to set/get gravity strength in an area.
- **AREA_PARAM_GRAVITY_VECTOR = 1** — Constant to set/get gravity vector/center in an area.
- **AREA_PARAM_GRAVITY_IS_POINT = 2** — Constant to set/get whether the gravity vector of an area is a direction, or a center point.
- **AREA_PARAM_GRAVITY_DISTANCE_SCALE = 3** — Constant to set/get the falloff factor for point gravity of an area. The greater this value is, the faster the strength of gravity decreases with the square of distance.
- **AREA_PARAM_GRAVITY_POINT_ATTENUATION = 4** — This constant was used to set/get the falloff factor for point gravity. It has been superseded by AREA_PARAM_GRAVITY_DISTANCE_SCALE.
- **AREA_PARAM_LINEAR_DAMP = 5** — Constant to set/get the linear dampening factor of an area.
- **AREA_PARAM_ANGULAR_DAMP = 6** — Constant to set/get the angular dampening factor of an area.
- **AREA_PARAM_PRIORITY = 7** — Constant to set/get the priority (order of processing) of an area.

enum AreaBodyStatus

- **AREA_BODY_ADDED = 0** — The value of the first parameter and area callback function receives, when an object enters one of its shapes.
- **AREA_BODY_REMOVED = 1** — The value of the first parameter and area callback function receives, when an object exits one of its shapes.

enum BodyParameter

- **BODY_PARAM_BOUNCE = 0** — Constant to set/get a body's bounce factor.
- **BODY_PARAM_FRICTION = 1** — Constant to set/get a body's friction.
- **BODY_PARAM_MASS = 2** — Constant to set/get a body's mass.
- **BODY_PARAM_INERTIA = 3** — Constant to set/get a body's inertia.

- **BODY_PARAM_GRAVITY_SCALE = 4** — Constant to set/get a body's gravity multiplier.
- **BODY_PARAM_LINEAR_DAMP = 5** — Constant to set/get a body's linear dampening factor.
- **BODY_PARAM_ANGULAR_DAMP = 6** — Constant to set/get a body's angular dampening factor.
- **BODY_PARAM_MAX = 7** — This is the last ID for body parameters. Any attempt to set this property is ignored. Any attempt to get it returns 0.

enum **BodyMode**

- **BODY_MODE_STATIC = 0** — Constant for static bodies.
- **BODY_MODE_KINEMATIC = 1** — Constant for kinematic bodies.
- **BODY_MODE_RIGID = 2** — Constant for rigid bodies.
- **BODY_MODE_CHARACTER = 3** — Constant for rigid bodies in character mode. In this mode, a body can not rotate, and only its linear velocity is affected by physics.

enum **DampedStringParam**

- **DAMPED_STRING_REST_LENGTH = 0** — Set the resting length of the spring joint. The joint will always try to go back this length when pulled apart.
- **DAMPED_STRING_STIFFNESS = 1** — Set the stiffness of the spring joint. The joint applies a force equal to the stiffness times the distance from its resting length.
- **DAMPED_STRING_DAMPING = 2** — Set the damping ratio of the spring joint. A value of 0 indicates an undamped spring, while 1 causes the system to reach equilibrium as fast as possible (critical damping).

enum **SpaceParameter**

- **SPACE_PARAM_CONTACT_RECYCLE_RADIUS = 0** — Constant to set/get the maximum distance a pair of bodies has to move before their collision status has to be recalculated.
- **SPACE_PARAM_CONTACT_MAX_SEPARATION = 1** — Constant to set/get the maximum distance a shape can be from another before they are considered separated.
- **SPACE_PARAM_BODY_MAX_ALLOWED_PENETRATION = 2** — Constant to set/get the maximum distance a shape can penetrate another shape before it is considered a collision.
- **SPACE_PARAM_BODY_LINEAR_VELOCITY_SLEEP_THRESHOLD = 3** — Constant to set/get the threshold linear velocity of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after the time given.
- **SPACE_PARAM_BODY_ANGULAR_VELOCITY_SLEEP_THRESHOLD = 4** — Constant to set/get the threshold angular velocity of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after the time given.
- **SPACE_PARAM_BODY_TIME_TO_SLEEP = 5** — Constant to set/get the maximum time of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after this time.
- **SPACE_PARAM_CONSTRAINT_DEFAULT_BIAS = 6** — Constant to set/get the default solver bias for all physics constraints. A solver bias is a factor controlling how much two objects "rebound", after violating a constraint, to avoid leaving them in that state because of numerical imprecision.

enum **AreaSpaceOverrideMode**

- **AREA_SPACE_OVERRIDE_DISABLED = 0** — This area does not affect gravity/damp. These are generally areas that exist only to detect collisions, and objects entering or exiting them.
- **AREA_SPACE_OVERRIDE_COMBINE = 1** — This area adds its gravity/damp values to whatever has been calculated so far. This way, many overlapping areas can combine their physics to make interesting effects.

- **AREA_SPACE_OVERRIDE_COMBINE_REPLACE = 2** — This area adds its gravity/damp values to whatever has been calculated so far. Then stops taking into account the rest of the areas, even the default one.
- **AREA_SPACE_OVERRIDE_REPLACE = 3** — This area replaces any gravity/damp, even the default one, and stops taking into account the rest of the areas.
- **AREA_SPACE_OVERRIDE_REPLACE_COMBINE = 4** — This area replaces any gravity/damp calculated so far, but keeps calculating the rest of the areas, down to the default one.

enum **JointType**

- **JOINT_PIN = 0** — Constant to create pin joints.
- **JOINT_GROOVE = 1** — Constant to create groove joints.
- **JOINT_DAMPED_SPRING = 2** — Constant to create damped spring joints.

25.274.4 Description

Physics 2D Server is the server responsible for all 2D physics. It can create many kinds of physics objects, but does not insert them on the node tree.

25.274.5 Member Function Description

- void **area_add_shape** (*RID* area, *RID* shape, *Transform2D* transform=*Transform2D*(1, 0, 0, 1, 0, 0))

Adds a shape to the area, along with a transform matrix. Shapes are usually referenced by their index, so you should track which shape has a given index.

- void **area_attach_object_instance_id** (*RID* area, *int* id)

Assigns the area to a descendant of *Object*, so it can exist in the node tree.

- void **area_clear_shapes** (*RID* area)

Removes all shapes from an area. It does not delete the shapes, so they can be reassigned later.

- *RID* **area_create** ()

Creates an *Area2D*.

- *int* **area_get_object_instance_id** (*RID* area) const

Gets the instance ID of the object the area is assigned to.

- *Variant* **area_get_param** (*RID* area, *int* param) const

Returns an area parameter value. A list of available parameters is on the AREA_PARAM_* constants.

- *RID* **area_get_shape** (*RID* area, *int* shape_idx) const

Returns the *RID* of the nth shape of an area.

- *int* **area_get_shape_count** (*RID* area) const

Returns the number of shapes assigned to an area.

- *Transform2D* **area_get_shape_transform** (*RID* area, *int* shape_idx) const

Returns the transform matrix of a shape within an area.

- *RID* **area_get_space** (*RID* area) const

Returns the space assigned to the area.

- `int area_get_space_override_mode (RID area) const`

Returns the space override mode for the area.

- `Transform2D area_get_transform (RID area) const`

Returns the transform matrix for an area.

- `void area_remove_shape (RID area, int shape_idx)`

Removes a shape from an area. It does not delete the shape, so it can be reassigned later.

- `void area_set_collision_layer (RID area, int layer)`

Assigns the area to one or many physics layers.

- `void area_set_collision_mask (RID area, int mask)`

Sets which physics layers the area will monitor.

- `void area_set_monitor_callback (RID area, Object receiver, String method)`

Sets the function to call when any body/area enters or exits the area. This callback will be called for any object interacting with the area, and takes five parameters:

1: AREA_BODY_ADDED or AREA_BODY_REMOVED, depending on whether the object entered or exited the area.

2: *RID* of the object that entered/exited the area.

3: Instance ID of the object that entered/exited the area.

4: The shape index of the object that entered/exited the area.

5: The shape index of the area where the object entered/exited.

- `void area_set_param (RID area, int param, Variant value)`

Sets the value for an area parameter. A list of available parameters is on the AREA_PARAM_* constants.

- `void area_set_shape (RID area, int shape_idx, RID shape)`

Substitutes a given area shape by another. The old shape is selected by its index, the new one by its *RID*.

- `void area_set_shape_disabled (RID area, int shape_idx, bool disable)`

Disables a given shape in an area.

- `void area_set_shape_transform (RID area, int shape_idx, Transform2D transform)`

Sets the transform matrix for an area shape.

- `void area_set_space (RID area, RID space)`

Assigns a space to the area.

- `void area_set_space_override_mode (RID area, int mode)`

Sets the space override mode for the area. The modes are described in the constants AREA_SPACE_OVERRIDE_*.

- `void area_set_transform (RID area, Transform2D transform)`

Sets the transform matrix for an area.

- `void body_add_collision_exception (RID body, RID excepted_body)`

Adds a body to the list of bodies exempt from collisions.

- `void body_add_force (RID body, Vector2 offset, Vector2 force)`

Adds a positioned force to the applied force and torque. As with `body_apply_impulse`, both the force and the offset from the body origin are in global coordinates. A force differs from an impulse in that, while the two are forces, the impulse clears itself after being applied.

- `void body_add_shape (RID body, RID shape, Transform2D transform=Transform2D(1, 0, 0, 1, 0, 0))`

Adds a shape to the body, along with a transform matrix. Shapes are usually referenced by their index, so you should track which shape has a given index.

- `void body_apply_impulse (RID body, Vector2 position, Vector2 impulse)`

Adds a positioned impulse to the applied force and torque. Both the force and the offset from the body origin are in global coordinates.

- `void body_attach_object_instance_id (RID body, int id)`

Assigns the area to a descendant of `Object`, so it can exist in the node tree.

- `void body_clear_shapes (RID body)`

Removes all shapes from a body.

- `RID body_create ()`

Creates a physics body. The first parameter can be any value from constants BODY_MODE*, for the type of body created. Additionally, the body can be created in sleeping state to save processing time.

- `int body_get_collision_layer (RID body) const`

Returns the physics layer or layers a body belongs to.

- `int body_get_collision_mask (RID body) const`

Returns the physics layer or layers a body can collide with.

- `int body_get_continuous_collision_detection_mode (RID body) const`

Returns the continuous collision detection mode.

- `Physics2DDirectBodyState body_get_direct_state (RID body)`

Returns the `Physics2DDirectBodyState` of the body.

- `int body_get_max_contacts_reported (RID body) const`

Returns the maximum contacts that can be reported. See `body_set_max_contacts_reported`.

- `int body_get_mode (RID body) const`

Returns the body mode.

- `int body_get_object_instance_id (RID body) const`

Gets the instance ID of the object the area is assigned to.

- `float body_get_param (RID body, int param) const`

Returns the value of a body parameter. A list of available parameters is on the BODY_PARAM_* constants.

- `RID body_get_shape (RID body, int shape_idx) const`

Returns the `RID` of the nth shape of a body.

- `int body_get_shape_count (RID body) const`

Returns the number of shapes assigned to a body.

- `Variant body_get_shape_metadata (RID body, int shape_idx) const`

Returns the metadata of a shape of a body.

- *Transform2D* **body_get_shape_transform** (*RID* body, *int* shape_idx) const

Returns the transform matrix of a body shape.

- *RID* **body_get_space** (*RID* body) const

Returns the *RID* of the space assigned to a body.

- *Variant* **body_get_state** (*RID* body, *int* state) const

Returns a body state.

- *bool* **body_is_omitting_force_integration** (*RID* body) const

Returns whether a body uses a callback function to calculate its own physics (see *body_set_force_integration_callback*).

- *void* **body_remove_collision_exception** (*RID* body, *RID* excepted_body)

Removes a body from the list of bodies exempt from collisions.

- *void* **body_remove_shape** (*RID* body, *int* shape_idx)

Removes a shape from a body. The shape is not deleted, so it can be reused afterwards.

- *void* **body_set_axis_velocity** (*RID* body, *Vector2* axis_velocity)

Sets an axis velocity. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

- *void* **body_set_collision_layer** (*RID* body, *int* layer)

Sets the physics layer or layers a body belongs to.

- *void* **body_set_collision_mask** (*RID* body, *int* mask)

Sets the physics layer or layers a body can collide with.

- *void* **body_set_continuous_collision_detection_mode** (*RID* body, *int* mode)

Sets the continuous collision detection mode from any of the CCD_MODE_* constants.

Continuous collision detection tries to predict where a moving body will collide, instead of moving it and correcting its movement if it collided.

- *void* **body_set_force_integration_callback** (*RID* body, *Object* receiver, *String* method, *Variant* userdata=null)

Sets the function used to calculate physics for an object, if that object allows it (see *body_set OMIT force integration*).

- *void* **body_set_max_contacts_reported** (*RID* body, *int* amount)

Sets the maximum contacts to report. Bodies can keep a log of the contacts with other bodies, this is enabled by setting the maximum amount of contacts reported to a number greater than 0.

- *void* **body_set_mode** (*RID* body, *int* mode)

Sets the body mode, from one of the constants BODY_MODE*.

- *void* **body_set OMIT force integration** (*RID* body, *bool* enable)

Sets whether a body uses a callback function to calculate its own physics (see *body_set_force_integration_callback*).

- *void* **body_set_param** (*RID* body, *int* param, *float* value)

Sets a body parameter. A list of available parameters is on the BODY_PARAM_* constants.

- *void* **body_set_shape** (*RID* body, *int* shape_idx, *RID* shape)

Substitutes a given body shape by another. The old shape is selected by its index, the new one by its *RID*.

- void **body_set_shape_as_one_way_collision** (*RID body, int shape_idx, bool enable*)

Enables one way collision on body if *enable* is `true`.

- void **body_set_shape_disabled** (*RID body, int shape_idx, bool disable*)

Disables shape in body if *disable* is `true`.

- void **body_set_shape_metadata** (*RID body, int shape_idx, Variant metadata*)

Sets metadata of a shape within a body. This metadata is different from *Object.set_meta*, and can be retrieved on shape queries.

- void **body_set_shape_transform** (*RID body, int shape_idx, Transform2D transform*)

Sets the transform matrix for a body shape.

- void **body_set_space** (*RID body, RID space*)

Assigns a space to the body (see *create_space*).

- void **body_set_state** (*RID body, int state, Variant value*)

Sets a body state (see *BODY_STATE** constants).

- bool **body_test_motion** (*RID body, Transform2D from, Vector2 motion, float margin=0.08, Physics2DTestMotionResult result=null*)

Returns whether a body can move from a given point in a given direction. Apart from the boolean return value, a *Physics2DTestMotionResult* can be passed to return additional information in.

- *RID capsule_shape_create* ()

- *RID circle_shape_create* ()

- *RID concave_polygon_shape_create* ()

- *RID convex_polygon_shape_create* ()

- *RID damped_spring_joint_create* (*Vector2 anchor_a, Vector2 anchor_b, RID body_a, RID body_b*)

Creates a damped spring joint between two bodies. If not specified, the second body is assumed to be the joint itself.

- *float damped_string_joint_get_param* (*RID joint, int param*) const

Returns the value of a damped spring joint parameter.

- void **damped_string_joint_set_param** (*RID joint, int param, float value*)

Sets a damped spring joint parameter. Parameters are explained in the *DAMPED_STRING** constants.

- void **free_rid** (*RID rid*)

Destroys any of the objects created by Physics2DServer. If the *RID* passed is not one of the objects that can be created by Physics2DServer, an error will be sent to the console.

- *int get_process_info* (*int process_info*)

Returns information about the current state of the 2D physics engine. The states are listed under the *INFO_** constants.

- *RID groove_joint_create* (*Vector2 groove1_a, Vector2 groove2_a, Vector2 anchor_b, RID body_a, RID body_b*)

Creates a groove joint between two bodies. If not specified, the bodies are assumed to be the joint itself.

- *float joint_get_param* (*RID joint, int param*) const

Returns the value of a joint parameter.

- *int joint_get_type* (*RID joint*) const

Returns the type of a joint (see JOINT_* constants).

- `void joint_set_param (RID joint, int param, float value)`

Sets a joint parameter. Parameters are explained in the JOINT_PARAM* constants.

- `RID line_shape_create ()`
- `RID pin_joint_create (Vector2 anchor, RID body_a, RID body_b)`

Creates a pin joint between two bodies. If not specified, the second body is assumed to be the joint itself.

- `RID ray_shape_create ()`
- `RID rectangle_shape_create ()`
- `RID segment_shape_create ()`
- `void set_active (bool active)`

Activates or deactivates the 2D physics engine.

- `Variant shape_get_data (RID shape) const`

Returns the shape data.

- `int shape_get_type (RID shape) const`

Returns the type of shape (see SHAPE_* constants).

- `void shape_set_data (RID shape, Variant data)`

Sets the shape data that defines its shape and size. The data to be passed depends on the kind of shape created `shape_get_type`.

- `RID space_create ()`

Creates a space. A space is a collection of parameters for the physics engine that can be assigned to an area or a body. It can be assigned to an area with `area_set_space`, or to a body with `body_set_space`.

- `Physics2DDirectSpaceState space_get_direct_state (RID space)`

Returns the state of a space, a `Physics2DDirectSpaceState`. This object can be used to make collision/intersection queries.

- `float space_get_param (RID space, int param) const`

Returns the value of a space parameter.

- `bool space_is_active (RID space) const`

Returns whether the space is active.

- `void space_set_active (RID space, bool active)`

Marks a space as active. It will not have an effect, unless it is assigned to an area or body.

- `void space_set_param (RID space, int param, float value)`

Sets the value for a space parameter. A list of available parameters is on the SPACE_PARAM_* constants.

25.275 Physics2DServerSW

Inherits: `Physics2DServer < Object`

Category: Core

25.275.1 Brief Description

Software implementation of *Physics2DServer*.

25.275.2 Description

This class exposes no new methods or properties and should not be used, as *Physics2DServer* automatically selects the best implementation available.

25.276 Physics2DShapeQueryParameters

Inherits: *Reference < Object*

Category: Core

25.276.1 Brief Description

Parameters to be sent to a 2D shape physics query.

25.276.2 Member Functions

void	<i>set_shape (Resource shape)</i>
------	-------------------------------------

25.276.3 Member Variables

- *int collision_layer* - The physics layer the query should be made on.
- *Array exclude* - The list of objects or object *RIDs*, that will be excluded from collisions.
- *float margin* - The collision margin for the shape.
- *Vector2 motion* - The motion of the shape being queried for.
- *RID shape_rid* - The *RID* of the queried shape. See *set_shape* also.
- *Transform2D transform* - the transform matrix of the queried shape.

25.276.4 Description

This class contains the shape and other parameters for intersection/collision queries.

25.276.5 Member Function Description

- void *set_shape (Resource shape)*

Set the *Shape2D* that will be used for collision/intersection queries.

25.277 Physics2DShapeQueryResult

Inherits: *Reference < Object*

Category: Core

25.277.1 Brief Description

25.277.2 Member Functions

<i>int</i>	<code>get_result_count () const</code>
<i>Object</i>	<code>get_result_object (int idx) const</code>
<i>int</i>	<code>get_result_object_id (int idx) const</code>
<i>int</i>	<code>get_result_object_shape (int idx) const</code>
<i>RID</i>	<code>get_result_rid (int idx) const</code>

25.277.3 Member Function Description

- `int get_result_count () const`
- `Object get_result_object (int idx) const`
- `int get_result_object_id (int idx) const`
- `int get_result_object_shape (int idx) const`
- `RID get_result_rid (int idx) const`

25.278 Physics2DTestMotionResult

Inherits: *Reference < Object*

Category: Core

25.278.1 Brief Description

25.278.2 Member Variables

- `Object collider`
- `int collider_id`
- `RID collider_rid`
- `int collider_shape`
- `Vector2 collider_velocity`
- `Vector2 collision_normal`
- `Vector2 collision_point`

- *Vector2 motion*
- *Vector2 motion_remainder*

25.279 PhysicsBody

Inherits: *CollisionObject < Spatial < Node < Object*

Inherited By: *StaticBody, KinematicBody, RigidBody*

Category: Core

25.279.1 Brief Description

Base class for all objects affected by physics in 3D space.

25.279.2 Member Functions

void	<i>add_collision_exception_with (Node body)</i>
bool	<i>get_collision_layer_bit (int bit) const</i>
bool	<i>get_collision_mask_bit (int bit) const</i>
void	<i>remove_collision_exception_with (Node body)</i>
void	<i>set_collision_layer_bit (int bit, bool value)</i>
void	<i>set_collision_mask_bit (int bit, bool value)</i>

25.279.3 Member Variables

- *int collision_layer* - The physics layers this area is in.

Collidable objects can exist in any of 32 different layers. These layers work like a tagging system, and are not visual. A collidable can use these layers to select with which objects it can collide, using the collision_mask property.

A contact is detected if object A is in any of the layers that object B scans, or object B is in any layer scanned by object A.

- *int collision_mask* - The physics layers this area can scan for collisions.

25.279.4 Description

PhysicsBody is an abstract base class for implementing a physics body. All *Body types inherit from it.

25.279.5 Member Function Description

- void **add_collision_exception_with (Node body)**

Adds a body to the list of bodies that this body can't collide with.

- bool **get_collision_layer_bit (int bit) const**

- `bool get_collision_mask_bit (int bit) const`
- `void remove_collision_exception_with (Node body)`

Removes a body from the list of bodies that this body can't collide with.

- `void set_collision_layer_bit (int bit, bool value)`
- `void set_collision_mask_bit (int bit, bool value)`

25.280 PhysicsBody2D

Inherits: `CollisionObject2D < Node2D < CanvasItem < Node < Object`

Inherited By: `RigidBody2D, StaticBody2D, KinematicBody2D`

Category: Core

25.280.1 Brief Description

Base class for all objects affected by physics in 2D space.

25.280.2 Member Functions

<code>void</code>	<code>add_collision_exception_with (Node body)</code>
<code>bool</code>	<code>get_collision_layer_bit (int bit) const</code>
<code>bool</code>	<code>get_collision_mask_bit (int bit) const</code>
<code>void</code>	<code>remove_collision_exception_with (Node body)</code>
<code>void</code>	<code>set_collision_layer_bit (int bit, bool value)</code>
<code>void</code>	<code>set_collision_mask_bit (int bit, bool value)</code>

25.280.3 Member Variables

- `int collision_layer` - The physics layers this area is in.

Collidable objects can exist in any of 32 different layers. These layers work like a tagging system, and are not visual. A collidable can use these layers to select with which objects it can collide, using the `collision_mask` property.

A contact is detected if object A is in any of the layers that object B scans, or object B is in any layer scanned by object A.

- `int collision_mask` - The physics layers this area can scan for collisions.
- `int layers` - Both `collision_layer` and `collision_mask`. Returns `collision_layer` when accessed. Updates `collision_layers` and `collision_mask` when modified.

25.280.4 Description

PhysicsBody2D is an abstract base class for implementing a physics body. All *Body2D types inherit from it.

25.280.5 Member Function Description

- void **add_collision_exception_with** (*Node* body)

Adds a body to the list of bodies that this body can't collide with.

- *bool* **get_collision_layer_bit** (*int* bit) const

Return an individual bit on the collision mask.

- *bool* **get_collision_mask_bit** (*int* bit) const

Return an individual bit on the collision mask.

- void **remove_collision_exception_with** (*Node* body)

Removes a body from the list of bodies that this body can't collide with.

- void **set_collision_layer_bit** (*int* bit, *bool* value)

Set/clear individual bits on the layer mask. This makes getting a body in/out of only one layer easier.

- void **set_collision_mask_bit** (*int* bit, *bool* value)

Set/clear individual bits on the collision mask. This makes selecting the areas scanned easier.

25.281 PhysicsDirectBodyState

Inherits: *Object*

Inherited By: *BulletPhysicsDirectBodyState*

Category: Core

25.281.1 Brief Description

25.281.2 Member Functions

void	<i>add_force</i> (<i>Vector3</i> force, <i>Vector3</i> position)
void	<i>apply_impulse</i> (<i>Vector3</i> position, <i>Vector3</i> j)
void	<i>apply_torque_impulse</i> (<i>Vector3</i> j)
<i>RID</i>	<i>get_contact.collider</i> (<i>int</i> contact_idx) const
<i>int</i>	<i>get_contact.collider_id</i> (<i>int</i> contact_idx) const
<i>Object</i>	<i>get_contact.collider_object</i> (<i>int</i> contact_idx) const
<i>Vector3</i>	<i>get_contact.collider_position</i> (<i>int</i> contact_idx) const
<i>int</i>	<i>get_contact.collider_shape</i> (<i>int</i> contact_idx) const
<i>Vector3</i>	<i>get_contact.collider_velocity_at_position</i> (<i>int</i> contact_idx) const
<i>int</i>	<i>get_contact.count</i> () const
<i>Vector3</i>	<i>get_contact.local_normal</i> (<i>int</i> contact_idx) const
<i>Vector3</i>	<i>get_contact.local_position</i> (<i>int</i> contact_idx) const
<i>int</i>	<i>get_contact.local_shape</i> (<i>int</i> contact_idx) const
<i>PhysicsDirectSpaceState</i>	<i>get_space_state</i> ()
void	<i>integrate_forces</i> ()

25.281.3 Member Variables

- *Vector3 angular_velocity* - The angular velocity of the body.
- *Vector3 center_of_mass*
- *Vector3 inverse_inertia* - The inverse of the inertia of the body.
- *float inverse_mass* - The inverse of the mass of the body.
- *Vector3 linear_velocity* - The linear velocity of the body.
- *Basis principal_inertia_axes*
- *bool sleeping* - `true` if this body is currently sleeping (not active).
- *float step* - The timestep (delta) used for the simulation.
- *float total-angular_damp* - The rate at which the body stops rotating, if there are not any other forces moving it.
- *Vector3 total_gravity* - The total gravity vector being currently applied to this body.
- *float total_linear_damp* - The rate at which the body stops moving, if there are not any other forces moving it.
- *Transform transform* - The transformation matrix of the body.

25.281.4 Member Function Description

- `void add_force (Vector3 force, Vector3 position)`
- `void apply_impulse (Vector3 position, Vector3 j)`
- `void apply_torque_impulse (Vector3 j)`
- `RID get_contact.collider (int contact_idx) const`
- `int get_contact.collider_id (int contact_idx) const`
- `Object get_contact.collider_object (int contact_idx) const`
- `Vector3 get_contact.collider_position (int contact_idx) const`
- `int get_contact.collider_shape (int contact_idx) const`
- `Vector3 get_contact.collider_velocity_at_position (int contact_idx) const`
- `int get_contact_count () const`
- `Vector3 get_contact.local_normal (int contact_idx) const`
- `Vector3 get_contact.local_position (int contact_idx) const`
- `int get_contact.local_shape (int contact_idx) const`
- `PhysicsDirectSpaceState get_space_state ()`
- `void integrate_forces ()`

25.282 PhysicsDirectSpaceState

Inherits: *Object*

Category: Core

25.282.1 Brief Description

Direct access object to a space in the *PhysicsServer*.

25.282.2 Member Functions

Array	<code>cast_motion (PhysicsShapeQueryParameters shape, Vector3 motion)</code>
Array	<code>collide_shape (PhysicsShapeQueryParameters shape, int max_results=32)</code>
Dictionary	<code>get_rest_info (PhysicsShapeQueryParameters shape)</code>
Dictionary	<code>intersect_ray (Vector3 from, Vector3 to, Array exclude=[], int collision_layer=2147483647)</code>
Array	<code>intersect_shape (PhysicsShapeQueryParameters shape, int max_results=32)</code>

25.282.3 Description

Direct access object to a space in the *PhysicsServer*. It's used mainly to do queries against objects and areas residing in a given space.

25.282.4 Member Function Description

- **Array `cast_motion` (*PhysicsShapeQueryParameters* shape, *Vector3* motion)**

Checks whether the shape can travel to a point. The method will return an array with two floats between 0 and 1, both representing a fraction of motion. The first is how far the shape can move without triggering a collision, and the second is the point at which a collision will occur. If no collision is detected, the returned array will be 1, 1.

If the shape can not move, the array will be empty (`dir.empty () == true`).

- **Array `collide_shape` (*PhysicsShapeQueryParameters* shape, *int* max_results=32)**

Checks the intersections of a shape, given through a *PhysicsShapeQueryParameters* object, against the space. The resulting array contains a list of points where the shape intersects another. Like with `intersect_shape`, the number of returned results can be limited to save processing time.

- **Dictionary `get_rest_info` (*PhysicsShapeQueryParameters* shape)**

Checks the intersections of a shape, given through a *PhysicsShapeQueryParameters* object, against the space. If it collides with more than a shape, the nearest one is selected. The returned object is a dictionary containing the following fields:

`collider_id`: The colliding object's ID.

`linear_velocity`: The colliding object's velocity *Vector3*. If the object is an *Area*, the result is (0, 0, 0).

`normal`: The object's surface normal at the intersection point.

`point`: The intersection point.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

If the shape did not intersect anything, then an empty dictionary (`dir.empty() == true`) is returned instead.

- `Dictionary intersect_ray (Vector3 from, Vector3 to, Array exclude=[], int collision_layer=2147483647)`

Intersects a ray in a given space. The returned object is a dictionary with the following fields:

`collider`: The colliding object.

`collider_id`: The colliding object's ID.

`normal`: The object's surface normal at the intersection point.

`position`: The intersection point.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

If the ray did not intersect anything, then an empty dictionary (`dir.empty() == true`) is returned instead.

Additionally, the method can take an array of objects or *RIDs* that are to be excluded from collisions, or a bitmask representing the physics layers to check in.

- `Array intersect_shape (PhysicsShapeQueryParameters shape, int max_results=32)`

Checks the intersections of a shape, given through a *PhysicsShapeQueryParameters* object, against the space. The intersected shapes are returned in an array containing dictionaries with the following fields:

`collider`: The colliding object.

`collider_id`: The colliding object's ID.

`rid`: The intersecting object's *RID*.

`shape`: The shape index of the colliding shape.

The number of intersections can be limited with the second parameter, to reduce the processing time.

25.283 PhysicsServer

Inherits: *Object*

Inherited By: *BulletPhysicsServer*

Category: Core

25.283.1 Brief Description

Server interface for low level physics access.

25.283.2 Member Functions

<code>void</code>	<code>area_add_shape (RID area, RID shape, Transform transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0))</code>
<code>void</code>	<code>area_attach_object_instance_id (RID area, int id)</code>
<code>void</code>	<code>area_clear_shapes (RID area)</code>
<code>RID</code>	<code>area_create ()</code>
<code>int</code>	<code>area_get_object_instance_id (RID area) const</code>

Continued on next page

Table 25.18 – continued from previous page

<i>Variant</i>	<code>area_get_param (RID area, int param) const</code>
<i>RID</i>	<code>area_get_shape (RID area, int shape_idx) const</code>
<i>int</i>	<code>area_get_shape_count (RID area) const</code>
<i>Transform</i>	<code>area_get_shape_transform (RID area, int shape_idx) const</code>
<i>RID</i>	<code>area_get_space (RID area) const</code>
<i>int</i>	<code>area_get_space_override_mode (RID area) const</code>
<i>Transform</i>	<code>area_get_transform (RID area) const</code>
<i>bool</i>	<code>area_is_ray_pickable (RID area) const</code>
<i>void</i>	<code>area_remove_shape (RID area, int shape_idx)</code>
<i>void</i>	<code>area_set_collision_layer (RID area, int layer)</code>
<i>void</i>	<code>area_set_collision_mask (RID area, int mask)</code>
<i>void</i>	<code>area_set_monitor_callback (RID area, Object receiver, String method)</code>
<i>void</i>	<code>area_set_param (RID area, int param, Variant value)</code>
<i>void</i>	<code>area_set_ray_pickable (RID area, bool enable)</code>
<i>void</i>	<code>area_set_shape (RID area, int shape_idx, RID shape)</code>
<i>void</i>	<code>area_set_shape_transform (RID area, int shape_idx, Transform transform)</code>
<i>void</i>	<code>area_set_space (RID area, RID space)</code>
<i>void</i>	<code>area_set_space_override_mode (RID area, int mode)</code>
<i>void</i>	<code>area_set_transform (RID area, Transform transform)</code>
<i>void</i>	<code>body_add_collision_exception (RID body, RID excepted_body)</code>
<i>void</i>	<code>body_add_shape (RID body, RID shape, Transform transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0))</code>
<i>void</i>	<code>body_apply_impulse (RID body, Vector3 position, Vector3 impulse)</code>
<i>void</i>	<code>body_apply_torque_impulse (RID body, Vector3 impulse)</code>
<i>void</i>	<code>body_attach_object_instance_id (RID body, int id)</code>
<i>void</i>	<code>body_clear_shapes (RID body)</code>
<i>RID</i>	<code>body_create (int mode=2, bool init_sleeping=false)</code>
<i>int</i>	<code>body_get_collision_layer (RID body) const</code>
<i>int</i>	<code>body_get_collision_mask (RID body) const</code>
<i>PhysicsDirectBodyState</i>	<code>body_get_direct_state (RID body)</code>
<i>float</i>	<code>body_get_kinematic_safe_margin (RID body) const</code>
<i>int</i>	<code>body_get_max_contacts_reported (RID body) const</code>
<i>int</i>	<code>body_get_mode (RID body) const</code>
<i>int</i>	<code>body_get_object_instance_id (RID body) const</code>
<i>float</i>	<code>body_get_param (RID body, int param) const</code>
<i>RID</i>	<code>body_get_shape (RID body, int shape_idx) const</code>
<i>int</i>	<code>body_get_shape_count (RID body) const</code>
<i>Transform</i>	<code>body_get_shape_transform (RID body, int shape_idx) const</code>
<i>RID</i>	<code>body_get_space (RID body) const</code>
<i>Variant</i>	<code>body_get_state (RID body, int state) const</code>
<i>bool</i>	<code>body_is_axis_locked (RID body, int axis) const</code>
<i>bool</i>	<code>body_is_continuous_collision_detection_enabled (RID body) const</code>
<i>bool</i>	<code>body_is_omitting_force_integration (RID body) const</code>
<i>bool</i>	<code>body_is_ray_pickable (RID body) const</code>
<i>void</i>	<code>body_remove_collision_exception (RID body, RID excepted_body)</code>
<i>void</i>	<code>body_remove_shape (RID body, int shape_idx)</code>
<i>void</i>	<code>body_set_axis_lock (RID body, int axis, bool lock)</code>
<i>void</i>	<code>body_set_axis_velocity (RID body, Vector3 axis_velocity)</code>
<i>void</i>	<code>body_set_collision_layer (RID body, int layer)</code>
<i>void</i>	<code>body_set_collision_mask (RID body, int mask)</code>
<i>void</i>	<code>body_set_enable_continuous_collision_detection (RID body, bool enable)</code>

Continued on next page

Table 25.18 – continued from previous page

void	<i>body_set_force_integration_callback</i> (<i>RID</i> body, <i>Object</i> receiver, <i>String</i> method, <i>Variant</i> userdata=null)
void	<i>body_set_kinematic_safe_margin</i> (<i>RID</i> body, <i>float</i> margin)
void	<i>body_set_max_contacts_reported</i> (<i>RID</i> body, <i>int</i> amount)
void	<i>body_set_mode</i> (<i>RID</i> body, <i>int</i> mode)
void	<i>body_set OMIT force_integration</i> (<i>RID</i> body, <i>bool</i> enable)
void	<i>body_set_param</i> (<i>RID</i> body, <i>int</i> param, <i>float</i> value)
void	<i>body_set_ray_pickable</i> (<i>RID</i> body, <i>bool</i> enable)
void	<i>body_set_shape</i> (<i>RID</i> body, <i>int</i> shape_idx, <i>RID</i> shape)
void	<i>body_set_shape_transform</i> (<i>RID</i> body, <i>int</i> shape_idx, <i>Transform</i> transform)
void	<i>body_set_space</i> (<i>RID</i> body, <i>RID</i> space)
void	<i>body_set_state</i> (<i>RID</i> body, <i>int</i> state, <i>Variant</i> value)
<i>float</i>	<i>cone_twist_joint_get_param</i> (<i>RID</i> joint, <i>int</i> param) const
void	<i>cone_twist_joint_set_param</i> (<i>RID</i> joint, <i>int</i> param, <i>float</i> value)
void	<i>free_rid</i> (<i>RID</i> rid)
<i>bool</i>	<i>generic_6dof_joint_get_flag</i> (<i>RID</i> joint, <i>int</i> axis, <i>int</i> flag)
<i>float</i>	<i>generic_6dof_joint_get_param</i> (<i>RID</i> joint, <i>int</i> axis, <i>int</i> param)
void	<i>generic_6dof_joint_set_flag</i> (<i>RID</i> joint, <i>int</i> axis, <i>int</i> flag, <i>bool</i> enable)
void	<i>generic_6dof_joint_set_param</i> (<i>RID</i> joint, <i>int</i> axis, <i>int</i> param, <i>float</i> value)
<i>int</i>	<i>get_process_info</i> (<i>int</i> process_info)
<i>bool</i>	<i>hinge_joint_get_flag</i> (<i>RID</i> joint, <i>int</i> flag) const
<i>float</i>	<i>hinge_joint_get_param</i> (<i>RID</i> joint, <i>int</i> param) const
void	<i>hinge_joint_set_flag</i> (<i>RID</i> joint, <i>int</i> flag, <i>bool</i> enabled)
void	<i>hinge_joint_set_param</i> (<i>RID</i> joint, <i>int</i> param, <i>float</i> value)
<i>RID</i>	<i>joint_create_cone_twist</i> (<i>RID</i> body_A, <i>Transform</i> local_ref_A, <i>RID</i> body_B, <i>Transform</i> local_ref_B)
<i>RID</i>	<i>joint_create_generic_6dof</i> (<i>RID</i> body_A, <i>Transform</i> local_ref_A, <i>RID</i> body_B, <i>Transform</i> local_ref_B)
<i>RID</i>	<i>joint_create_hinge</i> (<i>RID</i> body_A, <i>Transform</i> hinge_A, <i>RID</i> body_B, <i>Transform</i> hinge_B)
<i>RID</i>	<i>joint_create_pin</i> (<i>RID</i> body_A, <i>Vector3</i> local_A, <i>RID</i> body_B, <i>Vector3</i> local_B)
<i>RID</i>	<i>joint_create_slider</i> (<i>RID</i> body_A, <i>Transform</i> local_ref_A, <i>RID</i> body_B, <i>Transform</i> local_ref_B)
<i>int</i>	<i>joint_get_solver_priority</i> (<i>RID</i> joint) const
<i>int</i>	<i>joint_get_type</i> (<i>RID</i> joint) const
void	<i>joint_set_solver_priority</i> (<i>RID</i> joint, <i>int</i> priority)
<i>Vector3</i>	<i>pin_joint_get_local_a</i> (<i>RID</i> joint) const
<i>Vector3</i>	<i>pin_joint_get_local_b</i> (<i>RID</i> joint) const
<i>float</i>	<i>pin_joint_get_param</i> (<i>RID</i> joint, <i>int</i> param) const
void	<i>pin_joint_set_local_a</i> (<i>RID</i> joint, <i>Vector3</i> local_A)
void	<i>pin_joint_set_local_b</i> (<i>RID</i> joint, <i>Vector3</i> local_B)
void	<i>pin_joint_set_param</i> (<i>RID</i> joint, <i>int</i> param, <i>float</i> value)
void	<i>set_active</i> (<i>bool</i> active)
<i>RID</i>	<i>shape_create</i> (<i>int</i> type)
<i>Variant</i>	<i>shape_get_data</i> (<i>RID</i> shape) const
<i>int</i>	<i>shape_get_type</i> (<i>RID</i> shape) const
void	<i>shape_set_data</i> (<i>RID</i> shape, <i>Variant</i> data)
<i>float</i>	<i>slider_joint_get_param</i> (<i>RID</i> joint, <i>int</i> param) const
void	<i>slider_joint_set_param</i> (<i>RID</i> joint, <i>int</i> param, <i>float</i> value)
<i>RID</i>	<i>space_create</i> ()
<i>PhysicsDirectSpaceState</i>	<i>space_get_direct_state</i> (<i>RID</i> space)
<i>float</i>	<i>space_get_param</i> (<i>RID</i> space, <i>int</i> param) const
<i>bool</i>	<i>space_is_active</i> (<i>RID</i> space) const
void	<i>space_set_active</i> (<i>RID</i> space, <i>bool</i> active)
void	<i>space_set_param</i> (<i>RID</i> space, <i>int</i> param, <i>float</i> value)

25.283.3 Enums

enum BodyState

- **BODY_STATE_TRANSFORM = 0** — Constant to set/get the current transform matrix of the body.
- **BODY_STATE_LINEAR_VELOCITY = 1** — Constant to set/get the current linear velocity of the body.
- **BODY_STATE_ANGULAR_VELOCITY = 2** — Constant to set/get the current angular velocity of the body.
- **BODY_STATE_SLEEPING = 3** — Constant to sleep/wake up a body, or to get whether it is sleeping.
- **BODY_STATE_CAN_SLEEP = 4** — Constant to set/get whether the body can sleep.

enum G6DOFJointAxisParam

- **G6DOF_JOINT_LINEAR_LOWER_LIMIT = 0** — The minimum difference between the pivot points' axes.
- **G6DOF_JOINT_LINEAR_UPPER_LIMIT = 1** — The maximum difference between the pivot points' axes.
- **G6DOF_JOINT_LINEAR_LIMIT_SOFTNESS = 2** — A factor that gets applied to the movement across the axes. The lower, the slower the movement.
- **G6DOF_JOINT_LINEAR_RESTITUTION = 3** — The amount of restitution on the axes movement. The lower, the more velocity-energy gets lost.
- **G6DOF_JOINT_LINEAR_DAMPING = 4** — The amount of damping that happens at the linear motion across the axes.
- **G6DOF_JOINT_ANGULAR_LOWER_LIMIT = 5** — The minimum rotation in negative direction to break loose and rotate around the axes.
- **G6DOF_JOINT_ANGULAR_UPPER_LIMIT = 6** — The minimum rotation in positive direction to break loose and rotate around the axes.
- **G6DOF_JOINT_ANGULAR_LIMIT_SOFTNESS = 7** — A factor that gets multiplied onto all rotations across the axes.
- **G6DOF_JOINT_ANGULAR_DAMPING = 8** — The amount of rotational damping across the axes. The lower, the more dampening occurs.
- **G6DOF_JOINT_ANGULAR_RESTITUTION = 9** — The amount of rotational restitution across the axes. The lower, the more restitution occurs.
- **G6DOF_JOINT_ANGULAR_FORCE_LIMIT = 10** — The maximum amount of force that can occur, when rotating around the axes.
- **G6DOF_JOINT_ANGULAR_ERP = 11** — When correcting the crossing of limits in rotation across the axes, this error tolerance factor defines how much the correction gets slowed down. The lower, the slower.
- **G6DOF_JOINT_ANGULAR_MOTOR_TARGET_VELOCITY = 12** — Target speed for the motor at the axes.
- **G6DOF_JOINT_ANGULAR_MOTOR_FORCE_LIMIT = 13** — Maximum acceleration for the motor at the axes.

enum ProcessInfo

- **INFO_ACTIVE_OBJECTS = 0** — Constant to get the number of objects that are not sleeping.
- **INFO_COLLISION_PAIRS = 1** — Constant to get the number of possible collisions.
- **INFO_ISLAND_COUNT = 2** — Constant to get the number of space regions where a collision could occur.

enum ShapeType

- **SHAPE_PLANE = 0** — The *Shape* is a *PlaneShape*.
- **SHAPE_RAY = 1** — The *Shape* is a *RayShape*.
- **SHAPE_SPHERE = 2** — The *Shape* is a *SphereShape*.
- **SHAPE_BOX = 3** — The *Shape* is a *BoxShape*.
- **SHAPE_CAPSULE = 4** — The *Shape* is a *CapsuleShape*.
- **SHAPE_CONVEX_POLYGON = 5** — The *Shape* is a *ConvexPolygonShape*.
- **SHAPE_CONCAVE_POLYGON = 6** — The *Shape* is a *ConcavePolygonShape*.
- **SHAPE_HEIGHTMAP = 7** — The *Shape* is a *HeightMapShape*.
- **SHAPE_CUSTOM = 8** — This constant is used internally by the engine. Any attempt to create this kind of shape results in an error.

enum HingeJointFlag

- **HINGE_JOINT_FLAG_USE_LIMIT = 0** — If true the Hinge has a maximum and a minimum rotation.
- **HINGE_JOINT_FLAG_ENABLE_MOTOR = 1** — If true a motor turns the Hinge

enum AreaParameter

- **AREA_PARAM_GRAVITY = 0** — Constant to set/get gravity strength in an area.
- **AREA_PARAM_GRAVITY_VECTOR = 1** — Constant to set/get gravity vector/center in an area.
- **AREA_PARAM_GRAVITY_IS_POINT = 2** — Constant to set/get whether the gravity vector of an area is a direction, or a center point.
- **AREA_PARAM_GRAVITY_DISTANCE_SCALE = 3** — Constant to set/get the falloff factor for point gravity of an area. The greater this value is, the faster the strength of gravity decreases with the square of distance.
- **AREA_PARAM_GRAVITY_POINT_ATTENUATION = 4** — This constant was used to set/get the falloff factor for point gravity. It has been superseded by AREA_PARAM_GRAVITY_DISTANCE_SCALE.
- **AREA_PARAM_LINEAR_DAMP = 5** — Constant to set/get the linear dampening factor of an area.
- **AREA_PARAM_ANGULAR_DAMP = 6** — Constant to set/get the angular dampening factor of an area.
- **AREA_PARAM_PRIORITY = 7** — Constant to set/get the priority (order of processing) of an area.

enum PinJointParam

- **PIN_JOINT_BIAS = 0** — The strength with which the pinned objects try to stay in positional relation to each other.

The higher, the stronger. - **PIN_JOINT_DAMPING = 1** — The strength with which the pinned objects try to stay in velocity relation to each other.

The higher, the stronger. - **PIN_JOINT_IMPULSE_CLAMP = 2** — If above 0, this value is the maximum value for an impulse that this Joint puts on it's ends.

enum BodyParameter

- **BODY_PARAM_BOUNCE = 0** — Constant to set/get a body's bounce factor.
- **BODY_PARAM_FRICTION = 1** — Constant to set/get a body's friction.
- **BODY_PARAM_MASS = 2** — Constant to set/get a body's mass.
- **BODY_PARAM_GRAVITY_SCALE = 3** — Constant to set/get a body's gravity multiplier.

- **BODY_PARAM_LINEAR_DAMP = 4** — Constant to set/get a body's linear dampening factor.
- **BODY_PARAM_ANGULAR_DAMP = 5** — Constant to set/get a body's angular dampening factor.
- **BODY_PARAM_MAX = 6** — This is the last ID for body parameters. Any attempt to set this property is ignored. Any attempt to get it returns 0.

enum **BodyMode**

- **BODY_MODE_STATIC = 0** — Constant for static bodies.
- **BODY_MODE_KINEMATIC = 1** — Constant for kinematic bodies.
- **BODY_MODE_RIGID = 2** — Constant for rigid bodies.
- **BODY_MODE_SOFT = 3**
- **BODY_MODE_CHARACTER = 4** — Constant for rigid bodies in character mode. In this mode, a body can not rotate, and only its linear velocity is affected by physics.

enum **SpaceParameter**

- **SPACE_PARAM_CONTACT_RECYCLE_RADIUS = 0** — Constant to set/get the maximum distance a pair of bodies has to move before their collision status has to be recalculated.
- **SPACE_PARAM_CONTACT_MAX_SEPARATION = 1** — Constant to set/get the maximum distance a shape can be from another before they are considered separated.
- **SPACE_PARAM_BODY_MAX_ALLOWED_PENETRATION = 2** — Constant to set/get the maximum distance a shape can penetrate another shape before it is considered a collision.
- **SPACE_PARAM_BODY_LINEAR_VELOCITY_SLEEP_THRESHOLD = 3** — Constant to set/get the threshold linear velocity of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after the time given.
- **SPACE_PARAM_BODY_ANGULAR_VELOCITY_SLEEP_THRESHOLD = 4** — Constant to set/get the threshold angular velocity of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after the time given.
- **SPACE_PARAM_BODY_TIME_TO_SLEEP = 5** — Constant to set/get the maximum time of activity. A body marked as potentially inactive for both linear and angular velocity will be put to sleep after this time.
- **SPACE_PARAM_BODY_ANGULAR_VELOCITY_DAMP_RATIO = 6**
- **SPACE_PARAM_CONSTRAINT_DEFAULT_BIAS = 7** — Constant to set/get the default solver bias for all physics constraints. A solver bias is a factor controlling how much two objects “rebound”, after violating a constraint, to avoid leaving them in that state because of numerical imprecision.

enum **AreaBodyStatus**

- **AREA_BODY_ADDED = 0** — The value of the first parameter and area callback function receives, when an object enters one of its shapes.
- **AREA_BODY_REMOVED = 1** — The value of the first parameter and area callback function receives, when an object exits one of its shapes.

enum **BodyAxis**

- **BODY_AXIS_LINEAR_X = 1**
- **BODY_AXIS_LINEAR_Y = 2**
- **BODY_AXIS_LINEAR_Z = 4**
- **BODY_AXIS_ANGULAR_X = 8**
- **BODY_AXIS_ANGULAR_Y = 16**

- **BODY_AXIS_ANGULAR_Z = 32**

enum **JointType**

- **JOINT_PIN = 0** — The *Joint* is a *PinJoint*.
- **JOINT_HINGE = 1** — The *Joint* is a *HingeJoint*.
- **JOINT_SLIDER = 2** — The *Joint* is a *SliderJoint*.
- **JOINT_CONE_TWIST = 3** — The *Joint* is a *ConeTwistJoint*.
- **JOINT_6DOF = 4** — The *Joint* is a *Generic6DOFJoint*.

enum **AreaSpaceOverrideMode**

- **AREA_SPACE_OVERRIDE_DISABLED = 0** — This area does not affect gravity/damp. These are generally areas that exist only to detect collisions, and objects entering or exiting them.
- **AREA_SPACE_OVERRIDE_COMBINE = 1** — This area adds its gravity/damp values to whatever has been calculated so far. This way, many overlapping areas can combine their physics to make interesting effects.
- **AREA_SPACE_OVERRIDE_REPLACE_COMBINE = 2** — This area adds its gravity/damp values to whatever has been calculated so far. Then stops taking into account the rest of the areas, even the default one.
- **AREA_SPACE_OVERRIDE_REPLACE = 3** — This area replaces any gravity/damp, even the default one, and stops taking into account the rest of the areas.
- **AREA_SPACE_OVERRIDE_REPLACE_COMBINE = 4** — This area replaces any gravity/damp calculated so far, but keeps calculating the rest of the areas, down to the default one.

enum **G6DOFJointAxisFlag**

- **G6DOF_JOINT_FLAG_ENABLE_LINEAR_LIMIT = 0** — If set there is linear motion possible within the given limits.
- **G6DOF_JOINT_FLAG_ENABLE_ANGULAR_LIMIT = 1** — If set there is rotational motion possible.
- **G6DOF_JOINT_FLAG_ENABLE_MOTOR = 2** — If set there is a rotational motor across these axes.

enum **SliderJointParam**

- **SLIDER_JOINT_LINEAR_LIMIT_UPPER = 0** — The maximum difference between the pivot points on their x-axis before damping happens.
- **SLIDER_JOINT_LINEAR_LIMIT_LOWER = 1** — The minimum difference between the pivot points on their x-axis before damping happens.
- **SLIDER_JOINT_LINEAR_LIMIT_SOFTNESS = 2** — A factor applied to the movement across the slider axis once the limits get surpassed. The lower, the slower the movement.
- **SLIDER_JOINT_LINEAR_LIMIT_RESTITUTION = 3** — The amount of restitution once the limits are surpassed. The lower, the more velocityenergy gets lost.
- **SLIDER_JOINT_LINEAR_LIMIT_DAMPING = 4** — The amount of damping once the slider limits are surpassed.
- **SLIDER_JOINT_LINEAR_MOTION_SOFTNESS = 5** — A factor applied to the movement across the slider axis as long as the slider is in the limits. The lower, the slower the movement.
- **SLIDER_JOINT_LINEAR_MOTION_RESTITUTION = 6** — The amount of restitution inside the slider limits.
- **SLIDER_JOINT_LINEAR_MOTION_DAMPING = 7** — The amount of damping inside the slider limits.
- **SLIDER_JOINT_LINEAR_ORTHOGONAL_SOFTNESS = 8** — A factor applied to the movement across axes orthogonal to the slider.

- **SLIDER_JOINT_LINEAR_ORTHOGONAL_RESTITUTION = 9** — The amount of restitution when movement is across axes orthogonal to the slider.
- **SLIDER_JOINT_LINEAR_ORTHOGONAL_DAMPING = 10** — The amount of damping when movement is across axes orthogonal to the slider.
- **SLIDER_JOINT_ANGULAR_LIMIT_UPPER = 11** — The upper limit of rotation in the slider.
- **SLIDER_JOINT_ANGULAR_LIMIT_LOWER = 12** — The lower limit of rotation in the slider.
- **SLIDER_JOINT_ANGULAR_LIMIT_SOFTNESS = 13** — A factor applied to the all rotation once the limit is surpassed.
- **SLIDER_JOINT_ANGULAR_LIMIT_RESTITUTION = 14** — The amount of restitution of the rotation when the limit is surpassed.
- **SLIDER_JOINT_ANGULAR_LIMIT_DAMPING = 15** — The amount of damping of the rotation when the limit is surpassed.
- **SLIDER_JOINT_ANGULAR_MOTION_SOFTNESS = 16** — A factor that gets applied to the all rotation in the limits.
- **SLIDER_JOINT_ANGULAR_MOTION_RESTITUTION = 17** — The amount of restitution of the rotation in the limits.
- **SLIDER_JOINT_ANGULAR_MOTION_DAMPING = 18** — The amount of damping of the rotation in the limits.
- **SLIDER_JOINT_ANGULAR_ORTHOGONAL_SOFTNESS = 19** — A factor that gets applied to the all rotation across axes orthogonal to the slider.
- **SLIDER_JOINT_ANGULAR_ORTHOGONAL_RESTITUTION = 20** — The amount of restitution of the rotation across axes orthogonal to the slider.
- **SLIDER_JOINT_ANGULAR_ORTHOGONAL_DAMPING = 21** — The amount of damping of the rotation across axes orthogonal to the slider.
- **SLIDER_JOINT_MAX = 22** — End flag of SLIDER_JOINT_* constants, used internally.

enum **ConeTwistJointParam**

- **CONE_TWIST_JOINT_SWING_SPAN = 0** — Swing is rotation from side to side, around the axis perpendicular to the twist axis.

The swing span defines, how much rotation will not get corrected allong the swing axis.

Could be defined as looseness in the *ConeTwistJoint*.

If below 0.05, this behaviour is locked. Default value: $\pi/4$. - **CONE_TWIST_JOINT_TWIST_SPAN = 1** — Twist is the rotation around the twist axis, this value defined how far the joint can twist.

Twist is locked if below 0.05. - **CONE_TWIST_JOINT_BIAS = 2** — The speed with which the swing or twist will take place.

The higher, the faster. - **CONE_TWIST_JOINT_SOFTNESS = 3** — The ease with which the Joint twists, if it's too low, it takes more force to twist the joint. - **CONE_TWIST_JOINT_RELAXATION = 4** — Defines, how fast the swing- and twist-speed-difference on both sides gets synced.

enum **HingeJointParam**

- **HINGE_JOINT_BIAS = 0** — The speed with which the two bodies get pulled together when they move in different directions.
- **HINGE_JOINT_LIMIT_UPPER = 1** — The maximum rotation across the Hinge.

- **HINGE_JOINT_LIMIT_LOWER = 2** — The minimum rotation across the Hinge.
- **HINGE_JOINT_LIMIT_BIAS = 3** — The speed with which the rotation across the axis perpendicular to the hinge gets corrected.
- **HINGE_JOINT_LIMIT_SOFTNESS = 4**
- **HINGE_JOINT_LIMIT_RELAXATION = 5** — The lower this value, the more the rotation gets slowed down.
- **HINGE_JOINT_MOTOR_TARGET_VELOCITY = 6** — Target speed for the motor.
- **HINGE_JOINT_MOTOR_MAX_IMPULSE = 7** — Maximum acceleration for the motor.

25.283.4 Description

Everything related to physics in 3D.

25.283.5 Member Function Description

- **void area_add_shape (RID area, RID shape, Transform transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0))**

Adds a shape to the area, along with a transform matrix. Shapes are usually referenced by their index, so you should track which shape has a given index.

- **void area_attach_object_instance_id (RID area, int id)**

Assigns the area to a descendant of *Object*, so it can exist in the node tree.

- **void area_clear_shapes (RID area)**

Removes all shapes from an area. It does not delete the shapes, so they can be reassigned later.

- **RID area_create ()**

Creates an *Area*.

- **int area_get_object_instance_id (RID area) const**

Gets the instance ID of the object the area is assigned to.

- **Variant area_get_param (RID area, int param) const**

Returns an area parameter value. A list of available parameters is on the AREA_PARAM_* constants.

- **RID area_get_shape (RID area, int shape_idx) const**

Returns the *RID* of the nth shape of an area.

- **int area_get_shape_count (RID area) const**

Returns the number of shapes assigned to an area.

- **Transform area_get_shape_transform (RID area, int shape_idx) const**

Returns the transform matrix of a shape within an area.

- **RID area_get_space (RID area) const**

Returns the space assigned to the area.

- **int area_get_space_override_mode (RID area) const**

Returns the space override mode for the area.

- **Transform area_get_transform (RID area) const**

Returns the transform matrix for an area.

- `bool area_is_ray_pickable (RID area) const`

If true area collides with rays.

- `void area_remove_shape (RID area, int shape_idx)`

Removes a shape from an area. It does not delete the shape, so it can be reassigned later.

- `void area_set_collision_layer (RID area, int layer)`

Assigns the area to one or many physics layers.

- `void area_set_collision_mask (RID area, int mask)`

Sets which physics layers the area will monitor.

- `void area_set_monitor_callback (RID area, Object receiver, String method)`

Sets the function to call when any body/area enters or exits the area. This callback will be called for any object interacting with the area, and takes five parameters:

1: AREA_BODY_ADDED or AREA_BODY_REMOVED, depending on whether the object entered or exited the area.

2: *RID* of the object that entered/exited the area.

3: Instance ID of the object that entered/exited the area.

4: The shape index of the object that entered/exited the area.

5: The shape index of the area where the object entered/exited.

- `void area_set_param (RID area, int param, Variant value)`

Sets the value for an area parameter. A list of available parameters is on the AREA_PARAM_* constants.

- `void area_set_ray_pickable (RID area, bool enable)`

Sets object pickable with rays.

- `void area_set_shape (RID area, int shape_idx, RID shape)`

Substitutes a given area shape by another. The old shape is selected by its index, the new one by its *RID*.

- `void area_set_shape_transform (RID area, int shape_idx, Transform transform)`

Sets the transform matrix for an area shape.

- `void area_set_space (RID area, RID space)`

Assigns a space to the area.

- `void area_set_space_override_mode (RID area, int mode)`

Sets the space override mode for the area. The modes are described in the constants AREA_SPACE_OVERRIDE_*.

- `void area_set_transform (RID area, Transform transform)`

Sets the transform matrix for an area.

- `void body_add_collision_exception (RID body, RID excepted_body)`

Adds a body to the list of bodies exempt from collisions.

- `void body_add_shape (RID body, RID shape, Transform transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0))`

Adds a shape to the body, along with a transform matrix. Shapes are usually referenced by their index, so you should track which shape has a given index.

- `void body_apply_impulse (RID body, Vector3 position, Vector3 impulse)`

Gives the body a push at a `position` in the direction of the `impulse`.

- `void body_apply_torque_impulse (RID body, Vector3 impulse)`

Gives the body a push to rotate it.

- `void body_attach_object_instance_id (RID body, int id)`

Assigns the area to a descendant of `Object`, so it can exist in the node tree.

- `void body_clear_shapes (RID body)`

Removes all shapes from a body.

- `RID body_create (int mode=2, bool init_sleeping=false)`

Creates a physics body. The first parameter can be any value from constants `BODY_MODE*`, for the type of body created. Additionally, the body can be created in sleeping state to save processing time.

- `int body_get_collision_layer (RID body) const`

Returns the physics layer or layers a body belongs to.

- `int body_get_collision_mask (RID body) const`

Returns the physics layer or layers a body can collide with.

-

- `PhysicsDirectBodyState body_get_direct_state (RID body)`

Returns the `PhysicsDirectBodyState` of the body.

- `float body_get_kinematic_safe_margin (RID body) const`

- `int body_get_max_contacts_reported (RID body) const`

Returns the maximum contacts that can be reported. See `body_set_max_contacts_reported`.

- `int body_get_mode (RID body) const`

Returns the body mode.

- `int body_get_object_instance_id (RID body) const`

Gets the instance ID of the object the area is assigned to.

- `float body_get_param (RID body, int param) const`

Returns the value of a body parameter. A list of available parameters is on the `BODY_PARAM_*` constants.

- `RID body_get_shape (RID body, int shape_idx) const`

Returns the `RID` of the nth shape of a body.

- `int body_get_shape_count (RID body) const`

Returns the number of shapes assigned to a body.

- `Transform body_get_shape_transform (RID body, int shape_idx) const`

Returns the transform matrix of a body shape.

- `RID body_get_space (RID body) const`

Returns the `RID` of the space assigned to a body.

- *Variant* **body_get_state** (*RID* body, *int* state) const

Returns a body state.

- *bool* **body_is_axis_locked** (*RID* body, *int* axis) const

- *bool* **body_is_continuous_collision_detection_enabled** (*RID* body) const

If true the continuous collision detection mode is enabled.

- *bool* **body_is omitting force integration** (*RID* body) const

Returns whether a body uses a callback function to calculate its own physics (see *body_set_force_integration_callback*).

- *bool* **body_is_ray_pickable** (*RID* body) const

If true the body can be detected by rays

- *void* **body_remove_collision_exception** (*RID* body, *RID* excepted_body)

Removes a body from the list of bodies exempt from collisions.

Continuous collision detection tries to predict where a moving body will collide, instead of moving it and correcting its movement if it collided.

- *void* **body_remove_shape** (*RID* body, *int* shape_idx)

Removes a shape from a body. The shape is not deleted, so it can be reused afterwards.

- *void* **body_set_axis_lock** (*RID* body, *int* axis, *bool* lock)

- *void* **body_set_axis_velocity** (*RID* body, *Vector3* axis_velocity)

Sets an axis velocity. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

- *void* **body_set_collision_layer** (*RID* body, *int* layer)

Sets the physics layer or layers a body belongs to.

- *void* **body_set_collision_mask** (*RID* body, *int* mask)

Sets the physics layer or layers a body can collide with.

- *void* **body_set_enable_continuous_collision_detection** (*RID* body, *bool* enable)

If true the continuous collision detection mode is enabled.

Continuous collision detection tries to predict where a moving body will collide, instead of moving it and correcting its movement if it collided.

- *void* **body_set_force_integration_callback** (*RID* body, *Object* receiver, *String* method, *Variant* userdata=null)

Sets the function used to calculate physics for an object, if that object allows it (see *body_set_omit_force_integration*).

- *void* **body_set_kinematic_safe_margin** (*RID* body, *float* margin)

- *void* **body_set_max_contacts_reported** (*RID* body, *int* amount)

Sets the maximum contacts to report. Bodies can keep a log of the contacts with other bodies, this is enabled by setting the maximum amount of contacts reported to a number greater than 0.

- *void* **body_set_mode** (*RID* body, *int* mode)

Sets the body mode, from one of the constants BODY_MODE*.

- *void* **body_set_omit_force_integration** (*RID* body, *bool* enable)

Sets whether a body uses a callback function to calculate its own physics (see `body_set_force_integration_callback`).

- `void body_set_param (RID body, int param, float value)`

Sets a body parameter. A list of available parameters is on the BODY_PARAM_* constants.

- `void body_set_ray_pickable (RID body, bool enable)`

Sets the body pickable with rays if `enabled` is set.

- `void body_set_shape (RID body, int shape_idx, RID shape)`

Substitutes a given body shape by another. The old shape is selected by its index, the new one by its *RID*.

- `void body_set_shape_transform (RID body, int shape_idx, Transform transform)`

Sets the transform matrix for a body shape.

- `void body_set_space (RID body, RID space)`

Assigns a space to the body (see `create_space`).

- `void body_set_state (RID body, int state, Variant value)`

Sets a body state (see BODY_STATE* constants).

- `float cone_twist_joint_get_param (RID joint, int param) const`

Gets a cone_twist_joint parameter (see CONE_TWIST_JOINT* constants).

- `void cone_twist_joint_set_param (RID joint, int param, float value)`

Sets a cone_twist_joint parameter (see CONE_TWIST_JOINT* constants).

- `void free_rid (RID rid)`

Destroys any of the objects created by PhysicsServer. If the *RID* passed is not one of the objects that can be created by PhysicsServer, an error will be sent to the console.

- `bool generic_6dof_joint_get_flag (RID joint, int axis, int flag)`

Gets a generic_6_DOF_joint flag (see G6DOF_JOINT_FLAG* constants).

- `float generic_6dof_joint_get_param (RID joint, int axis, int param)`

Gets a generic_6_DOF_joint parameter (see G6DOF_JOINT* constants without the G6DOF_JOINT_FLAG*).

- `void generic_6dof_joint_set_flag (RID joint, int axis, int flag, bool enable)`

Sets a generic_6_DOF_joint flag (see G6DOF_JOINT_FLAG* constants).

- `void generic_6dof_joint_set_param (RID joint, int axis, int param, float value)`

Sets a generic_6_DOF_joint parameter (see G6DOF_JOINT* constants without the G6DOF_JOINT_FLAG*).

- `int get_process_info (int process_info)`

Returns an Info defined by the ProcessInfo input given.

- `bool hinge_joint_get_flag (RID joint, int flag) const`

Gets a hinge_joint flag (see HINGE_JOINT_FLAG* constants).

- `float hinge_joint_get_param (RID joint, int param) const`

Gets a hinge_joint parameter (see HINGE_JOINT* constants without the HINGE_JOINT_FLAG*).

- `void hinge_joint_set_flag (RID joint, int flag, bool enabled)`

Sets a hinge_joint flag (see HINGE_JOINT_FLAG* constants).

- `void hinge_joint_set_param (RID joint, int param, float value)`

Sets a hinge_joint parameter (see HINGE_JOINT* constants without the HINGE_JOINT_FLAG*).

- `RID joint_create_cone_twist (RID body_A, Transform local_ref_A, RID body_B, Transform local_ref_B)`

Creates a *ConeTwistJoint*.

- `RID joint_create_generic_6dof (RID body_A, Transform local_ref_A, RID body_B, Transform local_ref_B)`

Creates a *Generic6DOFJoint*.

- `RID joint_create_hinge (RID body_A, Transform hinge_A, RID body_B, Transform hinge_B)`

Creates a *HingeJoint*.

- `RID joint_create_pin (RID body_A, Vector3 local_A, RID body_B, Vector3 local_B)`

Creates a *PinJoint*.

- `RID joint_create_slider (RID body_A, Transform local_ref_A, RID body_B, Transform local_ref_B)`

Creates a *SliderJoint*.

- `int joint_get_solver_priority (RID joint) const`

Gets the priority value of the Joint.

- `int joint_get_type (RID joint) const`

Returns the type of the Joint.

- `void joint_set_solver_priority (RID joint, int priority)`

Sets the priority value of the Joint.

- `Vector3 pin_joint_get_local_a (RID joint) const`

Returns position of the joint in the local space of body a of the joint.

- `Vector3 pin_joint_get_local_b (RID joint) const`

Returns position of the joint in the local space of body b of the joint.

- `float pin_joint_get_param (RID joint, int param) const`

Gets a pin_joint parameter (see PIN_JOINT* constants).

- `void pin_joint_set_local_a (RID joint, Vector3 local_A)`

Sets position of the joint in the local space of body a of the joint.

- `void pin_joint_set_local_b (RID joint, Vector3 local_B)`

Sets position of the joint in the local space of body b of the joint.

- `void pin_joint_set_param (RID joint, int param, float value)`

Sets a pin_joint parameter (see PIN_JOINT* constants).

- `void set_active (bool active)`

Activates or deactivates the 3D physics engine.

- `RID shape_create (int type)`

Creates a shape of type SHAPE_*. Does not assign it to a body or an area. To do so, you must use *area_set_shape* or *body_set_shape*.

- `Variant shape_get_data (RID shape) const`

Returns the shape data.

- `int shape_get_type (RID shape) const`

Returns the type of shape (see SHAPE_* constants).

- `void shape_set_data (RID shape, Variant data)`

Sets the shape data that defines its shape and size. The data to be passed depends on the kind of shape created `shape_get_type`.

- `float slider_joint_get_param (RID joint, int param) const`

Gets a slider_joint parameter (see SLIDER_JOINT* constants).

- `void slider_joint_set_param (RID joint, int param, float value)`

Gets a slider_joint parameter (see SLIDER_JOINT* constants).

- `RID space_create ()`

Creates a space. A space is a collection of parameters for the physics engine that can be assigned to an area or a body. It can be assigned to an area with `area_set_space`, or to a body with `body_set_space`.

- `PhysicsDirectSpaceState space_get_direct_state (RID space)`

Returns the state of a space, a `PhysicsDirectSpaceState`. This object can be used to make collision/intersection queries.

- `float space_get_param (RID space, int param) const`

Returns the value of a space parameter.

- `bool space_is_active (RID space) const`

Returns whether the space is active.

- `void space_set_active (RID space, bool active)`

Marks a space as active. It will not have an effect, unless it is assigned to an area or body.

- `void space_set_param (RID space, int param, float value)`

Sets the value for a space parameter. A list of available parameters is on the SPACE_PARAM_* constants.

25.284 PhysicsShapeQueryParameters

Inherits: Reference < Object

Category: Core

25.284.1 Brief Description

25.284.2 Member Functions

void	<code>set_shape (Resource shape)</code>
------	---

25.284.3 Member Variables

- *int* **collision_mask**
- *Array* **exclude**
- *float* **margin**
- *RID* **shape_rid**
- *Transform* **transform**

25.284.4 Member Function Description

- void **set_shape** (*Resource* shape)

25.285 PhysicsShapeQueryResult

Inherits: *Reference < Object*

Category: Core

25.285.1 Brief Description

Result of a shape query in Physics2DServer.

25.285.2 Member Functions

<i>int</i>	<code>get_result_count () const</code>
<i>Object</i>	<code>get_result_object (int idx) const</code>
<i>int</i>	<code>get_result_object_id (int idx) const</code>
<i>int</i>	<code>get_result_object_shape (int idx) const</code>
<i>RID</i>	<code>get_result_rid (int idx) const</code>

25.285.3 Member Function Description

- *int* **get_result_count** () const
- *Object* **get_result_object** (*int* idx) const
- *int* **get_result_object_id** (*int* idx) const
- *int* **get_result_object_shape** (*int* idx) const
- *RID* **get_result_rid** (*int* idx) const

25.286 PinJoint

Inherits: *Joint < Spatial < Node < Object*

Category: Core

25.286.1 Brief Description

Pin Joint for 3D Shapes.

25.286.2 Member Variables

- *float params/bias* - The force with which the pinned objects stay in positional relation to each other.

The higher, the stronger.

- *float params/damping* - The force with which the pinned objects stay in velocity relation to each other.

The higher, the stronger.

- *float params/impulse_clamp* - If above 0, this value is the maximum value for an impulse that this Joint produces.

25.286.3 Enums

enum **Param**

- **PARAM_BIAS = 0** — The force with which the pinned objects stay in positional relation to each other.

The higher, the stronger. - **PARAM_DAMPING = 1** — The force with which the pinned objects stay in velocity relation to each other.

The higher, the stronger. - **PARAM_IMPULSE_CLAMP = 2** — If above 0, this value is the maximum value for an impulse that this Joint produces.

25.286.4 Description

Pin Joint for 3D Rigid Bodies. It pins 2 bodies (rigid or static) together.

25.287 PinJoint2D

Inherits: *Joint2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.287.1 Brief Description

Pin Joint for 2D Shapes.

25.287.2 Member Variables

- *float softness* - The higher this value, the more the bond to the pinned partner can flex.

25.287.3 Description

Pin Joint for 2D Rigid Bodies. It pins 2 bodies (rigid or static) together, or a single body to a fixed position in space.

25.288 Plane

Category: Built-In Types

25.288.1 Brief Description

Plane in hessian form.

25.288.2 Member Functions

<i>Plane</i>	<code>Plane (float a, float b, float c, float d)</code>
<i>Plane</i>	<code>Plane (Vector3 v1, Vector3 v2, Vector3 v3)</code>
<i>Plane</i>	<code>Plane (Vector3 normal, float d)</code>
<i>Vector3</i>	<code>center ()</code>
<i>float</i>	<code>distance_to (Vector3 point)</code>
<i>Vector3</i>	<code>get_any_point ()</code>
<i>bool</i>	<code>has_point (Vector3 point, float epsilon=0.00001)</code>
<i>Vector3</i>	<code>intersect_3 (Plane b, Plane c)</code>
<i>Vector3</i>	<code>intersects_ray (Vector3 from, Vector3 dir)</code>
<i>Vector3</i>	<code>intersects_segment (Vector3 begin, Vector3 end)</code>
<i>bool</i>	<code>is_point_over (Vector3 point)</code>
<i>Plane</i>	<code>normalized ()</code>
<i>Vector3</i>	<code>project (Vector3 point)</code>

25.288.3 Member Variables

- *float d*
- *Vector3 normal*
- *float x*
- *float y*

- *float z*

25.288.4 Description

Plane represents a normalized plane equation. Basically, “normal” is the normal of the plane (a,b,c normalized), and “d” is the distance from the origin to the plane (in the direction of “normal”). “Over” or “Above” the plane is considered the side of the plane towards where the normal is pointing.

25.288.5 Member Function Description

- *Plane Plane (float a, float b, float c, float d)*

Creates a plane from the three parameters “a”, “b”, “c” and “d”.

- *Plane Plane (Vector3 v1, Vector3 v2, Vector3 v3)*

Creates a plane from three points.

- *Plane Plane (Vector3 normal, float d)*

Creates a plane from the normal and the plane’s distance to the origin.

- *Vector3 center ()*

Returns the center of the plane.

- *float distance_to (Vector3 point)*

Returns the shortest distance from the plane to the position “point”.

- *Vector3 get_any_point ()*

Returns a point on the plane.

- *bool has_point (Vector3 point, float epsilon=0.00001)*

Returns true if “point” is inside the plane (by a very minimum threshold).

- *Vector3 intersect_3 (Plane b, Plane c)*

Returns the intersection point of the three planes “b”, “c” and this plane. If no intersection is found null is returned.

- *Vector3 intersects_ray (Vector3 from, Vector3 dir)*

Returns the intersection point of a ray consisting of the position “from” and the direction normal “dir” with this plane. If no intersection is found null is returned.

- *Vector3 intersects_segment (Vector3 begin, Vector3 end)*

Returns the intersection point of a segment from position “begin” to position “end” with this plane. If no intersection is found null is returned.

- *bool is_point_over (Vector3 point)*

Returns true if “point” is located above the plane.

- *Plane normalized ()*

Returns a copy of the plane, normalized.

- *Vector3 project (Vector3 point)*

Returns the orthogonal projection of point “p” into a point in the plane.

25.289 PlaneMesh

Inherits: *PrimitiveMesh < Mesh < Resource < Reference < Object*

Category: Core

25.289.1 Brief Description

Class representing a planar *PrimitiveMesh*.

25.289.2 Member Variables

- *Vector2 size* - Size of the generated plane. Defaults to (2.0, 2.0).
- *int subdivide_depth* - Number of subdivision along the z-axis. Defaults to 0.
- *int subdivide_width* - Number of subdivision along the x-axis. Defaults to 0.

25.289.3 Description

Class representing a planar *PrimitiveMesh*. This flat mesh does not have a thickness.

25.290 PlaneShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.290.1 Brief Description

25.290.2 Member Variables

- *Plane plane*

25.291 PluginScript

Inherits: *Script < Resource < Reference < Object*

Category: Core

25.291.1 Brief Description

25.292 Polygon2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.292.1 Brief Description

A 2D polygon.

25.292.2 Member Variables

- *bool antialiased* - If true polygon edges will be anti-aliased. Default value: false.
- *Color color* - The polygon's fill color. If texture is defined, it will be multiplied by this color. It will also be the default color for vertices not set in vertex_colors.
- *float invert_border* - Added padding applied to the bounding box when using invert. Setting this value too small may result in a "Bad Polygon" error. Default value: 100.
- *bool invert_enable* - If true polygon will be inverted, containing the area outside the defined points and extending to the invert_border. Default value: false.
- *Vector2 offset* - The offset applied to each vertex.
- *PoolVector2Array polygon* - The polygon's list of vertices. The final point will be connected to the first.
- *Texture texture* - The polygon's fill texture. Use uv to set texture coordinates.
- *Vector2 texture_offset* - Amount to offset the polygon's texture. If (0, 0) the texture's origin (its top-left corner) will be placed at the polygon's position.
- *float texture_rotation* - The texture's rotation in radians.
- *float texture_rotation_degrees* - The texture's rotation in degrees.
- *Vector2 texture_scale* - Amount to multiply the uv coordinates when using a texture. Larger values make the texture smaller, and vice versa.
- *PoolVector2Array uv* - Texture coordinates for each vertex of the polygon. There should be one uv per polygon vertex. If there are fewer, undefined vertices will use (0, 0).
- *PoolColorArray vertex_colors* - Color for each vertex. Colors are interpolated between vertices, resulting in smooth gradients. There should be one per polygon vertex. If there are fewer, undefined vertices will use color.

25.292.3 Description

A Polygon2D is defined by a set of points. Each point is connected to the next, with the final point being connected to the first, resulting in a closed polygon. Polygon2Ds can be filled with color (solid or gradient) or filled with a given texture.

25.293 PolygonPathFinder

Inherits: *Resource < Reference < Object*

Category: Core

25.293.1 Brief Description

25.293.2 Member Functions

<i>PoolVector2Array</i>	<code>find_path (Vector2 from, Vector2 to)</code>
<i>Rect2</i>	<code>get_bounds () const</code>
<i>Vector2</i>	<code>get_closest_point (Vector2 point) const</code>
<i>PoolVector2Array</i>	<code>get_intersections (Vector2 from, Vector2 to) const</code>
<i>float</i>	<code>get_point_penalty (int idx) const</code>
<i>bool</i>	<code>is_point_inside (Vector2 point) const</code>
<i>void</i>	<code>set_point_penalty (int idx, float penalty)</code>
<i>void</i>	<code>setup (PoolVector2Array points, PoolIntArray connections)</code>

25.293.3 Member Function Description

- *PoolVector2Array* **find_path** (*Vector2* from, *Vector2* to)
- *Rect2* **get_bounds** () const
- *Vector2* **get_closest_point** (*Vector2* point) const
- *PoolVector2Array* **get_intersections** (*Vector2* from, *Vector2* to) const
- *float* **get_point_penalty** (*int* idx) const
- *bool* **is_point_inside** (*Vector2* point) const
- *void* **set_point_penalty** (*int* idx, *float* penalty)
- *void* **setup** (*PoolVector2Array* points, *PoolIntArray* connections)

25.294 PoolByteArray

Category: Built-In Types

25.294.1 Brief Description

Raw byte array.

25.294.2 Member Functions

<i>PoolByteArray</i>	<i>PoolByteArray (Array from)</i>
<i>void</i>	<i>append (int byte)</i>
<i>void</i>	<i>append_array (PoolByteArray array)</i>
<i>PoolByteArray</i>	<i>compress (int compression_mode=0)</i>
<i>PoolByteArray</i>	<i>decompress (int buffer_size, int compression_mode=0)</i>
<i>String</i>	<i>get_string_from_ascii ()</i>
<i>String</i>	<i>get_string_from_utf8 ()</i>
<i>int</i>	<i>insert (int idx, int byte)</i>
<i>void</i>	<i>invert ()</i>
<i>void</i>	<i>push_back (int byte)</i>
<i>void</i>	<i>remove (int idx)</i>
<i>void</i>	<i>resize (int idx)</i>
<i>void</i>	<i>set (int idx, int byte)</i>
<i>int</i>	<i>size ()</i>
<i>PoolByteArray</i>	<i>subarray (int from, int to)</i>

25.294.3 Description

Raw byte array. Contains bytes. Optimized for memory usage, can't fragment the memory.

25.294.4 Member Function Description

- *PoolByteArray PoolByteArray (Array from)*

Create from a generic array.

- **void append (int byte)**

Append an element at the end of the array (alias of *push_back*).

- **void append_array (PoolByteArray array)**

Append a *PoolByteArray* at the end of this array.

- **PoolByteArray compress (int compression_mode=0)**

Returns a new *PoolByteArray* with the data compressed. Set the compression mode using one of *File*'s COMPRESS_* constants.

- **PoolByteArray decompress (int buffer_size, int compression_mode=0)**

Returns a new *PoolByteArray* with the data decompressed. Set *buffer_size* to the size of the uncompressed data. Set the compression mode using one of *File*'s COMPRESS_* constants.

- **String get_string_from_ascii ()**

Returns a copy of the array's contents as *String*. Fast alternative to *PoolByteArray.get_string_from_utf8* if the content is ASCII-only. Unlike the UTF-8 function this function maps every byte to a character in the array. Multibyte sequences will not be interpreted correctly. For parsing user input always use *PoolByteArray.get_string_from_utf8*.

- **String get_string_from_utf8 ()**

Returns a copy of the array's contents as *String*. Slower than *PoolByteArray.get_string_from_ascii* but supports UTF-8 encoded data. Use this function if you are unsure about the source of the data. For user input this function should always be preferred.

- `int insert (int idx, int byte)`

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (`pos==size()`).

- `void invert ()`

Reverse the order of the elements in the array (so first element will now be the last).

- `void push_back (int byte)`

Append an element at the end of the array.

- `void remove (int idx)`

Remove an element from the array by index.

- `void resize (int idx)`

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- `void set (int idx, int byte)`

Change the byte at the given index.

- `int size ()`

Return the size of the array.

- `PoolByteArray subarray (int from, int to)`

Returns the slice of the `PoolByteArray` between indices (inclusive) as a new `PoolByteArray`. Any negative index is considered to be from the end of the array.

25.295 PoolColorArray

Category: Built-In Types

25.295.1 Brief Description

Array of Colors

25.295.2 Member Functions

<code>PoolColorArray</code>	<code>PoolColorArray (Array from)</code>
<code>void</code>	<code>append (Color color)</code>
<code>void</code>	<code>append_array (PoolColorArray array)</code>
<code>int</code>	<code>insert (int idx, Color color)</code>
<code>void</code>	<code>invert ()</code>
<code>void</code>	<code>push_back (Color color)</code>
<code>void</code>	<code>remove (int idx)</code>
<code>void</code>	<code>resize (int idx)</code>
<code>void</code>	<code>set (int idx, Color color)</code>
<code>int</code>	<code>size ()</code>

25.295.3 Description

Array of Color, Contains colors. Optimized for memory usage, can't fragment the memory.

25.295.4 Member Function Description

- *PoolColorArray PoolColorArray (Array from)*

Create from a generic array.

- **void append (Color color)**

Append an element at the end of the array (alias of *push_back*).

- **void append_array (PoolColorArray array)**

Append a PoolColorArray at the end of this array.

- **int insert (int idx, Color color)**

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (*pos==size()*).

- **void invert ()**

Reverse the order of the elements in the array (so first element will now be the last).

- **void push_back (Color color)**

Append a value to the array.

- **void remove (int idx)**

Remove an element from the array by index.

- **void resize (int idx)**

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- **void set (int idx, Color color)**

Change the *Color* at the given index.

- **int size ()**

Return the size of the array.

25.296 PoolIntArray

Category: Built-In Types

25.296.1 Brief Description

Integer Array.

25.296.2 Member Functions

<i>PoolIntArray</i>	<i>PoolIntArray (Array from)</i>
void	<i>append (int integer)</i>
void	<i>append_array (PoolIntArray array)</i>
<i>int</i>	<i>insert (int idx, int integer)</i>
void	<i>invert ()</i>
void	<i>push_back (int integer)</i>
void	<i>remove (int idx)</i>
void	<i>resize (int idx)</i>
void	<i>set (int idx, int integer)</i>
<i>int</i>	<i>size ()</i>

25.296.3 Description

Integer Array. Contains integers. Optimized for memory usage, can't fragment the memory.

25.296.4 Member Function Description

- *PoolIntArray PoolIntArray (Array from)*

Create from a generic array.

- void **append** (*int integer*)

Append an element at the end of the array (alias of *push_back*).

- void **append_array** (*PoolIntArray array*)

Append an *PoolIntArray* at the end of this array.

- *int insert (int idx, int integer)*

Insert a new int at a given position in the array. The position must be valid, or at the end of the array (*pos==size()*).

- void **invert** ()

Reverse the order of the elements in the array (so first element will now be the last).

- void **push_back** (*int integer*)

Append a value to the array.

- void **remove** (*int idx*)

Remove an element from the array by index.

- void **resize** (*int idx*)

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- void **set** (*int idx, int integer*)

Change the int at the given index.

- *int size ()*

Return the array size.

25.297 PoolRealArray

Category: Built-In Types

25.297.1 Brief Description

Real Array.

25.297.2 Member Functions

<i>PoolRealArray</i>	<i>PoolRealArray (Array from)</i>
<i>void</i>	<i>append (float value)</i>
<i>void</i>	<i>append_array (PoolRealArray array)</i>
<i>int</i>	<i>insert (int idx, float value)</i>
<i>void</i>	<i>invert ()</i>
<i>void</i>	<i>push_back (float value)</i>
<i>void</i>	<i>remove (int idx)</i>
<i>void</i>	<i>resize (int idx)</i>
<i>void</i>	<i>set (int idx, float value)</i>
<i>int</i>	<i>size ()</i>

25.297.3 Description

Real Array. Array of floating point values. Can only contain floats. Optimized for memory usage, can't fragment the memory.

25.297.4 Member Function Description

- *PoolRealArray PoolRealArray (Array from)*

Create from a generic array.

- *void append (float value)*

Append an element at the end of the array (alias of *push_back*).

- *void append_array (PoolRealArray array)*

Append an RealArray at the end of this array.

- *int insert (int idx, float value)*

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (*pos==size()*).

- *void invert ()*

Reverse the order of the elements in the array (so first element will now be the last).

- *void push_back (float value)*

Append an element at the end of the array.

- *void remove (int idx)*

Remove an element from the array by index.

- void **resize** (*int idx*)

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- void **set** (*int idx, float value*)

Change the float at the given index.

- *int size* ()

Return the size of the array.

25.298 PoolStringArray

Category: Built-In Types

25.298.1 Brief Description

String Array.

25.298.2 Member Functions

<i>PoolStringArray</i>	<i>PoolStringArray (Array from)</i>
void	<i>append (String string)</i>
void	<i>append_array (PoolStringArray array)</i>
<i>int</i>	<i>insert (int idx, String string)</i>
void	<i>invert ()</i>
<i>String</i>	<i>join (String delimiter)</i>
void	<i>push_back (String string)</i>
void	<i>remove (int idx)</i>
void	<i>resize (int idx)</i>
void	<i>set (int idx, String string)</i>
<i>int</i>	<i>size ()</i>

25.298.3 Description

String Array. Array of strings. Can only contain strings. Optimized for memory usage, can't fragment the memory.

25.298.4 Member Function Description

- **PoolStringArray PoolStringArray (Array from)**

Create from a generic array.

- void **append** (*String string*)

Append an element at the end of the array (alias of *push_back*).

- void **append_array** (*PoolStringArray array*)

Append an *StringArray* at the end of this array.

- `int insert (int idx, String string)`

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (`pos==size()`).

- `void invert ()`

Reverse the order of the elements in the array (so first element will now be the last).

- `String join (String delimiter)`

Returns a `String` with each element of the array joined with the delimiter.

- `void push_back (String string)`

Append a string element at end of the array.

- `void remove (int idx)`

Remove an element from the array by index.

- `void resize (int idx)`

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- `void set (int idx, String string)`

Change the `String` at the given index.

- `int size ()`

Return the size of the array.

25.299 PoolVector2Array

Category: Built-In Types

25.299.1 Brief Description

An Array of Vector2.

25.299.2 Member Functions

<code>PoolVector2Array</code>	<code>PoolVector2Array (Array from)</code>
<code>void</code>	<code>append (Vector2 vector2)</code>
<code>void</code>	<code>append_array (PoolVector2Array array)</code>
<code>int</code>	<code>insert (int idx, Vector2 vector2)</code>
<code>void</code>	<code>invert ()</code>
<code>void</code>	<code>push_back (Vector2 vector2)</code>
<code>void</code>	<code>remove (int idx)</code>
<code>void</code>	<code>resize (int idx)</code>
<code>void</code>	<code>set (int idx, Vector2 vector2)</code>
<code>int</code>	<code>size ()</code>

25.299.3 Description

An Array specifically designed to hold Vector2.

25.299.4 Member Function Description

- *PoolVector2Array PoolVector2Array (Array from)*

Construct a new PoolVector2Array. Optionally, you can pass in an Array that will be converted.

- **void append (Vector2 vector2)**

Append an element at the end of the array (alias of *push_back*).

- **void append_array (PoolVector2Array array)**

Append an PoolVector2Array at the end of this array.

- **int insert (int idx, Vector2 vector2)**

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (*pos==size()*).

- **void invert ()**

Reverse the order of the elements in the array (so first element will now be the last).

- **void push_back (Vector2 vector2)**

Insert a *Vector2* at the end.

- **void remove (int idx)**

Remove an element from the array by index.

- **void resize (int idx)**

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- **void set (int idx, Vector2 vector2)**

Change the *Vector2* at the given index.

- **int size ()**

Return the size of the array.

25.300 PoolVector3Array

Category: Built-In Types

25.300.1 Brief Description

An Array of Vector3.

25.300.2 Member Functions

<i>PoolVector3Array</i>	<i>PoolVector3Array (Array from)</i>
void	<i>append (Vector3 vector3)</i>
void	<i>append_array (PoolVector3Array array)</i>
<i>int</i>	<i>insert (int idx, Vector3 vector3)</i>
void	<i>invert ()</i>
void	<i>push_back (Vector3 vector3)</i>
void	<i>remove (int idx)</i>
void	<i>resize (int idx)</i>
void	<i>set (int idx, Vector3 vector3)</i>
<i>int</i>	<i>size ()</i>

25.300.3 Description

An Array specifically designed to hold Vector3.

25.300.4 Member Function Description

- *PoolVector3Array PoolVector3Array (Array from)*

Construct a new PoolVector3Array. Optionally, you can pass in an Array that will be converted.

- void **append** (*Vector3* *vector3*)

Append an element at the end of the array (alias of *push_back*).

- void **append_array** (*PoolVector3Array* *array*)

Append an *PoolVector3Array* at the end of this array.

- *int insert (int idx, Vector3 vector3)*

Insert a new element at a given position in the array. The position must be valid, or at the end of the array (*pos==size()*).

- void **invert** ()

Reverse the order of the elements in the array (so first element will now be the last).

- void **push_back** (*Vector3* *vector3*)

Insert a *Vector3* at the end.

- void **remove** (*int idx*)

Remove an element from the array by index.

- void **resize** (*int idx*)

Set the size of the array. If the array is grown reserve elements at the end of the array. If the array is shrunk truncate the array to the new size.

- void **set** (*int idx, Vector3 vector3*)

Change the *Vector3* at the given index.

- *int size ()*

Return the size of the array.

25.301 Popup

Inherits: *Control < CanvasItem < Node < Object*

Inherited By: *PopupPanel, PopupDialog, PopupMenu, WindowDialog*

Category: Core

25.301.1 Brief Description

Base container control for popups and dialogs.

25.301.2 Member Functions

void	<i>popup (Rect2 bounds=Rect2(0, 0, 0, 0))</i>
void	<i>popup_centered (Vector2 size=Vector2(0, 0))</i>
void	<i>popup_centered_minsize (Vector2 minsize=Vector2(0, 0))</i>
void	<i>popup_centered_ratio (float ratio=0.75)</i>

25.301.3 Signals

- **about_to_show ()**

This signal is emitted when a popup is about to be shown. (often used in *PopupMenu* for clearing the list of options and creating a new one according to the current context).

- **popup_hide ()**

This signal is emitted when a popup is hidden.

25.301.4 Member Variables

- *bool popup_exclusive*

25.301.5 Numeric Constants

- **NOTIFICATION_POST_POPUP = 80** — Notification sent right after the popup is shown.
- **NOTIFICATION_POPUP_HIDE = 81** — Notification sent right after the popup is hidden.

25.301.6 Description

Popup is a base *Control* used to show dialogs and popups. It's a subwindow and modal by default (see *Control*) and has helpers for custom popup behavior.

25.301.7 Member Function Description

- void **popup** (*Rect2 bounds=Rect2(0, 0, 0, 0)*)

Popup (show the control in modal form).

- void **popup_centered** (*Vector2 size=Vector2(0, 0)*)

Popup (show the control in modal form) in the center of the screen, at the current size, or at a size determined by “size”.

- void **popup_centered_minsize** (*Vector2 minsize=Vector2(0, 0)*)

Popup (show the control in modal form) in the center of the screen, ensuring the size is never smaller than *minsize*.

- void **popup_centered_ratio** (*float ratio=0.75*)

Popup (show the control in modal form) in the center of the screen, scaled at a ratio of size of the screen.

25.302 PopupDialog

Inherits: *Popup < Control < CanvasItem < Node < Object*

Category: Core

25.302.1 Brief Description

Base class for Popup Dialogs.

25.302.2 Description

PopupDialog is a base class for popup dialogs, along with *WindowDialog*.

25.303 PopupMenu

Inherits: *Popup < Control < CanvasItem < Node < Object*

Category: Core

25.303.1 Brief Description

PopupMenu displays a list of options.

25.303.2 Member Functions

void	<i>add_check_item (String label, int id=-1, int accel=0)</i>
void	<i>add_check_shortcut (ShortCut shortcut, int id=-1, bool global=false)</i>
void	<i>add_icon_check_item (Texture texture, String label, int id=-1, int accel=0)</i>
void	<i>add_icon_check_shortcut (Texture texture, ShortCut shortcut, int id=-1, bool global=false)</i>
void	<i>add_icon_item (Texture texture, String label, int id=-1, int accel=0)</i>

Continued on next page

Table 25.19 – continued from previous page

void	<i>add_icon_shortcut (Texture texture, ShortCut shortcut, int id=-1, bool global=false)</i>
void	<i>add_item (String label, int id=-1, int accel=0)</i>
void	<i>add_separator ()</i>
void	<i>add_shortcut (ShortCut shortcut, int id=-1, bool global=false)</i>
void	<i>add_submenu_item (String label, String submenu, int id=-1)</i>
void	<i>clear ()</i>
<i>int</i>	<i>get_item_accelerator (int idx) const</i>
<i>int</i>	<i>get_item_count () const</i>
<i>Texture</i>	<i>get_item_icon (int idx) const</i>
<i>int</i>	<i>get_item_id (int idx) const</i>
<i>int</i>	<i>get_item_index (int id) const</i>
<i>Variant</i>	<i>get_item_metadata (int idx) const</i>
<i>ShortCut</i>	<i>get_item_shortcut (int idx) const</i>
<i>String</i>	<i>get_item_submenu (int idx) const</i>
<i>String</i>	<i>get_item_text (int idx) const</i>
<i>String</i>	<i>get_item_tooltip (int idx) const</i>
<i>bool</i>	<i>is_item_checkable (int idx) const</i>
<i>bool</i>	<i>is_item_checked (int idx) const</i>
<i>bool</i>	<i>is_item_disabled (int idx) const</i>
<i>bool</i>	<i>is_item_separator (int idx) const</i>
void	<i>remove_item (int idx)</i>
void	<i>set_item_accelerator (int idx, int accel)</i>
void	<i>set_item_as_checkable (int idx, bool enable)</i>
void	<i>set_item_as_separator (int idx, bool enable)</i>
void	<i>set_item_checked (int idx, bool checked)</i>
void	<i>set_item_disabled (int idx, bool disabled)</i>
void	<i>set_item_icon (int idx, Texture icon)</i>
void	<i>set_item_id (int idx, int id)</i>
void	<i>set_item_metadata (int idx, Variant metadata)</i>
void	<i>set_item_multistate (int idx, int state)</i>
void	<i>set_item_shortcut (int idx, ShortCut shortcut, bool global=false)</i>
void	<i>set_item_submenu (int idx, String submenu)</i>
void	<i>set_item_text (int idx, String text)</i>
void	<i>set_item_tooltip (int idx, String tooltip)</i>
void	<i>toggle_item_checked (int idx)</i>
void	<i>toggle_item_multistate (int idx)</i>

25.303.3 Signals

- **id_pressed (int ID)**

This event is emitted when an item of some id is pressed or its accelerator is activated.

- **index_pressed (int index)**

This event is emitted when an item of some index is pressed or its accelerator is activated.

25.303.4 Member Variables

- *bool hide_on_checkable_item_selection*

- *bool* **hide_on_item_selection**
- *bool* **hide_on_state_item_selection**

25.303.5 Description

PopupMenu is the typical Control that displays a list of options. They are popular in toolbars or context menus.

25.303.6 Member Function Description

- **void add_check_item (String label, int id=-1, int accel=0)**

Add a new checkable item with text “label”. An id can optionally be provided, as well as an accelerator. If no id is provided, one will be created from the index. Note that checkable items just display a checkmark, but don’t have any built-in checking behavior and must be checked/unchecked manually.

- **void add_check_shortcut (ShortCut shortcut, int id=-1, bool global=false)**
- **void add_icon_check_item (Texture texture, String label, int id=-1, int accel=0)**

Add a new checkable item with text “label” and icon “texture”. An id can optionally be provided, as well as an accelerator. If no id is provided, one will be

created from the index. Note that checkable items just display a checkmark, but don’t have any built-in checking behavior and must be checked/unchecked manually.

- **void add_icon_check_shortcut (Texture texture, ShortCut shortcut, int id=-1, bool global=false)**
- **void add_icon_item (Texture texture, String label, int id=-1, int accel=0)**

Add a new item with text “label” and icon “texture”. An id can optionally be provided, as well as an accelerator keybinding. If no id is provided, one will be created from the index.

- **void add_icon_shortcut (Texture texture, ShortCut shortcut, int id=-1, bool global=false)**
- **void add_item (String label, int id=-1, int accel=0)**

Add a new item with text “label”. An id can optionally be provided, as well as an accelerator keybinding. If no id is provided, one will be created from the index.

- **void add_separator ()**

Add a separator between items. Separators also occupy an index.

- **void add_shortcut (ShortCut shortcut, int id=-1, bool global=false)**
- **void add_submenu_item (String label, String submenu, int id=-1)**

Adds an item with a submenu. The submenu is the name of a child PopupMenu node that would be shown when the item is clicked. An id can optionally be provided, but if it isn’t provided, one will be created from the index.

- **void clear ()**

Clear the popup menu, in effect removing all items.

- **int get_item_accelerator (int idx) const**

Return the accelerator of the item at index “idx”. Accelerators are special combinations of keys that activate the item, no matter which control is focused.

- **int get_item_count () const**

Return the amount of items.

- *Texture get_item_icon (int idx) const*

Return the icon of the item at index “idx”.

- *int get_item_id (int idx) const*

Return the id of the item at index “idx”.

- *int get_item_index (int id) const*

Find and return the index of the item containing a given id.

- *Variant get_item_metadata (int idx) const*

Return the metadata of an item, which might be of any type. You can set it with *set_item_metadata*, which provides a simple way of assigning context data to items.

- *ShortCut get_item_shortcut (int idx) const*

- *String get_item_submenu (int idx) const*

Return the submenu name of the item at index “idx”.

- *String get_item_text (int idx) const*

Return the text of the item at index “idx”.

- *String get_item_tooltip (int idx) const*

- *bool is_item_checkable (int idx) const*

Return whether the item at index “idx” has a checkbox. Note that checkable items just display a checkmark, but don’t have any built-in checking behavior and must be checked/unchecked manually.

- *bool is_item_checked (int idx) const*

Return the checkstate status of the item at index “idx”.

- *bool is_item_disabled (int idx) const*

Return whether the item at index “idx” is disabled. When it is disabled it can’t be selected, or its action invoked.

- *bool is_item_separator (int idx) const*

Return whether the item is a separator. If it is, it would be displayed as a line.

- *void remove_item (int idx)*

Removes the item at index “idx” from the menu. Note that the indexes of items after the removed item are going to be shifted by one.

- *void set_item_accelerator (int idx, int accel)*

Set the accelerator of the item at index “idx”. Accelerators are special combinations of keys that activate the item, no matter which control is focused.

- *void set_item_as_checkable (int idx, bool enable)*

Set whether the item at index “idx” has a checkbox. Note that checkable items just display a checkmark, but don’t have any built-in checking behavior and must be checked/unchecked manually.

- *void set_item_as_separator (int idx, bool enable)*

Mark the item at index “idx” as a separator, which means that it would be displayed as a mere line.

- *void set_item_checked (int idx, bool checked)*

Set the checkstate status of the item at index “idx”.

- *void set_item_disabled (int idx, bool disabled)*

Sets whether the item at index “idx” is disabled or not. When it is disabled it can’t be selected, or its action invoked.

- void **set_item_icon** (*int idx, Texture icon*)
- void **set_item_id** (*int idx, int id*)

Set the id of the item at index “idx”.

- void **set_item_metadata** (*int idx, Variant metadata*)

Sets the metadata of an item, which might be of any type. You can later get it with *get_item_metadata*, which provides a simple way of assigning context data to items.

- void **set_item_multistate** (*int idx, int state*)
- void **set_item_shortcut** (*int idx, ShortCut shortcut, bool global=false*)
- void **set_item_submenu** (*int idx, String submenu*)

Sets the submenu of the item at index “idx”. The submenu is the name of a child PopupMenu node that would be shown when the item is clicked.

- void **set_item_text** (*int idx, String text*)

Set the text of the item at index “idx”.

- void **set_item_tooltip** (*int idx, String tooltip*)
- void **toggle_item_checked** (*int idx*)
- void **toggle_item_multistate** (*int idx*)

25.304 PopupPanel

Inherits: *Popup < Control < CanvasItem < Node < Object*

Category: Core

25.304.1 Brief Description

Class for displaying popups with a panel background.

25.304.2 Description

Class for displaying popups with a panel background. In some cases it might be simpler to use than *Popup*, since it provides a configurable background. If you are making windows, better check *WindowDialog*.

25.305 Position2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.305.1 Brief Description

Generic 2D Position hint for editing.

25.305.2 Description

Generic 2D Position hint for editing. It's just like a plain *Node2D* but displays as a cross in the 2D-Editor at all times.

25.306 Position3D

Inherits: *Spatial < Node < Object*

Category: Core

25.306.1 Brief Description

Generic 3D Position hint for editing

25.306.2 Description

Generic 3D Position hint for editing. It's just like a plain *Spatial* but displays as a cross in the 3D-Editor at all times.

25.307 PrimitiveMesh

Inherits: *Mesh < Resource < Reference < Object*

Inherited By: *PlaneMesh, CubeMesh, SphereMesh, CylinderMesh, CapsuleMesh, QuadMesh, PrismMesh*

Category: Core

25.307.1 Brief Description

Base class for all primitive meshes. Handles applying a *Material* to a primitive mesh.

25.307.2 Member Functions

<i>Array</i>	<i>get_mesh_arrays() const</i>
--------------	---------------------------------

25.307.3 Member Variables

- **Material material** - The current *Material* of the primitive mesh.

25.307.4 Description

Base class for all primitive meshes. Handles applying a *Material* to a primitive mesh.

25.307.5 Member Function Description

- *Array* **get_mesh_arrays () const**

25.308 PrismMesh

Inherits: *PrimitiveMesh < Mesh < Resource < Reference < Object*

Category: Core

25.308.1 Brief Description

Class representing a prism-shaped *PrimitiveMesh*.

25.308.2 Member Variables

- *float* **left_to_right** - Displacement of the upper edge along the x-axis. 0.0 positions edge straight above the bottom left edge. Defaults to 0.5 (positioned on the midpoint).
- *Vector3* **size** - Size of the prism. Defaults to (2.0, 2.0, 2.0).
- *int* **subdivide_depth** - Number of added edge loops along the z-axis. Defaults to 0.
- *int* **subdivide_height** - Number of added edge loops along the y-axis. Defaults to 0.
- *int* **subdivide_width** - Number of added edge loops along the x-axis. Defaults to 0.

25.308.3 Description

Class representing a prism-shaped *PrimitiveMesh*.

25.309 ProceduralSky

Inherits: *Sky < Resource < Reference < Object*

Category: Core

25.309.1 Brief Description

25.309.2 Member Variables

- *Color* **ground_bottom_color**
- *float* **ground_curve**
- *float* **ground_energy**
- *Color* **ground_horizon_color**

- *float* `sky_curve`
- *float* `sky_energy`
- *Color* `sky_horizon_color`
- *Color* `sky_top_color`
- *float* `sun_angle_max`
- *float* `sun_angle_min`
- *Color* `sun_color`
- *float* `sun_curve`
- *float* `sun_energy`
- *float* `sun_latitude`
- *float* `sun_longitude`
- *TextureSize* `texture_size`

25.309.3 Enums

enum **TextureSize**

- `TEXTURE_SIZE_256 = 0`
- `TEXTURE_SIZE_512 = 1`
- `TEXTURE_SIZE_1024 = 2`
- `TEXTURE_SIZE_2048 = 3`
- `TEXTURE_SIZE_4096 = 4`
- `TEXTURE_SIZE_MAX = 5`

25.310 ProgressBar

Inherits: *Range < Control < CanvasItem < Node < Object*

Category: Core

25.310.1 Brief Description

General purpose progress bar.

25.310.2 Member Variables

- *bool* `percent_visible`

25.310.3 Description

General purpose progress bar. Shows fill percentage from right to left.

25.311 ProjectSettings

Inherits: *Object*

Category: Core

25.311.1 Brief Description

Contains global variables accessible from everywhere.

25.311.2 Member Functions

void	<i>add_property_info</i> (<i>Dictionary</i> hint)
void	<i>clear</i> (<i>String</i> name)
<i>int</i>	<i>get_order</i> (<i>String</i> name) const
<i>Variant</i>	<i>get_setting</i> (<i>String</i> name) const
<i>String</i>	<i>globalize_path</i> (<i>String</i> path) const
<i>bool</i>	<i>has_setting</i> (<i>String</i> name) const
<i>bool</i>	<i>load_resource_pack</i> (<i>String</i> pack)
<i>String</i>	<i>localize_path</i> (<i>String</i> path) const
<i>bool</i>	<i>property_can_revert</i> (<i>String</i> name)
<i>Variant</i>	<i>property_get_revert</i> (<i>String</i> name)
<i>int</i>	<i>save</i> ()
<i>int</i>	<i>save_custom</i> (<i>String</i> file)
void	<i>set_initial_value</i> (<i>String</i> name, <i>Variant</i> value)
void	<i>set_order</i> (<i>String</i> name, <i>int</i> position)
void	<i>set_setting</i> (<i>String</i> name, <i>Variant</i> value)

25.311.3 Description

Contains global variables accessible from everywhere. Use “ProjectSettings.get_setting(variable)”, “ProjectSettings.set_setting(variable,value)” or “ProjectSettings.has_setting(variable)” to access them. Variables stored in project.godot are also loaded into ProjectSettings, making this object very useful for reading custom game configuration options.

25.311.4 Member Function Description

- void **add_property_info** (*Dictionary* hint)

Add a custom property info to a property. The dictionary must contain: name:String), and optionally hint:int), hint_string:String.

Example:

```
ProjectSettings.set("category/property_name", 0)

var property_info = {
    "name": "category/property_name",
    "type": TYPE_INT,
    "hint": PROPERTY_HINT_ENUM,
    "hint_string": "one,two,three"
}

ProjectSettings.add_property_info(property_info)
```

- `void clear (String name)`

Clear the whole configuration (not recommended, may break things).

- `int get_order (String name) const`

Return the order of a configuration value (influences when saved to the config file).

- `Variant get_setting (String name) const`
- `String globalize_path (String path) const`

Convert a localized path (`res://`) to a full native OS path.

- `bool has_setting (String name) const`

Return true if a configuration value is present.

- `bool load_resource_pack (String pack)`
- `String localize_path (String path) const`

Convert a path to a localized path (`res://` path).

- `bool property_can_revert (String name)`
- `Variant property_get_revert (String name)`
- `int save ()`
- `int save_custom (String file)`
- `void set_initial_value (String name, Variant value)`
- `void set_order (String name, int position)`

Set the order of a configuration value (influences when saved to the config file).

- `void set_setting (String name, Variant value)`

25.312 ProximityGroup

Inherits: *Spatial < Node < Object*

Category: Core

25.312.1 Brief Description

General purpose proximity-detection node.

25.312.2 Member Functions

void	broadcast (<i>String</i> name, <i>Variant</i> parameters)
------	---

25.312.3 Signals

- broadcast (*String* group_name, *Array* parameters)

25.312.4 Member Variables

- *DispatchMode* **dispatch_mode**
- *Vector3* **grid_radius**
- *String* **group_name**

25.312.5 Enums

enum **DispatchMode**

- **MODE_PROXY** = 0
- **MODE_SIGNAL** = 1

25.312.6 Description

General purpose proximity-detection node.

25.312.7 Member Function Description

- void broadcast (*String* name, *Variant* parameters)

25.313 ProxyTexture

Inherits: Texture < Resource < Reference < Object

Category: Core

25.313.1 Brief Description

25.313.2 Member Variables

- *Texture* **base**

25.314 QuadMesh

Inherits: *PrimitiveMesh < Mesh < Resource < Reference < Object*

Category: Core

25.314.1 Brief Description

Class representing a square mesh.

25.314.2 Member Variables

- *Vector2 size*

25.314.3 Description

Class representing a square mesh with size (2,2,0). Consider using a *PlaneMesh* if you require a differently sized plane.

25.315 Quat

Category: Built-In Types

25.315.1 Brief Description

Quaternion.

25.315.2 Member Functions

<i>Quat</i>	<i>Quat (float x, float y, float z, float w)</i>
<i>Quat</i>	<i>Quat (Vector3 axis, float angle)</i>
<i>Quat</i>	<i>Quat (Basis from)</i>
<i>Quat</i>	<i>cubic_slerp (Quat b, Quat pre_a, Quat post_b, float t)</i>
<i>float</i>	<i>dot (Quat b)</i>
<i>Quat</i>	<i>inverse ()</i>
<i>bool</i>	<i>is_normalized ()</i>
<i>float</i>	<i>length ()</i>
<i>float</i>	<i>length_squared ()</i>
<i>Quat</i>	<i>normalized ()</i>
<i>Quat</i>	<i>slerp (Quat b, float t)</i>
<i>Quat</i>	<i>slerpni (Quat b, float t)</i>
<i>Vector3</i>	<i>xform (Vector3 v)</i>

25.315.3 Member Variables

- *float w* - W component of the quaternion. Default value: 1
- *float x* - X component of the quaternion. Default value: 0
- *float y* - Y component of the quaternion. Default value: 0
- *float z* - Z component of the quaternion. Default value: 0

25.315.4 Description

A 4-dimensional vector representing a rotation.

The vector represents a 4 dimensional complex number where multiplication of the basis elements is not commutative (multiplying i with j gives a different result than multiplying j with i).

Multiplying quaternions reproduces rotation sequences. However quaternions need to be often renormalized, or else they suffer from precision issues.

It can be used to perform SLERP (spherical-linear interpolation) between two rotations.

25.315.5 Member Function Description

- *Quat Quat (float x, float y, float z, float w)*

Returns a quaternion defined by these values.

- *Quat Quat (Vector3 axis, float angle)*

Returns a quaternion that will rotate around the given axis by the specified angle. The axis must be a normalized vector.

- *Quat Quat (Basis from)*

Returns the rotation matrix corresponding to the given quaternion.

- *Quat cubic_slerp (Quat b, Quat pre_a, Quat post_b, float t)*

Performs a cubic spherical-linear interpolation with another quaternion.

- *float dot (Quat b)*

Returns the dot product of two quaternions.

- *Quat inverse ()*

Returns the inverse of the quaternion.

- *bool is_normalized ()*

Returns whether the quaternion is normalized or not.

- *float length ()*

Returns the length of the quaternion.

- *float length_squared ()*

Returns the length of the quaternion, squared.

- *Quat normalized ()*

Returns a copy of the quaternion, normalized to unit length.

- *Quat slerp (Quat b, float t)*

Performs a spherical-linear interpolation with another quaternion.

- *Quat slerpni (Quat b, float t)*

Performs a spherical-linear interpolation with another quaternion without checking if the rotation path is not bigger than 90°.

- *Vector3 xform (Vector3 v)*

Transforms the vector v by this quaternion.

25.316 Range

Inherits: *Control < CanvasItem < Node < Object*

Inherited By: *SpinBox, ScrollBar, ProgressBar, TextureProgress, Slider*

Category: Core

25.316.1 Brief Description

Abstract base class for range-based controls.

25.316.2 Member Functions

void	<i>share (Node with)</i>
void	<i>unshare ()</i>

25.316.3 Signals

- **changed ()**

This signal is emitted when min, max, range or step change.

- **value_changed (float value)**

This signal is emitted when value changes.

25.316.4 Member Variables

- **bool exp_edit** - If true and min_value is greater than 0, value will be represented exponentially rather than linearly.
- **float max_value** - Maximum value. Range is clamped if value is greater than max_value. Default value: 100.
- **float min_value** - Minimum value. Range is clamped if value is less than min_value. Default value: 0.

- *float page* - Page size. Used mainly for *ScrollBar*. ScrollBar's length is its size multiplied by page over the difference between `min_value` and `max_value`.
- *float ratio* - The value mapped between 0 and 1.
- *bool rounded* - If `true`, `value` will always be rounded to the nearest integer.
- *float step* - If greater than 0, `value` will always be rounded to a multiple of `step`. If `rounded` is also `true`, `value` will first be rounded to a multiple of `step` then rounded to the nearest integer.
- *float value* - Range's current value.

25.316.5 Description

Range is a base class for *Control* nodes that change a floating point *value* between a *minimum* and a *maximum*, using *step* and *page*, for example a *ScrollBar*.

25.316.6 Member Function Description

- void **share** (*Node* with)

Binds two Ranges together along with any Ranges previously grouped with either of them. When any of Range's member variables change, it will share the new value with all other Ranges in its group.

- void **unshare** ()

Stop Range from sharing its member variables with any other Range.

25.317 RayCast

Inherits: *Spatial* < *Node* < *Object*

Category: Core

25.317.1 Brief Description

Query the closest object intersecting a ray.

25.317.2 Member Functions

void	<i>add_exception</i> (<i>Object</i> node)
void	<i>add_exception_rid</i> (<i>RID</i> rid)
void	<i>clear_exceptions</i> ()
void	<i>force_raycast_update</i> ()
<i>Object</i>	<i>get_collider</i> () const
<i>int</i>	<i>get_collider_shape</i> () const
<i>bool</i>	<i>get_collision_mask_bit</i> (<i>int</i> bit) const
<i>Vector3</i>	<i>get_collision_normal</i> () const
<i>Vector3</i>	<i>get_collision_point</i> () const
<i>bool</i>	<i>is_colliding</i> () const
void	<i>remove_exception</i> (<i>Object</i> node)
void	<i>remove_exception_rid</i> (<i>RID</i> rid)
void	<i>set_collision_mask_bit</i> (<i>int</i> bit, <i>bool</i> value)

25.317.3 Member Variables

- *Vector3 cast_to* - The ray's destination point, relative to the RayCast's position.
- *int collision_mask* - The ray's collision mask. Only objects in at least one collision layer enabled in the mask will be detected.
- *bool enabled* - If true collisions will be reported. Default value: false.
- *bool exclude_parent* - If true collisions will be ignored for this RayCast's immediate parent. Default value: true.

25.317.4 Description

A RayCast represents a line from its origin to its destination position, `cast_to`. It is used to query the 3D space in order to find the closest object along the path of the ray.

RayCast can ignore some objects by adding them to the exception list via `add_exception`, by setting proper filtering with collision layers, or by filtering object types with type masks.

Only enabled raycasts will be able to query the space and report collisions.

RayCast calculates intersection every physics frame (see *Node*), and the result is cached so it can be used later until the next frame. If multiple queries are required between physics frames (or during the same frame) use `force_raycast_update` after adjusting the raycast.

25.317.5 Member Function Description

- void **add_exception** (*Object* node)

Adds a collision exception so the ray does not report collisions with the specified node.

- void **add_exception_rid** (*RID* rid)

Adds a collision exception so the ray does not report collisions with the specified *RID*.

- void **clear_exceptions** ()

Removes all collision exceptions for this ray.

- `void force_raycast_update()`

Updates the collision information for the ray.

Use this method to update the collision information immediately instead of waiting for the next `_physics_process` call, for example if the ray or its parent has changed state. Note: `enabled == true` is not required for this to work.

- `Object get.collider()` const

Return the closest object the ray is pointing to. Note that this does not consider the length of the ray, so you must also use `is_colliding` to check if the object returned is actually colliding with the ray.

Example:

```
if RayCast.is_colliding():
    var collider = RayCast.get.collider()
```

- `int get.collider_shape()` const

Returns the collision shape of the closest object the ray is pointing to. Note that this does not consider the length of the ray, so you must also use `is_colliding` to check if the object returned is actually colliding with the ray.

Example:

```
if RayCast.is_colliding():
    var shape = RayCast.get.collider_shape()
```

- `bool get.collision_mask_bit(int bit)` const

Returns `true` if the bit index passed is turned on. Note that bit indexes range from 0-19.

- `Vector3 get.collision_normal()` const

Returns the normal of the intersecting object's shape at the collision point.

- `Vector3 get.collision_point()` const

Returns the collision point at which the ray intersects the closest object. Note: this point is in the **global** coordinate system.

- `bool is_colliding()` const

Return whether the closest object the ray is pointing to is colliding with the vector (considering the vector length).

- `void remove_exception(Object node)`

Removes a collision exception so the ray does report collisions with the specified node.

- `void remove_exception_rid(RID rid)`

Removes a collision exception so the ray does report collisions with the specified `RID`.

- `void set.collision_mask_bit(int bit, bool value)`

Sets the bit index passed to the `value` passed. Note that bit indexes range from 0-19.

25.318 RayCast2D

Inherits: `Node2D < CanvasItem < Node < Object`

Category: Core

25.318.1 Brief Description

Query the closest object intersecting a ray.

25.318.2 Member Functions

void	<i>add_exception (Object node)</i>
void	<i>add_exception_rid (RID rid)</i>
void	<i>clear_exceptions ()</i>
void	<i>force_raycast_update ()</i>
<i>Object</i>	<i>get_collider () const</i>
<i>int</i>	<i>get_collider_shape () const</i>
<i>bool</i>	<i>get_collision_mask_bit (int bit) const</i>
<i>Vector2</i>	<i>get_collision_normal () const</i>
<i>Vector2</i>	<i>get_collision_point () const</i>
<i>bool</i>	<i>is_colliding () const</i>
void	<i>remove_exception (Object node)</i>
void	<i>remove_exception_rid (RID rid)</i>
void	<i>set_collision_mask_bit (int bit, bool value)</i>

25.318.3 Member Variables

- *Vector2 cast_to* - The ray's destination point, relative to the RayCast's position.
- *int collision_mask* - The ray's collision mask. Only objects in at least one collision layer enabled in the mask will be detected.
- *bool enabled* - If `true`, collisions will be reported. Default value: `false`.
- *bool exclude_parent* - If `true`, the parent node will be excluded from collision detection. Default value: `true`.

25.318.4 Description

A RayCast represents a line from its origin to its destination position, `cast_to`. It is used to query the 2D space in order to find the closest object along the path of the ray.

RayCast2D can ignore some objects by adding them to the exception list via `add_exception`, by setting proper filtering with collision layers, or by filtering object types with type masks.

Only enabled raycasts will be able to query the space and report collisions.

RayCast2D calculates intersection every physics frame (see `Node`), and the result is cached so it can be used later until the next frame. If multiple queries are required between physics frames (or during the same frame) use `force_raycast_update` after adjusting the raycast.

25.318.5 Member Function Description

- void `add_exception (Object node)`

Adds a collision exception so the ray does not report collisions with the specified node.

- void `add_exception_rid (RID rid)`

Adds a collision exception so the ray does not report collisions with the specified *RID*.

- void **clear_exceptions ()**

Removes all collision exceptions for this ray.

- void **force_raycast_update ()**

Updates the collision information for the ray. Use this method to update the collision information immediately instead of waiting for the next `_physics_process` call, for example if the ray or its parent has changed state. Note: `enabled == true` is not required for this to work.

- *Object get_collider () const*

Returns the closest object the ray is pointing to. Note that this does not consider the length of the ray, so you must also use `is_colliding` to check if the object returned is actually colliding with the ray.

Example:

```
if RayCast2D.is_colliding():
    var collider = RayCast2D.get_collider()
```

- *int get_collider_shape () const*

Returns the collision shape of the closest object the ray is pointing to. Note that this does not consider the length of the ray, so you must also use `is_colliding` to check if the object returned is actually colliding with the ray.

Example:

```
if RayCast2D.is_colliding():
    var shape = RayCast2D.get_collider_shape()
```

- *bool get_collision_mask_bit (int bit) const*

Return an individual bit on the collision mask.

- *Vector2 get_collision_normal () const*

Returns the normal of the intersecting object's shape at the collision point.

- *Vector2 get_collision_point () const*

Returns the collision point at which the ray intersects the closest object. Note: this point is in the **global** coordinate system.

- *bool is_colliding () const*

Return whether the closest object the ray is pointing to is colliding with the vector (considering the vector length).

- void **remove_exception (Object node)**

Removes a collision exception so the ray does report collisions with the specified node.

- void **remove_exception_rid (RID rid)**

Removes a collision exception so the ray does report collisions with the specified *RID*.

- void **set_collision_mask_bit (int bit, bool value)**

Set/clear individual bits on the collision mask. This makes selecting the areas scanned easier.

25.319 RayShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.319.1 Brief Description

Ray shape for 3D collisions.

25.319.2 Member Variables

- *float length* - The ray's length.

25.319.3 Description

Ray shape for 3D collisions, which can be set into a *PhysicsBody* or *Area*. A ray is not really a collision body, instead it tries to separate itself from whatever is touching its far endpoint. It's often useful for characters.

25.320 RayShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.320.1 Brief Description

Ray shape for 2D collisions.

25.320.2 Member Variables

- *float length* - The ray's length.

25.320.3 Description

Ray shape for 2D collisions. A ray is not really a collision body, instead it tries to separate itself from whatever is touching its far endpoint. It's often useful for characters.

25.321 Rect2

Category: Built-In Types

25.321.1 Brief Description

2D Axis-aligned bounding box.

25.321.2 Member Functions

<i>Rect2</i>	<code>Rect2 (Vector2 position, Vector2 size)</code>
<i>Rect2</i>	<code>Rect2 (float x, float y, float width, float height)</code>
<i>Rect2</i>	<code>abs ()</code>
<i>Rect2</i>	<code>clip (Rect2 b)</code>
<i>bool</i>	<code>encloses (Rect2 b)</code>
<i>Rect2</i>	<code>expand (Vector2 to)</code>
<i>float</i>	<code>get_area ()</code>
<i>Rect2</i>	<code>grow (float by)</code>
<i>Rect2</i>	<code>grow_individual (float left, float top, float right, float bottom)</code>
<i>Rect2</i>	<code>grow_margin (int margin, float by)</code>
<i>bool</i>	<code>has_no_area ()</code>
<i>bool</i>	<code>has_point (Vector2 point)</code>
<i>bool</i>	<code>intersects (Rect2 b)</code>
<i>Rect2</i>	<code>merge (Rect2 b)</code>

25.321.3 Member Variables

- *Vector2 end* - Ending corner.
- *Vector2 position* - Position (starting corner).
- *Vector2 size* - Size from position to end.

25.321.4 Description

`Rect2` consists of a position, a size, and several utility functions. It is typically used for fast overlap tests.

25.321.5 Member Function Description

- *Rect2 Rect2 (Vector2 position, Vector2 size)*

Constructs a `Rect2` by position and size.

- *Rect2 Rect2 (float x, float y, float width, float height)*

Constructs a `Rect2` by x, y, width, and height.

- *Rect2 abs ()*
- *Rect2 clip (Rect2 b)*

Returns the intersection of this `Rect2` and b.

- *bool encloses (Rect2 b)*

Returns true if this `Rect2` completely encloses another one.

- *Rect2 expand (Vector2 to)*

Returns this `Rect2` expanded to include a given point.

- *float get_area ()*

Returns the area of the Rect2.

- *Rect2 grow (float by)*

Returns a copy of the Rect2 grown a given amount of units towards all the sides.

- *Rect2 grow_individual (float left, float top, float right, float bottom)*

Returns a copy of the Rect2 grown a given amount of units towards each direction individually.

- *Rect2 grow_margin (int margin, float by)*

Returns a copy of the Rect2 grown a given amount of units towards the Margin direction.

- *bool has_no_area ()*

Returns true if the Rect2 is flat or empty.

- *bool has_point (Vector2 point)*

Returns true if the Rect2 contains a point.

- *bool intersects (Rect2 b)*

Returns true if the Rect2 overlaps with another.

- *Rect2 merge (Rect2 b)*

Returns a larger Rect2 that contains this Rect2 and with.

25.322 RectangleShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.322.1 Brief Description

Rectangle shape for 2D collisions.

25.322.2 Member Variables

- *Vector2 extents* - The rectangle's half extents. The width and height of this shape is twice the half extents.

25.322.3 Description

Rectangle shape for 2D collisions. This shape is useful for modeling box-like 2D objects.

25.323 Reference

Inherits: *Object*

Inherited By: *RegExMatch, RegEx, EncodedObjectAsID, SpatialGizmo, TriangleMesh, EditorScenePostImport, PhysicsShapeQueryResult, EditorSceneImporter, Physics2DTestMotionResult, FuncRef, File, TCP_Server,*

Physics2DShapeQueryResult, ConfigFile, StreamPeer, GDScriptNativeClass, HTTPClient, AudioStreamPlayback, VisualScriptFunctionState, Resource, KinematicCollision, SurfaceTool, JSONParseResult, SpatialVelocityTracker, EditorResourcePreviewGenerator, Physics2DShapeQueryParameters, EditorExportPlugin, ARVRInterface, EditorScript, Mutex, PacketPeer, Semaphore, XMLParser, EditorImportPlugin, Directory, WeakRef, GDScriptFunctionState, Marshalls, SceneState, PCKPacker, MeshDataTool, AStar, ResourceImporter, EditorResourceConversionPlugin, SceneTreeTimer, Thread, ResourceInteractiveLoader, PackedDataContainerRef, KinematicCollision2D, GDNative, PhysicsShapeQueryParameters

Category: Core

25.323.1 Brief Description

Base class for anything that keeps a reference count.

25.323.2 Member Functions

<i>bool</i>	<i>init_ref ()</i>
<i>bool</i>	<i>reference ()</i>
<i>bool</i>	<i>unreference ()</i>

25.323.3 Description

Base class for anything that keeps a reference count. Resource and many other helper objects inherit this. References keep an internal reference counter so they are only released when no longer in use.

25.323.4 Member Function Description

- *bool init_ref ()*
- *bool reference ()*

Increase the internal reference counter. Use this only if you really know what you are doing.

- *bool unreference ()*

Decrease the internal reference counter. Use this only if you really know what you are doing.

25.324 ReferenceRect

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.324.1 Brief Description

Reference frame for GUI.

25.324.2 Description

Reference frame for GUI. It's just like an empty control, except a red box is displayed while editing around its size at all times.

25.325 ReflectionProbe

Inherits: *VisualInstance < Spatial < Node < Object*

Category: Core

25.325.1 Brief Description

25.325.2 Member Variables

- *bool* **box_projection**
- *int* **cull_mask**
- *bool* **enable_shadows**
- *Vector3* **extents**
- *float* **intensity**
- *Color* **interior_ambient_color**
- *float* **interior_ambient_contrib**
- *float* **interior_ambient_energy**
- *bool* **interior_enable**
- *float* **max_distance**
- *Vector3* **origin_offset**
- *UpdateMode* **update_mode**

25.325.3 Enums

enum **UpdateMode**

- **UPDATE_ONCE = 0**
- **UPDATE_ALWAYS = 1**

25.326 RegEx

Inherits: *Reference < Object*

Category: Core

25.326.1 Brief Description

Class for searching text for patterns using regular expressions.

25.326.2 Member Functions

<code>void</code>	<code>clear ()</code>
<code>int</code>	<code>compile (String pattern)</code>
<code>int</code>	<code>get_group_count () const</code>
<code>Array</code>	<code>get_names () const</code>
<code>String</code>	<code>get_pattern () const</code>
<code>bool</code>	<code>is_valid () const</code>
<code>RegExMatch</code>	<code>search (String subject, int offset=0, int end=-1) const</code>
<code>Array</code>	<code>search_all (String subject, int offset=0, int end=-1) const</code>
<code>String</code>	<code>sub (String subject, String replacement, bool all=false, int offset=0, int end=-1) const</code>

25.326.3 Description

Regular Expression (or regex) is a compact programming language that can be used to recognise strings that follow a specific pattern, such as URLs, email addresses, complete sentences, etc. For instance, a regex of `ab[0-9]` would find any string that is ab followed by any number from 0 to 9. For a more in-depth look, you can easily find various tutorials and detailed explanations on the Internet.

To begin, the `RegEx` object needs to be compiled with the search pattern using `compile` before it can be used.

```
var regex = RegEx.new()
regex.compile("\w-(\d+)"")
```

The search pattern must be escaped first for gdscript before it is escaped for the expression. For example, `compile("\d+")` would be read by `RegEx` as `\d+`. Similarly, `compile("\"(?:\\\\.|[^\"]+)*\\"")` would be read as `"(?:\\.|[^\"]+)*`"

Using `search` you can find the pattern within the given text. If a pattern is found, `RegExMatch` is returned and you can retrieve details of the results using functions such as `RegExMatch.get_string` and `RegExMatch.get_start`.

```
var regex = RegEx.new()
regex.compile("\w-(\d+)"")
var result = regex.search("abc n-0123")
if result:
    print(result.get_string()) # Would print n-0123
```

The results of capturing groups () can be retrieved by passing the group number to the various functions in `RegExMatch`. Group 0 is the default and would always refer to the entire pattern. In the above example, calling `result.get_string(1)` would give you 0123.

This version of `RegEx` also supports named capturing groups, and the names can be used to retrieve the results. If two or more groups have the same name, the name would only refer to the first one with a match.

```
var regex = RegEx.new()
regex.compile("d(?<digit>[0-9]+)|x(?<digit>[0-9a-f]+)"")
var result = regex.search("the number is x2f")
if result:
    print(result.get_string("digit")) # Would print 2f
```

If you need to process multiple results, `search_all` generates a list of all non-overlapping results. This can be combined with a for-loop for convenience.

```
for result in regex.search_all("d01, d03, d0c, x3f and x42"):  
    print(result.get_string("digit"))  
# Would print 01 03 3f 42  
# Note that d0c would not match
```

25.326.4 Member Function Description

- `void clear()`

This method resets the state of the object, as it was freshly created. Namely, it unassigns the regular expression of this object.

- `int compile (String pattern)`

Compiles and assign the search pattern to use. Returns OK if the compilation is successful. If an error is encountered the details are printed to STDOUT and FAILED is returned.

- `int get_group_count () const`

Returns the number of capturing groups in compiled pattern.

- `Array get_names () const`

Returns an array of names of named capturing groups in the compiled pattern. They are ordered by appearance.

- `String get_pattern () const`

Returns the original search pattern that was compiled.

- `bool is_valid () const`

Returns whether this object has a valid search pattern assigned.

- `RegExMatch search (String subject, int offset=0, int end=-1) const`

Searches the text for the compiled pattern. Returns a `RegExMatch` container of the first matching result if found, otherwise null. The region to search within can be specified without modifying where the start and end anchor would be.

- `Array search_all (String subject, int offset=0, int end=-1) const`

Searches the text for the compiled pattern. Returns an array of `RegExMatch` containers for each non-overlapping result. If no results were found an empty array is returned instead. The region to search within can be specified without modifying where the start and end anchor would be.

- `String sub (String subject, String replacement, bool all=false, int offset=0, int end=-1) const`

Searches the text for the compiled pattern and replaces it with the specified string. Escapes and backreferences such as `\1` and `\g<name>` expanded and resolved. By default only the first instance is replaced but it can be changed for all instances (global replacement). The region to search within can be specified without modifying where the start and end anchor would be.

25.327 RegExMatch

Inherits: `Reference < Object`

Category: Core

25.327.1 Brief Description

Contains the results of a regex search.

25.327.2 Member Functions

<i>int</i>	<code>get_end (Variant name=0) const</code>
<i>int</i>	<code>get_group_count () const</code>
<i>int</i>	<code>get_start (Variant name=0) const</code>
<i>String</i>	<code>get_string (Variant name=0) const</code>

25.327.3 Member Variables

- *Dictionary names* - A dictionary of named groups and its corresponding group number. Only groups with that were matched are included. If multiple groups have the same name, that name would refer to the first matching one.
- *Array strings* - An *Array* of the match and its capturing groups.
- *String subject* - The source string used with the search pattern to find this matching result.

25.327.4 Description

Contains the results of a single regex match returned by `RegEx.search` and `RegEx.search_all`. It can be used to find the position and range of the match and its capturing groups, and it can extract its sub-string for you.

25.327.5 Member Function Description

- `int get_end (Variant name=0) const`

Returns the end position of the match within the source string. The end position of capturing groups can be retrieved by providing its group number as an integer or its string name (if it's a named group). The default value of 0 refers to the whole pattern.

Returns -1 if the group did not match or doesn't exist.

- `int get_group_count () const`

Returns the number of capturing groups.

- `int get_start (Variant name=0) const`

Returns the starting position of the match within the source string. The starting position of capturing groups can be retrieved by providing its group number as an integer or its string name (if it's a named group). The default value of 0 refers to the whole pattern.

Returns -1 if the group did not match or doesn't exist.

- `String get_string (Variant name=0) const`

Returns the substring of the match from the source string. Capturing groups can be retrieved by providing its group number as an integer or its string name (if it's a named group). The default value of 0 refers to the whole pattern.

Returns an empty string if the group did not match or doesn't exist.

25.328 RemoteTransform

Inherits: *Spatial < Node < Object*

Category: Core

25.328.1 Brief Description

RemoteTransform leads the *Transform* of another *Spatial* derived Node in the scene.

25.328.2 Member Variables

- *NodePath remote_path* - The *NodePath* to the remote node, relative to the RemoteTransform's position in the scene.
- *bool update_position* - If `true` the remote node's position is tracked. Default value: `true`.
- *bool update_rotation* - If `true` the remote node's rotation is tracked. Default value: `true`.
- *bool update_scale* - If `true` the remote node's scale is tracked. Default value: `true`.
- *bool use_global_coordinates* - If `true` global coordinates are used. If `false` local coordinates are used. Default value: `true`.

25.328.3 Description

RemoteTransform leads the *Transform* of another *Spatial* derived Node (called the remote node) in the scene.

It can be set to track another Node's position, rotation and/or scale. It can update using either global or local coordinates.

25.329 RemoteTransform2D

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.329.1 Brief Description

RemoteTransform2D leads the *Transform2D* of another *CanvasItem* derived Node in the scene.

25.329.2 Member Variables

- *NodePath remote_path* - The *NodePath* to the remote node, relative to the RemoteTransform2D's position in the scene.
- *bool update_position* - If `true` the remote node's position is tracked. Default value: `true`.
- *bool update_rotation* - If `true` the remote node's rotation is tracked. Default value: `true`.

- *bool update_scale* - If `true` the remote node's scale is tracked. Default value: `true`.
- *bool use_global_coordinates* - If `true` global coordinates are used. If `false` local coordinates are used. Default value: `true`.

25.329.3 Description

`RemoteTransform2D` leads the `Transform2D` of another `CanvasItem` derived Node (called the remote node) in the scene.

It can be set to track another Node's position, rotation and/or scale. It can update using either global or local coordinates.

25.330 Resource

Inherits: `Reference < Object`

Inherited By: `Theme, AudioStream, Sky, CubeMap, DynamicFontData, InputEvent, Translation, Curve2D, Shape, Shape2D, Curve, StyleBox, Environment, GDNativeLibrary, Material, VideoStream, PackedScene, Texture, Mesh, ButtonGroup, TileSet, ShortCut, BitMap, Animation, BakedLightmapData, PolygonPathFinder, Shader, Script, AudioBusLayout, World, AudioEffect, VisualScriptNode, World2D, GIProbeData, Font, SpriteFrames, MeshLibrary, Curve3D, NavigationPolygon, EditorSettings, Image, Gradient, OccluderPolygon2D, MultiMesh, PackedDataContainer, NavigationMesh`

Category: Core

25.330.1 Brief Description

Base class for all resources.

25.330.2 Member Functions

<code>void</code>	<code>_setup_local_to_scene () virtual</code>
<code>Resource</code>	<code>duplicate (bool subresources=false) const</code>
<code>Node</code>	<code>get_local_scene () const</code>
<code>RID</code>	<code>get_rid () const</code>
<code>void</code>	<code>setup_local_to_scene ()</code>
<code>void</code>	<code>take_over_path (String path)</code>

25.330.3 Signals

- `changed ()`

25.330.4 Member Variables

- *bool resource_local_to_scene*
- *String resource_name*

- *String resource_path*

25.330.5 Description

Resource is the base class for all resource types. Resources are primarily data containers. They are reference counted and freed when no longer in use. They are also loaded only once from disk, and further attempts to load the resource will return the same reference (all this in contrast to a *Node*, which is not reference counted and can be instanced from disk as many times as desired). Resources can be saved externally on disk or bundled into another object, such as a *Node* or another resource.

25.330.6 Member Function Description

- void `_setup_local_to_scene ()` virtual
- *Resource* `duplicate (bool subresources=false)` const
- *Node* `get_local_scene ()` const
- *RID* `get_rid ()` const

Return the RID of the resource (or an empty RID). Many resources (such as *Texture*, *Mesh*, etc) are high level abstractions of resources stored in a server, so this function will return the original RID.

- void `setup_local_to_scene ()`
- void `take_over_path (String path)`

Set the path of the resource. Differs from `set_path()`, if another *Resource* exists with “path” it over-takes it, instead of failing.

25.331 ResourceImporter

Inherits: *Reference < Object*

Inherited By: *ResourceImporterTheora, ResourceImporterOGGVorbis, ResourceImporterWebm*

Category: Core

25.331.1 Brief Description

25.332 ResourceImporterOGGVorbis

Inherits: *ResourceImporter < Reference < Object*

Category: Core

25.332.1 Brief Description

25.333 ResourceImporterTheora

Inherits: *ResourceImporter < Reference < Object*

Category: Core

25.333.1 Brief Description

25.334 ResourceImporterWebm

Inherits: *ResourceImporter < Reference < Object*

Category: Core

25.334.1 Brief Description

25.335 ResourceInteractiveLoader

Inherits: *Reference < Object*

Category: Core

25.335.1 Brief Description

Interactive Resource Loader.

25.335.2 Member Functions

<i>Resource</i>	<i>get_resource ()</i>
<i>int</i>	<i>get_stage () const</i>
<i>int</i>	<i>get_stage_count () const</i>
<i>int</i>	<i>poll ()</i>
<i>int</i>	<i>wait ()</i>

25.335.3 Description

Interactive Resource Loader. This object is returned by ResourceLoader when performing an interactive load. It allows to load with high granularity, so this is mainly useful for displaying load bars/percentages.

25.335.4 Member Function Description

- *Resource get_resource ()*

Return the loaded resource (only if loaded). Otherwise, returns null.

- *int get_stage () const*

Return the load stage. The total amount of stages can be queried with *get_stage_count*

- *int get_stage_count () const*

Return the total amount of stages (calls to *poll*) needed to completely load this resource.

- *int poll ()*

Poll the load. If OK is returned, this means poll will have to be called again. If ERR_FILE_EOF is returned, then the load has finished and the resource can be obtained by calling *get_resource*.

- `int wait()`

25.336 ResourceLoader

Inherits: `Object`

Category: Core

25.336.1 Brief Description

Resource Loader.

25.336.2 Member Functions

<code>PoolStringArray</code>	<code>get_dependencies (String path)</code>
<code>PoolStringArray</code>	<code>get_recognized_extensions_for_type (String type)</code>
<code>bool</code>	<code>has (String path)</code>
<code>Resource</code>	<code>load (String path, String type_hint="" , bool p_no_cache=false)</code>
<code>ResourceInteractiveLoader</code>	<code>load_interactive (String path, String type_hint="")</code>
<code>void</code>	<code>set_abort_on_missing_resources (bool abort)</code>

25.336.3 Description

Resource Loader. This is a static object accessible as `ResourceLoader`. GDScript has a simplified `load()` function, though.

25.336.4 Member Function Description

- `PoolStringArray get_dependencies (String path)`
- `PoolStringArray get_recognized_extensions_for_type (String type)`

Return the list of recognized extensions for a resource type.

- `bool has (String path)`
- `Resource load (String path, String type_hint="" , bool p_no_cache=false)`
- `ResourceInteractiveLoader load_interactive (String path, String type_hint="")`

Load a resource interactively, the returned object allows to load with high granularity.

- `void set_abort_on_missing_resources (bool abort)`

Change the behavior on missing sub-resources. Default is to abort load.

25.337 ResourcePreloader

Inherits: `Node < Object`

Category: Core

25.337.1 Brief Description

Resource Preloader Node.

25.337.2 Member Functions

void	<i>add_resource (String name, Resource resource)</i>
<i>Resource</i>	<i>get_resource (String name) const</i>
<i>PoolStringArray</i>	<i>get_resource_list () const</i>
<i>bool</i>	<i>has_resource (String name) const</i>
void	<i>remove_resource (String name)</i>
void	<i>rename_resource (String name, String newname)</i>

25.337.3 Description

Resource Preloader Node. This node is used to preload sub-resources inside a scene, so when the scene is loaded all the resources are ready to use and be retrieved from here.

25.337.4 Member Function Description

- void **add_resource** (*String name, Resource resource*)
- *Resource get_resource (String name) const*

Return the resource given a text-id.

- *PoolStringArray get_resource_list () const*

Return the list of resources inside the preloader.

- *bool has_resource (String name) const*

Return true if the preloader has a given resource.

- void **remove_resource** (*String name*)

Remove a resource from the preloader by text id.

- void **rename_resource** (*String name, String newname*)

Rename a resource inside the preloader, from a text-id to a new text-id.

25.338 ResourceSaver

Inherits: *Object*

Category: Core

25.338.1 Brief Description

Resource Saving Interface.

25.338.2 Member Functions

<i>PoolStringArray</i>	<code>get_recognized_extensions (Resource type)</code>
<i>int</i>	<code>save (String path, Resource resource, int flags=0)</code>

25.338.3 Enums

enum **SaverFlags**

- **FLAG_RELATIVE_PATHS** = 1
- **FLAG_BUNDLE_RESOURCES** = 2
- **FLAG_CHANGE_PATH** = 4
- **FLAG OMIT_EDITOR_PROPERTIES** = 8
- **FLAG_SAVE_BIG_ENDIAN** = 16
- **FLAG_COMPRESS** = 32

25.338.4 Description

Resource Saving Interface. This interface is used for saving resources to disk.

25.338.5 Member Function Description

- *PoolStringArray* **get_recognized_extensions** (*Resource* type)

Return the list of extensions available for saving a resource of a given type.

- *int* **save** (*String* path, *Resource* resource, *int* flags=0)

Save a resource to disk, to a given path.

25.339 RichTextLabel

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

25.339.1 Brief Description

Label that displays rich text.

25.339.2 Member Functions

void	<code>add_image (Texture image)</code>
void	<code>add_text (String text)</code>
int	<code>append_bbcode (String bbcode)</code>
void	<code>clear ()</code>
int	<code>get_line_count () const</code>
int	<code>get_total_character_count () const</code>
VScrollBar	<code>get_v_scroll ()</code>
int	<code>get_visible_line_count () const</code>
void	<code>newline ()</code>
int	<code>parse_bbcode (String bbcode)</code>
void	<code>pop ()</code>
void	<code>push_align (int align)</code>
void	<code>push_cell ()</code>
void	<code>push_color (Color color)</code>
void	<code>push_font (Font font)</code>
void	<code>push_indent (int level)</code>
void	<code>push_list (int type)</code>
void	<code>push_meta (Variant data)</code>
void	<code>push_table (int columns)</code>
void	<code>push_underline ()</code>
bool	<code>remove_line (int line)</code>
void	<code>scroll_to_line (int line)</code>
void	<code>set_table_column_expand (int column, bool expand, int ratio)</code>

25.339.3 Signals

- **meta_clicked** (*Nil meta*)

Triggered when the user clicks on content between url tags. If the meta is defined in text, e.g. `[url={"data"="hi"}]hi[/url]`, then the parameter for this signal will be a *String* type. If a particular type or an object is desired, the `push_meta` method must be used to manually insert the data into the tag stack.

- **meta_hover_ended** (*Nil meta*)

Triggers when the mouse exits a meta tag.

- **meta_hover_started** (*Nil meta*)

Triggers when the mouse enters a meta tag.

25.339.4 Member Variables

- *bool bbcode_enabled* - If `true` the label uses BBCode formatting. Default value: `false`.
- *String bbcode_text* - The label's text in BBCode format. Is not representative of manual modifications to the internal tag stack. Erases changes made by other methods when edited.
- *bool meta_underlined* - If `true`, the label underlines meta tags such as `url{text}`. Default value: `true`.
- *bool override_selected_font_color* - If `true` the label uses the custom font color. Default value: `false`.

- *float percent_visible* - The text's visibility, as a *float* between 0.0 and 1.0.
- *bool scroll_active* - If `true`, the scrollbar is visible. Does not block scrolling completely. See `scroll_to_line`. Default value: `true`.
- *bool scroll_following* - If `true`, the window scrolls down to display new content automatically. Default value: `false`.
- *bool selection_enabled* - If `true`, the label allows text selection.
- *int tab_size* - The number of spaces associated with a single tab length. Does not affect “t” in text tags, only indent tags.
- *String text* - The raw text of the label.

When set, clears the tag stack and adds a raw text tag to the top of it. Does not parse bbcodes. Does not modify `bbcde_text`.

- *int visible_characters* - The restricted number of characters to display in the label.

25.339.5 Enums

enum **ListType**

- **LIST_NUMBERS = 0**
- **LIST_LETTERS = 1**
- **LIST_DOTS = 2**

enum **Align**

- **ALIGN_LEFT = 0**
- **ALIGN_CENTER = 1**
- **ALIGN_RIGHT = 2**
- **ALIGN_FILL = 3**

enum **ItemType**

- **ITEM_FRAME = 0**
- **ITEM_TEXT = 1**
- **ITEM_IMAGE = 2**
- **ITEM_NEWLINE = 3**
- **ITEM_FONT = 4**
- **ITEM_COLOR = 5**
- **ITEM_UNDERLINE = 6**
- **ITEM_ALIGN = 7**
- **ITEM_INDENT = 8**
- **ITEM_LIST = 9**
- **ITEM_TABLE = 10**

- **ITEM_META = 11**

25.339.6 Description

Rich text can contain custom text, fonts, images and some basic formatting. The label manages these as an internal tag stack. It also adapts itself to given width/heights.

Note that assignments to `bbcode_text` clear the tag stack and reconstruct it from the property's contents. Any edits made to `bbcode_text` will erase previous edits made from other manual sources such as `append_bbcode` and the `push__*` / `pop` methods.

25.339.7 Member Function Description

- `void add_image (Texture image)`

Adds an image's opening and closing tags to the tag stack.

- `void add_text (String text)`

Adds raw non-bbcode-parsed text to the tag stack.

- `int append_bbcode (String bbcode)`

Parses `bbcode` and adds tags to the tag stack as needed. Returns the result of the parsing, `OK` if successful.

- `void clear ()`

Clears the tag stack and sets `bbcode_text` to an empty string.

- `int get_line_count () const`

Returns the total number of newlines in the tag stack's text tags. Considers wrapped text as one line.

- `int get_total_character_count () const`

Returns the total number of characters from text tags. Does not include bbcodes.

- `VScrollBar get_v_scroll ()`

Returns the vertical scrollbar.

- `int get_visible_line_count () const`

Returns the number of visible lines.

- `void newline ()`

Adds a newline tag to the tag stack.

- `int parse_bbcode (String bbcode)`

The assignment version of `append_bbcode`. Clears the tag stack and inserts the new content. Returns `OK` if parses `bbcode` successfully.

- `void pop ()`

Terminates the current tag. Use after `push__*` methods to close bbcodes manually. Does not need to follow `add__*` methods.

- `void push_align (int align)`

Adds a `[right]` tag to the tag stack.

- `void push_cell ()`

Adds a [cell] tag to the tag stack. Must be inside a table tag. See `push_table` for details.

- `void push_color (Color color)`

Adds a [color] tag to the tag stack.

- `void push_font (Font font)`

Adds a [font] tag to the tag stack. Overrides default fonts for its duration.

- `void push_indent (int level)`

Adds an [indent] tag to the tag stack. Multiplies “level” by current `tab_size` to determine new margin length.

- `void push_list (int type)`

Adds a list tag to the tag stack. Similar to the bbcodes [ol] or [ul], but supports more list types. Not fully implemented!

- `void push_meta (Variant data)`

Adds a meta tag to the tag stack. Similar to the bbcode [url=something]{text}[/url], but supports non-`String` metadata types.

- `void push_table (int columns)`

Adds a [table=columns] tag to the tag stack.

- `void push_underline ()`

Adds a [u] tag to the tag stack.

- `bool remove_line (int line)`

Removes a line of content from the label. Returns `true` if the line exists.

- `void scroll_to_line (int line)`

Scrolls the window’s top line to match `line`.

- `void set_table_column_expand (int column, bool expand, int ratio)`

Edits the selected columns expansion options. If `expand` is `true`, the column expands in proportion to its expansion ratio versus the other columns’ ratios.

For example, 2 columns with ratios 3 and 4 plus 70 pixels in available width would expand 30 and 40 pixels, respectively.

Columns with a `false` `expand` will not contribute to the total ratio.

25.340 RID

Category: Built-In Types

25.340.1 Brief Description

Handle for a *Resource*’s unique ID.

25.340.2 Member Functions

<i>RID</i>	<i>RID (Object from)</i>
<i>int</i>	<i>get_id ()</i>

25.340.3 Description

The RID type is used to access the unique integer ID of a resource. They are opaque, so they do not grant access to the associated resource by themselves. They are used by and with the low-level Server classes such as *VisualServer*.

25.340.4 Member Function Description

- *RID RID (Object from)*

Create a new RID instance with the ID of a given resource. When not handed a valid resource, silently stores the unused ID 0.

- *int get_id ()*

Retrieve the ID of the referenced resource.

25.341 RigidBody

Inherits: *PhysicsBody < CollisionObject < Spatial < Node < Object*

Inherited By: *VehicleBody*

Category: Core

25.341.1 Brief Description

Physics Body whose position is determined through physics simulation in 3D space.

25.341.2 Member Functions

<i>void</i>	<i>_integrate_forces (PhysicsDirectBodyState state) virtual</i>
<i>void</i>	<i>apply_impulse (Vector3 position, Vector3 impulse)</i>
<i>Array</i>	<i>get_colliding_bodies () const</i>
<i>void</i>	<i>set_axis_velocity (Vector3 axis_velocity)</i>

25.341.3 Signals

- **body_entered (Object body)**

Emitted when a body enters into contact with this one. Contact monitor and contacts reported must be enabled for this to work.

- **body_exited (Object body)**

Emitted when a body shape exits contact with this one. Contact monitor and contacts reported must be enabled for this to work.

- **body_shape_entered** (*int body_id, Object body, int body_shape, int local_shape*)

Emitted when a body enters into contact with this one. Contact monitor and contacts reported must be enabled for this to work.

This signal not only receives the body that collided with this one, but also its *RID* (*body_id*), the shape index from the colliding body (*body_shape*), and the shape index from this body (*local_shape*) the other body collided with.

- **body_shape_exited** (*int body_id, Object body, int body_shape, int local_shape*)

Emitted when a body shape exits contact with this one. Contact monitor and contacts reported must be enabled for this to work.

This signal not only receives the body that stopped colliding with this one, but also its *RID* (*body_id*), the shape index from the colliding body (*body_shape*), and the shape index from this body (*local_shape*) the other body stopped colliding with.

- **sleeping_state_changed** ()

Emitted when the body changes its sleeping state. Either by sleeping or waking up.

25.341.4 Member Variables

- *float angular_damp* - Damps RigidBody's rotational forces.
- *Vector3 angular_velocity* - RigidBody's rotational velocity.
- *bool axis_lock_angular_x*
- *bool axis_lock_angular_y*
- *bool axis_lock_angular_z*
- *bool axis_lock_linear_x*
- *bool axis_lock_linear_y*
- *bool axis_lock_linear_z*
- *float bounce* - RigidBody's bounciness.
- *bool can_sleep* - If `true` the RigidBody will not calculate forces and will act as a static body while there is no movement. It will wake up when forces are applied through other collisions or when the `apply_impulse` method is used.
- *bool contact_monitor* - If `true`, the RigidBody will emit signals when it collides with another RigidBody.
- *int contacts_reported* - The maximum contacts to report. Bodies can keep a log of the contacts with other bodies, this is enabled by setting the maximum amount of contacts reported to a number greater than 0.
- *bool continuous_cd* - If `true` continuous collision detection is used.

Continuous collision detection tries to predict where a moving body will collide, instead of moving it and correcting its movement if it collided. Continuous collision detection is more precise, and misses less impacts by small, fast-moving objects. Not using continuous collision detection is faster to compute, but can miss small, fast-moving objects.

- *bool custom_integrator* - If `true` internal force integration will be disabled (like gravity or air friction) for this body. Other than collision response, the body will only move as determined by the `_integrate_forces` function, if defined.
- *float friction* - The body friction, from 0 (frictionless) to 1 (max friction).
- *float gravity_scale* - This is multiplied by the global 3D gravity setting found in “Project > Project Settings > Physics > 3d” to produce RigidBody’s gravity. E.g. a value of 1 will be normal gravity, 2 will apply double gravity, and 0.5 will apply half gravity to this object.
- *float linear_damp* - RigidBody’s linear damp. Default value: -1, cannot be less than -1. If this value is different from -1, any linear damp derived from the world or areas will be overridden.
- *Vector3 linear_velocity* - RigidBody’s linear velocity. Can be used sporadically, but **DON’T SET THIS IN EVERY FRAME**, because physics may run in another thread and runs at a different granularity. Use `_integrate_forces` as your process loop for precise control of the body state.
- *float mass* - RigidBody’s mass.
- *Mode mode* - The body mode from the `MODE_*` enum. Modes include: `MODE_STATIC`, `MODE_KINEMATIC`, `MODE_RIGID`, and `MODE_CHARACTER`.
- *bool sleeping* - If `true` RigidBody is sleeping and will not calculate forces until woken up by a collision or the `apply_impulse` method.
- *float weight* - RigidBody’s weight based on its mass and the global 3D gravity. Global values are set in “Project > Project Settings > Physics > 3d”.

25.341.5 Enums

enum Mode

- **MODE_RIGID = 0** — Rigid body. This is the “natural” state of a rigid body. It is affected by forces, and can move, rotate, and be affected by user code.
- **MODE_STATIC = 1** — Static mode. The body behaves like a *StaticBody*, and can only move by user code.
- **MODE_CHARACTER = 2** — Character body. This behaves like a rigid body, but can not rotate.
- **MODE_KINEMATIC = 3** — Kinematic body. The body behaves like a *KinematicBody*, and can only move by user code.

25.341.6 Description

This is the node that implements full 3D physics. This means that you do not control a RigidBody directly. Instead you can apply forces to it (gravity, impulses, etc.), and the physics simulation will calculate the resulting movement, collision, bouncing, rotating, etc.

This node can use custom force integration, for writing complex physics motion behavior per node.

This node can shift state between regular Rigid body, Kinematic, Character or Static.

Character mode forbids this node from being rotated.

As a warning, don’t change RigidBody’s position every frame or very often. Sporadic changes work fine, but physics runs at a different granularity (fixed hz) than usual rendering (process callback) and maybe even in a separate thread, so changing this from a process loop will yield strange behavior.

25.341.7 Member Function Description

- void **_integrate_forces** (*PhysicsDirectBodyState* state) virtual

Called during physics processing, allowing you to read and safely modify the simulation state for the object. By default it works in addition to the usual physics behavior, but `set_use_custom_integrator` allows you to disable the default behavior and do fully custom force integration for a body.

- void **apply_impulse** (*Vector3* position, *Vector3* impulse)

Apply a positioned impulse (which will be affected by the body mass and shape). This is the equivalent of hitting a billiard ball with a cue: a force that is applied once, and only once. Both the impulse and the position are in global coordinates, and the position is relative to the object's origin.

- *Array* **get_colliding_bodies** () const

Return a list of the bodies colliding with this one. By default, number of max contacts reported is at 0 , see `set_max_contacts_reported` to increase it. Note that the result of this test is not immediate after moving objects. For performance, list of collisions is updated once per frame and before the physics step. Consider using signals instead.

- void **set_axis_velocity** (*Vector3* axis_velocity)

Set an axis velocity. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

25.342 RigidBody2D

Inherits: *PhysicsBody2D* < *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

25.342.1 Brief Description

A body that is controlled by the 2D physics engine.

25.342.2 Member Functions

void	_integrate_forces (<i>Physics2DDirectBodyState</i> state) virtual
void	add_force (<i>Vector2</i> offset, <i>Vector2</i> force)
void	apply_impulse (<i>Vector2</i> offset, <i>Vector2</i> impulse)
<i>Array</i>	get_colliding_bodies () const
void	set_axis_velocity (<i>Vector2</i> axis_velocity)
<i>bool</i>	test_motion (<i>Vector2</i> motion, <i>float</i> margin=0.08, <i>Physics2DTestMotionResult</i> result=null)

25.342.3 Signals

- **body_entered** (*Object* body)

Emitted when a body enters into contact with this one. `contact_monitor` must be `true` and `contacts_reported` greater than 0 .

- **body_exited** (*Object* body)

Emitted when a body exits contact with this one. *contact_monitor* must be `true` and *contacts_reported* greater than 0.

- **body_shape_entered** (*int body_id, Object body, int body_shape, int local_shape*)

Emitted when a body enters into contact with this one. Reports colliding shape information. See *CollisionObject2D* for shape index information. *contact_monitor* must be `true` and *contacts_reported* greater than 0.

- **body_shape_exited** (*int body_id, Object body, int body_shape, int local_shape*)

Emitted when a body shape exits contact with this one. Reports colliding shape information. See *CollisionObject2D* for shape index information. *contact_monitor* must be `true` and *contacts_reported* greater than 0.

- **sleeping_state_changed** ()

Emitted when *sleeping* changes.

25.342.4 Member Variables

- **float angular_damp** - Damps the body's *angular_velocity*. If `-1` the body will use the "Default Angular Damp" in "Project > Project Settings > Physics > 2d". Default value: `-1`.
- **float angular_velocity** - The body's rotational velocity.
- **Vector2 applied_force** - The body's total applied force.
- **float applied_torque** - The body's total applied torque.
- **float bounce** - The body's bounciness. Default value: `0`.
- **bool can_sleep** - If `true` the body will not calculate forces and will act as a static body if there is no movement. The body will wake up when other forces are applied via collisions or by using *apply_impulse* or *add_force*. Default value: `true`.
- **bool contact_monitor** - If `true` the body will emit signals when it collides with another *RigidBody2D*. See also *contacts_reported*. Default value: `false`.
- **int contacts_reported** - The maximum number of contacts to report. Default value: `0`.
- **CCDMode continuous_cd** - Continuous collision detection mode. Default value: `CCD_MODE_DISABLED`.

Continuous collision detection tries to predict where a moving body will collide instead of moving it and correcting its movement after collision. Continuous collision detection is slower, but more precise and misses fewer collisions with small, fast-moving objects. Raycasting and shapecasting methods are available. See `CCD_MODE__constants` for details.

- **bool custom_integrator** - If `true` internal force integration is disabled for this body. Aside from collision response, the body will only move as determined by the *_integrate_forces* function.
- **float friction** - The body's friction. Values range from `0` (frictionless) to `1` (maximum friction). Default value: `1`.
- **float gravity_scale** - Multiplies the gravity applied to the body. The body's gravity is calculated from the "Default Gravity" value in "Project > Project Settings > Physics > 2d" and/or any additional gravity vector applied by *Area2Ds*. Default value: `1`.
- **float inertia** - The body's moment of inertia. This is like mass, but for rotation: it determines how much torque it takes to rotate the body. The moment of inertia is usually computed automatically from the mass and the shapes, but this function allows you to set a custom value. Set `0` (or negative) inertia to return to automatically computing it.

- *float linear_damp* - Damps the body's *linear_velocity*. If `-1` the body will use the "Default Linear Damp" in "Project > Project Settings > Physics > 2d". Default value: `-1`.
- *Vector2 linear_velocity* - The body's linear velocity.
- *float mass* - The body's mass. Default value: `1`.
- *Mode mode* - The body's mode. See `MODE__*` constants. Default value: `MODE_RIGID`.
- *bool sleeping* - If `true` the body is sleeping and will not calculate forces until woken up by a collision or by using *apply_impulse* or *add_force*.
- *float weight* - The body's weight based on its mass and the "Default Gravity" value in "Project > Project Settings > Physics > 2d".

25.342.5 Enums

enum CCDMode

- **CCD_MODE_DISABLED = 0** — Continuous collision detection disabled. This is the fastest way to detect body collisions, but can miss small, fast-moving objects.
- **CCD_MODE_CAST_RAY = 1** — Continuous collision detection enabled using raycasting. This is faster than shapecasting but less precise.
- **CCD_MODE_CAST_SHAPE = 2** — Continuous collision detection enabled using shapecasting. This is the slowest CCD method and the most precise.

enum Mode

- **MODE_RIGID = 0** — Rigid mode. The body behaves as a physical object. It collides with other bodies and responds to forces applied to it. This is the default mode.
- **MODE_STATIC = 1** — Static mode. The body behaves like a *StaticBody2D* and does not move.
- **MODE_CHARACTER = 2** — Character mode. Similar to `MODE_RIGID`, but the body can not rotate.
- **MODE_KINEMATIC = 3** — Kinematic mode. The body behaves like a *KinematicBody2D*, and must be moved by code.

25.342.6 Description

This node implements simulated 2D physics. You do not control a *RigidBody2D* directly. Instead you apply forces to it (gravity, impulses, etc.) and the physics simulation calculates the resulting movement based on its mass, friction, and other physical properties.

A *RigidBody2D* has 4 behavior modes (see *mode*):

- **Rigid**: The body behaves as a physical object. It collides with other bodies and responds to forces applied to it. This is the default mode.
- **Static**: The body behaves like a *StaticBody2D* and does not move.
- **Character**: Similar to `Rigid` mode, but the body can not rotate.
- **Kinematic**: The body behaves like a *KinematicBody2D*, and must be moved by code.

Note: You should not change a *RigidBody2D*'s position or *linear_velocity* every frame or even very often. If you need to directly affect the body's state, use *_integrate_forces*, which allows you to directly access the physics state.

If you need to override the default physics behavior, you can write a custom force integration. See *custom_integrator*.

25.342.7 Member Function Description

- `void _integrate_forces (Physics2DDirectBodyState state) virtual`

Allows you to read and safely modify the simulation state for the object. Use this instead of `Node._physics_process` if you need to directly change the body's position or other physics properties. By default it works in addition to the usual physics behavior, but *custom_integrator* allows you to disable the default behavior and write custom force integration for a body.

- `void add_force (Vector2 offset, Vector2 force)`

Adds a positioned force to the body. Both the force and the offset from the body origin are in global coordinates.

- `void apply_impulse (Vector2 offset, Vector2 impulse)`

Applies a positioned impulse to the body (which will be affected by the body mass and shape). This is the equivalent of hitting a billiard ball with a cue: a force that is applied instantaneously. Both the impulse and the offset from the body origin are in global coordinates.

- `Array get_colliding_bodies () const`

Returns a list of the bodies colliding with this one. Use `contacts_reported` to set the maximum number reported. You must also set `contact_monitor` to `true`. Note that the result of this test is not immediate after moving objects. For performance, list of collisions is updated once per frame and before the physics step. Consider using signals instead.

- `void set_axis_velocity (Vector2 axis_velocity)`

Sets the body's velocity on the given axis. The velocity in the given vector axis will be set as the given vector length. This is useful for jumping behavior.

- `bool test_motion (Vector2 motion, float margin=0.08, Physics2DTestMotionResult result=null)`

Returns `true` if a collision would result from moving in the given vector. `margin` increases the size of the shapes involved in the collision detection, and `result` is an object of type `Physics2DTestMotionResult`, which contains additional information about the collision (should there be one).

25.343 SceneState

Inherits: `Reference < Object`

Category: Core

25.343.1 Brief Description

A script interface to a scene file's data.

25.343.2 Member Functions

<i>Array</i>	<code>get_connection_binds (int idx) const</code>
<i>int</i>	<code>get_connection_count () const</code>
<i>int</i>	<code>get_connection_flags (int idx) const</code>
<i>String</i>	<code>get_connection_method (int idx) const</code>
<i>String</i>	<code>get_connection_signal (int idx) const</code>
<i>NodePath</i>	<code>get_connection_source (int idx) const</code>
<i>NodePath</i>	<code>get_connection_target (int idx) const</code>
<i>int</i>	<code>get_node_count () const</code>
<i>PoolStringArray</i>	<code>get_node_groups (int idx) const</code>
<i>int</i>	<code>get_node_index (int idx) const</code>
<i>PackedScene</i>	<code>get_node_instance (int idx) const</code>
<i>String</i>	<code>get_node_instance_placeholder (int idx) const</code>
<i>String</i>	<code>get_node_name (int idx) const</code>
<i>NodePath</i>	<code>get_node_owner_path (int idx) const</code>
<i>NodePath</i>	<code>get_node_path (int idx, bool for_parent=false) const</code>
<i>int</i>	<code>get_node_property_count (int idx) const</code>
<i>String</i>	<code>get_node_property_name (int idx, int prop_idx) const</code>
<i>Variant</i>	<code>get_node_property_value (int idx, int prop_idx) const</code>
<i>String</i>	<code>get_node_type (int idx) const</code>
<i>bool</i>	<code>is_node_instance_placeholder (int idx) const</code>

25.343.3 Enums

enum GenEditState

- **GEN_EDIT_STATE_DISABLED = 0** — If passed to `PackedScene.instance`, blocks edits to the scene state.
- **GEN_EDIT_STATE_INSTANCE = 1** — If passed to `PackedScene.instance`, provides inherited scene resources to the local scene. Requires tools compiled.
- **GEN_EDIT_STATE_MAIN = 2** — If passed to `PackedScene.instance`, provides local scene resources to the local scene. Only the main scene should receive the main edit state. Requires tools compiled.

25.343.4 Description

Maintains a list of resources, nodes, exported and overridden properties, and built-in scripts associated with a scene.

25.343.5 Member Function Description

- *Array* `get_connection_binds (int idx) const`

Returns the list of bound parameters for the signal at `idx`.

- *int* `get_connection_count () const`

Returns the number of signal connections in the scene.

- *int* `get_connection_flags (int idx) const`

Returns the flags for the signal at `idx`. See `Object`'s CONNECT__* flags.

- `String get_connection_method (int idx) const`

Returns the method connected to the signal at `idx`.

- `String get_connection_signal (int idx) const`

Returns the name of the signal at `idx`.

- `NodePath get_connection_source (int idx) const`

Returns the path to the node that owns the signal at `idx`, relative to the root node.

- `NodePath get_connection_target (int idx) const`

Returns the path to the node that owns the method connected to the signal at `idx`, relative to the root node.

- `int get_node_count () const`

Returns the number of nodes in the scene.

- `PoolStringArray get_node_groups (int idx) const`

Returns the list of group names associated with the node at `idx`.

- `int get_node_index (int idx) const`

- `PackedScene get_node_instance (int idx) const`

Returns the scene for the node at `idx` or `null` if the node is not an instance.

- `String get_node_instance_placeholder (int idx) const`

Returns the path to the represented scene file if the node at `idx` is an `InstancePlaceholder`.

- `String get_node_name (int idx) const`

Returns the name of the node at `idx`.

- `NodePath get_node_owner_path (int idx) const`

Returns the path to the owner of the node at `idx`, relative to the root node.

- `NodePath get_node_path (int idx, bool for_parent=false) const`

Returns the path to the node at `idx`.

- `int get_node_property_count (int idx) const`

Returns the number of exported or overridden properties for the node at `idx`.

- `String get_node_property_name (int idx, int prop_idx) const`

Returns the name of the property at `prop_idx` for the node at `idx`.

- `Variant get_node_property_value (int idx, int prop_idx) const`

Returns the value of the property at `prop_idx` for the node at `idx`.

- `String get_node_type (int idx) const`

Returns the type of the node at `idx`.

- `bool is_node_instance_placeholder (int idx) const`

Returns `true` if the node at `idx` is an `InstancePlaceholder`.

25.344 SceneTree

Inherits: *MainLoop < Object*

Category: Core

25.344.1 Brief Description

25.344.2 Member Functions

<i>Variant</i>	<code>call_group (String group, String method) vararg</code>
<i>Variant</i>	<code>call_group_flags (int flags, String group, String method) vararg</code>
<i>int</i>	<code>change_scene (String path)</code>
<i>int</i>	<code>change_scene_to (PackedScene packed_scene)</code>
<i>SceneTreeTimer</i>	<code>create_timer (float time_sec, bool pause_mode_process=true)</code>
<i>int</i>	<code>get_frame () const</code>
<i>PoolIntArray</i>	<code>get_network_connected_peers () const</code>
<i>int</i>	<code>get_network_unique_id () const</code>
<i>int</i>	<code>get_node_count () const</code>
<i>Array</i>	<code>get_nodes_in_group (String group)</code>
<i>int</i>	<code>get_rpc_sender_id () const</code>
<i>bool</i>	<code>has_group (String name) const</code>
<i>bool</i>	<code>has_network_peer () const</code>
<i>bool</i>	<code>is_input_handled ()</code>
<i>bool</i>	<code>is_network_server () const</code>
<i>void</i>	<code>notify_group (String group, int notification)</code>
<i>void</i>	<code>notify_group_flags (int call_flags, String group, int notification)</code>
<i>void</i>	<code>queue_delete (Object obj)</code>
<i>void</i>	<code>quit ()</code>
<i>int</i>	<code>reload_current_scene ()</code>
<i>void</i>	<code>set_auto_accept_quit (bool enabled)</code>
<i>void</i>	<code>set_group (String group, String property, Variant value)</code>
<i>void</i>	<code>set_group_flags (int call_flags, String group, String property, Variant value)</code>
<i>void</i>	<code>set_input_as_handled ()</code>
<i>void</i>	<code>set_quit_on_go_back (bool enabled)</code>
<i>void</i>	<code>set_screen_stretch (int mode, int aspect, Vector2 minsize, float shrink=1)</code>

25.344.3 Signals

- **connected_to_server ()**
- **connection_failed ()**
- **files_dropped (PoolStringArray files, int screen)**
- **idle_frame ()**
- **network_peer_connected (int id)**
- **network_peer_disconnected (int id)**
- **node_added (Object node)**

- **node_configuration_warning_changed** (*Object* node)
- **node_removed** (*Object* node)
- **physics_frame** ()
- **screen_resized** ()
- **server_disconnected** ()
- **tree_changed** ()

25.344.4 Member Variables

- *Node* **current_scene**
- *bool* **debug_collisions_hint**
- *bool* **debug_navigation_hint**
- *Node* **edited_scene_root**
- *NetworkedMultiplayerPeer* **network_peer** - The peer object to handle the RPC system (effectively enabling networking when set). Depending on the peer itself, the SceneTree will become a network server (check with `is_network_server()`) and will set root node's network mode to master (see `NETWORK_MODE_*` constants in *Node*), or it will become a regular peer with root node set to slave. All child nodes are set to inherit the network mode by default. Handling of networking-related events (connection, disconnection, new clients) is done by connecting to SceneTree's signals.
- *bool* **paused**
- *bool* **refuse_new_network_connections**
- *Viewport* **root**
- *bool* **use_font_oversampling**

25.344.5 Enums

enum **GroupCallFlags**

- **GROUP_CALL_DEFAULT** = 0
- **GROUP_CALL_REVERSE** = 1
- **GROUP_CALL_REALTIME** = 2
- **GROUP_CALL_UNIQUE** = 4

enum **StretchMode**

- **STRETCH_MODE_DISABLED** = 0
- **STRETCH_MODE_2D** = 1
- **STRETCH_MODE_VIEWPORT** = 2

enum **StretchAspect**

- **STRETCH_ASPECT_IGNORE** = 0

- **STRETCH_ASPECT_KEEP = 1**
- **STRETCH_ASPECT_KEEP_WIDTH = 2**
- **STRETCH_ASPECT_KEEP_HEIGHT = 3**
- **STRETCH_ASPECT_EXPAND = 4**

25.344.6 Member Function Description

- *Variant call_group* (*String group, String method*) vararg
- *Variant call_group_flags* (*int flags, String group, String method*) vararg
- *int change_scene* (*String path*)
- *int change_scene_to* (*PackedScene packed_scene*)
- *SceneTreeTimer create_timer* (*float time_sec, bool pause_mode_process=true*)
- *int get_frame* () const
- *PoolIntArray get_network_connected_peers* () const
- *int get_network_unique_id* () const
- *int get_node_count* () const
- *Array get_nodes_in_group* (*String group*)
- *int get_rpc_sender_id* () const
- *bool has_group* (*String name*) const
- *bool has_network_peer* () const

Returns true if there is a *NetworkedMultiplayerPeer* set (with SceneTree.set_network_peer).

- *bool is_input_handled* ()
- *bool is_network_server* () const

Returns true if this SceneTree's *NetworkedMultiplayerPeer* is in server mode (listening for connections).

- *void notify_group* (*String group, int notification*)
- *void notify_group_flags* (*int call_flags, String group, int notification*)
- *void queue_delete* (*Object obj*)
- *void quit* ()
- *int reload_current_scene* ()
- *void set_auto_accept_quit* (*bool enabled*)
- *void set_group* (*String group, String property, Variant value*)
- *void set_group_flags* (*int call_flags, String group, String property, Variant value*)
- *void set_input_as_handled* ()
- *void set_quit_on_go_back* (*bool enabled*)
- *void set_screen_stretch* (*int mode, int aspect, Vector2 minsize, float shrink=1*)

25.345 SceneTreeTimer

Inherits: *Reference < Object*

Category: Core

25.345.1 Brief Description

25.345.2 Signals

- `timeout ()`

25.345.3 Member Variables

- `float time_left`

25.346 Script

Inherits: *Resource < Reference < Object*

Inherited By: *VisualScript, GDScript, CSharpScript, NativeScript, PluginScript*

Category: Core

25.346.1 Brief Description

A class stored as a resource.

25.346.2 Member Functions

<code>bool</code>	<code>can_instance () const</code>
<code>Script</code>	<code>get_base_script () const</code>
<code>String</code>	<code>get_instance_base_type () const</code>
<code>bool</code>	<code>has_script_signal (String signal_name) const</code>
<code>bool</code>	<code>has_source_code () const</code>
<code>bool</code>	<code>instance_has (Object base_object) const</code>
<code>bool</code>	<code>is_tool () const</code>
<code>int</code>	<code>reload (bool keep_state=false)</code>

25.346.3 Member Variables

- **`String source_code`** - The script source code, or an empty string if source code is not available. When set, does not reload the class implementation automatically.

25.346.4 Description

A class stored as a resource. The script extends the functionality of all objects that instance it.

The ‘new’ method of a script subclass creates a new instance. `Object.set_script` extends an existing object, if that object’s class matches one of the script’s base classes.

25.346.5 Member Function Description

- `bool can_instance () const`

Returns true if the script can be instanced.

- `Script get_base_script () const`
- `String get_instance_base_type () const`
- `bool has_script_signal (String signal_name) const`

Returns true if the script, or a base class, defines a signal with the given name.

- `bool has_source_code () const`

Returns true if the script contains non-empty source code.

- `bool instance_has (Object base_object) const`

Returns true if ‘base_object’ is an instance of this script.

- `bool is_tool () const`

Returns true if the script is a tool script. A tool script can run in the editor.

- `int reload (bool keep_state=false)`

Reloads the script’s class implementation. Returns an error code.

25.347 ScriptEditor

Inherits: `PanelContainer < Container < Control < CanvasItem < Node < Object`

Category: Core

25.347.1 Brief Description

25.347.2 Member Functions

<code>bool</code>	<code>can_drop_data_fw (Vector2 point, Variant data, Control from) const</code>
<code>void</code>	<code>drop_data_fw (Vector2 point, Variant data, Control from)</code>
<code>Script</code>	<code>get_current_script ()</code>
<code>Variant</code>	<code>get_drag_data_fw (Vector2 point, Control from)</code>
<code>Array</code>	<code>get_open_scripts () const</code>
<code>void</code>	<code>open_script_create_dialog (String base_name, String base_path)</code>

25.347.3 Signals

- **editor_script_changed** (*Object script*)

Emitted when user changed active script. Argument is a freshly activated *Script*.

- **script_close** (*Object script*)

Emitted when editor is about to close the active script. Argument is a *Script* that is going to be closed.

25.347.4 Member Function Description

- *bool can_drop_data_fw* (*Vector2 point*, *Variant data*, *Control from*) const
- *void drop_data_fw* (*Vector2 point*, *Variant data*, *Control from*)
- *Script get_current_script* ()

Returns a *Script* that is currently active in editor.

- *Variant get_drag_data_fw* (*Vector2 point*, *Control from*)
- *Array get_open_scripts* () const

Returns an array with all *Script* objects which are currently open in editor.

- *void open_script_create_dialog* (*String base_name*, *String base_path*)

25.348 ScrollBar

Inherits: *Range < Control < CanvasItem < Node < Object*

Inherited By: *HScrollBar*, *VScrollBar*

Category: Core

25.348.1 Brief Description

Base class for scroll bars.

25.348.2 Signals

- **scrolling** ()

25.348.3 Member Variables

- *float custom_step*

25.348.4 Description

Scrollbars are a *Range* based *Control*, that display a draggable area (the size of the page). Horizontal (*HScrollBar*) and Vertical (*VScrollBar*) versions are available.

25.349 ScrollContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Category: Core

25.349.1 Brief Description

A helper node for displaying scrollable elements (e.g. lists).

25.349.2 Member Variables

- *int scroll_horizontal* - The current horizontal scroll value.
- *bool scroll_horizontal_enabled* - If `true`, enables horizontal scrolling.
- *int scroll_vertical* - The current vertical scroll value.
- *bool scroll_vertical_enabled* - If `true`, enables vertical scrolling.

25.349.3 Description

A ScrollContainer node with a *Control* child and scrollbar child (HScrollbar, VScrollbar, or both) will only draw the Control within the ScrollContainer area. Scrollbars will automatically be drawn at the right (for vertical) or bottom (for horizontal) and will enable dragging to move the viewable Control (and its children) within the ScrollContainer. Scrollbars will also automatically resize the grabber based on the `minimum_size` of the Control relative to the ScrollContainer. Works great with a *Panel* control. You can set EXPAND on children size flags, so they will upscale to ScrollContainer size if ScrollContainer size is bigger (scroll is invisible for chosen dimension).

25.350 SegmentShape2D

Inherits: *Shape2D < Resource < Reference < Object*

Category: Core

25.350.1 Brief Description

Segment shape for 2D collisions.

25.350.2 Member Variables

- *Vector2 a* - The segment's first point position.
- *Vector2 b* - The segment's second point position.

25.350.3 Description

Segment shape for 2D collisions. Consists of two points, a and b.

25.351 Semaphore

Inherits: *Reference < Object*

Category: Core

25.351.1 Brief Description

A synchronization Semaphore.

25.351.2 Member Functions

<i>int</i>	<i>post ()</i>
<i>int</i>	<i>wait ()</i>

25.351.3 Description

A synchronization Semaphore. Element used to synchronize multiple *Threads*. Initialized to zero on creation. Be careful to avoid deadlocks. For a binary version, see *Mutex*.

25.351.4 Member Function Description

- *int post ()*

Lowers the Semaphore, allowing one more thread in. Returns OK on success, ERR_BUSY otherwise.

- *int wait ()*

Tries to wait for the Semaphore, if its value is zero, blocks until non-zero. Returns OK on success, ERR_BUSY otherwise.

25.352 Separator

Inherits: *Control < CanvasItem < Node < Object*

Inherited By: *HSeparator, VSeparator*

Category: Core

25.352.1 Brief Description

Base class for separators.

25.352.2 Description

Separator is a *Control* used for separating other controls. It's purely a visual decoration. Horizontal (*HSeparator*) and Vertical (*VSeparator*) versions are available.

25.353 Shader

Inherits: *Resource < Reference < Object*

Category: Core

25.353.1 Brief Description

To be changed, ignore.

25.353.2 Member Functions

<i>Texture</i>	<i>get_default_texture_param</i> (<i>String param</i>) const
<i>int</i>	<i>get_mode</i> () const
<i>bool</i>	<i>has_param</i> (<i>String name</i>) const
<i>void</i>	<i>set_default_texture_param</i> (<i>String param, Texture texture</i>)

25.353.3 Member Variables

- *String code*

25.353.4 Enums

enum Mode

- **MODE_SPATIAL = 0**
- **MODE_CANVAS_ITEM = 1**
- **MODE_PARTICLES = 2**

25.353.5 Description

To be changed, ignore.

25.353.6 Member Function Description

- *Texture get_default_texture_param (String param) const*
- *int get_mode () const*
- *bool has_param (String name) const*
- *void set_default_texture_param (String param, Texture texture)*

25.354 ShaderMaterial

Inherits: *Material < Resource < Reference < Object*

Category: Core

25.354.1 Brief Description

25.354.2 Member Functions

<i>Variant</i>	<i>get_shader_param (String param) const</i>
<i>void</i>	<i>set_shader_param (String param, Variant value)</i>

25.354.3 Member Variables

- *Shader shader*

25.354.4 Member Function Description

- *Variant get_shader_param (String param) const*
- *void set_shader_param (String param, Variant value)*

25.355 Shape

Inherits: *Resource < Reference < Object*

Inherited By: *PlaneShape, SphereShape, CapsuleShape, BoxShape, ConvexPolygonShape, RayShape, ConcavePolygonShape*

Category: Core

25.355.1 Brief Description

Base class for all 3D shape resources.

25.355.2 Description

Base class for all 3D shape resources. All 3D shapes that inherit from this can be set into a *PhysicsBody* or *Area*.

25.356 Shape2D

Inherits: Resource < Reference < Object

Inherited By: RayShape2D, CapsuleShape2D, LineShape2D, CircleShape2D, ConcavePolygonShape2D, Convex-PolygonShape2D, RectangleShape2D, SegmentShape2D

Category: Core

25.356.1 Brief Description

Base class for all 2D Shapes.

25.356.2 Member Functions

	<code>bool collide (Transform2D local_xform, Shape2D with_shape, Transform2D shape_xform)</code>
<i>Variant</i>	<code>collide_and_get_contacts (Transform2D local_xform, Shape2D with_shape, Transform2D shape_xform)</code>
	<code>bool collide_with_motion (Transform2D local_xform, Vector2 local_motion, Shape2D with_shape, Transform2D shape_xform, Vector2 shape_motion)</code>
<i>Variant</i>	<code>collide_with_motion_and_get_contacts (Transform2D local_xform, Vector2 local_motion, Shape2D with_shape, Transform2D shape_xform, Vector2 shape_motion)</code>

25.356.3 Member Variables

- `float custom_solver_bias`

25.356.4 Description

Base class for all 2D Shapes. All 2D shape types inherit from this.

25.356.5 Member Function Description

- `bool collide (Transform2D local_xform, Shape2D with_shape, Transform2D shape_xform)`

Return whether this shape is colliding with another.

This method needs the transformation matrix for this shape (`local_xform`), the shape to check collisions with (`with_shape`), and the transformation matrix of that shape (`shape_xform`).

- *Variant* `collide_and_get_contacts (Transform2D local_xform, Shape2D with_shape, Transform2D shape_xform)`

Return a list of the points where this shape touches another. If there are no collisions, the list is empty.

This method needs the transformation matrix for this shape (`local_xform`), the shape to check collisions with (`with_shape`), and the transformation matrix of that shape (`shape_xform`).

- `bool collide_with_motion (Transform2D local_xform, Vector2 local_motion, Shape2D with_shape, Transform2D shape_xform, Vector2 shape_motion)`

Return whether this shape would collide with another, if a given movement was applied.

This method needs the transformation matrix for this shape (`local_xform`), the movement to test on this shape (`local_motion`), the shape to check collisions with (`with_shape`), the transformation matrix of that shape (`shape_xform`), and the movement to test onto the other object (`shape_motion`).

- `Variant collide_with_motion_and_get_contacts (Transform2D local_xform, Vector2 local_motion, Shape2D with_shape, Transform2D shape_xform, Vector2 shape_motion)`

Return a list of the points where this shape would touch another, if a given movement was applied. If there are no collisions, the list is empty.

This method needs the transformation matrix for this shape (`local_xform`), the movement to test on this shape (`local_motion`), the shape to check collisions with (`with_shape`), the transformation matrix of that shape (`shape_xform`), and the movement to test onto the other object (`shape_motion`).

25.357 ShortCut

Inherits: `Resource < Reference < Object`

Category: Core

25.357.1 Brief Description

A shortcut for binding input.

25.357.2 Member Functions

<code>String</code>	<code>get_as_text () const</code>
<code>bool</code>	<code>is_shortcut (InputEvent event) const</code>
<code>bool</code>	<code>is_valid () const</code>

25.357.3 Member Variables

- `InputEvent shortcut` - The Shortcut's `InputEvent`.

Generally the `InputEvent` is a keyboard key, though it can be any `InputEvent`.

25.357.4 Description

A shortcut for binding input.

Shortcuts are commonly used for interacting with a `Control` element from a `InputEvent`.

25.357.5 Member Function Description

- `String get_as_text () const`

Returns the Shortcut's *InputEvent* as a *String*.

- `bool is_shortcut (InputEvent event) const`

Returns `true` if the Shortcut's *InputEvent* equals *event*.

- `bool is_valid () const`

If `true` this Shortcut is valid.

25.358 Skeleton

Inherits: *Spatial* < *Node* < *Object*

Category: Core

25.358.1 Brief Description

Skeleton for characters and animated objects.

25.358.2 Member Functions

<code>void</code>	<code>add_bone (String name)</code>
<code>void</code>	<code>bind_child_node_to_bone (int bone_idx, Node node)</code>
<code>void</code>	<code>clear_bones ()</code>
<code>int</code>	<code>find_bone (String name) const</code>
<code>int</code>	<code>get_bone_count () const</code>
<code>Transform</code>	<code>get_bone_custom_pose (int bone_idx) const</code>
<code>Transform</code>	<code>get_bone_global_pose (int bone_idx) const</code>
<code>String</code>	<code>get_bone_name (int bone_idx) const</code>
<code>int</code>	<code>get_bone_parent (int bone_idx) const</code>
<code>Transform</code>	<code>get_bone_pose (int bone_idx) const</code>
<code>Transform</code>	<code>get_bone_rest (int bone_idx) const</code>
<code>Transform</code>	<code>get_bone_transform (int bone_idx) const</code>
<code>Array</code>	<code>get_bound_child_nodes_to_bone (int bone_idx) const</code>
<code>bool</code>	<code>is_bone_rest_disabled (int bone_idx) const</code>
<code>void</code>	<code>set_bone_custom_pose (int bone_idx, Transform custom_pose)</code>
<code>void</code>	<code>set_bone_disable_rest (int bone_idx, bool disable)</code>
<code>void</code>	<code>set_bone_global_pose (int bone_idx, Transform pose)</code>
<code>void</code>	<code>set_bone_parent (int bone_idx, int parent_idx)</code>
<code>void</code>	<code>set_bone_pose (int bone_idx, Transform pose)</code>
<code>void</code>	<code>set_bone_rest (int bone_idx, Transform rest)</code>
<code>void</code>	<code>unbind_child_node_from_bone (int bone_idx, Node node)</code>
<code>void</code>	<code>unparent_bone_and_rest (int bone_idx)</code>

25.358.3 Numeric Constants

- **NOTIFICATION_UPDATE_SKELETON = 50**

25.358.4 Description

Skeleton provides a hierarchical interface for managing bones, including pose, rest and animation (see *Animation*). Skeleton will support rag doll dynamics in the future.

25.358.5 Member Function Description

- **void add_bone (String name)**

Add a bone, with name “name”. *get_bone_count* will become the bone index.

- **void bind_child_node_to_bone (int bone_idx, Node node)**

Deprecated soon.

- **void clear_bones ()**

Clear all the bones in this skeleton.

- **int find_bone (String name) const**

Return the bone index that matches “name” as its name.

- **int get_bone_count () const**

Return the amount of bones in the skeleton.

- **Transform get_bone_custom_pose (int bone_idx) const**

- **Transform get_bone_global_pose (int bone_idx) const**

- **String get_bone_name (int bone_idx) const**

Return the name of the bone at index “index”

- **int get_bone_parent (int bone_idx) const**

Return the bone index which is the parent of the bone at “bone_idx”. If -1, then bone has no parent. Note that the parent bone returned will always be less than “bone_idx”.

- **Transform get_bone_pose (int bone_idx) const**

Return the pose transform for bone “bone_idx”.

- **Transform get_bone_rest (int bone_idx) const**

Return the rest transform for a bone “bone_idx”.

- **Transform get_bone_transform (int bone_idx) const**

- **Array get_bound_child_nodes_to_bone (int bone_idx) const**

Deprecated soon.

- **bool is_bone_rest_disabled (int bone_idx) const**

- **void set_bone_custom_pose (int bone_idx, Transform custom_pose)**

- **void set_bone_disable_rest (int bone_idx, bool disable)**

- **void set_bone_global_pose (int bone_idx, Transform pose)**

- void **set_bone_parent** (*int bone_idx, int parent_idx*)

Set the bone index “parent_idx” as the parent of the bone at “bone_idx”. If -1, then bone has no parent. Note: “parent_idx” must be less than “bone_idx”.

- void **set_bone_pose** (*int bone_idx, Transform pose*)

Return the pose transform for bone “bone_idx”.

- void **set_bone_rest** (*int bone_idx, Transform rest*)

Set the rest transform for bone “bone_idx”

- void **unbind_child_node_from_bone** (*int bone_idx, Node node*)

Deprecated soon.

- void **unparent_bone_and_rest** (*int bone_idx*)

25.359 Sky

Inherits: Resource < Reference < Object

Inherited By: PanoramaSky, ProceduralSky

Category: Core

25.359.1 Brief Description

The base class for *PanoramaSky* and *ProceduralSky*.

25.359.2 Member Variables

- *RadianceSize* **radiance_size** - The Sky’s radiance map size.

The higher the radiance map size, the more detailed the lighting from the Sky will be.

See RADIANCE_SIZE_* constants for values. Default size is RADIANCE_SIZE_512.

25.359.3 Enums

enum RadianceSize

- **RADIANCE_SIZE_32 = 0** — Radiance texture size is 32x32 pixels.
- **RADIANCE_SIZE_64 = 1** — Radiance texture size is 64x64 pixels.
- **RADIANCE_SIZE_128 = 2** — Radiance texture size is 128x128 pixels.
- **RADIANCE_SIZE_256 = 3** — Radiance texture size is 256x256 pixels.
- **RADIANCE_SIZE_512 = 4** — Radiance texture size is 512x512 pixels.
- **RADIANCE_SIZE_1024 = 5** — Radiance texture size is 1024x1024 pixels.
- **RADIANCE_SIZE_2048 = 6** — Radiance texture size is 2048x2048 pixels.

- **RADIANCE_SIZE_MAX = 7** — Radiance texture size is the largest size it can be.

25.359.4 Description

The base class for *PanoramaSky* and *ProceduralSky*.

25.360 Slider

Inherits: *Range < Control < CanvasItem < Node < Object*

Inherited By: *HSlider, VSlider*

Category: Core

25.360.1 Brief Description

Base class for GUI Sliders.

25.360.2 Member Variables

- *bool editable*
- *FocusMode focus_mode*
- *int tick_count*
- *bool ticks_on_borders*

25.360.3 Description

Base class for GUI Sliders.

25.361 SliderJoint

Inherits: *Joint < Spatial < Node < Object*

Category: Core

25.361.1 Brief Description

Piston kind of slider between two bodies in 3D.

25.361.2 Member Variables

- *float angular_limit/damping* - The amount of damping of the rotation when the limit is surpassed.

A lower damping value allows a rotation initiated by body A to travel to body B slower.

- *float angular_limit/lower_angle* - The lower limit of rotation in the slider.
- *float angular_limit/restitution* - The amount of restitution of the rotation when the limit is surpassed.

Does not affect damping.

- *float angular_limit/softness* - A factor applied to the all rotation once the limit is surpassed.

Makes all rotation slower when between 0 and 1.

- *float angular_limit/upper_angle* - The upper limit of rotation in the slider.
- *float angular_motion/damping* - The amount of damping of the rotation in the limits.
- *float angular_motion/restitution* - The amount of restitution of the rotation in the limits.
- *float angular_motion/softness* - A factor applied to the all rotation in the limits.
- *float angular_ortho/damping* - The amount of damping of the rotation across axes orthogonal to the slider.
- *float angular_ortho/restitution* - The amount of restitution of the rotation across axes orthogonal to the slider.
- *float angular_ortho/softness* - A factor applied to the all rotation across axes orthogonal to the slider.
- *float linear_limit/damping* - The amount of damping that happens once the limit defined by *linear_limit/lower_distance* and *linear_limit/upper_distance* is surpassed.
- *float linear_limit/lower_distance* - The minimum difference between the pivot points on their x-axis before damping happens.
- *float linear_limit/restitution* - The amount of restitution once the limits are surpassed. The lower, the more velocity-energy gets lost.
- *float linear_limit/softness* - A factor applied to the movement across the slider axis once the limits get surpassed. The lower, the slower the movement.
- *float linear_limit/upper_distance* - The maximum difference between the pivot points on their x-axis before damping happens.
- *float linear_motion/damping* - The amount of damping inside the slider limits.
- *float linear_motion/restitution* - The amount of restitution inside the slider limits.
- *float linear_motion/softness* - A factor applied to the movement across the slider axis as long as the slider is in the limits. The lower, the slower the movement.
- *float linear_ortho/damping* - The amount of damping when movement is across axes orthogonal to the slider.
- *float linear_ortho/restitution* - The amount of restitution when movement is across axes orthogonal to the slider.
- *float linear_ortho/softness* - A factor applied to the movement across axes orthogonal to the slider.

25.361.3 Enums

enum Param

- **PARAM_LINEAR_LIMIT_UPPER = 0** — The maximum difference between the pivot points on their x-axis before damping happens.
- **PARAM_LINEAR_LIMIT_LOWER = 1** — The minimum difference between the pivot points on their x-axis before damping happens.
- **PARAM_LINEAR_LIMIT_SOFTNESS = 2** — A factor applied to the movement across the slider axis once the limits get surpassed. The lower, the slower the movement.
- **PARAM_LINEAR_LIMIT_RESTITUTION = 3** — The amount of restitution once the limits are surpassed. The lower, the more velocityenergy gets lost.
- **PARAM_LINEAR_LIMIT_DAMPING = 4** — The amount of damping once the slider limits are surpassed.
- **PARAM_LINEAR_MOTION_SOFTNESS = 5** — A factor applied to the movement across the slider axis as long as the slider is in the limits. The lower, the slower the movement.
- **PARAM_LINEAR_MOTION_RESTITUTION = 6** — The amount of restitution inside the slider limits.
- **PARAM_LINEAR_MOTION_DAMPING = 7** — The amount of damping inside the slider limits.
- **PARAM_LINEAR_ORTHOGONAL_SOFTNESS = 8** — A factor applied to the movement across axes orthogonal to the slider.
- **PARAM_LINEAR_ORTHOGONAL_RESTITUTION = 9** — The amount of restitution when movement is across axes orthogonal to the slider.
- **PARAM_LINEAR_ORTHOGONAL_DAMPING = 10** — The amount of damping when movement is across axes orthogonal to the slider.
- **PARAM_ANGULAR_LIMIT_UPPER = 11** — The upper limit of rotation in the slider.
- **PARAM_ANGULAR_LIMIT_LOWER = 12** — The lower limit of rotation in the slider.
- **PARAM_ANGULAR_LIMIT_SOFTNESS = 13** — A factor applied to the all rotation once the limit is surpassed.
- **PARAM_ANGULAR_LIMIT_RESTITUTION = 14** — The amount of restitution of the rotation when the limit is surpassed.
- **PARAM_ANGULAR_LIMIT_DAMPING = 15** — The amount of damping of the rotation when the limit is surpassed.
- **PARAM_ANGULAR_MOTION_SOFTNESS = 16** — A factor applied to the all rotation in the limits.
- **PARAM_ANGULAR_MOTION_RESTITUTION = 17** — The amount of restitution of the rotation in the limits.
- **PARAM_ANGULAR_MOTION_DAMPING = 18** — The amount of damping of the rotation in the limits.
- **PARAM_ANGULAR_ORTHOGONAL_SOFTNESS = 19** — A factor applied to the all rotation across axes orthogonal to the slider.
- **PARAM_ANGULAR_ORTHOGONAL_RESTITUTION = 20** — The amount of restitution of the rotation across axes orthogonal to the slider.
- **PARAM_ANGULAR_ORTHOGONAL_DAMPING = 21** — The amount of damping of the rotation across axes orthogonal to the slider.
- **PARAM_MAX = 22** — End flag of PARAM_* constants, used internally.

25.361.4 Description

Slides across the x-axis of the Pivot object.

25.362 Spatial

Inherits: *Node < Object*

Inherited By: *Joint, RayCast, Camera, BoneAttachment, CollisionShape, AudioStreamPlayer3D, ARVRController, Path, VisualInstance, VehicleWheel, Position3D, ProximityGroup, ARVRAuthority, RemoteTransform, CollisionObject, PathFollow, NavigationMeshInstance, Listener, VisibilityNotifier, Navigation, CollisionPolygon, GridMap, Skeleton, ARVROrigin*

Category: Core

25.362.1 Brief Description

Most basic 3D game object, parent of all 3D related nodes.

25.362.2 Member Functions

<i>Spatial</i>	<code>get_parent_spatial () const</code>
<i>World</i>	<code>get_world () const</code>
void	<code>global_rotate (Vector3 axis, float angle)</code>
void	<code>global_scale (Vector3 scale)</code>
void	<code>global_translate (Vector3 offset)</code>
void	<code>hide ()</code>
<i>bool</i>	<code>is_local_transform_notification_enabled () const</code>
<i>bool</i>	<code>is_set_as_toplevel () const</code>
<i>bool</i>	<code>is_transform_notification_enabled () const</code>
<i>bool</i>	<code>is_visible_in_tree () const</code>
void	<code>look_at (Vector3 target, Vector3 up)</code>
void	<code>look_at_from_position (Vector3 position, Vector3 target, Vector3 up)</code>
void	<code>orthonormalize ()</code>
void	<code>rotate (Vector3 axis, float angle)</code>
void	<code>rotate_object_local (Vector3 axis, float angle)</code>
void	<code>rotate_x (float angle)</code>
void	<code>rotate_y (float angle)</code>
void	<code>rotate_z (float angle)</code>
void	<code>scale_object_local (Vector3 scale)</code>
void	<code>set_as_toplevel (bool enable)</code>
void	<code>set_identity ()</code>
void	<code>set_ignore_transform_notification (bool enabled)</code>
void	<code>set_notify_local_transform (bool enable)</code>
void	<code>set_notify_transform (bool enable)</code>
void	<code>show ()</code>
<i>Vector3</i>	<code>to_global (Vector3 local_point) const</code>
<i>Vector3</i>	<code>to_local (Vector3 global_point) const</code>
void	<code>translate (Vector3 offset)</code>
void	<code>translate_object_local (Vector3 offset)</code>
void	<code>update_gizmo ()</code>

25.362.3 Signals

- **visibility_changed ()**

Emitted when node visibility changes.

25.362.4 Member Variables

- ***SpatialGizmo gizmo*** - The SpatialGizmo for this node. Used for example in *EditorSpatialGizmo* as custom visualization and editing handles in Editor.
- ***Transform global_transform*** - World space (global) *Transform* of this node.
- ***Vector3 rotation*** - Rotation part of the local transformation, specified in terms of YXZ-Euler angles in the format (X-angle, Y-angle, Z-angle), in radians.

Note that in the mathematical sense, rotation is a matrix and not a vector. The three Euler angles, which are the three independent parameters of the Euler-angle parametrization of the rotation matrix, are stored in a *Vector3* data structure

not because the rotation is a vector, but only because *Vector3* exists as a convenient data-structure to store 3 floating point numbers. Therefore, applying affine operations on the rotation “vector” is not meaningful.

- *Vector3 rotation_degrees* - Rotation part of the local transformation, specified in terms of YXZ-Euler angles in the format (X-angle, Y-angle, Z-angle), in degrees.
- *Vector3 scale* - Scale part of the local transformation.
- *Transform transform* - Local space *Transform* of this node, with respect to the parent node.
- *Vector3 translation* - Local translation of this node.
- *bool visible* - Visibility of this node. Toggles if this node is rendered.

25.362.5 Numeric Constants

- **NOTIFICATION_TRANSFORM_CHANGED = 29** — Spatial nodes receives this notification when their global transform changes. This means that either the current or a parent node changed its transform.

In order for NOTIFICATION_TRANSFORM_CHANGED to work user first needs to ask for it, with `set_notify_transform(true)`. - **NOTIFICATION_ENTER_WORLD = 41** — Spatial nodes receives this notification when they are registered to new *World* resource. - **NOTIFICATION_EXIT_WORLD = 42** — Spatial nodes receives this notification when they are unregistered from current *World* resource. - **NOTIFICATION_VISIBILITY_CHANGED = 43** — Spatial nodes receives this notification when their visibility changes.

25.362.6 Description

Most basic 3D game object, with a 3D *Transform* and visibility settings. All other 3D game objects inherit from *Spatial*. Use *Spatial* as a parent node to move, scale, rotate and show/hide children in a 3D project.

Affine operations (rotate, scale, translate) happen in parent’s local coordinate system, unless the *Spatial* object is set as top level. Affine operations in this coordinate system correspond to direct affine operations on the *Spatial*’s transform. The word local below refers to this coordinate system. The coordinate system that is attached to the *Spatial* object itself is referred to as object-local coordinate system.

25.362.7 Member Function Description

- *Spatial get_parent_spatial () const*

Returns the parent *Spatial*, or an empty *Object* if no parent exists or parent is not of type *Spatial*.

- *World get_world () const*

Returns the current *World* resource this *Spatial* node is registered to.

- *void global_rotate (Vector3 axis, float angle)*

Rotates the global (world) transformation around axis, a unit *Vector3*, by specified angle in radians. The rotation axis is in global coordinate system.

- *void global_scale (Vector3 scale)*
- *void global_translate (Vector3 offset)*

Moves the global (world) transformation by *Vector3* offset. The offset is in global coordinate system.

- *void hide ()*

Disables rendering of this node. Change Spatial Visible property to false.

- `bool is_local_transform_notification_enabled () const`

Returns whether node notifies about its local transformation changes. Spatial will not propagate this by default.

- `bool is_set_as_toplevel () const`

Returns whether this node is set as Toplevel, that is whether it ignores its parent nodes transformations.

- `bool is_transform_notification_enabled () const`

Returns whether the node notifies about its global and local transformation changes. Spatial will not propagate this by default.

- `bool is_visible_in_tree () const`

Returns whether the node is visible, taking into consideration that its parents visibility.

- `void look_at (Vector3 target, Vector3 up)`

Rotates itself to point into direction of target position. Operations take place in global space.

- `void look_at_from_position (Vector3 position, Vector3 target, Vector3 up)`

Moves the node to specified position and then rotates itself to point into direction of target position. Operations take place in global space.

- `void orthonormalize ()`

Resets this node's transformations (like scale, skew and taper) preserving its rotation and translation by performing Gram-Schmidt orthonormalization on this node's Transform3D.

- `void rotate (Vector3 axis, float angle)`

Rotates the local transformation around axis, a unit `Vector3`, by specified angle in radians.

- `void rotate_object_local (Vector3 axis, float angle)`

Rotates the local transformation around axis, a unit `Vector3`, by specified angle in radians. The rotation axis is in object-local coordinate system.

- `void rotate_x (float angle)`

Rotates the local transformation around the X axis by angle in radians

- `void rotate_y (float angle)`

Rotates the local transformation around the Y axis by angle in radians.

- `void rotate_z (float angle)`

Rotates the local transformation around the Z axis by angle in radians.

- `void scale_object_local (Vector3 scale)`

Scales the local transformation by given 3D scale factors in object-local coordinate system.

- `void set_as_toplevel (bool enable)`

Makes the node ignore its parents transformations. Node transformations are only in global space.

- `void set_identity ()`

Reset all transformations for this node. Set its Transform3D to identity matrix.

- `void set_ignore_transform_notification (bool enabled)`

Set whether the node ignores notification that its transformation (global or local) changed.

- void **set_notify_local_transform** (*bool* enable)

Set whether the node notifies about its local transformation changes. Spatial will not propagate this by default.

- void **set_notify_transform** (*bool* enable)

Set whether the node notifies about its global and local transformation changes. Spatial will not propagate this by default.

- void **show** ()

Enables rendering of this node. Change Spatial Visible property to “True”.

- *Vector3* **to_global** (*Vector3* local_point) const

Transforms *Vector3* “local_point” from this node’s local space to world space.

- *Vector3* **to_local** (*Vector3* global_point) const

Transforms *Vector3* “global_point” from world space to this node’s local space.

- void **translate** (*Vector3* offset)

Changes the node’s position by given offset *Vector3*.

- void **translate_object_local** (*Vector3* offset)

- void **update_gizmo** ()

Updates the *SpatialGizmo* of this node.

25.363 SpatialGizmo

Inherits: *Reference* < *Object*

Inherited By: *EditorSpatialGizmo*

Category: Core

25.363.1 Brief Description

25.364 SpatialMaterial

Inherits: *Material* < *Resource* < *Reference* < *Object*

Category: Core

25.364.1 Brief Description

25.364.2 Member Variables

- *Color* **albedo_color**
- *Texture* **albedo_texture**
- *float* **anisotropy**
- *bool* **anisotropy_enabled**

- *Texture* **anisotropy_flowmap**
- *bool* **ao_enabled**
- *float* **ao_light_affect**
- *bool* **ao_on_uv2**
- *Texture* **ao_texture**
- *TextureChannel* **ao_texture_channel**
- *float* **clearcoat**
- *bool* **clearcoat_enabled**
- *float* **clearcoat_gloss**
- *Texture* **clearcoat_texture**
- *bool* **depth_deep_parallax**
- *bool* **depth_enabled**
- *int* **depth_max_layers**
- *int* **depth_min_layers**
- *float* **depth_scale**
- *Texture* **depth_texture**
- *Texture* **detail_albedo**
- *BlendMode* **detail_blend_mode**
- *bool* **detail_enabled**
- *Texture* **detail_mask**
- *Texture* **detail_normal**
- *DetailUV* **detail_uv_layer**
- *bool* **distance_fade_enable**
- *float* **distance_fade_max_distance**
- *float* **distance_fade_min_distance**
- *Color* **emission**
- *bool* **emission_enabled**
- *float* **emission_energy**
- *bool* **emission_on_uv2**
- *EmissionOperator* **emission_operator**
- *Texture* **emission_texture**
- *bool* **flags_albedo_tex_force_srgb**
- *bool* **flags_fixed_size**
- *bool* **flags_no_depth_test**
- *bool* **flags_transparent**
- *bool* **flags_unshaded**

- *bool* **flags_use_point_size**
- *bool* **flags_vertex_lighting**
- *bool* **flags_world_triplanar**
- *float* **metallic**
- *float* **metallic_specular**
- *Texture* **metallic_texture**
- *TextureChannel* **metallic_texture_channel**
- *bool* **normal_enabled**
- *float* **normal_scale**
- *Texture* **normal_texture**
- *float* **params_alpha_scissor_threshold**
- *BillboardMode* **params_billboard_mode**
- *BlendMode* **params_blend_mode**
- *CullMode* **params_cull_mode**
- *DepthDrawMode* **params_depth_draw_mode**
- *DiffuseMode* **params_diffuse_mode**
- *bool* **params_grow**
- *float* **params_grow_amount**
- *float* **params_line_width**
- *float* **params_point_size**
- *SpecularMode* **params_specular_mode**
- *bool* **params_use_alpha_scissor**
- *int* **particles_anim_h_frames**
- *int* **particles_anim_loop**
- *int* **particles_anim_v_frames**
- *float* **proximity_fade_distance**
- *bool* **proximity_fade_enable**
- *bool* **refraction_enabled**
- *float* **refraction_scale**
- *Texture* **refraction_texture**
- *TextureChannel* **refraction_texture_channel**
- *float* **rim**
- *bool* **rim_enabled**
- *Texture* **rim_texture**
- *float* **rim_tint**
- *float* **roughness**

- *Texture* roughness_texture
- *TextureChannel* roughness_texture_channel
- *bool* subsurf_scatter_enabled
- *float* subsurf_scatter_strength
- *Texture* subsurf_scatter_texture
- *Color* transmission
- *bool* transmission_enabled
- *Texture* transmission_texture
- *Vector3* uv1_offset
- *Vector3* uv1_scale
- *bool* uv1_triplanar
- *float* uv1_triplanar_sharpness
- *Vector3* uv2_offset
- *Vector3* uv2_scale
- *bool* uv2_triplanar
- *float* uv2_triplanar_sharpness
- *bool* vertex_color_is_srgb
- *bool* vertex_color_use_as_albedo

25.364.3 Enums

enum **DetailUV**

- DETAIL_UV_1 = 0
- DETAIL_UV_2 = 1

enum **TextureParam**

- TEXTURE_ALBEDO = 0
- TEXTURE_METALLIC = 1
- TEXTURE_ROUGHNESS = 2
- TEXTURE_EMISSION = 3
- TEXTURE_NORMAL = 4
- TEXTURE_RIM = 5
- TEXTURE_CLEARCOAT = 6
- TEXTURE_FLOWMAP = 7
- TEXTURE_AMBIENT_OCCLUSION = 8
- TEXTURE_DEPTH = 9
- TEXTURE_SUBSURFACE_SCATTERING = 10

- **TEXTURE_TRANSMISSION** = 11
- **TEXTURE_REFRACTION** = 12
- **TEXTURE_DETAIL_MASK** = 13
- **TEXTURE_DETAIL_ALBEDO** = 14
- **TEXTURE_DETAIL_NORMAL** = 15
- **TEXTURE_MAX** = 16

enum **DepthDrawMode**

- **DEPTH_DRAW_OPAQUE_ONLY** = 0
- **DEPTH_DRAW_ALWAYS** = 1
- **DEPTH_DRAW_DISABLED** = 2
- **DEPTH_DRAW_ALPHA_OPAQUE_PREPASS** = 3

enum **DiffuseMode**

- **DIFFUSE_BURLEY** = 0
- **DIFFUSE_LAMBERT** = 1
- **DIFFUSE_LAMBERT_WRAP** = 2
- **DIFFUSE_OREN_NAYAR** = 3
- **DIFFUSE_TOON** = 4

enum **CullMode**

- **CULL_BACK** = 0
- **CULL_FRONT** = 1
- **CULL_DISABLED** = 2

enum **Feature**

- **FEATURE_TRANSPARENT** = 0
- **FEATURE_EMISSION** = 1
- **FEATURE_NORMAL_MAPPING** = 2
- **FEATURE_RIM** = 3
- **FEATURE_CLEARCOAT** = 4
- **FEATURE_ANISOTROPY** = 5
- **FEATURE_AMBIENT_OCCLUSION** = 6
- **FEATURE_DEPTH_MAPPING** = 7
- **FEATURE_SUBSURFACE_SCATTERING** = 8
- **FEATURE_TRANSMISSION** = 9
- **FEATURE_REFRACTION** = 10
- **FEATURE_DETAIL** = 11
- **FEATURE_MAX** = 12

enum **Flags**

- **FLAG_UNSHADED** = 0
- **FLAG_USE_VERTEX_LIGHTING** = 1
- **FLAG_DISABLE_DEPTH_TEST** = 2
- **FLAG_ALBEDO_FROM_VERTEX_COLOR** = 3
- **FLAG_SRGB_VERTEX_COLOR** = 4
- **FLAG_USE_POINT_SIZE** = 5
- **FLAG_FIXED_SIZE** = 6
- **FLAG_UV1_USE_TRIPLANAR** = 7
- **FLAG_UV2_USE_TRIPLANAR** = 8
- **FLAG_AO_ON_UV2** = 10
- **FLAG_EMISSION_ON_UV2** = 11
- **FLAG_USE_ALPHA_SCISSOR** = 12
- **FLAG_TRIPLANAR_USE_WORLD** = 9
- **FLAG_ALBEDO_TEXTURE_FORCE_SRGB** = 13
- **FLAG_MAX** = 14

enum **BlendMode**

- **BLEND_MODE_MIX** = 0
- **BLEND_MODE_ADD** = 1
- **BLEND_MODE_SUB** = 2
- **BLEND_MODE_MUL** = 3

enum **SpecularMode**

- **SPECULAR_SCHLICK_GGX** = 0
- **SPECULAR_BLINN** = 1
- **SPECULAR_PHONG** = 2
- **SPECULAR_TOON** = 3
- **SPECULAR_DISABLED** = 4

enum **TextureChannel**

- **TEXTURE_CHANNEL_RED** = 0
- **TEXTURE_CHANNEL_GREEN** = 1
- **TEXTURE_CHANNEL_BLUE** = 2
- **TEXTURE_CHANNEL_ALPHA** = 3
- **TEXTURE_CHANNEL_GRAYSCALE** = 4

enum **BillboardMode**

- **BILLBOARD_DISABLED** = 0
- **BILLBOARD_ENABLED** = 1
- **BILLBOARD_FIXED_Y** = 2

- **BILLBOARD_PARTICLES** = 3

enum **EmissionOperator**

- **EMISSION_OP_ADD** = 0
- **EMISSION_OP_MULTIPLY** = 1

25.365 SpatialVelocityTracker

Inherits: Reference < Object

Category: Core

25.365.1 Brief Description

25.365.2 Member Functions

<i>Vector3</i>	<i>get_tracked_linear_velocity () const</i>
<i>void</i>	<i>reset (Vector3 position)</i>
<i>void</i>	<i>update_position (Vector3 position)</i>

25.365.3 Member Variables

- *bool track_physics_step*

25.365.4 Member Function Description

- **Vector3 get_tracked_linear_velocity () const**
- **void reset (Vector3 position)**
- **void update_position (Vector3 position)**

25.366 SphereMesh

Inherits: PrimitiveMesh < Mesh < Resource < Reference < Object

Category: Core

25.366.1 Brief Description

Class representing a spherical *PrimitiveMesh*.

25.366.2 Member Variables

- *float height* - Full height of the sphere. Defaults to 2.0.
- *bool is_hemisphere* - Determines whether a full sphere or a hemisphere is created. Attention: To get a regular hemisphere the height and radius of the sphere have to equal. Defaults to false.
- *int radial_segments* - Number of radial segments on the sphere. Defaults to 64.
- *float radius* - Radius of sphere. Defaults to 1.0.
- *int rings* - Number of segments along the height of the sphere. Defaults to 32.

25.366.3 Description

Class representing a spherical *PrimitiveMesh*.

25.367 SphereShape

Inherits: *Shape < Resource < Reference < Object*

Category: Core

25.367.1 Brief Description

Sphere shape for 3D collisions.

25.367.2 Member Variables

- *float radius* - The sphere's radius. The shape's diameter is double the radius.

25.367.3 Description

Sphere shape for 3D collisions, which can be set into a *PhysicsBody* or *Area*. This shape is useful for modeling sphere-like 3D objects.

25.368 SpinBox

Inherits: *Range < Control < CanvasItem < Node < Object*

Category: Core

25.368.1 Brief Description

Numerical input text field.

25.368.2 Member Functions

<i>LineEdit</i>	<code>get_line_edit()</code>
-----------------	------------------------------

25.368.3 Member Variables

- *bool* **editable**
- *String* **prefix**
- *String* **suffix**

25.368.4 Description

SpinBox is a numerical input text field. It allows entering integers and floats.

25.368.5 Member Function Description

- *LineEdit* `get_line_edit()`

25.369 SplitContainer

Inherits: *Container < Control < CanvasItem < Node < Object*

Inherited By: *HSplitContainer, VSplitContainer*

Category: Core

25.369.1 Brief Description

Container for splitting and adjusting.

25.369.2 Signals

- **dragged** (*int offset*)

Emitted when the dragger is dragged by user.

25.369.3 Member Variables

- *bool* **collapsed**
- *DraggerVisibility* **dragger_visibility**
- *int* **split_offset**

25.369.4 Enums

enum DraggerVisibility

- **DRAGGER_VISIBLE = 0** — The split dragger is visible.
- **DRAGGER_HIDDEN = 1** — The split dragger is invisible.
- **DRAGGER_HIDDEN_COLLAPSED = 2** — The split dragger is invisible and collapsed.

25.369.5 Description

Container for splitting two controls vertically or horizontally, with a grabber that allows adjusting the split offset or ratio.

25.370 SpotLight

Inherits: *Light < VisualInstance < Spatial < Node < Object*

Category: Core

25.370.1 Brief Description

Spotlight *Light*, such as a reflector spotlight or a lantern.

25.370.2 Member Variables

- *float spot_angle*
- *float spot_angle_attenuation*
- *float spot_attenuation*
- *float spot_range*

25.370.3 Description

A SpotLight light is a type of *Light* node that emits lights in a specific direction, in the shape of a cone. The light is attenuated through the distance and this attenuation can be configured by changing the energy, radius and attenuation parameters of *Light*. TODO: Image of a spotlight.

25.371 Sprite

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.371.1 Brief Description

General purpose Sprite node.

25.371.2 Signals

- **frame_changed ()**

Emitted when the *frame* changes.

- **texture_changed ()**

Emitted when the *texture* changes.

25.371.3 Member Variables

- *bool centered* - If `true` texture is centered. Default value: `true`.
- *bool flip_h* - If `true` texture is flipped horizontally. Default value: `false`.
- *bool flip_v* - If `true` texture is flipped vertically. Default value: `false`.
- *int frame* - Current frame to display from sprite sheet. *vframes* or *hframes* must be greater than 1.
- *int hframes* - The number of columns in the sprite sheet.
- *Texture normal_map* - The normal map gives depth to the Sprite.
- *Vector2 offset* - The texture's drawing offset.
- *bool region_enabled* - If `true` texture is cut from a larger atlas texture. See *region_rect*. Default value: `false`.
- *bool region_filter_clip* - If `true` the outermost pixels get blurred out.
- *Rect2 region_rect* - The region of the atlas texture to display. *region_enabled* must be `true`.
- *Texture texture* - *Texture* object to draw.
- *int vframes* - The number of rows in the sprite sheet.

25.371.4 Description

A node that displays a 2D texture. The texture displayed can be a region from a larger atlas texture, or a frame from a sprite sheet animation.

25.372 Sprite3D

Inherits: *SpriteBase3D < GeometryInstance < VisualInstance < Spatial < Node < Object*

Category: Core

25.372.1 Brief Description

2D Sprite node in 3D world.

25.372.2 Signals

- **frame_changed ()**

Emitted when the *frame* changes.

25.372.3 Member Variables

- *int frame* - Current frame to display from sprite sheet. *vframes* or *hframes* must be greater than 1.
- *int hframes* - The number of columns in the sprite sheet.
- *bool region_enabled* - If `true` texture will be cut from a larger atlas texture. See *region_rect*. Default value: `false`.
- *Rect2 region_rect* - The region of the atlas texture to display. *region_enabled* must be `true`.
- *Texture texture* - *Texture* object to draw.
- *int vframes* - The number of rows in the sprite sheet.

25.372.4 Description

A node that displays a 2D texture in a 3D environment. The texture displayed can be a region from a larger atlas texture, or a frame from a sprite sheet animation.

25.373 SpriteBase3D

Inherits: *GeometryInstance < VisualInstance < Spatial < Node < Object*

Inherited By: *AnimatedSprite3D, Sprite3D*

Category: Core

25.373.1 Brief Description

2D Sprite node in 3D environment.

25.373.2 Member Functions

<i>Rect2</i>	<i>get_item_rect () const</i>
--------------	-------------------------------

25.373.3 Member Variables

- *AlphaCutMode alpha_cut*
- Axis **axis** - The direction in which the front of the texture faces.
- *bool centered* - If `true` texture will be centered. Default value: `true`.

- *bool double_sided* - If `true` texture can be seen from the back as well, if `false`, it is invisible when looking at it from behind. Default value: `true`.
- *bool flip_h* - If `true` texture is flipped horizontally. Default value: `false`.
- *bool flip_v* - If `true` texture is flipped vertically. Default value: `false`.
- *Color modulate* - A color value that gets multiplied on, could be used for mood-coloring or to simulate the color of light.
- *Vector2 offset* - The texture's drawing offset.
- *float opacity* - The object's visibility on a scale from 0 fully invisible to 1 fully visible.
- *float pixel_size* - The size of one pixel's width on the Sprite to scale it in 3D.
- *bool shaded* - If `true` the *Light* in the *Environment* has effects on the Sprite. Default value: `false`.
- *bool transparent* - If `true` the texture's transparency and the opacity are used to make those parts of the Sprite invisible. Default value: `true`.

25.373.4 Enums

enum **AlphaCutMode**

- **ALPHA_CUT_DISABLED = 0**
- **ALPHA_CUT_DISCARD = 1**
- **ALPHA_CUT_OPAQUE_PREPASS = 2**

enum **DrawFlags**

- **FLAG_TRANSPARENT = 0** — If set, the texture's transparency and the opacity are used to make those parts of the Sprite invisible.
- **FLAG_SHADED = 1** — If set, the Light in the Environment has effects on the Sprite.
- **FLAG_DOUBLE_SIDED = 2** — If set, texture can be seen from the back as well, if not, it is invisible when looking at it from behind.
- **FLAG_MAX = 3** — Used internally to mark the end of the Flags section.

25.373.5 Description

A node that displays 2D texture information in a 3D environment.

25.373.6 Member Function Description

- *Rect2 get_item_rect () const*

25.374 SpriteFrames

Inherits: Resource < Reference < Object

Category: Core

25.374.1 Brief Description

Sprite frame library for *AnimatedSprite*.

25.374.2 Member Functions

void	<i>add_animation (String anim)</i>
void	<i>add_frame (String anim, Texture frame, int at_position=-1)</i>
void	<i>clear (String anim)</i>
void	<i>clear_all ()</i>
bool	<i>get_animation_loop (String anim) const</i>
float	<i>get_animation_speed (String anim) const</i>
Texture	<i>get_frame (String anim, int idx) const</i>
int	<i>get_frame_count (String anim) const</i>
bool	<i>has_animation (String anim) const</i>
void	<i>remove_animation (String anim)</i>
void	<i>remove_frame (String anim, int idx)</i>
void	<i>rename_animation (String anim, String newname)</i>
void	<i>set_animation_loop (String anim, bool loop)</i>
void	<i>set_animation_speed (String anim, float speed)</i>
void	<i>set_frame (String anim, int idx, Texture txt)</i>

25.374.3 Member Variables

- *Array frames*

25.374.4 Description

Sprite frame library for *AnimatedSprite*. Contains frames and animation data for playback.

25.374.5 Member Function Description

- void **add_animation** (*String anim*)

Adds a new animation to the the library.

- void **add_frame** (*String anim, Texture frame, int at_position=-1*)

Adds a frame to the given animation.

- void **clear** (*String anim*)

Removes all frames from the given animation.

- void **clear_all** ()

Removes all animations. A “default” animation will be created.

- bool **get_animation_loop** (*String anim*) const

If true the given animation will loop.

- `float get_animation_speed (String anim) const`

The animation's speed in frames per second.

- `Texture get_frame (String anim, int idx) const`

Returns the animation's selected frame.

- `int get_frame_count (String anim) const`

Returns the number of frames in the animation.

- `bool has_animation (String anim) const`

If true the named animation exists.

- `void remove_animation (String anim)`

Removes the given animation.

- `void remove_frame (String anim, int idx)`

Removes the animation's selected frame.

- `void rename_animation (String anim, String newname)`

Changes the animation's name to newname.

- `void set_animation_loop (String anim, bool loop)`

If true the animation will loop.

- `void set_animation_speed (String anim, float speed)`

The animation's speed in frames per second.

- `void set_frame (String anim, int idx, Texture txt)`

Sets the texture of the given frame.

25.375 StaticBody

Inherits: `PhysicsBody < CollisionObject < Spatial < Node < Object`

Category: Core

25.375.1 Brief Description

Static body for 3D Physics.

25.375.2 Member Variables

- `float bounce` - The body bounciness.
- `Vector3 constant_angular_velocity` - The constant angular velocity for the body. This does not rotate the body, but affects other bodies that touch it, as if it was in a state of rotation.
- `Vector3 constant_linear_velocity` - The constant linear velocity for the body. This does not move the body, but affects other bodies that touch it, as if it was in a state of movement.

- *float friction* - The body friction, from 0 (frictionless) to 1 (full friction).

25.375.3 Description

Static body for 3D Physics. A static body is a simple body that is not intended to move. They don't consume any CPU resources in contrast to a RigidBody3D so they are great for scenario collision.

A static body can also be animated by using simulated motion mode. This is useful for implementing functionalities such as moving platforms. When this mode is active the body can be animated and automatically computes linear and angular velocity to apply in that frame and to influence other bodies.

Alternatively, a constant linear or angular velocity can be set for the static body, so even if it doesn't move, it affects other bodies as if it was moving (this is useful for simulating conveyor belts or conveyor wheels).

25.376 StaticBody2D

Inherits: *PhysicsBody2D < CollisionObject2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.376.1 Brief Description

Static body for 2D Physics.

25.376.2 Member Variables

- *float bounce* - The body's bounciness. Values range from 0 (no bounce) to 1 (full bounciness).
- *float constant_angular_velocity* - Constant angular velocity for the body. This does not rotate the body, but affects colliding bodies, as if it were rotating.
- *Vector2 constant_linear_velocity* - Constant linear velocity for the body. This does not move the body, but affects colliding bodies, as if it were moving.
- *float friction* - The body's friction. Values range from 0 (no friction) to 1 (full friction).

25.376.3 Description

Static body for 2D Physics. A StaticBody2D is a body that is not intended to move. It is ideal for implementing objects in the environment, such as walls or platforms.

Additionally, a constant linear or angular velocity can be set for the static body, which will affect colliding bodies as if it were moving (for example, a conveyor belt).

25.377 StreamPeer

Inherits: *Reference < Object*

Inherited By: *StreamPeerBuffer, StreamPeerSSL, StreamPeerTCP*

Category: Core

25.377.1 Brief Description

Abstraction and base class for stream-based protocols.

25.377.2 Member Functions

<i>int</i>	<code>get_16()</code>
<i>int</i>	<code>get_32()</code>
<i>int</i>	<code>get_64()</code>
<i>int</i>	<code>get_8()</code>
<i>int</i>	<code>get_available_bytes() const</code>
<i>Array</i>	<code>get_data(int bytes)</code>
<i>float</i>	<code>get_double()</code>
<i>float</i>	<code>get_float()</code>
<i>Array</i>	<code>get_partial_data(int bytes)</code>
<i>String</i>	<code>get_string(int bytes)</code>
<i>int</i>	<code>get_u16()</code>
<i>int</i>	<code>get_u32()</code>
<i>int</i>	<code>get_u64()</code>
<i>int</i>	<code>get_u8()</code>
<i>String</i>	<code>get_utf8_string(int bytes)</code>
<i>Variant</i>	<code>get_var()</code>
<i>void</i>	<code>put_16(int value)</code>
<i>void</i>	<code>put_32(int value)</code>
<i>void</i>	<code>put_64(int value)</code>
<i>void</i>	<code>put_8(int value)</code>
<i>int</i>	<code>put_data(PoolByteArray data)</code>
<i>void</i>	<code>put_double(float value)</code>
<i>void</i>	<code>put_float(float value)</code>
<i>Array</i>	<code>put_partial_data(PoolByteArray data)</code>
<i>void</i>	<code>put_u16(int value)</code>
<i>void</i>	<code>put_u32(int value)</code>
<i>void</i>	<code>put_u64(int value)</code>
<i>void</i>	<code>put_u8(int value)</code>
<i>void</i>	<code>put_utf8_string(String value)</code>
<i>void</i>	<code>put_var(Variant value)</code>

25.377.3 Member Variables

- *bool big_endian* - If `true`, this `StreamPeer` will use big-endian format for encoding and decoding.

25.377.4 Description

`StreamPeer` is an abstraction and base class for stream-based protocols (such as TCP or Unix Sockets). It provides an API for sending and receiving data through streams as raw data or strings.

25.377.5 Member Function Description

- *int get_16()*

Get a signed 16 bit value from the stream.

- *int get_32()*

Get a signed 32 bit value from the stream.

- *int get_64()*

Get a signed 64 bit value from the stream.

- *int get_8()*

Get a signed byte from the stream.

- *int get_available_bytes() const*

Return the amount of bytes this StreamPeer has available.

- *Array get_data(int bytes)*

Return a chunk data with the received bytes. The amount of bytes to be received can be requested in the “bytes” argument. If not enough bytes are available, the function will block until the desired amount is received. This function returns two values, an Error code and a data array.

- *float get_double()*

Get a double-precision float from the stream.

- *float get_float()*

Get a single-precision float from the stream.

- *Array get_partial_data(int bytes)*

Return a chunk data with the received bytes. The amount of bytes to be received can be requested in the “bytes” argument. If not enough bytes are available, the function will return how many were actually received. This function returns two values, an Error code, and a data array.

- *String get_string(int bytes)*

Get a string with byte-length “bytes” from the stream.

- *int get_u16()*

Get an unsigned 16 bit value from the stream.

- *int get_u32()*

Get an unsigned 32 bit value from the stream.

- *int get_u64()*

Get an unsigned 64 bit value from the stream.

- *int get_u8()*

Get an unsigned byte from the stream.

- *String get_utf8_string(int bytes)*

Get a utf8 string with byte-length “bytes” from the stream (this decodes the string sent as utf8).

- *Variant get_var()*

Get a Variant from the stream.

- `void put_16 (int value)`

Put a signed 16 bit value into the stream.

- `void put_32 (int value)`

Put a signed 32 bit value into the stream.

- `void put_64 (int value)`

Put a signed 64 bit value into the stream.

- `void put_8 (int value)`

Put a signed byte into the stream.

- `int put_data (PoolByteArray data)`

Send a chunk of data through the connection, blocking if necessary until the data is done sending. This function returns an Error code.

- `void put_double (float value)`

Put a double-precision float into the stream.

- `void put_float (float value)`

Put a single-precision float into the stream.

- `Array put_partial_data (PoolByteArray data)`

Send a chunk of data through the connection, if all the data could not be sent at once, only part of it will. This function returns two values, an Error code and an integer, describing how much data was actually sent.

- `void put_u16 (int value)`

Put an unsigned 16 bit value into the stream.

- `void put_u32 (int value)`

Put an unsigned 32 bit value into the stream.

- `void put_u64 (int value)`

Put an unsigned 64 bit value into the stream.

- `void put_u8 (int value)`

Put an unsigned byte into the stream.

- `void put_utf8_string (String value)`

Put a zero-terminated utf8 string into the stream.

- `void put_var (Variant value)`

Put a Variant into the stream.

25.378 StreamPeerBuffer

Inherits: *StreamPeer < Reference < Object*

Category: Core

25.378.1 Brief Description

25.378.2 Member Functions

void	<i>clear ()</i>
<i>StreamPeerBuffer</i>	<i>duplicate () const</i>
<i>int</i>	<i>get_position () const</i>
<i>int</i>	<i>get_size () const</i>
void	<i>resize (int size)</i>
void	<i>seek (int position)</i>

25.378.3 Member Variables

- *PoolByteArray data_array*

25.378.4 Member Function Description

- void **clear ()**
- *StreamPeerBuffer* **duplicate () const**
- *int* **get_position () const**
- *int* **get_size () const**
- void **resize (int size)**
- void **seek (int position)**

25.379 StreamPeerSSL

Inherits: *StreamPeer < Reference < Object*

Category: Core

25.379.1 Brief Description

SSL Stream peer.

25.379.2 Member Functions

<i>int</i>	<i>accept_stream (StreamPeer stream)</i>
<i>int</i>	<i>connect_to_stream (StreamPeer stream, bool validate_certs=false, String for_hostname="")</i>
<i>void</i>	<i>disconnect_from_stream ()</i>
<i>int</i>	<i>get_status () const</i>

25.379.3 Enums

enum **Status**

- **STATUS_DISCONNECTED = 0** — A status representing a StreamPeerSSL that is disconnected.
- **STATUS_CONNECTED = 1** — A status representing a StreamPeerSSL that is connected to a host.
- **STATUS_ERROR_NO_CERTIFICATE = 2** — An error status that shows the peer did not present a SSL certificate and validation was requested.
- **STATUS_ERROR_HOSTNAME_MISMATCH = 3** — An error status that shows a mismatch in the SSL certificate domain presented by the host and the domain requested for validation.

25.379.4 Description

SSL Stream peer. This object can be used to connect to SSL servers.

25.379.5 Member Function Description

- *int accept_stream (StreamPeer stream)*
- *int connect_to_stream (StreamPeer stream, bool validate_certs=false, String for_hostname="")*

Connect to a peer using an underlying *StreamPeer* “stream”, when “validate_certs” is true, *StreamPeerSSL* will validate that the certificate presented by the peer matches the “for_hostname”.

- *void disconnect_from_stream ()*

Disconnect from host.

- *int get_status () const*

Return the status of the connection, one of STATUS_* enum.

25.380 StreamPeerTCP

Inherits: StreamPeer < Reference < Object

Category: Core

25.380.1 Brief Description

TCP Stream peer.

25.380.2 Member Functions

<i>int</i>	<code>connect_to_host (String host, int port)</code>
<i>void</i>	<code>disconnect_from_host ()</code>
<i>String</i>	<code>get_connected_host () const</code>
<i>int</i>	<code>get_connected_port () const</code>
<i>int</i>	<code>get_status () const</code>
<i>bool</i>	<code>is_connected_to_host () const</code>

25.380.3 Enums

enum Status

- **STATUS_NONE = 0** — The initial status of the StreamPeerTCP, also the status after a disconnect.
- **STATUS_CONNECTING = 1** — A status representing a StreamPeerTCP that is connecting to a host.
- **STATUS_CONNECTED = 2** — A status representing a StreamPeerTCP that is connected to a host.
- **STATUS_ERROR = 3** — A status representing a StreamPeerTCP in error state.

25.380.4 Description

TCP Stream peer. This object can be used to connect to TCP servers, or also is returned by a tcp server.

25.380.5 Member Function Description

- *int* **connect_to_host (String host, int port)**

Connect to the specified host:port pair. A hostname will be resolved if valid. Returns OK on success or FAILED on failure.

- *void* **disconnect_from_host ()**

Disconnect from host.

- *String* **get_connected_host () const**

Return the IP of this peer.

- *int* **get_connected_port () const**

Return the port of this peer.

- *int* **get_status () const**

Return the status of the connection, one of STATUS_* enum.

- *bool* **is_connected_to_host () const**

25.381 StreamTexture

Inherits: *Texture < Resource < Reference < Object*

Category: Core

25.381.1 Brief Description

A .stex texture.

25.381.2 Member Variables

- *String load_path* - The StreamTexture's filepath to a .stex file.

25.381.3 Description

A texture that is loaded from a .stex file.

25.382 String

Category: Built-In Types

25.382.1 Brief Description

Built-in string class.

25.382.2 Member Functions

<i>String</i>	<i>String (bool from)</i>
<i>String</i>	<i>String (int from)</i>
<i>String</i>	<i>String (float from)</i>
<i>String</i>	<i>String (Vector2 from)</i>
<i>String</i>	<i>String (Rect2 from)</i>
<i>String</i>	<i>String (Vector3 from)</i>
<i>String</i>	<i>String (Transform2D from)</i>
<i>String</i>	<i>String (Plane from)</i>
<i>String</i>	<i>String (Quat from)</i>
<i>String</i>	<i>String (AABB from)</i>
<i>String</i>	<i>String (Basis from)</i>
<i>String</i>	<i>String (Transform from)</i>
<i>String</i>	<i>String (Color from)</i>
<i>String</i>	<i>String (NodePath from)</i>
<i>String</i>	<i>String (RID from)</i>
<i>String</i>	<i>String (Dictionary from)</i>
<i>String</i>	<i>String (Array from)</i>
<i>String</i>	<i>String (PoolByteArray from)</i>
<i>String</i>	<i>String (PoolIntArray from)</i>
<i>String</i>	<i>String (PoolRealArray from)</i>
<i>String</i>	<i>String (PoolStringArray from)</i>
<i>String</i>	<i>String (PoolVector2Array from)</i>
<i>String</i>	<i>String (PoolVector3Array from)</i>

Continued on next page

Table 25.20 – continued from previous page

<i>String</i>	<i>String</i> (<i>PoolColorArray</i> from)
<i>bool</i>	<i>begins_with</i> (<i>String</i> <i>text</i>)
<i>PoolStringArray</i>	<i>bigrams</i> ()
<i>String</i>	<i>c_escape</i> ()
<i>String</i>	<i>c_unescape</i> ()
<i>String</i>	<i>capitalize</i> ()
<i>int</i>	<i>casecmp_to</i> (<i>String</i> <i>to</i>)
<i>String</i>	<i>dedent</i> ()
<i>bool</i>	<i>empty</i> ()
<i>bool</i>	<i>ends_with</i> (<i>String</i> <i>text</i>)
<i>void</i>	<i>erase</i> (<i>int</i> <i>position</i> , <i>int</i> <i>chars</i>)
<i>int</i>	<i>find</i> (<i>String</i> <i>what</i> , <i>int</i> <i>from=0</i>)
<i>int</i>	<i>find_last</i> (<i>String</i> <i>what</i>)
<i>int</i>	<i>findn</i> (<i>String</i> <i>what</i> , <i>int</i> <i>from=0</i>)
<i>String</i>	<i>format</i> (<i>var</i> <i>values</i> , <i>String</i> <i>placeholder={_}</i>)
<i>String</i>	<i>get_base_dir</i> ()
<i>String</i>	<i>get_basename</i> ()
<i>String</i>	<i>get_extension</i> ()
<i>String</i>	<i>get_file</i> ()
<i>int</i>	<i>hash</i> ()
<i>int</i>	<i>hex_to_int</i> ()
<i>String</i>	<i>insert</i> (<i>int</i> <i>position</i> , <i>String</i> <i>what</i>)
<i>bool</i>	<i>is_abs_path</i> ()
<i>bool</i>	<i>is_rel_path</i> ()
<i>bool</i>	<i>is_subsequence_of</i> (<i>String</i> <i>text</i>)
<i>bool</i>	<i>is_subsequence_ofi</i> (<i>String</i> <i>text</i>)
<i>bool</i>	<i>is_valid_float</i> ()
<i>bool</i>	<i>is_valid_html_color</i> ()
<i>bool</i>	<i>is_valid_identifier</i> ()
<i>bool</i>	<i>is_valid_integer</i> ()
<i>bool</i>	<i>is_valid_ip_address</i> ()
<i>String</i>	<i>json_escape</i> ()
<i>String</i>	<i>left</i> (<i>int</i> <i>position</i>)
<i>int</i>	<i>length</i> ()
<i>bool</i>	<i>match</i> (<i>String</i> <i>expr</i>)
<i>bool</i>	<i>matchn</i> (<i>String</i> <i>expr</i>)
<i>PoolByteArray</i>	<i>md5_buffer</i> ()
<i>String</i>	<i>md5_text</i> ()
<i>int</i>	<i>nocasecmp_to</i> (<i>String</i> <i>to</i>)
<i>int</i>	<i>ord_at</i> (<i>int</i> <i>at</i>)
<i>String</i>	<i>pad_decimals</i> (<i>int</i> <i>digits</i>)
<i>String</i>	<i>pad_zeros</i> (<i>int</i> <i>digits</i>)
<i>String</i>	<i>percent_decode</i> ()
<i>String</i>	<i>percent_encode</i> ()
<i>String</i>	<i>plus_file</i> (<i>String</i> <i>file</i>)
<i>String</i>	<i>replace</i> (<i>String</i> <i>what</i> , <i>String</i> <i>forwhat</i>)
<i>String</i>	<i>replacen</i> (<i>String</i> <i>what</i> , <i>String</i> <i>forwhat</i>)
<i>int</i>	<i>rfind</i> (<i>String</i> <i>what</i> , <i>int</i> <i>from=-1</i>)
<i>int</i>	<i>rfindn</i> (<i>String</i> <i>what</i> , <i>int</i> <i>from=-1</i>)
<i>String</i>	<i>right</i> (<i>int</i> <i>position</i>)

Continued on next page

Table 25.20 – continued from previous page

<i>PoolByteArray</i>	<i>sha256_buffer ()</i>
<i>String</i>	<i>sha256_text ()</i>
<i>float</i>	<i>similarity (String text)</i>
<i>PoolStringArray</i>	<i>split (String divisor, bool allow_empty=True, int maxsplit=0)</i>
<i>PoolRealArray</i>	<i>split_floats (String divisor, bool allow_empty=True)</i>
<i>String</i>	<i>strip_edges (bool left=True, bool right=True)</i>
<i>String</i>	<i>substr (int from, int len)</i>
<i>PoolByteArray</i>	<i>to_ascii ()</i>
<i>float</i>	<i>to_float ()</i>
<i>int</i>	<i>to_int ()</i>
<i>String</i>	<i>to_lower ()</i>
<i>String</i>	<i>to_upper ()</i>
<i>PoolByteArray</i>	<i>to_utf8 ()</i>
<i>String</i>	<i>xml_escape ()</i>
<i>String</i>	<i>xml_unescape ()</i>

25.382.3 Description

This is the built-in string class (and the one used by GDScript). It supports Unicode and provides all necessary means for string handling. Strings are reference counted and use a copy-on-write approach, so passing them around is cheap in resources.

25.382.4 Member Function Description

- *String String (bool from)*

Constructs a new String from the given *bool*.

- *String String (int from)*

Constructs a new String from the given *int*.

- *String String (float from)*

Constructs a new String from the given *float*.

- *String String (Vector2 from)*

Constructs a new String from the given *Vector2*.

- *String String (Rect2 from)*

Constructs a new String from the given *Rect2*.

- *String String (Vector3 from)*

Constructs a new String from the given *Vector3*.

- *String String (Transform2D from)*

Constructs a new String from the given *Transform2D*.

- *String String (Plane from)*

Constructs a new String from the given *Plane*.

- *String String (Quat from)*

Constructs a new String from the given *Quat*.

- **String String (*AABB* from)**

Constructs a new String from the given *AABB*.

- **String String (*Basis* from)**

Constructs a new String from the given *Basis*.

- **String String (*Transform* from)**

Constructs a new String from the given *Transform*.

- **String String (*Color* from)**

Constructs a new String from the given *Color*.

- **String String (*NodePath* from)**

Constructs a new String from the given *NodePath*.

- **String String (*RID* from)**

Constructs a new String from the given *RID*.

- **String String (*Dictionary* from)**

Constructs a new String from the given *Dictionary*.

- **String String (*Array* from)**

Constructs a new String from the given *Array*.

- **String String (*PoolByteArray* from)**

Constructs a new String from the given *PoolByteArray*.

- **String String (*PoolIntArray* from)**

Constructs a new String from the given *PoolIntArray*.

- **String String (*PoolRealArray* from)**

Constructs a new String from the given *PoolRealArray*.

- **String String (*PoolStringArray* from)**

Constructs a new String from the given *PoolStringArray*.

- **String String (*PoolVector2Array* from)**

Constructs a new String from the given *PoolVector2Array*.

- **String String (*PoolVector3Array* from)**

Constructs a new String from the given *PoolVector3Array*.

- **String String (*PoolColorArray* from)**

Constructs a new String from the given *PoolColorArray*.

- **bool begins_with (*String* text)**

Returns `true` if the string begins with the given string.

- **PoolStringArray bigrams ()**

Returns the bigrams (pairs of consecutive letters) of this string.

- **String c_escape ()**

Returns a copy of the string with special characters escaped using the C language standard.

- *String **c_unescape** ()*

Returns a copy of the string with escaped characters replaced by their meanings according to the C language standard.

- *String **capitalize** ()*

Changes the case of some letters. Replaces underscores with spaces, converts all letters to lower-case, then capitalizes first and every letter following the space character. For capitalize camelCase mixed_with_underscores it will return Capitalize Camelcase Mixed With Underscores.

- *int **casecmp_to** (String to)*

Performs a case-sensitive comparison to another string. Returns -1 if less than, +1 if greater than, or 0 if equal.

- *String **dedent** ()*

Removes indentation from string.

- *bool **empty** ()*

Returns true if the string is empty.

- *bool **ends_with** (String text)*

Returns true if the string ends with the given string.

- *void **erase** (int position, int chars)*

Erases chars characters from the string starting from position.

- *int **find** (String what, int from=0)*

Finds the first occurrence of a substring. Returns the starting position of the substring or -1 if not found. Optionally, the initial search index can be passed.

- *int **find_last** (String what)*

Finds the last occurrence of a substring. Returns the starting position of the substring or -1 if not found.

- *int **findn** (String what, int from=0)*

Finds the first occurrence of a substring, ignoring case. Returns the starting position of the substring or -1 if not found. Optionally, the initial search index can be passed.

- *String **format** (var values, String placeholder={_})*

Formats the string by replacing all occurrences of placeholder with values.

- *String **get_base_dir** ()*

If the string is a valid file path, returns the base directory name.

- *String **get_basename** ()*

If the string is a valid file path, returns the full file path without the extension.

- *String **get_extension** ()*

If the string is a valid file path, returns the extension.

- *String **get_file** ()*

If the string is a valid file path, returns the filename.

- *int **hash** ()*

Hashes the string and returns a 32-bit integer.

- *int **hex_to_int** ()*

Converts a string containing a hexadecimal number into an integer.

- *String insert (int position, String what)*

Inserts a substring at a given position.

- *bool is_abs_path ()*

If the string is a path to a file or directory, returns `true` if the path is absolute.

- *bool is_rel_path ()*

If the string is a path to a file or directory, returns `true` if the path is relative.

- *bool is_subsequence_of (String text)*

Returns `true` if this string is a subsequence of the given string.

- *bool is_subsequence_ofi (String text)*

Returns `true` if this string is a subsequence of the given string, without considering case.

- *bool is_valid_float ()*

Returns `true` if this string contains a valid float.

- *bool is_valid_html_color ()*

Returns `true` if this string contains a valid color in HTML notation.

- *bool is_valid_identifier ()*

Returns `true` if this string is a valid identifier. A valid identifier may contain only letters, digits and underscores (`_`) and the first character may not be a digit.

- *bool is_valid_integer ()*

Returns `true` if this string contains a valid integer.

- *bool is_valid_ip_address ()*

Returns `true` if this string contains a valid IP address.

- *String json_escape ()*

Returns a copy of the string with special characters escaped using the JSON standard.

- *String left (int position)*

Returns a number of characters from the left of the string.

- *int length ()*

Returns the string's amount of characters.

- *bool match (String expr)*

Does a simple expression match, where '*' matches zero or more arbitrary characters and '?' matches any single character except '.'.

- *bool matchn (String expr)*

Does a simple case insensitive expression match, using ? and * wildcards (see *match*).

- *PoolByteArray md5_buffer ()*

Returns the MD5 hash of the string as an array of bytes.

- *String md5_text ()*

Returns the MD5 hash of the string as a string.

- `int nocasecmp_to (String to)`

Performs a case-insensitive comparison to another string. Returns `-1` if less than, `+1` if greater than, or `0` if equal.

- `int ord_at (int at)`

Returns the character code at position `at`.

- `String pad_decimals (int digits)`

Formats a number to have an exact number of `digits` after the decimal point.

- `String pad_zeros (int digits)`

Formats a number to have an exact number of `digits` before the decimal point.

- `String percent_decode ()`

Decode a percent-encoded string. See `percent_encode`.

- `String percent_encode ()`

Percent-encodes a string. Encodes parameters in a URL when sending a HTTP GET request (and bodies of form-urlencoded POST requests).

- `String plus_file (String file)`

If the string is a path, this concatenates `file` at the end of the string as a subpath. E.g. `"this/is".plus_file("path") == "this/is/path"`.

- `String replace (String what, String forwhat)`

Replaces occurrences of a substring with the given one inside the string.

- `String replacen (String what, String forwhat)`

Replaces occurrences of a substring with the given one inside the string. Ignores case.

- `int rfind (String what, int from=-1)`

Performs a search for a substring, but starts from the end of the string instead of the beginning.

- `int rfindn (String what, int from=-1)`

Performs a search for a substring, but starts from the end of the string instead of the beginning. Ignores case.

- `String right (int position)`

Returns the right side of the string from a given position.

- `PoolByteArray sha256_buffer ()`

- `String sha256_text ()`

Returns the SHA-256 hash of the string as a string.

- `float similarity (String text)`

Returns the similarity index of the text compared to this string. 1 means totally similar and 0 means totally dissimilar.

- `PoolStringArray split (String divisor, bool allow_empty=True, int maxsplit=0)`

Splits the string by a divisor string and returns an array of the substrings. Example “One,Two,Three” will return “One”,“Two”,“Three” if split by “,”.

If `maxsplit` is given, at most `maxsplit` number of splits occur, and the remainder of the string is returned as the final element of the list (thus, the list will have at most `maxsplit+1` elements)

- `PoolRealArray split_floats (String divisor, bool allow_empty=True)`

Splits the string in floats by using a divisor string and returns an array of the substrings. Example “1,2.5,3” will return 1,2.5,3 if split by “,”.

- *String strip_edges (bool left=True, bool right=True)*

Returns a copy of the string stripped of any non-printable character at the beginning and the end. The optional arguments are used to toggle stripping on the left and right edges respectively.

- *String substr (int from, int len)*

Returns part of the string from the position `from` with length `len`.

- *PoolByteArray to_ascii ()*

Converts the String (which is a character array) to *PoolByteArray* (which is an array of bytes). The conversion is sped up in comparison to `to_utf8()` with the assumption that all the characters the String contains are only ASCII characters.

- *float to_float ()*

Converts a string containing a decimal number into a `float`.

- *int to_int ()*

Converts a string containing an integer number into an `int`.

- *String to_lower ()*

Returns the string converted to lowercase.

- *String to_upper ()*

Returns the string converted to uppercase.

- *PoolByteArray to_utf8 ()*

Converts the String (which is an array of characters) to *PoolByteArray* (which is an array of bytes). The conversion is a bit slower than `to_ascii()`, but supports all UTF-8 characters. Therefore, you should prefer this function over `to_ascii()`.

- *String xml_escape ()*

Returns a copy of the string with special characters escaped using the XML standard.

- *String xml_unescape ()*

Returns a copy of the string with escaped characters replaced by their meanings according to the XML standard.

25.383 StyleBox

Inherits: *Resource < Reference < Object*

Inherited By: *StyleBoxLine, StyleBoxFlat, StyleBoxTexture, StyleBoxEmpty*

Category: Core

25.383.1 Brief Description

Base class for drawing stylized boxes for the UI.

25.383.2 Member Functions

<code>void</code>	<code>draw (RID canvas_item, Rect2 rect) const</code>
<code>Vector2</code>	<code>get_center_size () const</code>
<code>float</code>	<code>get_margin (int margin) const</code>
<code>Vector2</code>	<code>get_minimum_size () const</code>
<code>Vector2</code>	<code>get_offset () const</code>
<code>bool</code>	<code>test_mask (Vector2 point, Rect2 rect) const</code>

25.383.3 Member Variables

- `float content_margin_bottom`
- `float content_margin_left`
- `float content_margin_right`
- `float content_margin_top`

25.383.4 Description

StyleBox is *Resource* that provides an abstract base class for drawing stylized boxes for the UI. StyleBoxes are used for drawing the styles of buttons, line edit backgrounds, tree backgrounds, etc. and also for testing a transparency mask for pointer signals. If mask test fails on a StyleBox assigned as mask to a control, clicks and motion signals will go through it to the one below.

25.383.5 Member Function Description

- `void draw (RID canvas_item, Rect2 rect) const`
- `Vector2 get_center_size () const`
- `float get_margin (int margin) const`

Return the offset of margin “margin” (see MARGIN_* enum).

- `Vector2 get_minimum_size () const`

Return the minimum size that this stylebox can be shrunk to.

- `Vector2 get_offset () const`

Return the “offset” of a stylebox, this is a helper function, like writing `Vector2(style.get_margin(MARGIN_LEFT), style.get_margin(MARGIN_TOP))`.

- `bool test_mask (Vector2 point, Rect2 rect) const`

Test a position in a rectangle, return whether it passes the mask test.

25.384 StyleBoxEmpty

Inherits: `StyleBox < Resource < Reference < Object`

Category: Core

25.384.1 Brief Description

Empty stylebox (does not display anything).

25.384.2 Description

Empty stylebox (really does not display anything).

25.385 StyleBoxFlat

Inherits: *StyleBox < Resource < Reference < Object*

Category: Core

25.385.1 Brief Description

Customizable Stylebox with a given set of parameters. (no texture required)

25.385.2 Member Functions

<i>int</i>	<i>get_border_width_min () const</i>
<i>void</i>	<i>set_border_width_all (int width)</i>
<i>void</i>	<i>set_corner_radius_all (int radius)</i>
<i>void</i>	<i>set_corner_radius_individual (int radius_top_left, int radius_top_right, int radius_bottom_right, int radius_bottom_left)</i>
<i>void</i>	<i>set_expand_margin_all (float size)</i>
<i>void</i>	<i>set_expand_margin_individual (float size_left, float size_top, float size_right, float size_bottom)</i>

25.385.3 Member Variables

- *bool anti_aliasing* - Anti Aliasing draws a small ring around edges. This ring fades to transparent. As a result edges look much smoother. This is only noticeable when using rounded corners.
- *int anti_aliasing_size* - This changes the size of the faded ring. Higher values can be used to achieve a “blurry” effect.
- *Color bg_color* - The background color of the stylebox.
- *bool border_blend* - When set to true, the border will fade into the background color.
- *Color border_color* - Sets the color of the border.
- *int border_width_bottom* - Border width for the bottom border.
- *int border_width_left* - Border width for the left border.
- *int border_width_right* - Border width for the right border.
- *int border_width_top* - Border width for the top border.

- *int corner_detail* - This sets the amount of vertices used for each corner. Higher values result in rounder corners but take more processing power to compute. When choosing a value you should take the corner radius (`set_corner_radius`) into account.

For corner radius smaller than 10: 4-5 should be enough

For corner radius smaller than 30: 8-12 should be enough ...

- *int corner_radius_bottom_left* - The corner radius of the bottom left corner. When set to 0 the corner is not rounded.
- *int corner_radius_bottom_right* - The corner radius of the bottom right corner. When set to 0 the corner is not rounded.
- *int corner_radius_top_left* - The corner radius of the top left corner. When set to 0 the corner is not rounded.
- *int corner_radius_top_right* - The corner radius of the top right corner. When set to 0 the corner is not rounded.
- *bool draw_center* - Toggles drawing of the inner part of the stylebox.
- *float expand_margin_bottom* - Expands the stylebox outside of the control rect on the bottom edge. Useful in combination with `border_width_bottom`. To draw a border outside the control rect.
- *float expand_margin_left* - Expands the stylebox outside of the control rect on the left edge. Useful in combination with `border_width_left`. To draw a border outside the control rect.
- *float expand_margin_right* - Expands the stylebox outside of the control rect on the right edge. Useful in combination with `border_width_right`. To draw a border outside the control rect.
- *float expand_margin_top* - Expands the stylebox outside of the control rect on the top edge. Useful in combination with `border_width_top`. To draw a border outside the control rect.
- *Color shadow_color* - The color of the shadow. (This has no effect when `shadow_size < 1`)
- *int shadow_size* - The shadow size in pixels.

25.385.4 Description

This stylebox can be used to achieve all kinds of looks without the need of a texture. Those properties are customizable:

- Color
- Border width (individual width for each border)
- Rounded corners (individual radius for each corner)
- Shadow

About corner radius:

Setting corner radius to high values is allowed. As soon as corners would overlap the stylebox will switch to a relative system. Example:

```
height = 30
corner_radius_top_left = 50
corner_radius_bottom_left = 100
```

The relative system now would take the 1:2 ratio of the two left corners to calculate the actual corner width. Both corners added will **never** be more than the height. Result:

```
corner_radius_top_left: 10
corner_radius_bottom_left: 20
```

25.385.5 Member Function Description

- `int get_border_width_min () const`
- `void set_border_width_all (int width)`
- `void set_corner_radius_all (int radius)`
- `void set_corner_radius_individual (int radius_top_left, int radius_top_right, int radius_bottom_right, int radius_bottom_left)`
- `void set_expand_margin_all (float size)`
- `void set_expand_margin_individual (float size_left, float size_top, float size_right, float size_bottom)`

25.386 StyleBoxLine

Inherits: `StyleBox < Resource < Reference < Object`

Category: Core

25.386.1 Brief Description

25.386.2 Member Variables

- `Color color`
- `float grow`
- `int thickness`
- `bool vertical`

25.387 StyleBoxTexture

Inherits: `StyleBox < Resource < Reference < Object`

Category: Core

25.387.1 Brief Description

Texture Based 3x3 scale style.

25.387.2 Member Functions

<code>void</code>	<code>set_expand_margin_all (float size)</code>
<code>void</code>	<code>set_expand_margin_individual (float size_left, float size_top, float size_right, float size_bottom)</code>

25.387.3 Signals

- `texture_changed()`

25.387.4 Member Variables

- `AxisStretchMode axis_stretch_horizontal`
- `AxisStretchMode axis_stretch_vertical`
- `bool draw_center`
- `float expand_margin_bottom`
- `float expand_margin_left`
- `float expand_margin_right`
- `float expand_margin_top`
- `float margin_bottom`
- `float margin_left`
- `float margin_right`
- `float margin_top`
- `Color modulate_color`
- `Resource normal_map`
- `Rect2 region_rect`
- `Resource texture`

25.387.5 Enums

enum **AxisStretchMode**

- `AXIS_STRETCH_MODE_STRETCH = 0`
- `AXIS_STRETCH_MODE_TILE = 1`
- `AXIS_STRETCH_MODE_TILE_FIT = 2`

25.387.6 Description

Texture Based 3x3 scale style. This stylebox performs a 3x3 scaling of a texture, where only the center cell is fully stretched. This allows for the easy creation of bordered styles.

25.387.7 Member Function Description

- `void set_expand_margin_all (float size)`
- `void set_expand_margin_individual (float size_left, float size_top, float size_right, float size_bottom)`

25.388 SurfaceTool

Inherits: Reference < Object

Category: Core

25.388.1 Brief Description

Helper tool to create geometry.

25.388.2 Member Functions

void	<i>add_bones (PoolIntArray bones)</i>
void	<i>add_color (Color color)</i>
void	<i>add_index (int index)</i>
void	<i>add_normal (Vector3 normal)</i>
void	<i>add_smooth_group (bool smooth)</i>
void	<i>add_tangent (Plane tangent)</i>
void	<i>add_to_format (int flags)</i>
void	<i>add_triangle_fan (PoolVector3Array vertexes, PoolVector2Array uvs=PoolVector2Array(), PoolColorArray colors=PoolColorArray(), PoolVector2Array uv2s=PoolVector2Array(), PoolVector3Array normals=PoolVector3Array(), Array tangents=[])</i>
void	<i>add_uv (Vector2 uv)</i>
void	<i>add_uv2 (Vector2 uv2)</i>
void	<i>add_vertex (Vector3 vertex)</i>
void	<i>add_weights (PoolRealArray weights)</i>
void	<i>append_from (Mesh existing, int surface, Transform transform)</i>
void	<i>begin (int primitive)</i>
void	<i>clear ()</i>
Ar-rayMesh	<i>commit (ArrayMesh existing=null, int flags=97792)</i>
void	<i>create_from (Mesh existing, int surface)</i>
void	<i>deindex ()</i>
void	<i>generate_normals ()</i>
void	<i>generate_tangents ()</i>
void	<i>index ()</i>
void	<i>set_material (Material material)</i>

25.388.3 Description

The `SurfaceTool` is used to construct a `Mesh` by specifying vertex attributes individually. It can be used to construct a `Mesh` from script. All properties except `index` need to be added before a call to `add_vertex`. For example adding vertex colors and UVs looks like

```
var st = SurfaceTool.new()
st.begin(Mesh.PRIMITIVE_TRIANGLES)
st.add_color(Color(1, 0, 0))
st.add_uv(Vector2(0, 0))
st.add_vertex(Vector3(0, 0, 0))
```

The `SurfaceTool` now contains one vertex of a triangle which has a UV coordinate and a specified `Color`. If another vertex were added without calls to `add_uv` or `add_color` then the last values would be used.

It is very important that vertex attributes are passed **before** the call to `add_vertex`, failure to do this will result in an error when committing the vertex information to a mesh.

25.388.4 Member Function Description

- void **add_bones** (*PoolIntArray* bones)

Add an array of bones for the next Vertex to use.

- void **add_color** (*Color* color)

Specify a `Color` for the next Vertex to use.

- void **add_index** (*int* index)

Adds an index to index array if you are using indexed Vertices. Does not need to be called before adding Vertex.

- void **add_normal** (*Vector3* normal)

Specify a normal for the next Vertex to use.

- void **add_smooth_group** (*bool* smooth)

Specify whether current Vertex (if using only Vertex arrays) or current index (if also using index arrays) should utilize smooth normals for normal calculation.

- void **add_tangent** (*Plane* tangent)

Specify a Tangent for the next Vertex to use.

- void **add_to_format** (*int* flags)

- void **add_triangle_fan** (*PoolVector3Array* vertexes, *PoolVector2Array* uvs=PoolVector2Array(), *PoolColorArray* colors=PoolColorArray(), *PoolVector2Array* uv2s=PoolVector2Array(), *PoolVector3Array* normals=PoolVector3Array(), *Array* tangents=[])

Insert a triangle fan made of array data into *Mesh* being constructed.

- void **add_uv** (*Vector2* uv)

Specify UV Coordinate for next Vertex to use.

- void **add_uv2** (*Vector2* uv2)

Specify an optional second set of UV coordinates for next Vertex to use.

- void **add_vertex** (*Vector3* vertex)

Specify position of current Vertex. Should be called after specifying other vertex properties (e.g. Color, UV).

- void **add_weights** (*PoolRealArray* weights)

Specify weight value for next Vertex to use.

- void **append_from** (*Mesh* existing, *int* surface, *Transform* transform)

- void **begin** (*int* primitive)

Called before adding any Vertices. Takes the primitive type as an argument (e.g. `Mesh.PRIMITIVE_TRIANGLES`).

- void **clear** ()

Clear all information passed into the surface tool so far.

- *ArrayMesh* **commit** (*ArrayMesh* existing=null, *int* flags=97792)

Returns a constructed *ArrayMesh* from current information passed in. If an existing *ArrayMesh* is passed in as an argument, will add an extra surface to the existing *ArrayMesh*.

- void **create_from** (*Mesh* existing, *int* surface)
- void **deindex** ()

Removes index array by expanding Vertex array.

- void **generate_normals** ()

Generates normals from Vertices so you do not have to do it manually.

- void **generate_tangents** ()
- void **index** ()

Shrinks Vertex array by creating an index array. Avoids reusing Vertices.

- void **set_material** (*Material* material)

Sets *Material* to be used by the *Mesh* you are constructing.

25.389 TabContainer

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

25.389.1 Brief Description

Tabbed Container.

25.389.2 Member Functions

<i>Control</i>	<i>get_current_tab_control</i> () const
<i>Popup</i>	<i>get_popup</i> () const
<i>int</i>	<i>get_previous_tab</i> () const
<i>Control</i>	<i>get_tab_control</i> (<i>int</i> idx) const
<i>int</i>	<i>get_tab_count</i> () const
<i>bool</i>	<i>get_tab_disabled</i> (<i>int</i> tab_idx) const
<i>Texture</i>	<i>get_tab_icon</i> (<i>int</i> tab_idx) const
<i>String</i>	<i>get_tab_title</i> (<i>int</i> tab_idx) const
<i>void</i>	<i>set_popup</i> (<i>Node</i> popup)
<i>void</i>	<i>set_tab_disabled</i> (<i>int</i> tab_idx, <i>bool</i> disabled)
<i>void</i>	<i>set_tab_icon</i> (<i>int</i> tab_idx, <i>Texture</i> icon)
<i>void</i>	<i>set_tab_title</i> (<i>int</i> tab_idx, <i>String</i> title)

25.389.3 Signals

- **pre_popup_pressed** ()

Emitted when the TabContainer's *Popup* button is clicked. See *set_popup* for details.

- **tab_changed** (*int* tab)

Emitted when switching to another tab.

- **tab_selected (int tab)**

Emitted when a tab is selected, even if it is the current tab.

25.389.4 Member Variables

- **int current_tab** - The current tab index. When set, this index's *Control* node's `visible` property is set to `true` and all others are set to `false`.
- **TabAlign tab_align** - The alignment of all tabs in the tab container. See the `ALIGN__*` constants for details.
- **bool tabs_visible** - If `true` tabs are visible. If `false` tabs' content and titles are hidden. Default value: `true`.

25.389.5 Enums

enum **TabAlign**

- **ALIGN_LEFT = 0**
- **ALIGN_CENTER = 1**
- **ALIGN_RIGHT = 2**

25.389.6 Description

Sets the active tab's `visible` property to the value `true`. Sets all other children's to `false`.

Ignores non-*Control* children.

Individual tabs are always visible unless you use `set_tab_disabled` and `set_tab_title` to hide it.

To hide only a tab's content, nest the content inside a child *Control*, so it receives the *TabContainer*'s visibility setting instead.

25.389.7 Member Function Description

- **Control get_current_tab_control () const**

Returns the child *Control* node located at the active tab index.

- **Popup get_popup () const**

Returns the *Popup* node instance if one has been set already with `set_popup`.

- **int get_previous_tab () const**

Returns the previously active tab index.

- **Control get_tab_control (int idx) const**

Returns the currently visible tab's *Control* node.

- **int get_tab_count () const**

Returns the number of tabs.

- `bool get_tab_disabled (int tab_idx) const`

Returns `true` if the tab at index `tab_idx` is disabled.

- `Texture get_tab_icon (int tab_idx) const`

Returns the `Texture` for the tab at index `tab_idx` or null if the tab has no `Texture`.

- `String get_tab_title (int tab_idx) const`

Returns the title of the tab at index `tab_idx`. Tab titles default to the name of the indexed child node, but this can be overridden with `set_tab_title`.

- `void set_popup (Node popup)`

If set on a `Popup` node instance, a popup menu icon appears in the top-right corner of the `TabContainer`. Clicking it will expand the `Popup` node.

- `void set_tab_disabled (int tab_idx, bool disabled)`

If `disabled` is false, hides the tab at index `tab_idx`. Note that its title text will remain, unless also removed with `set_tab_title`.

- `void set_tab_icon (int tab_idx, Texture icon)`

Sets an icon for the tab at index `tab_idx`.

- `void set_tab_title (int tab_idx, String title)`

Sets a title for the tab at index `tab_idx`. Tab titles default to the name of the indexed child node, but this can be overridden with `set_tab_title`.

25.390 Tabs

Inherits: `Control < CanvasItem < Node < Object`

Category: Core

25.390.1 Brief Description

Tabs Control.

25.390.2 Member Functions

void	<i>add_tab (String title="" , Texture icon=null)</i>
void	<i>ensure_tab_visible (int idx)</i>
bool	<i>get_offset_buttons_visible () const</i>
int	<i>get_tab_count () const</i>
bool	<i>get_tab_disabled (int tab_idx) const</i>
Texture	<i>get_tab_icon (int tab_idx) const</i>
int	<i>get_tab_offset () const</i>
Rect2	<i>get_tab_rect (int tab_idx) const</i>
String	<i>get_tab_title (int tab_idx) const</i>
void	<i>move_tab (int from, int to)</i>
void	<i>remove_tab (int tab_idx)</i>
void	<i>set_tab_disabled (int tab_idx, bool disabled)</i>
void	<i>set_tab_icon (int tab_idx, Texture icon)</i>
void	<i>set_tab_title (int tab_idx, String title)</i>

25.390.3 Signals

- **reposition_active_tab_request (int idx_to)**
- **right_button_pressed (int tab)**
- **tab_changed (int tab)**
- **tab_clicked (int tab)**
- **tab_close (int tab)**
- **tab_hover (int tab)**

25.390.4 Member Variables

- *int current_tab*
- *bool scrolling_enabled*
- *TabAlign tab_align*
- *CloseButtonDisplayPolicy tab_close_display_policy*

25.390.5 Enums

enum **CloseButtonDisplayPolicy**

- **CLOSE_BUTTON_SHOW_NEVER = 0**
- **CLOSE_BUTTON_SHOW_ACTIVE_ONLY = 1**
- **CLOSE_BUTTON_SHOW_ALWAYS = 2**
- **CLOSE_BUTTON_MAX = 3**

enum **TabAlign**

- **ALIGN_LEFT = 0**
- **ALIGN_CENTER = 1**
- **ALIGN_RIGHT = 2**
- **ALIGN_MAX = 3**

25.390.6 Description

Simple tabs control, similar to *TabContainer* but is only in charge of drawing tabs, not interact with children.

25.390.7 Member Function Description

- void **add_tab** (*String title=""*, *Texture icon=null*)
- void **ensure_tab_visible** (*int idx*)
- *bool get_offset_buttons_visible () const*
- *int get_tab_count () const*
- *bool get_tab_disabled (int tab_idx) const*
- *Texture get_tab_icon (int tab_idx) const*
- *int get_tab_offset () const*
- *Rect2 get_tab_rect (int tab_idx) const*

Returns tab *Rect2* with local position and size.

- *String get_tab_title (int tab_idx) const*
- void **move_tab** (*int from, int to*)

Rearrange tab.

- void **remove_tab** (*int tab_idx*)
- void **set_tab_disabled** (*int tab_idx, bool disabled*)
- void **set_tab_icon** (*int tab_idx, Texture icon*)
- void **set_tab_title** (*int tab_idx, String title*)

25.391 TCP_Server

Inherits: *Reference < Object*

Category: Core

25.391.1 Brief Description

TCP Server.

25.391.2 Member Functions

<code>bool</code>	<code>is_connection_available () const</code>
<code>int</code>	<code>listen (int port, String bind_address="*")</code>
<code>void</code>	<code>stop ()</code>
<code>StreamPeerTCP</code>	<code>take_connection ()</code>

25.391.3 Description

TCP Server class. Listens to connections on a port and returns a *StreamPeerTCP* when got a connection.

25.391.4 Member Function Description

- `bool is_connection_available () const`

Return true if a connection is available for taking.

- `int listen (int port, String bind_address="*")`

Listen on the “port” binding to “bind_address”.

If “bind_address” is set as “*” (default), the server will listen on all available addresses (both IPv4 and IPv6).

If “bind_address” is set as “0.0.0.0” (for IPv4) or “::” (for IPv6), the server will listen on all available addresses matching that IP type.

If “bind_address” is set to any valid address (e.g. “192.168.1.101”, “::1”, etc), the server will only listen on the interface with that addresses (or fail if no interface with the given address exists).

- `void stop ()`

Stop listening.

- `StreamPeerTCP take_connection ()`

If a connection is available, return a StreamPeerTCP with the connection/

25.392 TextEdit

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.392.1 Brief Description

Multiline text editing control.

25.392.2 Member Functions

<code>void</code>	<code>add_color_region (String begin_key, String end_key, Color color, bool line_only=false)</code>
<code>void</code>	<code>add_keyword_color (String keyword, Color color)</code>

Continued on next page

Table 25.21 – continued from previous page

<i>bool</i>	<i>can_fold (int line) const</i>
<i>void</i>	<i>clear_colors ()</i>
<i>void</i>	<i>clear_undo_history ()</i>
<i>void</i>	<i>copy ()</i>
<i>int</i>	<i>cursor_get_column () const</i>
<i>int</i>	<i>cursor_get_line () const</i>
<i>void</i>	<i>cursor_set_column (int column, bool adjust_viewport=true)</i>
<i>void</i>	<i>cursor_set_line (int line, bool adjust_viewport=true, bool can_be_hidden=true)</i>
<i>void</i>	<i>cut ()</i>
<i>void</i>	<i>deselect ()</i>
<i>void</i>	<i>fold_all_lines ()</i>
<i>void</i>	<i>fold_line (int line)</i>
<i>String</i>	<i>get_line (int line) const</i>
<i>int</i>	<i>get_line_count () const</i>
<i>PopupMenu</i>	<i>get_menu () const</i>
<i>int</i>	<i>get_selection_from_column () const</i>
<i>int</i>	<i>get_selection_from_line () const</i>
<i>String</i>	<i>get_selection_text () const</i>
<i>int</i>	<i>get_selection_to_column () const</i>
<i>int</i>	<i>get_selection_to_line () const</i>
<i>String</i>	<i>get_word_under_cursor () const</i>
<i>void</i>	<i>insert_text_at_cursor (String text)</i>
<i>bool</i>	<i>is_folded (int line) const</i>
<i>bool</i>	<i>is_line_hidden (int line) const</i>
<i>bool</i>	<i>is_selection_active () const</i>
<i>void</i>	<i>menu_option (int option)</i>
<i>void</i>	<i>paste ()</i>
<i>void</i>	<i>redo ()</i>
<i>PoolIntArray</i>	<i>search (String key, int flags, int from_line, int from_column) const</i>
<i>void</i>	<i>select (int from_line, int from_column, int to_line, int to_column)</i>
<i>void</i>	<i>select_all ()</i>
<i>void</i>	<i>set_line_as_hidden (int line, bool enable)</i>
<i>void</i>	<i>toggle_fold_line (int line)</i>
<i>void</i>	<i>undo ()</i>
<i>void</i>	<i>unfold_line (int line)</i>
<i>void</i>	<i>unhide_all_lines ()</i>

25.392.3 Signals

- **breakpoint_toggled (int row)**

Emitted when a breakpoint is placed via the breakpoint gutter.

- **cursor_changed ()**

Emitted when the cursor changes.

- **request_completion ()**
- **symbol_lookup (String symbol, int row, int column)**
- **text_changed ()**

Emitted when the text changes.

25.392.4 Member Variables

- *bool caret_blink* - If `true` the caret (visual cursor) blinks.
- *float caret_blink_speed* - Duration (in seconds) of a caret's blinking cycle.
- *bool caret_block_mode* - If `true` the caret displays as a rectangle.

If `false` the caret displays as a bar.

- *bool caret_moving_by_right_click* - If `true` a right click moves the cursor at the mouse position before displaying the context menu.

If `false` the context menu disregards mouse location.

- *bool context_menu_enabled* - If `true` a right click displays the context menu.
- *int hiding_enabled*
- *bool highlight_all_occurrences*
- *bool highlight_current_line* - If `true` the line containing the cursor is highlighted.
- *bool override_selected_font_color*
- *bool readonly* - If `true` read-only mode is enabled. Existing text cannot be modified and new text cannot be added.
- *bool show_line_numbers* - If `true` line numbers are displayed to the left of the text.
- *bool smooth_scrolling*
- *bool syntax_highlighting*
- *String text* - String value of the *TextEdit*.
- *float v_scroll_speed* - If `true`, enables text wrapping when it goes beyond the edge of what is visible.
- *bool wrap_lines*

25.392.5 Enums

enum MenuItem

- **MENU_CUT = 0** — Cuts (Copies and clears) the selected text.
- **MENU_COPY = 1** — Copies the selected text.
- **MENU_PASTE = 2** — Pastes the clipboard text over the selected text (or at the cursor's position).
- **MENU_CLEAR = 3** — Erases the whole *TextEdit* text.
- **MENU_SELECT_ALL = 4** — Selects the whole *TextEdit* text.
- **MENU_UNDO = 5** — Undoes the previous action.
- **MENU_MAX = 6**

enum SearchFlags

- **SEARCH_MATCH_CASE = 1** — Match case when searching.
- **SEARCH_WHOLE_WORDS = 2** — Match whole words when searching.
- **SEARCH_BACKWARDS = 4** — Search from end to beginning.

25.392.6 Description

TextEdit is meant for editing large, multiline text. It also has facilities for editing code, such as syntax highlighting support and multiple levels of undo/redo.

25.392.7 Member Function Description

- void **add_color_region** (*String begin_key, String end_key, Color color, bool line_only=false*)

Add color region (given the delimiters) and its colors.

- void **add_keyword_color** (*String keyword, Color color*)

Add a keyword and its color.

- *bool can_fold (int line) const*
- void **clear_colors** ()

Clear all the syntax coloring information.

- void **clear_undo_history** ()

Clear the undo history.

- void **copy** ()

Copy the current selection.

- *int cursor_get_column () const*

Return the column the editing cursor is at.

- *int cursor_get_line () const*

Return the line the editing cursor is at.

- void **cursor_set_column** (*int column, bool adjust_viewport=true*)
- void **cursor_set_line** (*int line, bool adjust_viewport=true, bool can_be_hidden=true*)
- void **cut** ()

Cut the current selection.

- void **deselect** ()

Clears the current selection.

- void **fold_all_lines** ()
- void **fold_line** (*int line*)
- *String get_line (int line) const*

Return the text of a specific line.

- *int get_line_count () const*

Return the amount of total lines in the text.

- *PopupMenu* **get_menu()** const
- *int* **get_selection_from_column()** const

Return the selection begin column.

- *int* **get_selection_from_line()** const

Return the selection begin line.

- *String* **get_selection_text()** const

Return the text inside the selection.

- *int* **get_selection_to_column()** const

Return the selection end column.

- *int* **get_selection_to_line()** const

Return the selection end line.

- *String* **get_word_under_cursor()** const
- *void* **insert_text_at_cursor(String text)**

Insert a given text at the cursor position.

- *bool* **is_folded(int line)** const
- *bool* **is_line_hidden(int line)** const
- *bool* **is_selection_active()** const

Return true if the selection is active.

- *void* **menu_option(int option)**
- *void* **paste()**

Paste the current selection.

- *void* **redo()**

Perform redo operation.

- *PoolIntArray* **search(String key, int flags, int from_line, int from_column)** const

Perform a search inside the text. Search flags can be specified in the SEARCH_* enum.

- *void* **select(int from_line, int from_column, int to_line, int to_column)**

Perform selection, from line/column to line/column.

- *void* **select_all()**

Select all the text.

- *void* **set_line_as_hidden(int line, bool enable)**
- *void* **toggle_fold_line(int line)**

Toggle the folding of the code block at the given line.

- *void* **undo()**

Perform undo operation.

- *void* **unfold_line(int line)**
- *void* **unhide_all_lines()**

25.393 Texture

Inherits: *Resource < Reference < Object*

Inherited By: *CurveTexture, AtlasTexture, ProxyTexture, GradientTexture, ViewportTexture, StreamTexture, ImageTexture, LargeTexture*

Category: Core

25.393.1 Brief Description

Texture for 2D and 3D.

25.393.2 Member Functions

void	<i>draw (RID canvas_item, Vector2 position, Color modulate=Color(1, 1, 1, 1), bool transpose=false, Texture normal_map=null) const</i>
void	<i>draw_rect (RID canvas_item, Rect2 rect, bool tile, Color modulate=Color(1, 1, 1, 1), bool transpose=false, Texture normal_map=null) const</i>
void	<i>draw_rect_region (RID canvas_item, Rect2 rect, Rect2 src_rect, Color modulate=Color(1, 1, 1, 1), bool transpose=false, Texture normal_map=null, bool clip_uv=true) const</i>
<i>Image</i>	<i>get_data () const</i>
<i>int</i>	<i>get_height () const</i>
<i>Vector2</i>	<i>get_size () const</i>
<i>int</i>	<i>get_width () const</i>
<i>bool</i>	<i>has_alpha () const</i>

25.393.3 Member Variables

- *int flags* - The texture's flags.

25.393.4 Enums

enum Flags

- **FLAGS_DEFAULT = 7** — Default flags. Generate mipmaps, repeat, and filter are enabled.
- **FLAG_MIPMAPS = 1** — Generate mipmaps, which are smaller versions of the same texture to use when zoomed out, keeping the aspect ratio.
- **FLAG_REPEAT = 2** — Repeats texture (instead of clamp to edge).
- **FLAG_FILTER = 4** — Magnifying filter, to enable smooth zooming in of the texture.
- **FLAG_ANISOTROPIC_FILTER = 8** — Anisotropic mipmap filtering. Generates smaller versions of the same texture with different aspect ratios.

More effective on planes often shown going to the horizon as those textures (Walls or Ground for example) get squashed in the viewport to different aspect ratios and regular mipmaps keep the aspect ratio so they don't optimize storage that well in those cases.

- **FLAG_CONVERT_TO_LINEAR = 16** — Converts texture to SRGB color space.
- **FLAG_MIRRORED_REPEAT = 32** — Repeats texture with alternate sections mirrored.
- **FLAG_VIDEO_SURFACE = 4096** — Texture is a video surface.

25.393.5 Description

A texture works by registering an image in the video hardware, which then can be used in 3D models or 2D *Sprite* or *GUI Control*.

25.393.6 Member Function Description

- void **draw** (*RID canvas_item*, *Vector2 position*, *Color modulate=Color(1, 1, 1, 1)*, *bool transpose=false*, *Texture normal_map=null*) const
- void **draw_rect** (*RID canvas_item*, *Rect2 rect*, *bool tile*, *Color modulate=Color(1, 1, 1, 1)*, *bool transpose=false*, *Texture normal_map=null*) const
- void **draw_rect_region** (*RID canvas_item*, *Rect2 rect*, *Rect2 src_rect*, *Color modulate=Color(1, 1, 1, 1)*, *bool transpose=false*, *Texture normal_map=null*, *bool clip_uv=true*) const
- *Image get_data* () const
- *int get_height* () const

Return the texture height.

- *Vector2 get_size* () const

Return the texture size.

- *int get_width* () const

Return the texture width.

- *bool has_alpha* () const

25.394 TextureButton

Inherits: *BaseButton < Control < CanvasItem < Node < Object*

Category: Core

25.394.1 Brief Description

Texture-based button. Supports Pressed, Hover, Disabled and Focused states.

25.394.2 Member Variables

- *bool expand* - If *true* the texture stretches to the edges of the node's bounding rectangle using the *stretch_mode*. If *false* the texture will not scale with the node. Default value: *false*.

- *StretchMode stretch_mode* - Controls the texture's behavior when you resize the node's bounding rectangle, **only if expand** is true. Set it to one of the STRETCH__* constants. See the constants to learn more.
- *BitMap texture_click_mask* - Pure black and white Bitmap image to use for click detection. On the mask, white pixels represent the button's clickable area. Use it to create buttons with curved shapes.
- *Texture texture_disabled* - Texture to display when the node is disabled. See *BaseButton.disabled*.
- *Texture texture_focused* - Texture to display when the node has mouse or keyboard focus.
- *Texture texture_hover* - Texture to display when the mouse hovers the node.
- *Texture texture_normal* - Texture to display by default, when the node is **not** in the disabled, focused, hover or pressed state.
- *Texture texture_pressed* - Texture to display on mouse down over the node, if the node has keyboard focus and the player presses the enter key or if the player presses the *BaseButton.shortcut* key.

25.394.3 Enums

enum StretchMode

- **STRETCH_SCALE = 0** — Scale to fit the node's bounding rectangle.
- **STRETCH_TILE = 1** — Tile inside the node's bounding rectangle.
- **STRETCH_KEEP = 2** — The texture keeps its original size and stays in the bounding rectangle's top-left corner.
- **STRETCH_KEEP_CENTERED = 3** — The texture keeps its original size and stays centered in the node's bounding rectangle.
- **STRETCH_KEEP_ASPECT = 4** — Scale the texture to fit the node's bounding rectangle, but maintain the texture's aspect ratio.
- **STRETCH_KEEP_ASPECT_CENTERED = 5** — Scale the texture to fit the node's bounding rectangle, center it, and maintain its aspect ratio.
- **STRETCH_KEEP_ASPECT_COVERED = 6** — Scale the texture so that the shorter side fits the bounding rectangle. The other side clips to the node's limits.

25.394.4 Description

TextureButton has the same functionality as *Button*, except it uses sprites instead of Godot's *Theme* resource. It is faster to create, but it doesn't support localization like more complex Controls.

The Normal state's texture is required. Others are optional.

25.395 TextureProgress

Inherits: Range < Control < CanvasItem < Node < Object

Category: Core

25.395.1 Brief Description

Texture-based progress bar. Useful for loading screens and life or stamina bars.

25.395.2 Member Variables

- *int fill_mode* - The fill direction. Uses FILL_* constants.
- *bool nine_patch_stretch* - If true Godot treats the bar's textures like *NinePatchRect*. Use stretch_margin_*, like *stretch_margin_bottom*, to set up the nine patch's 3x3 grid. Default value: false.
- *Vector2 radial_center_offset* - Offsets *texture_progress* if *fill_mode* is FILL_CLOCKWISE or FILL_COUNTER_CLOCKWISE.
- *float radial_fill_degrees* - Upper limit for the fill of *texture_progress* if *fill_mode* is FILL_CLOCKWISE or FILL_COUNTER_CLOCKWISE. When the node's value is equal to its *max_value*, the texture fills up to this angle.

See *Range.value*, *Range.max_value*.

- *float radial_initial_angle* - Starting angle for the fill of *texture_progress* if *fill_mode* is FILL_CLOCKWISE or FILL_COUNTER_CLOCKWISE. When the node's value is equal to its *min_value*, the texture doesn't show up at all. When the value increases, the texture fills and tends towards *radial_fill_degrees*.
- *int stretch_margin_bottom* - The height of the 9-patch's bottom row. A margin of 16 means the 9-slice's bottom corners and side will have a height of 16 pixels. You can set all 4 margin values individually to create panels with non-uniform borders.
- *int stretch_margin_left* - The width of the 9-patch's left column.
- *int stretch_margin_right* - The width of the 9-patch's right column.
- *int stretch_margin_top* - The height of the 9-patch's top row.
- *Texture texture_over* - *Texture* that draws over the progress bar. Use it to add highlights or an upper-frame that hides part of *texture_progress*.
- *Texture texture_progress* - *Texture* that clips based on the node's value and *fill_mode*. As value increased, the texture fills up. It shows entirely when value reaches *max_value*. It doesn't show at all if value is equal to *min_value*.

The *value* property comes from *Range*. See *Range.value*, *Range.min_value*, *Range.max_value*.

- *Texture texture_under* - *Texture* that draws under the progress bar. The bar's background.

25.395.3 Enums

enum FillMode

- **FILL_LEFT_TO_RIGHT = 0** — The *texture_progress* fills from left to right.
- **FILL_RIGHT_TO_LEFT = 1** — The *texture_progress* fills from right to left.

- **FILL_TOP_TO_BOTTOM = 2** — The *texture_progress* fills from top to bottom.
- **FILL_BOTTOM_TO_TOP = 3** — The *texture_progress* fills from bottom to top.
- **FILL_CLOCKWISE = 4** — Turns the node into a radial bar. The *texture_progress* fills clockwise. See *radial_center_offset*, *radial_initial_angle* and *radial_fill_degrees* to refine its behavior.
- **FILL_COUNTER_CLOCKWISE = 5** — Turns the node into a radial bar. The *texture_progress* fills counter-clockwise. See *radial_center_offset*, *radial_initial_angle* and *radial_fill_degrees* to refine its behavior.

25.395.4 Description

TextureProgress works like *ProgressBar* but it uses up to 3 textures instead of Godot's *Theme* resource. Works horizontally, vertically, and radially.

25.396 TextureRect

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

25.396.1 Brief Description

Draws a sprite or a texture inside a User Interface. The texture can tile or not.

25.396.2 Member Variables

- *bool expand* - If `true`, the texture scales to fit its bounding rectangle. Default value: `false`.
- *StretchMode stretch_mode* - Controls the texture's behavior when you resize the node's bounding rectangle. Set it to one of the `STRETCH___*` constants. See the constants to learn more.
- *Texture texture* - The node's *Texture* resource.

25.396.3 Enums

enum StretchMode

- **STRETCH_SCALE_ON_EXPAND = 0** — Scale to fit the node's bounding rectangle, only if `expand` is `true`. Default `stretch_mode`, for backwards compatibility. Until you set `expand` to `true`, the texture will behave like `STRETCH_KEEP`.
- **STRETCH_SCALE = 1** — Scale to fit the node's bounding rectangle.
- **STRETCH_TILE = 2** — Tile inside the node's bounding rectangle.
- **STRETCH_KEEP = 3** — The texture keeps its original size and stays in the bounding rectangle's top-left corner.
- **STRETCH_KEEP_CENTERED = 4** — The texture keeps its original size and stays centered in the node's bounding rectangle.

- **STRETCH_KEEP_ASPECT = 5** — Scale the texture to fit the node's bounding rectangle, but maintain the texture's aspect ratio.
- **STRETCH_KEEP_ASPECT_CENTERED = 6** — Scale the texture to fit the node's bounding rectangle, center it and maintain its aspect ratio.
- **STRETCH_KEEP_ASPECT_COVERED = 7** — Scale the texture so that the shorter side fits the bounding rectangle. The other side clips to the node's limits.

25.396.4 Description

Use TextureRect to draw icons and sprites in your User Interfaces. To create panels and menu boxes, take a look at NinePatchFrame. Its Stretch Mode property controls the texture's scale and placement. It can scale, tile and stay centered inside its bounding rectangle. TextureRect is one of the 5 most common nodes to create game UI.

25.397 Theme

Inherits: *Resource < Reference < Object*

Category: Core

25.397.1 Brief Description

Theme for controls.

25.397.2 Member Functions

void	<i>clear_color</i> (String name, String type)
void	<i>clear_constant</i> (String name, String type)
void	<i>clear_font</i> (String name, String type)
void	<i>clear_icon</i> (String name, String type)
void	<i>clear_stylebox</i> (String name, String type)
void	<i>copy_default_theme</i> ()
<i>Color</i>	<i>get_color</i> (String name, String type) const
<i>PoolStringArray</i>	<i>get_color_list</i> (String type) const
<i>int</i>	<i>get_constant</i> (String name, String type) const
<i>PoolStringArray</i>	<i>get_constant_list</i> (String type) const
<i>Font</i>	<i>get_font</i> (String name, String type) const
<i>PoolStringArray</i>	<i>get_font_list</i> (String type) const
<i>Texture</i>	<i>get_icon</i> (String name, String type) const
<i>PoolStringArray</i>	<i>get_icon_list</i> (String type) const
<i>StyleBox</i>	<i>get_stylebox</i> (String name, String type) const
<i>PoolStringArray</i>	<i>get_stylebox_list</i> (String type) const
<i>PoolStringArray</i>	<i>get_stylebox_types</i> () const
<i>PoolStringArray</i>	<i>get_type_list</i> (String type) const
<i>bool</i>	<i>has_color</i> (String name, String type) const
<i>bool</i>	<i>has_constant</i> (String name, String type) const
<i>bool</i>	<i>has_font</i> (String name, String type) const
<i>bool</i>	<i>has_icon</i> (String name, String type) const
<i>bool</i>	<i>has_stylebox</i> (String name, String type) const
void	<i>set_color</i> (String name, String type, Color color)
void	<i>set_constant</i> (String name, String type, int constant)
void	<i>set_font</i> (String name, String type, Font font)
void	<i>set_icon</i> (String name, String type, Texture texture)
void	<i>set_stylebox</i> (String name, String type, StyleBox texture)

25.397.3 Member Variables

- *Font default_font* - The theme's default font.

25.397.4 Description

Theme for skinning controls. Controls can be skinned individually, but for complex applications it's more efficient to just create a global theme that defines everything. This theme can be applied to any *Control*, and it and its children will automatically use it.

Theme resources can be alternatively loaded by writing them in a .theme file, see docs for more info.

25.397.5 Member Function Description

- void **clear_color** (String name, String type)

Clears theme *Color* at name if Theme has type.

- void **clear_constant** (*String name, String type*)

Clears theme constant at name if Theme has type.

- void **clear_font** (*String name, String type*)

Clears *Font* at name if Theme has type.

- void **clear_icon** (*String name, String type*)

Clears icon at name if Theme has type.

- void **clear_stylebox** (*String name, String type*)

Clears *StyleBox* at name if Theme has type.

- void **copy_default_theme** ()

Sets theme values to a copy of the default theme values.

- *Color* **get_color** (*String name, String type*) const

Returns the *Color* at name if Theme has type.

- *PoolStringArray* **get_color_list** (*String type*) const

Returns all of the *Colors* as a *PoolStringArray* filled with each *Color*'s name, for use in *get_color*, if Theme has type.

- *int* **get_constant** (*String name, String type*) const

Returns the constant at name if Theme has type.

- *PoolStringArray* **get_constant_list** (*String type*) const

Returns all of the constants as a *PoolStringArray* filled with each constant's name, for use in *get_constant*, if Theme has type.

- *Font* **get_font** (*String name, String type*) const

Returns the *Font* at name if Theme has type.

- *PoolStringArray* **get_font_list** (*String type*) const

Returns all of the *Fonts* as a *PoolStringArray* filled with each *Font*'s name, for use in *get_font*, if Theme has type.

- *Texture* **get_icon** (*String name, String type*) const

Returns the icon *Texture* at name if Theme has type.

- *PoolStringArray* **get_icon_list** (*String type*) const

Returns all of the icons as a *PoolStringArray* filled with each *Texture*'s name, for use in *get_icon*, if Theme has type.

- *StyleBox* **get_stylebox** (*String name, String type*) const

Returns the icon *StyleBox* at name if Theme has type.

- *PoolStringArray* **get_stylebox_list** (*String type*) const

Returns all of the *StyleBox*s as a *PoolStringArray* filled with each *StyleBox*'s name, for use in *get_stylebox*, if Theme has type.

- *PoolStringArray* **get_stylebox_types** () const

Returns all of the *StyleBox* types as a *PoolStringArray* filled with each *StyleBox*'s type, for use in *get_stylebox* and/or *get_stylebox_list*, if Theme has type.

- *PoolStringArray* **get_type_list** (*String type*) const

Returns all of the types in type as a *PoolStringArray* for use in any of the *get_** functions, if Theme has type.

- `bool has_color (String name, String type) const`

Returns `true` if `Color` with name is in type.

Returns `false` if Theme does not have type.

- `bool has_constant (String name, String type) const`

Returns `true` if constant with name is in type.

Returns `false` if Theme does not have type.

- `bool has_font (String name, String type) const`

Returns `true` if `Font` with name is in type.

Returns `false` if Theme does not have type.

- `bool has_icon (String name, String type) const`

Returns `true` if icon `Texture` with name is in type.

Returns `false` if Theme does not have type.

- `bool has_stylebox (String name, String type) const`

Returns `true` if `StyleBox` with name is in type.

Returns `false` if Theme does not have type.

- `void set_color (String name, String type, Color color)`

Sets Theme's `Color` to `color` at name in type.

Does nothing if Theme does not have type.

- `void set_constant (String name, String type, int constant)`

Sets Theme's constant to `constant` at name in type.

Does nothing if Theme does not have type.

- `void set_font (String name, String type, Font font)`

Sets Theme's `Font` to `font` at name in type.

Does nothing if Theme does not have type.

- `void set_icon (String name, String type, Texture texture)`

Sets Theme's icon `Texture` to `texture` at name in type.

Does nothing if Theme does not have type.

- `void set_stylebox (String name, String type, StyleBox texture)`

Sets Theme's `StyleBox` to `stylebox` at name in type.

Does nothing if Theme does not have type.

25.398 Thread

Inherits: `Reference < Object`

Category: Core

25.398.1 Brief Description

A unit of execution in a process.

25.398.2 Member Functions

<i>String</i>	<i>get_id () const</i>
<i>bool</i>	<i>is_active () const</i>
<i>int</i>	<i>start (Object instance, String method, Variant userdata=null, int priority=1)</i>
<i>Variant</i>	<i>wait_to_finish ()</i>

25.398.3 Enums

enum **Priority**

- **PRIORITY_LOW = 0**
- **PRIORITY_NORMAL = 1**
- **PRIORITY_HIGH = 2**

25.398.4 Description

A unit of execution in a process. Can run methods on *Objects* simultaneously. The use of synchronization via *Mutex*, *Semaphore* is advised if working with shared objects.

25.398.5 Member Function Description

- *String get_id () const*

Returns the current *Threads* id, uniquely identifying it among all threads.

- *bool is_active () const*

Returns true if this *Thread* is currently active. An active *Thread* cannot start work on a new method but can be joined with *wait_to_finish*.

- *int start (Object instance, String method, Variant userdata=null, int priority=1)*

Starts a new *Thread* that runs “method” on object “instance” with “userdata” passed as an argument. The “priority” of the *Thread* can be changed by passing a *PRIORITY_** enum.

Returns OK on success, or *ERR_CANT_CREATE* on failure.

- *Variant wait_to_finish ()*

Joins the *Thread* and waits for it to finish. Returns what the method called returned.

25.399 TileMap

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.399.1 Brief Description

Node for 2D tile-based maps.

25.399.2 Member Functions

void	<code>clear ()</code>
<i>int</i>	<code>get_cell (int x, int y) const</code>
<i>int</i>	<code>get_cellv (Vector2 position) const</code>
<i>bool</i>	<code>get_collision_layer_bit (int bit) const</code>
<i>bool</i>	<code>get_collision_mask_bit (int bit) const</code>
<i>Array</i>	<code>get_used_cells () const</code>
<i>Array</i>	<code>get_used_cells_by_id (int id) const</code>
<i>Rect2</i>	<code>get_used_rect ()</code>
<i>bool</i>	<code>is_cell_transposed (int x, int y) const</code>
<i>bool</i>	<code>is_cell_x_flipped (int x, int y) const</code>
<i>bool</i>	<code>is_cell_y_flipped (int x, int y) const</code>
<i>Vector2</i>	<code>map_to_world (Vector2 map_position, bool ignore_half_ofs=false) const</code>
void	<code>set_cell (int x, int y, int tile, bool flip_x=false, bool flip_y=false, bool transpose=false, Vector2 autotile_coord=Vector2(0, 0))</code>
void	<code>set_cellv (Vector2 position, int tile, bool flip_x=false, bool flip_y=false, bool transpose=false)</code>
void	<code>set_collision_layer_bit (int bit, bool value)</code>
void	<code>set_collision_mask_bit (int bit, bool value)</code>
void	<code>update_bitmask_area (Vector2 position)</code>
void	<code>update_bitmask_region (Vector2 start=Vector2(0, 0), Vector2 end=Vector2(0, 0))</code>
<i>Vector2</i>	<code>world_to_map (Vector2 world_position) const</code>

25.399.3 Signals

- `settings_changed ()`

Emitted when a tilemap setting has changed.

25.399.4 Member Variables

- `bool cell_clip_uv`
- `Transform2D cell_custom_transform` - The custom *Transform2D* to be applied to the TileMap's cells.
- `HalfOffset cell_half_offset` - Amount to offset alternating tiles. Uses HALF_OFFSET_* constants. Default value: HALF_OFFSET_DISABLED.

- *int cell_quadrant_size* - The TileMap's quadrant size. Optimizes drawing by batching, using chunks of this size. Default value: 16.
- *Vector2 cell_size* - The TileMap's cell size.
- *TileOrigin cell_tile_origin* - Position for tile origin. Uses TILE_ORIGIN_* constants. Default value: TILE_ORIGIN_TOP_LEFT.
- *bool cell_y_sort* - If true the TileMap's children will be drawn in order of their Y coordinate. Default value: false.
- *float collision_bounce* - Bounce value for static body collisions (see collision_use_kinematic). Default value: 0.
- *float collision_friction* - Friction value for static body collisions (see collision_use_kinematic). Default value: 1.
- *int collision_layer* - The collision layer(s) for all colliders in the TileMap.
- *int collision_mask* - The collision mask(s) for all colliders in the TileMap.
- *bool collision_use_kinematic* - If true TileMap collisions will be handled as a kinematic body. If false collisions will be handled as static body. Default value: false.
- *Mode mode* - The TileMap orientation mode. Uses MODE_* constants. Default value: MODE_SQUARE.
- *int occluder_light_mask* - The light mask assigned to all light occluders in the TileMap. The TileSet's light occluders will cast shadows only from Light2D(s) that have the same light mask(s).
- *TileSet tile_set* - The assigned TileSet.

25.399.5 Numeric Constants

- **INVALID_CELL = -1** — Returned when a cell doesn't exist.

25.399.6 Enums

enum HalfOffset

- **HALF_OFFSET_X = 0** — Half offset on the X coordinate.
- **HALF_OFFSET_Y = 1** — Half offset on the Y coordinate.
- **HALF_OFFSET_DISABLED = 2** — Half offset disabled.

enum TileOrigin

- **TILE_ORIGIN_TOP_LEFT = 0** — Tile origin at its top-left corner.
- **TILE_ORIGIN_CENTER = 1** — Tile origin at its center.
- **TILE_ORIGIN_BOTTOM_LEFT = 2**

enum Mode

- **MODE_SQUARE = 0** — Orthogonal orientation mode.
- **MODE_ISOMETRIC = 1** — Isometric orientation mode.
- **MODE_CUSTOM = 2** — Custom orientation mode.

25.399.7 Description

Node for 2D tile-based maps. Tilemaps use a *TileSet* which contain a list of tiles (textures plus optional collision, navigation, and/or occluder shapes) which are used to create grid-based maps.

25.399.8 Member Function Description

- `void clear()`

Clear all cells.

- `int get_cell(int x, int y) const`

Return the tile index of the referenced cell.

- `int get_cellv(Vector2 position) const`

Return the tile index of the cell referenced by a Vector2.

- `bool get_collision_layer_bit(int bit) const`
- `bool get_collision_mask_bit(int bit) const`
- `Array get_used_cells()` const

Return an array of all cells containing a tile from the tileset (i.e. a tile index different from -1).

- `Array get_used_cells_by_id(int id) const`
- `Rect2 get_used_rect()`
- `bool is_cell_transposed(int x, int y) const`

Return whether the referenced cell is transposed, i.e. the X and Y axes are swapped (mirroring with regard to the (1,1) vector).

- `bool is_cell_x_flipped(int x, int y) const`

Return whether the referenced cell is flipped over the X axis.

- `bool is_cell_y_flipped(int x, int y) const`

Return whether the referenced cell is flipped over the Y axis.

- `Vector2 map_to_world(Vector2 map_position, bool ignore_half_ofs=false) const`

Return the absolute world position corresponding to the tilemap (grid-based) coordinates given as an argument.

Optionally, the tilemap's potential half offset can be ignored.

- `void set_cell(int x, int y, int tile, bool flip_x=false, bool flip_y=false, bool transpose=false, Vector2 autotile_coord=Vector2(0, 0))`

Set the tile index for the cell referenced by its grid-based X and Y coordinates.

A tile index of -1 clears the cell.

Optionally, the tile can also be flipped over the X and Y coordinates, transposed, or be given autotile coordinates.

- `void set_cellv(Vector2 position, int tile, bool flip_x=false, bool flip_y=false, bool transpose=false)`

Set the tile index for the cell referenced by a Vector2 of grid-based coordinates.

A tile index of -1 clears the cell.

Optionally, the tile can also be flipped over the X and Y axes or transposed.

- void **set_collision_layer_bit** (int bit, bool value)
- void **set_collision_mask_bit** (int bit, bool value)
- void **update_bitmask_area** (Vector2 position)

Applies autotiling rules to the cell (and its adjacent cells) referenced by its grid-based X and Y coordinates.

- void **update_bitmask_region** (Vector2 start=Vector2(0, 0), Vector2 end=Vector2(0, 0))

Applies autotiling rules to the cells in the given region (specified by grid-based X and Y coordinates).

Calling with invalid (or missing) parameters applies autotiling rules for the entire TileMap.

- Vector2 **world_to_map** (Vector2 world_position) const

Return the tilemap (grid-based) coordinates corresponding to the absolute world position given as an argument.

25.400 TileSet

Inherits: Resource < Reference < Object

Category: Core

25.400.1 Brief Description

Tile library for tilemaps.

25.400.2 Member Functions

<code>Vector2</code>	<code>_forward_subtile_selection (int autotile_id, int bitmask, Object tilemap, Vector2 tile_location) virtual</code>
<code>bool</code>	<code>_is_tile_bound (int drawn_id, int neighbor_id) virtual</code>
<code>int</code>	<code>autotile_get_bitmask_mode (int id) const</code>
<code>void</code>	<code>autotile_set_bitmask_mode (int id, int mode)</code>
<code>void</code>	<code>clear ()</code>
<code>void</code>	<code>create_tile (int id)</code>
<code>int</code>	<code>find_tile_by_name (String name) const</code>
<code>int</code>	<code>get_last_unused_tile_id () const</code>
<code>Array</code>	<code>get_tiles_ids () const</code>
<code>void</code>	<code>remove_tile (int id)</code>
<code>void</code>	<code>tile_add_shape (int id, Shape2D shape, Transform2D shape_transform, bool one_way=false, Vector2 autotile_</code>
<code>OccluderPolygon2D</code>	<code>tile_get_light_occluder (int id) const</code>
<code>ShaderMaterial</code>	<code>tile_get_material (int id) const</code>
<code>String</code>	<code>tile_get_name (int id) const</code>
<code>NavigationPolygon</code>	<code>tile_get_navigation_polygon (int id) const</code>
<code>Vector2</code>	<code>tile_get_navigation_polygon_offset (int id) const</code>
<code>Texture</code>	<code>tile_get_normal_map (int id) const</code>
<code>Vector2</code>	<code>tile_get_occluder_offset (int id) const</code>
<code>Rect2</code>	<code>tile_get_region (int id) const</code>
<code>Shape2D</code>	<code>tile_get_shape (int id, int shape_id) const</code>
<code>int</code>	<code>tile_get_shape_count (int id) const</code>
<code>bool</code>	<code>tile_get_shape_one_way (int id, int shape_id) const</code>

Continued on next page

Table 25.22 – continued from previous page

<code>Transform2D</code>	<code>tile_get_shape_transform (int id, int shape_id) const</code>
<code>Array</code>	<code>tile_get_shapes (int id) const</code>
<code>Texture</code>	<code>tile_get_texture (int id) const</code>
<code>Vector2</code>	<code>tile_get_texture_offset (int id) const</code>
<code>void</code>	<code>tile_set_light_occluder (int id, OccluderPolygon2D light_occluder)</code>
<code>void</code>	<code>tile_set_material (int id, ShaderMaterial material)</code>
<code>void</code>	<code>tile_set_name (int id, String name)</code>
<code>void</code>	<code>tile_set_navigation_polygon (int id, NavigationPolygon navigation_polygon)</code>
<code>void</code>	<code>tile_set_navigation_polygon_offset (int id, Vector2 navigation_polygon_offset)</code>
<code>void</code>	<code>tile_set_normal_map (int id, Texture normal_map)</code>
<code>void</code>	<code>tile_set_occluder_offset (int id, Vector2 occluder_offset)</code>
<code>void</code>	<code>tile_set_region (int id, Rect2 region)</code>
<code>void</code>	<code>tile_set_shape (int id, int shape_id, Shape2D shape)</code>
<code>void</code>	<code>tile_set_shape_one_way (int id, int shape_id, bool one_way)</code>
<code>void</code>	<code>tile_set_shape_transform (int id, int shape_id, Transform2D shape_transform)</code>
<code>void</code>	<code>tile_set_shapes (int id, Array shapes)</code>
<code>void</code>	<code>tile_set_texture (int id, Texture texture)</code>
<code>void</code>	<code>tile_set_texture_offset (int id, Vector2 texture_offset)</code>

25.400.3 Enums

enum **BitmaskMode**

- **BITMASK_2X2 = 0**
- **BITMASK_3X3 = 1**

enum **AutotileBindings**

- **BIND_TOPLEFT = 1**
- **BIND_TOP = 2**
- **BIND_TOPRIGHT = 4**
- **BIND_LEFT = 8**
- **BIND_RIGHT = 32**
- **BIND_BOTTOMLEFT = 64**
- **BIND_BOTTOM = 128**
- **BIND_BOTTOMRIGHT = 256**

25.400.4 Description

A TileSet is a library of tiles for a *TileMap*. It contains a list of tiles, each consisting of a sprite and optional collision shapes.

Tiles are referenced by a unique integer ID.

25.400.5 Member Function Description

- `Vector2 _forward_subtile_selection (int autotile_id, int bitmask, Object tilemap, Vector2 tile_location) virtual`

- *bool _is_tile_bound (int drawn_id, int neighbor_id) virtual*
- *int autotile_get_bitmask_mode (int id) const*
- *void autotile_set_bitmask_mode (int id, int mode)*
- *void clear ()*

Clear all tiles.

- *void create_tile (int id)*

Create a new tile which will be referenced by the given ID.

- *int find_tile_by_name (String name) const*

Find the first tile matching the given name.

- *int get_last_unused_tile_id () const*

Return the ID following the last currently used ID, useful when creating a new tile.

- *Array get_tiles_ids () const*

Return an array of all currently used tile IDs.

- *void remove_tile (int id)*

Remove the tile referenced by the given ID.

- *void tile_add_shape (int id, Shape2D shape, Transform2D shape_transform, bool one_way=false, Vector2 autotile_coord=Vector2(0, 0))*
- *OccluderPolygon2D tile_get_light_occluder (int id) const*

Return the light occluder of the tile.

- *ShaderMaterial tile_get_material (int id) const*

Return the material of the tile.

- *String tile_get_name (int id) const*

Return the name of the tile.

- *NavigationPolygon tile_get_navigation_polygon (int id) const*

Return the navigation polygon of the tile.

- *Vector2 tile_get_navigation_polygon_offset (int id) const*

Return the offset of the tile's navigation polygon.

- *Texture tile_get_normal_map (int id) const*
- *Vector2 tile_get_occluder_offset (int id) const*

Return the offset of the tile's light occluder.

- *Rect2 tile_get_region (int id) const*

Return the tile sub-region in the texture.

- *Shape2D tile_get_shape (int id, int shape_id) const*
- *int tile_get_shape_count (int id) const*
- *bool tile_get_shape_one_way (int id, int shape_id) const*
- *Transform2D tile_get_shape_transform (int id, int shape_id) const*

- *Array* **tile_get_shapes** (*int* id) const

Return the array of shapes of the tile.

- *Texture* **tile_get_texture** (*int* id) const

Return the texture of the tile.

- *Vector2* **tile_get_texture_offset** (*int* id) const

Return the texture offset of the tile.

- **void tile_set_light_occluder** (*int* id, *OccluderPolygon2D* light_occluder)

Set a light occluder for the tile.

- **void tile_set_material** (*int* id, *ShaderMaterial* material)

Set the material of the tile.

- **void tile_set_name** (*int* id, *String* name)

Set the name of the tile, for descriptive purposes.

- **void tile_set_navigation_polygon** (*int* id, *NavigationPolygon* navigation_polygon)

Set a navigation polygon for the tile.

- **void tile_set_navigation_polygon_offset** (*int* id, *Vector2* navigation_polygon_offset)

Set an offset for the tile's navigation polygon.

- **void tile_set_normal_map** (*int* id, *Texture* normal_map)

- **void tile_set_occluder_offset** (*int* id, *Vector2* occluder_offset)

Set an offset for the tile's light occluder.

- **void tile_set_region** (*int* id, *Rect2* region)

Set the tile sub-region in the texture. This is common in texture atlases.

- **void tile_set_shape** (*int* id, *int* shape_id, *Shape2D* shape)

- **void tile_set_shape_one_way** (*int* id, *int* shape_id, *bool* one_way)

- **void tile_set_shape_transform** (*int* id, *int* shape_id, *Transform2D* shape_transform)

- **void tile_set_shapes** (*int* id, *Array* shapes)

Set an array of shapes for the tile, enabling physics to collide with it.

- **void tile_set_texture** (*int* id, *Texture* texture)

Set the texture of the tile.

- **void tile_set_texture_offset** (*int* id, *Vector2* texture_offset)

Set the texture offset of the tile.

25.401 Timer

Inherits: *Node* < *Object*

Category: Core

25.401.1 Brief Description

A countdown timer.

25.401.2 Member Functions

<i>bool</i>	<i>is_stopped () const</i>
<i>void</i>	<i>start ()</i>
<i>void</i>	<i>stop ()</i>

25.401.3 Signals

- **timeout ()**

Emitted when the Timer reaches 0.

25.401.4 Member Variables

- *bool* **autostart** - If `true`, Timer will automatically start when entering the scene tree. Default value: `false`.
- *bool* **one_shot** - If `true`, Timer will stop when reaching 0. If `false`, it will restart. Default value: `false`.
- *bool* **paused** - If `true`, the timer is paused and will not process until it is unpause again, even if *start* is called.
- *TimerProcessMode* **process_mode** - Processing mode. Uses `TIMER_PROCESS_*` constants as value.
- *float* **time_left** - The timer's remaining time in seconds. Returns 0 if the timer is inactive.
- *float* **wait_time** - Wait time in seconds.

25.401.5 Enums

enum TimerProcessMode

- **TIMER_PROCESS_PHYSICS = 0** — Update the Timer during the physics step at each frame (fixed framerate processing).
- **TIMER_PROCESS_IDLE = 1** — Update the Timer during the idle time at each frame.

25.401.6 Description

Counts down a specified interval and emits a signal on reaching 0. Can be set to repeat or “one shot” mode.

25.401.7 Member Function Description

- *bool* **is_stopped () const**

Returns `true` if the timer is stopped.

- *void* **start ()**

Starts the timer. This also resets the remaining time to `wait_time`.

Note: this method will not resume a paused timer. See `set_paused`.

- `void stop()`

Stop (cancel) the Timer.

25.402 ToolButton

Inherits: `Button < BaseButton < Control < CanvasItem < Node < Object`

Category: Core

25.402.1 Brief Description

Flat button helper class.

25.402.2 Description

This is a helper class to generate a flat `Button` (see `Button.set_flat`), creating a `ToolButton` is equivalent to:

```
var btn = Button.new()
btn.set_flat(true)
```

25.403 TouchScreenButton

Inherits: `Node2D < CanvasItem < Node < Object`

Category: Core

25.403.1 Brief Description

Button for touch screen devices.

25.403.2 Member Functions

<code>bool</code>	<code>is_pressed()</code> const
-------------------	---------------------------------

25.403.3 Signals

- `pressed()`

Emitted when the button is pressed (down).

- `released()`

Emitted when the button is released (up).

25.403.4 Member Variables

- *String action* - The button's action. Actions can be handled with *InputEventAction*.
- *BitMap bitmask* - The button's bitmask.
- *Texture normal* - The button's texture for the normal state.
- *bool passby_press* - If true passby presses are enabled.
- *Texture pressed* - The button's texture for the pressed state.
- *Shape2D shape* - The button's shape.
- *bool shape_centered* - If true the button's shape is centered.
- *bool shape_visible* - If true the button's shape is visible.
- *VisibilityMode visibility_mode* - The button's visibility mode. See `VISIBILITY__*` constants.

25.403.5 Enums

enum VisibilityMode

- `VISIBILITY_ALWAYS = 0` — Always visible.
- `VISIBILITY_TOUCHSCREEN_ONLY = 1` — Visible on touch screens only.

25.403.6 Description

Button for touch screen devices. You can set it to be visible on all screens, or only on touch devices.

25.403.7 Member Function Description

- `bool is_pressed () const`

Returns true if this button is currently pressed.

25.404 Transform

Category: Built-In Types

25.404.1 Brief Description

3D Transformation. 3x4 matrix.

25.404.2 Member Functions

<i>Transform</i>	<code>Transform (Vector3 x_axis, Vector3 y_axis, Vector3 z_axis, Vector3 origin)</code>
<i>Transform</i>	<code>Transform (Basis basis, Vector3 origin)</code>
<i>Transform</i>	<code>Transform (Transform2D from)</code>
<i>Transform</i>	<code>Transform (Quat from)</code>
<i>Transform</i>	<code>Transform (Basis from)</code>
<i>Transform</i>	<code>affine_inverse ()</code>
<i>Transform</i>	<code>interpolate_with (Transform transform, float weight)</code>
<i>Transform</i>	<code>inverse ()</code>
<i>Transform</i>	<code>looking_at (Vector3 target, Vector3 up)</code>
<i>Transform</i>	<code>orthonormalized ()</code>
<i>Transform</i>	<code>rotated (Vector3 axis, float phi)</code>
<i>Transform</i>	<code>scaled (Vector3 scale)</code>
<i>Transform</i>	<code>translated (Vector3 ofs)</code>
var	<code>xform (var v)</code>
var	<code>xform_inv (var v)</code>

25.404.3 Member Variables

- **Basis basis** - The basis is a matrix containing 3 *Vector3* as its columns: X axis, Y axis, and Z axis. These vectors can be interpreted as the basis vectors of local coordinate system traveling with the object.
- **Vector3 origin** - The translation offset of the transform.

25.404.4 Description

Represents one or many transformations in 3D space such as translation, rotation, or scaling. It consists of a *Basis* “basis” and an *Vector3* “origin”. It is similar to a 3x4 matrix.

25.404.5 Member Function Description

- *Transform* **Transform** (*Vector3* x_axis, *Vector3* y_axis, *Vector3* z_axis, *Vector3* origin)

Constructs the Transform from four *Vector3*. Each axis corresponds to local basis vectors (some of which may be scaled).

- *Transform* **Transform** (*Basis* basis, *Vector3* origin)

Constructs the Transform from a *Basis* and *Vector3*.

- *Transform* **Transform** (*Transform2D* from)

Constructs the Transform from a *Transform2D*.

- *Transform* **Transform** (*Quat* from)

Constructs the Transform from a *Quat*. The origin will be *Vector3*(0, 0, 0).

- *Transform* **Transform** (*Basis* from)

Constructs the Transform from a *Basis*. The origin will be *Vector3*(0, 0, 0).

- *Transform* **affine_inverse** ()

Returns the inverse of the transform, under the assumption that the transformation is composed of rotation, scaling and translation.

- *Transform interpolate_with (Transform transform, float weight)*

Interpolates the transform to other Transform by weight amount (0-1).

- *Transform inverse ()*

Returns the inverse of the transform, under the assumption that the transformation is composed of rotation and translation (no scaling, use affine_inverse for transforms with scaling).

- *Transform looking_at (Vector3 target, Vector3 up)*

Rotate the transform around the up vector to face the target.

- *Transform orthonormalized ()*

Returns the transform with the basis orthogonal (90 degrees), and normalized axis vectors.

- *Transform rotated (Vector3 axis, float phi)*

Rotates the transform around given axis by phi. The axis must be a normalized vector.

- *Transform scaled (Vector3 scale)*

Scales the transform by the specified 3D scaling factors.

- *Transform translated (Vector3 ofs)*

Translates the transform by the specified offset.

- *var xform (var v)*

Transforms the given vector “v” by this transform.

- *var xform_inv (var v)*

Inverse-transforms the given vector “v” by this transform.

25.405 Transform2D

Category: Built-In Types

25.405.1 Brief Description

2D Transformation. 3x2 matrix.

25.405.2 Member Functions

<i>Transform2D</i>	<code>Transform2D (Transform from)</code>
<i>Transform2D</i>	<code>Transform2D (Vector2 x_axis, Vector2 y_axis, Vector2 origin)</code>
<i>Transform2D</i>	<code>Transform2D (float rotation, Vector2 position)</code>
<i>Transform2D</i>	<code>affine_inverse ()</code>
<i>Transform2D</i>	<code>basis_xform (var v)</code>
<i>Transform2D</i>	<code>basis_xform_inv (var v)</code>
<i>Vector2</i>	<code>get_origin ()</code>
<i>float</i>	<code>get_rotation ()</code>
<i>Vector2</i>	<code>get_scale ()</code>
<i>Transform2D</i>	<code>interpolate_with (Transform2D transform, float weight)</code>
<i>Transform2D</i>	<code>inverse ()</code>
<i>Transform2D</i>	<code>orthonormalized ()</code>
<i>Transform2D</i>	<code>rotated (float phi)</code>
<i>Transform2D</i>	<code>scaled (Vector2 scale)</code>
<i>Transform2D</i>	<code>translated (Vector2 offset)</code>
<i>Transform2D</i>	<code>xform (var v)</code>
<i>Transform2D</i>	<code>xform_inv (var v)</code>

25.405.3 Member Variables

- *Vector2 origin* - The transform's translation offset.
- *Vector2 x* - The X axis of 2x2 basis matrix containing 2 *Vector2*s as its columns: X axis and Y axis. These vectors can be interpreted as the basis vectors of local coordinate system traveling with the object.
- *Vector2 y* - The Y axis of 2x2 basis matrix containing 2 *Vector2*s as its columns: X axis and Y axis. These vectors can be interpreted as the basis vectors of local coordinate system traveling with the object.

25.405.4 Description

Represents one or many transformations in 2D space such as translation, rotation, or scaling. It consists of a two *Vector2* x, y and *Vector2* "origin". It is similar to a 3x2 matrix.

25.405.5 Member Function Description

- **Transform2D Transform2D (Transform from)**

Constructs the transform from a 3D *Transform*.

- **Transform2D Transform2D (Vector2 x_axis, Vector2 y_axis, Vector2 origin)**

Constructs the transform from 3 *Vector2*s representing x, y, and origin.

- **Transform2D Transform2D (float rotation, Vector2 position)**

Constructs the transform from a given angle (in radians) and position.

- **Transform2D affine_inverse ()**

Returns the inverse of the matrix.

- *Transform2D basis_xform (var v)*

Transforms the given vector by this transform's basis (no translation).

- *Transform2D basis_xform_inv (var v)*

Inverse-transforms the given vector by this transform's basis (no translation).

- *Vector2 get_origin ()*

Returns the transform's origin (translation).

- *float get_rotation ()*

Returns the transform's rotation (in radians).

- *Vector2 get_scale ()*

Returns the scale.

- *Transform2D interpolate_with (Transform2D transform, float weight)*

Returns a transform interpolated between this transform and another by a given weight (0-1).

- *Transform2D inverse ()*

Returns the inverse of the transform, under the assumption that the transformation is composed of rotation and translation (no scaling, use affine_inverse for transforms with scaling).

- *Transform2D orthonormalized ()*

Returns the transform with the basis orthogonal (90 degrees), and normalized axis vectors.

- *Transform2D rotated (float phi)*

Rotates the transform by the given angle (in radians).

- *Transform2D scaled (Vector2 scale)*

Scales the transform by the given factor.

- *Transform2D translated (Vector2 offset)*

Translates the transform by the given offset.

- *Transform2D xform (var v)*

Transforms the given vector "v" by this transform.

- *Transform2D xform_inv (var v)*

Inverse-transforms the given vector "v" by this transform.

25.406 Translation

Inherits: Resource < Reference < Object

Inherited By: PHashTranslation

Category: Core

25.406.1 Brief Description

Language Translation.

25.406.2 Member Functions

void	<i>add_message (String src_message, String xlated_message)</i>
void	<i>erase_message (String src_message)</i>
<i>String</i>	<i>get_message (String src_message) const</i>
<i>int</i>	<i>get_message_count () const</i>
<i>PoolStringArray</i>	<i>get_message_list () const</i>

25.406.3 Member Variables

- *String locale*

25.406.4 Description

Translations are resources that can be loaded/unloaded on demand. They map a string to another string.

25.406.5 Member Function Description

- void **add_message** (*String src_message, String xlated_message*)

Add a message for translation.

- void **erase_message** (*String src_message*)

Erase a message.

- *String get_message (String src_message) const*

Return a message for translation.

- *int get_message_count () const*

- *PoolStringArray get_message_list () const*

Return all the messages (keys).

25.407 TranslationServer

Inherits: *Object*

Category: Core

25.407.1 Brief Description

Server that manages all translations. Translations can be set to it and removed from it.

25.407.2 Member Functions

void	<i>add_translation</i> (<i>Translation</i> translation)
void	<i>clear</i> ()
<i>String</i>	<i>get_locale</i> () const
<i>String</i>	<i>get_locale_name</i> (<i>String</i> locale) const
void	<i>remove_translation</i> (<i>Translation</i> translation)
void	<i>set_locale</i> (<i>String</i> locale)
<i>String</i>	<i>translate</i> (<i>String</i> message) const

25.407.3 Member Function Description

- void **add_translation** (*Translation* translation)
- void **clear** ()
- *String* **get_locale** () const
- *String* **get_locale_name** (*String* locale) const
- void **remove_translation** (*Translation* translation)
- void **set_locale** (*String* locale)
- *String* **translate** (*String* message) const

25.408 Tree

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

25.408.1 Brief Description

Control to show a tree of items.

25.408.2 Member Functions

<code>bool</code>	<code>are_column_titles_visible () const</code>
<code>void</code>	<code>clear ()</code>
<code>Object</code>	<code>create_item (Object parent=null, int idx=-1)</code>
<code>void</code>	<code>ensure_cursor_is_visible ()</code>
<code>int</code>	<code>get_column_at_position (Vector2 position) const</code>
<code>String</code>	<code>get_column_title (int column) const</code>
<code>int</code>	<code>get_column_width (int column) const</code>
<code>Rect2</code>	<code>get_custom_popup_rect () const</code>
<code>int</code>	<code>get_drop_section_at_position (Vector2 position) const</code>
<code>TreeItem</code>	<code>get_edited () const</code>
<code>int</code>	<code>get_edited_column () const</code>
<code>Rect2</code>	<code>get_item_area_rect (Object item, int column=-1) const</code>
<code>TreeItem</code>	<code>get_item_at_position (Vector2 position) const</code>
<code>TreeItem</code>	<code>get_next_selected (Object from)</code>
<code>int</code>	<code>get_pressed_button () const</code>
<code>TreeItem</code>	<code>get_root ()</code>
<code>Vector2</code>	<code>get_scroll () const</code>
<code>TreeItem</code>	<code>get_selected () const</code>
<code>int</code>	<code>get_selected_column () const</code>
<code>void</code>	<code>set_column_expand (int column, bool expand)</code>
<code>void</code>	<code>set_column_min_width (int column, int min_width)</code>
<code>void</code>	<code>set_column_title (int column, String title)</code>
<code>void</code>	<code>set_column_titles_visible (bool visible)</code>

25.408.3 Signals

- **button_pressed** (*Object* item, *int* column, *int* id)

Emitted when a button on the tree was pressed (see *TreeItem.add_button*).

- **cell_selected** ()

Emitted when a cell is selected.

- **column_title_pressed** (*int* column)

Emitted when a column's title is pressed.

- **custom_popup_edited** (*bool* arrow_clicked)

Emitted when a cell with the CELL_MODE_CUSTOM is clicked to be edited.

- **empty_tree_rmb_selected** (*Vector2* position)

Emitted when the right mouse button is pressed if RMB selection is active and the tree is empty.

- **item_activated** ()

Emitted when an item is activated (double-clicked).

- **item_collapsed** (*Object* item)

Emitted when an item is collapsed by a click on the folding arrow.

- **item_custom_button_pressed** ()

- **item_double_clicked** ()

Emitted when an item is double clicked.

- **item_edited ()**

Emitted when an item is edited.

- **item_rmb_edited ()**

Emitted when an item is edited using the right mouse button.

- **item_rmb_selected (Vector2 position)**

Emitted when an item is selected with right mouse button.

- **item_selected ()**

Emitted when an item is selected with right mouse button.

- **multi_selected (Object item, int column, bool selected)**

Emitted instead of `item_selected` when `select_mode` is `SELECT_MULTI`.

- **nothing_selected ()**

25.408.4 Member Variables

- **bool allow_reselect** - If `true` the currently selected cell may be selected again.
- **bool allow_rmb_select** - If `true` a right mouse button click can select items.
- **int columns** - The amount of columns.
- **int drop_mode_flags** - The drop mode as an OR combination of flags. See `DROP_MODE__*` constants.
- **bool hide_folding** - If `true` the folding arrow is hidden.
- **bool hide_root** - If `true` the tree's root is hidden.
- **SelectMode select_mode** - Allow single or multiple selection. See the `SELECT__*` constants.

25.408.5 Enums

enum DropModeFlags

- **DROP_MODE_DISABLED = 0**
- **DROP_MODE_ON_ITEM = 1**
- **DROP_MODE_INBETWEEN = 2**

enum SelectMode

- **SELECT_SINGLE = 0** — Allow selection of a single item at a time.
- **SELECT_ROW = 1**
- **SELECT_MULTI = 2** — Allow selection of multiple items at the same time.

25.408.6 Description

This shows a tree of items that can be selected, expanded and collapsed. The tree can have multiple columns with custom controls like text editing, buttons and popups. It can be useful for structured displays and interactions.

Trees are built via code, using *TreeItem* objects to create the structure. They have a single root but multiple roots can be simulated if a dummy hidden root is added.

```
func ready() :
    var tree = Tree.new()
    var root = tree.create_item()
    tree.set_hide_root(true)
    var child1 = tree.create_item(root)
    var child2 = tree.create_item(root)
    var subchild1 = tree.create_item(child1)
    subchild1.set_text(0, "Subchild1")
```

25.408.7 Member Function Description

- *bool are_column_titles_visible () const*

Returns `true` if the column titles are being shown.

- *void clear ()*

Clears the tree. This removes all items.

- *Object create_item (Object parent=null, int idx=-1)*

Create an item in the tree and add it as the last child of `parent`. If `parent` is not given, it will be added as the root's last child, or it'll be the root itself if the tree is empty.

- *void ensure_cursor_is_visible ()*

Makes the currently selected item visible. This will scroll the tree to make sure the selected item is visible.

- *int get_column_at_position (Vector2 position) const*

Returns the column index under the given point.

- *String get_column_title (int column) const*

Returns the column's title.

- *int get_column_width (int column) const*

Returns the column's width in pixels.

- *Rect2 get_custom_popup_rect () const*

Returns the rectangle for custom popups. Helper to create custom cell controls that display a popup. See `TreeItem.set_cell_mode`.

- *int get_drop_section_at_position (Vector2 position) const*

- *TreeItem get_edited () const*

Returns the currently edited item. This is only available for custom cell mode.

- *int get_edited_column () const*

Returns the column for the currently edited item. This is only available for custom cell mode.

- *Rect2 get_item_area_rect (Object item, int column=-1) const*

Returns the rectangle area for the specified item. If column is specified, only get the position and size of that column, otherwise get the rectangle containing all columns.

- `TreeItem get_item_at_position (Vector2 position) const`

Returns the tree item at the specified position (relative to the tree origin position).

- `TreeItem get_next_selected (Object from)`

Returns the next selected item after the given one.

- `int get_pressed_button () const`

Returns the last pressed button's index.

- `TreeItem get_root ()`

Returns the tree's root item.

- `Vector2 get_scroll () const`

Returns the current scrolling position.

- `TreeItem get_selected () const`

Returns the currently selected item.

- `int get_selected_column () const`

Returns the current selection's column.

- `void set_column_expand (int column, bool expand)`

If true the column will have the “Expand” flag of *Control*.

- `void set_column_min_width (int column, int min_width)`

Set the minimum width of a column.

- `void set_column_title (int column, String title)`

Set the title of a column.

- `void set_column_titles_visible (bool visible)`

If true column titles are visible.

25.409 TreeItem

Inherits: *Object*

Category: Core

25.409.1 Brief Description

Control for a single item inside a *Tree*.

25.409.2 Member Functions

<code>void add_button (int column, Texture button, int button_idx=-1, bool disabled=false, String tooltip="")</code>	
--	--

Continued on next page

Table 25.23 – continued from previous page

void	<code>clear_custom_bg_color (int column)</code>
void	<code>clear_custom_color (int column)</code>
void	<code>deselect (int column)</code>
void	<code>erase_button (int column, int button_idx)</code>
<i>Texture</i>	<code>get_button (int column, int button_idx) const</code>
<i>int</i>	<code>get_button_count (int column) const</code>
<i>int</i>	<code>get_cell_mode (int column) const</code>
<i>TreeItem</i>	<code>get_children ()</code>
<i>Color</i>	<code>get_custom_bg_color (int column) const</code>
<i>bool</i>	<code>get_expand_right (int column) const</code>
<i>Texture</i>	<code>get_icon (int column) const</code>
<i>int</i>	<code>get_icon_max_width (int column) const</code>
<i>Rect2</i>	<code>get_icon_region (int column) const</code>
<i>Variant</i>	<code>get_metadata (int column) const</code>
<i>TreeItem</i>	<code>get_next ()</code>
<i>TreeItem</i>	<code>get_next_visible ()</code>
<i>TreeItem</i>	<code>get_parent ()</code>
<i>TreeItem</i>	<code>get_prev ()</code>
<i>TreeItem</i>	<code>get_prev_visible ()</code>
<i>float</i>	<code>get_range (int column) const</code>
<i>Dictionary</i>	<code>get_range_config (int column)</code>
<i>String</i>	<code>get_text (int column) const</code>
<i>int</i>	<code>get_text_align (int column) const</code>
<i>String</i>	<code>get_tooltip (int column) const</code>
<i>bool</i>	<code>is_button_disabled (int column, int button_idx) const</code>
<i>bool</i>	<code>is_checked (int column) const</code>
<i>bool</i>	<code>is_custom_set_as_button (int column) const</code>
<i>bool</i>	<code>is_editable (int column)</code>
<i>bool</i>	<code>is_selectable (int column) const</code>
<i>bool</i>	<code>is_selected (int column)</code>
void	<code>move_to_bottom ()</code>
void	<code>move_to_top ()</code>
void	<code>remove_child (Object child)</code>
void	<code>select (int column)</code>
void	<code>set_button (int column, int button_idx, Texture button)</code>
void	<code>set_cell_mode (int column, int mode)</code>
void	<code>set_checked (int column, bool checked)</code>
void	<code>set_custom_as_button (int column, bool enable)</code>
void	<code>set_custom_bg_color (int column, Color color, bool just_outline=false)</code>
void	<code>set_custom_color (int column, Color color)</code>
void	<code>set_custom_draw (int column, Object object, String callback)</code>
void	<code>set_editable (int column, bool enabled)</code>
void	<code>set_expand_right (int column, bool enable)</code>
void	<code>set_icon (int column, Texture texture)</code>
void	<code>set_icon_max_width (int column, int width)</code>
void	<code>set_icon_region (int column, Rect2 region)</code>
void	<code>set_metadata (int column, Variant meta)</code>
void	<code>set_range (int column, float value)</code>
void	<code>set_range_config (int column, float min, float max, float step, bool expr=false)</code>
void	<code>set_selectable (int column, bool selectable)</code>

Continued on next page

Table 25.23 – continued from previous page

void	<code>set_text (int column, String text)</code>
void	<code>set_text_align (int column, int text_align)</code>
void	<code>set_tooltip (int column, String tooltip)</code>

25.409.3 Member Variables

- `bool collapsed` - If `true` the TreeItem is collapsed.
- `int custom_minimum_height` - The custom minimum height.
- `bool disable_folding` - If `true` folding is disabled for this TreeItem.

25.409.4 Enums

enum TreeCellMode

- **CELL_MODE_STRING = 0** — Cell contains a string.
- **CELL_MODE_CHECK = 1** — Cell can be checked.
- **CELL_MODE_RANGE = 2** — Cell contains a range.
- **CELL_MODE_RANGE_EXPRESSION = 3** — Cell contains a range expression.
- **CELL_MODE_ICON = 4** — Cell contains an icon.
- **CELL_MODE_CUSTOM = 5**

enum TextAlign

- **ALIGN_LEFT = 0** — Align text to the left. See `set_text_align()`.
- **ALIGN_CENTER = 1** — Center text. See `set_text_align()`.
- **ALIGN_RIGHT = 2** — Align text to the right. See `set_text_align()`.

25.409.5 Description

Control for a single item inside a *Tree*. May have child TreeItems and be styled as well as contain buttons.

25.409.6 Member Function Description

- `void add_button (int column, Texture button, int button_idx=-1, bool disabled=false, String tooltip="")`

Adds a button with *Texture* `button` at column `column`. The `button_idx` index is used to identify the button when calling other methods. If not specified, the next available index is used, which may be retrieved by calling `get_button_count()` immediately after this method. Optionally, the button can be disabled and have a tooltip.

- `void clear_custom_bg_color (int column)`

Resets the background color for the given column to default.

- `void clear_custom_color (int column)`

Resets the color for the given column to default.

- `void deselect (int column)`

Deselects the given column.

- `void erase_button (int column, int button_idx)`

Removes the button at index `button_idx` in column `column`.

- `Texture get_button (int column, int button_idx) const`

Returns the `Texture` of the button at index `button_idx` in column `column`.

- `int get_button_count (int column) const`

Returns the number of buttons in column `column`. May be used to get the most recently added button's index, if no index was specified.

- `int get_cell_mode (int column) const`

Returns the column's cell mode. See `CELL_MODE_*` constants.

- `TreeItem get_children ()`

Returns the TreeItem's child items.

- `Color get_custom_bg_color (int column) const`

Returns the custom background color of column `column`.

- `bool get_expand_right (int column) const`

Returns `true` if `expand_right` is set.

- `Texture get_icon (int column) const`

Returns the given column's icon `Texture`. Error if no icon is set.

- `int get_icon_max_width (int column) const`

Returns the column's icon's maximum width.

- `Rect2 get_icon_region (int column) const`

Returns the icon `Texture` region as `Rect2`.

- `Variant get_metadata (int column) const`

- `TreeItem get_next ()`

Returns the next TreeItem in the tree.

- `TreeItem get_next_visible ()`

Returns the next visible TreeItem in the tree.

- `TreeItem get_parent ()`

Returns the parent TreeItem.

- `TreeItem get_prev ()`

Returns the previous TreeItem in the tree.

- `TreeItem get_prev_visible ()`

Returns the previous visible TreeItem in the tree.

- `float get_range (int column) const`

- *Dictionary* **get_range_config** (*int column*)
- *String* **get_text** (*int column*) const

Returns the given column's text.

- *int* **get_text_align** (*int column*) const

Returns the given column's text alignment.

- *String* **get_tooltip** (*int column*) const

Returns the given column's tooltip.

- *bool* **is_button_disabled** (*int column, int button_idx*) const

Returns `true` if the button at index `button_idx` for the given column is disabled.

- *bool* **is_checked** (*int column*) const

Returns `true` if the given column is checked.

- *bool* **is_custom_set_as_button** (*int column*) const
- *bool* **is_editable** (*int column*)

Returns `true` if column `column` is editable.

- *bool* **is_selectable** (*int column*) const

Returns `true` if column `column` is selectable.

- *bool* **is_selected** (*int column*)

Returns `true` if column `column` is selected.

- *void* **move_to_bottom** ()

Moves this TreeItem to the bottom in the *Tree* hierarchy.

- *void* **move_to_top** ()

Moves this TreeItem to the top in the *Tree* hierarchy.

- *void* **remove_child** (*Object child*)

Removes the child TreeItem at index `index`.

- *void* **select** (*int column*)

Selects the column `column`.

- *void* **set_button** (*int column, int button_idx, Texture button*)

Sets the given column's button *Texture* at index `button_idx` to `button`.

- *void* **set_cell_mode** (*int column, int mode*)

Sets the given column's cell mode to `mode`. See `CELL_MODE__*` constants.

- *void* **set_checked** (*int column, bool checked*)

If `true` the column `column` is checked.

- *void* **set_custom_as_button** (*int column, bool enable*)

- *void* **set_custom_bg_color** (*int column, Color color, bool just_outline=false*)

Sets the given column's custom background color and whether to just use it as an outline.

- *void* **set_custom_color** (*int column, Color color*)

Sets the given column's custom color.

- void **set_custom_draw** (*int* column, *Object* object, *String* callback)

Sets the given column's custom draw callback to `callback` method on `object`.

- void **set_editable** (*int* column, *bool* enabled)

If `true` column `column` is editable.

- void **set_expand_right** (*int* column, *bool* enable)

If `true` column `column` is expanded to the right.

- void **set_icon** (*int* column, *Texture* texture)

Sets the given column's icon `Texture`.

- void **set_icon_max_width** (*int* column, *int* width)

Sets the given column's icon's maximum width.

- void **set_icon_region** (*int* column, *Rect2* region)

Sets the given column's icon's texture region.

- void **set_metadata** (*int* column, *Variant* meta)

- void **set_range** (*int* column, *float* value)

- void **set_range_config** (*int* column, *float* min, *float* max, *float* step, *bool* expr=false)

- void **set_selectable** (*int* column, *bool* selectable)

If `true` the given column is selectable.

- void **set_text** (*int* column, *String* text)

- void **set_text_align** (*int* column, *int* text_align)

Sets the given column's text alignment. See `ALIGN__*` constants.

- void **set_tooltip** (*int* column, *String* tooltip)

Sets the given column's tooltip text.

25.410 TriangleMesh

Inherits: *Reference < Object*

Category: Core

25.410.1 Brief Description

25.411 Tween

Inherits: *Node < Object*

Category: Core

25.411.1 Brief Description

Node useful for animations with unknown start and end points.

25.411.2 Member Functions

<code>bool</code>	<code>follow_method (Object object, String method, Variant initial_val, Object target, String target_method, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>bool</code>	<code>follow_property (Object object, NodePath property, Variant initial_val, Object target, NodePath target_property, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>float</code>	<code>get_runtime () const</code>
<code>bool</code>	<code>interpolate_callback (Object object, float duration, String callback, Variant arg1=null, Variant arg2=null, Variant arg3=null, Variant arg4=null, Variant arg5=null)</code>
<code>bool</code>	<code>interpolate_deferred_callback (Object object, float duration, String callback, Variant arg1=null, Variant arg2=null, Variant arg3=null, Variant arg4=null, Variant arg5=null)</code>
<code>bool</code>	<code>interpolate_method (Object object, String method, Variant initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>bool</code>	<code>interpolate_property (Object object, NodePath property, Variant initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>bool</code>	<code>is_active () const</code>
<code>bool</code>	<code>remove (Object object, String key="")</code>
<code>bool</code>	<code>remove_all ()</code>
<code>bool</code>	<code>reset (Object object, String key="")</code>
<code>bool</code>	<code>reset_all ()</code>
<code>bool</code>	<code>resume (Object object, String key="")</code>
<code>bool</code>	<code>resume_all ()</code>
<code>bool</code>	<code>seek (float time)</code>
<code>void</code>	<code>set_active (bool active)</code>
<code>bool</code>	<code>start ()</code>
<code>bool</code>	<code>stop (Object object, String key="")</code>
<code>bool</code>	<code>stop_all ()</code>
<code>bool</code>	<code>targeting_method (Object object, String method, Object initial, String initial_method, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>bool</code>	<code>targeting_property (Object object, NodePath property, Object initial, NodePath initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)</code>
<code>float</code>	<code>tell () const</code>

25.411.3 Signals

- `tween_completed (Object object, NodePath key)`

This signal is emitted when a tween ends.

- `tween_started (Object object, NodePath key)`

This signal is emitted when a tween starts.

- `tween_step (Object object, NodePath key, float elapsed, Object value)`

This signal is emitted each step of the tweening.

25.411.4 Member Variables

- *TweenProcessMode* **playback_process_mode**
- *float playback_speed* - The speed multiplier of the tween. Set it to 1 for normal speed, 2 for two times nromal speed, and 0.5 for half of the normal speed. Setting it to 0 would pause the animation, but you might consider using *set_active* or *stop_all* and *resume_all* for this.
- *bool repeat* - If true, the tween will repeat.

25.411.5 Enums

enum **TweenProcessMode**

- **TWEEN_PROCESS_PHYSICS = 0** — The Tween should use *_physics_process* for timekeeping when this is enabled.
- **TWEEN_PROCESS_IDLE = 1** — The Tween should use *_process* for timekeeping when this is enabled (default).

enum **EaseType**

- **EASE_IN = 0** — Signifies that the interpolation should be focused in the beginning.
- **EASE_OUT = 1** — Signifies that the interpolation should be focused in the end.
- **EASE_IN_OUT = 2** — Signifies that the interpolation should be focused in both ends.
- **EASE_OUT_IN = 3** — Signifies that the interpolation should be focused in both ends, but they should be switched (a bit hard to explain, try it for yourself to be sure).

enum **TransitionType**

- **TRANS_LINEAR = 0** — Means that the animation is interpolated linearly.
- **TRANS_SINE = 1** — Means that the animation is interpolated using a sine wave.
- **TRANS_QUINT = 2** — Means that the animation is interpolated with a quinary (to the power of 5) function.
- **TRANS_QUART = 3** — Means that the animation is interpolated with a quartic (to the power of 4) function.
- **TRANS_QUAD = 4** — Means that the animation is interpolated with a quadratic (to the power of 2) function.
- **TRANS_EXPO = 5** — Means that the animation is interpolated with an exponential (some number to the power of x) function.
- **TRANS_ELASTIC = 6** — Means that the animation is interpolated with elasticity, wiggling around the edges.
- **TRANS_CUBIC = 7** — Means that the animation is interpolated with a cubic (to the power of 3) function.
- **TRANS_CIRC = 8** — Means that the animation is interpolated with a function using square roots.
- **TRANS_BOUNCE = 9** — Means that the animation is interpolated by bouncing at, but never surpassing, the end.
- **TRANS_BACK = 10** — Means that the animation is interpolated backing out at edges.

25.411.6 Description

Node useful for animations with unknown start and end points, procedural animations, making one node follow another, and other simple behavior.

Because it is easy to get it wrong, here is a quick usage example:

```
var tween = get_node("Tween")
tween.interpolate_property(get_node("Node2D_to_move"), "transform/origin", Vector2(0,
→0), Vector2(100,100), 1, Tween.TRANS_LINEAR, Tween.EASE_IN_OUT)
tween.start()
```

Some of the methods of this class require a property name. You can get the property name by hovering over the property in the inspector of the editor.

Many of the methods accept `trans_type` and `ease_type`. The first accepts an `TRANS_*` constant, and refers to the way the timing of the animation is handled (you might want to see <http://easings.net/> for some examples). The second accepts an `EASE_*` constant, and controls the where `trans_type` is applied to the interpolation (in the beginning, the end, or both). If you don't know which transition and easing to pick, you can try different `TRANS_*` constants with `EASE_IN_OUT`, and use the one that looks best.

25.411.7 Member Function Description

- `bool follow_method (Object object, String method, Variant initial_val, Object target, String target_method, float duration, int trans_type, int ease_type, float delay=0)`

`Follow` method of `object` and apply the returned value on `target_method` of `target`, beginning from `initial_val` for `duration` seconds, `delay` later. Methods are animated by calling them with consecutive values.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- `bool follow_property (Object object, NodePath property, Variant initial_val, Object target, NodePath target_property, float duration, int trans_type, int ease_type, float delay=0)`

`Follow` property of `object` and apply it on `target_property` of `target`, beginning from `initial_val` for `duration` seconds, `delay` seconds later. Note that `target:target_property` would equal `object:property` at the end of the tween.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- `float get_runtime () const`

Returns the time needed for all tweens to end in seconds, measured from the start. Thus, if you have two tweens, one ending 10 seconds after the start and the other - 20 seconds, it would return 20 seconds, as by that time all tweens would have finished.

- `bool interpolate_callback (Object object, float duration, String callback, Variant arg1=null, Variant arg2=null, Variant arg3=null, Variant arg4=null, Variant arg5=null)`

Call `callback` of `object` after `duration`. `arg1-arg5` are arguments to be passed to the callback.

- `bool interpolate_deferred_callback (Object object, float duration, String callback, Variant arg1=null, Variant arg2=null, Variant arg3=null, Variant arg4=null, Variant arg5=null)`

Call callback of object after duration on the main thread (similar to *Object.call_deferred*). arg1-arg5 are arguments to be passed to the callback.

- *bool interpolate_method (Object object, String method, Variant initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)*

Animate method of object from `initial_val` to `final_val` for duration seconds, delay seconds later. Methods are animated by calling them with consecutive values.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- *bool interpolate_property (Object object, NodePath property, Variant initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)*

Animate property of object from `initial_val` to `final_val` for duration seconds, delay seconds later.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- *bool is_active () const*

Returns true if any tweens are currently running, and false otherwise. Note that this method doesn't consider tweens that have ended.

- *bool remove (Object object, String key="")*

Stop animating and completely remove a tween, given its object and property/method pair. Passing empty String as key will remove all tweens for given object.

- *bool remove_all ()*

Stop animating and completely remove all tweens.

- *bool reset (Object object, String key="")*

Resets a tween to the initial value (the one given, not the one before the tween), given its object and property/method pair. Passing empty String as key will reset all tweens for given object.

- *bool reset_all ()*

Resets all tweens to their initial values (the ones given, not those before the tween).

- *bool resume (Object object, String key="")*

Continue animating a stopped tween, given its object and property/method pair. Passing empty String as key will resume all tweens for given object.

- *bool resume_all ()*

Continue animating all stopped tweens.

- *bool seek (float time)*

Seek the animation to the given `time` in seconds.

- *void set_active (bool active)*

Activate/deactivate the tween. You can use this for pausing animations, though `stop_all` and `resume_all` might be more fit for this.

- *bool start ()*

Start the tween node. You can define tweens both before and after this.

- *bool stop (Object object, String key="")*

Stop animating a tween, given its object and property/method pair. Passing empty String as key will stop all tweens for given object.

- *bool stop_all ()*

Stop animating all tweens.

- *bool targeting_method (Object object, String method, Object initial, String initial_method, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)*

Animate method of object from the value returned by `initial.initial_method` to `final_val` for duration seconds, `delay` seconds later. Methods are animated by calling them with consecutive values.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- *bool targeting_property (Object object, NodePath property, Object initial, NodePath initial_val, Variant final_val, float duration, int trans_type, int ease_type, float delay=0)*

Animate property of object from the current value of the `initial_val` property of `initial` to `final_val` for duration seconds, `delay` seconds later.

`trans_type` accepts `TRANS_*` constants, and is the way the animation is interpolated, while `ease_type` accepts `EASE_*` constants, and controls the place of the interpolation (the beginning, the end, or both). You can read more about them in the class description.

- *float tell () const*

Returns the current time of the tween.

25.412 UndoRedo

Inherits: *Object*

Category: Core

25.412.1 Brief Description

Helper to manage UndoRedo in the editor or custom tools.

25.412.2 Member Functions

<i>Variant</i>	<code>add_do_method (Object object, String method) vararg</code>
<code>void</code>	<code>add_do_property (Object object, String property, Variant value)</code>
<code>void</code>	<code>add_do_reference (Object object)</code>
<i>Variant</i>	<code>add_undo_method (Object object, String method) vararg</code>
<code>void</code>	<code>add_undo_property (Object object, String property, Variant value)</code>
<code>void</code>	<code>add_undo_reference (Object object)</code>
<code>void</code>	<code>clear_history ()</code>
<code>void</code>	<code>commit_action ()</code>
<code>void</code>	<code>create_action (String name, int merge_mode=0)</code>
<i>String</i>	<code>get_current_action_name () const</code>
<i>int</i>	<code>get_version () const</code>
<code>void</code>	<code>redo ()</code>
<code>void</code>	<code>undo ()</code>

25.412.3 Enums

enum **MergeMode**

- **MERGE_DISABLE = 0**
- **MERGE_ENDS = 1**
- **MERGE_ALL = 2**

25.412.4 Description

Helper to manage UndoRedo in the editor or custom tools. It works by storing calls to functions in both ‘do’ and ‘undo’ lists.

Common behavior is to create an action, then add do/undo calls to functions or property changes, then committing the action.

25.412.5 Member Function Description

- *Variant* **add_do_method** (*Object* object, *String* method) vararg
- `void add_do_property (Object object, String property, Variant value)`

Set a property with a custom value.

- `void add_do_reference (Object object)`

Add a ‘do’ reference that will be erased if the ‘do’ history is lost. This is useful mostly for new nodes created for the ‘do’ call. Do not use for resources.

- *Variant* **add_undo_method** (*Object* object, *String* method) vararg
- `void add_undo_property (Object object, String property, Variant value)`

Undo setting of a property with a custom value.

- `void add_undo_reference (Object object)`

Add an ‘undo’ reference that will be erased if the ‘undo’ history is lost. This is useful mostly for nodes removed with the ‘do’ call (not the ‘undo’ call!).

- void **clear_history ()**

Clear the undo/redo history and associated references.

- void **commit_action ()**

Commit the action. All ‘do’ methods/properties are called/set when this function is called.

- void **create_action (String name, int merge_mode=0)**

Create a new action. After this is called, do all your calls to *add_do_method*, *add_undo_method*, *add_do_property* and *add_undo_property*.

- String **get_current_action_name () const**

Get the name of the current action.

- int **get_version () const**

Get the version, each time a new action is committed, the version number of the UndoRedo is increased automatically.

This is useful mostly to check if something changed from a saved version.

- void **redo ()**

- void **undo ()**

25.413 Variant

Category: Core

25.413.1 Brief Description

The most important data type in Godot.

25.413.2 Description

A Variant takes up only 20 bytes and can store almost any engine datatype inside of it. Variants are rarely used to hold information for long periods of time, instead they are used mainly for communication, editing, serialization and moving data around.

25.414 VBoxContainer

Inherits: *BoxContainer < Container < Control < CanvasItem < Node < Object*

Category: Core

25.414.1 Brief Description

Vertical box container.

25.414.2 Description

Vertical box container. See *BoxContainer*.

25.415 Vector2

Category: Built-In Types

25.415.1 Brief Description

Vector used for 2D Math.

25.415.2 Member Functions

<code>Vector2</code>	<code>Vector2 (float x, float y)</code>
<code>Vector2</code>	<code>abs ()</code>
<code>float</code>	<code>angle ()</code>
<code>float</code>	<code>angle_to (Vector2 to)</code>
<code>float</code>	<code>angle_to_point (Vector2 to)</code>
<code>float</code>	<code>aspect ()</code>
<code>Vector2</code>	<code>bounce (Vector2 n)</code>
<code>Vector2</code>	<code>clamped (float length)</code>
<code>Vector2</code>	<code>cubic_interpolate (Vector2 b, Vector2 pre_a, Vector2 post_b, float t)</code>
<code>float</code>	<code>distance_squared_to (Vector2 to)</code>
<code>float</code>	<code>distance_to (Vector2 to)</code>
<code>float</code>	<code>dot (Vector2 with)</code>
<code>Vector2</code>	<code>floor ()</code>
<code>bool</code>	<code>is_normalized ()</code>
<code>float</code>	<code>length ()</code>
<code>float</code>	<code>length_squared ()</code>
<code>Vector2</code>	<code>linear_interpolate (Vector2 b, float t)</code>
<code>Vector2</code>	<code>normalized ()</code>
<code>Vector2</code>	<code>reflect (Vector2 n)</code>
<code>Vector2</code>	<code>rotated (float phi)</code>
<code>Vector2</code>	<code>slide (Vector2 n)</code>
<code>Vector2</code>	<code>snapped (Vector2 by)</code>
<code>Vector2</code>	<code>tangent ()</code>

25.415.3 Member Variables

- `float x` - X component of the vector.
- `float y` - Y component of the vector.

25.415.4 Description

2-element structure that can be used to represent positions in 2d-space, or any other pair of numeric values.

25.415.5 Member Function Description

- *Vector2 Vector2 (float x, float y)*

Constructs a new Vector2 from the given x and y.

- *Vector2 abs ()*

Returns a new vector with all components in absolute values (i.e. positive).

- *float angle ()*

Returns the result of atan2 when called with the Vector's x and y as parameters (Math::atan2(x,y)).

Be aware that it therefore returns an angle oriented clockwise with regard to the (0, 1) unit vector, and not an angle oriented counter-clockwise with regard to the (1, 0) unit vector (which would be the typical trigonometric representation of the angle when calling Math::atan2(y,x)).

- *float angle_to (Vector2 to)*

Returns the angle in radians between the two vectors.

- *float angle_to_point (Vector2 to)*

Returns the angle in radians between the line connecting the two points and the x coordinate.

- *float aspect ()*

Returns the ratio of X to Y.

- *Vector2 bounce (Vector2 n)*

Bounce returns the vector “bounced off” from the given plane, specified by its normal vector.

- *Vector2 clamped (float length)*

Returns the vector with a maximum length.

- *Vector2 cubic_interpolate (Vector2 b, Vector2 pre_a, Vector2 post_b, float t)*

Cubicly interpolates between this Vector and “b”, using “pre_a” and “post_b” as handles, and returning the result at position “t”. “t” should be a float of 0.0-1.0, a percentage of how far along the interpolation is.

- *float distance_squared_to (Vector2 to)*

Returns the squared distance to vector “b”. Prefer this function over “distance_to” if you need to sort vectors or need the squared distance for some formula.

- *float distance_to (Vector2 to)*

Returns the distance to vector “b”.

- *float dot (Vector2 with)*

Returns the dot product with vector “b”.

- *Vector2 floor ()*

Remove the fractional part of x and y.

- *bool is_normalized ()*

Returns whether the vector is normalized or not.

- *float length ()*

Returns the length of the vector.

- *float length_squared ()*

Returns the squared length of the vector. Prefer this function over “length” if you need to sort vectors or need the squared length for some formula.

- *Vector2 linear_interpolate (Vector2 b, float t)*

Returns the result of the linear interpolation between this vector and “b”, by amount “t”. “t” should be a float of 0.0-1.0, a percentage of how far along the interpolation is.

- *Vector2 normalized ()*

Returns a normalized vector to unit length.

- *Vector2 reflect (Vector2 n)*

Reflects the vector along the given plane, specified by its normal vector.

- *Vector2 rotated (float phi)*

Rotates the vector by “phi” radians.

- *Vector2 slide (Vector2 n)*

Slide returns the component of the vector along the given plane, specified by its normal vector.

- *Vector2 snapped (Vector2 by)*

Snaps the vector to a grid with the given size.

- *Vector2 tangent ()*

Returns a perpendicular vector.

25.416 Vector3

Category: Built-In Types

25.416.1 Brief Description

Vector class, which performs basic 3D vector math operations.

25.416.2 Member Functions

<i>Vector3</i>	<i>Vector3 (float x, float y, float z)</i>
<i>Vector3</i>	<i>abs ()</i>
<i>float</i>	<i>angle_to (Vector3 to)</i>
<i>Vector3</i>	<i>bounce (Vector3 n)</i>
<i>Vector3</i>	<i>ceil ()</i>
<i>Vector3</i>	<i>cross (Vector3 b)</i>
<i>Vector3</i>	<i>cubic_interpolate (Vector3 b, Vector3 pre_a, Vector3 post_b, float t)</i>
<i>float</i>	<i>distance_squared_to (Vector3 b)</i>
<i>float</i>	<i>distance_to (Vector3 b)</i>
<i>float</i>	<i>dot (Vector3 b)</i>
<i>Vector3</i>	<i>floor ()</i>
<i>Vector3</i>	<i>inverse ()</i>
<i>bool</i>	<i>is_normalized ()</i>
<i>float</i>	<i>length ()</i>
<i>float</i>	<i>length_squared ()</i>
<i>Vector3</i>	<i>linear_interpolate (Vector3 b, float t)</i>
<i>int</i>	<i>max_axis ()</i>
<i>int</i>	<i>min_axis ()</i>
<i>Vector3</i>	<i>normalized ()</i>
<i>Basis</i>	<i>outer (Vector3 b)</i>
<i>Vector3</i>	<i>reflect (Vector3 n)</i>
<i>Vector3</i>	<i>rotated (Vector3 axis, float phi)</i>
<i>Vector3</i>	<i>slide (Vector3 n)</i>
<i>Vector3</i>	<i>snapped (float by)</i>
<i>Basis</i>	<i>to_diagonal_matrix ()</i>

25.416.3 Member Variables

- *float x* - X component of the vector.
- *float y* - Y component of the vector.
- *float z* - Z component of the vector.

25.416.4 Numeric Constants

- **AXIS_X = 0** — Enumerated value for the X axis. Returned by functions like `max_axis` or `min_axis`.
- **AXIS_Y = 1** — Enumerated value for the Y axis.
- **AXIS_Z = 2** — Enumerated value for the Z axis.

25.416.5 Description

`Vector3` is one of the core classes of the engine, and includes several built-in helper functions to perform basic vector math operations.

25.416.6 Member Function Description

- `Vector3 Vector3 (float x, float y, float z)`

Returns a Vector3 with the given components.

- `Vector3 abs ()`

Returns a new vector with all components in absolute values (i.e. positive).

- `float angle_to (Vector3 to)`

Returns the vector's minimum angle to the vector `to`.

- `Vector3 bounce (Vector3 n)`

Bounce returns the vector “bounced off” from the given plane, specified by its normal vector.

- `Vector3 ceil ()`

Returns a new vector with all components rounded up.

- `Vector3 cross (Vector3 b)`

Returns the cross product with `b`.

- `Vector3 cubic_interpolate (Vector3 b, Vector3 pre_a, Vector3 post_b, float t)`

Performs a cubic interpolation between vectors `pre_a`, `a`, `b`, `post_b` (`a` is current), by the given amount (`t`). (`t`) should be a float of 0.0-1.0, a percentage of how far along the interpolation is.

- `float distance_squared_to (Vector3 b)`

Returns the squared distance to `b`. Prefer this function over `distance_to` if you need to sort vectors or need the squared distance for some formula.

- `float distance_to (Vector3 b)`

Returns the distance to `b`.

- `float dot (Vector3 b)`

Returns the dot product with `b`.

- `Vector3 floor ()`

Returns a new vector with all components rounded down.

- `Vector3 inverse ()`

Returns the inverse of the vector. This is the same as `Vector3(1.0 / v.x, 1.0 / v.y, 1.0 / v.z)`

- `bool is_normalized ()`

Returns whether the vector is normalized or not.

- `float length ()`

Returns the length of the vector.

- `float length_squared ()`

Returns the length of the vector, squared. Prefer this function over “length” if you need to sort vectors or need the squared length for some formula.

- `Vector3 linear_interpolate (Vector3 b, float t)`

Linearly interpolates the vector to a given one (`b`), by the given amount (`t`). (`t`) should be a float of 0.0-1.0, a percentage of how far along the interpolation is.

- *int max_axis ()*

Returns AXIS_X, AXIS_Y or AXIS_Z depending on which axis is the largest.

- *int min_axis ()*

Returns AXIS_X, AXIS_Y or AXIS_Z depending on which axis is the smallest.

- *Vector3 normalized ()*

Returns a copy of the normalized vector to unit length. This is the same as v / v.length().

- *Basis outer (Vector3 b)*

Returns the outer product with b.

- *Vector3 reflect (Vector3 n)*

Reflects the vector along the given plane, specified by its normal vector.

- *Vector3 rotated (Vector3 axis, float phi)*

Rotates the vector around some axis by phi radians. The axis must be a normalized vector.

- *Vector3 slide (Vector3 n)*

Slide returns the component of the vector along the given plane, specified by its normal vector.

- *Vector3 snapped (float by)*

Returns a copy of the vector, snapped to the lowest neared multiple.

- *Basis to_diagonal_matrix ()*

Returns a diagonal matrix with the vector as main diagonal.

25.417 VehicleBody

Inherits: *RigidBody < PhysicsBody < CollisionObject < Spatial < Node < Object*

Category: Core

25.417.1 Brief Description

25.417.2 Member Variables

- *float brake*
- *float engine_force*
- *float steering*

25.418 VehicleWheel

Inherits: *Spatial < Node < Object*

Category: Core

25.418.1 Brief Description

25.418.2 Member Functions

<i>float</i>	<code>get_skidinfo () const</code>
<i>bool</i>	<code>is_in_contact () const</code>

25.418.3 Member Variables

- *float* `damping_compression`
- *float* `damping_relaxation`
- *float* `suspension_max_force`
- *float* `suspension_stiffness`
- *float* `suspension_travel`
- *bool* `use_as_steering`
- *bool* `use_as_traction`
- *float* `wheel_friction_slip`
- *float* `wheel_radius`
- *float* `wheel_rest_length`
- *float* `wheel_roll_influence`

25.418.4 Member Function Description

- *float* `get_skidinfo () const`
- *bool* `is_in_contact () const`

25.419 VideoPlayer

Inherits: *Control < CanvasItem < Node < Object*

Category: Core

25.419.1 Brief Description

Control to play video files.

25.419.2 Member Functions

<i>String</i>	<code>get_stream_name () const</code>
<i>Texture</i>	<code>get_video_texture ()</code>
<i>bool</i>	<code>is_playing () const</code>
<i>void</i>	<code>play ()</code>
<i>void</i>	<code>stop ()</code>

25.419.3 Member Variables

- *int audio_track*
- *bool autoplay*
- *int buffering_msec* - The amount of milliseconds to store in buffer while playing.
- *String bus*
- *bool expand*
- *bool paused*
- *VideoStream stream*
- *float stream_position* - The current position of the stream, in seconds.
- *float volume* - The volume of the audio track as a linear value.
- *float volume_db*

25.419.4 Description

This control has the ability to play video streams. The only format accepted is the OGV Theora, so any other format must be converted before using in a project.

25.419.5 Member Function Description

- *String get_stream_name () const*

Get the name of the video stream.

- *Texture get_video_texture ()*

Get the current frame of the video as a *Texture*.

- *bool is_playing () const*

Get whether or not the video is playing.

- *void play ()*

Start the video playback.

- *void stop ()*

Stop the video playback.

25.420 VideoStream

Inherits: *Resource < Reference < Object*

Inherited By: *VideoStreamWebm, VideoStreamTheora*

Category: Core

25.420.1 Brief Description

25.421 VideoStreamTheora

Inherits: *VideoStream < Resource < Reference < Object*

Category: Core

25.421.1 Brief Description

25.421.2 Member Functions

<i>String</i>	<code>get_file()</code>
<code>void</code>	<code>set_file(String file)</code>

25.421.3 Member Function Description

- `String get_file()`
- `void set_file(String file)`

25.422 VideoStreamWebm

Inherits: *VideoStream < Resource < Reference < Object*

Category: Core

25.422.1 Brief Description

25.422.2 Member Functions

<i>String</i>	<code>get_file()</code>
<code>void</code>	<code>set_file(String file)</code>

25.422.3 Member Function Description

- `String get_file()`
- `void set_file(String file)`

25.423 Viewport

Inherits: *Node < Object*

Category: Core

25.423.1 Brief Description

Creates a sub-view into the screen.

25.423.2 Member Functions

<i>World</i>	<code>find_world() const</code>
<i>World2D</i>	<code>find_world_2d() const</code>
<i>Camera</i>	<code>get_camera() const</code>
<i>Transform2D</i>	<code>get_final_transform() const</code>
<i>Vector2</i>	<code>get_mouse_position() const</code>
<i>int</i>	<code>get_render_info(int info)</code>
<i>Vector2</i>	<code>get_size_override() const</code>
<i>ViewportTexture</i>	<code>get_texture() const</code>
<i>RID</i>	<code>get_viewport_rid() const</code>
<i>Rect2</i>	<code>get_visible_rect() const</code>
<i>Variant</i>	<code>gui_get_drag_data() const</code>
<i>bool</i>	<code>gui_has_modal_stack() const</code>
<i>void</i>	<code>input(InputEvent local_event)</code>
<i>bool</i>	<code>is_size_override_enabled() const</code>
<i>bool</i>	<code>is_size_override_stretch_enabled() const</code>
<i>void</i>	<code>set_attach_to_screen_rect(Rect2 rect)</code>
<i>void</i>	<code>set_size_override(bool enable, Vector2 size=Vector2(-1, -1), Vector2 margin=Vector2(0, 0))</code>
<i>void</i>	<code>set_size_override_stretch(bool enabled)</code>
<i>void</i>	<code>unhandled_input(InputEvent local_event)</code>
<i>void</i>	<code>update_worlds()</code>
<i>void</i>	<code>warp_mouse(Vector2 to_position)</code>

25.423.3 Signals

- `size_changed()`

Emitted when the size of the viewport is changed, whether by `set_size_override`, resize of window, or some other means.

25.423.4 Member Variables

- `bool arvr`
- `bool audio_listener_enable_2d`

- *bool* **audio_listener_enable_3d**
- *Transform2D* **canvas_transform** - The canvas transform of the viewport, useful for changing the on-screen positions of all child *CanvasItems*. This is relative to the global canvas transform of the viewport.
- *DebugDraw* **debug_draw**
- *bool* **disable_3d**
- *Transform2D* **global_canvas_transform** - The global canvas transform of the viewport. The canvas transform is relative to this.
- *bool* **gui_disable_input**
- *bool* **gui_snap_controls_to_pixels**
- *bool* **hdr**
- **MSAA msaa**
- *bool* **own_world**
- *bool* **physics_object_picking**
- *ClearMode* **render_target_clear_mode**
- *UpdateMode* **render_target_update_mode**
- *bool* **render_target_v_flip**
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_0**
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_1**
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_2**
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_3**
- *int* **shadow_atlas_size**
- *Vector2* **size**
- *bool* **transparent_bg**
- *Usage* **usage**
- **World world**
- **World2D world_2d**

25.423.5 Enums

enum **UpdateMode**

- **UPDATE_DISABLED = 0** — Do not update the render target.
- **UPDATE_ONCE = 1** — Update the render target once, then switch to UPDATE_DISABLED
- **UPDATE_WHEN_VISIBLE = 2** — Update the render target only when it is visible. This is the default value.
- **UPDATE_ALWAYS = 3**

enum **RenderInfo**

- **RENDER_INFO_OBJECTS_IN_FRAME = 0**
- **RENDER_INFO_VERTICES_IN_FRAME = 1**

- **RENDER_INFO_MATERIAL_CHANGES_IN_FRAME = 2**
- **RENDER_INFO_SHADER_CHANGES_IN_FRAME = 3**
- **RENDER_INFO_SURFACE_CHANGES_IN_FRAME = 4**
- **RENDER_INFO_DRAW_CALLS_IN_FRAME = 5**
- **RENDER_INFO_MAX = 6**

enum **MSAA**

- **MSAA_DISABLED = 0**
- **MSAA_2X = 1**
- **MSAA_4X = 2**
- **MSAA_8X = 3**
- **MSAA_16X = 4**

enum **ClearMode**

- **CLEAR_MODE_ALWAYS = 0**
- **CLEAR_MODE_NEVER = 1**
- **CLEAR_MODE_ONLY_NEXT_FRAME = 2**

enum **Usage**

- **USAGE_2D = 0**
- **USAGE_2D_NO_SAMPLING = 1**
- **USAGE_3D = 2**
- **USAGE_3D_NO_EFFECTS = 3**

enum **DebugDraw**

- **DEBUG_DRAW_DISABLED = 0**
- **DEBUG_DRAW_UNSHADEDED = 1**
- **DEBUG_DRAW_OVERDRAW = 2**
- **DEBUG_DRAW_WIREFRAME = 3**

enum **ShadowAtlasQuadrantSubdiv**

- **SHADOW_ATLAS_QUADRANT_SUBDIV_DISABLED = 0**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_1 = 1**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_4 = 2**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_16 = 3**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_64 = 4**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_256 = 5**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_1024 = 6**
- **SHADOW_ATLAS_QUADRANT_SUBDIV_MAX = 7**

25.423.6 Description

A Viewport creates a different view into the screen, or a sub-view inside another viewport. Children 2D Nodes will display on it, and children Camera 3D nodes will render on it too.

Optionally, a viewport can have its own 2D or 3D world, so they don't share what they draw with other viewports.

If a viewport is a child of a *Control*, it will automatically take up its same rect and position, otherwise they must be set manually.

Viewports can also choose to be audio listeners, so they generate positional audio depending on a 2D or 3D camera child of it.

Also, viewports can be assigned to different screens in case the devices have multiple screens.

Finally, viewports can also behave as render targets, in which case they will not be visible unless the associated texture is used to draw.

25.423.7 Member Function Description

- *World* **find_world () const**

Return the 3D world of the viewport, or if no such present, the one of the parent viewport.

- *World2D* **find_world_2d () const**

Return the 2D world of the viewport.

- *Camera* **get_camera () const**

Return the active 3D camera.

- *Transform2D* **get_final_transform () const**

Get the total transform of the viewport.

- *Vector2* **get_mouse_position () const**

Get the mouse position, relative to the viewport.

- *int* **get_render_info (int info)**

- *Vector2* **get_size_override () const**

Get the size override set with *set_size_override*.

- *ViewportTexture* **get_texture () const**

Get the viewport's texture, for use with various objects that you want to texture with the viewport.

- *RID* **get_viewport_rid () const**

Get the viewport RID from the *VisualServer*.

- *Rect2* **get_visible_rect () const**

Return the final, visible rect in global screen coordinates.

- *Variant* **gui_get_drag_data () const**

Returns the drag data from the GUI, that was previously returned by *Control.get_drag_data*.

- *bool* **gui_has_modal_stack () const**

Returns whether there are shown modals on-screen.

- *void* **input (InputEvent local_event)**

- *bool* **is_size_override_enabled** () const

Get the enabled status of the size override set with *set_size_override*.

- *bool* **is_size_override_stretch_enabled** () const

Get the enabled status of the size stretch override set with *set_size_override_stretch*.

- *void* **set_attach_to_screen_rect** (*Rect2* rect)

- *void* **set_size_override** (*bool* enable, *Vector2* size=*Vector2*(-1, -1), *Vector2* margin=*Vector2*(0, 0))

Set the size override of the viewport. If the enable parameter is true, it would use the override, otherwise it would use the default size. If the size parameter is equal to (-1, -1), it won't update the size.

- *void* **set_size_override_stretch** (*bool* enabled)

Set whether the size override affects stretch as well.

- *void* **unhandled_input** (*InputEvent* local_event)

- *void* **update_worlds** ()

Force update of the 2D and 3D worlds.

- *void* **warp_mouse** (*Vector2* to_position)

Warp the mouse to a position, relative to the viewport.

25.424 ViewportContainer

Inherits: Container < Control < CanvasItem < Node < Object

Category: Core

25.424.1 Brief Description

25.424.2 Member Variables

- *bool* **stretch**
- *int* **stretch_shrink**

25.425 ViewportTexture

Inherits: Texture < Resource < Reference < Object

Category: Core

25.425.1 Brief Description

25.425.2 Member Variables

- *NodePath* **viewport_path**

25.426 VisibilityEnabler

Inherits: *VisibilityNotifier < Spatial < Node < Object*

Category: Core

25.426.1 Brief Description

Enable certain nodes only when visible.

25.426.2 Member Variables

- *bool* **freeze_bodies**
- *bool* **pause_animations**

25.426.3 Enums

enum **Enabler**

- **ENABLER_PAUSE_ANIMATIONS = 0** — This enabler will pause *AnimationPlayer* nodes.
- **ENABLER_FREEZE_BODIES = 1** — This enabler will freeze *RigidBody* nodes.
- **ENABLER_MAX = 2**

25.426.4 Description

The *VisibilityEnabler* will disable *RigidBody* and *AnimationPlayer* nodes when they are not visible. It will only affect other nodes within the same scene as the *VisibilityEnabler* itself.

25.427 VisibilityEnabler2D

Inherits: *VisibilityNotifier2D < Node2D < CanvasItem < Node < Object*

Category: Core

25.427.1 Brief Description

Enable certain nodes only when visible.

25.427.2 Member Variables

- `bool freeze_bodies`
- `bool pause_animated_sprites`
- `bool pause_animations`
- `bool pause_particles`
- `bool physics_process_parent`
- `bool process_parent`

25.427.3 Enums

enum **Enabler**

- **ENABLER_PAUSE_ANIMATIONS = 0** — This enabler will pause *AnimationPlayer* nodes.
- **ENABLER_FREEZE_BODIES = 1** — This enabler will freeze *RigidBody2D* nodes.
- **ENABLER_PAUSE_PARTICLES = 2** — This enabler will stop *Particles2D* nodes.
- **ENABLER_PARENT_PROCESS = 3** — This enabler will stop the parent's `_process` function.
- **ENABLER_PARENT_PHYSICS_PROCESS = 4** — This enabler will stop the parent's `_physics_process` function.
- **ENABLER_PAUSE_ANIMATED_SPRITES = 5**
- **ENABLER_MAX = 6**

25.427.4 Description

The *VisibilityEnabler2D* will disable *RigidBody2D*, *AnimationPlayer*, and other nodes when they are not visible. It will only affect other nodes within the same scene as the *VisibilityEnabler2D* itself.

25.428 VisibilityNotifier

Inherits: *Spatial < Node < Object*

Inherited By: *VisibilityEnabler*

Category: Core

25.428.1 Brief Description

Detects when the node is visible on screen.

25.428.2 Member Functions

<i>bool</i>	<i>is_on_screen () const</i>
-------------	------------------------------

25.428.3 Signals

- **camera_entered** (*Object camera*)

Emitted when the VisibilityNotifier enters a *Camera*'s view.

- **camera_exited** (*Object camera*)

Emitted when the VisibilityNotifier exits a *Camera*'s view.

- **screen_entered** ()

Emitted when the VisibilityNotifier enters the screen.

- **screen_exited** ()

Emitted when the VisibilityNotifier exits the screen.

25.428.4 Member Variables

- **AABB aabb** - The VisibilityNotifier's bounding box.

25.428.5 Description

The VisibilityNotifier detects when it is visible on the screen. It also notifies when its bounding rectangle enters or exits the screen or a *Camera*'s view.

25.428.6 Member Function Description

- *bool is_on_screen () const*

If `true` the bounding box is on the screen.

25.429 VisibilityNotifier2D

Inherits: *Node2D < CanvasItem < Node < Object*

Inherited By: *VisibilityEnabler2D*

Category: Core

25.429.1 Brief Description

Detects when the node is visible on screen.

25.429.2 Member Functions

<i>bool</i>	<i>is_on_screen () const</i>
-------------	------------------------------

25.429.3 Signals

- **screen_entered ()**

Emitted when the VisibilityNotifier2D enters the screen.

- **screen_exited ()**

Emitted when the VisibilityNotifier2D exits the screen.

- **viewport_entered (*Object* viewport)**

Emitted when the VisibilityNotifier2D enters a *Viewport*'s view.

- **viewport_exited (*Object* viewport)**

Emitted when the VisibilityNotifier2D exits a *Viewport*'s view.

25.429.4 Member Variables

- *Rect2 rect* - The VisibilityNotifier2D's bounding rectangle.

25.429.5 Description

The VisibilityNotifier2D detects when it is visible on the screen. It also notifies when its bounding rectangle enters or exits the screen or a viewport.

25.429.6 Member Function Description

- *bool is_on_screen () const*

If `true` the bounding rectangle is on the screen.

25.430 VisualInstance

Inherits: *Spatial < Node < Object*

Inherited By: *BakedLightmap, Light, ReflectionProbe, GIProbe, GeometryInstance*

Category: Core

25.430.1 Brief Description

25.430.2 Member Functions

AABB	<code>get_aabb () const</code>
AABB	<code>get_transformed_aabb () const</code>
void	<code>set_base (RID base)</code>

25.430.3 Member Variables

- `int layers` - The render layer(s) this VisualInstance is drawn on.

This object will only be visible for *Cameras* whose cull mask includes the render object this VisualInstance is set to.

25.430.4 Member Function Description

- **AABB get_aabb () const**

Returns the *AABB* (also known as the bounding box) for this VisualInstance.

- **AABB get_transformed_aabb () const**

Returns the transformed *AABB* (also known as the bounding box) for this VisualInstance.

Transformed in this case means the *AABB* plus the position, rotation, and scale of the *Spatial's Transform*

- `void set_base (RID base)`

Sets the base of the VisualInstance, which changes how the engine handles the VisualInstance under the hood.

It is recommended to only use `set_base` if you know what you're doing.

25.431 VisualScript

Inherits: *Script < Resource < Reference < Object*

Category: Core

25.431.1 Brief Description

A script implemented in the Visual Script programming environment.

25.431.2 Member Functions

void	<code>add_custom_signal (String name)</code>
void	<code>add_function (String name)</code>
void	<code>add_node (String func, int id, VisualScriptNode node, Vector2 position=Vector2(0, 0))</code>
void	<code>add_variable (String name, Variant default_value=null, bool export=false)</code>

Continued on next page

Table 25.24 – continued from previous page

void	<code>custom_signal_add_argument (String name, int type, String argname, int index=-1)</code>
int	<code>custom_signal_get_argument_count (String name) const</code>
String	<code>custom_signal_get_argument_name (String name, int argidx) const</code>
int	<code>custom_signal_get_argument_type (String name, int argidx) const</code>
void	<code>custom_signal_remove_argument (String name, int argidx)</code>
void	<code>custom_signal_set_argument_name (String name, int argidx, String argname)</code>
void	<code>custom_signal_set_argument_type (String name, int argidx, int type)</code>
void	<code>custom_signal_swap_argument (String name, int argidx, int withidx)</code>
void	<code>data_connect (String func, int from_node, int from_port, int to_node, int to_port)</code>
void	<code>data_disconnect (String func, int from_node, int from_port, int to_node, int to_port)</code>
int	<code>get_function_node_id (String name) const</code>
Vector2	<code>get_function_scroll (String name) const</code>
VisualScriptNode	<code>get_node (String func, int id) const</code>
Vector2	<code>get_node_position (String func, int id) const</code>
Variant	<code>get_variable_default_value (String name) const</code>
bool	<code>get_variable_export (String name) const</code>
Dictionary	<code>get_variable_info (String name) const</code>
bool	<code>has_custom_signal (String name) const</code>
bool	<code>has_data_connection (String func, int from_node, int from_port, int to_node, int to_port) const</code>
bool	<code>has_function (String name) const</code>
bool	<code>has_node (String func, int id) const</code>
bool	<code>has_sequence_connection (String func, int from_node, int from_output, int to_node) const</code>
bool	<code>has_variable (String name) const</code>
void	<code>remove_custom_signal (String name)</code>
void	<code>remove_function (String name)</code>
void	<code>remove_node (String func, int id)</code>
void	<code>remove_variable (String name)</code>
void	<code>rename_custom_signal (String name, String new_name)</code>
void	<code>rename_function (String name, String new_name)</code>
void	<code>rename_variable (String name, String new_name)</code>
void	<code>sequence_connect (String func, int from_node, int from_output, int to_node)</code>
void	<code>sequence_disconnect (String func, int from_node, int from_output, int to_node)</code>
void	<code>set_function_scroll (String name, Vector2 ofs)</code>
void	<code>set_instance_base_type (String type)</code>
void	<code>set_node_position (String func, int id, Vector2 position)</code>
void	<code>set_variable_default_value (String name, Variant value)</code>
void	<code>set_variable_export (String name, bool enable)</code>
void	<code>set_variable_info (String name, Dictionary value)</code>

25.431.3 Signals

- `node_ports_changed (String function, int id)`

Emitted when the ports of a node are changed.

25.431.4 Description

A script implemented in the Visual Script programming environment. The script extends the functionality of all objects that instance it.

`Object.set_script` extends an existing object, if that object's class matches one of the script's base classes.

You are most likely to use this class via the Visual Script editor or when writing plugins for it.

25.431.5 Member Function Description

- `void add_custom_signal (String name)`

Add a custom signal with the specified name to the VisualScript.

- `void add_function (String name)`

Add a function with the specified name to the VisualScript.

- `void add_node (String func, int id, VisualScriptNode node, Vector2 position=Vector2(0, 0))`

Add a node to a function of the VisualScript.

- `void add_variable (String name, Variant default_value=null, bool export=false)`

Add a variable to the VisualScript, optionally giving it a default value or marking it as exported.

- `void custom_signal_add_argument (String name, int type, String argname, int index=-1)`

Add an argument to a custom signal added with `add_custom_signal`.

- `int custom_signal_get_argument_count (String name) const`

Get the count of a custom signal's arguments.

- `String custom_signal_get_argument_name (String name, int argidx) const`

Get the name of a custom signal's argument.

- `int custom_signal_get_argument_type (String name, int argidx) const`

Get the type of a custom signal's argument.

- `void custom_signal_remove_argument (String name, int argidx)`

Remove a specific custom signal's argument.

- `void custom_signal_set_argument_name (String name, int argidx, String argname)`

Rename a custom signal's argument.

- `void custom_signal_set_argument_type (String name, int argidx, int type)`

Change the type of a custom signal's argument.

- `void custom_signal_swap_argument (String name, int argidx, int withidx)`

Swap two of the arguments of a custom signal.

- `void data_connect (String func, int from_node, int from_port, int to_node, int to_port)`

Connect two data ports. The value of `from_node`'s `from_port` would be fed into `to_node`'s `to_port`.

- `void data_disconnect (String func, int from_node, int from_port, int to_node, int to_port)`

Disconnect two data ports previously connected with `data_connect`.

- `int get_function_node_id (String name) const`

Returns the id of a function's entry point node.

- `Vector2 get_function_scroll (String name) const`

Returns the position of the center of the screen for a given function.

- `VisualScriptNode get_node (String func, int id) const`

Returns a node given its id and its function.

- `Vector2 get_node_position (String func, int id) const`

Returns a node's position in pixels.

- `Variant get_variable_default_value (String name) const`

Returns the default (initial) value of a variable.

- `bool get_variable_export (String name) const`

Returns whether a variable is exported.

- `Dictionary get_variable_info (String name) const`

Returns the info for a given variable as a dictionary. The information includes its name, type, hint and usage.

- `bool has_custom_signal (String name) const`

Returns whether a signal exists with the specified name.

- `bool has_data_connection (String func, int from_node, int from_port, int to_node, int to_port) const`

Returns whether the specified data ports are connected.

- `bool has_function (String name) const`

Returns whether a function exists with the specified name.

- `bool has_node (String func, int id) const`

Returns whether a node exists with the given id.

- `bool has_sequence_connection (String func, int from_node, int from_output, int to_node) const`

Returns whether the specified sequence ports are connected.

- `bool has_variable (String name) const`

Returns whether a variable exists with the specified name.

- `void remove_custom_signal (String name)`

Remove a custom signal with the given name.

- `void remove_function (String name)`

Remove a specific function and its nodes from the script.

- `void remove_node (String func, int id)`

Remove a specific node.

- `void remove_variable (String name)`

Remove a variable with the given name.

- `void rename_custom_signal (String name, String new_name)`

Change the name of a custom signal.

- `void rename_function (String name, String new_name)`

Change the name of a function.

- `void rename_variable (String name, String new_name)`

Change the name of a variable.

- `void sequence_connect (String func, int from_node, int from_output, int to_node)`

Connect two sequence ports. The execution will flow from of `from_node`'s `from_output` into `to_node`.

Unlike `data_connect`, there isn't a `to_port`, since the target node can have only one sequence port.

- void **sequence_disconnect** (*String func, int from_node, int from_output, int to_node*)

Disconnect two sequence ports previously connected with `sequence_connect`.

- void **set_function_scroll** (*String name, Vector2 ofs*)

Position the center of the screen for a function.

- void **set_instance_base_type** (*String type*)

Set the base type of the script.

- void **set_node_position** (*String func, int id, Vector2 position*)

Position a node on the screen.

- void **set_variable_default_value** (*String name, Variant value*)

Change the default (initial) value of a variable.

- void **set_variable_export** (*String name, bool enable*)

Change whether a variable is exported.

- void **set_variable_info** (*String name, Dictionary value*)

Set a variable's info, using the same format as `get_variable_info`.

25.432 VisualScriptBasicTypeConstant

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.432.1 Brief Description

A Visual Script node representing a constant from the base types.

25.432.2 Member Variables

- Type **basic_type** - The type to get the constant from.
- *String constant* - The name of the constant to return.

25.432.3 Description

A Visual Script node representing a constant from base types, such as `Vector3.AXIS_X`.

25.433 VisualScriptBuiltinFunc

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.433.1 Brief Description

A Visual Script node used to call built-in functions.

25.433.2 Member Variables

- **BuiltinFunc function** - The function to be executed.

25.433.3 Enums

enum BuiltinFunc

- **MATH_SIN = 0** — Return the sine of the input.
- **MATH_COS = 1** — Return the cosine of the input.
- **MATH_TAN = 2** — Return the tangent of the input.
- **MATH_SINH = 3** — Return the hyperbolic sine of the input.
- **MATH_COSH = 4** — Return the hyperbolic cosine of the input.
- **MATH_TANH = 5** — Return the hyperbolic tangent of the input.
- **MATH_ASIN = 6** — Return the arc sine of the input.
- **MATH_ACOS = 7** — Return the arc cosine of the input.
- **MATH_ATAN = 8** — Return the arc tangent of the input.
- **MATH_ATAN2 = 9** — Return the arc tangent of the input, using the signs of both parameters to determine the exact angle.
- **MATH_SQRT = 10** — Return the square root of the input.
- **MATH_FMOD = 11** — Return the remainder of one input divided by the other, using floating-point numbers.
- **MATH_FPOSMOD = 12** — Return the positive remainder of one input divided by the other, using floating-point numbers.
- **MATH_FLOOR = 13** — Return the input rounded down.
- **MATH_CEIL = 14** — Return the input rounded up.
- **MATH_ROUND = 15** — Return the input rounded to the nearest integer.
- **MATH_ABS = 16** — Return the absolute value of the input.
- **MATH_SIGN = 17** — Return the sign of the input, turning it into 1, -1, or 0. Useful to determine if the input is positive or negative.
- **MATH_POW = 18** — Return the input raised to a given power.

- **MATH_LOG = 19** — Return the natural logarithm of the input. Note that this is not the typical base-10 logarithm function calculators use.
- **MATH_EXP = 20** — Return e raised to the power of the input. e sometimes called “Euler’s number” is a mathematical constant whose value is approximately 2.71828.
- **MATH_ISNAN = 21** — Return whether the input is NaN (Not a Number) or not. NaN is usually produced by dividing 0 by 0, though other ways exist.
- **MATH_ISINF = 22** — Return whether the input is an infinite floating-point number or not. Infinity is usually produced by dividing a number by 0, though other ways exist.
- **MATH_EASE = 23** — Easing function, based on exponent. 0 is constant, 1 is linear, 0 to 1 is ease-in, 1+ is ease out. Negative values are in-out/out in.
- **MATH_DECIMALS = 24** — Return the number of digit places after the decimal that the first non-zero digit occurs.
- **MATH_STEPIFY = 25** — Return the input snapped to a given step.
- **MATH_LERP = 26** — Return a number linearly interpolated between the first two inputs, based on the third input. Uses the formula $a + (a - b) \star t$.
- **MATH_INVERSE_LERP = 27**
- **MATH_RANGE_LERP = 28**
- **MATH_DECTIME = 29** — Return the result of ‘value’ decreased by ‘step’ * ‘amount’.
- **MATH_RANDOMIZE = 30** — Randomize the seed (or the internal state) of the random number generator. Current implementation reseeds using a number based on time.
- **MATH_RAND = 31** — Return a random 32 bits integer value. To obtain a random value between 0 to N (where N is smaller than $2^{32} - 1$), you can use it with the remainder function.
- **MATH_RANDF = 32** — Return a random floating-point value between 0 and 1. To obtain a random value between 0 to N, you can use it with multiplication.
- **MATH_RANDOM = 33** — Return a random floating-point value between the two inputs.
- **MATH_SEED = 34** — Set the seed for the random number generator.
- **MATH RANDSEED = 35** — Return a random value from the given seed, along with the new seed.
- **MATH_DEG2RAD = 36** — Convert the input from degrees to radians.
- **MATH_RAD2DEG = 37** — Convert the input from radians to degrees.
- **MATH_LINEAR2DB = 38** — Convert the input from linear volume to decibel volume.
- **MATH_DB2LINEAR = 39** — Convert the input from decibel volume to linear volume.
- **MATH_POLAR2CARTESIAN = 40** — Converts a 2D point expressed in the polar coordinate system (a distance from the origin r and an angle θ) to the cartesian coordinate system (x and y axis).
- **MATH_CARTESIAN2POLAR = 41** — Converts a 2D point expressed in the cartesian coordinate system (x and y axis) to the polar coordinate system (a distance from the origin and an angle).
- **MATH_WRAP = 42**
- **MATH_WRAPF = 43**
- **LOGIC_MAX = 44** — Return the greater of the two numbers, also known as their maximum.
- **LOGIC_MIN = 45** — Return the lesser of the two numbers, also known as their minimum.

- **LOGIC_CLAMP = 46** — Return the input clamped inside the given range, ensuring the result is never outside it. Equivalent to $\min(\max(\text{input}, \text{range_low}), \text{range_high})$
- **LOGIC_NEAREST_PO2 = 47** — Return the nearest power of 2 to the input.
- **OBJ_WEAKREF = 48** — Create a *WeakRef* from the input.
- **FUNC_FUNCREF = 49** — Create a *FuncRef* from the input.
- **TYPE_CONVERT = 50** — Convert between types.
- **TYPE_OF = 51** — Return the type of the input as an integer. Check enum Variant.Type for the integers that might be returned.
- **TYPE_EXISTS = 52** — Checks if a type is registered in the *ClassDB*.
- **TEXT_CHAR = 53** — Return a character with the given ascii value.
- **TEXT_STR = 54** — Convert the input to a string.
- **TEXT_PRINT = 55** — Print the given string to the output window.
- **TEXT_PRINTERR = 56** — Print the given string to the standard error output.
- **TEXT_PRINTRAW = 57** — Print the given string to the standard output, without adding a newline.
- **VAR_TO_STR = 58** — Serialize a *Variant* to a string.
- **STR_TO_VAR = 59** — Deserialize a *Variant* from a string serialized using VAR_TO_STR.
- **VAR_TO_BYTES = 60** — Serialize a *Variant* to a *PoolByteArray*.
- **BYTES_TO_VAR = 61** — Deserialize a *Variant* from a *PoolByteArray* serialized using VAR_TO_BYTES.
- **COLORN = 62** — Return the *Color* with the given name and alpha ranging from 0 to 1. Note: names are defined in color_names.inc.
- **FUNC_MAX = 63** — The maximum value the *function* property can have.

25.433.4 Description

A built-in function used inside a *VisualScript*. It is usually a math function or an utility function.

See also @*GDS*cript, for the same functions in the GDScript language.

25.434 VisualScriptClassConstant

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

25.434.1 Brief Description

Gets a constant from a given class.

25.434.2 Member Variables

- *String base_type* - The constant's parent class.
- *String constant* - The constant to return. See the given class for its available constants.

25.434.3 Description

This node returns a constant from a given class, such as @GlobalScope.TYPE_INT. See the given class' documentation for available constants.

Input Ports:

none

Output Ports:

- Data (variant): *value*

25.435 VisualScriptComment

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.435.1 Brief Description

A Visual Script node used to annotate the script.

25.435.2 Member Variables

- *String description* - The text inside the comment node.
- *Vector2 size* - The comment node's size (in pixels).
- *String title* - The comment node's title.

25.435.3 Description

A Visual Script node used to display annotations in the script, so that code may be documented.

Comment nodes can be resized so they encompass a group of nodes.

25.436 VisualScriptCondition

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.436.1 Brief Description

A Visual Script node which branches the flow.

25.436.2 Description

A Visual Script node that checks a *bool* input port. If `true` it will exit via the “true” sequence port. If `false` it will exit via the “false” sequence port. After exiting either, it exits via the “done” port. Sequence ports may be left disconnected.

Input Ports:

- Sequence: `if (cond) is`
- Data (boolean): `cond`

Output Ports:

- Sequence: `true`
- Sequence: `false`
- Sequence: `done`

25.437 VisualScriptConstant

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.437.1 Brief Description

Gets a constant’s value.

25.437.2 Member Variables

- Type **type** - The constant’s type.
- Variant **value** - The constant’s value.

25.437.3 Description

This node returns a constant’s value.

Input Ports:

none

Output Ports:

- Data (variant): `get`

25.438 VisualScriptConstructor

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.438.1 Brief Description

A Visual Script node which calls a base type constructor.

25.438.2 Member Functions

<i>Dictionary</i>	<code>get_constructor () const</code>
<i>int</i>	<code>get_constructor_type () const</code>
<i>void</i>	<code>set_constructor (Dictionary constructor)</code>
<i>void</i>	<code>set_constructor_type (int type)</code>

25.438.3 Description

A Visual Script node which calls a base type constructor. It can be used for type conversion as well.

25.438.4 Member Function Description

- *Dictionary* `get_constructor () const`
- *int* `get_constructor_type () const`
- *void* `set_constructor (Dictionary constructor)`
- *void* `set_constructor_type (int type)`

25.439 VisualScriptCustomNode

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.439.1 Brief Description

A scripted Visual Script node.

25.439.2 Member Functions

<code>String</code>	<code>_get_caption () virtual</code>
<code>String</code>	<code>_get_category () virtual</code>
<code>int</code>	<code>_get_input_value_port_count () virtual</code>
<code>String</code>	<code>_get_input_value_port_name (int idx) virtual</code>
<code>int</code>	<code>_get_input_value_port_type (int idx) virtual</code>
<code>int</code>	<code>_get_output_sequence_port_count () virtual</code>
<code>String</code>	<code>_get_output_sequence_port_text (int idx) virtual</code>
<code>int</code>	<code>_get_output_value_port_count () virtual</code>
<code>String</code>	<code>_get_output_value_port_name (int idx) virtual</code>
<code>int</code>	<code>_get_output_value_port_type (int idx) virtual</code>
<code>String</code>	<code>_get_text () virtual</code>
<code>int</code>	<code>_get_working_memory_size () virtual</code>
<code>bool</code>	<code>_has_input_sequence_port () virtual</code>
<code>Variant</code>	<code>_step (Array inputs, Array outputs, int start_mode, Array working_mem) virtual</code>

25.439.3 Numeric Constants

- **STEP_PUSH_STACK_BIT = 16777216** — Hint used by `_step` to tell that control should return to it when there is no other node left to execute.

This is used by *VisualScriptCondition* to redirect the sequence to the “Done” port after the true/false branch has finished execution. - **STEP_GO_BACK_BIT = 33554432** — Hint used by `_step` to tell that control should return back, either hitting a previous STEP_PUSH_STACK_BIT or exiting the function. - **STEP_NO_ADVANCE_BIT = 67108864** - **STEP_EXIT_FUNCTION_BIT = 134217728** — Hint used by `_step` to tell that control should stop and exit the function. - **STEP_YIELD_BIT = 268435456** — Hint used by `_step` to tell that the function should be yielded.

Using this requires you to have at least one working memory slot, which is used for the *VisualScriptFunctionState*.

25.439.4 Enums

enum StartMode

- **START_MODE_BEGIN_SEQUENCE = 0** — The start mode used the first time when `_step` is called.
- **START_MODE_CONTINUE_SEQUENCE = 1** — The start mode used when `_step` is called after coming back from a STEP_PUSH_STACK_BIT.
- **START_MODE_RESUME_YIELD = 2** — The start mode used when `_step` is called after resuming from STEP_YIELD_BIT.

25.439.5 Description

A custom Visual Script node which can be scripted in powerful ways.

25.439.6 Member Function Description

- `String _get_caption () virtual`

Return the node's title.

- `String _get_category ()` virtual

Return the node's category.

- `int _get_input_value_port_count ()` virtual

Return the count of input value ports.

- `String _get_input_value_port_name (int idx)` virtual

Return the specified input port's name.

- `int _get_input_value_port_type (int idx)` virtual

Return the specified input port's type. See the `TYPE_*` enum in `@GlobalScope`.

- `int _get_output_sequence_port_count ()` virtual

Return the amount of output **sequence** ports.

- `String _get_output_sequence_port_text (int idx)` virtual

Return the specified **sequence** output's name.

- `int _get_output_value_port_count ()` virtual

Return the amount of output value ports.

- `String _get_output_value_port_name (int idx)` virtual

Return the specified output's name.

- `int _get_output_value_port_type (int idx)` virtual

Return the specified output's type. See the `TYPE_*` enum in `@GlobalScope`.

- `String _get_text ()` virtual

Return the custom node's text, which is shown right next to the input **sequence** port (if there is none, on the place that is usually taken by it).

- `int _get_working_memory_size ()` virtual

Return the size of the custom node's working memory. See `_step` for more details.

- `bool _has_input_sequence_port ()` virtual

Return whether the custom node has an input **sequence** port.

- `Variant _step (Array inputs, Array outputs, int start_mode, Array working_mem)` virtual

Execute the custom node's logic, returning the index of the output sequence port to use or a `String` when there is an error.

The `inputs` array contains the values of the input ports.

`outputs` is an array whose indices should be set to the respective outputs.

The `start_mode` is usually `START_MODE_BEGIN_SEQUENCE`, unless you have used the `STEP_*` constants.

`working_mem` is an array which can be used to persist information between runs of the custom node.

When returning, you can mask the returned value with one of the `STEP_*` constants.

25.440 VisualScriptDeconstruct

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.440.1 Brief Description

A Visual Script node which deconstructs a base type instance into its parts.

25.440.2 Member Variables

- Type **type** - The type to deconstruct.

25.440.3 Description

A Visual Script node which deconstructs a base type instance into its parts.

25.441 VisualScriptEditor

Inherits: *Object*

Category: Core

25.441.1 Brief Description

25.441.2 Member Functions

void	<i>add_custom_node (String name, String category, Script script)</i>
void	<i>remove_custom_node (String name, String category)</i>

25.441.3 Signals

- **custom_nodes_updated ()**

Emitted when a custom Visual Script node is added or removed.

25.441.4 Member Function Description

- void **add_custom_node (String name, String category, Script script)**

Add a custom Visual Script node to the editor. It'll be placed under "Custom Nodes" with the `category` as the parameter.

- void **remove_custom_node (String name, String category)**

Remove a custom Visual Script node from the editor. Custom nodes already placed on scripts won't be removed.

25.442 VisualScriptEmitSignal

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.442.1 Brief Description

Emits a specified signal.

25.442.2 Member Variables

- *String signal* - The signal to emit.

25.442.3 Description

Emits a specified signal when it is executed.

Input Ports:

- Sequence: `emit`

Output Ports:

- Sequence

25.443 VisualScriptEngineSingleton

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.443.1 Brief Description

A Visual Script node returning a singleton from `@GlobalScope`

25.443.2 Member Variables

- *String constant* - The singleton's name.

25.443.3 Description

A Visual Script node returning a singleton from `@GlobalScope`

25.444 VisualScriptExpression

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.444.1 Brief Description

25.445 VisualScriptFunction

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.445.1 Brief Description

25.446 VisualScriptFunctionCall

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.446.1 Brief Description

25.446.2 Member Variables

- *String base_script*
- *String base_type*
- Type **basic_type**
- *CallMode call_mode*
- *String function*
- *NodePath node_path*
- *RPCCallMode rpc_call_mode*
- *String singleton*
- *int use_default_args*
- *bool validate*

25.446.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**

- **CALL_MODE_NODE_PATH** = 1
- **CALL_MODE_INSTANCE** = 2
- **CALL_MODE_BASIC_TYPE** = 3
- **CALL_MODE_SINGLETON** = 4

enum **RPCCallMode**

- **RPC_DISABLED** = 0
- **RPC_RELIABLE** = 1
- **RPC_UNRELIABLE** = 2
- **RPC_RELIABLE_TO_ID** = 3
- **RPC_UNRELIABLE_TO_ID** = 4

25.447 VisualScriptFunctionState

Inherits: *Reference < Object*

Category: Core

25.447.1 Brief Description

25.447.2 Member Functions

<i>void</i>	<i>connect_to_signal (Object obj, String signals, Array args)</i>
<i>bool</i>	<i>is_valid () const</i>
<i>Variant</i>	<i>resume (Array args=null)</i>

25.447.3 Member Function Description

- *void connect_to_signal (Object obj, String signals, Array args)*
- *bool is_valid () const*
- *Variant resume (Array args=null)*

25.448 VisualScriptGlobalConstant

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.448.1 Brief Description

25.448.2 Member Variables

- *int* **constant**

25.449 VisualScriptIndexGet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.449.1 Brief Description

25.450 VisualScriptIndexSet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.450.1 Brief Description

25.451 VisualScriptInputAction

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.451.1 Brief Description

25.451.2 Member Variables

- *String* **action**
- *Mode* **mode**

25.451.3 Enums

enum **Mode**

- **MODE_PRESSED = 0**
- **MODE_RELEASED = 1**
- **MODE_JUST_PRESSED = 2**
- **MODE_JUST_RELEASED = 3**

25.452 VisualScriptIterator

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.452.1 Brief Description

Steps through items in a given input.

25.452.2 Description

This node steps through each item in a given input. Input can be any sequence data type, such as an *Array* or *String*. When each item has been processed, execution passes out the `exit` Sequence port.

Input Ports:

- Sequence: `for (elem) in (input)`
- Data (variant): `input`

Output Ports:

- Sequence: `each`
- Sequence: `exit`
- Data (variant): `elem`

25.453 VisualScriptLocalVar

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.453.1 Brief Description

Gets a local variable's value.

25.453.2 Member Variables

- Type **type** - The local variable's type.
- *String var_name* - The local variable's name.

25.453.3 Description

Returns a local variable's value. "Var Name" must be supplied, with an optional type.

Input Ports:

none

Output Ports:

- Data (variant): `get`

25.454 VisualScriptLocalVarSet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.454.1 Brief Description

Changes a local variable's value.

25.454.2 Member Variables

- Type **type** - The local variable's type.
- String **var_name** - The local variable's name.

25.454.3 Description

Changes a local variable's value to the given input. The new value is also provided on an output Data port.

Input Ports:

- Sequence
- Data (variant): `set`

Output Ports:

- Sequence
- Data (variant): `get`

25.455 VisualScriptMathConstant

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.455.1 Brief Description

Commonly used mathematical constants.

25.455.2 Member Variables

- *MathConstant constant* - The math constant.

25.455.3 Enums

enum **MathConstant**

- **MATH_CONSTANT_ONE = 0** — Unity: 1
- **MATH_CONSTANT_PI = 1** — Pi: 3.141593
- **MATH_CONSTANT_HALF_PI = 2** — Pi divided by two: 1.570796
- **MATH_CONSTANT_TAU = 3** — Tau: 6.283185
- **MATH_CONSTANT_E = 4** — Natural log: 2.718282
- **MATH_CONSTANT_SQRT2 = 5** — Square root of two: 1.414214
- **MATH_CONSTANT_INF = 6** — Infinity: inf
- **MATH_CONSTANT_NAN = 7** — Not a number: nan
- **MATH_CONSTANT_MAX = 8**

25.455.4 Description

Provides common math constants, such as Pi or Euler's constant, on an output Data port.

Input Ports:

none

Output Ports:

- Data (variant): get

25.456 VisualScriptNode

Inherits: Resource < Reference < Object

Inherited By: VisualScriptLocalVar, VisualScriptSceneNode, VisualScriptBasicTypeConstant, VisualScriptSequence, VisualScriptVariableSet, VisualScriptSelf, VisualScriptConstant, VisualScriptReturn, VisualScriptSceneTree, VisualScriptIndexSet, VisualScriptResourcePath, VisualScriptPropertyGet, VisualScriptVariableGet, VisualScriptInputAction, VisualScriptEmitSignal, VisualScriptDeconstruct, VisualScriptTypeCast, VisualScriptGlobalConstant,

VisualScriptFunctionCall, VisualScriptYield, VisualScriptSwitch, VisualScriptBuiltinFunc, VisualScriptClassConstant, VisualScriptEngineSingleton, VisualScriptCondition, VisualScriptOperator, VisualScriptIterator, VisualScriptCustomNode, VisualScriptSubCall, VisualScriptYieldSignal, VisualScriptIndexGet, VisualScriptLocalVarSet, VisualScriptWhile, VisualScriptConstructor, VisualScriptMathConstant, VisualScriptComment, VisualScriptExpression, VisualScriptPropertySet, VisualScriptFunction, VisualScriptPreload, VisualScriptSelect

Category: Core

25.456.1 Brief Description

A node which is part of a *VisualScript*.

25.456.2 Member Functions

<i>Variant</i>	<code>get_default_input_value (int port_idx) const</code>
<i>VisualScript</i>	<code>get_visual_script () const</code>
<code>void</code>	<code>ports_changed_notify ()</code>
<code>void</code>	<code>set_default_input_value (int port_idx, Variant value)</code>

25.456.3 Signals

- `ports_changed ()`

Emitted when the available input/output ports are changed.

25.456.4 Description

A node which is part of a *VisualScript*. Not to be confused with *Node*, which is a part of a *SceneTree*.

25.456.5 Member Function Description

- `Variant get_default_input_value (int port_idx) const`

Returns the default value of a given port. The default value is used when nothing is connected to the port.

- `VisualScript get_visual_script () const`

Returns the *VisualScript* instance the node is bound to.

- `void ports_changed_notify ()`

Notify that the node's ports have changed. Usually used in conjunction with *VisualScriptCustomNode*.

- `void set_default_input_value (int port_idx, Variant value)`

Change the default value of a given port.

25.457 VisualScriptOperator

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.457.1 Brief Description

25.457.2 Member Variables

- Operator **operator**
- Type **type**

25.457.3 Description

Input Ports:

- Data (variant): A
- Data (variant): B

Output Ports:

- Data (variant): `result`

25.458 VisualScriptPreload

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.458.1 Brief Description

Creates a new *Resource* or loads one from the filesystem.

25.458.2 Member Variables

- *Resource resource* - The *Resource* to load.

25.458.3 Description

Creates a new *Resource* or loads one from the filesystem.

Input Ports:

none

Output Ports:

- Data (object): `res`

25.459 VisualScriptPropertyGet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.459.1 Brief Description

25.459.2 Member Variables

- *String base_script*
- *String base_type*
- Type **basic_type**
- *String index*
- *NodePath node_path*
- *String property*
- *CallMode set_mode*

25.459.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**
- **CALL_MODE_NODE_PATH = 1**
- **CALL_MODE_INSTANCE = 2**

25.460 VisualScriptPropertySet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.460.1 Brief Description

25.460.2 Member Variables

- *AssignOp assign_op*
- *String base_script*
- *String base_type*
- Type **basic_type**

- *String* **index**
- *NodePath* **node_path**
- *String* **property**
- *CallMode* **set_mode**

25.460.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**
- **CALL_MODE_NODE_PATH = 1**
- **CALL_MODE_INSTANCE = 2**
- **CALL_MODE_BASIC_TYPE = 3**

enum **AssignOp**

- **ASSIGN_OP_NONE = 0**
- **ASSIGN_OP_ADD = 1**
- **ASSIGN_OP_SUB = 2**
- **ASSIGN_OP_MUL = 3**
- **ASSIGN_OP_DIV = 4**
- **ASSIGN_OP_MOD = 5**
- **ASSIGN_OP_SHIFT_LEFT = 6**
- **ASSIGN_OP_SHIFT_RIGHT = 7**
- **ASSIGN_OP_BIT_AND = 8**
- **ASSIGN_OP_BIT_OR = 9**
- **ASSIGN_OP_BIT_XOR = 10**

25.461 VisualScriptResourcePath

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.461.1 Brief Description

25.461.2 Member Variables

- *String* **path**

25.462 VisualScriptReturn

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.462.1 Brief Description

Exits a function and returns an optional value.

25.462.2 Member Variables

- *bool return_enabled* - If true the return input port is available.
- Type **return_type** - The return value's data type.

25.462.3 Description

Ends the execution of a function and returns control to the calling function. Optionally, it can return a *Variant* value.

Input Ports:

- Sequence
- Data (variant): `result` (optional)

Output Ports:

none

25.463 VisualScriptSceneNode

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.463.1 Brief Description

Node reference.

25.463.2 Member Variables

- *NodePath node_path* - The node's path in the scene tree.

25.463.3 Description

A direct reference to a node.

Input Ports:

none

Output Ports:

- Data: node (obj)

25.464 VisualScriptSceneTree

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.464.1 Brief Description

25.465 VisualScriptSelect

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.465.1 Brief Description

Chooses between two input values.

25.465.2 Member Variables

- Type **type** - The input variables' type.

25.465.3 Description

Chooses between two input values based on a Boolean condition.

Input Ports:

- Data (boolean): cond
- Data (variant): a
- Data (variant): b

Output Ports:

- Data (variant): out

25.466 VisualScriptSelf

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.466.1 Brief Description

Outputs a reference to the current instance.

25.466.2 Description

Provides a reference to the node running the visual script.

Input Ports:

none

Output Ports:

- Data (object): `instance`

25.467 VisualScriptSequence

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.467.1 Brief Description

Executes a series of Sequence ports.

25.467.2 Member Variables

- `int steps` - The number of steps in the sequence.

25.467.3 Description

Steps through a series of one or more output Sequence ports. The `current` data port outputs the currently executing item.

Input Ports:

- Sequence: `in` `order`

Output Ports:

- Sequence: `1`
- Sequence: `2 - n` (optional)
- Data (int): `current`

25.468 VisualScriptSubCall

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.468.1 Brief Description

25.468.2 Member Functions

<i>Variant</i>	<code>_subcall (Variant arguments) virtual</code>
----------------	---

25.468.3 Member Function Description

- *Variant* `_subcall (Variant arguments) virtual`

25.469 VisualScriptSwitch

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.469.1 Brief Description

Branches program flow based on a given input's value.

25.469.2 Description

Branches the flow based on an input's value. Use “Case Count” in the Inspector to set the number of branches and each comparison's optional type.

Input Ports:

- Sequence: 'input' is
- Data (variant): =
- Data (variant): = (optional)
- Data (variant): `input`

Output Ports:

- Sequence
- Sequence (optional)
- Sequence: done

25.470 VisualScriptTypeCast

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.470.1 Brief Description

25.470.2 Member Variables

- *String base_script*
- *String base_type*

25.471 VisualScriptVariableGet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.471.1 Brief Description

Gets a variable's value.

25.471.2 Member Variables

- *String var_name* - The variable's name.

25.471.3 Description

Returns a variable's value. “Var Name” must be supplied, with an optional type.

Input Ports:

none

Output Ports:

- Data (variant): *value*

25.472 VisualScriptVariableSet

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.472.1 Brief Description

Changes a variable's value.

25.472.2 Member Variables

- *String var_name* - The variable's name.

25.472.3 Description

Changes a variable's value to the given input.

Input Ports:

- Sequence
- Data (variant): set

Output Ports:

- Sequence

25.473 VisualScriptWhile

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.473.1 Brief Description

Conditional loop.

25.473.2 Description

Loops while a condition is true. Execution continues out the exit Sequence port when the loop terminates.

Input Ports:

- Sequence: while (cond)
- Data (bool): cond

Output Ports:

- Sequence: repeat
- Sequence: exit

25.474 VisualScriptYield

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.474.1 Brief Description

25.474.2 Member Variables

- *YieldMode mode*
- *float wait_time*

25.474.3 Enums

enum **YieldMode**

- **YIELD_FRAME = 1**
- **YIELD_PHYSICS_FRAME = 2**
- **YIELD_WAIT = 3**

25.475 VisualScriptYieldSignal

Inherits: *VisualScriptNode < Resource < Reference < Object*

Category: Core

25.475.1 Brief Description

25.475.2 Member Variables

- *String base_type*
- *CallMode call_mode*
- *NodePath node_path*
- *String signal*

25.475.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**

- CALL_MODE_NODE_PATH = 1
- CALL_MODE_INSTANCE = 2

25.476 VisualServer

Inherits: *Object*

Category: Core

25.476.1 Brief Description

Server for anything visible.

25.476.2 Member Functions

void	<i>black_bars_set_images</i> (<i>RID</i> left, <i>RID</i> top, <i>RID</i> right, <i>RID</i> bottom)
void	<i>black_bars_set_margins</i> (<i>int</i> left, <i>int</i> top, <i>int</i> right, <i>int</i> bottom)
<i>RID</i>	<i>canvas_create</i> ()
void	<i>canvas_item_add_circle</i> (<i>RID</i> item, <i>Vector2</i> pos, <i>float</i> radius, <i>Color</i> color)
void	<i>canvas_item_add_clip_ignore</i> (<i>RID</i> item, <i>bool</i> ignore)
void	<i>canvas_item_add_line</i> (<i>RID</i> item, <i>Vector2</i> from, <i>Vector2</i> to, <i>Color</i> color, <i>float</i> width=1.0, <i>bool</i> antialiased=false)
void	<i>canvas_item_add_mesh</i> (<i>RID</i> item, <i>RID</i> mesh, <i>RID</i> skeleton)
void	<i>canvas_item_add_multimesh</i> (<i>RID</i> item, <i>RID</i> mesh, <i>RID</i> skeleton)
void	<i>canvas_item_add_nine_patch</i> (<i>RID</i> item, <i>Rect2</i> rect, <i>Rect2</i> source, <i>RID</i> texture, <i>Vector2</i> topleft, <i>Vector2</i> bottomright, <i>int</i> x_axis, <i>int</i> y_axis)
void	<i>canvas_item_add_particles</i> (<i>RID</i> item, <i>RID</i> particles, <i>RID</i> texture, <i>RID</i> normal_map, <i>int</i> h_frames, <i>int</i> v_frames)
void	<i>canvas_item_add_polygon</i> (<i>RID</i> item, <i>PoolVector2Array</i> points, <i>PoolColorArray</i> colors, <i>PoolVector2Array</i> uvs=PoolVector2Array::empty())
void	<i>canvas_item_add_polyline</i> (<i>RID</i> item, <i>PoolVector2Array</i> points, <i>PoolColorArray</i> colors, <i>float</i> width=1.0, <i>bool</i> antialiased=false)
void	<i>canvas_item_add_primitive</i> (<i>RID</i> item, <i>PoolVector2Array</i> points, <i>PoolColorArray</i> colors, <i>PoolVector2Array</i> uvs, <i>RID</i> texture, <i>RID</i> normal_map)
void	<i>canvas_item_add_rect</i> (<i>RID</i> item, <i>Rect2</i> rect, <i>Color</i> color)
void	<i>canvas_item_add_set_transform</i> (<i>RID</i> item, <i>Transform2D</i> transform)
void	<i>canvas_item_add_texture_rect</i> (<i>RID</i> item, <i>Rect2</i> rect, <i>RID</i> texture, <i>bool</i> tile=false, <i>Color</i> modulate=Color(1, 1, 1, 1), <i>bool</i> transform=false)
void	<i>canvas_item_add_texture_rect_region</i> (<i>RID</i> item, <i>Rect2</i> rect, <i>RID</i> texture, <i>Rect2</i> src_rect, <i>Color</i> modulate=Color(1, 1, 1, 1), <i>bool</i> transform=false)
void	<i>canvas_item_add_triangle_array</i> (<i>RID</i> item, <i>PoolIntArray</i> indices, <i>PoolVector2Array</i> points, <i>PoolColorArray</i> colors, <i>PoolVector2Array</i> uvs)
void	<i>canvas_item_clear</i> (<i>RID</i> item)
<i>RID</i>	<i>canvas_item_create</i> ()
void	<i>canvas_item_set_clip</i> (<i>RID</i> item, <i>bool</i> clip)
void	<i>canvas_item_set_copy_to_backbuffer</i> (<i>RID</i> item, <i>bool</i> enabled, <i>Rect2</i> rect)
void	<i>canvas_item_set_custom_rect</i> (<i>RID</i> item, <i>bool</i> use_custom_rect, <i>Rect2</i> rect=Rect2(0, 0, 0, 0))
void	<i>canvas_item_set_distance_field_mode</i> (<i>RID</i> item, <i>bool</i> enabled)
void	<i>canvas_item_set_draw_behind_parent</i> (<i>RID</i> item, <i>bool</i> enabled)
void	<i>canvas_item_set_draw_index</i> (<i>RID</i> item, <i>int</i> index)
void	<i>canvas_item_set_light_mask</i> (<i>RID</i> item, <i>int</i> mask)
void	<i>canvas_item_set_material</i> (<i>RID</i> item, <i>RID</i> material)
void	<i>canvas_item_set_modulate</i> (<i>RID</i> item, <i>Color</i> color)
void	<i>canvas_item_set_parent</i> (<i>RID</i> item, <i>RID</i> parent)
void	<i>canvas_item_set_self_modulate</i> (<i>RID</i> item, <i>Color</i> color)
void	<i>canvas_item_set_sort_children_by_y</i> (<i>RID</i> item, <i>bool</i> enabled)
void	<i>canvas_item_set_transform</i> (<i>RID</i> item, <i>Transform2D</i> transform)

Continued on next page

Table 25.25 – continued from previous page

void	<i>canvas_item_set_use_parent_material</i> (<i>RID item</i> , <i>bool enabled</i>)
void	<i>canvas_item_set_visible</i> (<i>RID item</i> , <i>bool visible</i>)
void	<i>canvas_item_set_z_as_relative_to_parent</i> (<i>RID item</i> , <i>bool enabled</i>)
void	<i>canvas_item_set_z_index</i> (<i>RID item</i> , <i>int z_index</i>)
void	<i>canvas_light_attach_to_canvas</i> (<i>RID light</i> , <i>RID canvas</i>)
<i>RID</i>	<i>canvas_light_create</i> ()
void	<i>canvas_light_occluder_attach_to_canvas</i> (<i>RID occluder</i> , <i>RID canvas</i>)
<i>RID</i>	<i>canvas_light_occluder_create</i> ()
void	<i>canvas_light_occluder_set_enabled</i> (<i>RID occluder</i> , <i>bool enabled</i>)
void	<i>canvas_light_occluder_set_light_mask</i> (<i>RID occluder</i> , <i>int mask</i>)
void	<i>canvas_light_occluder_set_polygon</i> (<i>RID occluder</i> , <i>RID polygon</i>)
void	<i>canvas_light_occluder_set_transform</i> (<i>RID occluder</i> , <i>Transform2D transform</i>)
void	<i>canvas_light_set_color</i> (<i>RID light</i> , <i>Color color</i>)
void	<i>canvas_light_set_enabled</i> (<i>RID light</i> , <i>bool enabled</i>)
void	<i>canvas_light_set_energy</i> (<i>RID light</i> , <i>float energy</i>)
void	<i>canvas_light_set_height</i> (<i>RID light</i> , <i>float height</i>)
void	<i>canvas_light_set_item_cull_mask</i> (<i>RID light</i> , <i>int mask</i>)
void	<i>canvas_light_set_item_shadow_cull_mask</i> (<i>RID light</i> , <i>int mask</i>)
void	<i>canvas_light_set_layer_range</i> (<i>RID light</i> , <i>int min_layer</i> , <i>int max_layer</i>)
void	<i>canvas_light_set_mode</i> (<i>RID light</i> , <i>int mode</i>)
void	<i>canvas_light_set_scale</i> (<i>RID light</i> , <i>float scale</i>)
void	<i>canvas_light_set_shadow_buffer_size</i> (<i>RID light</i> , <i>int size</i>)
void	<i>canvas_light_set_shadow_color</i> (<i>RID light</i> , <i>Color color</i>)
void	<i>canvas_light_set_shadow_enabled</i> (<i>RID light</i> , <i>bool enabled</i>)
void	<i>canvas_light_set_shadow_filter</i> (<i>RID light</i> , <i>int filter</i>)
void	<i>canvas_light_set_shadow_gradient_length</i> (<i>RID light</i> , <i>float length</i>)
void	<i>canvas_light_set_shadow_smooth</i> (<i>RID light</i> , <i>float smooth</i>)
void	<i>canvas_light_set_texture</i> (<i>RID light</i> , <i>RID texture</i>)
void	<i>canvas_light_set_texture_offset</i> (<i>RID light</i> , <i>Vector2 offset</i>)
void	<i>canvas_light_set_transform</i> (<i>RID light</i> , <i>Transform2D transform</i>)
void	<i>canvas_light_set_z_range</i> (<i>RID light</i> , <i>int min_z</i> , <i>int max_z</i>)
<i>RID</i>	<i>canvas_occluder_polygon_create</i> ()
void	<i>canvas_occluder_polygon_set_cull_mode</i> (<i>RID occluder_polygon</i> , <i>int mode</i>)
void	<i>canvas_occluder_polygon_set_shape</i> (<i>RID occluder_polygon</i> , <i>PoolVector2Array shape</i> , <i>bool closed</i>)
void	<i>canvas_occluder_polygon_set_shape_as_lines</i> (<i>RID occluder_polygon</i> , <i>PoolVector2Array shape</i>)
void	<i>canvas_set_item_mirroring</i> (<i>RID canvas</i> , <i>RID item</i> , <i>Vector2 mirroring</i>)
void	<i>canvas_set_modulate</i> (<i>RID canvas</i> , <i>Color color</i>)
void	<i>draw</i> (<i>bool swap_buffers=true</i>)
void	<i>finish</i> ()
void	<i>force_draw</i> (<i>bool swap_buffers=true</i>)
void	<i>force_sync</i> ()
void	<i>free_rid</i> (<i>RID rid</i>)
<i>int</i>	<i>get_render_info</i> (<i>int info</i>)
<i>RID</i>	<i>get_test_cube</i> ()
<i>RID</i>	<i>get_test_texture</i> ()
<i>RID</i>	<i>get_white_texture</i> ()
<i>bool</i>	<i>has_changed</i> () const
<i>bool</i>	<i>has_feature</i> (<i>int feature</i>) const
<i>bool</i>	<i>has_os_feature</i> (<i>String feature</i>) const
<i>void</i>	<i>init</i> ()

Continued on next page

Table 25.25 – continued from previous page

<i>RID</i>	<code>make_sphere_mesh (int latitudes, int longitudes, float radius)</code>
<i>RID</i>	<code>material_create ()</code>
<i>Variant</i>	<code>material_get_param (RID material, String parameter) const</code>
<i>RID</i>	<code>material_get_shader (RID shader_material) const</code>
<i>void</i>	<code>material_set_line_width (RID material, float width)</code>
<i>void</i>	<code>material_set_next_pass (RID material, RID next_material)</code>
<i>void</i>	<code>material_set_param (RID material, String parameter, Variant value)</code>
<i>void</i>	<code>material_set_render_priority (RID material, int priority)</code>
<i>void</i>	<code>material_set_shader (RID shader_material, RID shader)</code>
<i>void</i>	<code>mesh_add_surface_from_arrays (RID mesh, int primitive, Array arrays, Array blend_shapes=[], int compress_format=977)</code>
<i>void</i>	<code>mesh_clear (RID mesh)</code>
<i>RID</i>	<code>mesh_create ()</code>
<i>int</i>	<code>mesh_get_blend_shape_count (RID mesh) const</code>
<i>int</i>	<code>mesh_get_blend_shape_mode (RID mesh) const</code>
<i>AABB</i>	<code>mesh_get_custom_aabb (RID mesh) const</code>
<i>int</i>	<code>mesh_get_surface_count (RID mesh) const</code>
<i>void</i>	<code>mesh_remove_surface (RID mesh, int index)</code>
<i>void</i>	<code>mesh_set_blend_shape_count (RID mesh, int amount)</code>
<i>void</i>	<code>mesh_set_blend_shape_mode (RID mesh, int mode)</code>
<i>void</i>	<code>mesh_set_custom_aabb (RID mesh, AABB aabb)</code>
<i>AABB</i>	<code>mesh_surface_get_aabb (RID mesh, int surface) const</code>
<i>PoolByteArray</i>	<code>mesh_surface_get_array (RID mesh, int surface) const</code>
<i>int</i>	<code>mesh_surface_get_array_index_len (RID mesh, int surface) const</code>
<i>int</i>	<code>mesh_surface_get_array_len (RID mesh, int surface) const</code>
<i>Array</i>	<code>mesh_surface_get_arrays (RID mesh, int surface) const</code>
<i>Array</i>	<code>mesh_surface_get_blend_shape_arrays (RID mesh, int surface) const</code>
<i>int</i>	<code>mesh_surface_get_format (RID mesh, int surface) const</code>
<i>PoolByteArray</i>	<code>mesh_surface_get_index_array (RID mesh, int surface) const</code>
<i>RID</i>	<code>mesh_surface_get_material (RID mesh, int surface) const</code>
<i>int</i>	<code>mesh_surface_get_primitive_type (RID mesh, int surface) const</code>
<i>Array</i>	<code>mesh_surface_get_skeleton_aabb (RID mesh, int surface) const</code>
<i>void</i>	<code>mesh_surface_set_material (RID mesh, int surface, RID material)</code>
<i>void</i>	<code>request_frame_drawn_callback (Object where, String method, Variant userdata)</code>
<i>void</i>	<code>set_boot_image (Image image, Color color, bool scale)</code>
<i>void</i>	<code>set_debug_generate_wireframes (bool generate)</code>
<i>void</i>	<code>set_default_clear_color (Color color)</code>
<i>RID</i>	<code>shader_create ()</code>
<i>String</i>	<code>shader_get_code (RID shader) const</code>
<i>RID</i>	<code>shader_get_default_texture_param (RID shader, String name) const</code>
<i>Array</i>	<code>shader_get_param_list (RID shader) const</code>
<i>void</i>	<code>shader_set_code (RID shader, String code)</code>
<i>void</i>	<code>shader_set_default_texture_param (RID shader, String name, RID texture)</code>
<i>RID</i>	<code>sky_create ()</code>
<i>void</i>	<code>sky_set_texture (RID sky, RID cube_map, int radiance_size)</code>
<i>void</i>	<code>sync ()</code>
<i>void</i>	<code>texture_allocate (RID texture, int width, int height, int format, int flags=7)</code>
<i>RID</i>	<code>texture_create ()</code>
<i>RID</i>	<code>texture_create_from_image (Image image, int flags=7)</code>
<i>Array</i>	<code>texture_debug_usage ()</code>
<i>Image</i>	<code>texture_get_data (RID texture, int cube_side=0) const</code>

Continued on next page

Table 25.25 – continued from previous page

<i>int</i>	<code>texture_get_flags (RID texture) const</code>
<i>int</i>	<code>texture_get_format (RID texture) const</code>
<i>int</i>	<code>texture_get_height (RID texture) const</code>
<i>String</i>	<code>texture_get_path (RID texture) const</code>
<i>int</i>	<code>texture_get_texid (RID texture) const</code>
<i>int</i>	<code>texture_get_width (RID texture) const</code>
<i>void</i>	<code>texture_set_data (RID texture, Image image, int cube_side=0)</code>
<i>void</i>	<code>texture_set_flags (RID texture, int flags)</code>
<i>void</i>	<code>texture_set_path (RID texture, String path)</code>
<i>void</i>	<code>texture_set_shrink_all_x2_on_set_data (bool shrink)</code>
<i>void</i>	<code>texture_set_size_override (RID texture, int width, int height)</code>
<i>void</i>	<code>textures_keep_original (bool enable)</code>
<i>void</i>	<code>viewport_attach_camera (RID viewport, RID camera)</code>
<i>void</i>	<code>viewport_attach_canvas (RID viewport, RID canvas)</code>
<i>void</i>	<code>viewport_attach_to_screen (RID viewport, Rect2 rect=Rect2(0, 0, 0, 0), int screen=0)</code>
<i>RID</i>	<code>viewport_create ()</code>
<i>void</i>	<code>viewport_detach (RID viewport)</code>
<i>int</i>	<code>viewport_get_render_info (RID viewport, int info)</code>
<i>RID</i>	<code>viewport_get_texture (RID viewport) const</code>
<i>void</i>	<code>viewport_remove_canvas (RID viewport, RID canvas)</code>
<i>void</i>	<code>viewport_set_active (RID viewport, bool active)</code>
<i>void</i>	<code>viewport_set_canvas_layer (RID viewport, RID canvas, int layer)</code>
<i>void</i>	<code>viewport_set_canvas_transform (RID viewport, RID canvas, Transform2D offset)</code>
<i>void</i>	<code>viewport_set_clear_mode (RID viewport, int clear_mode)</code>
<i>void</i>	<code>viewport_set_debug_draw (RID viewport, int draw)</code>
<i>void</i>	<code>viewport_set_disable_3d (RID viewport, bool disabled)</code>
<i>void</i>	<code>viewport_set_disable_environment (RID viewport, bool disabled)</code>
<i>void</i>	<code>viewport_set_global_canvas_transform (RID viewport, Transform2D transform)</code>
<i>void</i>	<code>viewport_set_hdr (RID viewport, bool enabled)</code>
<i>void</i>	<code>viewport_set_hide_canvas (RID viewport, bool hidden)</code>
<i>void</i>	<code>viewport_set_hide_scenario (RID viewport, bool hidden)</code>
<i>void</i>	<code>viewport_set_msaa (RID viewport, int msaa)</code>
<i>void</i>	<code>viewport_set_parent_viewport (RID viewport, RID parent_viewport)</code>
<i>void</i>	<code>viewport_set_scenario (RID viewport, RID scenario)</code>
<i>void</i>	<code>viewport_set_shadow_atlas_quadrant_subdivision (RID viewport, int quadrant, int subdivision)</code>
<i>void</i>	<code>viewport_set_shadow_atlas_size (RID viewport, int size)</code>
<i>void</i>	<code>viewport_set_size (RID viewport, int width, int height)</code>
<i>void</i>	<code>viewport_set_transparent_background (RID viewport, bool enabled)</code>
<i>void</i>	<code>viewport_set_update_mode (RID viewport, int update_mode)</code>
<i>void</i>	<code>viewport_set_usage (RID viewport, int usage)</code>
<i>void</i>	<code>viewport_set_use_arvr (RID viewport, bool use_arvr)</code>
<i>void</i>	<code>viewport_set_vflip (RID viewport, bool enabled)</code>

25.476.3 Signals

- `frame_drawn_in_thread ()`

25.476.4 Numeric Constants

- `NO_INDEX_ARRAY = -1` — Marks an error that shows that the index array is empty.

- **ARRAY_WEIGHTS_SIZE = 4**
- **CANVAS_ITEM_Z_MIN = -4096** — The minimum Z-layer for canvas items.
- **CANVAS_ITEM_Z_MAX = 4096** — The maximum Z-layer for canvas items.
- **MAX_GLOW_LEVELS = 7**
- **MAX_CURSORS = 8**
- **MATERIAL_RENDER_PRIORITY_MIN = -128** — The minimum renderpriority of all materials.
- **MATERIAL_RENDER_PRIORITY_MAX = 127** — The maximum renderpriority of all materials.

25.476.5 Enums

enum **ViewportRenderInfo**

- **VIEWPORT_RENDER_INFO_OBJECTS_IN_FRAME = 0**
- **VIEWPORT_RENDER_INFO_VERTICES_IN_FRAME = 1**
- **VIEWPORT_RENDER_INFO_MATERIAL_CHANGES_IN_FRAME = 2**
- **VIEWPORT_RENDER_INFO_SHADER_CHANGES_IN_FRAME = 3**
- **VIEWPORT_RENDER_INFO_SURFACE_CHANGES_IN_FRAME = 4**
- **VIEWPORT_RENDER_INFO_DRAW_CALLS_IN_FRAME = 5**
- **VIEWPORT_RENDER_INFO_MAX = 6** — Marks end of VIEWPORT_RENDER_INFO* constants. Used internally.

enum **CubeMapSide**

- **CUBEMAP_LEFT = 0** — Marks the left side of a cubemap.
- **CUBEMAP_RIGHT = 1** — Marks the right side of a cubemap.
- **CUBEMAP_BOTTOM = 2** — Marks the bottom side of a cubemap.
- **CUBEMAP_TOP = 3** — Marks the top side of a cubemap.
- **CUBEMAP_FRONT = 4** — Marks the front side of a cubemap.
- **CUBEMAP_BACK = 5** — Marks the back side of a cubemap.

enum **LightType**

- **LIGHT_DIRECTIONAL = 0** — Is a directional (sun) light.
- **LIGHT_OMNI = 1** — is an omni light.
- **LIGHT_SPOT = 2** — is a spot light.

enum **PrimitiveType**

- **PRIMITIVE_POINTS = 0** — Primitive to draw consists of points.
- **PRIMITIVE_LINES = 1** — Primitive to draw consists of lines.
- **PRIMITIVE_LINE_STRIP = 2** — Primitive to draw consists of a line strip from start to end.
- **PRIMITIVE_LINE_LOOP = 3** — Primitive to draw consists of a line loop (a line strip with a line between the last and the first vertex).
- **PRIMITIVE_TRIANGLES = 4** — Primitive to draw consists of triangles.

- **PRIMITIVE_TRIANGLE_STRIP = 5** — Primitive to draw consists of a triangle strip (the last 3 vertices are always combined to make a triangle).
- **PRIMITIVE_TRIANGLE_FAN = 6** — Primitive to draw consists of a triangle strip (the last 2 vertices are always combined with the first to make a triangle).
- **PRIMITIVE_MAX = 7** — Marks the primitive types endpoint. used internally.

enum **BlendShapeMode**

- **BLEND_SHAPE_MODE_NORMALIZED = 0**
- **BLEND_SHAPE_MODE_RELATIVE = 1**

enum **CanvasOccluderPolygonCullMode**

- **CANVAS_OCLUDER_POLYGON_CULL_DISABLED = 0** — Culling of the canvas occluder is disabled.
- **CANVAS_OCLUDER_POLYGON_CULL_CLOCKWISE = 1** — Culling of the canvas occluder is clockwise.
- **CANVAS_OCLUDER_POLYGON_CULL_COUNTER_CLOCKWISE = 2** — Culling of the canvas occluder is counterclockwise.

enum **ViewportMSAA**

- **VIEWPORT_MSAA_DISABLED = 0** — Multisample antialiasing is disabled.
- **VIEWPORT_MSAA_2X = 1** — Multisample antialiasing is set to 2X.
- **VIEWPORT_MSAA_4X = 2** — Multisample antialiasing is set to 4X.
- **VIEWPORT_MSAA_8X = 3** — Multisample antialiasing is set to 8X.
- **VIEWPORT_MSAA_16X = 4** — Multisample antialiasing is set to 16X.

enum **ViewportUpdateMode**

- **VIEWPORT_UPDATE_DISABLED = 0**
- **VIEWPORT_UPDATE_ONCE = 1**
- **VIEWPORT_UPDATE_WHEN_VISIBLE = 2**
- **VIEWPORT_UPDATE_ALWAYS = 3**

enum **LightParam**

- **LIGHT_PARAM_ENERGY = 0** — The light's energy.
- **LIGHT_PARAM_SPECULAR = 2** — The light's influence on specularity.
- **LIGHT_PARAM_RANGE = 3** — The light's range.
- **LIGHT_PARAM_ATTENUATION = 4** — The light's attenuation.
- **LIGHT_PARAM_SPOT_ANGLE = 5** — The spotlight's angle.
- **LIGHT_PARAM_SPOT_ATTENUATION = 6** — The spotlight's attenuation.
- **LIGHT_PARAM_CONTACT_SHADOW_SIZE = 7** — Scales the shadow color.
- **LIGHT_PARAM_SHADOW_MAX_DISTANCE = 8**
- **LIGHT_PARAM_SHADOW_SPLIT_1_OFFSET = 9**
- **LIGHT_PARAM_SHADOW_SPLIT_2_OFFSET = 10**
- **LIGHT_PARAM_SHADOW_SPLIT_3_OFFSET = 11**

- **LIGHT_PARAM_SHADOW_NORMAL_BIAS = 12**
- **LIGHT_PARAM_SHADOW_BIAS = 13**
- **LIGHT_PARAM_SHADOW_BIAS_SPLIT_SCALE = 14**
- **LIGHT_PARAM_MAX = 15** — The light parameters endpoint. Used internally.

enum ScenarioDebugMode

- **SCENARIO_DEBUG_DISABLED = 0**
- **SCENARIO_DEBUG_WIREFRAME = 1**
- **SCENARIO_DEBUG_OVERDRAW = 2**
- **SCENARIO_DEBUG_SHADELESS = 3**

enum InstanceType

- **INSTANCE_NONE = 0** — The instance does not have a type.
- **INSTANCE_MESH = 1** — The instance is a mesh.
- **INSTANCE_MULTIMESH = 2** — The instance is a multimesh.
- **INSTANCE_IMMEDIATE = 3** — The instance is an immediate geometry.
- **INSTANCE_PARTICLES = 4** — The instance is a particle emitter.
- **INSTANCE_LIGHT = 5** — The instance is a light.
- **INSTANCE_REFLECTION_PROBE = 6**
- **INSTANCE_GI_PROBE = 7**
- **INSTANCE_LIGHTMAP_CAPTURE = 8**
- **INSTANCE_MAX = 9** — The max value for INSTANCE_* constants, used internally.
- **INSTANCE_GEOMETRY_MASK = 30** — A combination of the flags of geometry instances (mesh, multimesh, immediate and particles).

enum ShaderMode

- **SHADER_SPATIAL = 0** — Shader is a 3D shader.
- **SHADER_CANVAS_ITEM = 1** — Shader is a 2D shader.
- **SHADER PARTICLES = 2** — Shader is a particle shader.
- **SHADER_MAX = 3** — Marks maximum of the shader types array. used internally.

enum ViewportUsage

- **VIEWPORT_USAGE_2D = 0** — The Viewport does not render 3D but samples.
- **VIEWPORT_USAGE_2D_NO_SAMPLING = 1** — The Viewport does not render 3D and does not sample.
- **VIEWPORT_USAGE_3D = 2** — The Viewport renders 3D with effects.
- **VIEWPORT_USAGE_3D_NO_EFFECTS = 3** — The Viewport renders 3D but without effects.

enum ViewportClearMode

- **VIEWPORT_CLEAR_ALWAYS = 0** — The viewport is always cleared before drawing.
- **VIEWPORT_CLEAR_NEVER = 1** — The viewport is never cleared before drawing.
- **VIEWPORT_CLEAR_ONLY_NEXT_FRAME = 2** — The viewport is cleared once, then the clear mode is set to VIEWPORT_CLEAR_NEVER.

enum ArrayType

- **ARRAY_VERTEX = 0** — Array is a vertex array.
- **ARRAY_NORMAL = 1** — Array is a normal array.
- **ARRAY_TANGENT = 2** — Array is a tangent array.
- **ARRAY_COLOR = 3** — Array is a color array.
- **ARRAY_TEX_UV = 4** — Array is a uv coordinates array.
- **ARRAY_TEX_UV2 = 5** — Array is a uv coordinates array for the second uv coordinates.
- **ARRAY_BONES = 6** — Array contains bone information.
- **ARRAY_WEIGHTS = 7** — Array is weight information.
- **ARRAY_INDEX = 8** — Array is index array.
- **ARRAY_MAX = 9** — Marks the maximum of the array types. Used internally.

enum CanvasLightMode

- **CANVAS_LIGHT_MODE_ADD = 0** — Adds light color additive to the canvas.
- **CANVAS_LIGHT_MODE_SUB = 1** — Adds light color subtractive to the canvas.
- **CANVAS_LIGHT_MODE_MIX = 2** — The light adds color depending on transparency.
- **CANVAS_LIGHT_MODE_MASK = 3** — The light adds color depending on mask.

enum ArrayFormat

- **ARRAY_FORMAT_VERTEX = 1** — Flag used to mark a vertex array.
- **ARRAY_FORMAT_NORMAL = 2** — Flag used to mark a normal array.
- **ARRAY_FORMAT_TANGENT = 4** — Flag used to mark a tangent array.
- **ARRAY_FORMAT_COLOR = 8** — Flag used to mark a color array.
- **ARRAY_FORMAT_TEX_UV = 16** — Flag used to mark a uv coordinates array.
- **ARRAY_FORMAT_TEX_UV2 = 32** — Flag used to mark a uv coordinates array for the second uv coordinates.
- **ARRAY_FORMAT_BONES = 64** — Flag used to mark a bone information array.
- **ARRAY_FORMAT_WEIGHTS = 128** — Flag used to mark a weights array.
- **ARRAY_FORMAT_INDEX = 256** — Flag used to mark a index array.
- **ARRAY_COMPRESS_VERTEX = 512** — Flag used to mark a compressed (half float) vertex array.
- **ARRAY_COMPRESS_NORMAL = 1024** — Flag used to mark a compressed (half float) normal array.
- **ARRAY_COMPRESS_TANGENT = 2048** — Flag used to mark a compressed (half float) tangent array.
- **ARRAY_COMPRESS_COLOR = 4096** — Flag used to mark a compressed (half float) color array.
- **ARRAY_COMPRESS_TEX_UV = 8192** — Flag used to mark a compressed (half float) uv coordinates array.
- **ARRAY_COMPRESS_TEX_UV2 = 16384** — Flag used to mark a compressed (half float) uv coordinates array for the second uv coordinates.
- **ARRAY_COMPRESS_BONES = 32768**
- **ARRAY_COMPRESS_WEIGHTS = 65536** — Flag used to mark a compressed (half float) weight array.
- **ARRAY_COMPRESS_INDEX = 131072**

- **ARRAY_FLAG_USE_2D_VERTICES = 262144** — Flag used to mark that the array contains 2D vertices.
- **ARRAY_FLAG_USE_16_BIT_BONES = 524288** — Flag used to mark that the array uses 16 bit bones instead of 8 bit.
- **ARRAY_COMPRESS_DEFAULT = 97792** — Used to set flags ARRAY_COMPRESS_VERTEX, ARRAY_COMPRESS_NORMAL, ARRAY_COMPRESS_TANGENT, ARRAY_COMPRESS_COLOR, ARRAY_COMPRESS_TEX_UV, ARRAY_COMPRESS_TEX_UV2 and ARRAY_COMPRESS_WEIGHTS quickly.

enum ViewportDebugDraw

- **VIEWPORT_DEBUG_DRAW_DISABLED = 0** — Debug draw is disabled. Default setting.
- **VIEWPORT_DEBUG_DRAW_UNSHADED = 1** — Debug draw sets objects to unshaded.
- **VIEWPORT_DEBUG_DRAW_OVERDRAW = 2** — Overwrites clear color to (0, 0, 0, 0).
- **VIEWPORT_DEBUG_DRAW_WIREFRAME = 3** — Debug draw draws objects in wireframe.

enum NinePatchAxisMode

- **NINE_PATCH_STRETCH = 0** — The nine patch gets stretched where needed.
- **NINE_PATCH_TILE = 1** — The nine patch gets filled with tiles where needed.
- **NINE_PATCH_TILE_FIT = 2** — The nine patch gets filled with tiles where needed and stretches them a bit if needed.

enum RenderInfo

- **INFO_OBJECTS_IN_FRAME = 0** — The amount of objects in the frame.
- **INFO_VERTICES_IN_FRAME = 1** — The amount of vertices in the frame.
- **INFO_MATERIAL_CHANGES_IN_FRAME = 2** — The amount of modified materials in the frame.
- **INFO_SHADER_CHANGES_IN_FRAME = 3** — The amount of shader rebinds in the frame.
- **INFO_SURFACE_CHANGES_IN_FRAME = 4** — The amount of surface changes in the frame.
- **INFO_DRAW_CALLS_IN_FRAME = 5** — The amount of draw calls in frame.
- **INFO_USAGE_VIDEO_MEM_TOTAL = 6**
- **INFO_VIDEO_MEM_USED = 7** — The amount of vertex memory and texture memory used.
- **INFO_TEXTURE_MEM_USED = 8** — The amount of texture memory used.
- **INFO_VERTEX_MEM_USED = 9** — The amount of vertex memory used.

enum CanvasLightShadowFilter

- **CANVAS_LIGHT_FILTER_NONE = 0**
- **CANVAS_LIGHT_FILTER_PCF3 = 1**
- **CANVAS_LIGHT_FILTER_PCF5 = 2**
- **CANVAS_LIGHT_FILTER_PCF7 = 3**
- **CANVAS_LIGHT_FILTER_PCF9 = 4**
- **CANVAS_LIGHT_FILTER_PCF13 = 5**

enum Features

- **FEATURE_SHADERS = 0**
- **FEATURE_MULTITHREADED = 1**

enum TextureFlags

- **TEXTURE_FLAG_MIPMAPS = 1** — Generate mipmaps, which are smaller versions of the same texture to use when zoomed out, keeping the aspect ratio.
- **TEXTURE_FLAG_REPEAT = 2** — Repeat (instead of clamp to edge).
- **TEXTURE_FLAG_FILTER = 4** — Turn on magnifying filter, to enable smooth zooming in of the texture.
- **TEXTURE_FLAG_ANISOTROPIC_FILTER = 8** — Anisotropic mipmap filtering. Generates smaller versions of the same texture with different aspect ratios.

More effective on planes often shown going to the horizon as those textures (Walls or Ground for example) get squashed in the viewport to different aspect ratios and regular mipmaps keep the aspect ratio so they don't optimize storage that well in those cases. - **TEXTURE_FLAG_CONVERT_TO_LINEAR = 16** — Converts texture to SRGB color space. - **TEXTURE_FLAG_MIRRORED_REPEAT = 32** — Repeat texture with alternate sections mirrored. - **TEXTURE_FLAG_CUBEMAP = 2048** — Texture is a cube-map. - **TEXTURE_FLAG_USED_FOR_STREAMING = 4096** — Texture is a video surface. - **TEXTURE_FLAGS_DEFAULT = 7** — Default flags. Generate mipmaps, repeat, and filter are enabled.

25.476.6 Description

Server for anything visible. The visual server is the API backend for everything visible. The whole scene system mounts on it to display.

The visual server is completely opaque, the internals are entirely implementation specific and cannot be accessed.

25.476.7 Member Function Description

- void **black_bars_set_images** (*RID* left, *RID* top, *RID* right, *RID* bottom)

Sets images to be rendered in the window margin.

- void **black_bars_set_margins** (*int* left, *int* top, *int* right, *int* bottom)

Sets margin size, where black bars (or images, if *black_bars_set_images* was used) are rendered.

- *RID* **canvas_create** ()

Creates a canvas and returns the assigned *RID*.

- void **canvas_item_add_circle** (*RID* item, *Vector2* pos, *float* radius, *Color* color)

Adds a circle command to the *CanvasItem*'s draw commands.

- void **canvas_item_add_clip_ignore** (*RID* item, *bool* ignore)

If ignore is `true`, the VisualServer does not perform clipping.

- void **canvas_item_add_line** (*RID* item, *Vector2* from, *Vector2* to, *Color* color, *float* width=1.0, *bool* antialiased=false)

Adds a line command to the *CanvasItem*'s draw commands.

- void **canvas_item_add_mesh** (*RID* item, *RID* mesh, *RID* skeleton)

Adds a *Mesh* to the *CanvasItem*'s draw commands. Only affects its aabb at the moment.

- void **canvas_item_add_multimesh** (*RID* item, *RID* mesh, *RID* skeleton)

Adds a *MultiMesh* to the *CanvasItem*'s draw commands. Only affects its aabb at the moment.

- void **canvas_item_add_nine_patch** (*RID item*, *Rect2 rect*, *Rect2 source*, *RID texture*, *Vector2 topleft*, *Vector2 bottomright*, *int x_axis_mode=0*, *int y_axis_mode=0*, *bool draw_center=true*, *Color modulate=Color(1, 1, 1, 1)*, *RID normal_map*)

Adds a nine patch image to the *CanvasItem*'s draw commands.

See *NinePatchRect* for more explanation.

- void **canvas_item_add_particles** (*RID item*, *RID particles*, *RID texture*, *RID normal_map*, *int h_frames*, *int v_frames*)

Adds a particles system to the *CanvasItem*'s draw commands.

- void **canvas_item_add_polygon** (*RID item*, *PoolVector2Array points*, *PoolColorArray colors*, *PoolVector2Array uvs=PoolVector2Array()*, *RID texture*, *RID normal_map*, *bool antialiased=false*)

Adds a polygon to the *CanvasItem*'s draw commands.

- void **canvas_item_add_polyline** (*RID item*, *PoolVector2Array points*, *PoolColorArray colors*, *float width=1.0*, *bool antialiased=false*)

Adds a polyline, which is a line from multiple points with a width, to the *CanvasItem*'s draw commands.

- void **canvas_item_add_primitive** (*RID item*, *PoolVector2Array points*, *PoolColorArray colors*, *PoolVector2Array uvs*, *RID texture*, *float width=1.0*, *RID normal_map*)

Adds a primitive to the *CanvasItem*'s draw commands.

- void **canvas_item_add_rect** (*RID item*, *Rect2 rect*, *Color color*)

Adds a rectangle to the *CanvasItem*'s draw commands.

- void **canvas_item_add_set_transform** (*RID item*, *Transform2D transform*)

Adds a *Transform2D* command to the *CanvasItem*'s draw commands.

This sets the *extra_matrix* uniform when executed. This affects the later command's of the canvas item.

- void **canvas_item_add_texture_rect** (*RID item*, *Rect2 rect*, *RID texture*, *bool tile=false*, *Color modulate=Color(1, 1, 1, 1)*, *bool transpose=false*, *RID normal_map*)

Adds a textured rect to the *CanvasItem*'s draw commands.

- void **canvas_item_add_texture_rect_region** (*RID item*, *Rect2 rect*, *RID texture*, *Rect2 src_rect*, *Color modulate=Color(1, 1, 1, 1)*, *bool transpose=false*, *RID normal_map*, *bool clip_uv=true*)

Adds a texture rect with region setting to the *CanvasItem*'s draw commands.

- void **canvas_item_add_triangle_array** (*RID item*, *PoolIntArray indices*, *PoolVector2Array points*, *PoolColorArray colors*, *PoolVector2Array uvs=PoolVector2Array()*, *RID texture*, *int count=-1*, *RID normal_map*)

Adds a triangle array to the *CanvasItem*'s draw commands.

- void **canvas_item_clear** (*RID item*)

Clears the *CanvasItem* and removes all commands in it.

- *RID canvas_item_create()*

Creates a new *CanvasItem* and returns its *RID*.

- void **canvas_item_set_clip** (*RID item*, *bool clip*)

Sets clipping for the *CanvasItem*.

- void **canvas_item_set_copy_to_backbuffer** (*RID item*, *bool enabled*, *Rect2 rect*)

Sets the *CanvasItem* to copy a rect to the backbuffer.

- void **canvas_item_set_custom_rect** (*RID item, bool use_custom_rect, Rect2 rect=Rect2(0, 0, 0, 0)*)

Defines a custom drawing rectangle for the *CanvasItem*.

- void **canvas_item_set_distance_field_mode** (*RID item, bool enabled*)
- void **canvas_item_set_draw_behind_parent** (*RID item, bool enabled*)

Sets *CanvasItem* to be drawn behind its parent.

- void **canvas_item_set_draw_index** (*RID item, int index*)

Sets the index for the *CanvasItem*.

- void **canvas_item_set_light_mask** (*RID item, int mask*)

The light mask. See *LightOccluder2D* for more information on light masks.

- void **canvas_item_set_material** (*RID item, RID material*)

Sets a new material to the *CanvasItem*.

- void **canvas_item_set_modulate** (*RID item, Color color*)

Sets the color that modulates the *CanvasItem* and its children.

- void **canvas_item_set_parent** (*RID item, RID parent*)

Sets the parent for the *CanvasItem*.

- void **canvas_item_set_self_modulate** (*RID item, Color color*)

Sets the color that modulates the *CanvasItem* without children.

- void **canvas_item_set_sort_children_by_y** (*RID item, bool enabled*)

Sets if *CanvasItem*'s children should be sorted by y-position.

- void **canvas_item_set_transform** (*RID item, Transform2D transform*)

Sets the *CanvasItem*'s *Transform2D*.

- void **canvas_item_set_use_parent_material** (*RID item, bool enabled*)

Sets if the *CanvasItem* uses its parent's material.

- void **canvas_item_set_visible** (*RID item, bool visible*)

Sets if the canvas item (including its children) is visible.

- void **canvas_item_set_z_as_relative_to_parent** (*RID item, bool enabled*)

If this is enabled, the z-index of the parent will be added to the children's z-index.

- void **canvas_item_set_z_index** (*RID item, int z_index*)

Sets the *CanvasItem*'s z-index, i.e. its draw order (lower indexes are drawn first).

- void **canvas_light_attach_to_canvas** (*RID light, RID canvas*)

Attaches the canvas light to the canvas. Removes it from its previous canvas.

- *RID canvas_light_create* ()

Creates a canvas light.

- void **canvas_light_occluder_attach_to_canvas** (*RID occluder, RID canvas*)

Attaches a light occluder to the canvas. Removes it from its previous canvas.

- *RID canvas_light_occluder_create* ()

Creates a light occluder.

- void **canvas_light_occluder_set_enabled** (*RID* occluder, *bool* enabled)

Enables or disables light occluder.

- void **canvas_light_occluder_set_light_mask** (*RID* occluder, *int* mask)

The light mask. See *LightOccluder2D* for more information on light masks

- void **canvas_light_occluder_set_polygon** (*RID* occluder, *RID* polygon)

Sets a light occluder's polygon.

- void **canvas_light_occluder_set_transform** (*RID* occluder, *Transform2D* transform)

Sets a light occluder's *Transform2D*.

- void **canvas_light_set_color** (*RID* light, *Color* color)

Sets the color for a light.

- void **canvas_light_set_enabled** (*RID* light, *bool* enabled)

Enables or disables a canvas light.

- void **canvas_light_set_energy** (*RID* light, *float* energy)

Sets a canvas light's energy.

- void **canvas_light_set_height** (*RID* light, *float* height)

Sets a canvas light's height.

- void **canvas_light_set_item_cull_mask** (*RID* light, *int* mask)

The light mask. See *LightOccluder2D* for more information on light masks

- void **canvas_light_set_item_shadow_cull_mask** (*RID* light, *int* mask)

The shadow mask. binary about which layers this canvas light affects which canvas item's shadows. See *LightOccluder2D* for more information on light masks.

- void **canvas_light_set_layer_range** (*RID* light, *int* min_layer, *int* max_layer)

The layer range that gets rendered with this light.

- void **canvas_light_set_mode** (*RID* light, *int* mode)

The mode of the light, see *CANVAS_LIGHT_MODE_** constants.

- void **canvas_light_set_scale** (*RID* light, *float* scale)

- void **canvas_light_set_shadow_buffer_size** (*RID* light, *int* size)

Sets the width of the shadow buffer, size gets scaled to the next power of two for this.

- void **canvas_light_set_shadow_color** (*RID* light, *Color* color)

Sets the color of the canvas light's shadow.

- void **canvas_light_set_shadow_enabled** (*RID* light, *bool* enabled)

Enables or disables the canvas light's shadow.

- void **canvas_light_set_shadow_filter** (*RID* light, *int* filter)

Sets the canvas light's shadow's filter, see *CANVAS_LIGHT_SHADOW_FILTER_** constants.

- void **canvas_light_set_shadow_gradient_length** (*RID* light, *float* length)

Sets the length of the shadow's gradient.

- `void canvas_light_set_shadow_smooth (RID light, float smooth)`

Smoothens the shadow. The lower, the more smooth.

- `void canvas_light_set_texture (RID light, RID texture)`
- `void canvas_light_set_texture_offset (RID light, Vector2 offset)`
- `void canvas_light_set_transform (RID light, Transform2D transform)`

Sets the canvas light's *Transform2D*.

- `void canvas_light_set_z_range (RID light, int min_z, int max_z)`
- `RID canvas_occluder_polygon_create ()`

Creates a new light occluder polygon.

- `void canvas_occluder_polygon_set_cull_mode (RID occluder_polygon, int mode)`

Sets an occluder polygons cull mode. See CANVAS_OCCLUDER_POLYGON_CULL_MODE_* constants.

- `void canvas_occluder_polygon_set_shape (RID occluder_polygon, PoolVector2Array shape, bool closed)`

Sets the shape of the occluder polygon.

- `void canvas_occluder_polygon_set_shape_as_lines (RID occluder_polygon, PoolVector2Array shape)`

Sets the shape of the occluder polygon as lines.

- `void canvas_set_item_mirroring (RID canvas, RID item, Vector2 mirroring)`

A copy of the canvas item will be drawn with a local offset of the mirroring *Vector2*.

- `void canvas_set_modulate (RID canvas, Color color)`

Modulates all colors in the given canvas.

- `void draw (bool swap_buffers=true)`

Draws a frame.

- `void finish ()`

Removes buffers and clears testcubes.

- `void force_draw (bool swap_buffers=true)`

Draws a frame. Same as *draw*.

- `void force_sync ()`

Synchronizes threads.

- `void free_rid (RID rid)`

Tries to free an object in the VisualServer.

- `int get_render_info (int info)`

Returns a certain information, see RENDER_INFO_* for options.

- `RID get_test_cube ()`

Returns the id of the test cube. Creates one if none exists.

- `RID get_test_texture ()`

Returns the id of the test texture. Creates one if none exists.

- *RID* **get_white_texture ()**

Returns the id of a white texture. Creates one if none exists.

- *bool* **has_changed () const**

Returns `true` if changes have been made to the VisualServer's data. `draw` is usually called if this happens.

- *bool* **has_feature (int feature) const**
- *bool* **has_os_feature (String feature) const**

Returns true, if the OS supports a certain feature. Features might be s3tc, etc, etc2 and pvrbc,

- *void* **init ()**

Initializes the visual server.

- *RID* **make_sphere_mesh (int latitudes, int longitudes, float radius)**

Returns a mesh of a sphere with the given amount of horizontal and vertical subdivisions.

- *RID* **material_create ()**

Returns an empty material.

- *Variant* **material_get_param (RID material, String parameter) const**

Returns the value of a certain material's parameter.

- *RID* **material_get_shader (RID shader_material) const**

Returns the shader of a certain material's shader. Returns an empty RID if the material doesn't have a shader.

- *void* **material_set_line_width (RID material, float width)**

Sets a materials line width.

- *void* **material_set_next_pass (RID material, RID next_material)**

Sets an objects next material.

- *void* **material_set_param (RID material, String parameter, Variant value)**

Sets a materials parameter.

- *void* **material_set_render_priority (RID material, int priority)**

Sets a material's render priority.

- *void* **material_set_shader (RID shader_material, RID shader)**

Sets a shader material's shader.

- *void* **mesh_add_surface_from_arrays (RID mesh, int primitive, Array arrays, Array blend_shapes=[], int compress_format=97792)**

Adds a surface generated from the Arrays to a mesh. See PRIMITIVE_TYPE_* constants for types.

- *void* **mesh_clear (RID mesh)**

Removes all surfaces from a mesh.

- *RID* **mesh_create ()**

Creates a new mesh.

- *int* **mesh_get_blend_shape_count (RID mesh) const**

Returns a mesh's blend shape count.

- `int mesh_get_blend_shape_mode (RID mesh) const`

Returns a mesh's blend shape mode.

- `AABB mesh_get_custom_aabb (RID mesh) const`

Returns a mesh's custom aabb.

- `int mesh_get_surface_count (RID mesh) const`

Returns a mesh's number of surfaces.

- `void mesh_remove_surface (RID mesh, int index)`

Removes a mesh's surface.

- `void mesh_set_blend_shape_count (RID mesh, int amount)`

Sets a mesh's blend shape count.

- `void mesh_set_blend_shape_mode (RID mesh, int mode)`

Sets a mesh's blend shape mode.

- `void mesh_set_custom_aabb (RID mesh, AABB aabb)`

Sets a mesh's custom aabb.

- `AABB mesh_surface_get_aabb (RID mesh, int surface) const`

Returns a mesh's surface's aabb.

- `PoolByteArray mesh_surface_get_array (RID mesh, int surface) const`

Returns a mesh's surface's vertex buffer.

- `int mesh_surface_get_array_index_len (RID mesh, int surface) const`

Returns a mesh's surface's amount of indices.

- `int mesh_surface_get_array_len (RID mesh, int surface) const`

Returns a mesh's surface's amount of vertices.

- `Array mesh_surface_get_arrays (RID mesh, int surface) const`

Returns a mesh's surface's buffer arrays.

- `Array mesh_surface_get_blend_shape_arrays (RID mesh, int surface) const`

Returns a mesh's surface's arrays for blend shapes

- `int mesh_surface_get_format (RID mesh, int surface) const`

Returns the format of a mesh's surface.

- `PoolByteArray mesh_surface_get_index_array (RID mesh, int surface) const`

Returns a mesh's surface's index buffer.

- `RID mesh_surface_get_material (RID mesh, int surface) const`

Returns a mesh's surface's material.

- `int mesh_surface_get_primitive_type (RID mesh, int surface) const`

Returns the primitive type of a mesh's surface.

- `Array mesh_surface_get_skeleton_aabb (RID mesh, int surface) const`

Returns the aabb of a mesh's surface's skeleton.

- void **mesh_surface_set_material** (*RID* mesh, *int* surface, *RID* material)

Sets a mesh's surface's material.

- void **request_frame_drawn_callback** (*Object* where, *String* method, *Variant* userdata)

Schedules a callback to the corresponding named ‘method’ on ‘where’ after a frame has been drawn.

The callback method must use only 1 argument which will be called with ‘userdata’.

- void **set_boot_image** (*Image* image, *Color* color, *bool* scale)

Sets a boot image. The color defines the background color and if scale is `true`, the image will be scaled to fit the screen size.

- void **set_debug_generate_wireframes** (*bool* generate)

- void **set_default_clear_color** (*Color* color)

- *RID* **shader_create** ()

Creates an empty shader.

- *String* **shader_get_code** (*RID* shader) const

Returns a shader's code.

- *RID* **shader_get_default_texture_param** (*RID* shader, *String* name) const

Returns a default texture from a shader searched by name.

- *Array* **shader_get_param_list** (*RID* shader) const

Returns the parameters of a shader.

- void **shader_set_code** (*RID* shader, *String* code)

Sets a shader's code.

- void **shader_set_default_texture_param** (*RID* shader, *String* name, *RID* texture)

Sets a shader's default texture. Overwrites the texture given by name.

- *RID* **sky_create** ()

Creates an empty sky.

- void **sky_set_texture** (*RID* sky, *RID* cube_map, *int* radiance_size)

Sets a sky's texture.

- void **sync** ()

- void **texture_allocate** (*RID* texture, *int* width, *int* height, *int* format, *int* flags=7)

Allocates space for a texture's image or video.

- *RID* **texture_create** ()

Creates an empty texture.

- *RID* **texture_create_from_image** (*Image* image, *int* flags=7)

Creates a texture, allocates the space for an image, and fills in the image.

- *Array* **texture_debug_usage** ()

Returns a list of all the textures and their information.

- *Image* **texture_get_data** (*RID* texture, *int* cube_side=0) const

Returns a copy of a texture's image unless it's a CubeMap, in which case it returns the *RID* of the image at one of the cubes sides.

- `int texture_get_flags (RID texture) const`

Returns the flags of a texture.

- `int texture_get_format (RID texture) const`

Returns the format of the texture's image.

- `int texture_get_height (RID texture) const`

Returns the texture's height.

- `String texture_get_path (RID texture) const`

Returns the texture's path.

- `int texture_get_texid (RID texture) const`

Returns the opengl id of the texture's image.

- `int texture_get_width (RID texture) const`

Returns the texture's width.

- `void texture_set_data (RID texture, Image image, int cube_side=0)`

Sets the texture's image data. If it's a CubeMap, it sets the image data at a cube side.

- `void texture_set_flags (RID texture, int flags)`

Sets the texture's flags. See enum TextureFlags for options

- `void texture_set_path (RID texture, String path)`

Sets the texture's path.

- `void texture_set_shrink_all_x2_on_set_data (bool shrink)`

If true, sets internal processes to shrink all image data to half the size.

- `void texture_set_size_override (RID texture, int width, int height)`

Overwrites the texture's width and height.

- `void textures_keep_original (bool enable)`

If true, the image will be stored in the texture's images array if overwritten.

- `void viewport_attach_camera (RID viewport, RID camera)`

Sets a viewport's camera.

- `void viewport_attach_canvas (RID viewport, RID canvas)`

Sets a viewport's canvas.

- `void viewport_attach_to_screen (RID viewport, Rect2 rect=Rect2(0, 0, 0, 0), int screen=0)`

Attaches a viewport to a screen.

- `RID viewport_create ()`

Creates an empty viewport.

- `void viewport_detach (RID viewport)`

Detaches the viewport from the screen.

- `int viewport_get_render_info (RID viewport, int info)`

Returns a viewport's render info. for options see VIEWPORT_RENDER_INFO* constants.

- `RID viewport_get_texture (RID viewport) const`

Returns the viewport's last rendered frame.

- `void viewport_remove_canvas (RID viewport, RID canvas)`

Detaches a viewport from a canvas and vice versa.

- `void viewport_set_active (RID viewport, bool active)`

If true, sets the viewport active, else sets it inactive.

- `void viewport_set_canvas_layer (RID viewport, RID canvas, int layer)`

Sets the renderlayer for a viewport's canvas.

- `void viewport_set_canvas_transform (RID viewport, RID canvas, Transform2D offset)`

Sets the transformation of a viewport's canvas.

- `void viewport_set_clear_mode (RID viewport, int clear_mode)`

Sets the clear mode of a viewport. See VIEWPORT_CLEAR_MODE_* constants for options.

- `void viewport_set_debug_draw (RID viewport, int draw)`

Sets the debug draw mode of a viewport. See VIEWPORT_DEBUG_DRAW_* constants for options.

- `void viewport_set_disable_3d (RID viewport, bool disabled)`

If true a viewport's 3D rendering should be disabled.

- `void viewport_set_disable_environment (RID viewport, bool disabled)`

If true rendering of a viewport's environment should be disabled.

- `void viewport_set_global_canvas_transform (RID viewport, Transform2D transform)`

Sets the viewport's global transformation matrix.

- `void viewport_set_hdr (RID viewport, bool enabled)`

If true the viewport should render to hdr.

- `void viewport_set_hide_canvas (RID viewport, bool hidden)`

If true the viewport's canvas should not be rendered.

- `void viewport_set_hide_scenario (RID viewport, bool hidden)`

- `void viewport_set_msaa (RID viewport, int msaa)`

Sets the anti-aliasing mode. see enum ViewportMSAA for options.

- `void viewport_set_parent_viewport (RID viewport, RID parent_viewport)`

Sets the viewport's parent to another viewport.

- `void viewport_set_scenario (RID viewport, RID scenario)`

Sets a viewport's scenario.

The scenario contains information about the enum ScenarioDebugMode, environment information, reflection atlas etc.

- `void viewport_set_shadow_atlas_quadrant_subdivision (RID viewport, int quadrant, int subdivision)`

Sets the shadow atlas quadrant's subdivision.

- void **viewport_set_shadow_atlas_size** (*RID* viewport, *int* size)

Sets the size of the shadow atlas's images.

- void **viewport_set_size** (*RID* viewport, *int* width, *int* height)

Sets the viewport's width and height.

- void **viewport_set_transparent_background** (*RID* viewport, *bool* enabled)

If `true` the viewport should render its background as transparent.

- void **viewport_set_update_mode** (*RID* viewport, *int* update_mode)

Sets when the viewport should be updated. See `VIEWPORT_UPDATE_MODE_*` constants for options.

- void **viewport_set_usage** (*RID* viewport, *int* usage)

Sets what should be rendered in the viewport. See `VIEWPORT_USAGE_*` constants for options.

- void **viewport_set_use_arvr** (*RID* viewport, *bool* use_arvr)

If `true` the viewport should use augmented or virtual reality technologies. See `ARVRInterface`.

- void **viewport_set_vflip** (*RID* viewport, *bool* enabled)

If `true` the viewport's rendering should be flipped vertically.

25.477 VScrollBar

Inherits: `ScrollBar < Range < Control < CanvasItem < Node < Object`

Category: Core

25.477.1 Brief Description

Vertical version of `ScrollBar`, which goes from left (min) to right (max).

25.478 VSeparator

Inherits: `Separator < Control < CanvasItem < Node < Object`

Category: Core

25.478.1 Brief Description

Vertical version of `Separator`.

25.478.2 Description

Vertical version of `Separator`. It is used to separate objects horizontally, though (but it looks vertical!).

25.479 VSlider

Inherits: *Slider < Range < Control < CanvasItem < Node < Object*

Category: Core

25.479.1 Brief Description

Vertical slider.

25.479.2 Description

Vertical slider. See *Slider*. This one goes from left (min) to right (max).

25.480 VSplitContainer

Inherits: *SplitContainer < Container < Control < CanvasItem < Node < Object*

Category: Core

25.480.1 Brief Description

Vertical split container.

25.480.2 Description

Vertical split container. See *SplitContainer*. This goes from left to right.

25.481 WeakRef

Inherits: *Reference < Object*

Category: Core

25.481.1 Brief Description

Holds an *Object*, but does not contribute to the reference count if the object is a reference.

25.481.2 Member Functions

<i>Variant</i>	<code>get_ref () const</code>
----------------	-------------------------------

25.481.3 Description

A weakref can hold a *Reference*, without contributing to the reference counter. A weakref can be created from an *Object* using `@GDScript.weakref`. If this object is not a reference, weakref still works, however, it does not have any effect on the object. Weakrefs are useful in cases where multiple classes have variables that refer to each other. Without weakrefs, using these classes could lead to memory leaks, since both references keep each other from being released. Making part of the variables a weakref can prevent this cyclic dependency, and allows the references to be released.

25.481.4 Member Function Description

- *Variant get_ref() const*

Returns the *Object* this weakref is referring to.

25.482 WindowDialog

Inherits: *Popup < Control < CanvasItem < Node < Object*

Inherited By: *AcceptDialog*

Category: Core

25.482.1 Brief Description

Base class for window dialogs.

25.482.2 Member Functions

<i>TextureButton</i>	<code>get_close_button()</code>
----------------------	---------------------------------

25.482.3 Member Variables

- *bool resizable* - If `true` the user can resize the window. Default value: `false`.
- *String window_title* - The text displayed in the window's title bar. Default value: "Save a File".

25.482.4 Description

Windowdialog is the base class for all window-based dialogs. It's a by-default toplevel *Control* that draws a window decoration and allows motion and resizing.

25.482.5 Member Function Description

- *TextureButton get_close_button()*

Return the close *TextureButton*.

25.483 World

Inherits: *Resource < Reference < Object*

Category: Core

25.483.1 Brief Description

Class that has everything pertaining to a world.

25.483.2 Member Variables

- *PhysicsDirectSpaceState* **direct_space_state** - The World's physics direct space state, used for making various queries. Might be used only during `_physics_process`.
- *Environment* **environment** - The World's *Environment*.
- *Environment* **fallback_environment** - The World's `fallback_environment` will be used if the World's *Environment* fails or is missing.
- **RID scenario** - The World's visual scenario.
- **RID space** - The World's physics space.

25.483.3 Description

Class that has everything pertaining to a world. A physics space, a visual scenario and a sound space. Spatial nodes register their resources into the current world.

25.484 World2D

Inherits: *Resource < Reference < Object*

Category: Core

25.484.1 Brief Description

Class that has everything pertaining to a 2D world.

25.484.2 Member Variables

- **RID canvas** - The *RID* of this world's canvas resource. Used by the *VisualServer* for 2D drawing.
- *Physics2DDirectSpaceState* **direct_space_state** - The state of this world's physics space. This allows arbitrary querying for collision.
- **RID space** - The *RID* of this world's physics space resource. Used by the *Physics2DServer* for 2D physics, treating it as both a space and an area.

25.484.3 Description

Class that has everything pertaining to a 2D world. A physics space, a visual scenario and a sound space. 2D nodes register their resources into the current 2D world.

25.485 WorldEnvironment

Inherits: *Node < Object*

Category: Core

25.485.1 Brief Description

Default environment properties for the entire scene (post-processing effects, lightning and background settings).

25.485.2 Member Variables

- **Environment environment** - The *Environment* resource used by this `WorldEnvironment`, defining the default properties.

25.485.3 Description

The `WorldEnvironment` node is used to configure the default *Environment* for the scene.

The parameters defined in the `WorldEnvironment` can be overridden by an *Environment* node set on the current *Camera*. Additionally, only one `WorldEnvironment` may be instanced in a given scene at a time.

The `WorldEnvironment` allows the user to specify default lighting parameters (e.g. ambient lighting), various post-processing effects (e.g. SSAO, DOF, Tonemapping), and how to draw the background (e.g. solid color, skybox). Usually, these are added in order to improve the realism/color balance of the scene.

25.486 XMLParser

Inherits: *Reference < Object*

Category: Core

25.486.1 Brief Description

Low-level class for creating parsers for XML files.

25.486.2 Member Functions

<i>int</i>	<code>get_attribute_count () const</code>
<i>String</i>	<code>get_attribute_name (int idx) const</code>
<i>String</i>	<code>get_attribute_value (int idx) const</code>
<i>int</i>	<code>get_current_line () const</code>
<i>String</i>	<code>get_named_attribute_value (String name) const</code>
<i>String</i>	<code>get_named_attribute_value_safe (String name) const</code>
<i>String</i>	<code>get_node_data () const</code>
<i>String</i>	<code>get_node_name () const</code>
<i>int</i>	<code>get_node_offset () const</code>
<i>int</i>	<code>get_node_type ()</code>
<i>bool</i>	<code>has_attribute (String name) const</code>
<i>bool</i>	<code>is_empty () const</code>
<i>int</i>	<code>open (String file)</code>
<i>int</i>	<code>open_buffer (PoolByteArray buffer)</code>
<i>int</i>	<code>read ()</code>
<i>int</i>	<code>seek (int position)</code>
<i>void</i>	<code>skip_section ()</code>

25.486.3 Enums

enum NodeType

- **NODE_NONE = 0** — There's no node (no file or buffer opened)
- **NODE_ELEMENT = 1** — Element (tag)
- **NODE_ELEMENT_END = 2** — End of element
- **NODE_TEXT = 3** — Text node
- **NODE_COMMENT = 4** — Comment node
- **NODE_CDATA = 5** — CDATA content
- **NODE_UNKNOWN = 6** — Unknown node

25.486.4 Description

This class can serve as base to make custom XML parsers. Since XML is a very flexible standard, this interface is low level so it can be applied to any possible schema.

25.486.5 Member Function Description

- `int get_attribute_count () const`

Get the amount of attributes in the current element.

- `String get_attribute_name (int idx) const`

Get the name of the attribute specified by the index in `idx` argument.

- `String get_attribute_value (int idx) const`

Get the value of the attribute specified by the index in `idx` argument.

- `int get_current_line() const`

Get the current line in the parsed file (currently not implemented).

- `String get_named_attribute_value(String name) const`

Get the value of a certain attribute of the current element by name. This will raise an error if the element has no such attribute.

- `String get_named_attribute_value_safe(String name) const`

Get the value of a certain attribute of the current element by name. This will return an empty `String` if the attribute is not found.

- `String get_node_data() const`

Get the contents of a text node. This will raise an error in any other type of node.

- `String get_node_name() const`

Get the name of the current element node. This will raise an error if the current node type is not `NODE_ELEMENT` nor `NODE_ELEMENT_END`

- `int get_node_offset() const`

Get the byte offset of the current node since the beginning of the file or buffer.

- `int get_node_type()`

Get the type of the current node. Compare with `NODE__*` constants.

- `bool has_attribute(String name) const`

Check whether or not the current element has a certain attribute.

- `bool is_empty()`

Check whether the current element is empty (this only works for completely empty tags, e.g. `<element>`).

- `int open(String file)`

Open a XML file for parsing. This returns an error code.

- `int open_buffer(PoolByteArray buffer)`

Open a XML raw buffer for parsing. This returns an error code.

- `int read()`

Read the next node of the file. This returns an error code.

- `int seek(int position)`

Move the buffer cursor to a certain offset (since the beginning) and read the next node there. This returns an error code.

- `void skip_section()`

Skips the current section. If the node contains other elements, they will be ignored and the cursor will go to the closing of the current element.

25.487 YSort

Inherits: *Node2D < CanvasItem < Node < Object*

Category: Core

25.487.1 Brief Description

Sort all child nodes based on their Y positions.

25.487.2 Member Variables

- *bool sort_enabled*

25.487.3 Description

Sort all child nodes based on their Y positions. The child node must inherit from *CanvasItem* for it to be sorted. Nodes that have a higher Y position will be drawn later, so they will appear on top of nodes that have a lower Y position.