Godot Engine Documentation

Release latest

Juan Linietsky, Ariel Manzur and the Godot community

Jun 06, 2018

General

1 About 3

2 Step by step 11

3 Editor manual 179

4 Scripting 195

5 Project workflow 285

6 2D 335

7 3D 375

8 Audio 605

9 Physics 619

10 Math 649

11 Animation 709

12 Inputs 747

13 I/O 755

14 Internationalization 767

15 GUI 779

16 Viewports 797

17 Shading 803

18 Networking 825

19 Asset Library 841

20 VR 851

21 Plugins 855

22 Platform-specific 877

23 Miscellaneous 887

24 Debug 899

25 Compiling 903

26 Engine development 933

27 Godot file formats 993

28 Community 997

29 Godot API 1025

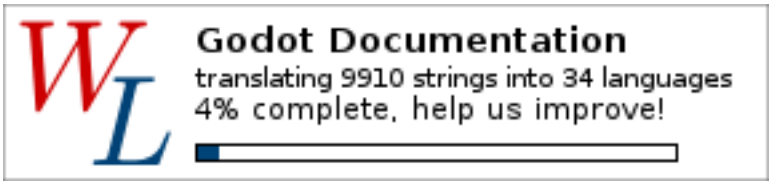
Attention: This is the documentation for the unstable (master) branch. Looking for the documentation of the current **stable** branch? Have a look here.

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. The documentation team can always use your feedback and help to improve the tutorials and class reference. If you don't 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, help us translate the documentation into your language, or discuss with us on either the `#documentation` channel on Discord, or the `#godotengine-doc` channel on irc.freenode.net!



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 yet 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 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, or translating it into your language on Hosted Weblate.

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 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 also points to various community channels like IRC and Discord and contains a list of recommended third-party tutorials outside of this documentation.
- Finally, the *Class reference* is the documentation of the Godot API, which is also available directly within the engine's script editor. It is generated automatically from a file in the main source repository, therefore the generated files of the documentation are not meant to be modified. See *Contribute to the Class Reference* for details.

In addition to this documentation you may also want to take a look at the various Godot demo projects.

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 systems 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 simple and easy to learn. Takes at most a day or two to get comfortable and it's 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 well supported by GDScript.

GDScript was designed to solve the issues above, and performs well in all the above scenarios. Please learn GDScript and enjoy a smooth integration of scripting with the game engine (yes, it's a rare but 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 restrictive license, that is incompatible with the open license provided by Godot.

That said, Godot's Collada support is 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 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

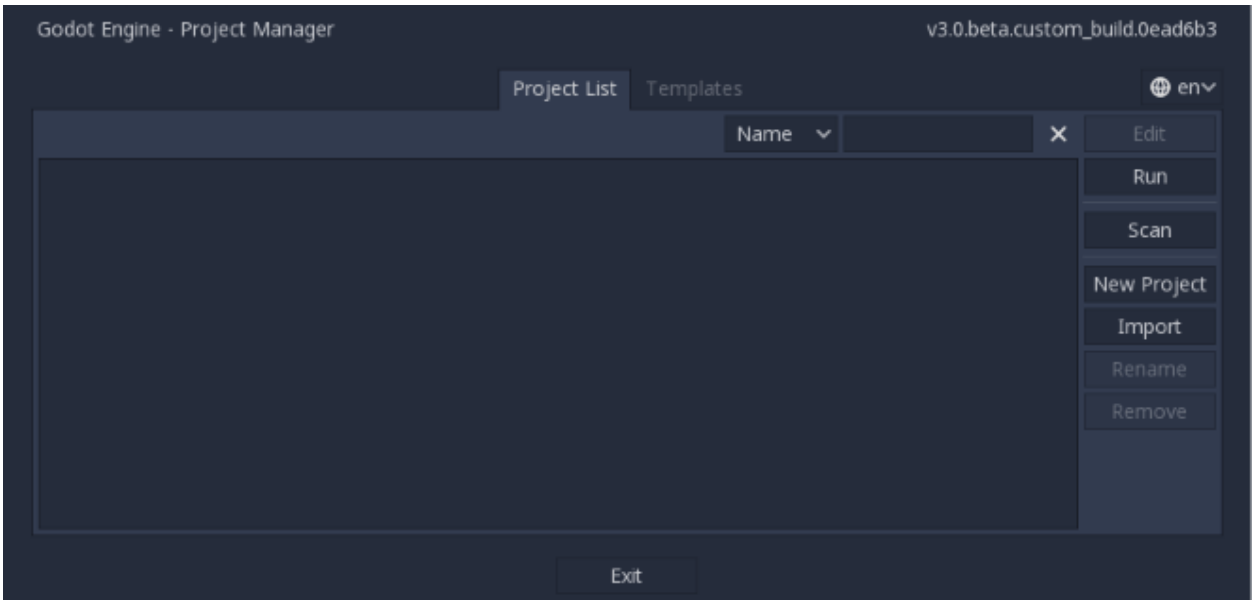
Step by step

2.1 Introduction to Godot's editor

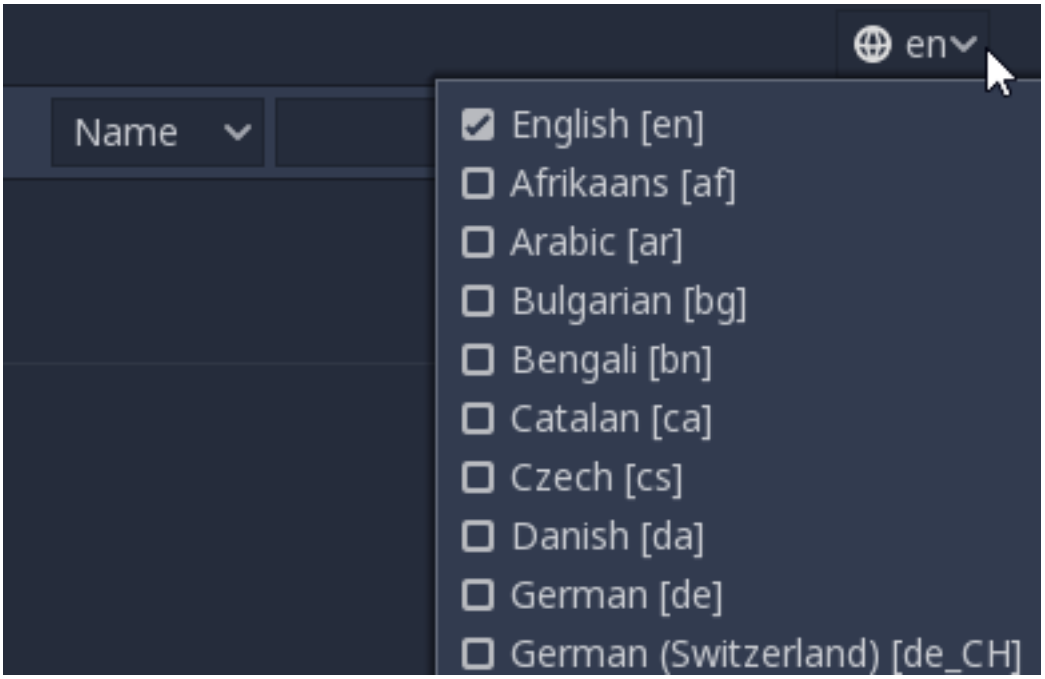
This tutorial will run you through Godot's interface. We're going to look at the **Project Manager, docks, workspaces** and everything you need to know to get started with the engine.

2.1.1 Project manager

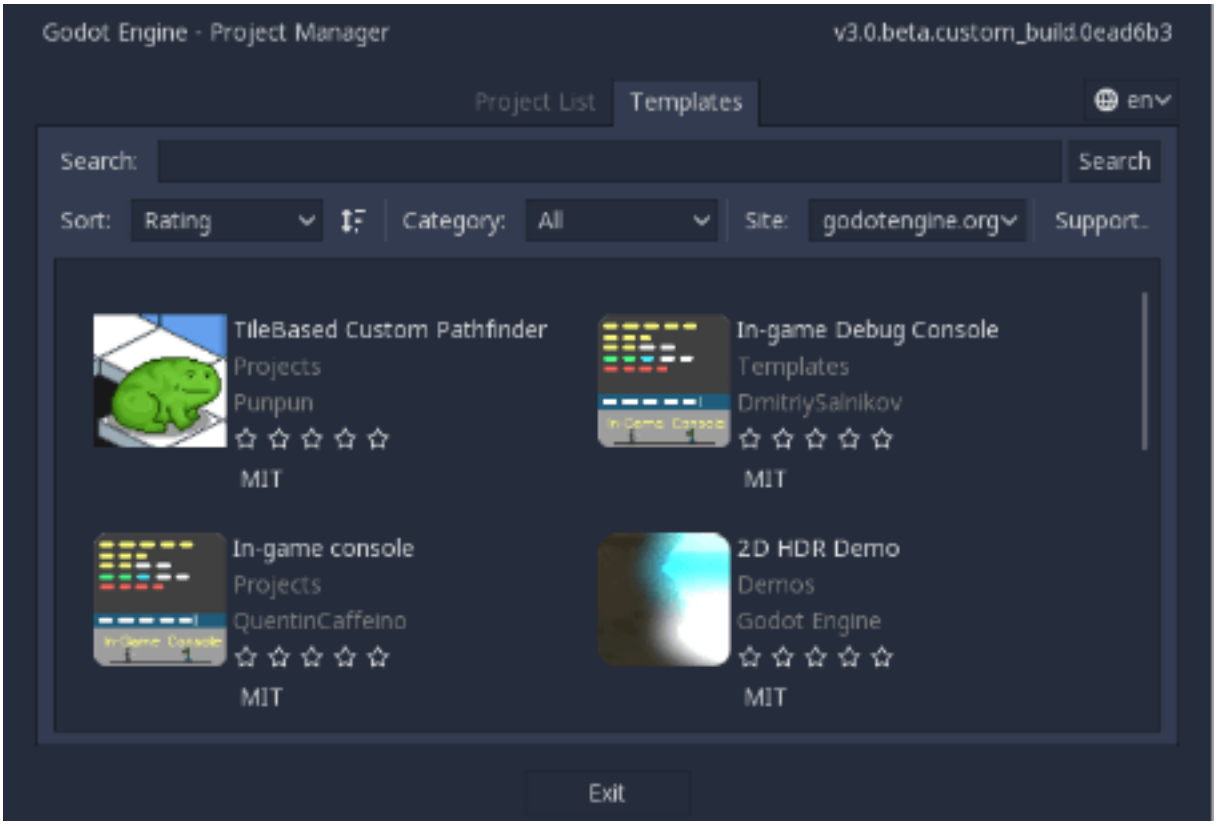
When you launch Godot, the first window you'll see is the Project Manager. It lets you create, remove, import or play game projects.



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

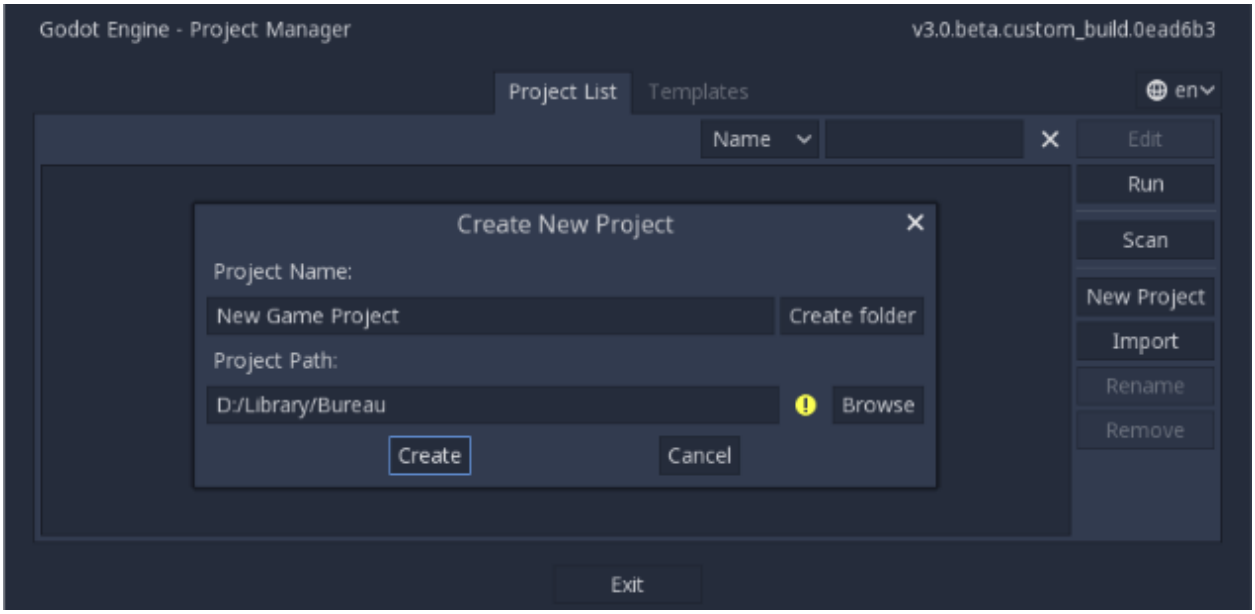


From the **Templates** tab you can download open source project templates and demos to help you get started faster.

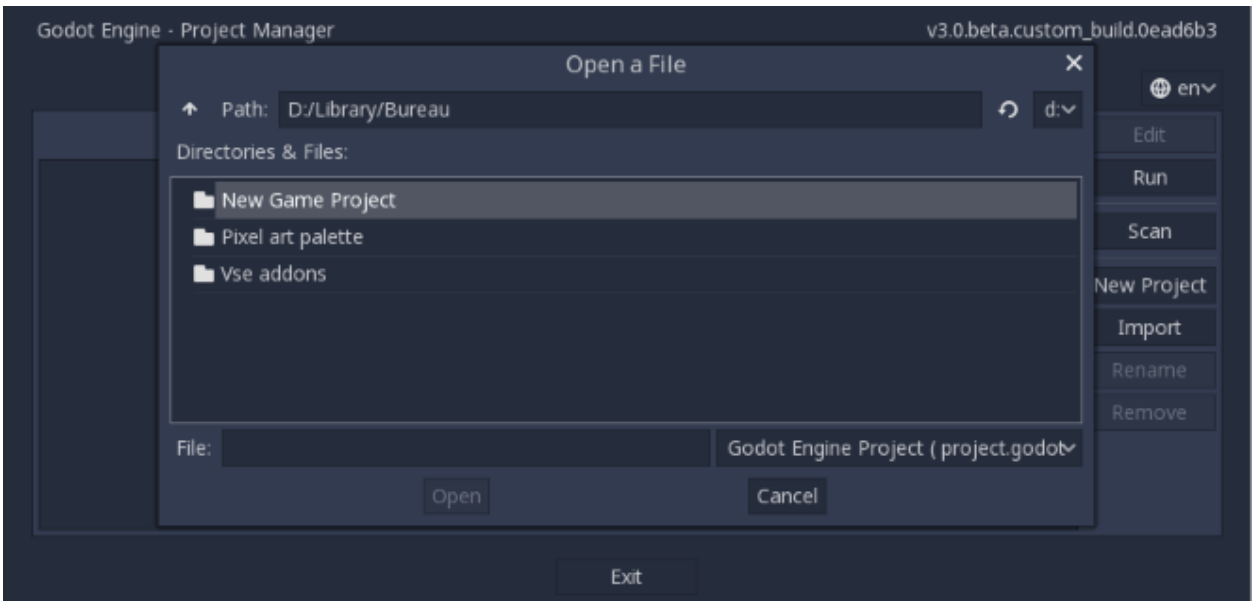


Create or import a project

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.

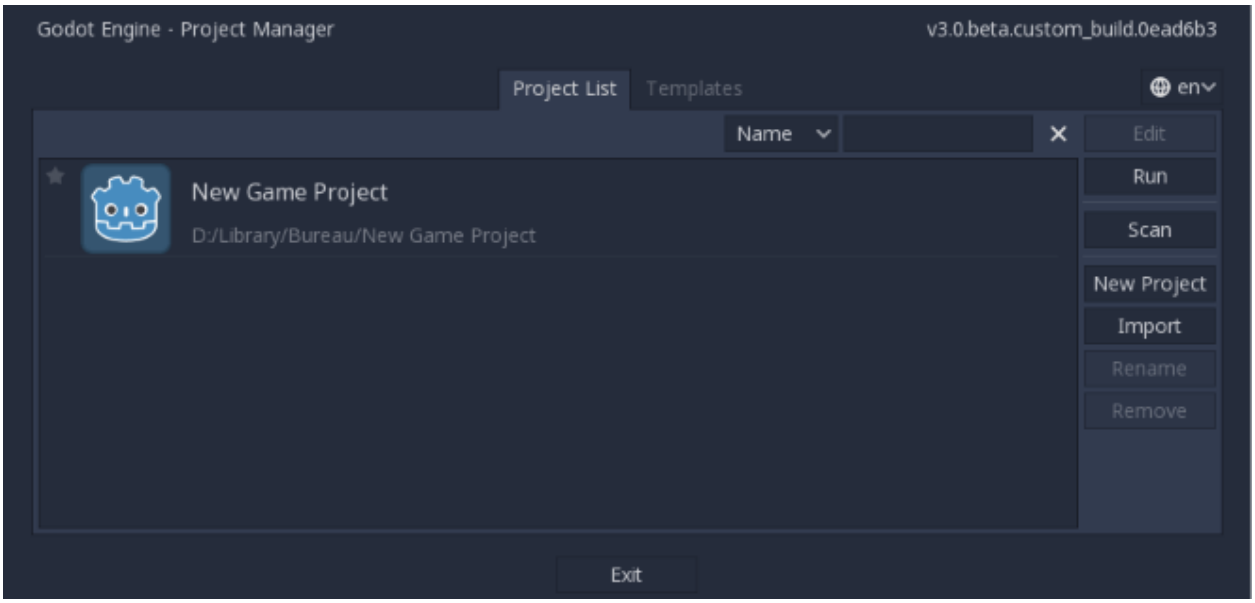


Click the Browse button to open Godot's file browser and pick a location or type the folder's path in the Project Path field.

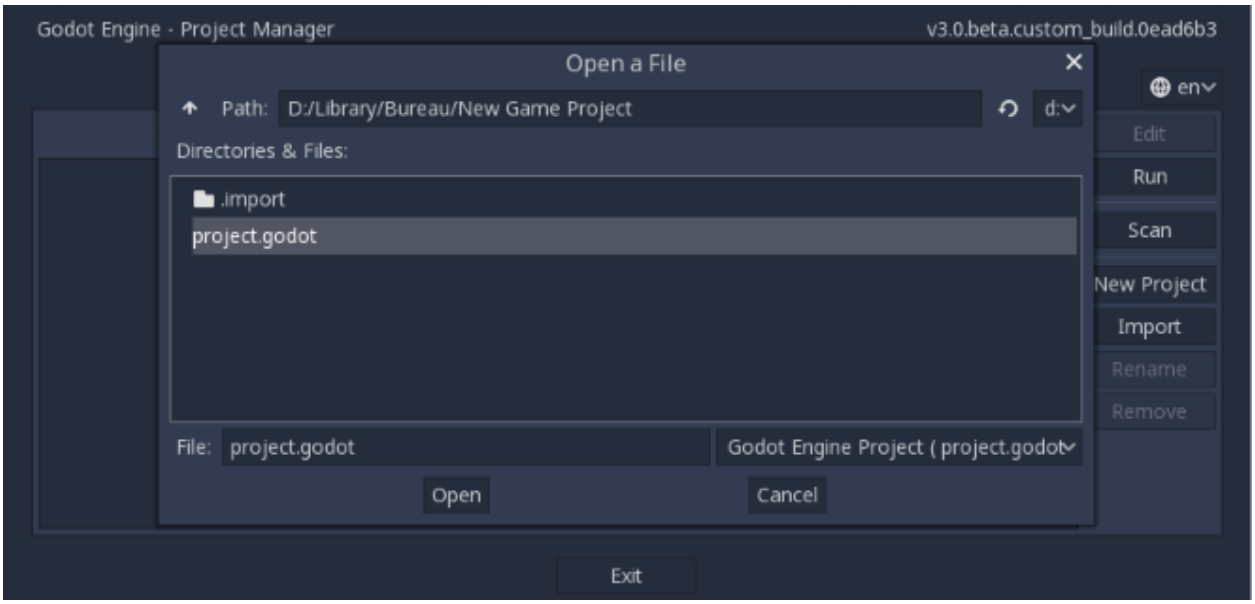


When you see the green tick on the right, it means the engine detects an empty folder and you may click `Create`. Godot will create the project for you and open it in the editor.

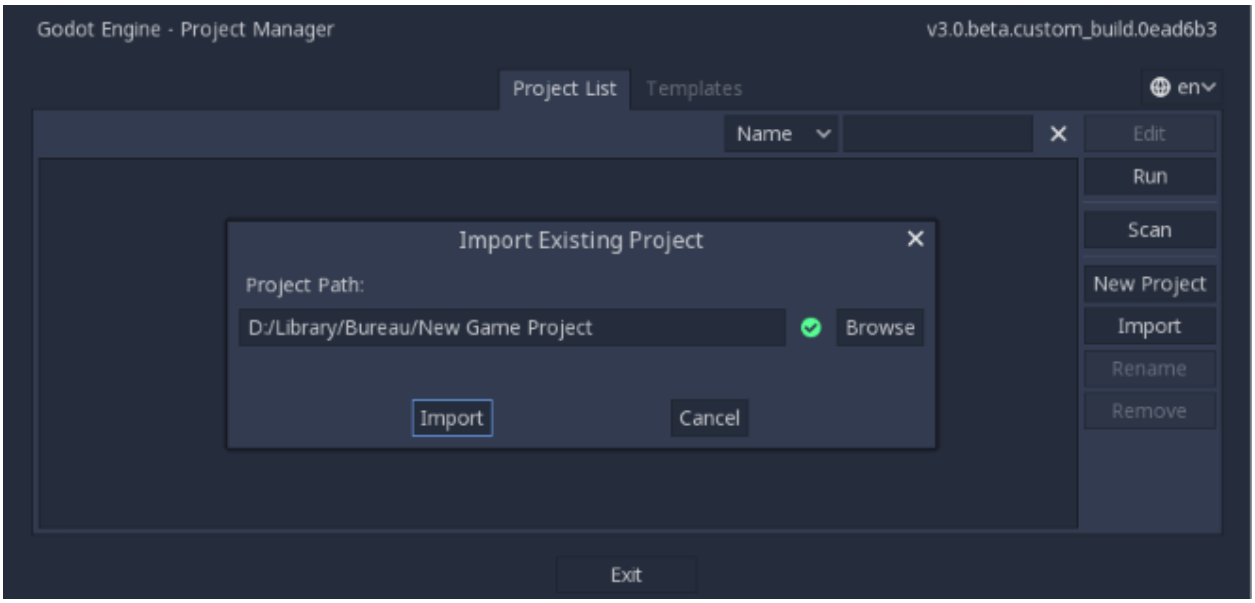
The next time you'll open Godot, you'll see your new project in the list. Double click on it to open it in the editor.



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.

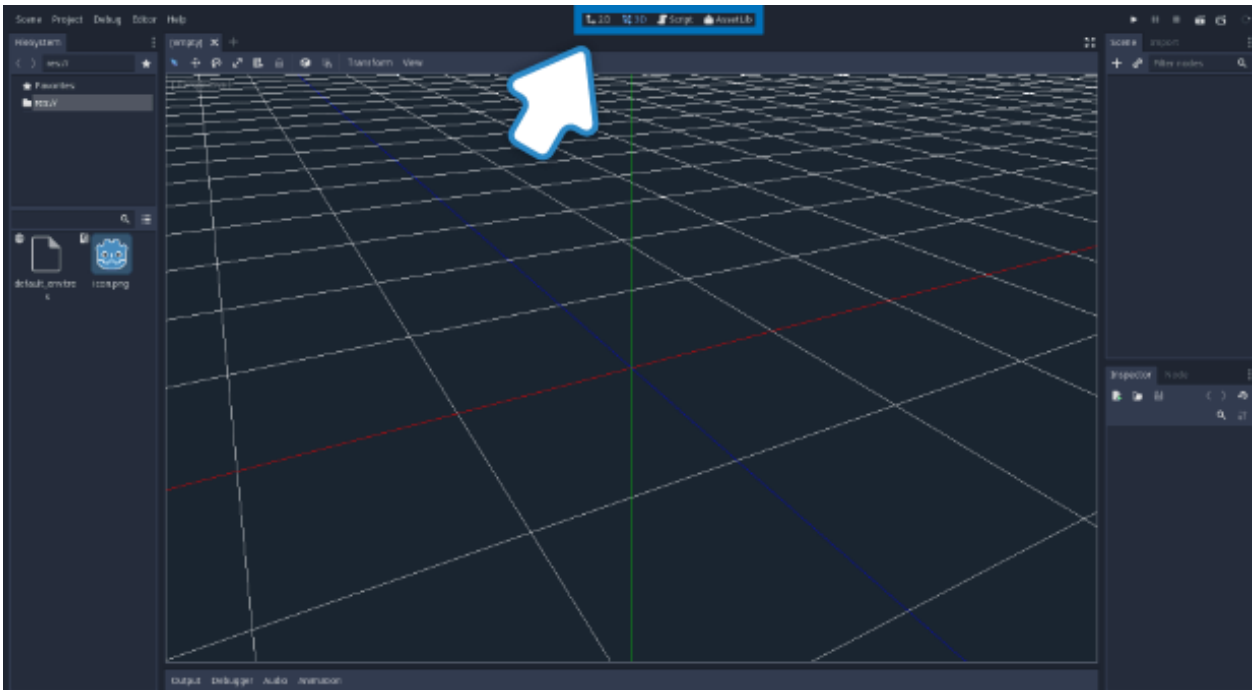


When the folder path is correct you'll see a green checkmark.

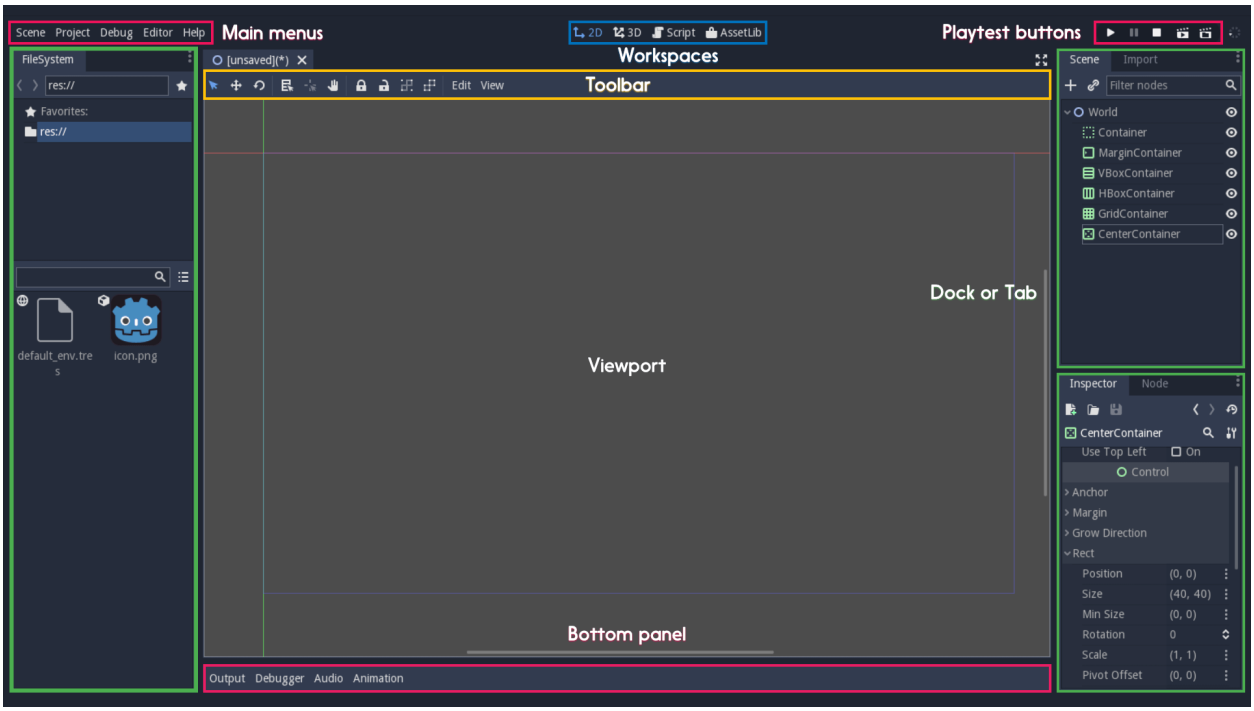


2.1.2 Your first look at Godot's editor

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.

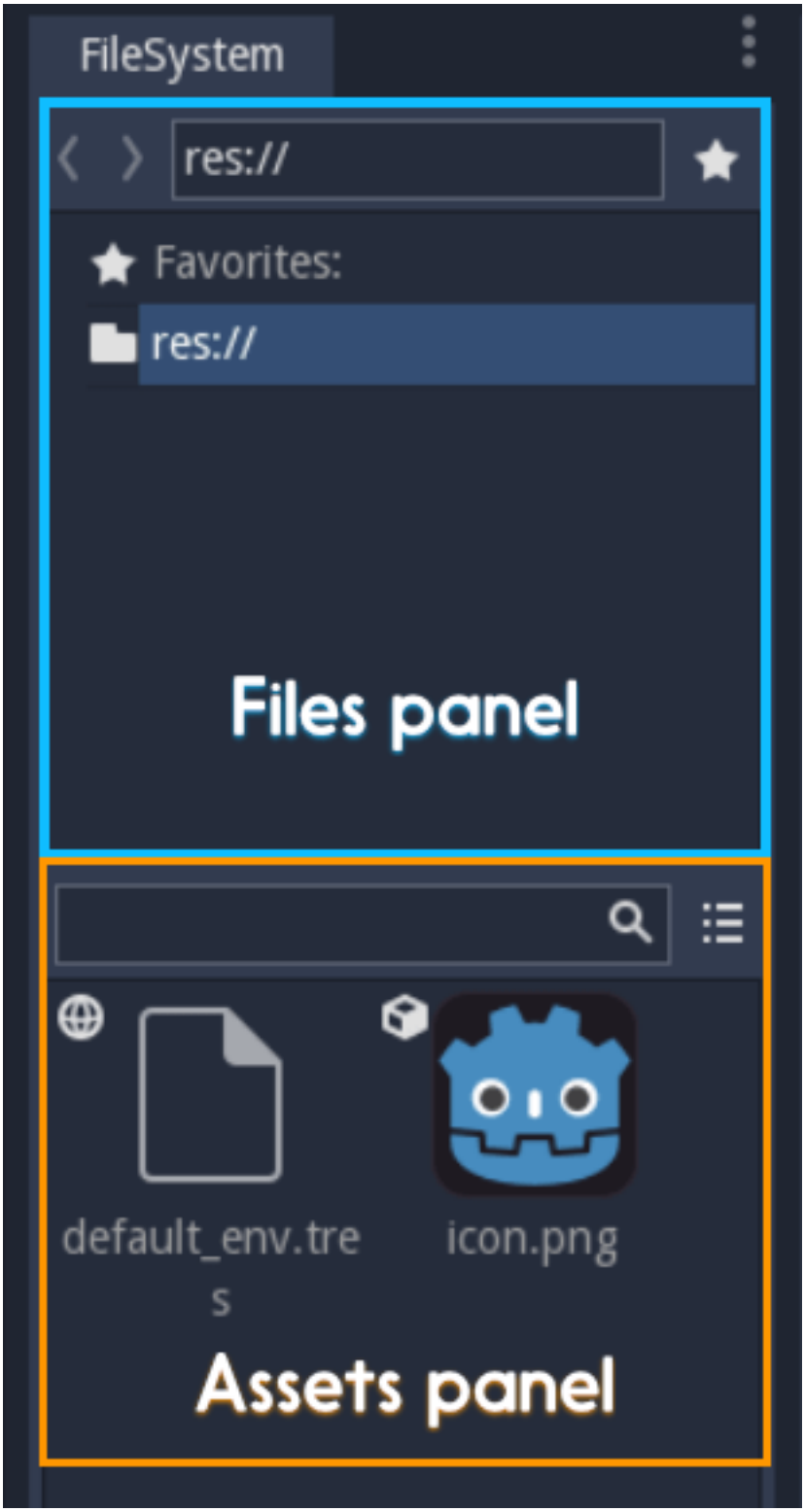


Now you should see this interface, with empty docks on the right side.

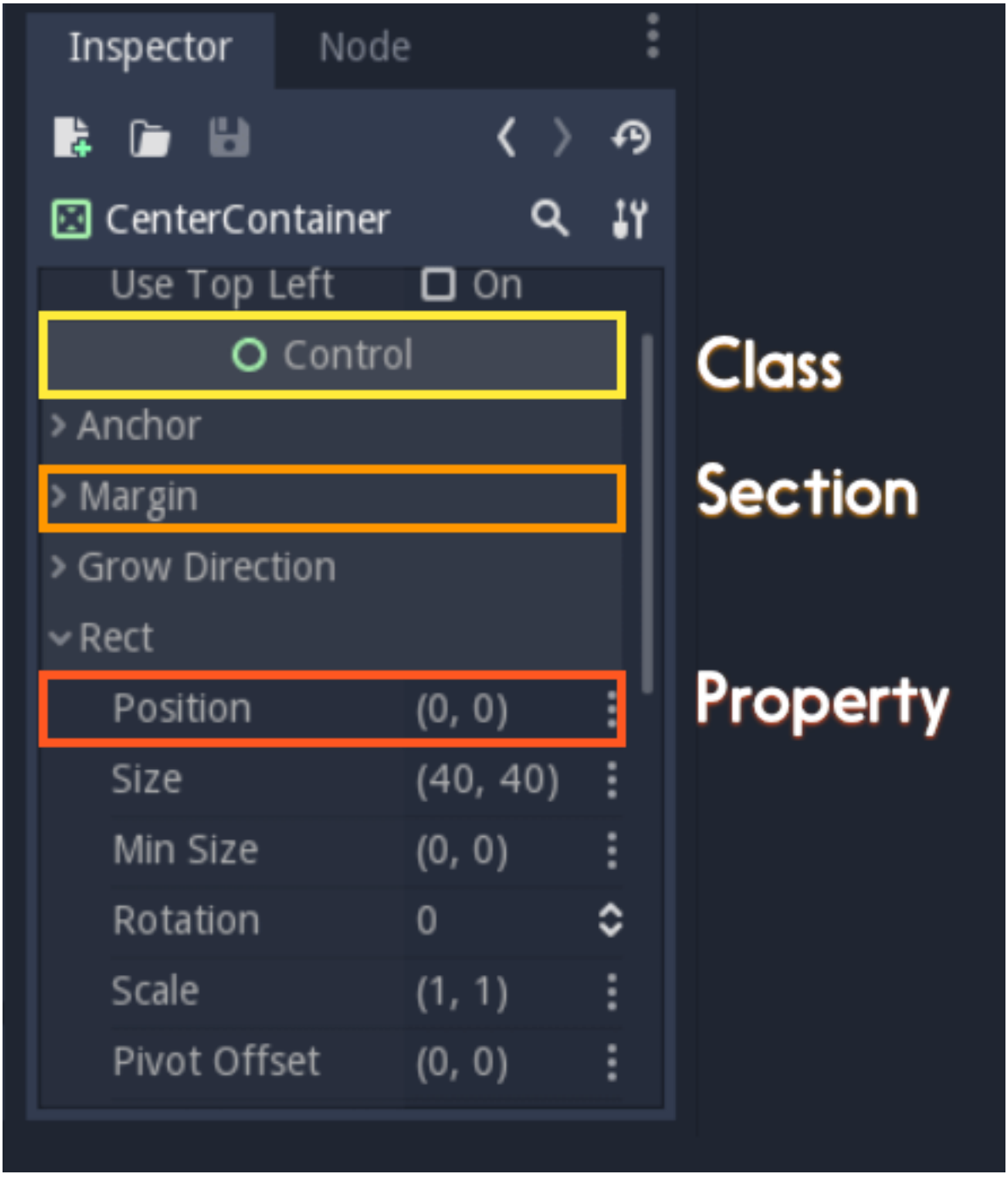


At the top, from left to right, you can see the **main menus**, the **workspaces**, and the **playtest buttons**.

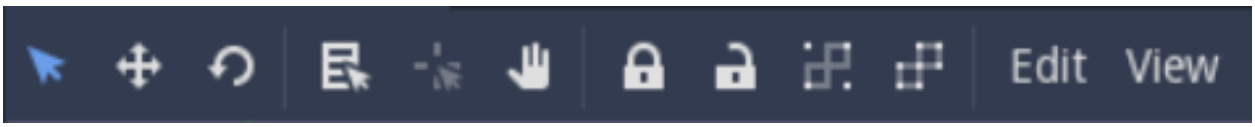
On the left side you have the **FileSystem dock**, where you'll manage your project files and assets.



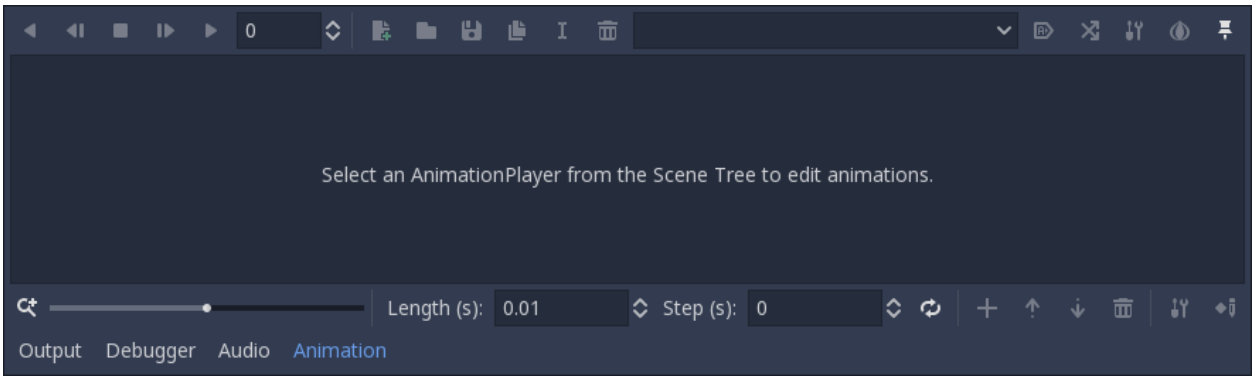
On the right side you'll find the **Scene dock** that lists the active scene's content and the **Inspector** in the bottom right corner.



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.



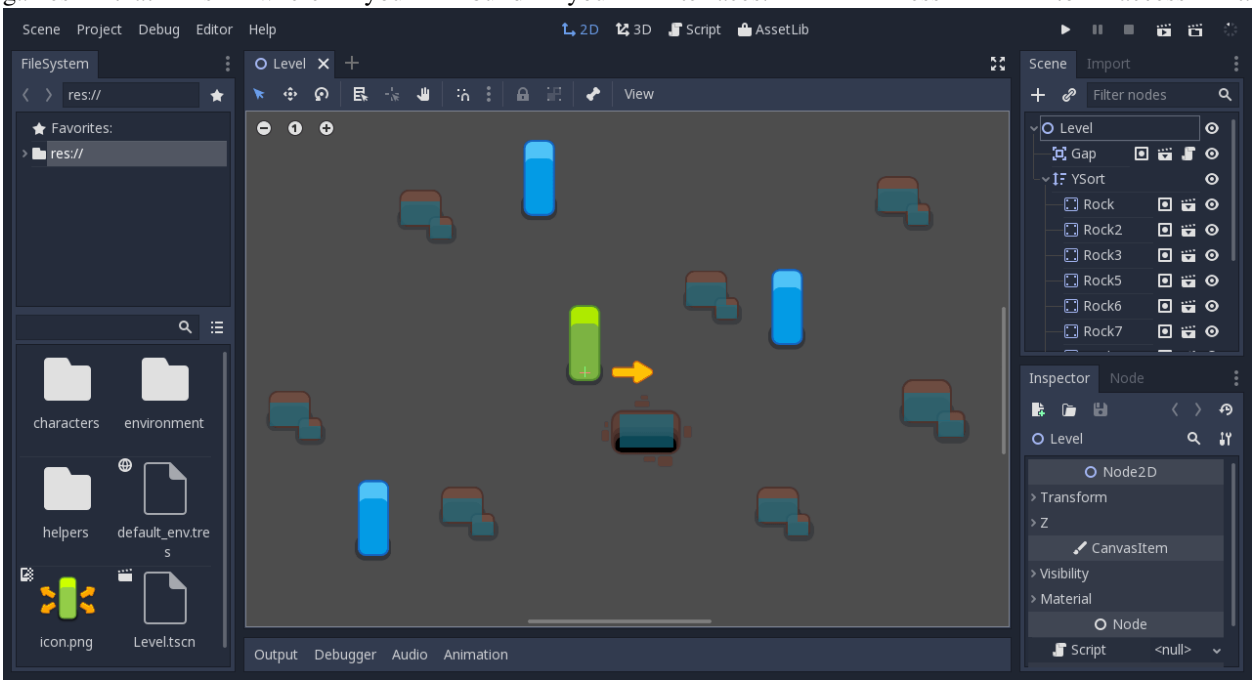
The **Bottom Panel** is the host for the debug console, the animation editor, the audio mixer... They are wide and can take precious space. That's why they're folded by default.



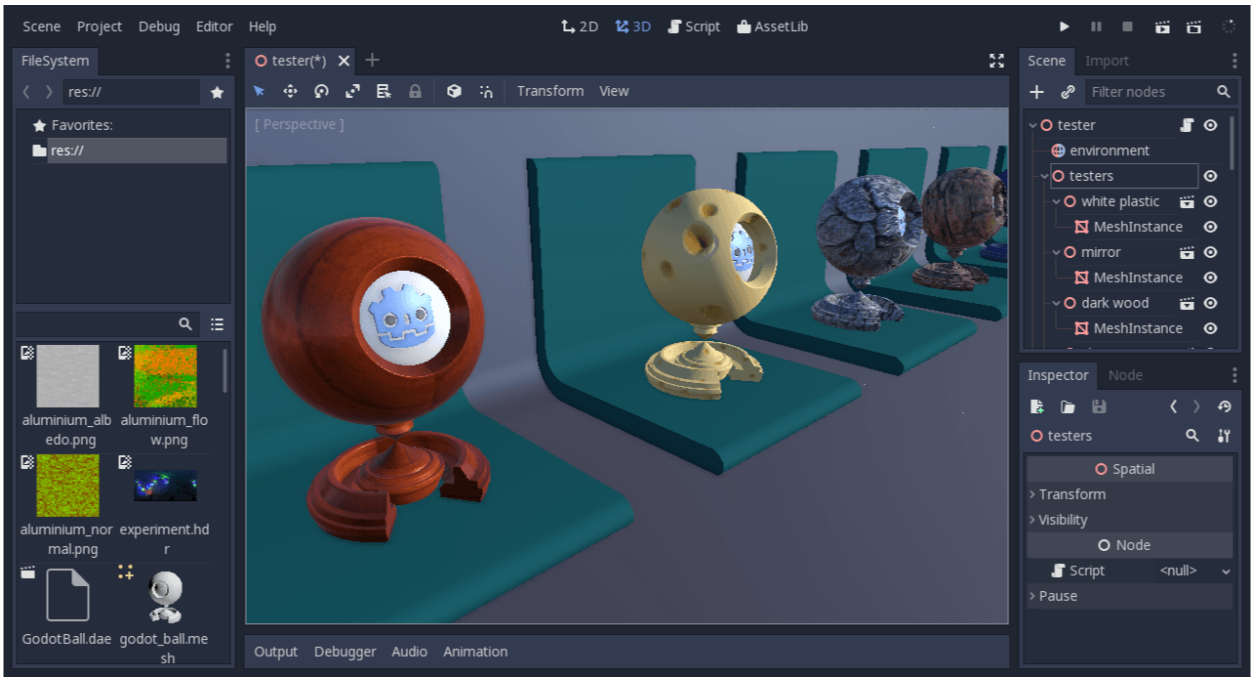
2.1.3 The workspaces

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

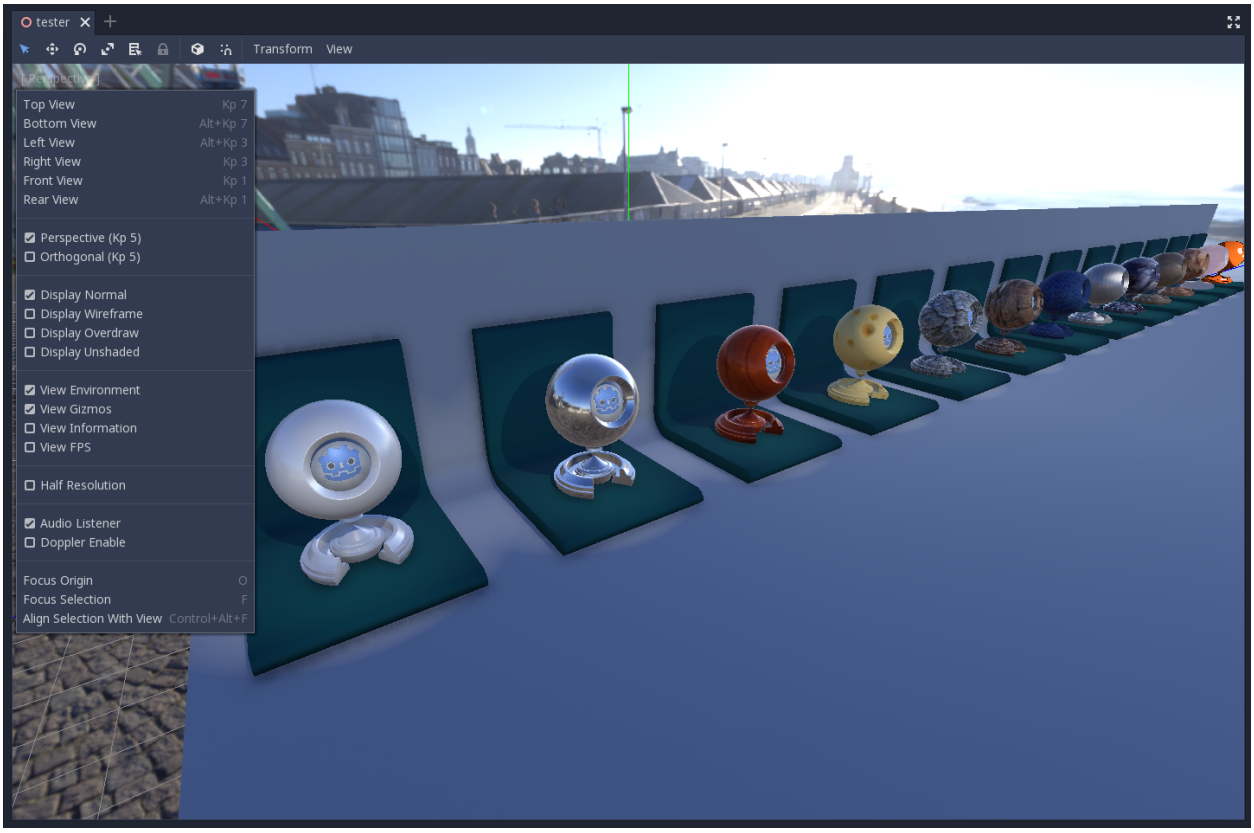
You'll use the **2D workspace** for all types of games. On top of 2D games that is where you'll build your interfaces. Press F1 to access it.



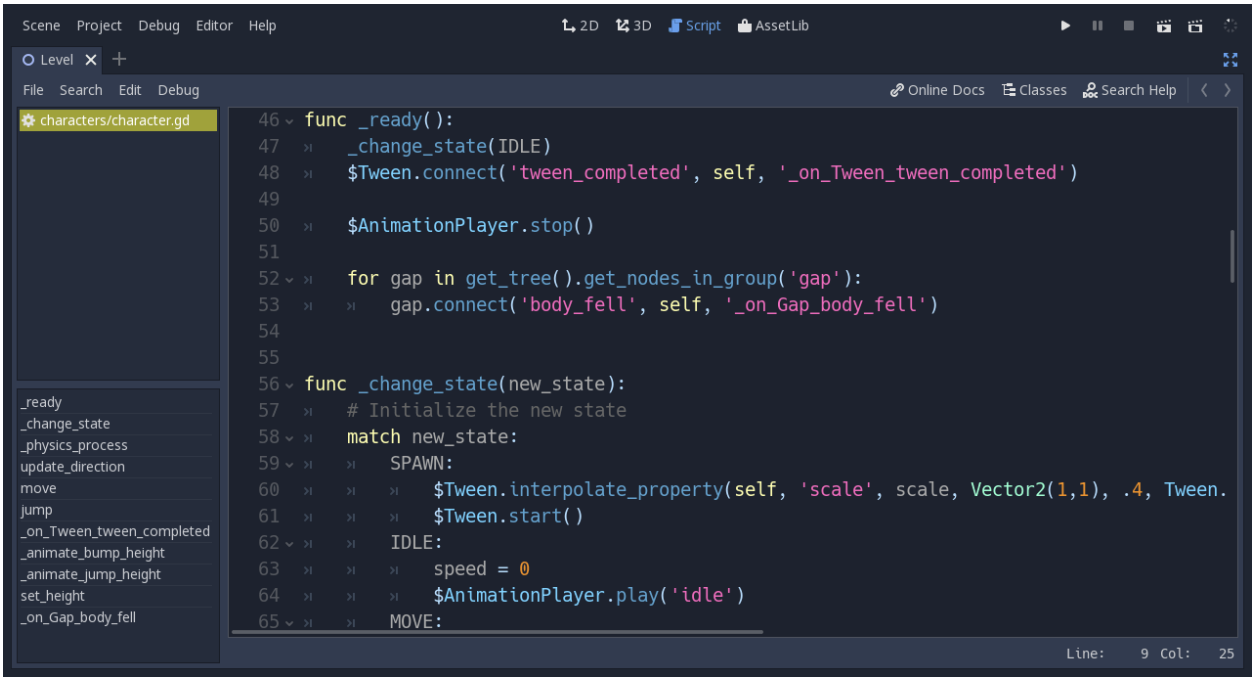
In the **3D workspace**, you can work with meshes, lights, and design levels for 3D games. Press F2 to access it.



Notice the text [perspective] under the toolbar, it is a button that opens a list of options related to the 3D viewport.



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.

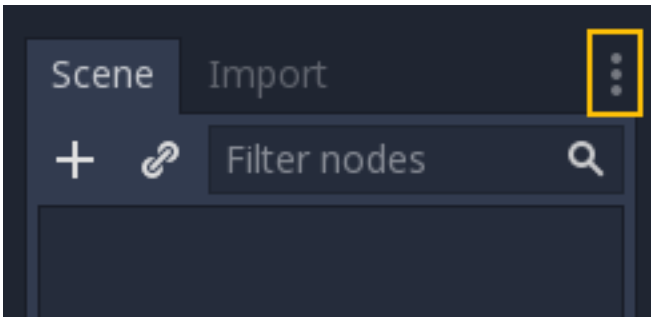
2.1.4 Modify the interface

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.

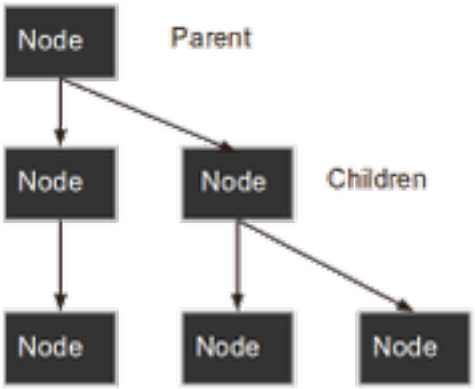
Making games in Godot feels pretty much the same way. Using the engine feels like being in a kitchen. In this kitchen, *nodes* are like a refrigerator full of fresh ingredients with which to cook.

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.

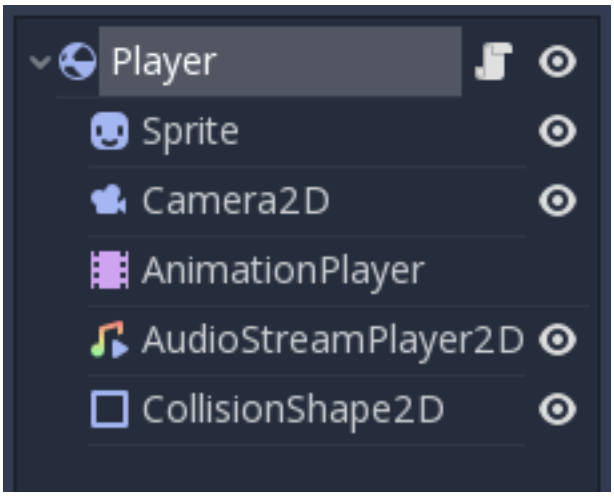


The last one is important. Nodes can have other nodes as children. When arranged in this way, the nodes become a **tree**.

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).

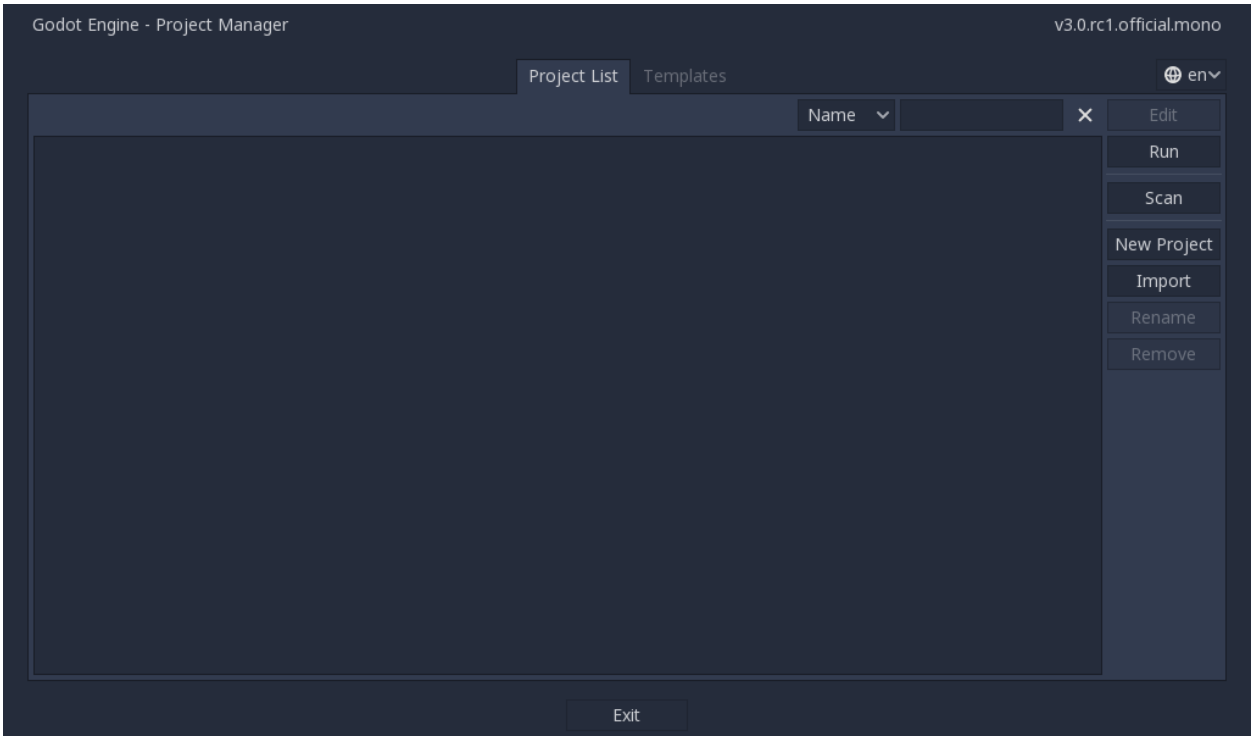
Running a game means running a scene. A project can contain several scenes, but for the game to start, one of them must be selected as the main scene.

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.

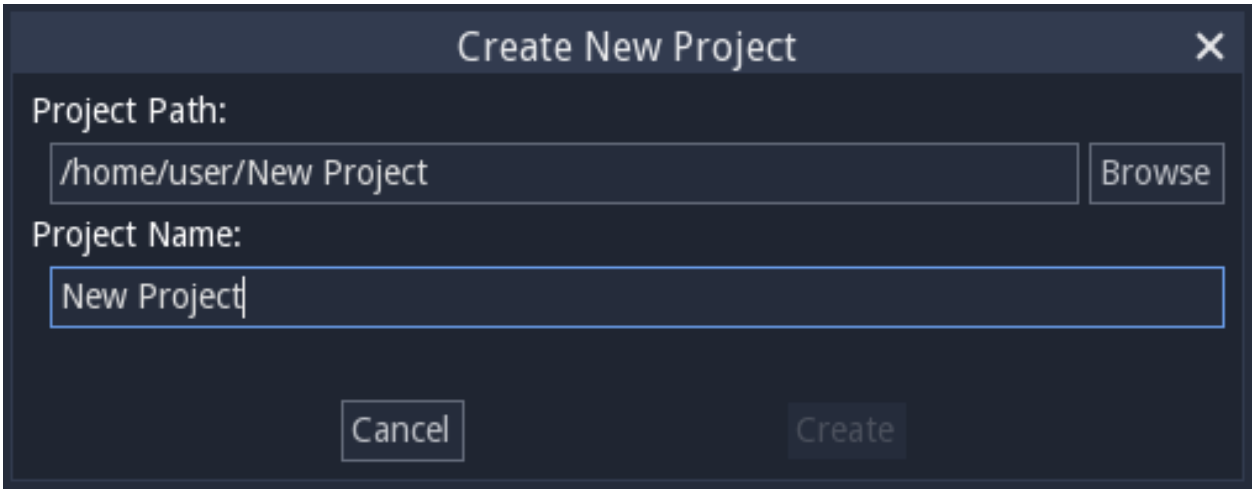
2.2.4 Creating a new project

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.

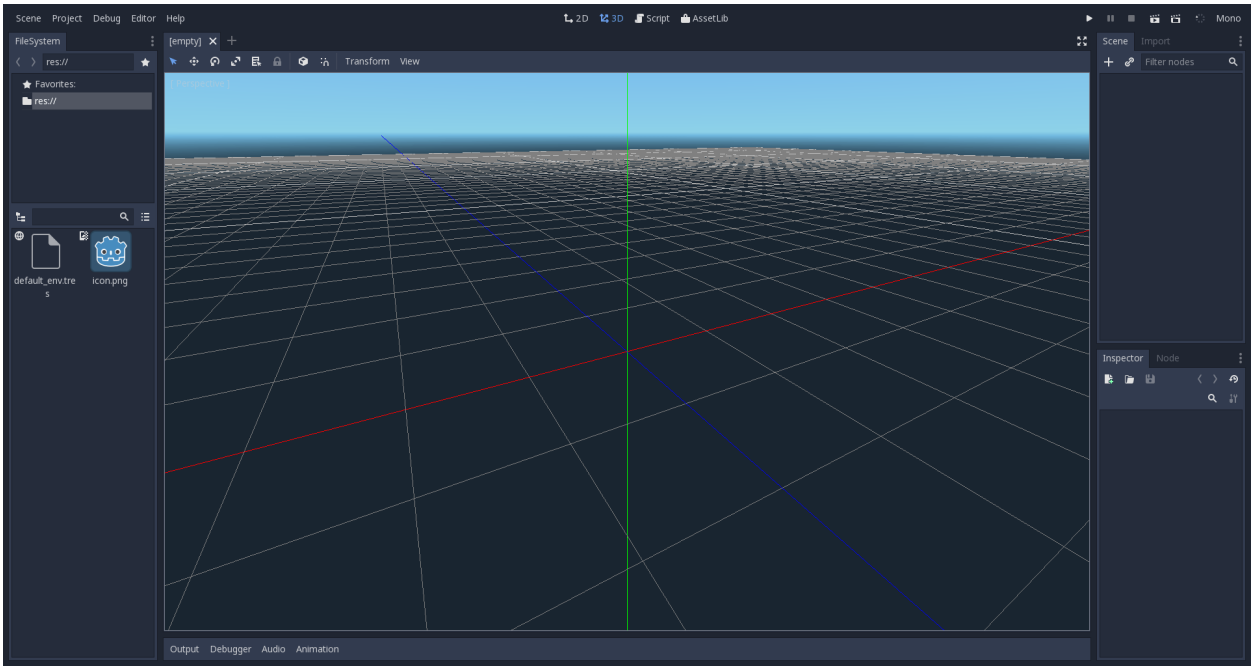


To create a new project, click the "New Project" option. Choose and create a path for the project and specify the project name "New Project":



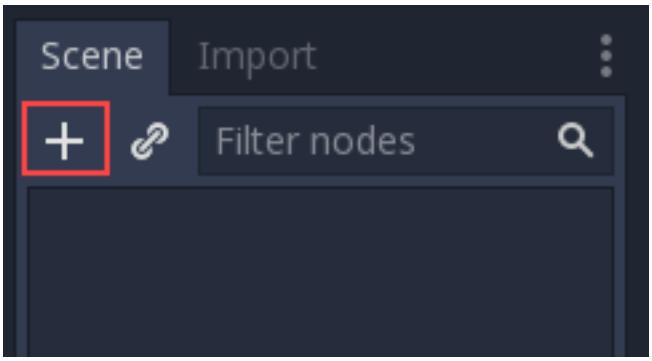
2.2.5 Editor

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

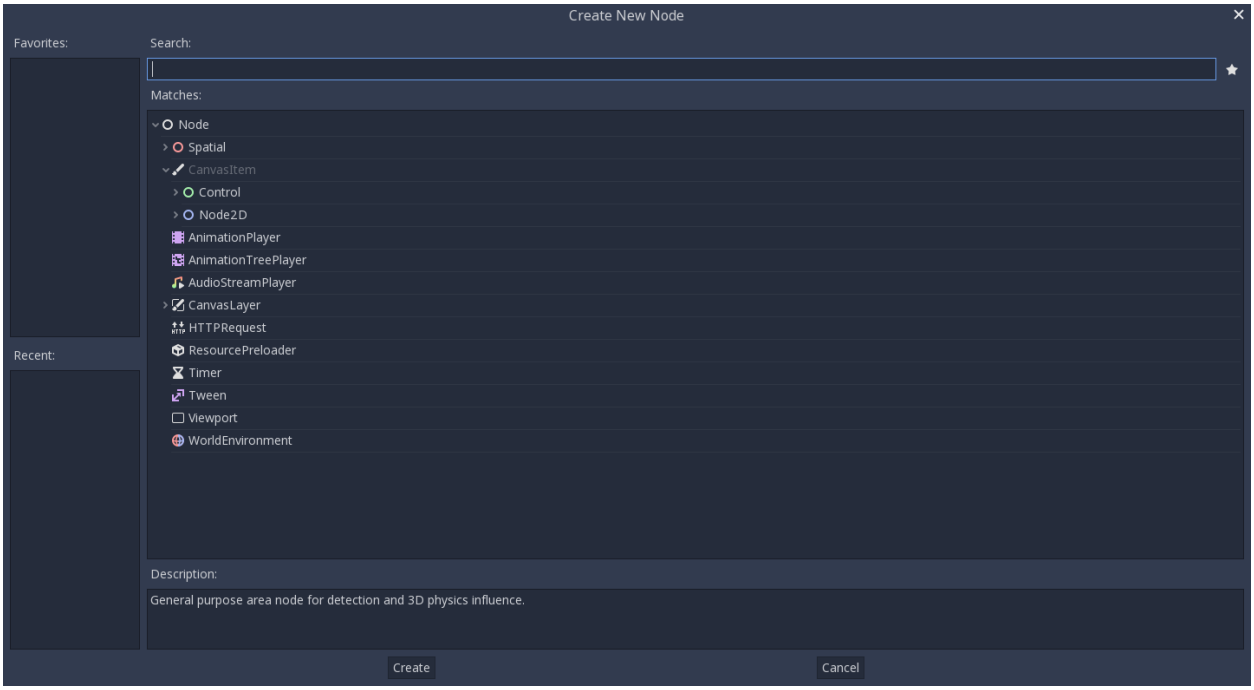


As mentioned before, making games in Godot feels like being in a kitchen, so let's open the refrigerator and add some fresh nodes to the project. We'll begin with a "Hello World!" message that we'll put on the screen.

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

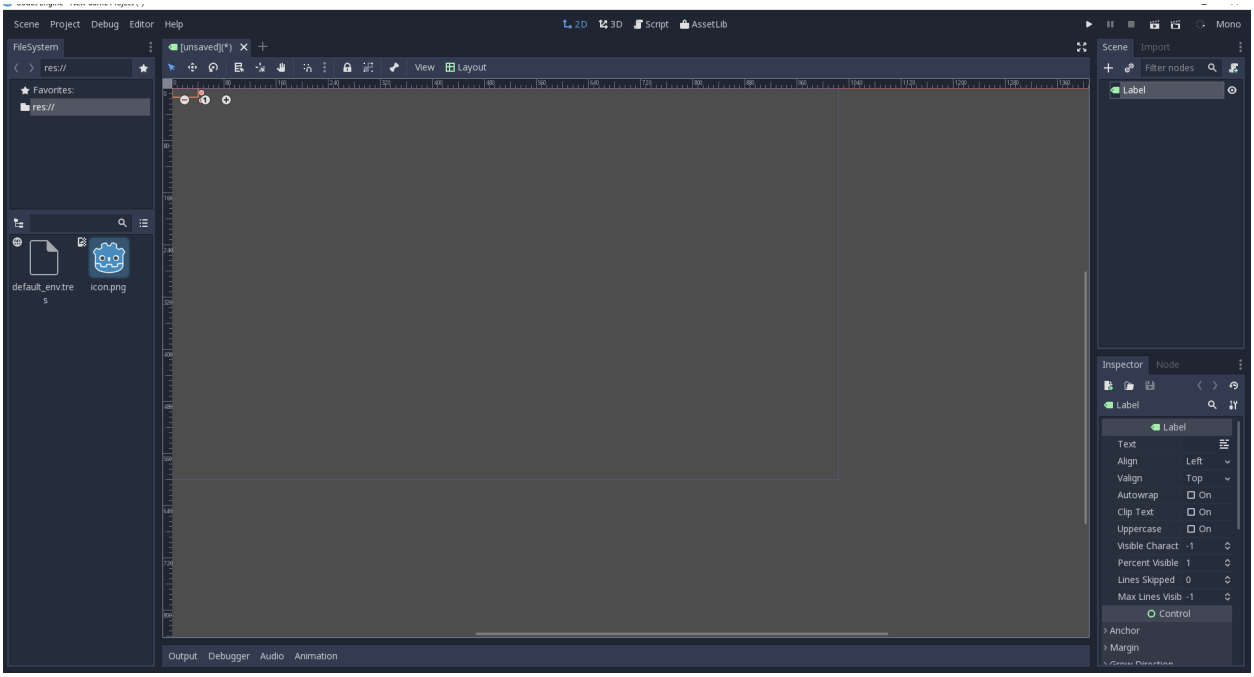


This will open the Create Node dialog, showing the long list of nodes that can be created:



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

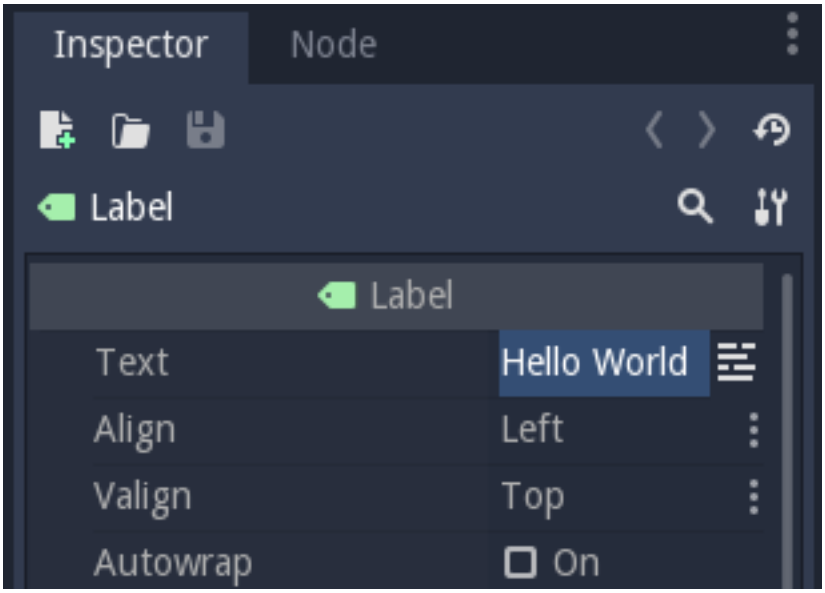
And finally, create the Label! A lot happens when Create is pressed:



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.

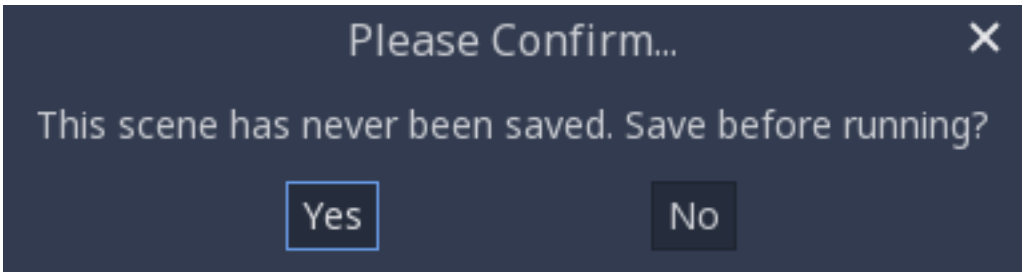
The node appears in the scene tree editor (box in the top right corner), and the label properties appear in the Inspector (box in the bottom right corner).

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

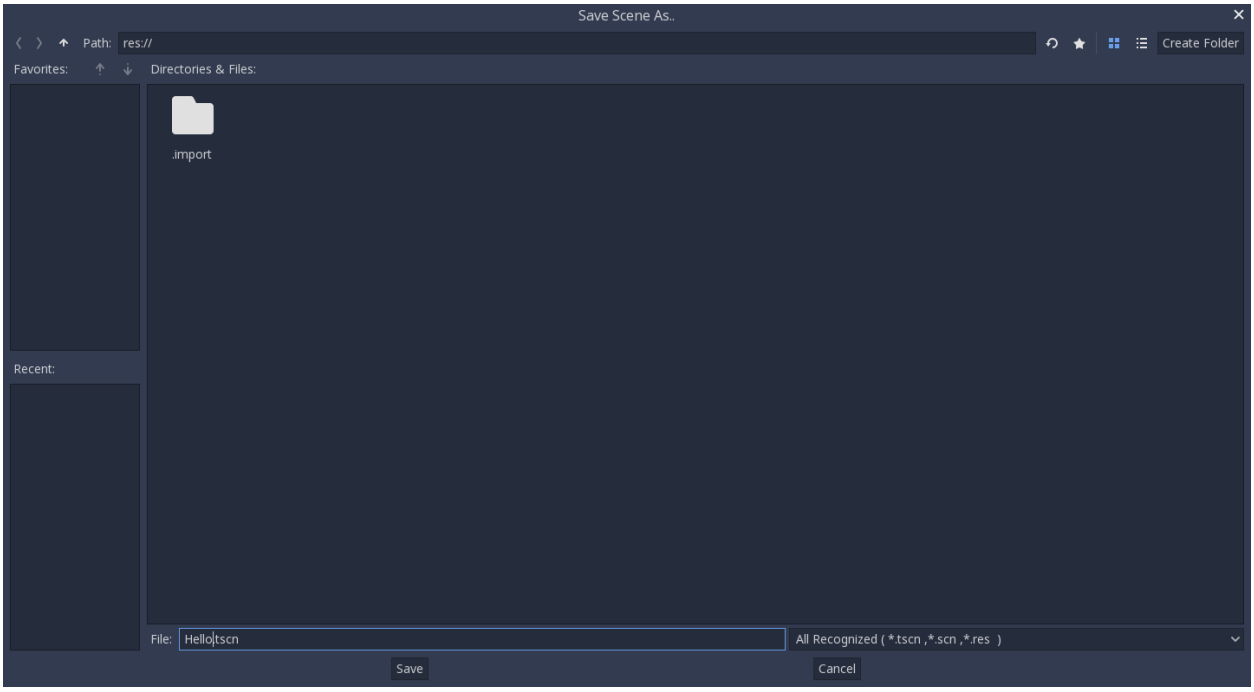


Ok, everything's ready to run the scene! Press the PLAY SCENE Button on the top bar (or hit F6):

Aaaand... Oops.

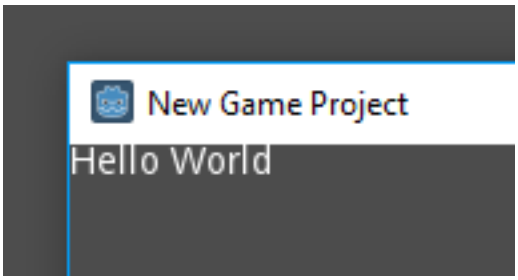


Scenes need to be saved to be run, so save the scene to something like hello.tscn in Scene -> Save:



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!

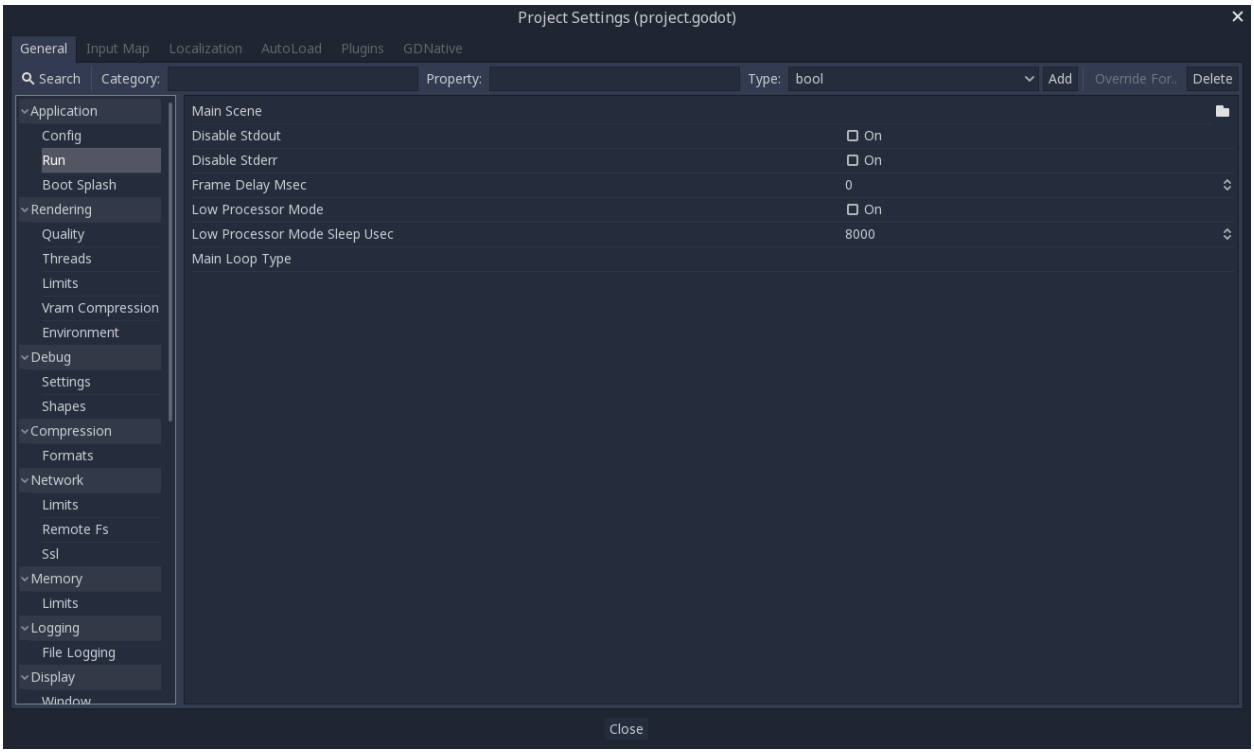
2.2.6 Configuring the project

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 *Project-Settings* singleton.

2.2.7 To be continued...

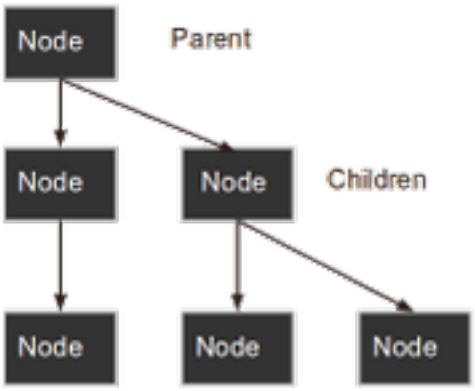
This tutorial talked about "scenes and nodes", but so far there has been only *one* scene and *one* node! Don't worry, the next tutorial will expand on that...

2.3 Instancing

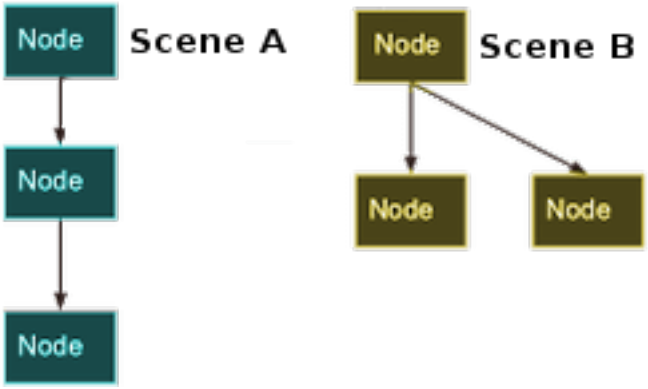
2.3.1 Introduction

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 of scenes. This provides you with a powerful tool that helps you organize the different components of your game.

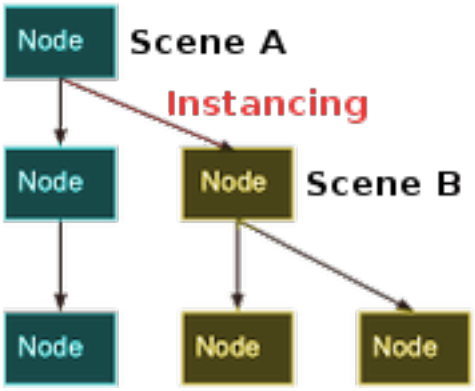
In *Scenes and nodes* you learned that a scene is a collection of nodes organized in a tree structure, with a single node as the tree root.



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 as if it were any other node.

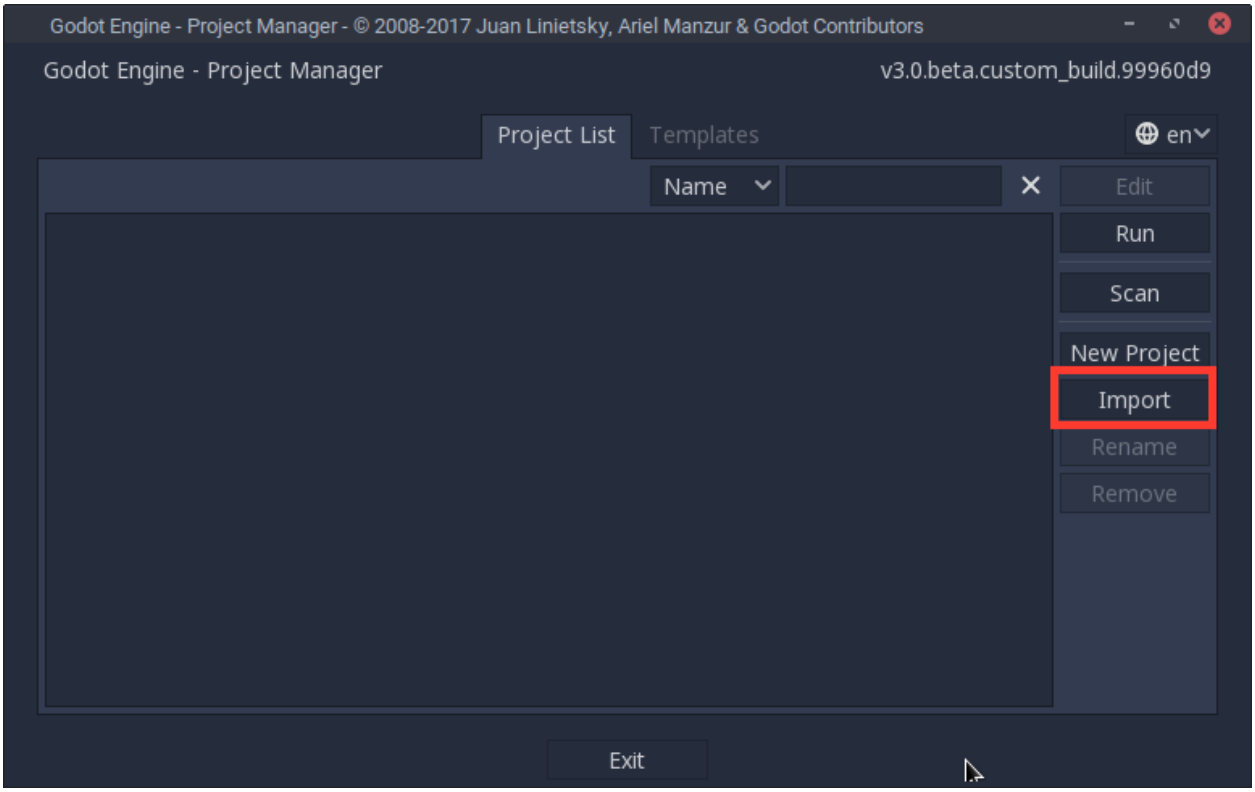


In the above picture, Scene B was added to Scene A as an instance.

2.3.2 Instancing By Example

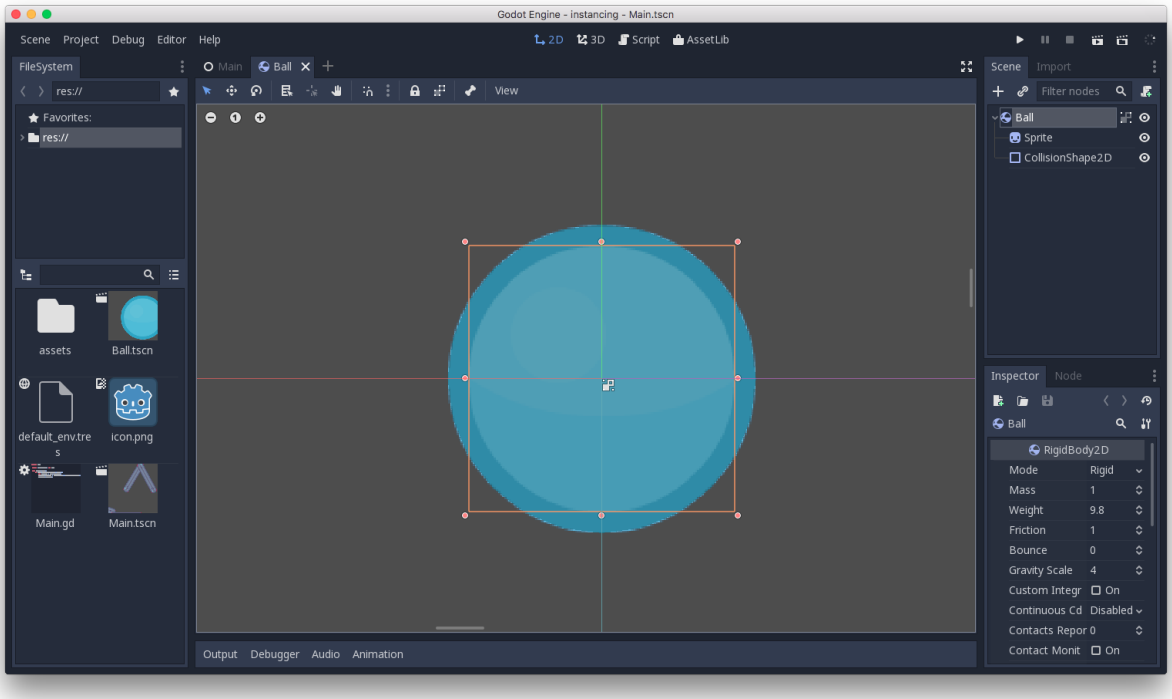
To learn how instancing works, let's start by downloading a sample project: `instancing.zip`.

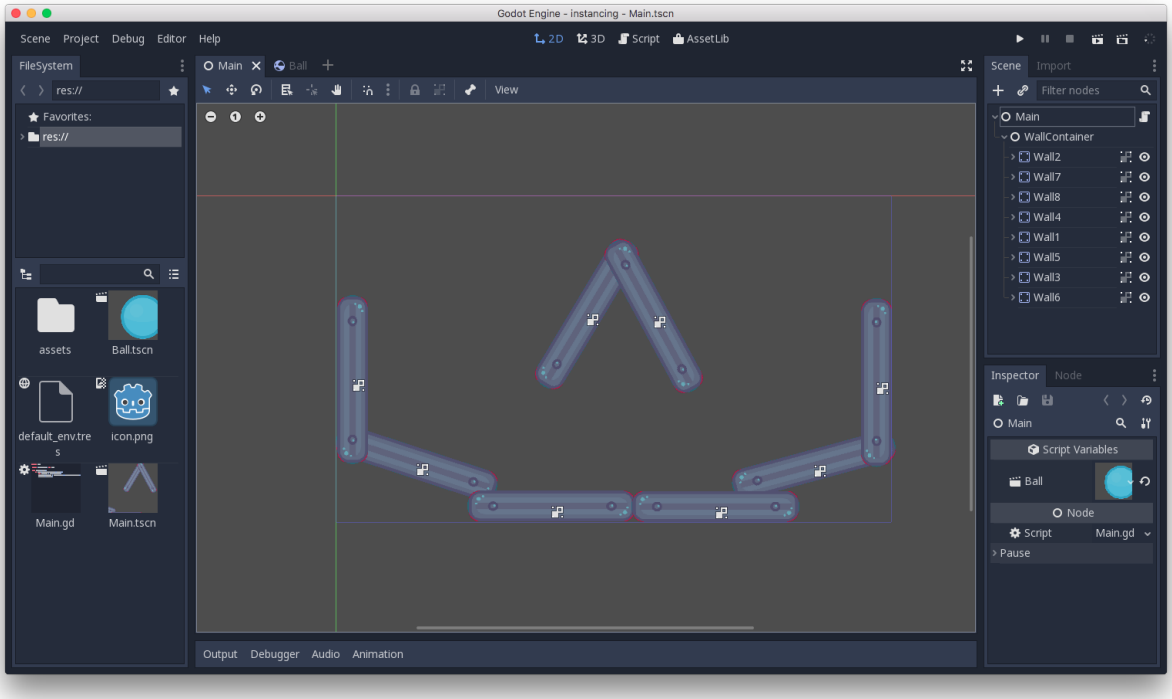
Unzip this project anywhere you like. Then open Godot and add this project to the project manager using the 'Import' button:



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.

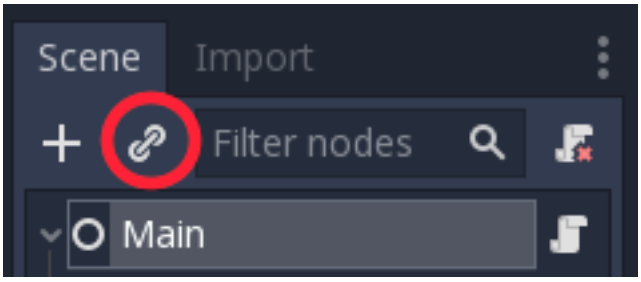
This project contains two scenes: "Ball.tscn" and "Main.tscn". The ball scene uses a *RigidBody2D* to provide physics behavior while the main scene has a set of obstacles for the ball to collide with (using *StaticBody2D*).



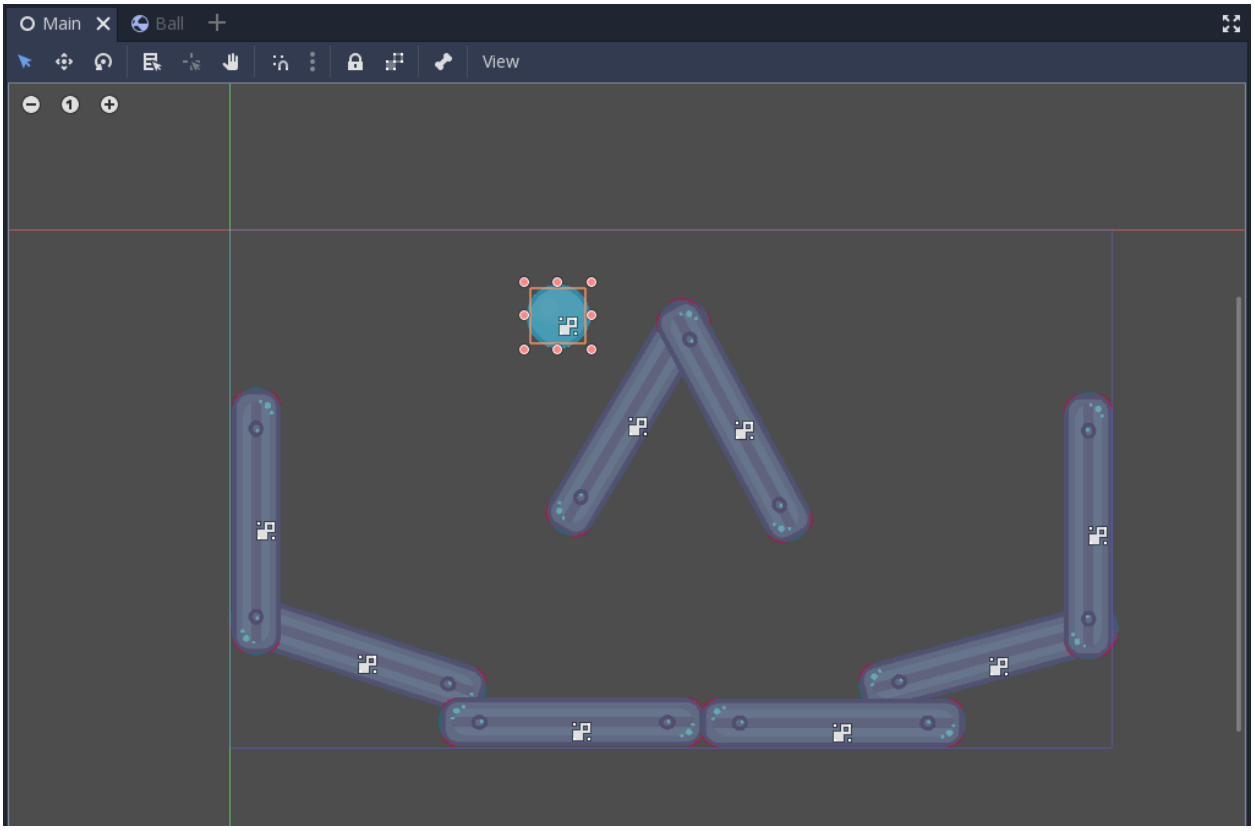


Open the `Main` scene, and then select the root node:

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.



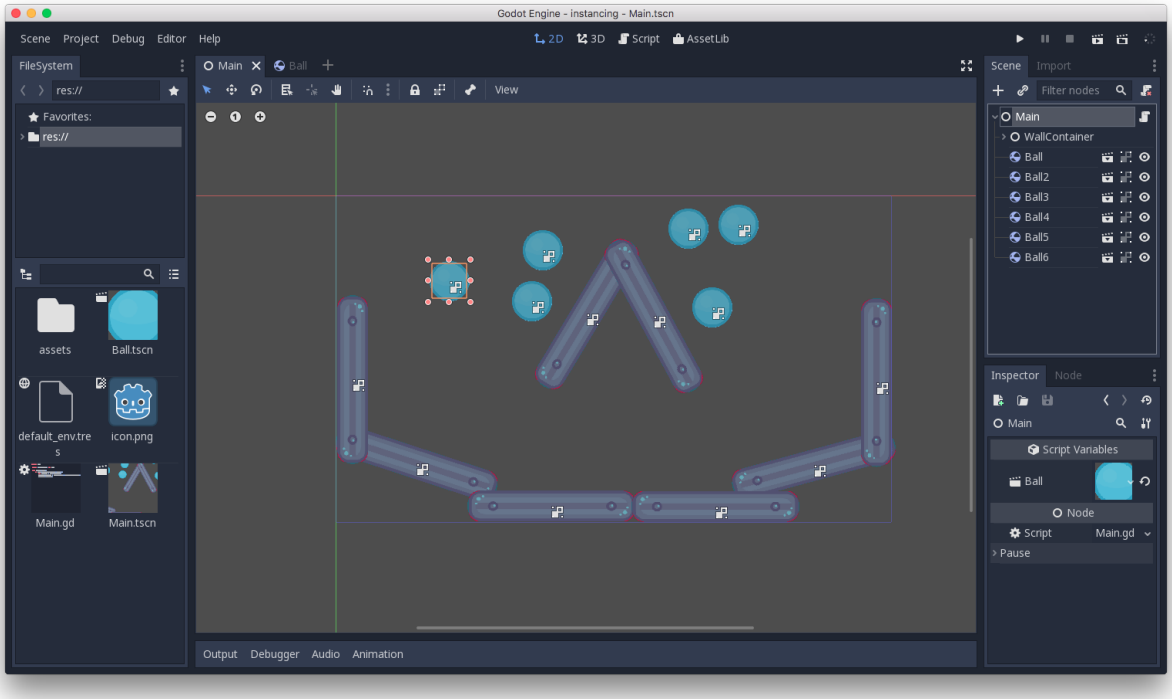
The ball will be placed at the top-left corner of the screen area (this is `(0, 0)` in screen coordinates). Click and drag the ball somewhere near the top-center of the scene:



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 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:

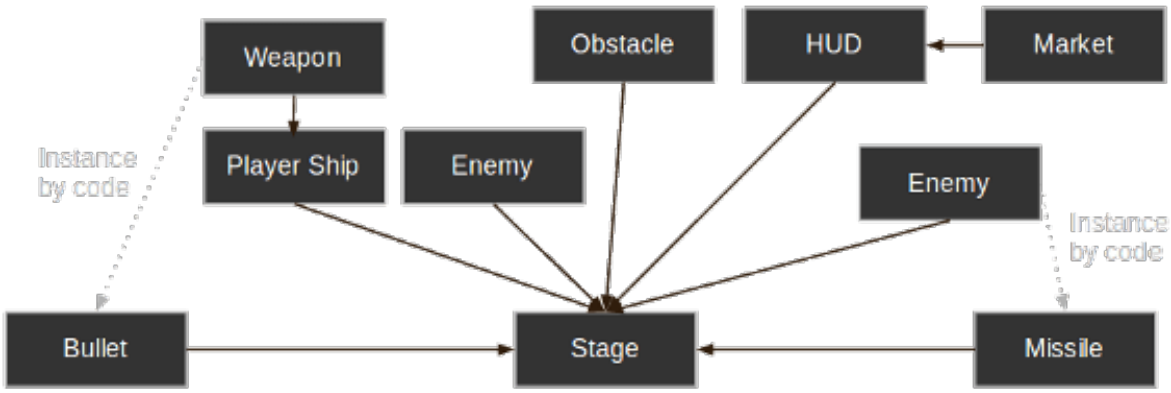
- The ability to subdivide scenes and make them easier to manage.
- A more flexible alternative to prefabs (and much more powerful given that instances can be nested).
- A way to organize and embed complex game flows or even UIs (in Godot, UI Elements are nodes, too).

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:

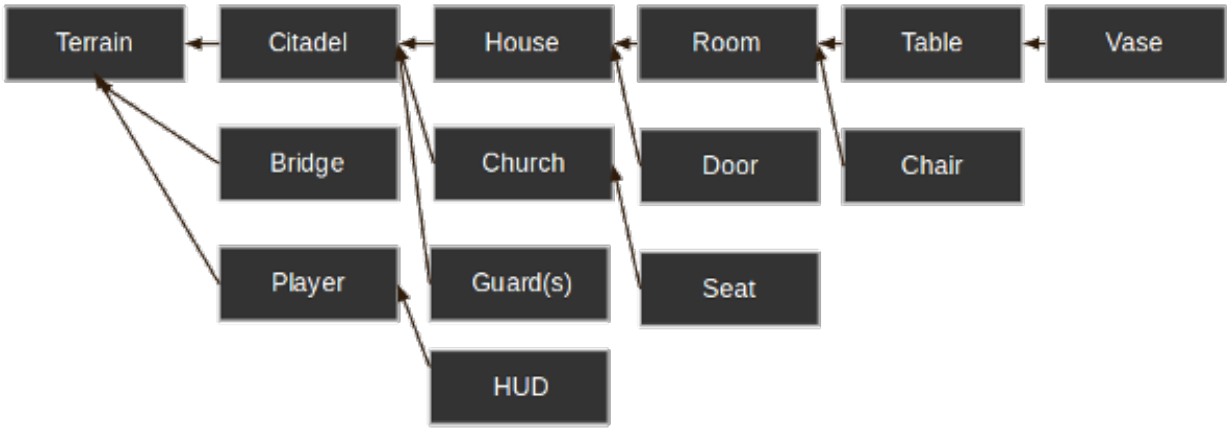


You can come up with a diagram like this for almost any kind of game. Write down the parts of the game that you can visualize, and then add arrows to represent ownership of one component by another.

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 go on to the next tutorial!

2.5 Scripting

2.5.1 Introduction

Before Godot 3.0, the only choice for scripting a game 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 supporting languages more difficult.

The "Main" languages in Godot, though, are GDScript and VisualScript. The main reason to choose them is their level of integration with Godot, as this makes the experience smoother; both have 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, go with C# and C++ instead.

GDScript

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

- It's simple, elegant, and designed to be familiar for 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 for nodes, signals, and many other items pertaining to the scene being edited.
- Has vector types built-in (such as Vectors, transforms, etc.), making it efficient for heavy use of linear algebra.
- Supports multiple threads as efficiently as statically typed languages - one of the limitations that made us avoid VMs such as Lua, Squirrel, etc.
- Uses no garbage collector, so it trades a small bit of automation (most objects are reference counted anyway), by determinism.
- Its dynamic nature makes it easy to optimize sections of code in C++ (via GDNative) if more performance is required, all without recompiling the engine.

If you're undecided and have experience with programming, especially dynamically typed languages, go for GDScript!

VisualScript

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

Visual scripting is a great tool for non-programmers, or even for experienced developers who want to make parts of the code more accessible to others, like game designers or artists.

It can also be used by programmers to build state machines or custom visual node workflows - for example, a dialogue system.

.NET / C#

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

It has an excellent tradeoff between performance and ease of use, although one must be aware of its garbage collector.

C# is usually the best choice for companies. The large amount of programmers familiar with it means less time can be spent learning Godot and more time can be spent programming with it.

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, one of our brightest additions for the 3.0 release: GDNative allows scripting in C++ without needing 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 throughout an entire game, as other parts can be written in GDScript or Visual Script. However the API is clear and easy to use as it resembles, mostly, Godot's actual C++ API.

More languages can be made available through the GDNative interface, but keep in mind we don't have official support for them.

2.5.2 Scripting a scene

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

- Writing a script and attaching it to a node.
- Hooking up UI elements via signals.
- Writing a script that can access other nodes in the scene.

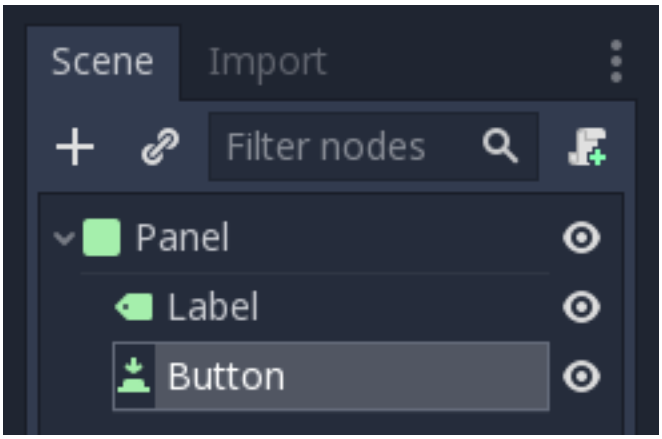
Before continuing, please make sure to read the *GDScript* reference. It's a language designed to be simple, and the reference is short, so it will not take more than a few minutes to get an overview of the concepts.

Scene setup

Use the "Add Child Node" dialogue accessed from the Scene tab (or by pressing `Ctrl+A`) to create a hierarchy with the following nodes:

- Panel
  - Label
  - Button

The scene tree should look like this:

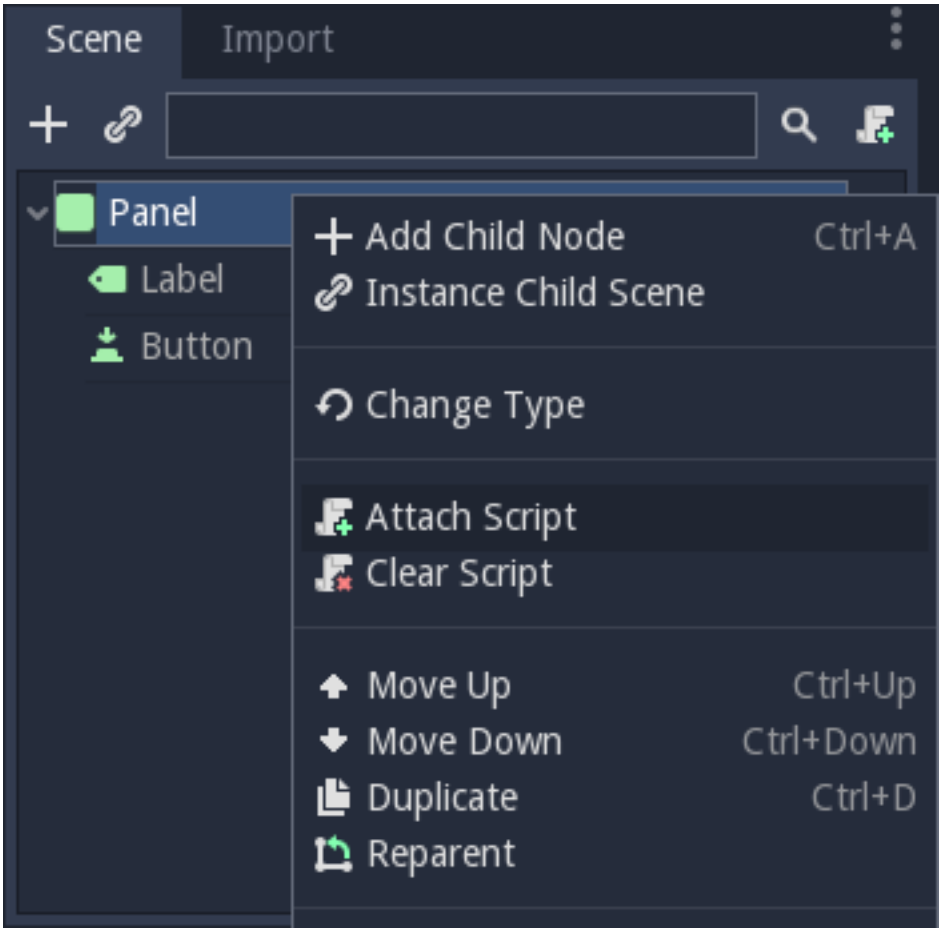


Use the 2D editor to position and resize the Button and Label so that they look like the image below. You can set the text from the Inspector tab.

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

Adding a script

Right click on the Panel node, then select "Attach Script" from the context menu:

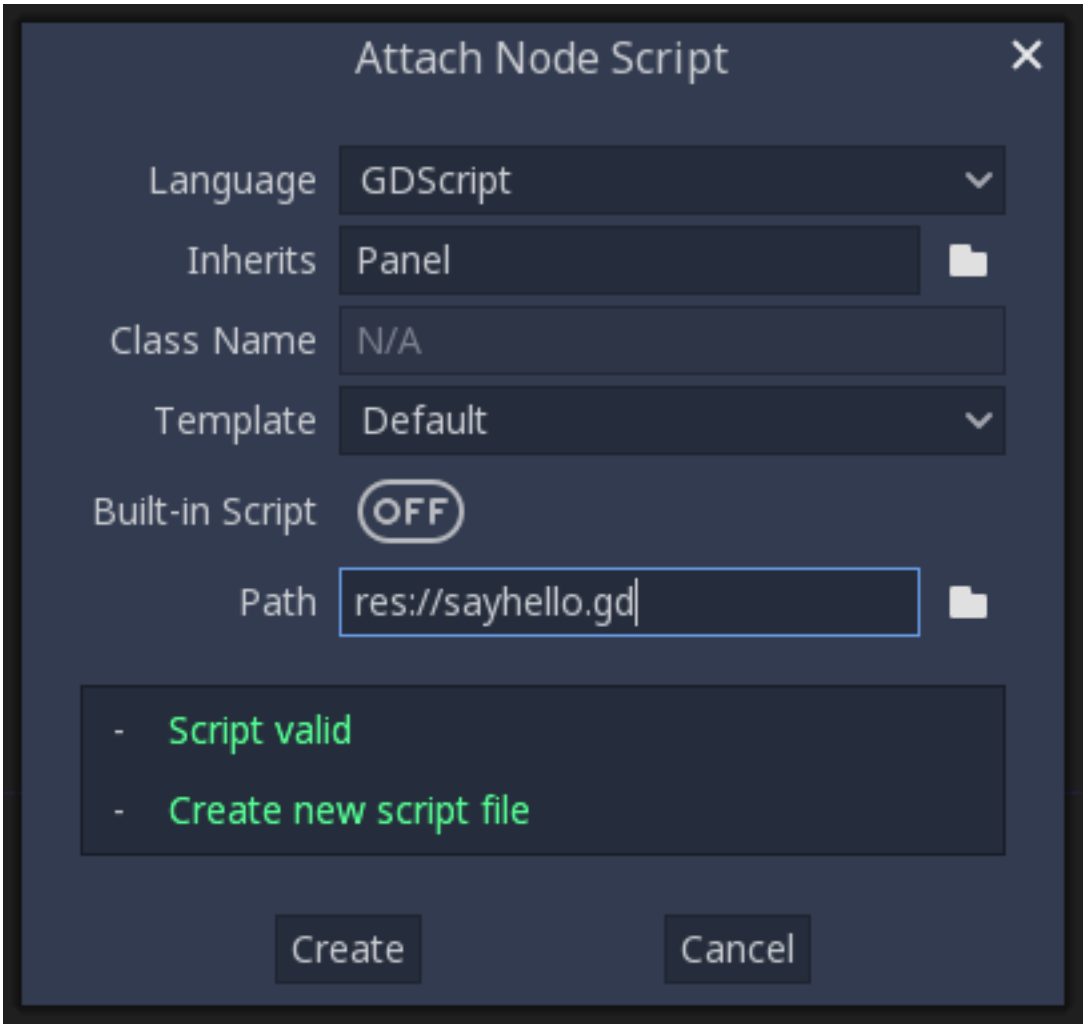


The script creation dialog will pop up. This dialog allows you to set the script's language, class name, and other relevant options.

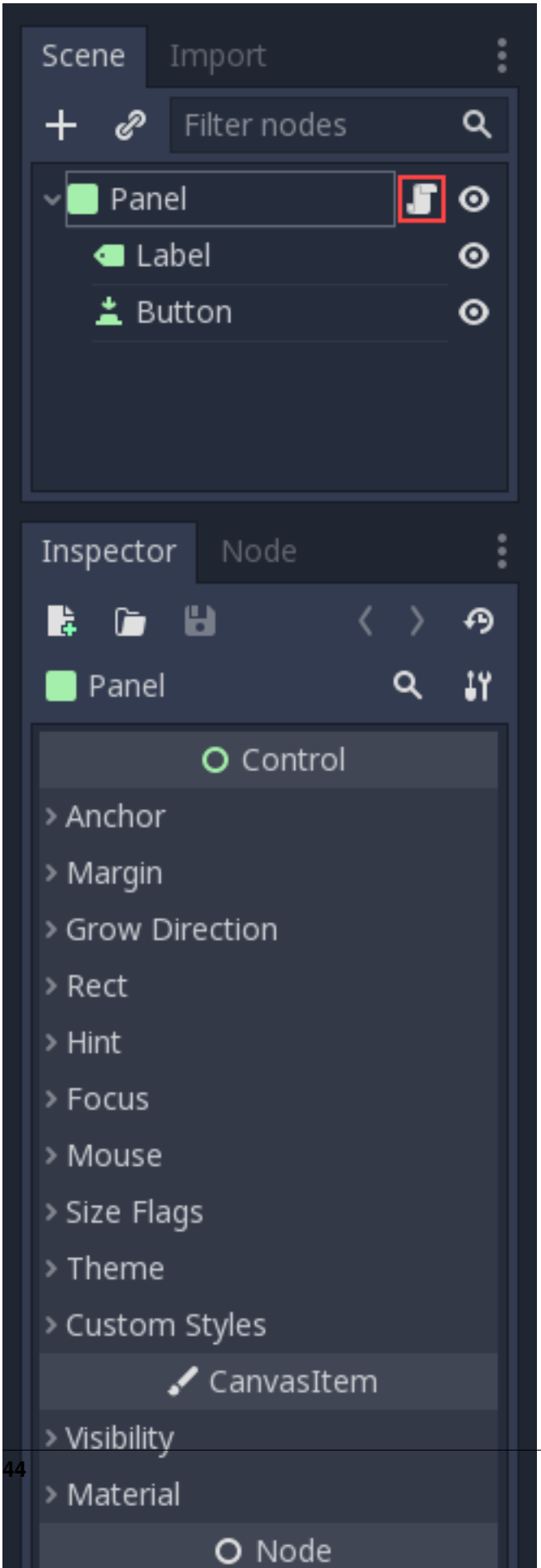
In GDScript the file itself represents the class, so the class name field is not editable.

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

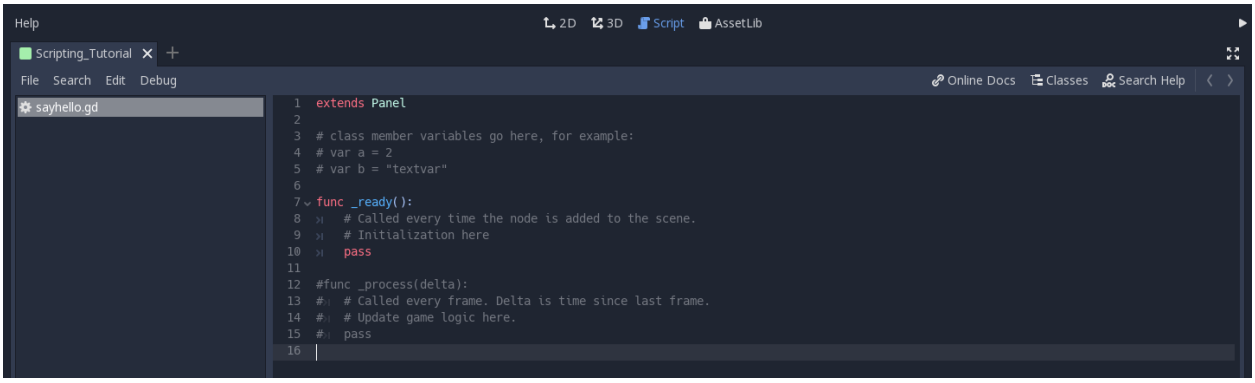
Finally, enter a path name for the script and select Create:



The script will then be created and added to the node. You can see this as an "Open script" icon next to the node in the Scene tab, as well as in the script property under Inspector:


Scene
Import
Filter nodes
Panel
Label
Button
Inspector
Node
Panel
Control
Anchor
Margin
Grow Direction
Rect
Hint
Focus
Mouse
Size Flags
Theme
Custom Styles
CanvasItem
Visibility
Material
Node

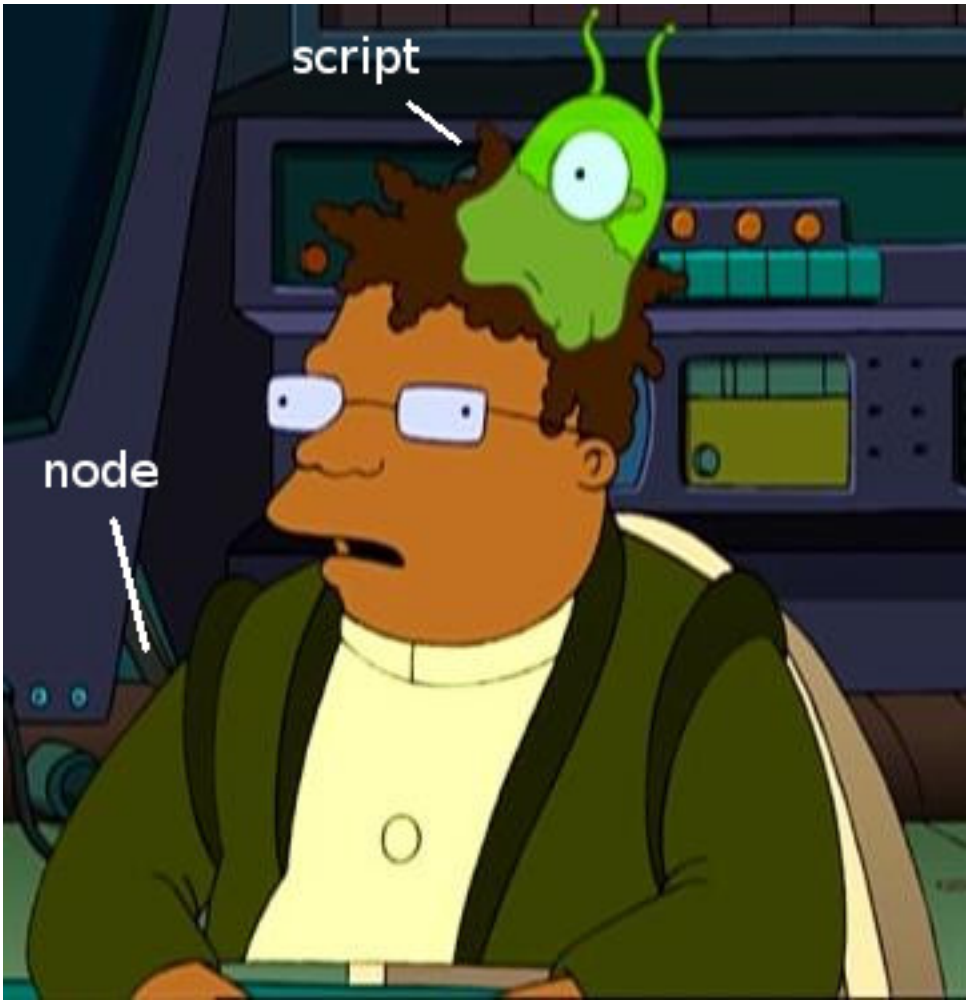
To edit the script, select either of these buttons, both of which are highlighted in the above image. This will bring you to the script editor where a default template will be included:



There's not much there. The `_ready()` function is called when the node, and all its children, enters the active scene. **Note:** `_ready()` is not the constructor; the constructor is instead `_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, and so on. The local scope of the script is the node. In other words, the script inherits the functions provided by that node.



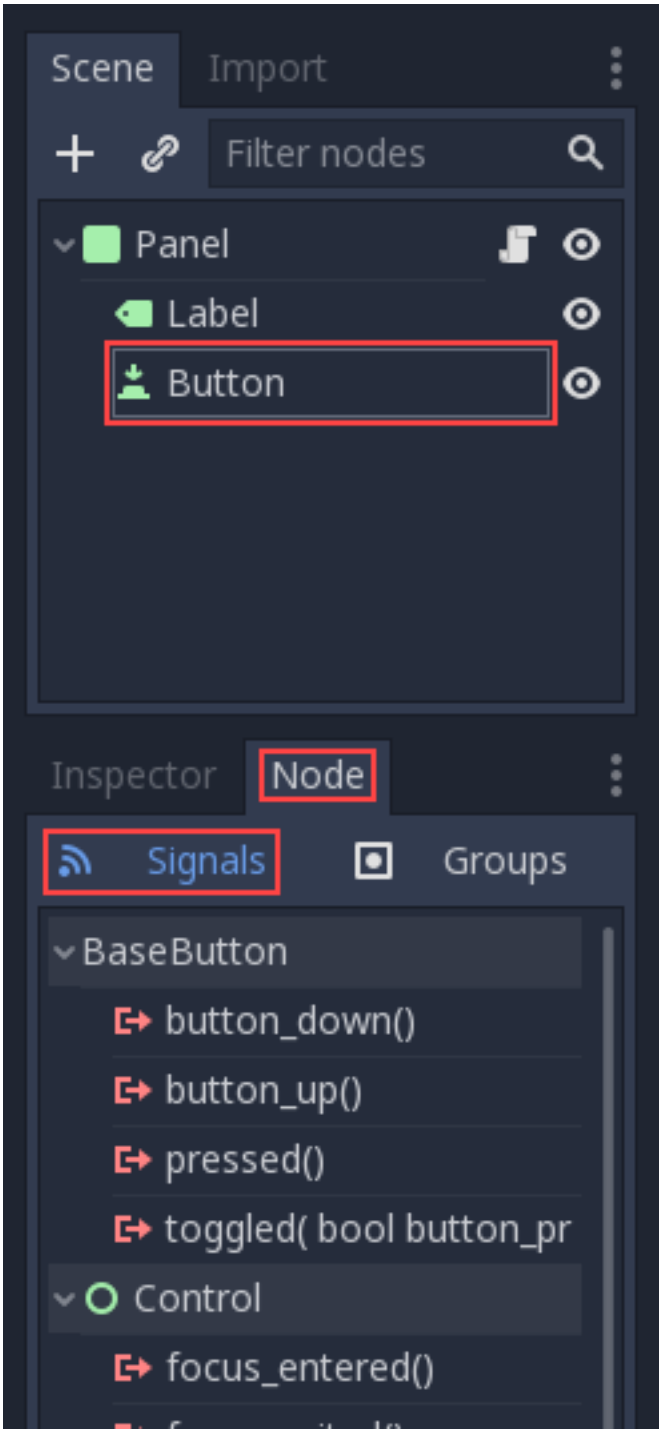
Handling a signal

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.

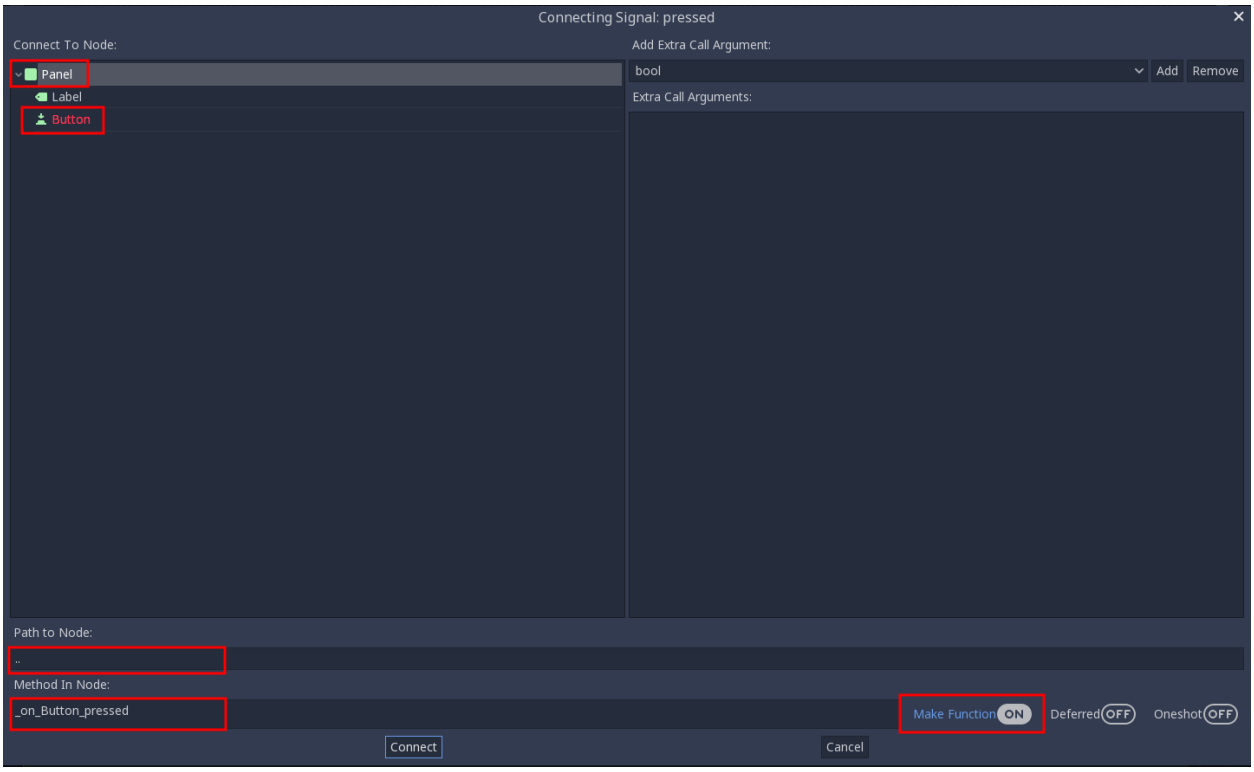
In this step, we'll connect the "pressed" signal to a custom function. Forming connections is the first part and defining the custom function is the second part. For the first part, Godot provides two ways to create connections: through a visual interface the editor provides or through code.

While we will use the code method for the remainder of this tutorial series, let's cover how the editor interface works for future reference.

Select the Button node in the scene tree and then select the "Node" tab. Next, make sure that you have "Signals" selected.



If you then select “pressed()” under “BaseButton” and click the “Connect. . . ” button in the bottom right, you’ll open up the connection creation dialogue.



In the bottom-left are the key things you need to create a connection: a node which implements the method you want to trigger (represented here as a NodePath) and the name of the method to trigger.

The top-left section displays a list of your scene's nodes with the emitting node's name highlighted in red. Select the "Panel" node here. When you select a node, the NodePath at the bottom will automatically update to point a relative path from the emitting node to the selected node.

By default, the method name will contain the emitting node's name ("Button" in this case), resulting in "_on_[EmitterNode]_[signal_name]". If you do have the "Make Function" check button checked, then the editor will generate the function for you before setting up the connection.

And that concludes the guide on how to use the visual interface. However, this is a scripting tutorial, so for the sake of learning, let's dive in to the manual process!

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

For the sake of convenience, delete everything underneath `extends Panel`. You will fill out the rest of the script manually.

Because the Button and Label are siblings under the Panel where the script is attached, you can fetch the Button by typing the following underneath the `_ready()` function:

GDScript

C#

```
func _ready():
    get_node("Button")
```

```
public override void _Ready()
{
```

(continues on next page)

(continued from previous page)

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

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

GDScript

C#

```
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 `_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 like this:

GDScript

C#

```
extends Panel

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

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

```
using Godot;

// IMPORTANT: the name of the class MUST match the filename exactly.
// this is case sensitive!
public class sayhello : Panel
{
    public override void _Ready()
    {
        GetNode("Button").Connect("pressed", this, nameof(_OnButtonPressed));
    }
```

(continues on next page)

(continued from previous page)

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

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 regarding this tutorial is how `get_node(path)` works. For a 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.

Note: The right-hand panel of the connect dialogue is for binding specific values to the connected function's parameters. You can add and remove values of different types.

The code approach also enables this with a 4th `Array` parameter that is empty by default. Feel free to read up on the `Object.connect` method for more information.

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.

However, it is still common to need a script to be processed on every frame. There are two types of processing: idle processing and physics processing.

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

This method will be called every time a frame is drawn, so it's fully dependent on how many frames per second (FPS) the application is running at:

GDScript

C#

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

```
public override void _Process(float delta)
{
    // do something...
}
```

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

This parameter can be used to make sure things always take the same amount of time, regardless of the game's FPS.

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

Physics processing with `_physics_process()` is similar, but it should be used for processes that must happen before each physics step, such as controlling a character. It always runs before a physics step and it is called at fixed time intervals: 60 times per second by default. You can change the interval from the Project Settings, under Physics -> Common -> Physics Fps.

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

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

GDScript

C#

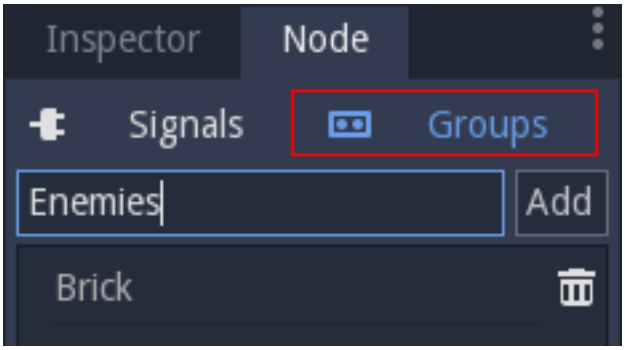
```
extends Label

var accum = 0

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

```
public class CustomLabel : Label
{
    private float _accum;

    public override void _Process(float delta)
    {
        _accum += delta;
        Text = _accum.ToString();
    }
}
```

Which will show a counter increasing each frame.

2.6.2 Groups

Nodes can be added to groups, as many as desired per node, and is a 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 way is from code. One example would be to tag scenes which are enemies:

GDScript

C#

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

```
public override void _Ready()
{
    base._Ready();

    AddToGroup("enemies");
}
```

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

GDScript

C#

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

```
public void _OnDiscovered() // this is a fictional function
{
    GetTree().CallGroup("enemies", "player_was_discovered");
}
```

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

It is also possible to get the full list of `enemies` nodes by calling *SceneTree.get_nodes_in_group()*:

GDScript

C#

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

```
var enemies = GetTree().GetNodesInGroup("enemies");
```

The *SceneTree* class provides many useful methods, like interacting with scenes, their node hierarchy and groups of nodes. It allows you to easily switch scenes or reload them, to quit the game or pause and unpause it. It even comes with interesting signals. So check it out if you got some time!

2.6.3 Notifications

Godot has a system of notifications. These are 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. For example, you may add an *Object._notification()* function in your script:

GDScript

C#

```
func _notification(what):
    match what:
        NOTIFICATION_READY:
            print("This is the same as overriding _ready()...")
        NOTIFICATION_PROCESS:
            print("This is the same as overriding _process()...")
```

```
public override void _Notification(int what)
{
    base._Notification(what);

    switch (what)
    {
        case NotificationReady:
            GD.Print("This is the same as overriding _Ready()...");
            break;
        case NotificationProcess:
            var delta = GetProcessDeltaTime();
            GD.Print("This is the same as overriding _Process()...");
            break;
    }
}
```

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

2.6.4 Overrideable functions

Such overrideable functions, which are described as follows, can be applied to nodes:

GDScript

C#

```
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
```

```
public override void _EnterTree()
{
    // 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.
    base._EnterTree();
}

public override void _Ready()
{
    // This function is called after _enter_tree, but it ensures
    // that all children nodes have also entered the _Scene Tree_,
    // and became active.
    base._Ready();
}

public override void _ExitTree()
{
    // 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.
    base._ExitTree();
```

(continues on next page)

(continued from previous page)

```
}

public override void _Process(float delta)
{
    // This function is called every frame.
    base._Process(delta);
}

public override void _PhysicsProcess(float delta)
{
    // This is called every physics frame.
    base._PhysicsProcess(delta);
}
```

As mentioned before, it's better to use these functions instead of the notification system.

2.6.5 Creating nodes

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

GDScript

C#

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

```
private Sprite _sprite;

public override void _Ready()
{
    base._Ready();

    _sprite = new Sprite(); // create a new sprite!
    AddChild(_sprite); // add it as a child of this node
}
```

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

GDScript

C#

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

```
public void _SomeAction()
{
    _sprite.Free();
}
```

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

A situation might occur where we want to delete a node that is currently "blocked", because it is emitting a signal or calling a function. This will crash the game. Running Godot with the debugger will often 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.

GDScript

C#

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

```
public void _SomeAction()
{
    _sprite.QueueFree(); // immediately removes the node from the scene and frees it
}
```

2.6.6 Instancing scenes

Instancing a scene from code is done in two steps. The first one is to load the scene from your hard drive:

GDScript

C#

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

```
var scene = (PackedScene)ResourceLoader.Load("res://myscene.tscn"); // will load when␣
↪the script is instanced
```

Preloading it can be more convenient, 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. 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:

GDScript

C#

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

```
var node = scene.Instance();
AddChild(node);
```

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

2.7 Your First Game

2.7.1 Overview

This tutorial will guide you through making your first Godot project. You will learn how the Godot 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

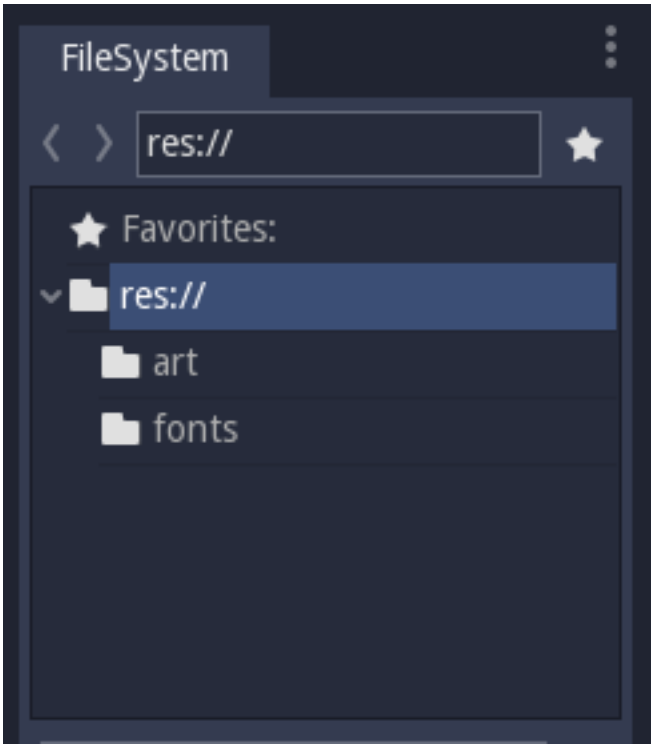
Launch Godot and create a new project. Then, download `dodge_assets.zip` - the images and sounds you'll be using to make the game. Unzip these files to your project folder.

Note: For this tutorial, we will assume you are familiar with the 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, referred to as `res://`. You can see your project folders in the FileSystem Dock in the upper left corner:

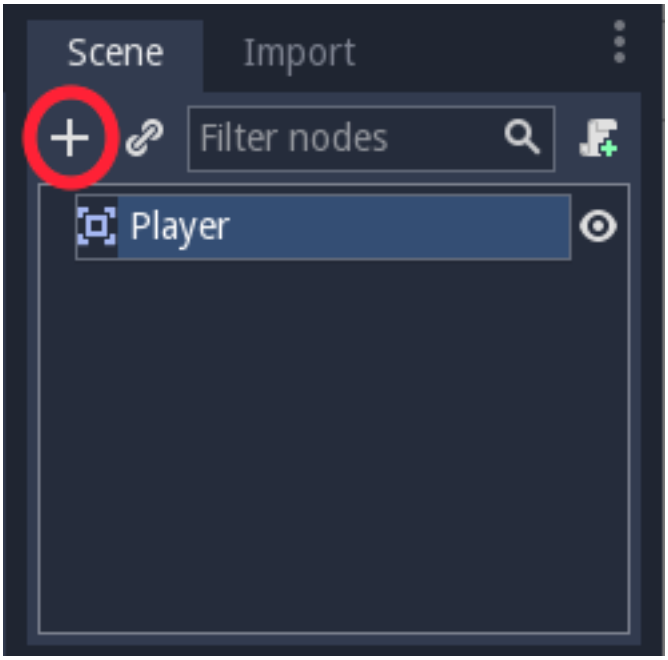


2.7.3 Player Scene

The first scene we will 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 other parts of the game.

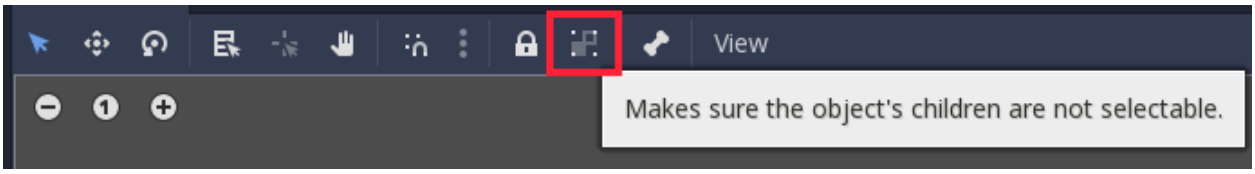
Node Structure

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



With `Area2D` we can detect objects that overlap or run into the player. Change its name to `Player` by clicking on the node's name. This is the scene's root 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 node and click the icon to the right of the lock; its tooltip says "Makes sure the object's children are not selectable."



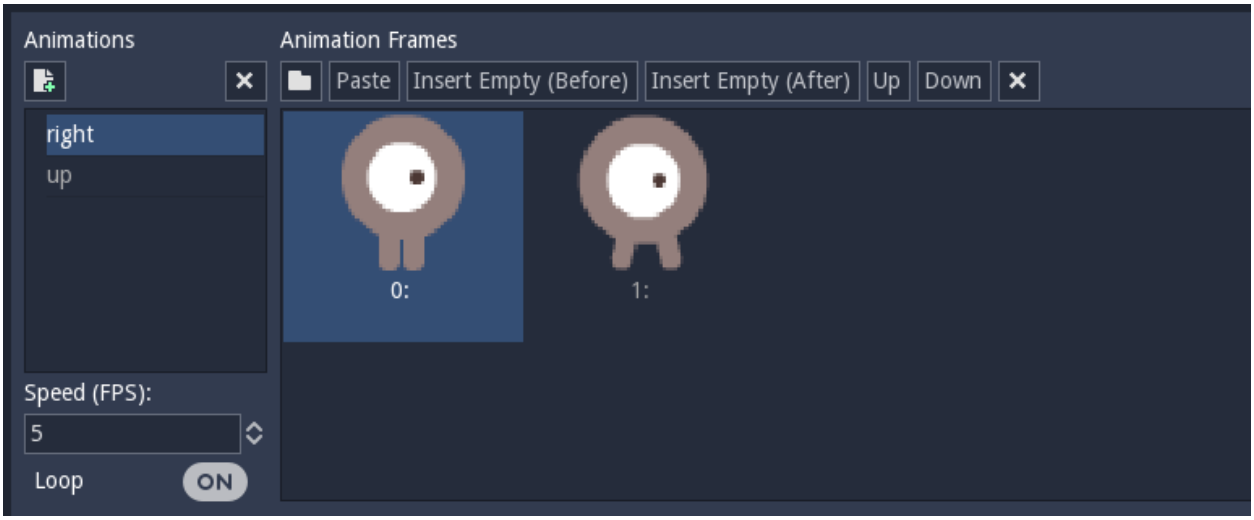
Save the scene. Click Scene -> Save, or press `Ctrl+S` on Windows/Linux or `Command+S` on Mac.

Note: For this project, we will be following the Godot naming conventions. Classes (nodes) use `PascalCase`, variables and functions use `snake_case`, and constants use `ALL_CAPS`.

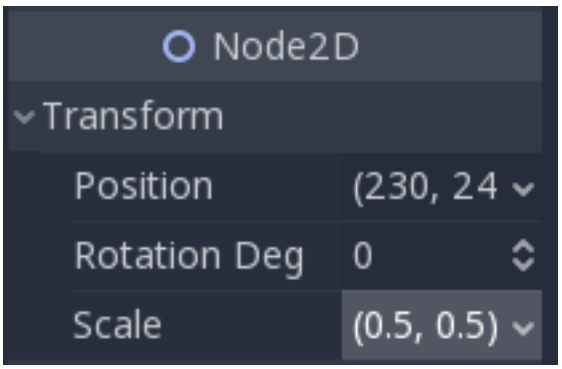
Sprite Animation

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 animations 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:

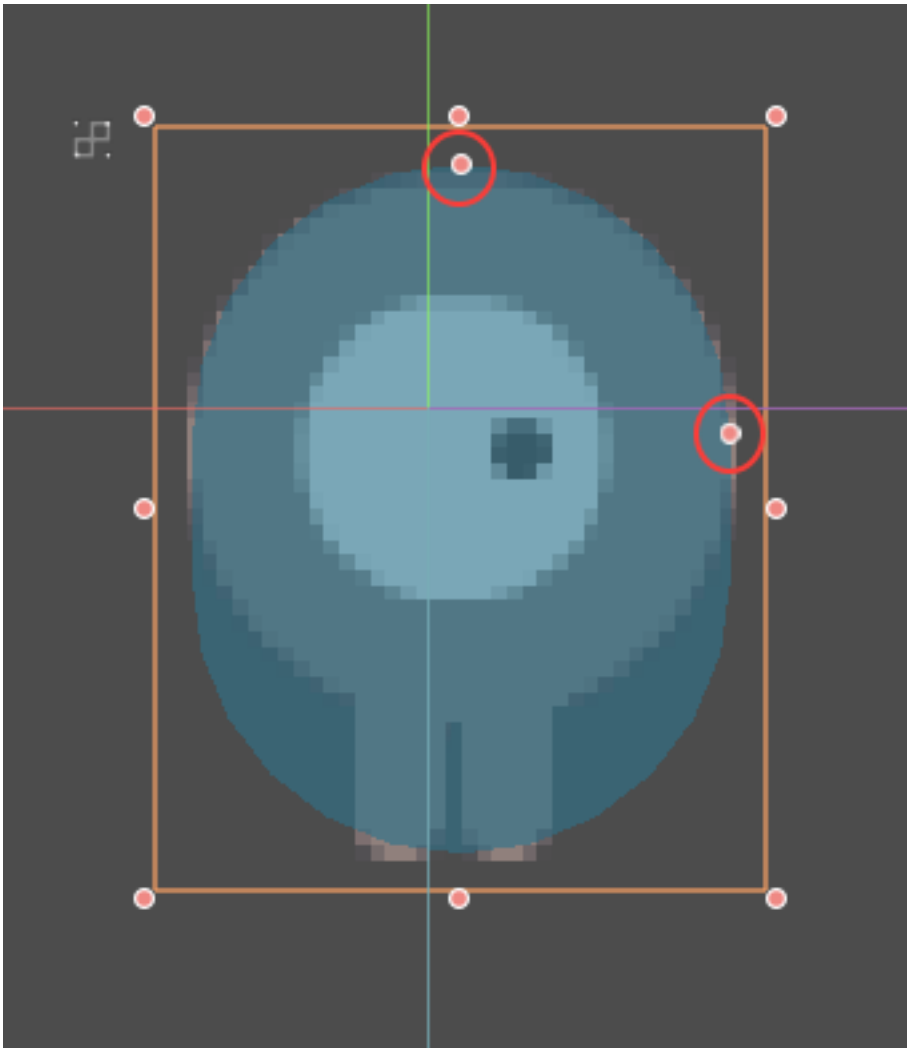
On the left is a list of animations. Click the "default" one and rename it to "right". Then click the "Add" button to create a second animation named "up". Drag the two images for each animation, named `playerGrey_up[1/2]` and `playerGrey_walk[1/2]`, into the "Animation Frames" side of the panel:



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.

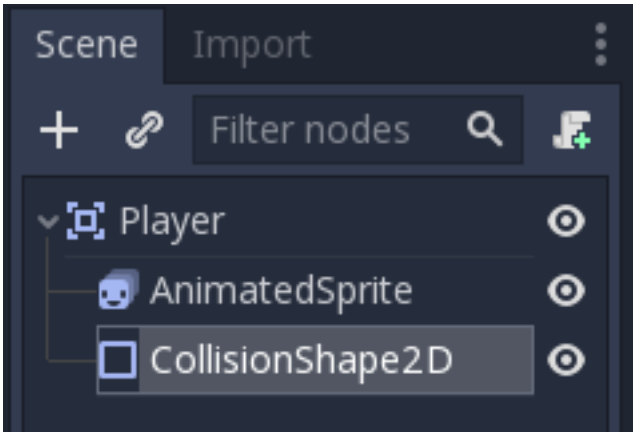


Finally, add a *CollisionShape2D* as a child of `Player`. This will determine the player's "hitbox", or the bounds of its collision area. For this character, a `CapsuleShape2D` node gives the best fit, so next to "Shape" in the Inspector, click "<null>"" -> "New CapsuleShape2D". Resize the shape to cover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



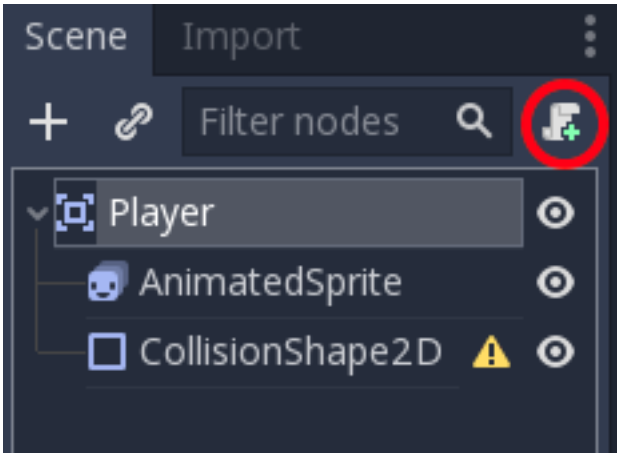
Warning: Don't scale the shape's outline! Only use the size handles (circled in red) to adjust the shape!

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



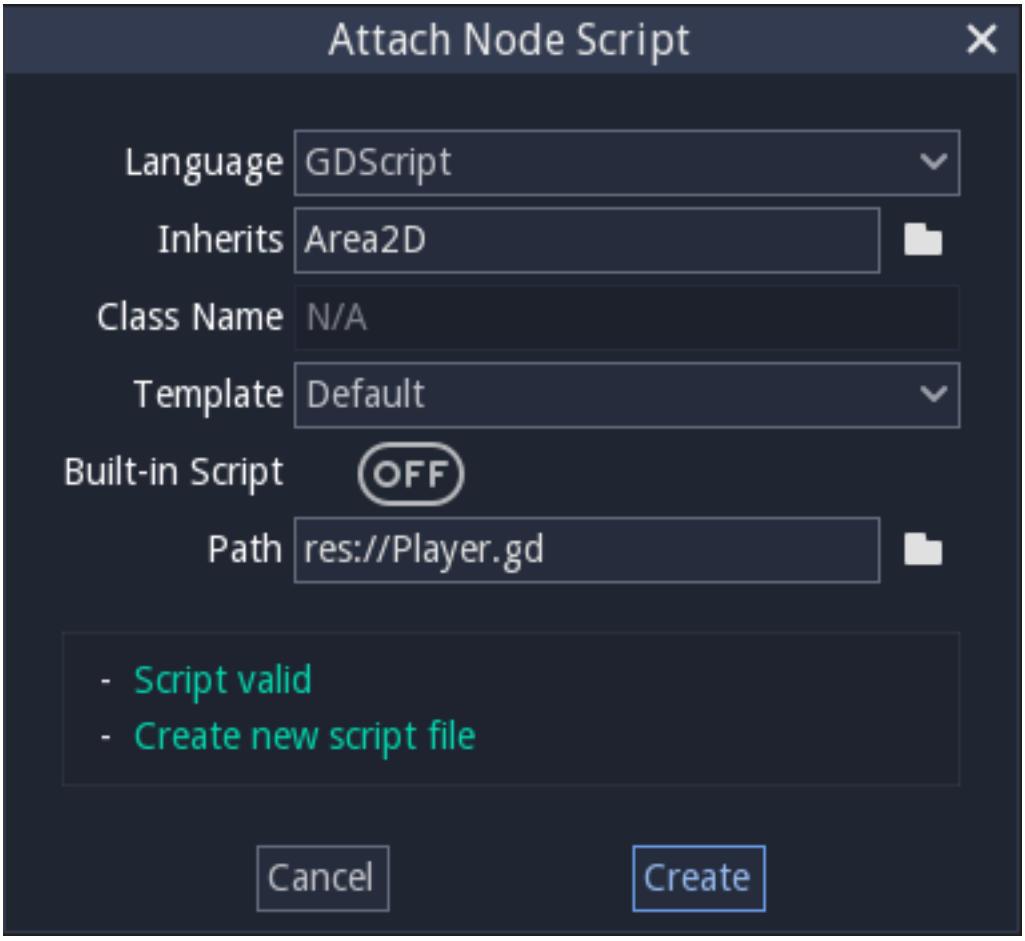
Moving the Player

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 alone. Just click "Create":

Note: If you're creating a C# script or other languages, select the language from the *language* drop down menu before hitting create.



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

Start by declaring the member variables this object will need:

GDScript

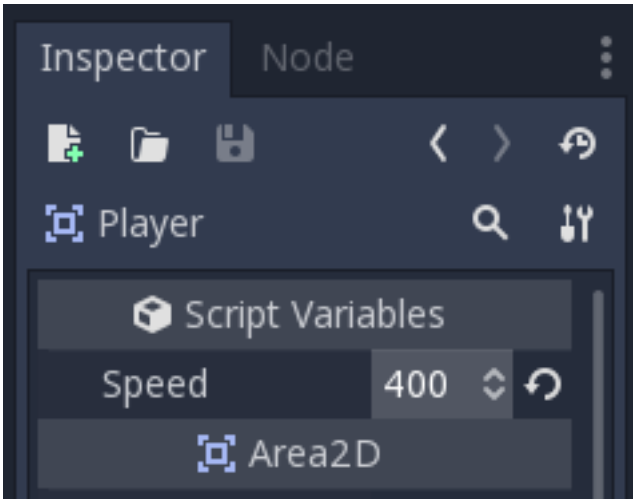
C#

```
extends Area2D

export (int) var SPEED  # how fast the player will move (pixels/sec)
var screensize  # size of the game window
```

```
public class Player : Area2D
{
    [Export]
    public int Speed = 0;

    private Vector2 _screenSize;
}
```

Using the `export` keyword on the first variable `SPEED` allows us to set its value in the Inspector. This can be 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`.

Warning: If you're using C#, you need to restart godot editor temporarily to see exported variables in the editor until it's fixed.



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

GDScript

C#

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

```
public override void _Ready()
{
    _screenSize = GetViewport().GetSize();
}
```

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

- Check for input.
- Move in the given direction.
- Play the appropriate animation.

First, we need to check for input - 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.

GDScript

C#

```
func _process(delta):
    var velocity = Vector2() # the player's movement vector
```

(continues on next page)

(continued from previous page)

```
    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()
```

```
public override void _Process(float delta)
{
    var velocity = new Vector2();
    if (Input.IsActionPressed("ui_right")) {
        velocity.x += 1;
    }

    if (Input.IsActionPressed("ui_left")) {
        velocity.x -= 1;
    }

    if (Input.IsActionPressed("ui_down")) {
        velocity.y += 1;
    }

    if (Input.IsActionPressed("ui_up")) {
        velocity.y -= 1;
    }

    var animatedSprite = (AnimatedSprite) GetNode("AnimatedSprite");
    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 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 Animat-

edSprite is a child of the current node, we can use `$AnimatedSprite`.

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

Now that we have a movement direction, we can update `Player`'s position and use `clamp()` to prevent it from leaving the screen by adding the following to the bottom of the `_process` function:

GDScript

C#

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

```
Position += velocity * delta;
Position = new Vector2(
    Mathf.Clamp(Position.x, 0, _screenSize.x),
    Mathf.Clamp(Position.y, 0, _screenSize.y)
);
```

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

Click "Play 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 with `flip_v` for downward movement. Let's place this code at the end of our `_process()` function:

GDScript

C#

```
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
```

```
if (velocity.x != 0) {
    animatedSprite.Animation = "right";
    animatedSprite.FlipH = velocity.x < 0;
    animatedSprite.FlipV = false;
```

(continues on next page)

(continued from previous page)

```
} else if(velocity.y != 0) {
    animatedSprite.Animation = "up";
    animatedSprite.FlipV = velocity.y > 0;
}
```

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

GDScript

C#

```
hide()
```

```
Hide();
```

Preparing for Collisions

We want `Player` to detect when it's 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`:

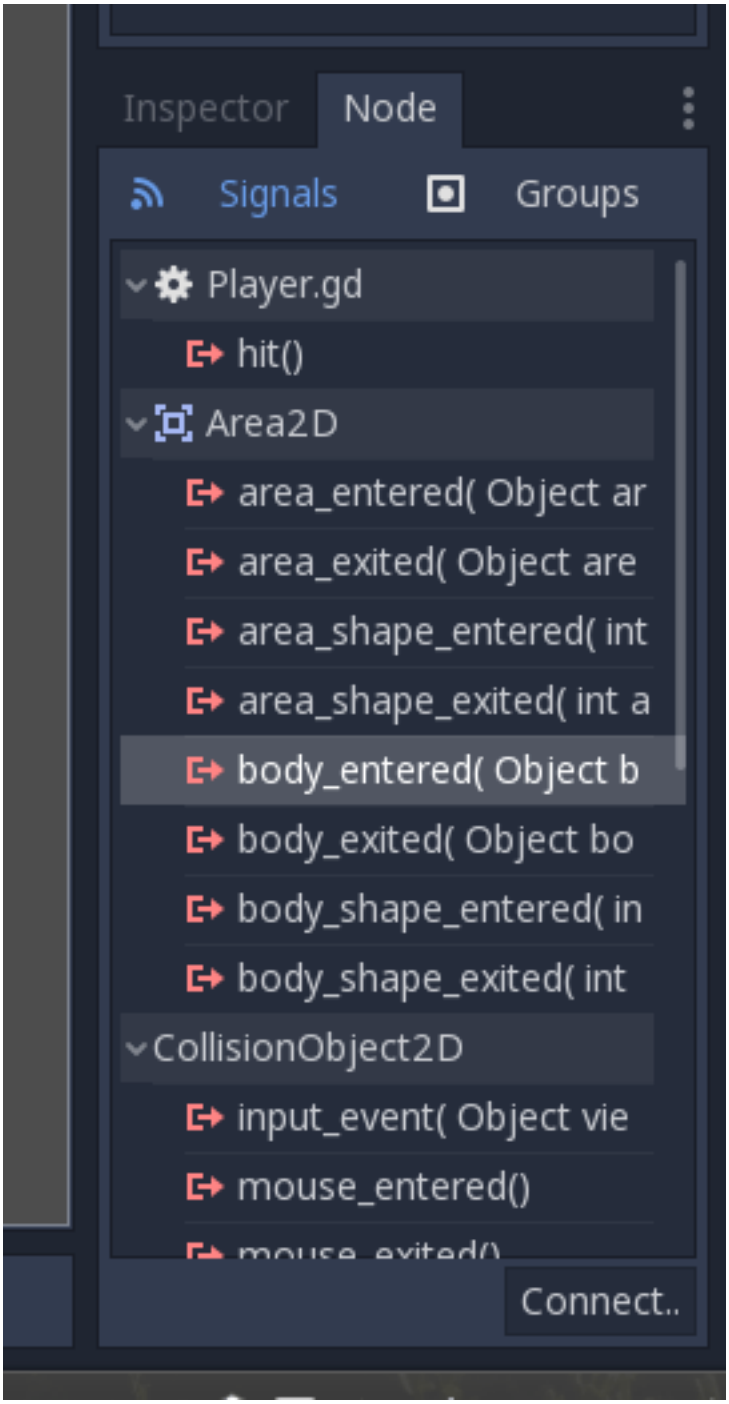
GDScript

C#

```
signal hit
```

```
[Signal]
public delegate void 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 `Area2D` to detect the collision. Select the `Player` node and click the "Node" tab next to the Inspector tab 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; this 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 these 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:

GDScript

C#

```
func _on_Player_body_entered( body ):
    hide() # Player disappears after being hit
    emit_signal("hit")
    $CollisionShape2D.disabled = true
```

```
public void OnPlayerBodyEntered(Godot.Object body)
{
    Hide();
    EmitSignal("Hit");

    // for the sake of this example, but it's better to create a class var
    // then assign the variable inside _Ready()
    var collisionShape2D = (CollisionShape2D) GetNode("CollisionShape2D");
    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.

GDScript

C#

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

```
public void Start(Vector2 pos)
{
    Position = pos;
    Show();

    var collisionShape2D = (CollisionShape2D) GetNode("CollisionShape2D");
    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 random direction in a straight line, 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

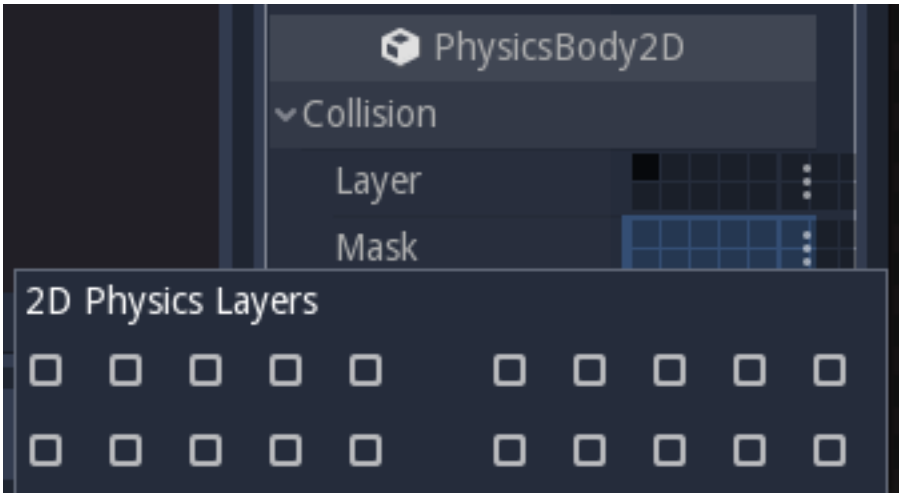
Click Scene -> New Scene and we'll create the Mob.

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 the mob will not fall downward. In addition, under the `PhysicsBody2D` section, click the `Mask` property and uncheck the first box. This will ensure 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 animations randomly so that the mobs will have some variety.

`fly` should be set to 3 FPS, with `swim` and `walk` set to 4 FPS.

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 Degrees` property to `90` under `Node2D`.

Enemy Script

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

GDScript

C#

```
extends RigidBody2D

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

```
public class Mob : RigidBody2D
{
    [Export]
    public int MinSpeed = 150;

    [Export]
    public int MaxSpeed = 250;

    private String[] _mobTypes = {"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 randomly choose one of the three animation types:

GDScript

C#

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

```
public override void _Ready()
{
    var animatedSprite = (AnimatedSprite) GetNode("AnimatedSprite");

    // C# doesn't implement gdscript's random methods, so we use Random
    // as an alternative.
    //
    // Note: Never define random multiple times in real projects. Create a
    // class memory and reuse it to get true random numbers.
    var randomMob = new Random();
    animatedSprite.Animation = _mobTypes[randomMob.Next(0, _mobTypes.Length)];
}
```

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:

GDScript

C#

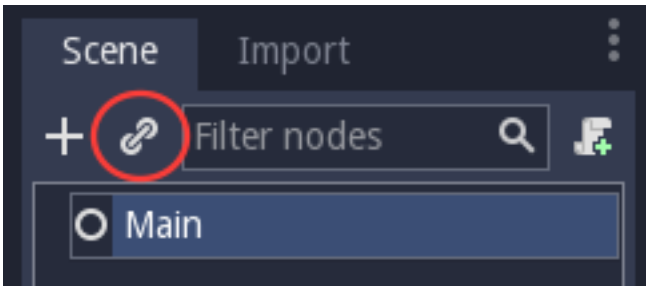
```
func _on_Visibility_screen_exited():
    queue_free()
```

```
public void onVisibilityScreenExited()
{
    QueueFree();
}
```

This 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

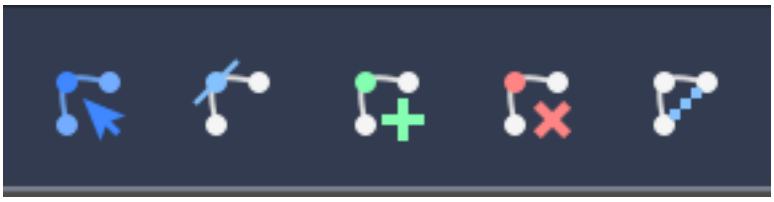
Set the `Wait Time` property of each of the `Timer` nodes as follows:

- `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* node named `MobPath` as a child of `Main`. When you select `Path2D`, you will see some new buttons at the top of the editor:



Select the middle one ("Add Point") and draw the path by clicking to add the points at the corners 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, appearing 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.

GDScript

C#

```
extends Node

export (PackedScene) var Mob
var score

func _ready():
    randomize()
```

```
public class Main : Node
{
    [Export]
    public PackedScene Mob;

    public int Score = 0;

    // note: we're going to use this many times, so instantiating it
    // allows our numbers to consistently be random
    private Random rand = new Random();

    public override void _Ready()
    {
    }

    // we'll use this later because c# doesn't support gdscript's randi()
    private float RandRand(float min, float max)
    {
```

(continues on next page)

(continued from previous page)

```
        return (float) (rand.NextDouble() * (max - min) + min);
    }
}
```

Drag `Mob.tscn` from the "FileSystem" panel and drop it in the `Mob` property under the Script Variables of the `Main` node.

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:

GDScript

C#

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

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

```
public void GameOver()
{
    //timers
    var mobTimer = (Timer) GetNode("MobTimer");
    var scoreTimer = (Timer) GetNode("ScoreTimer");

    scoreTimer.Stop();
    mobTimer.Stop();
}

public void NewGame()
{
    Score = 0;

    var player = (Player) GetNode("Player");
    var startTimer = (Timer) GetNode("StartTimer");
    var startPosition = (Position2D) GetNode("StartPosition");

    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.

GDScript

C#

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

(continues on next page)

(continued from previous page)

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

```
public void OnStartTimerTimeout()
{
    //timers
    var mobTimer = (Timer) GetNode("MobTimer");
    var scoreTimer = (Timer) GetNode("ScoreTimer");

    mobTimer.Start();
    scoreTimer.Start();
}

public void OnScoreTimerTimeout()
{
    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()`.

GDScript

C#

```
func _on_MobTimer_timeout():
    # choose a random location on 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 a 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))
```

```
public void OnMobTimerTimeout()
{
    //choose random location on path2d
    var mobSpawnLocation = (PathFollow2D) GetNode("MobPath/MobSpawnLocation");
    mobSpawnLocation.SetOffset(rand.Next());

    //set direction
    var direction = mobSpawnLocation.Rotation + Mathf.PI/2;
    direction += RandRand(-Mathf.PI/4, Mathf.PI/4);

    //create mob instance and add it to scene
```

(continues on next page)

(continued from previous page)

```
    var mobInstance = (RigidBody2D) Mob.Instance();
    mobInstance.Position = mobSpawnLocation.Position;
    mobInstance.Rotation = direction;
    mobInstance.SetLinearVelocity(new Vector2(RandRand(150f, 250f), 0).
↪Rotated(direction));

    AddChild(mobInstance);
}
```

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.

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", an informational display that appears as an overlay on top of the game view.

The *CanvasLayer* node lets us draw our UI elements on a layer above the rest of the game, so that the information it displays isn't covered up by any game elements like the player or 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.

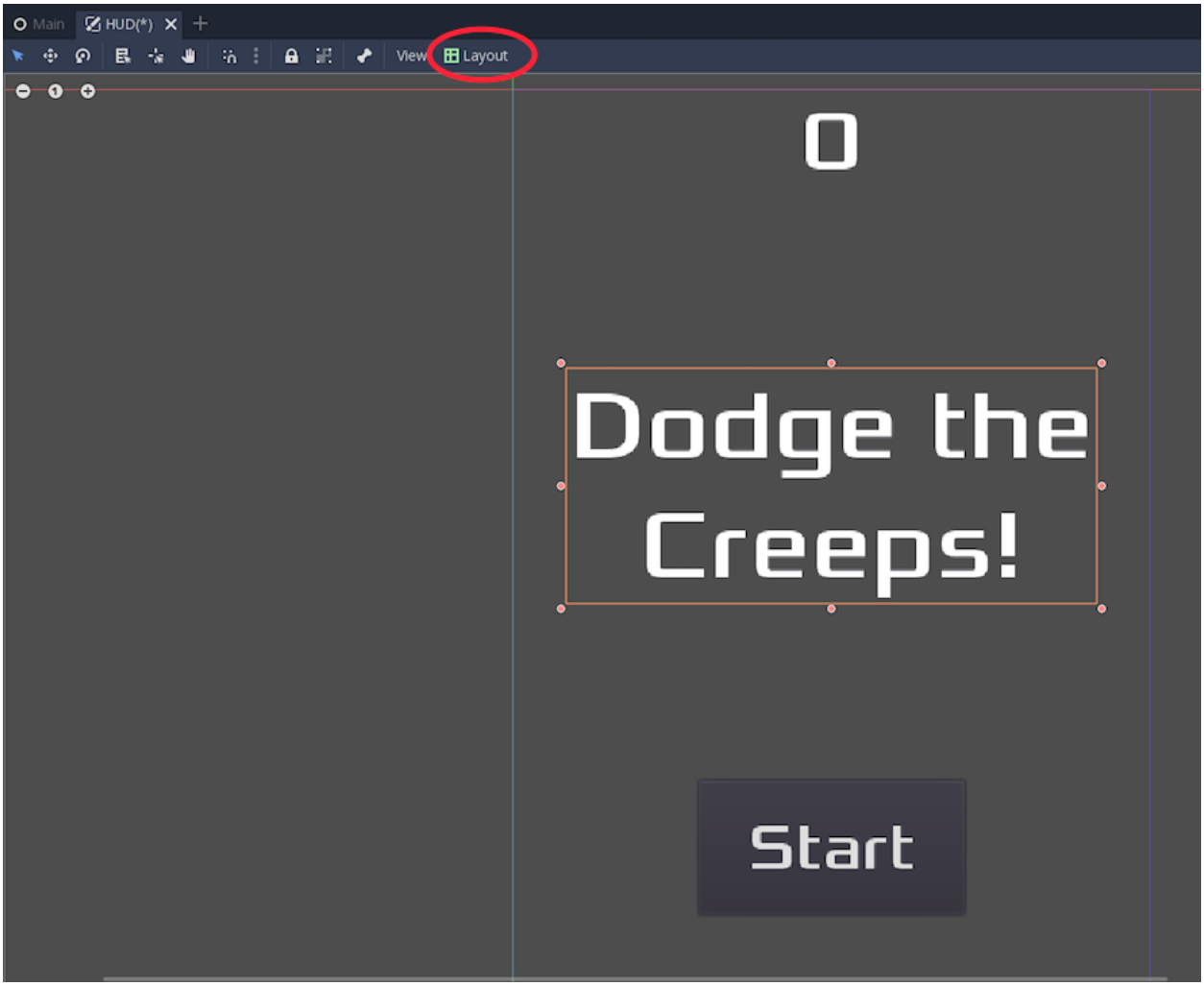
The basic node for UI elements is *Control*. To create our UI, we'll use two types of *Control* nodes: *Label* and *Button*.

Create the following as 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 - 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"
- `Margin`:

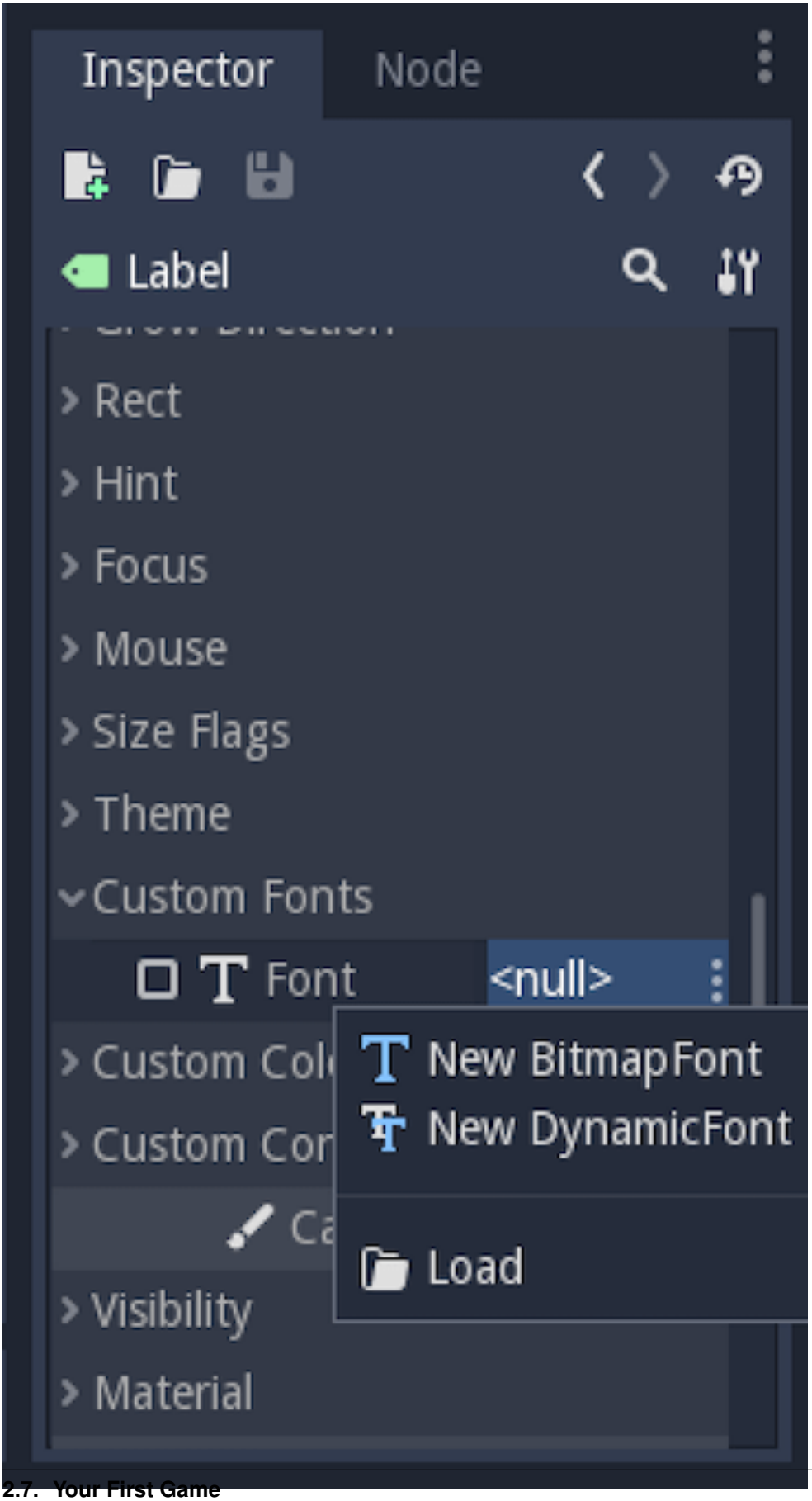
  - Left: `-200`
  - Top: `-150`
  - Right: `200`
  - Bottom: `0`
- Text: `Dodge the Creeps!`

StartButton

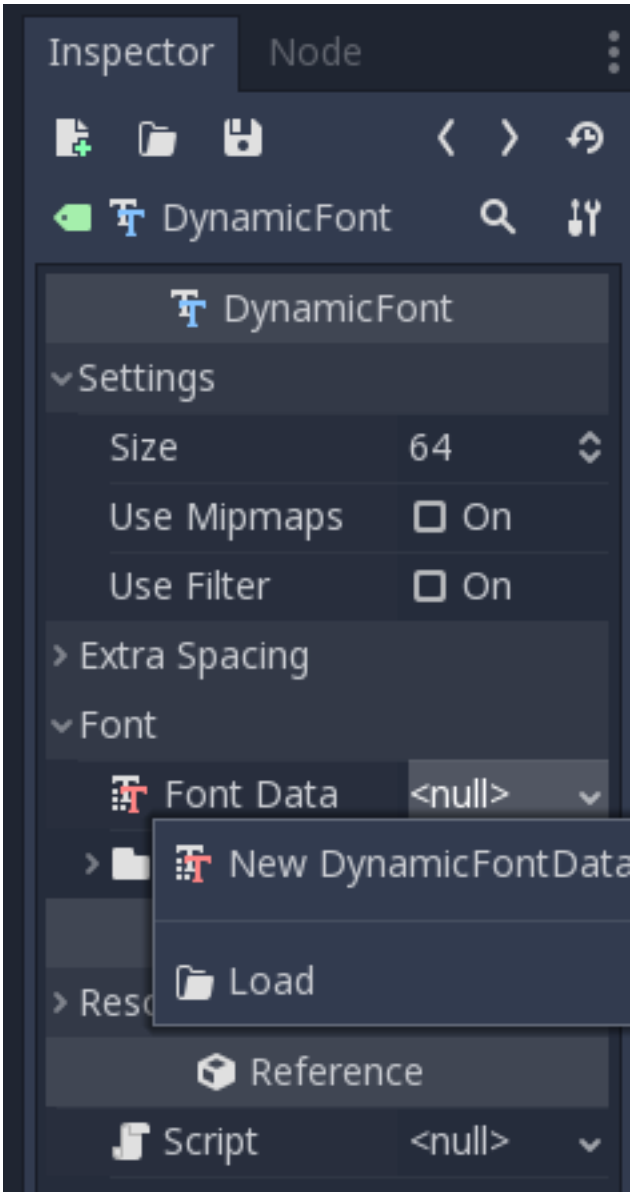
- `Layout`: "Center Bottom"
- `Margin`:
  - Left: `-100`
  - Top: `-200`
  - Right: `100`
  - Bottom: `-100`
- Text: `Start`

The default font for `Control` nodes is 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"


Inspector
Node
Label
Rect
Hint
Focus
Mouse
Size Flags
Theme
Custom Fonts
Font
<null>
New BitmapFont
New DynamicFont
Load
Visibility
Material

2. Click on the "DynamicFont" you 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 `HUD`:

GDScript

C#

```
extends CanvasLayer

signal start_game
```

```
public class HUD : CanvasLayer
{
    [Signal]
    public delegate void StartGame();
}
```

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

GDScript

C#

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

```
public void ShowMessage(string text)
{
    var messageTimer = (Timer) GetNode("MessageTimer");
    var messageLabel = (Label) GetNode("MessageLabel");

    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".

GDScript

C#

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

```
async public void ShowGameOver()
{
    var startButton = (Button) GetNode("StartButton");
    var messageTimer = (Timer) GetNode("MessageTimer");
    var messageLabel = (Label) GetNode("MessageLabel");

    ShowMessage("Game Over");
    await ToSignal(messageTimer, "timeout");
    messageLabel.Text = "Dodge the\nCreeps!";
    messageLabel.Show();
    startButton.Show();
}
```

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

GDScript

C#

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

```
public void UpdateScore(int score)
{
    var scoreLabel = (Label) GetNode("ScoreLabel");
    scoreLabel.Text = score.ToString();
}
```

This function is called in `Main` whenever the score changes.

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

GDScript

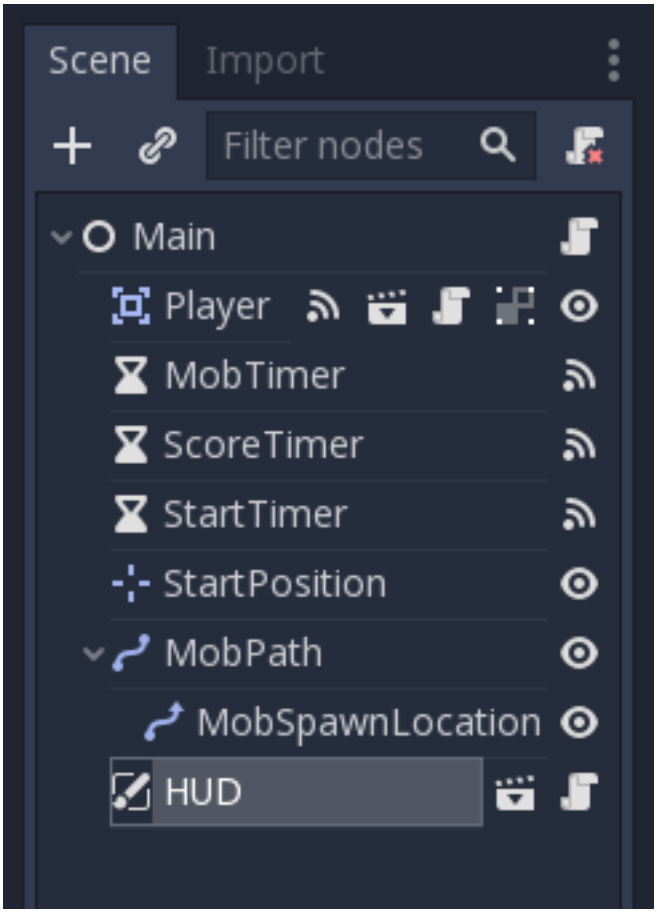
C#

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

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

```
public void OnStartButtonPressed()
{
    var startButton = (Button) GetNode("StartButton");
    startButton.Hide();

    EmitSignal("StartGame");
}

public void OnMessageTimerTimeout()
{
    var messageLabel = (Label) GetNode("MessageLabel");
    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 the 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:

GDScript

C#

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

```
var hud = (HUD) GetNode("HUD");
hud.UpdateScore(Score);
hud.ShowMessage("Get Ready!");
```

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

GDScript

C#

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

```
var hud = (HUD) GetNode("HUD");
hud.ShowGameOver();
```

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

GDScript

C#

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

```
var hud = (HUD) GetNode("HUD");
hud.UpdateScore(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 have now completed all the functionality for our game. Below are some remaining steps to add a bit more "juice" to 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.

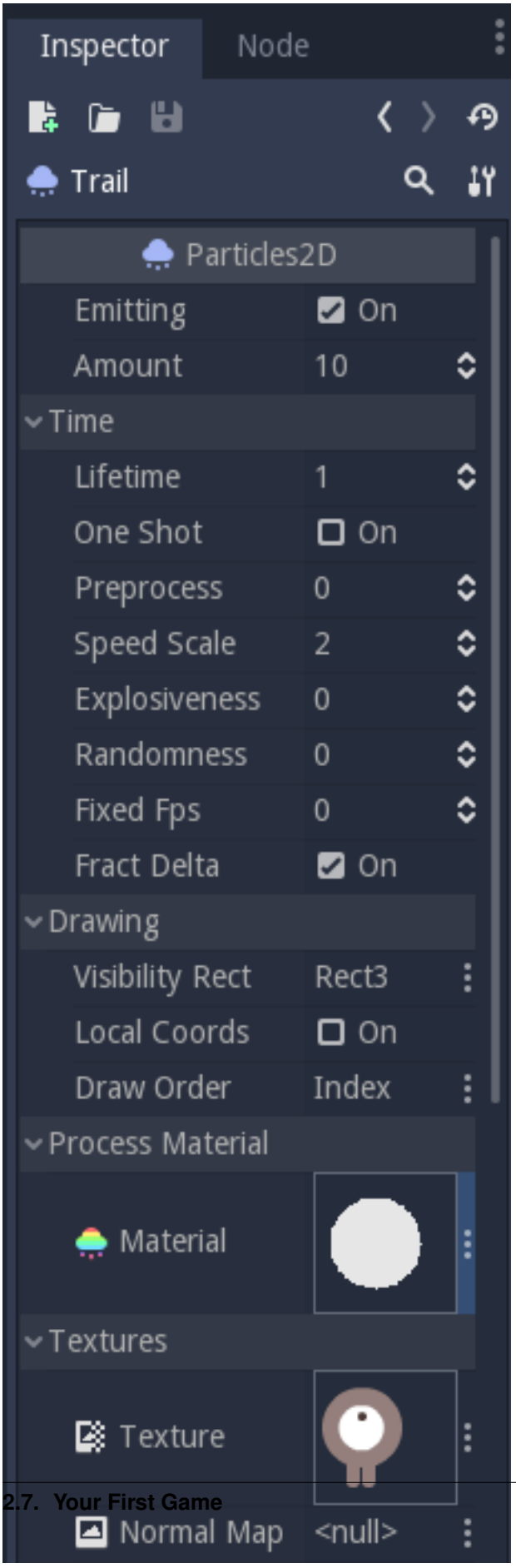
To play the music, add `$Music.play()` in the `new_game()` function and `$Music.stop()` in the `game_over()` function.

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

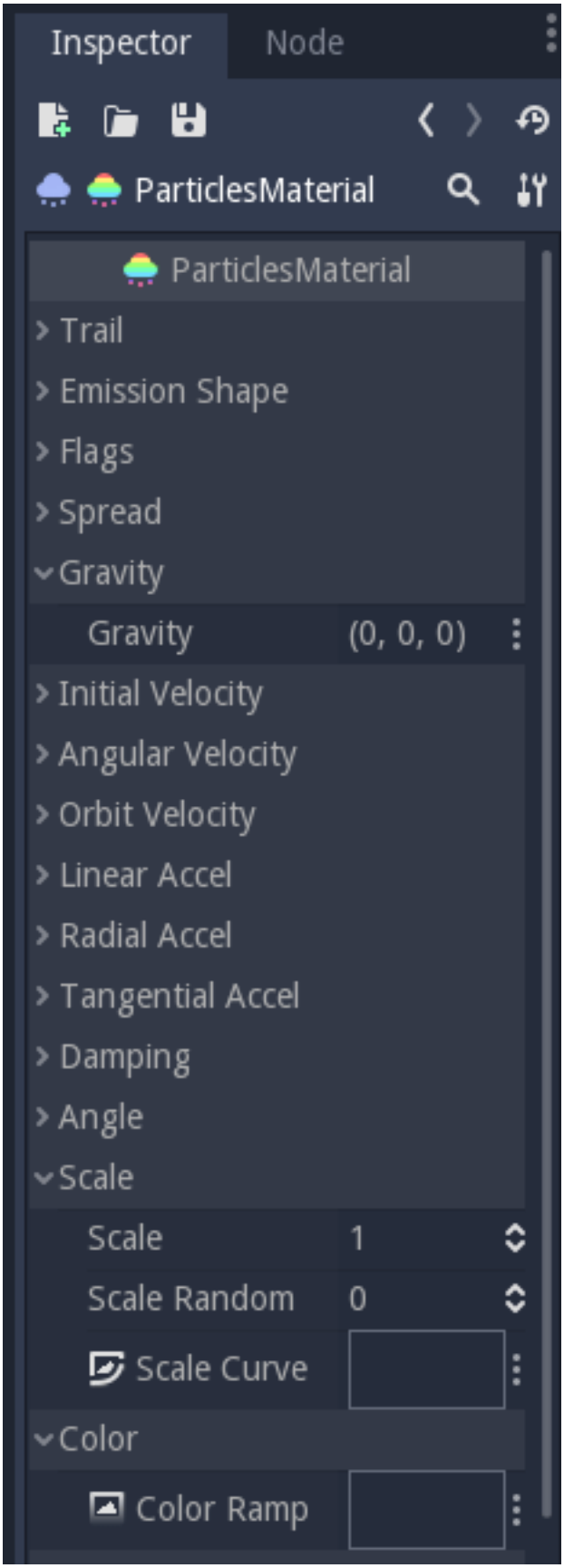
Particles

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

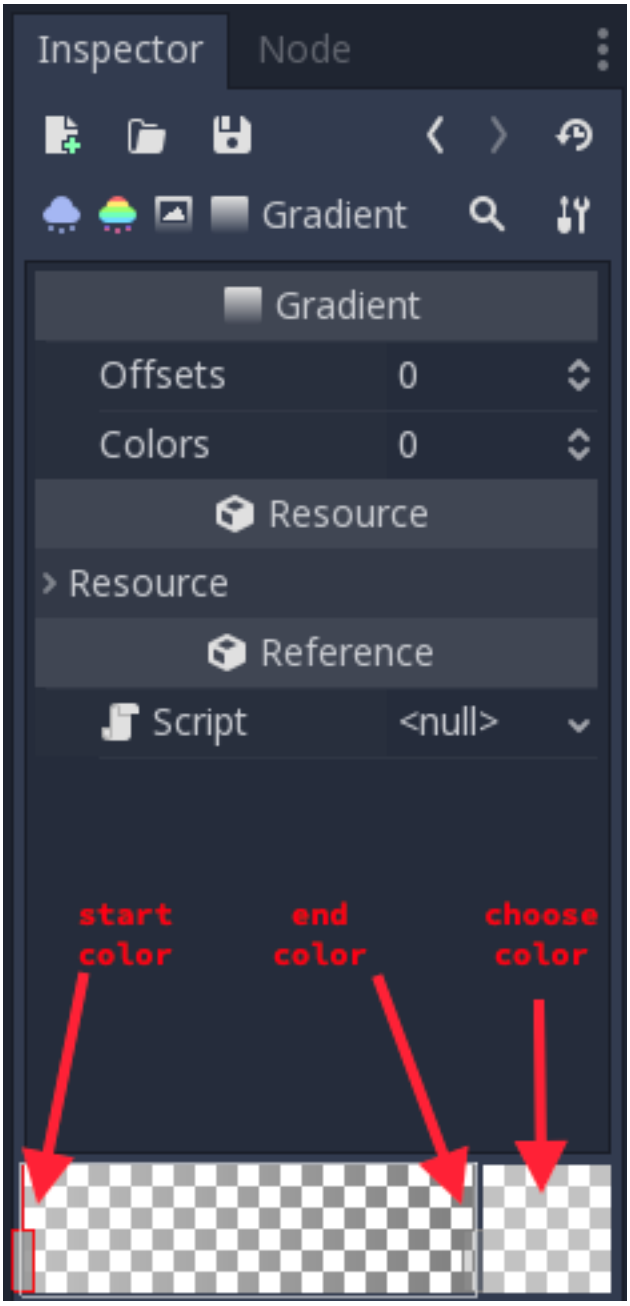

Inspector
Node
Trail
Particles2D
Emitting
On
Amount
10
Time
Lifetime
1
One Shot
On
Preprocess
0
Speed Scale
2
Explosiveness
0
Randomness
0
Fixed Fps
0
Fract Delta
On
Drawing
Visibility Rect
Rect3
Local Coords
On
Draw Order
Index
Process Material
Material
Textures
Texture
Normal Map
<null>

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


Inspector
Node
ParticlesMaterial
ParticlesMaterial
Trail
Emission Shape
Flags
Spread
Gravity
Gravity (0, 0, 0)
Initial Velocity
Angular Velocity
Orbit Velocity
Linear Accel
Radial Accel
Tangential Accel
Damping
Angle
Scale
Scale 1
Scale Random 0
Scale Curve
Color
Color Ramp

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 Exporting

2.8.1 Overview

Now that you have a working game, you probably want to share your success with others. However, it's not practical to ask your friends to download Godot just so they can open your project. Instead, you can *export* your project, converting it into a "package" that can be run by anyone.

The way you export your game depends on what platform you are targeting. In this tutorial, you'll learn how to export the "Dodge the Creeps" game for a variety of platforms. First, however, we need to make some changes to the way the game works.

Note: If you haven't made "Dodge the Creeps" yourself yet, please read *Your First Game* before continuing with this tutorial.

2.8.2 Preparing the project

In "Dodge the Creeps" we used keyboard controls to move the player's character. This is fine if your game is being played on a PC platform, but on a phone or tablet, you need to support touchscreen input. Because a click event can be treated the same as a touch event, we'll convert the game to a click-and-move input style.

The first step is to open "Project Settings" and find the *Handheld* section. Enable the *Emulate Touchscreen* option. This lets you treat mouse click events the same as touch events, so you can test the game on a computer without a touchscreen. Also, make sure to select "portrait" under *Orientation*.

In the *Stretch* section, set *Mode* to "2d" and *Aspect* to "keep". This ensures that the game scales consistently on different sized screens.

Next, we need to modify the `Player.gd` script to change the input method. We'll remove the key inputs and make the player move towards a "target" that's set by the touch (or click) event.

Here is the full script for the player, with comments noting what we've changed:

GDScript

```
extends Area2D

signal hit

export (int) var SPEED
var velocity = Vector2()
var screensize
# Add this variable to hold the clicked position.
var target = Vector2()

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

func start(pos):
    position = pos
    # Initial target is the start position.
    target = pos
    show()
    $Collision.disabled = false
```

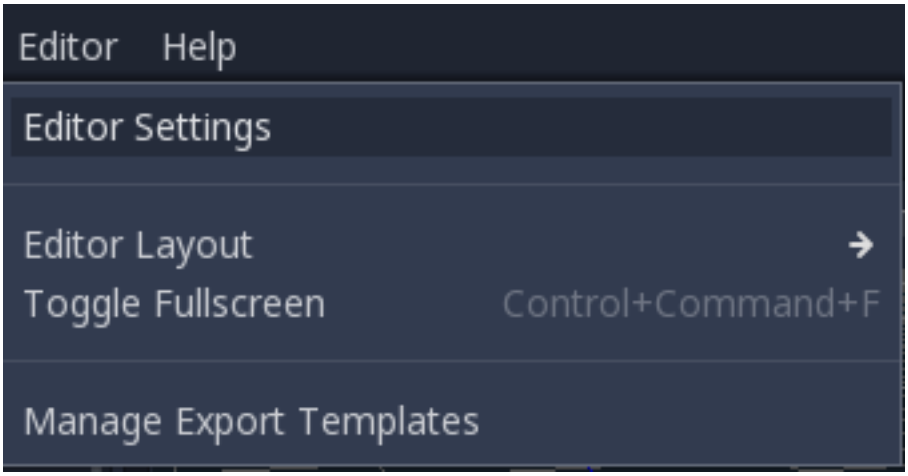
(continues on next page)

(continued from previous page)

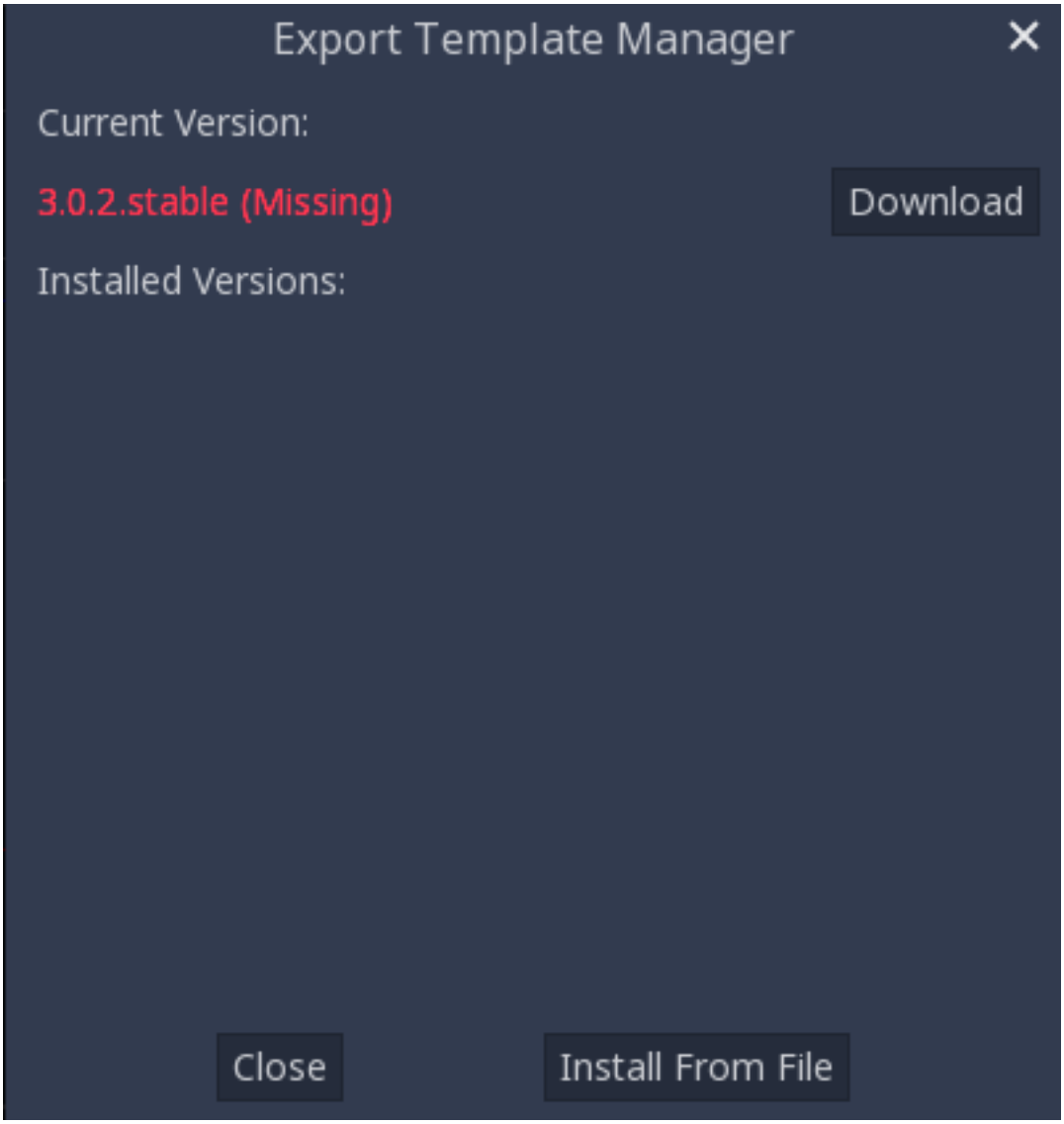
```
# Change the target whenever a touch event happens.
func _input(event):
    if event is InputEventScreenTouch and event.pressed:
        target = event.position

func _process(delta):
    # Move towards the target and stop when close.
    if position.distance_to(target) > 10:
        velocity = (target - position).normalized() * SPEED
    else:
        velocity = Vector2()

# Remove keyboard controls.
#   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()
        $Trail.emitting = true
    else:
        $AnimatedSprite.stop()
        $Trail.emitting = false

    position += velocity * delta
    # We don't need to clamp the player's position
    # because you can't click outside the screen.
    # position.x = clamp(position.x, 0, screensize.x)
    # position.y = clamp(position.y, 0, screensize.y)

    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

func _on_Player_body_entered( body ):
    $Collision.disabled = true
    hide()
    emit_signal("hit")
```

2.8.3 Export Templates

In order to export, you need to download the *export templates* from the http://godotengine.org/download. These templates are optimized versions of the engine without the editor pre-compiled for each platform . You can also download them in Godot by clicking on *Editor -> Manage Export Templates*:



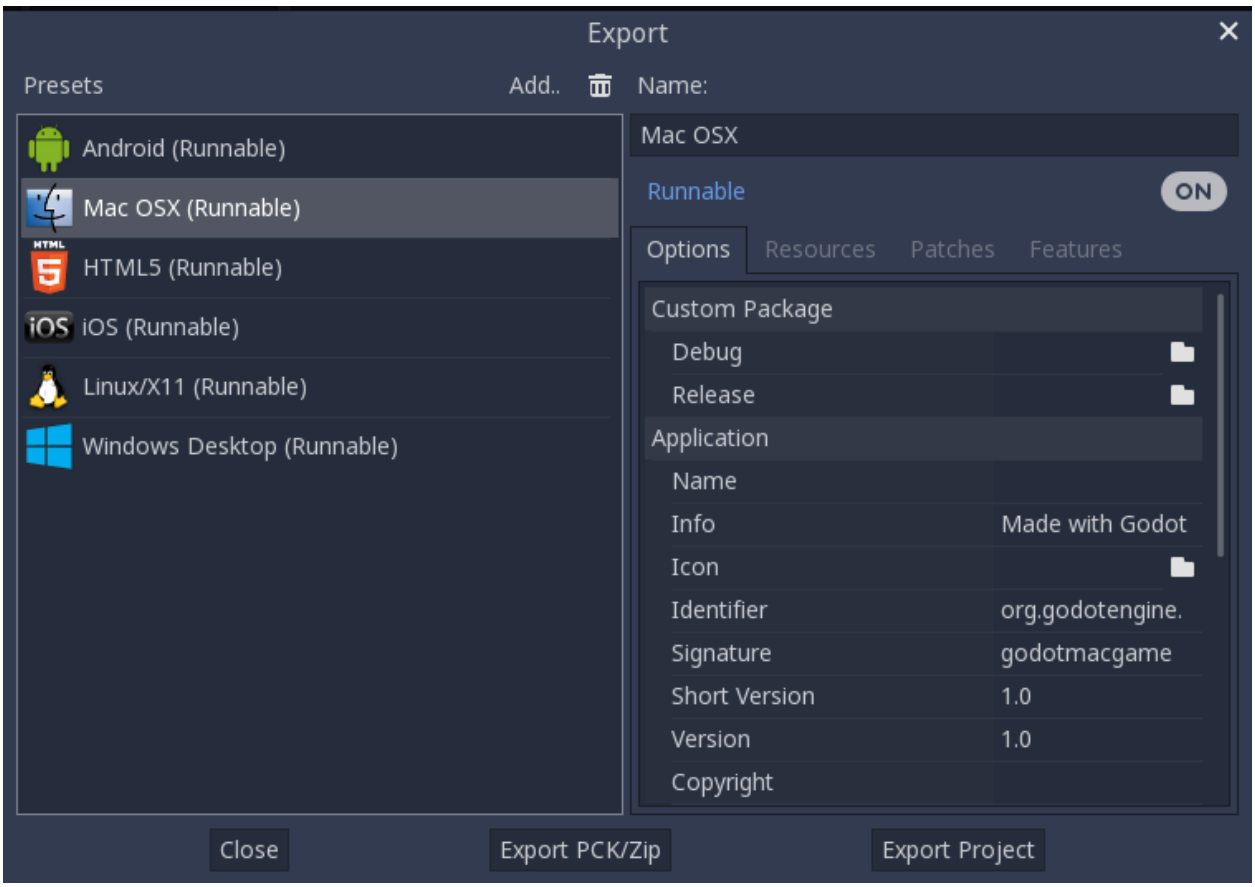
In the window that appears, you can click "Download" to get the template version that matches your version of Godot.



Note: If you upgrade Godot, you must download templates that match the new version or your exported projects may not work correctly.

2.8.4 Export Presets

Next, you can configure the export settings by clicking on *Project -> Export*:



Create a new export preset by clicking "Add..." and selecting a platform. You can make as many presets as you like with different settings.

At the bottom of the window are two buttons. "Export PCK/ZIP" only creates a packed version of your project's data. This doesn't include an executable so the project can't be run on its own.

The second button, "Export Project", creates a complete executable version of your game, such as an *.apk* for Android or an *.exe* for Windows.

In the "Resources" and "Features" tabs you can customize how the game is exported for each platform. We can leave those settings alone for now.

2.8.5 Exporting by Platform

In this section, we'll walk through the process for each platform, including any additional software or requirements you'll need.

PC (Linux/MacOS/Windows)

Exporting PC platforms works the same across the three supported operating systems. Open the export window and click "Add.." to create the preset(s) you want to make. Then click "Export Project" and choose a name and destination folder. Choose a location *outside* of your project folder.

Click "Save" and the engine will build the export files.

Note: When exporting for MacOS, if you export on a MacOS computer, you'll end up with a *.dmg* file, while using Linux or Windows produces a *.zip*. In either case, the compressed file contains a MacOS *.app* that you can double-click and run.

Note: On Windows, if you want your exported executable to have a different icon than the default one, you need to change it manually. See: *Changing application icon for windows*.

Android

Tip: Mobile devices come with a wide variety of capabilities. In most cases, Godot's default settings will work, but mobile development is sometimes more art than science, and you may need to do some experimenting and searching for help in order to get everything working.

Before you can export your project for Android, you must download the following software:

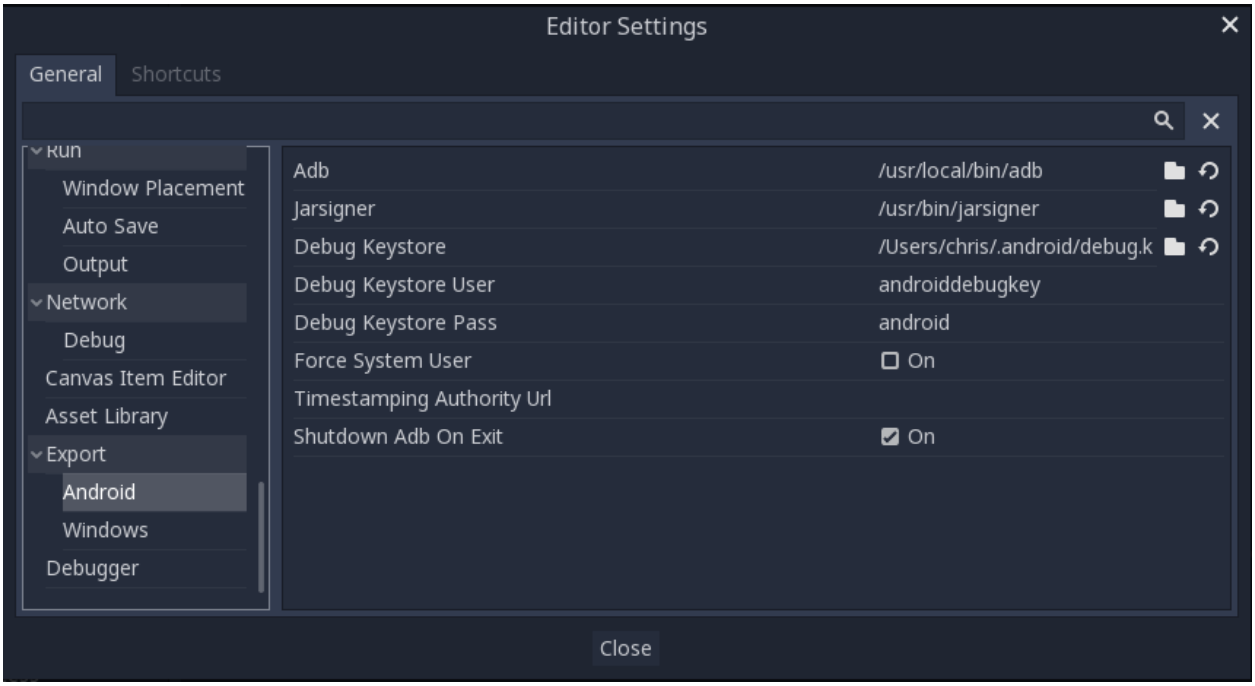
- Android SDK: https://developer.android.com/studio/
- Java JDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html

When you run Android Studio for the first time, click on *Configure -> SDK Manager* and install "Android SDK Platform Tools". This installs the *adb* command-line tool that Godot uses to communicate with your device.

Next, create a debug keystore with by running the following command on your system's command line:

```
keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android -keystore␣
↪debug.keystore -storepass android -dname "CN=Android Debug,O=Android,C=US" -
↪validity 9999
```

Click on *Editor -> Editor Settings* in Godot and select the *Export/Android* section. Here, you need to set the paths to the Android SDK applications on your system and the location of the keystore you just created.



Now you're ready to export. Click on *Project -> Export* and add a preset for Android (see above).

Click the "Export Project" button and Godot will build an APK you can download on your device. To do this on the command line, use the following:

```
adb install dodge.apk
```

Note: Your device may need to be in *developer mode*. Consult your device's documentation for details.

If your system supports it, connecting a compatible Android device will cause a "One-click Deploy" button to appear in Godot's playtest button area:

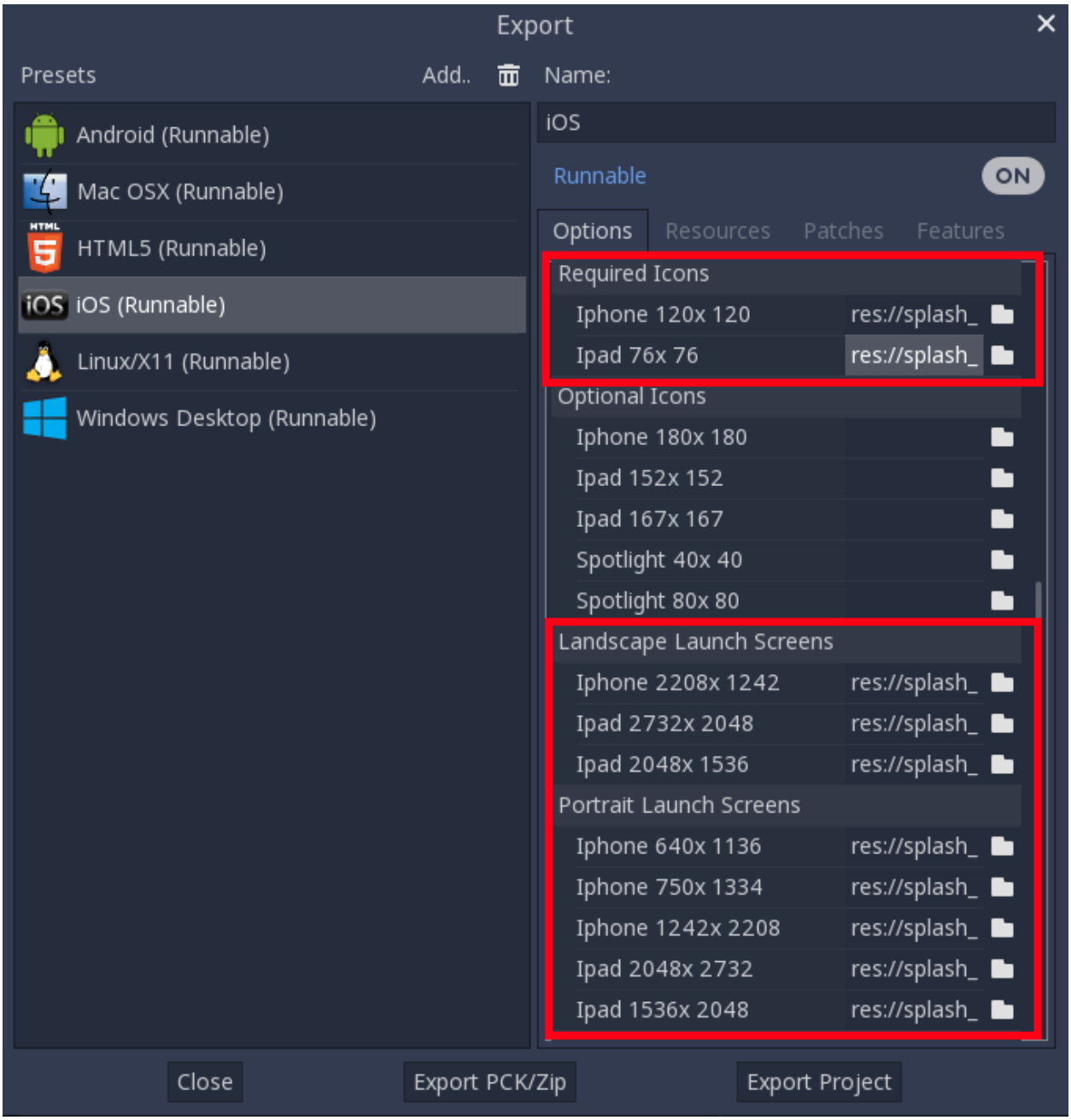
Clicking this button builds the APK and copies it onto your device in one step.

iOS

Note: In order to build your game for iOS, you must have a computer running MacOS with Xcode installed.

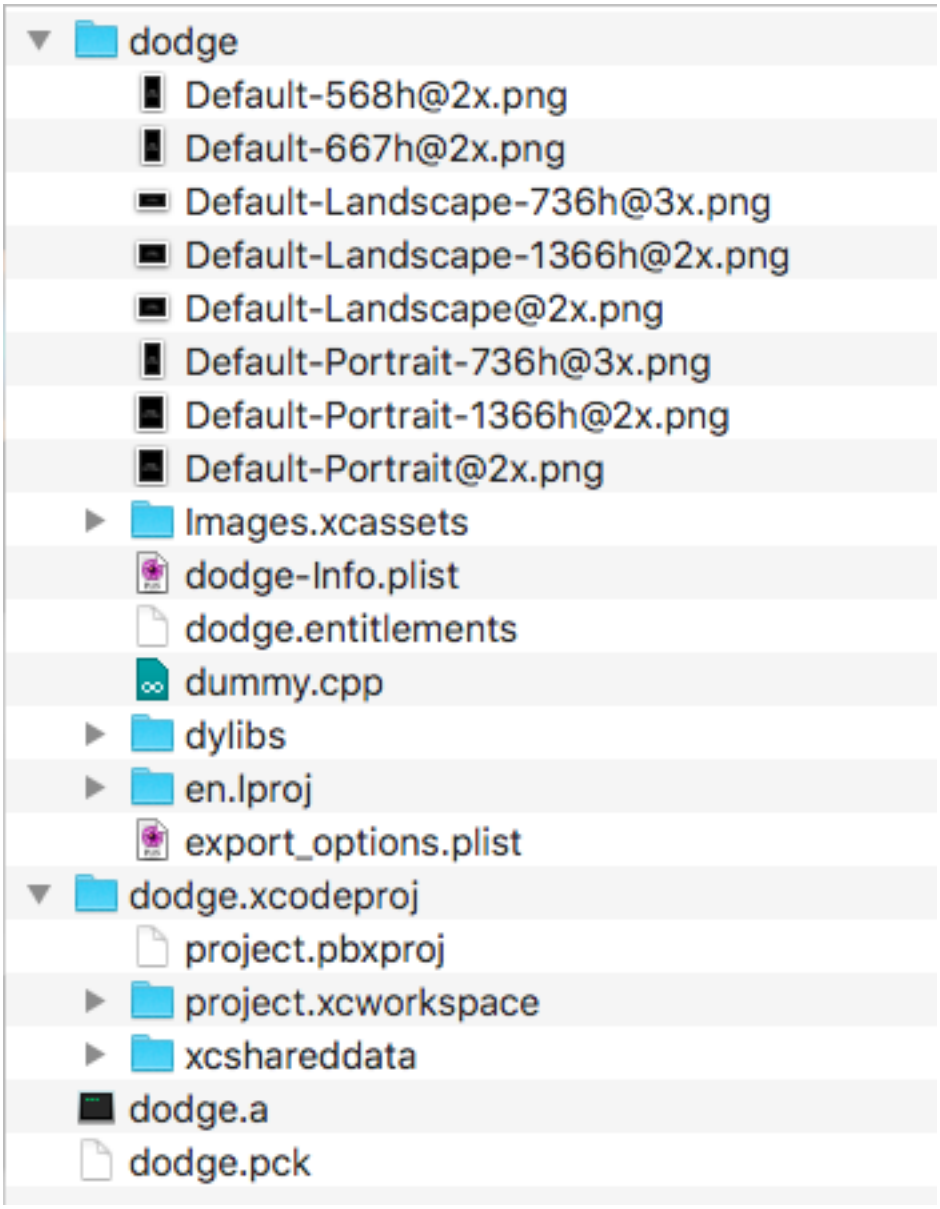
Before exporting, there are some settings that you *must* complete for the project to export successfully. First, the "App Store Team Id", which you can find by logging in to your Apple developer account and looking in the "Membership" section.

You must also provide icons and splash screen images as shown below:



Click “Export Project” and select a destination folder.

Once you have successfully exported the project, you’ll find the following folders and files have been created in your selected location:

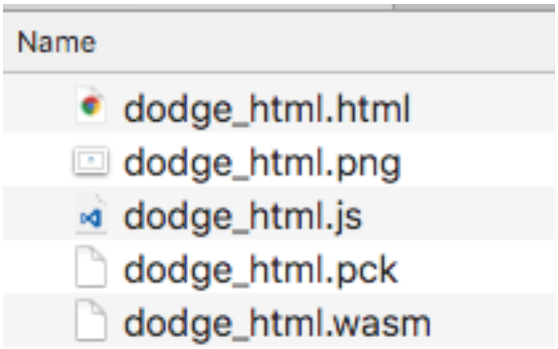


You can now open the project in Xcode and build the project for iOS. Xcode build procedure is beyond the scope of this tutorial. See https://help.apple.com/xcode/mac/current/#/devc8c2a6be1 for more information.

HTML5 (web)

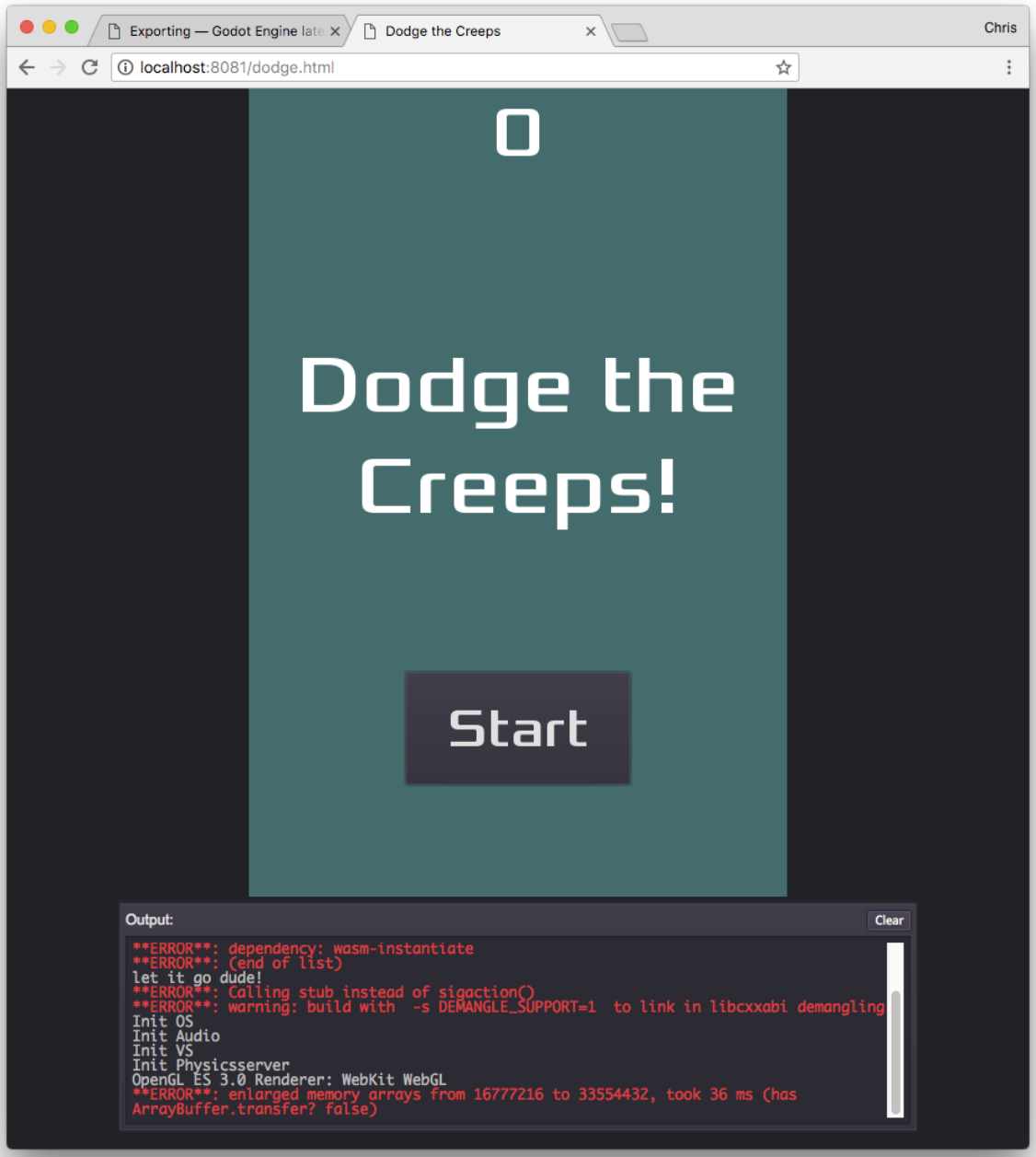
Click "Export Project" on the HTML5 preset. We don't need to change any of the default settings.

When the export is complete, you'll have a folder containing the following files:



Viewing the *.html* file in your browser lets you play the game. However, you can't open the file directly, it neds to be served by a web server. If you don't have one set up on your computer, you can use Google to find suggestions for your specific OS.

Point your browser at the URL where you've placed the html file. You may have to wait a few moments while the game loads before you see the start screen.



The console window beneath the game tells you if anything goes wrong. You can disable it by setting "Export With Debug" off when you export the project.

Note: Browser support for WASM is not very widespread. Firefox and Chrome both support it, but you may still find some things that don't work. Make sure you have updated your browser to the most recent version, and report any bugs you find at the Godot Github repository.

2.9 Godot's design philosophy

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

Every game engine is different and fits different needs. Not only do they offer a range of features, the design of each engine is unique. This leads to different workflows and different ways to form your games' structures. This all stems from their respective design philosophies.

This page is here to help you understand how Godot works, starting with some of its core pillars. It is not a list of available features, nor is it an engine comparison. To know if any engine can be a good fit for your project, you need to 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.

2.9.1 Object-oriented design and composition

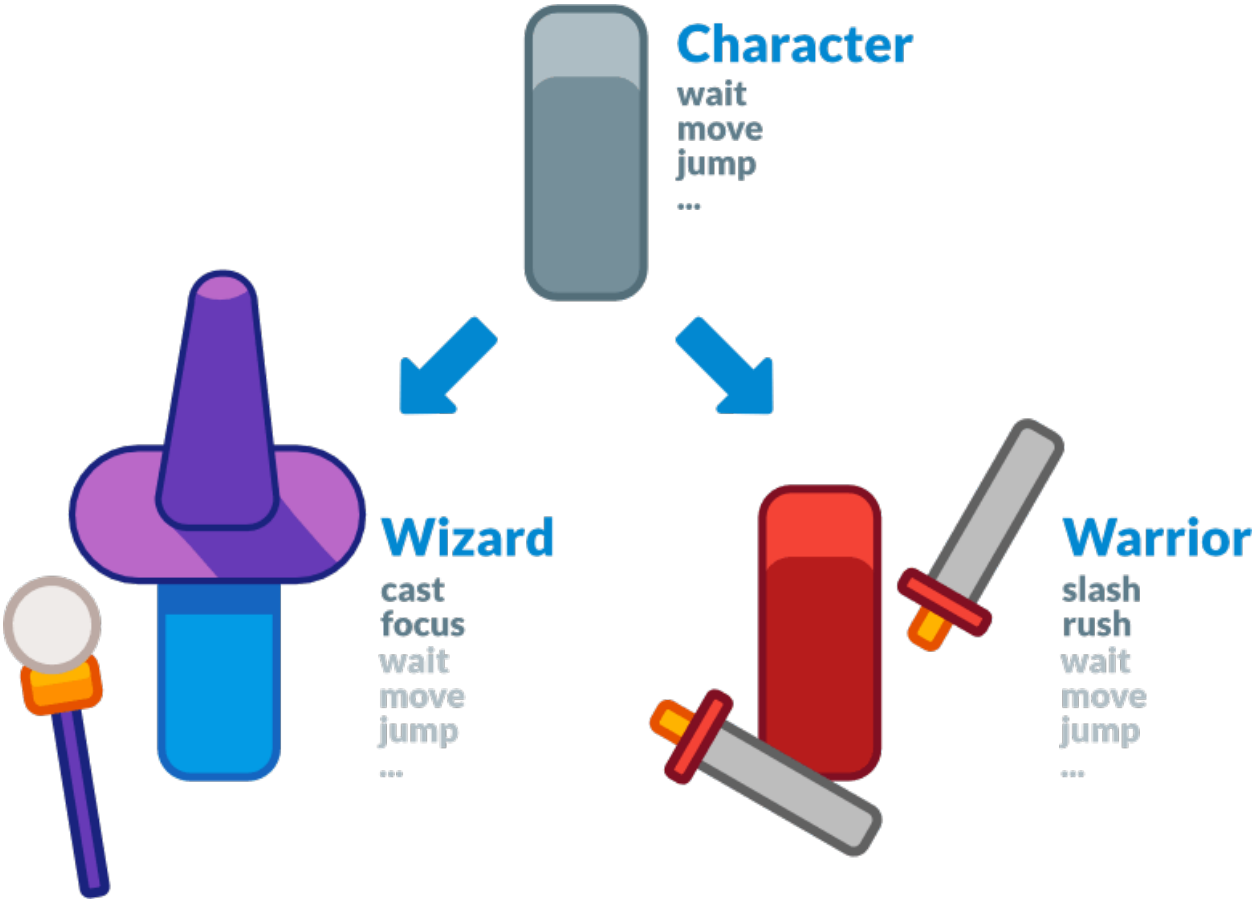
Godot embraces object-oriented design at its core with its flexible scene system and Node hierarchy. It tries to stay away from strict programming patterns to offer an intuitive way to structure your game.

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.

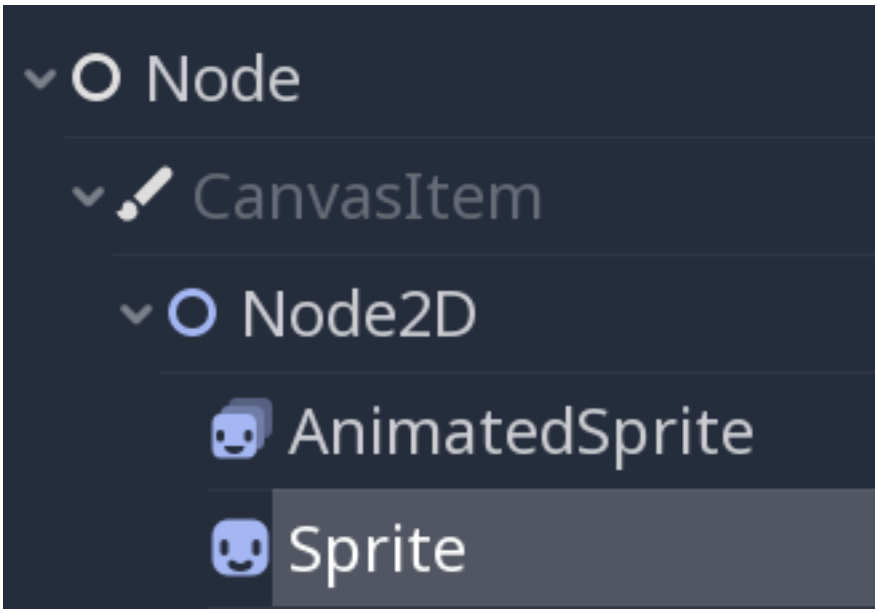
On top of that, you can **inherit** from any scene.

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 by 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 components like collision shapes, they're the exception, not the norm.



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.

2.9.2 All-inclusive package

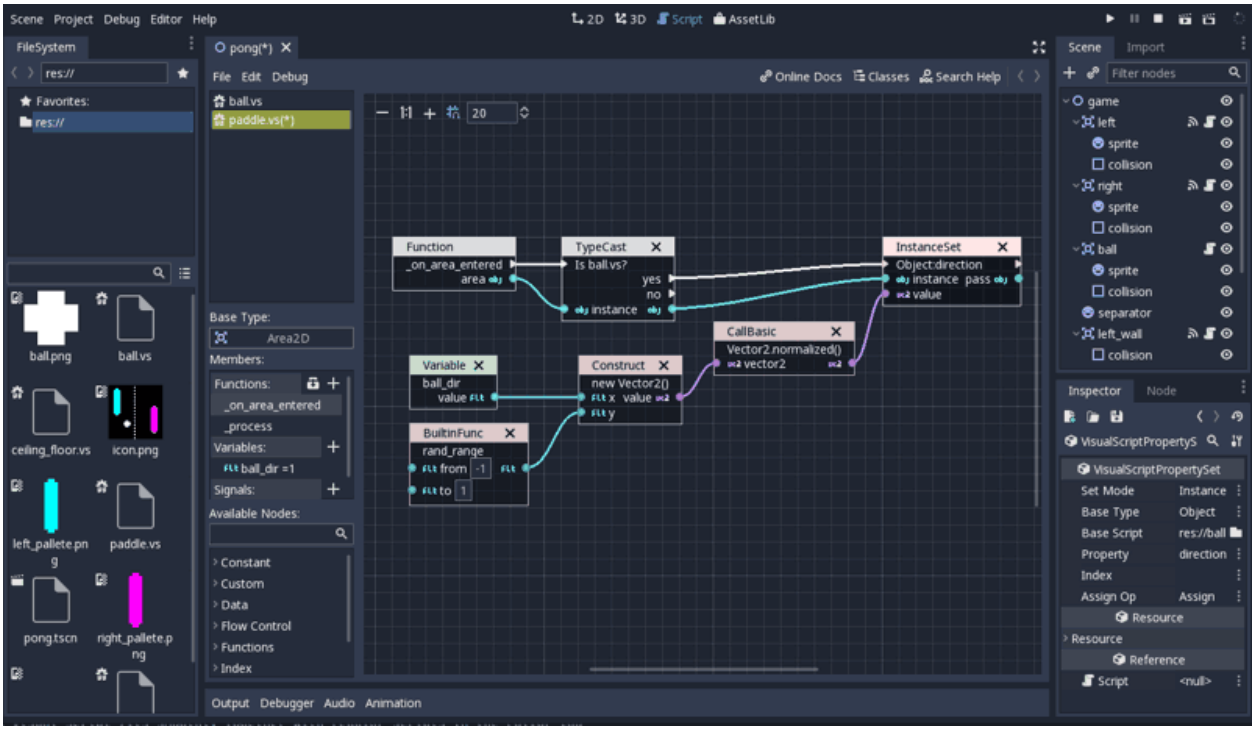
Godot tries to provide its own tools to answer most common needs. It has a dedicated scripting workspace, an animation editor, a tilemap editor, a shader editor, a debugger, a profiler, the ability to hot-reload locally and on remote devices, etc.

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 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 and game designers, and they're tightly integrated in the engine and the editor.

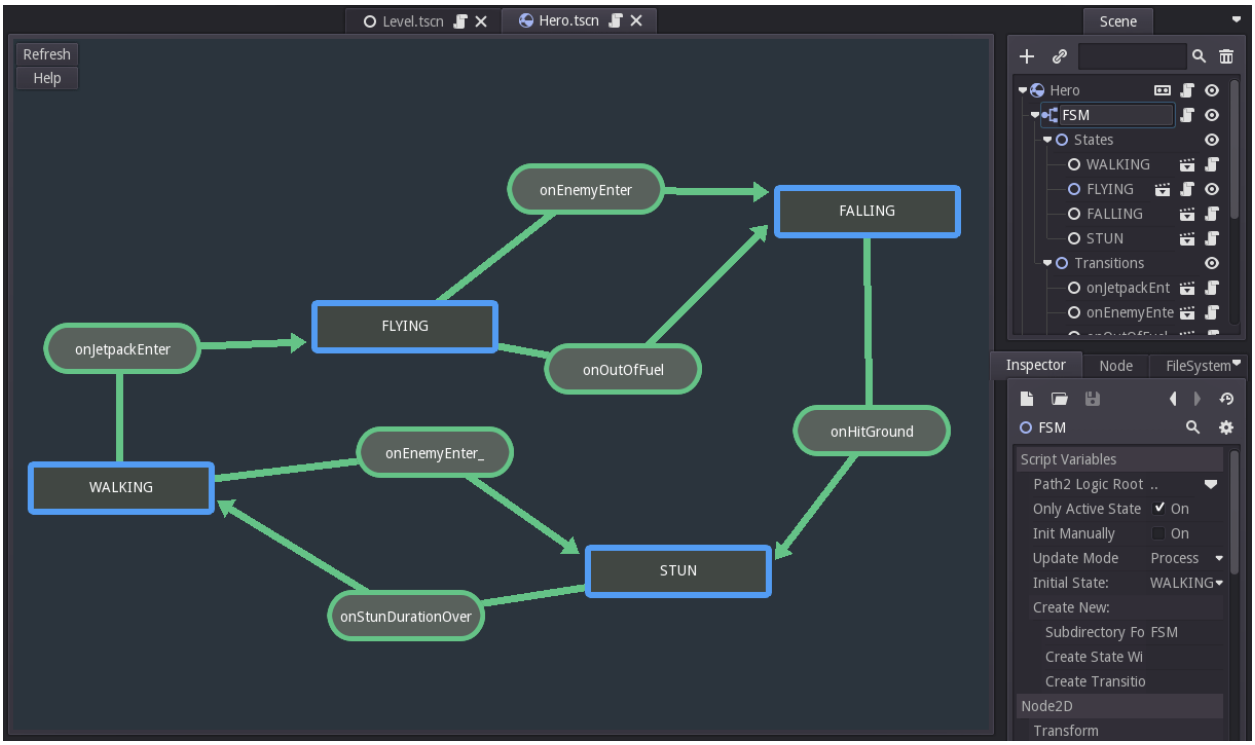
GDscript lets you write simple code using Python-like syntax, yet it detects types and offers a static-language's quality of auto-completion. It is also optimized for gameplay code with built-in types like Vectors and Colors.

Note that with GDNative, you can write high-performance code using compiled languages like C, C++, Rust, or Python (using the 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 the 3D workspace doesn't feature as many tools as the 2D workspace. You'll need external programs or add-ons to edit terrains, animate complex characters, and so on. 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

2.9.3 Open-source

Godot offers a fully open-source codebase under the **MIT license.** This means all the technologies that ship with it have to be Free (as in freedom) 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.

2.9.4 Community-driven

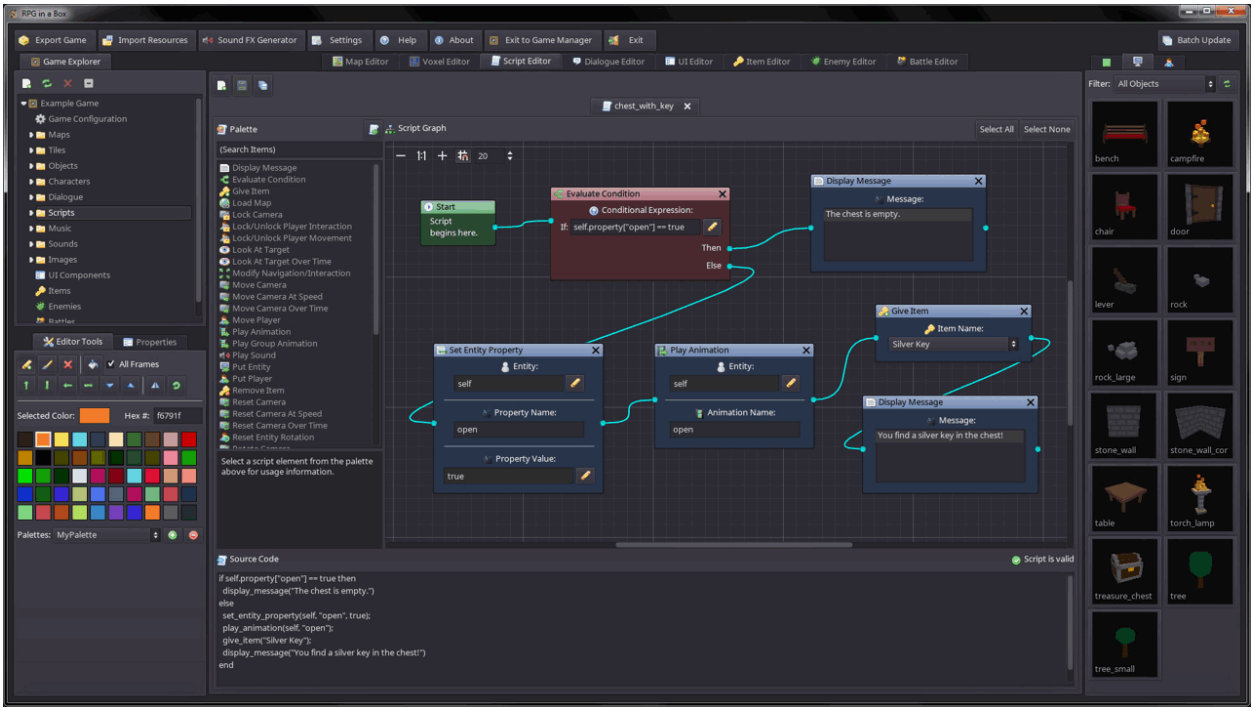
Godot is made by its community, for the community, and for all game creators out there. It's the needs of the users and open discussions that drive the core updates. New features from the core developers often focus on what will benefit the most users first.

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.

2.9.5 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 **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 import and export plugins, create plugins like custom level editors, or create scripts with the same nodes and API you use in your projects.

2.9.6 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.

2.10 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

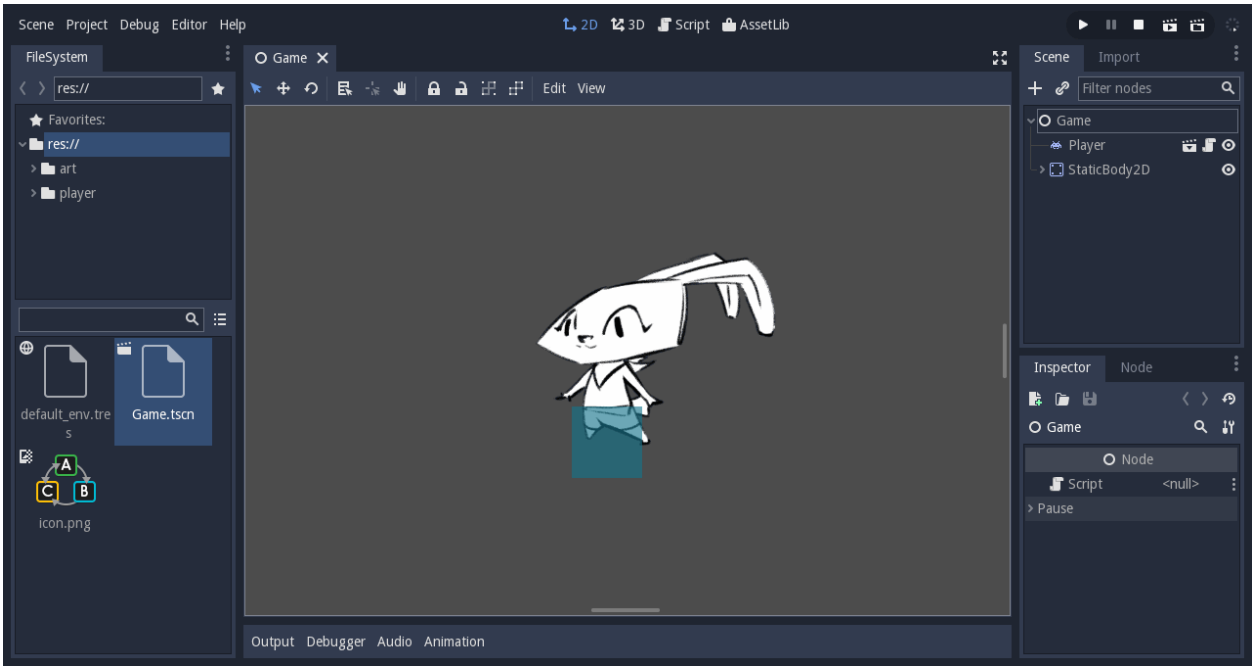


Fig. 1: Godot's editor is made with the engine's UI framework

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.

2.10.1 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.

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

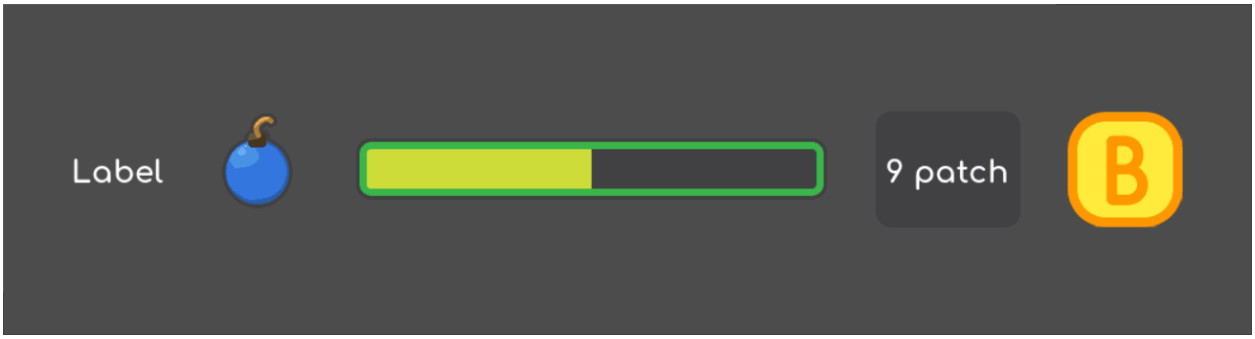


Fig. 2: The 5 most common Control nodes for UI design

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:

- `Scale On Expand (compat)` scales the texture to fit the node's bounding rectangle, only if `expand` property is `true`; otherwise, it behaves like `Keep` mode. Default mode for backwards compatibility.
- `Scale` scales the texture to fit the node's bounding rectangle
- `Tile` makes the texture repeat, but it won't scale
- `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
- `Keep Aspect` and `Keep Aspect Centered` scales the texture but force it to remain its original aspect ratio, in the top left corner or the center of the frame respectively
- `Keep Aspect Covered` works just like `Keep Aspect Centered` but the shorter side fits the bounding rectangle and the other one clips to the node's limits

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

Fig. 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.

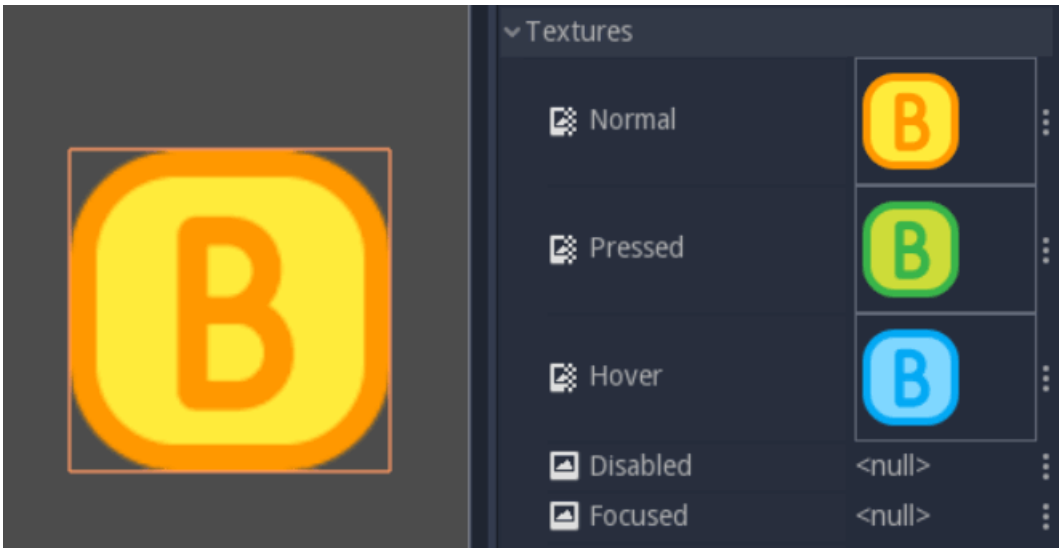


Fig. 4: TextureButton and its 5 texture slots

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.

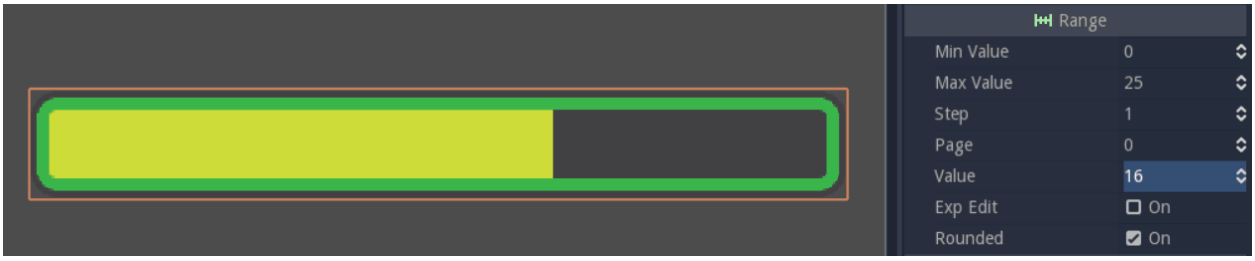


Fig. 5: TextureProgress bar, two thirds filled

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.

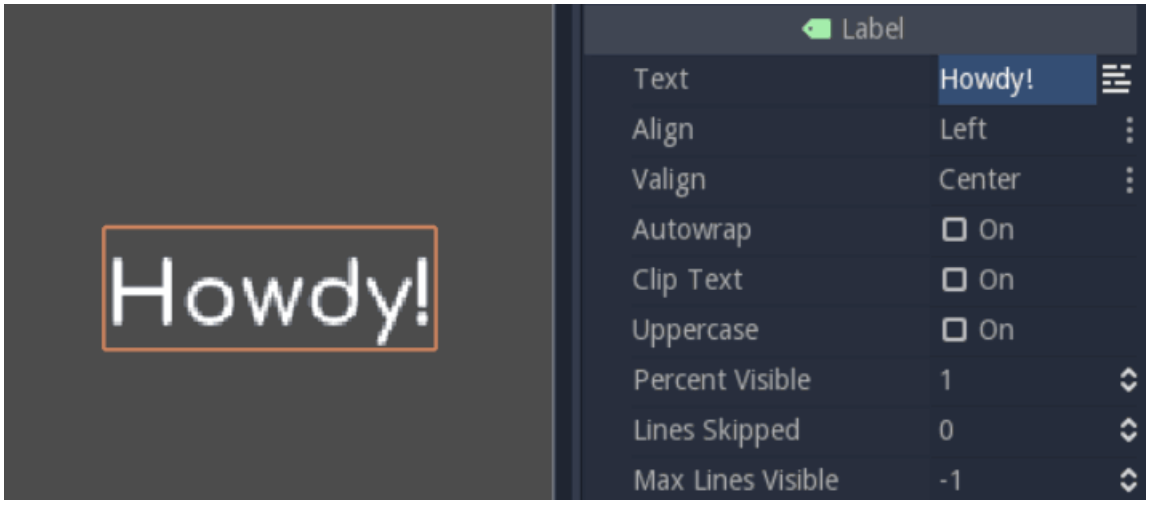


Fig. 6: Picture of a Label

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 useful to build panels, dialogue boxes and scalable backgrounds for your UI.

Fig. 7: NinePatchRect scaled with the min_size property

2.10.2 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.

2.10.3 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. 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.

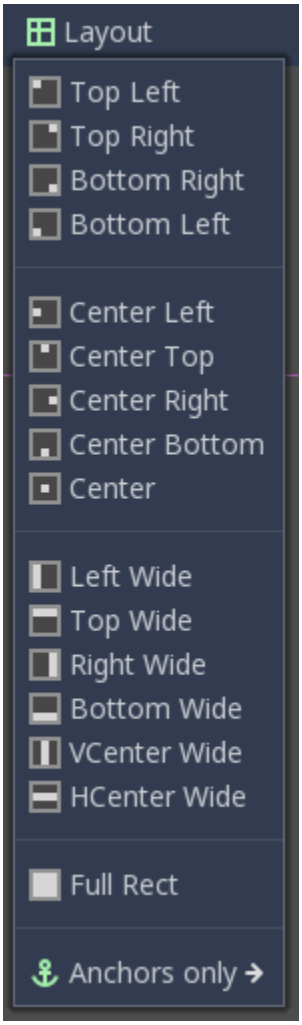


Fig. 9: The layout menu in the viewport

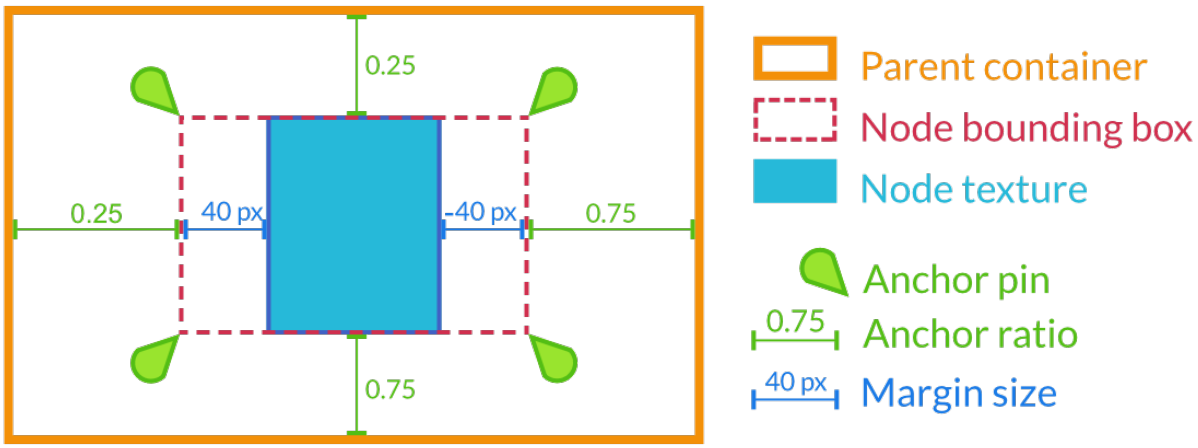


Fig. 10: Margins are relative to the anchor position, which is relative to the anchors. In practice, you'll often let the container update margins for you

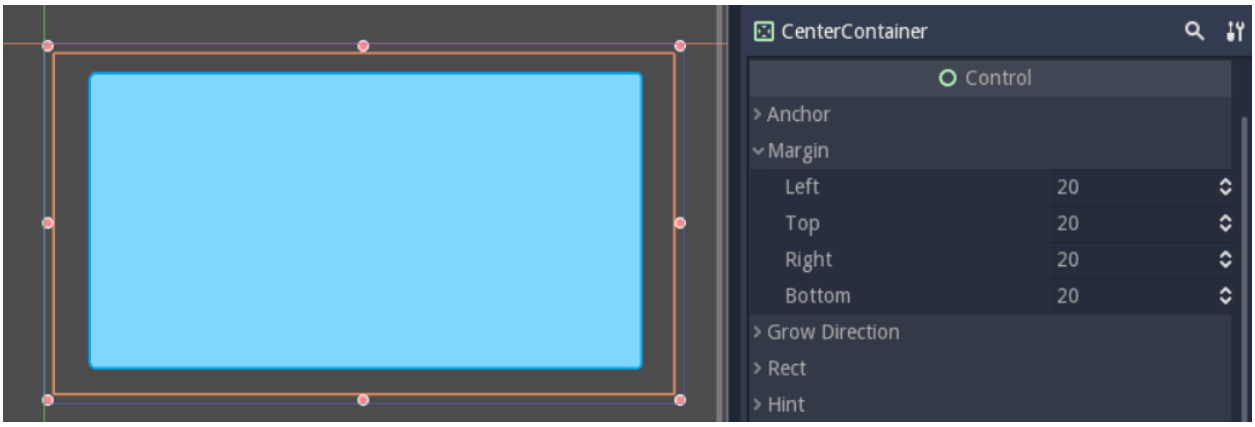


Fig. 11: Margins on a CenterContainer set to the "Full Rect" anchor

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 TextureRect nodes in an HBoxContainer, with the "Fill" flags on both axes, 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.

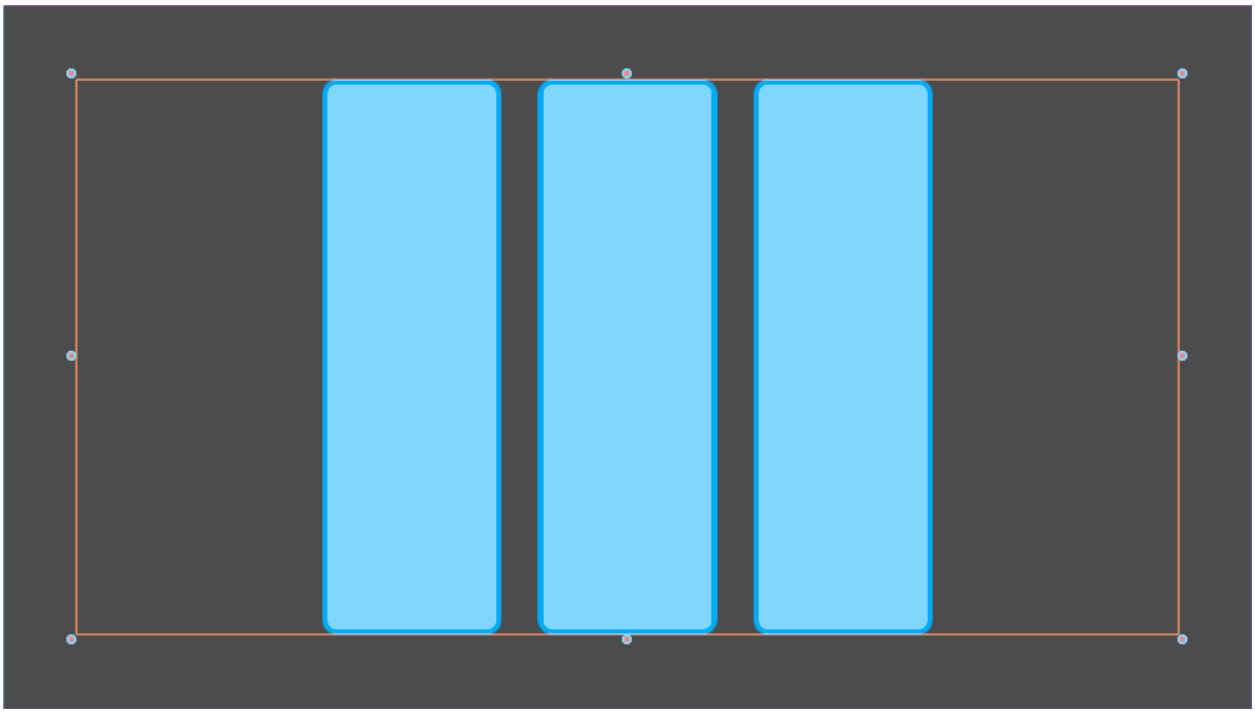


Fig. 12: 3 UI elements in an HBoxContainer, they align horizontally

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.

2.10.4 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

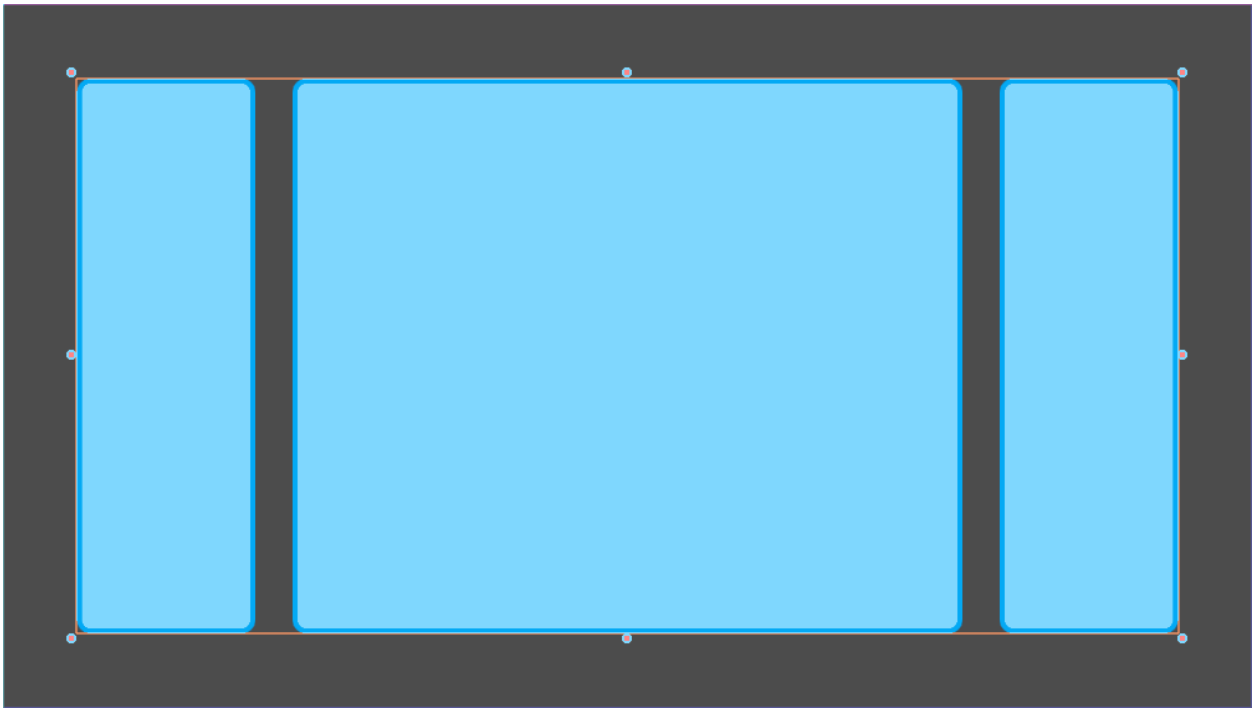


Fig. 13: The same example as above, but the center node has the "Expand" size flag

- 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

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.

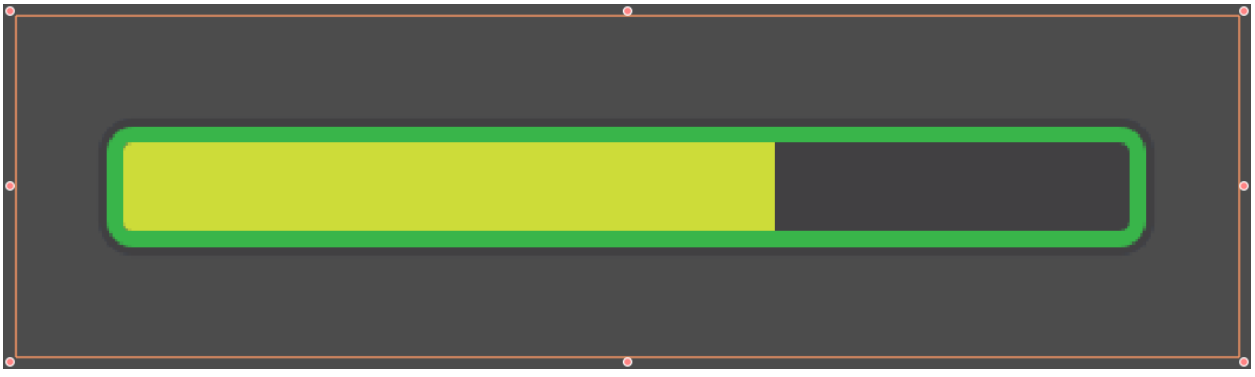


Fig. 14: CenterContainer in action. The life bar centers inside its parent container.

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,

Fig. 15: The MarginContainer adds a 40px margin around the Game User Interface

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.

Fig. 16: The HBoxContainer horizontally aligns UI elements

VBoxContainer automatically arranges its children into a column. It puts them one after the other. If you use the 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÷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. 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, head to the *GUI section* of the docs.

2.11 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 *Control the game's UI with code*

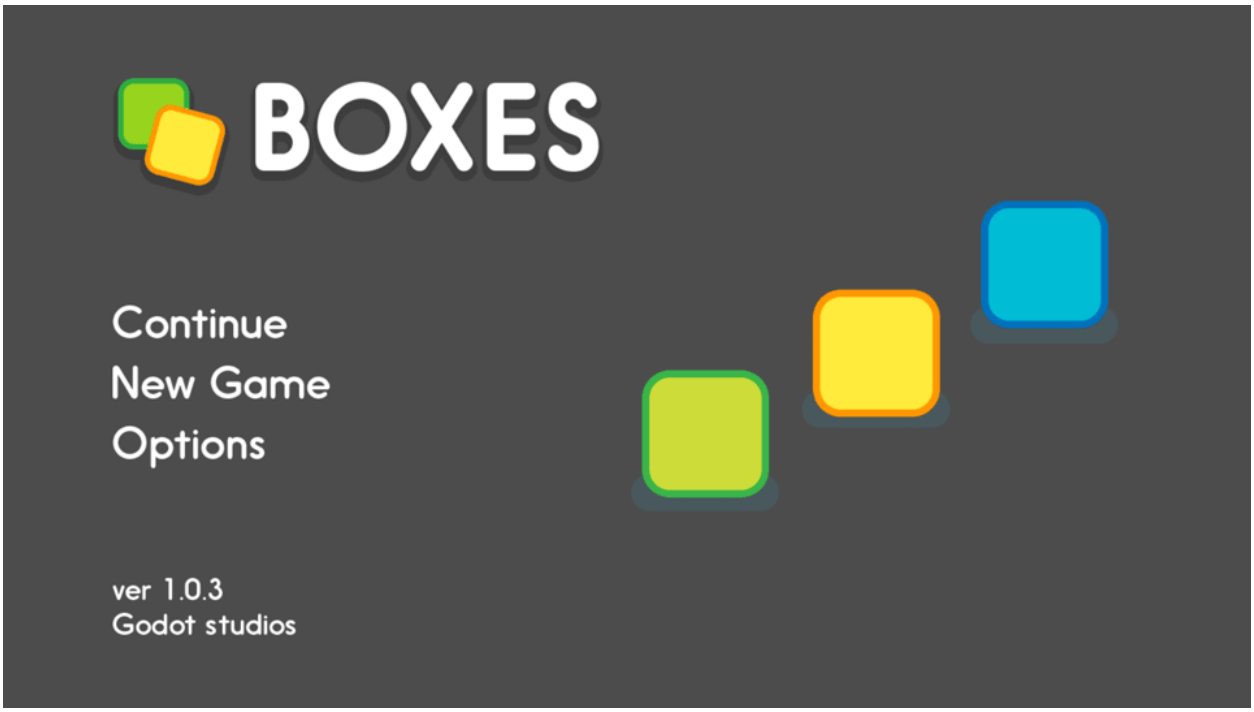


Fig. 18: The GUI you're going to create

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. You'll find all the sprites in the `start/assets/main_menu`` folder.

2.11.1 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.

Fig. 19: The UI's rough plan or mockup

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
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.

2.11.2 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

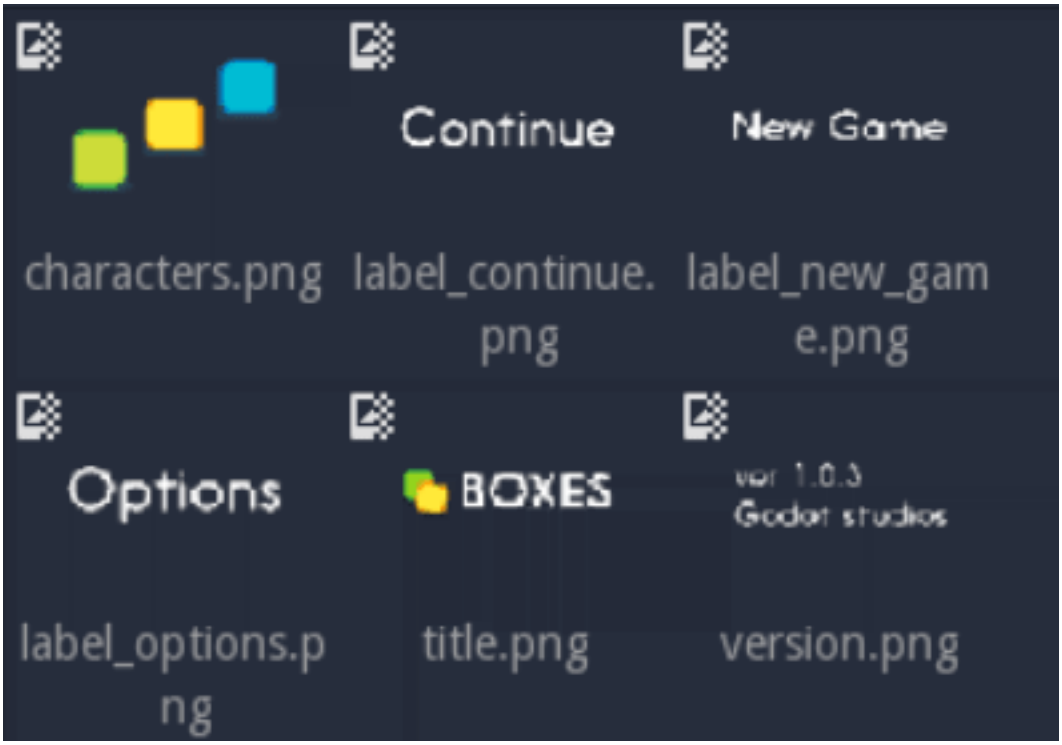


Fig. 20: The files you'll find in Godot. The graphics look cleaner than on the rough design, but they're still placeholders

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`.

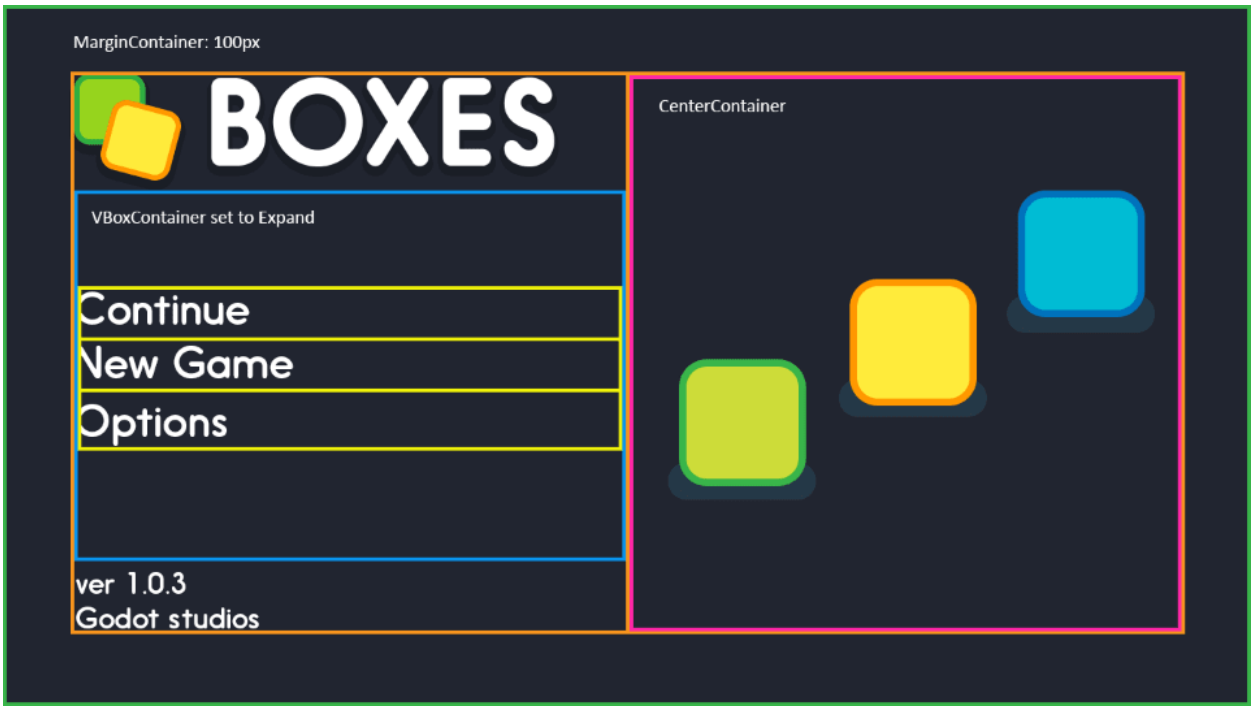


Fig. 21: Interface building blocks, broken down using the three rules of thumb

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

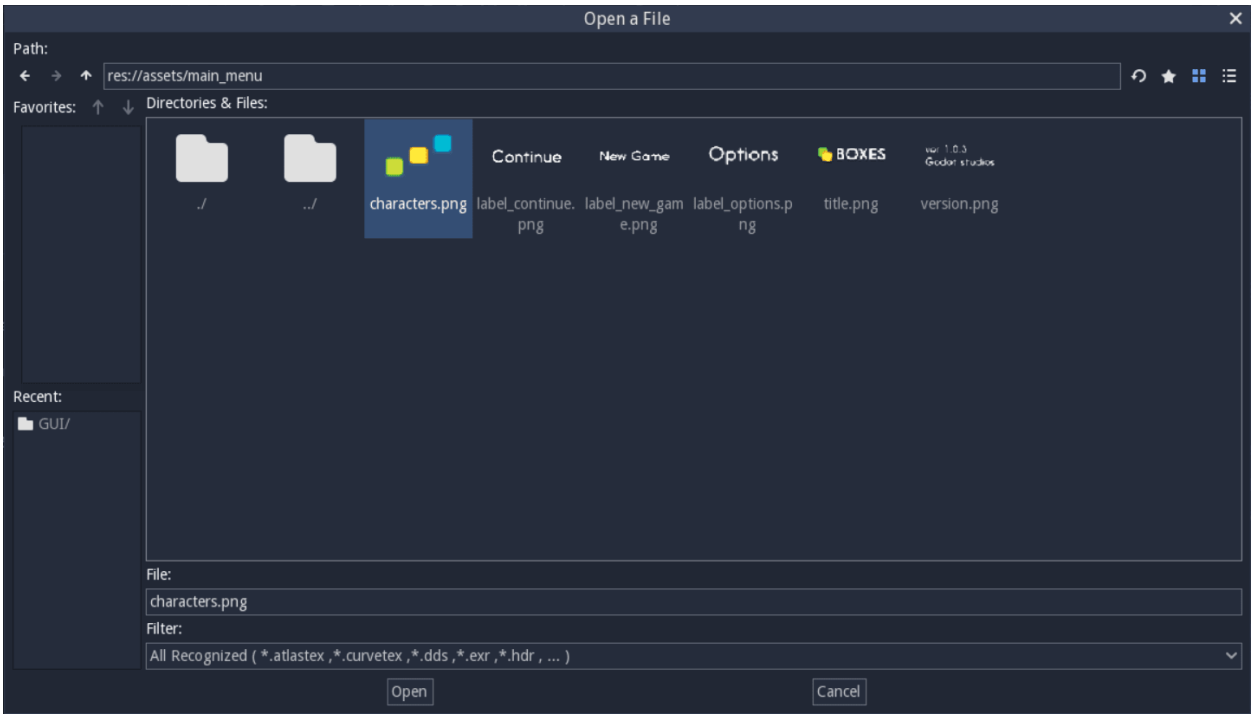


Fig. 22: The file browser lets you find and load textures

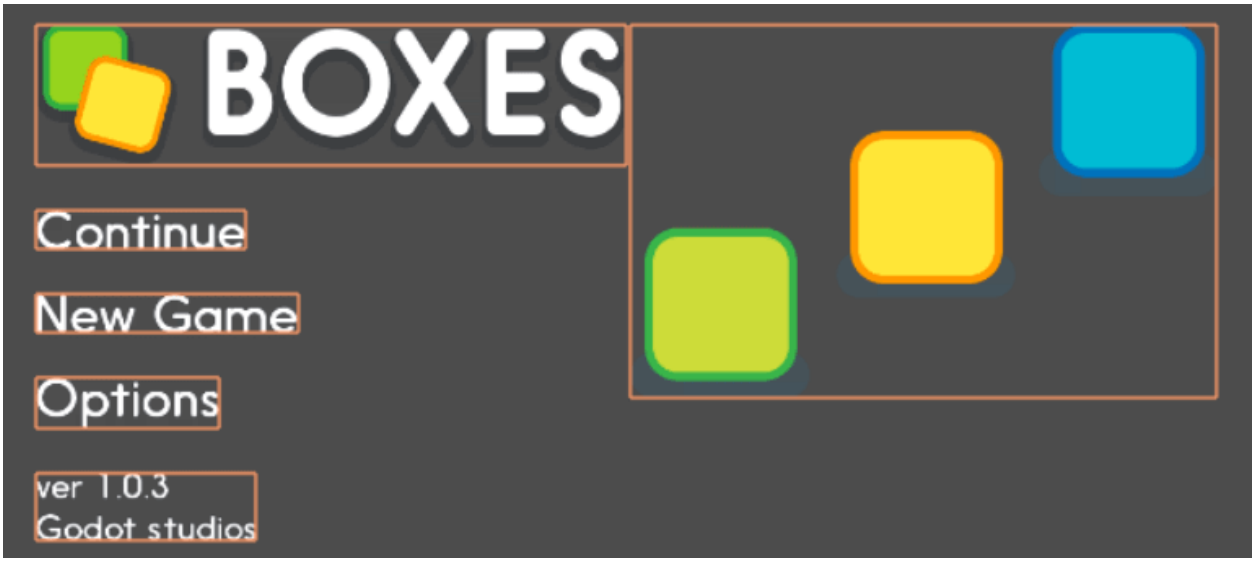


Fig. 23: The six nodes, with textures loaded

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.

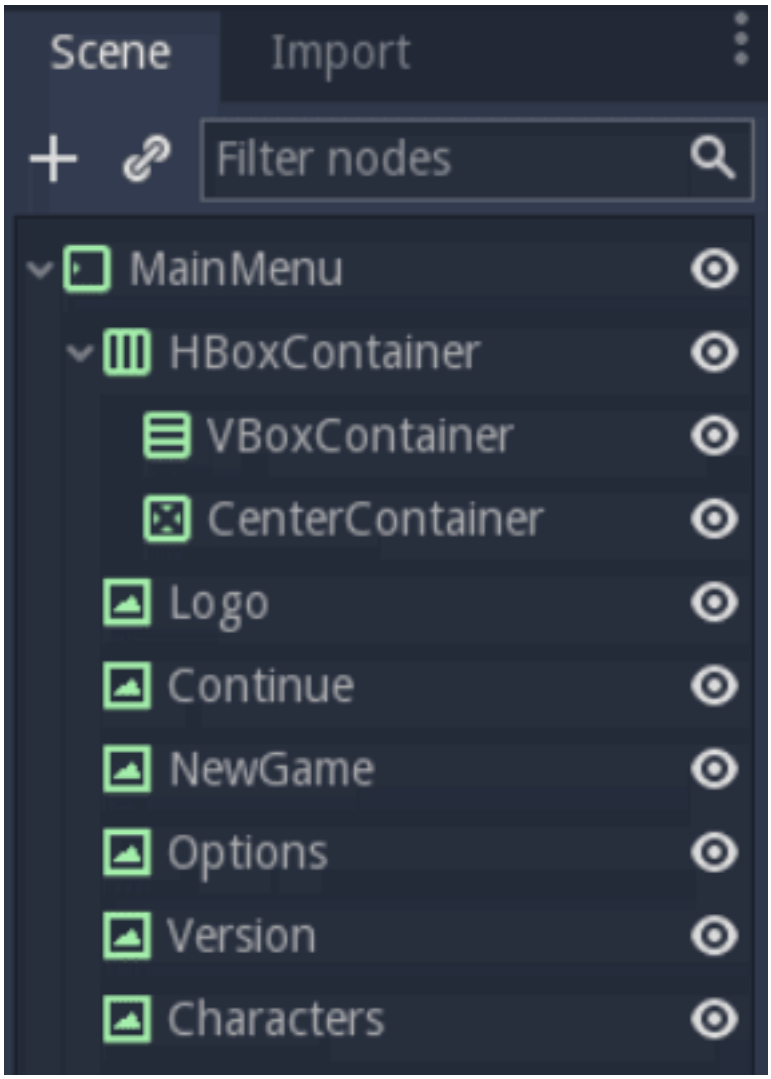


Fig. 24: You should have four nested containers, and the TextureRect nodes sitting aside from it

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 as a child of the `HBoxContainer`. Then in the Inspector, scroll down to the `Size Flags` category and click on the field to

Fig. 25: Containers automatically place and resize textures

the right of the `Vertical` property, and check `Expand`. Do the same for the `Horizontal` property. Finally drag and drop the Characters into the `CenterContainer`. The Characters element will center automatically.

Fig. 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.

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`.

To wrap things up, let's add some separation between the menu options. Expand the `Custom Constants` category

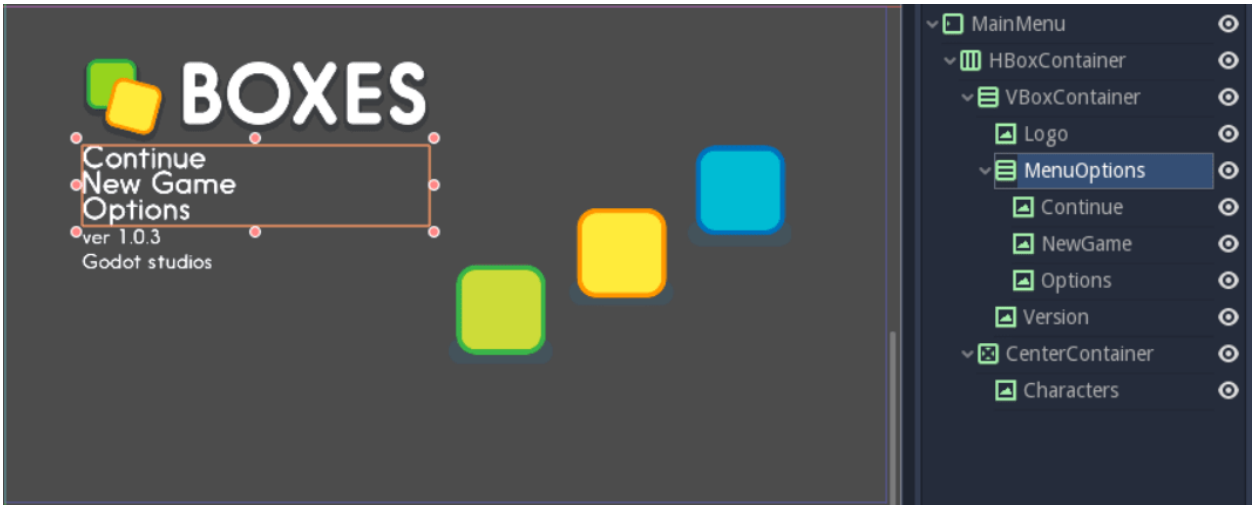


Fig. 27: Place the new container between the other two nodes to retain the UI's layout

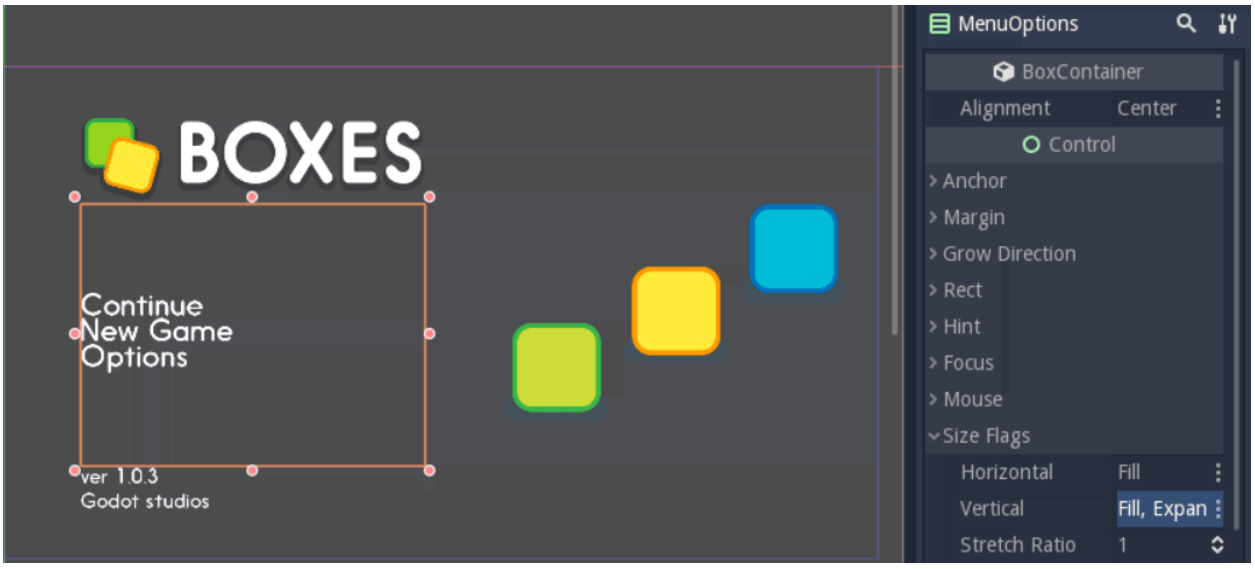


Fig. 28: The menu options should center vertically in the UI's left column

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.

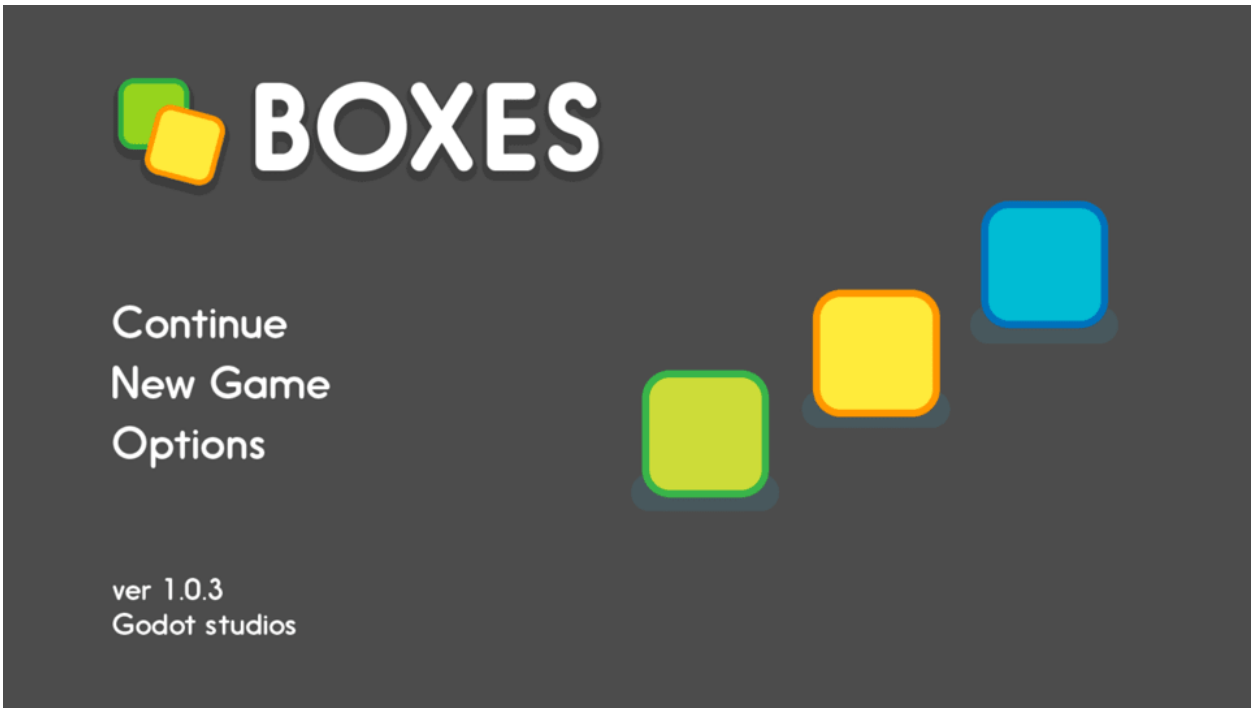


Fig. 29: The final interface

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*.

2.12 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:

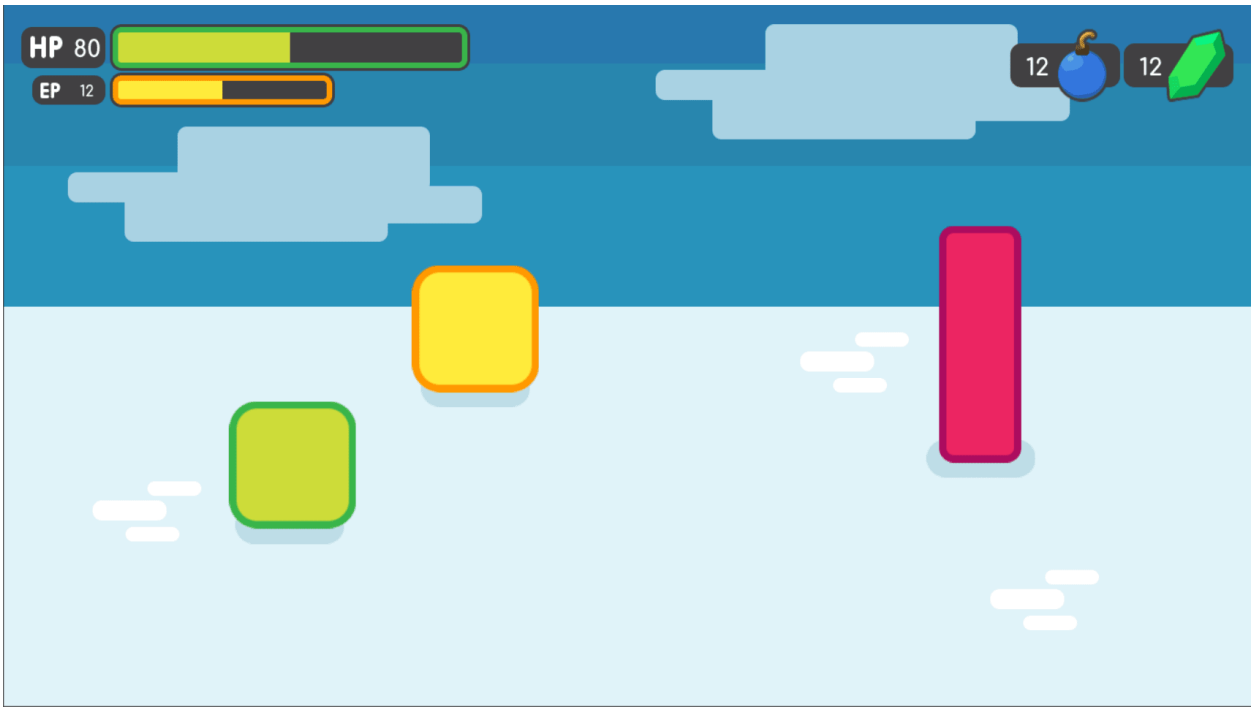


Fig. 30: The final result

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.

Note: You can watch this tutorial as a video on Youtube.

2.12.1 Breaking down the UI

Let's break down the final UI and plan the containers we'll use. As in the *Design a title screen*, 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.

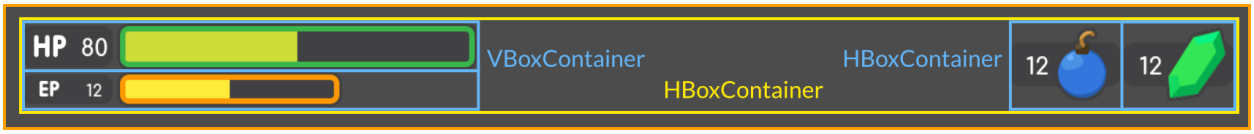


Fig. 31: We get a clean UI layout with only 4 containers

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.

2.12.2 Create the base GUI

There are two 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 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.

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 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,

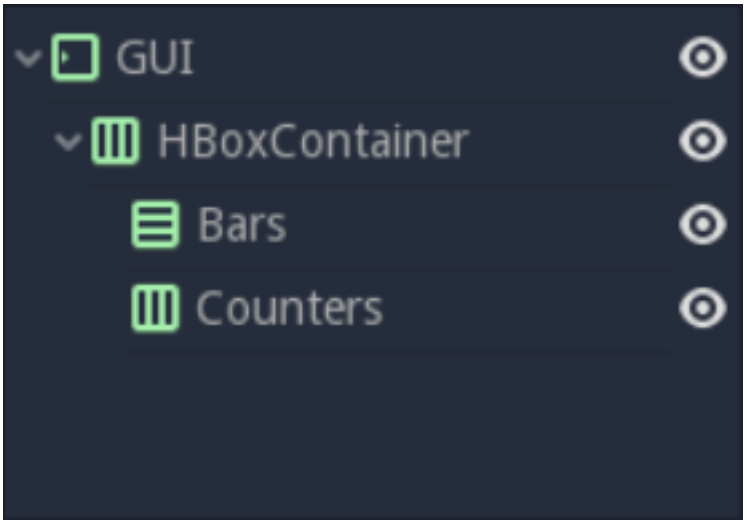


Fig. 32: You should have 4 containers that look like this

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.

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.

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`.

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`. Next find the `Rect` category in the Inspector and change the `Size` property to (50, 40) 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

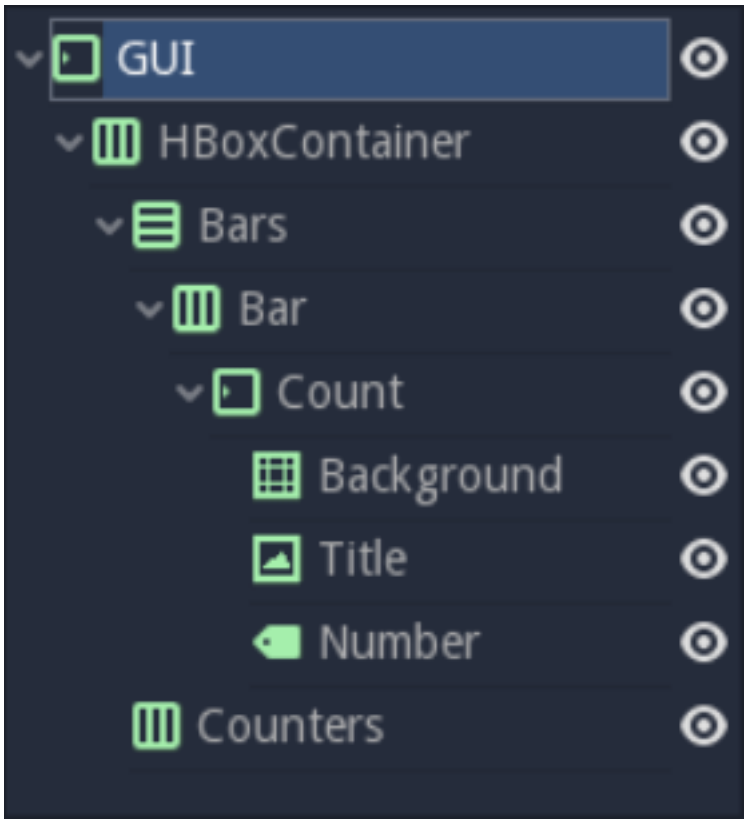


Fig. 33: Your scene tree should look like this. We're ready to throw in some textures

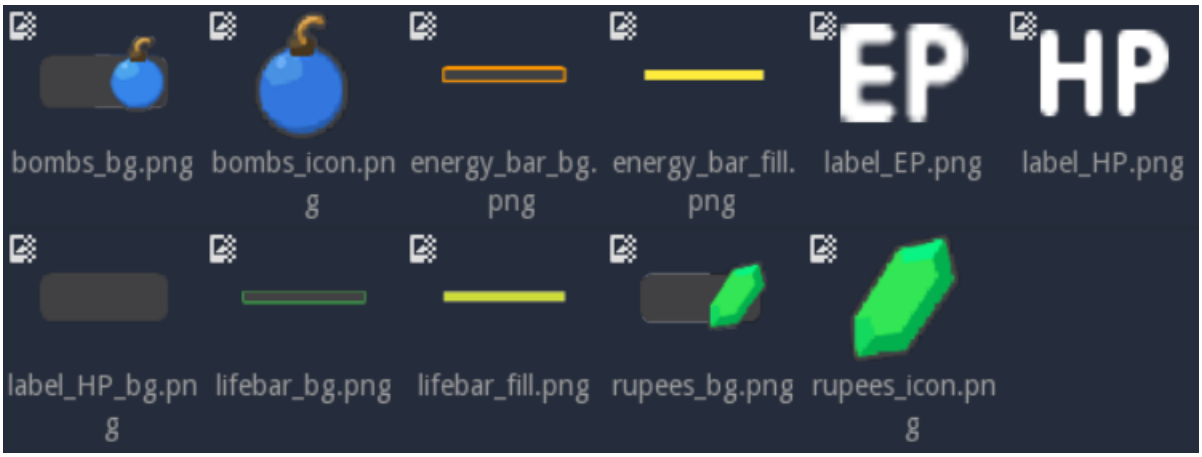


Fig. 34: You should see a list of textures that we'll use to skin our interface.

Fig. 35: If you select both nodes, you should see something like this

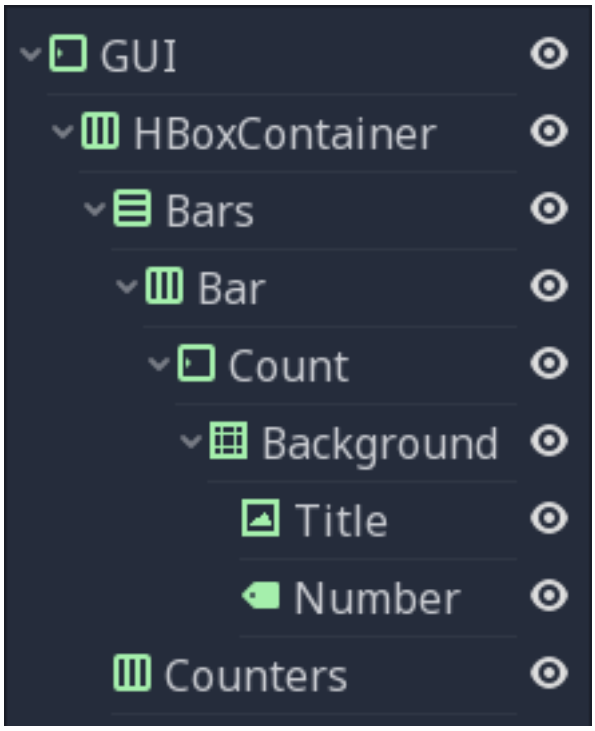


Fig. 36: By using the Background node as the two textures' parent, we take control away from the Count MarginContainer

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. 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.

With this, we 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.

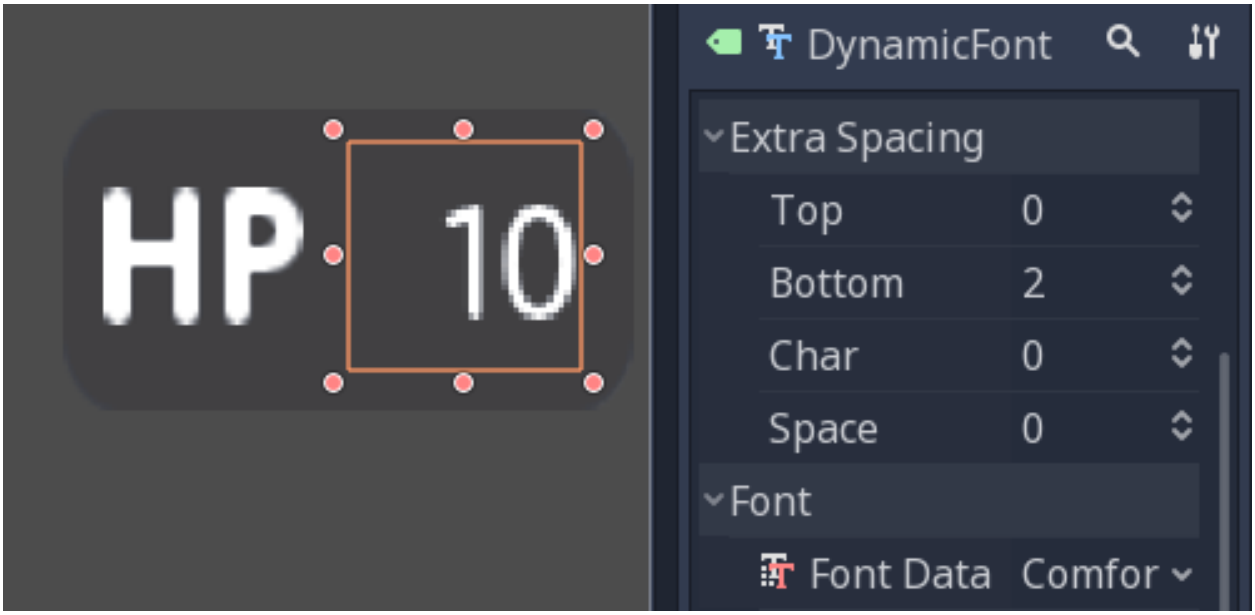


Fig. 38: With a Bottom value of 2 pixels, the Number aligns with the Title

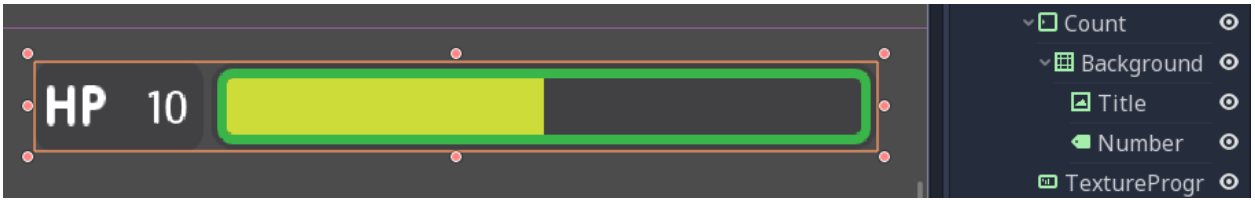


Fig. 39: That's it, our life bar is ready. This last part was quick, wasn't it? That's thanks to our robust container setup.

2.12.3 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`.

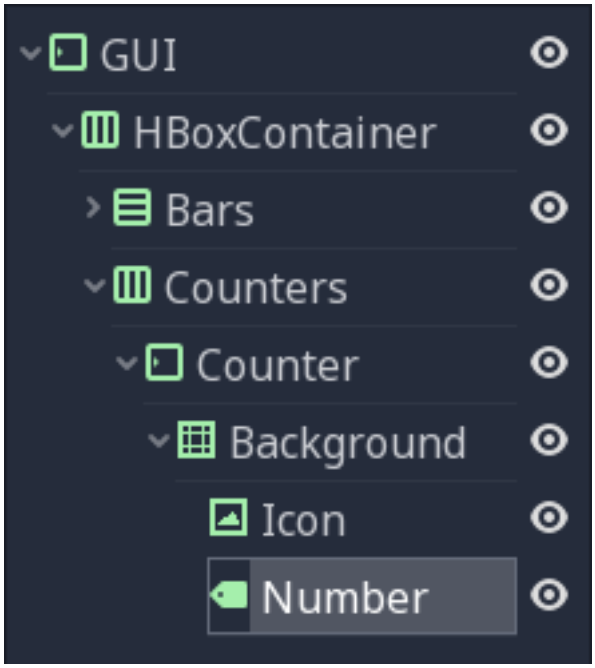


Fig. 40: Here's how your node tree should look so far

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 centre. Then resize its left edge a little bit to add some padding between the left edge of the `Background` and the text.

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`

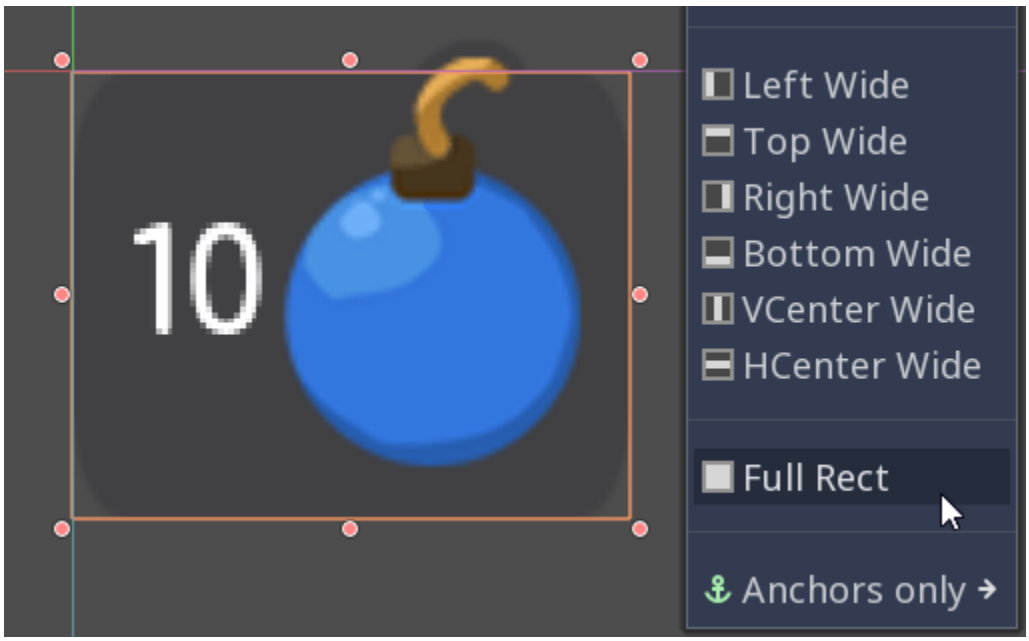


Fig. 41: The nodes anchor to the entire Background, but their position is off

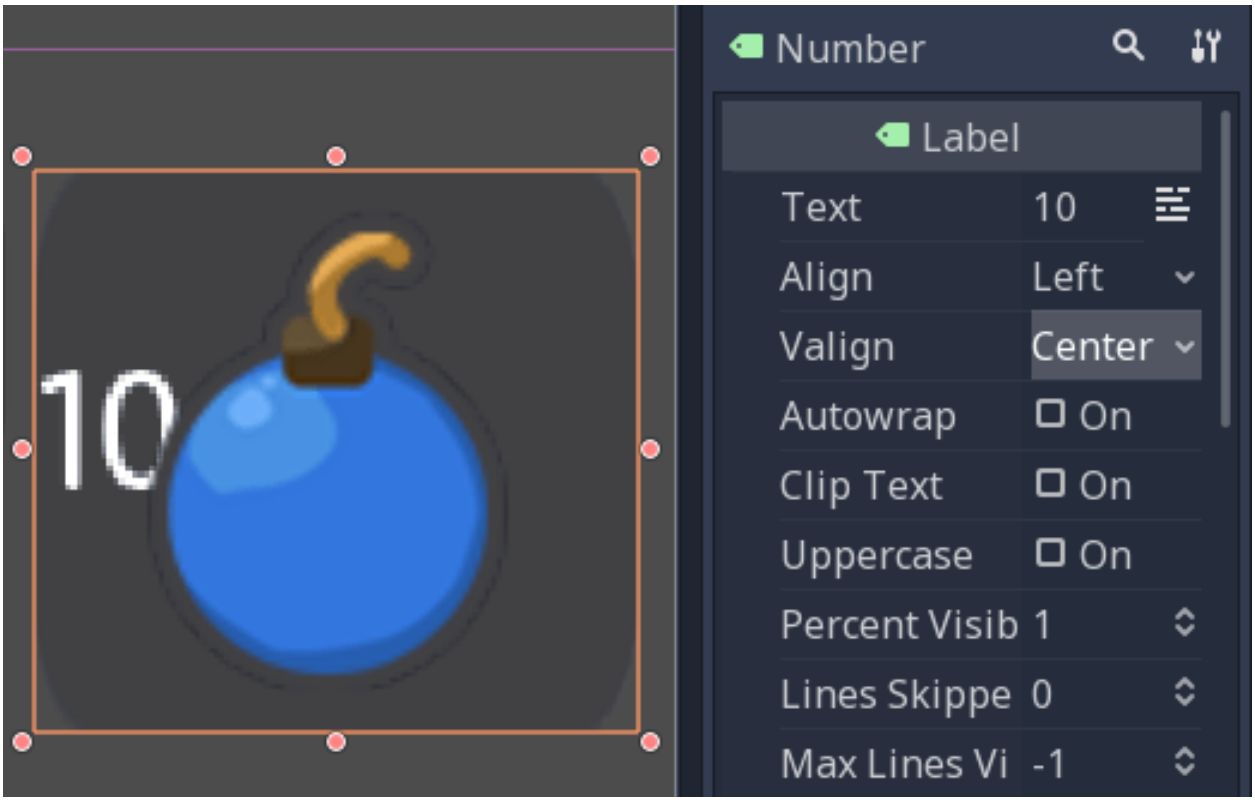


Fig. 42: The Number node aligned to the left and centre

Fig. 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

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.

Fig. 44: An expanding container eats all the space it can from its parent, pushing everything else along the way

2.12.4 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 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`.

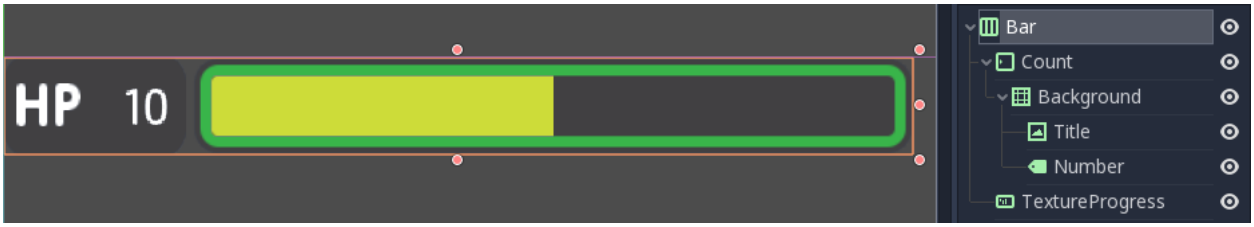


Fig. 45: With no extra changes, our Bar is ready to use

2.12.5 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**.

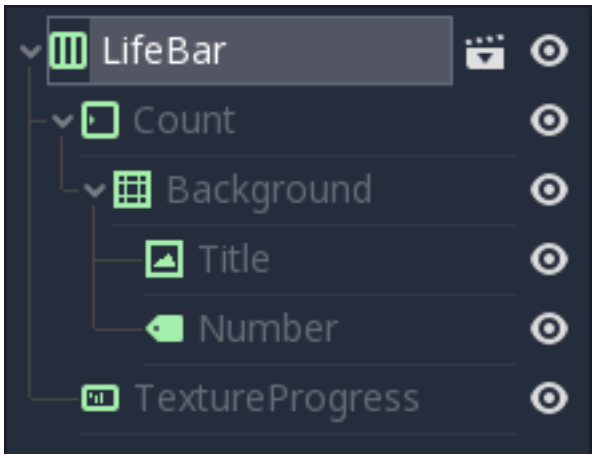


Fig. 46: Inherited scenes help us keep the GUI scene clean. In the end, we will only have containers and one node for each UI component.

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 overrides the parent's property. It's useful for 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`.

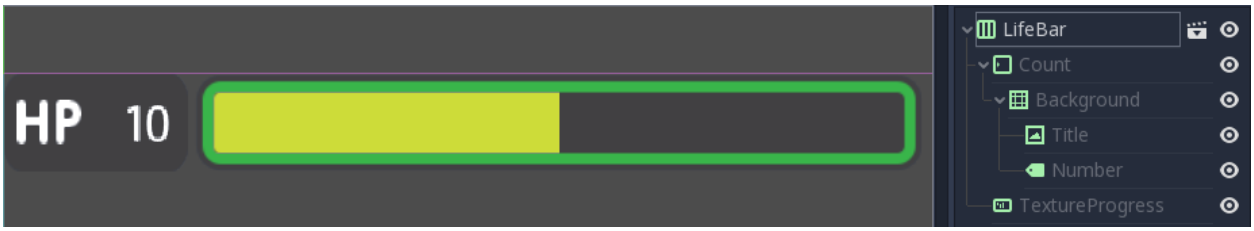


Fig. 47: You can't rename grey nodes. This tells you they have a parent scene

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.

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.

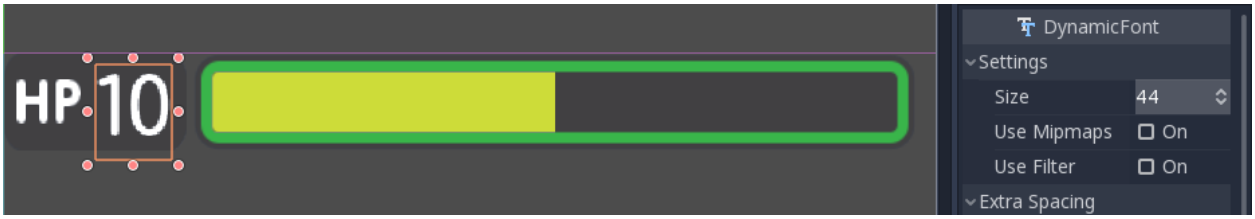


Fig. 48: If we change the font resource, all the nodes that use it are affected

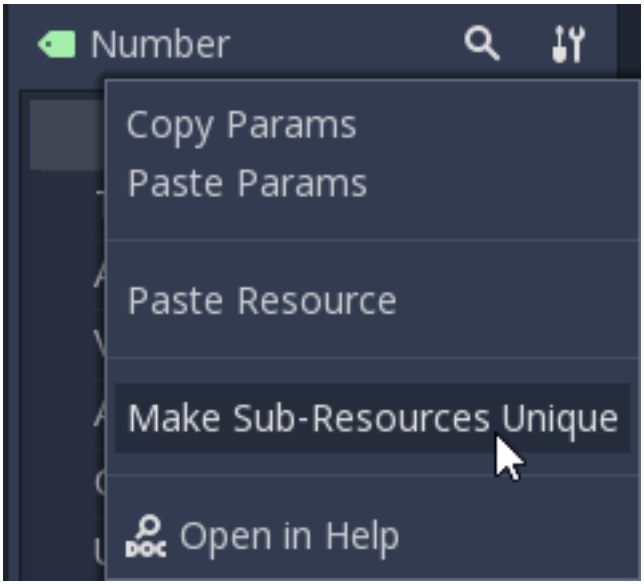


Fig. 49: Use this option to create unique copies of the resources for one node

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. 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. 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.

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.

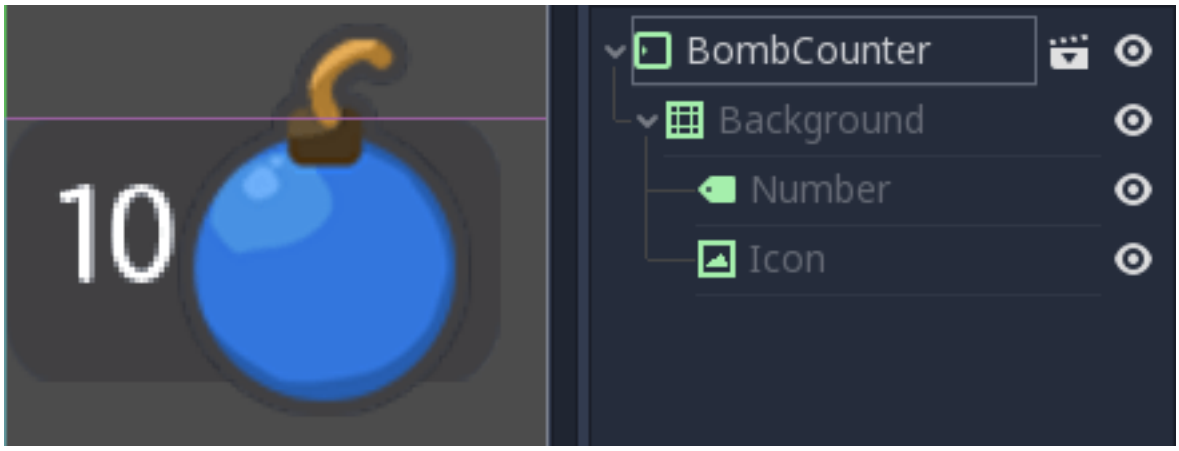


Fig. 52: The bomb counter is the same as the original Counter scene

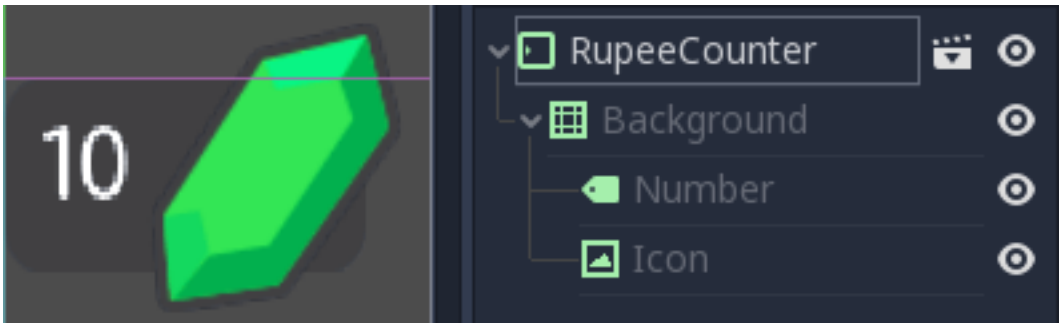


Fig. 53: The rupee counter should look about like this

2.12.6 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. 54: The LifeBar and the EnergyBar align automatically

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

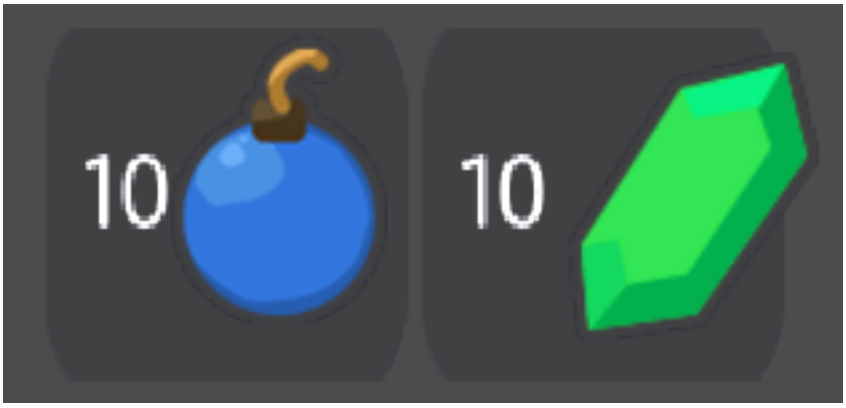


Fig. 55: The nodes resize to take all the available vertical space

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`.

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. 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 `LifeBar`.

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!

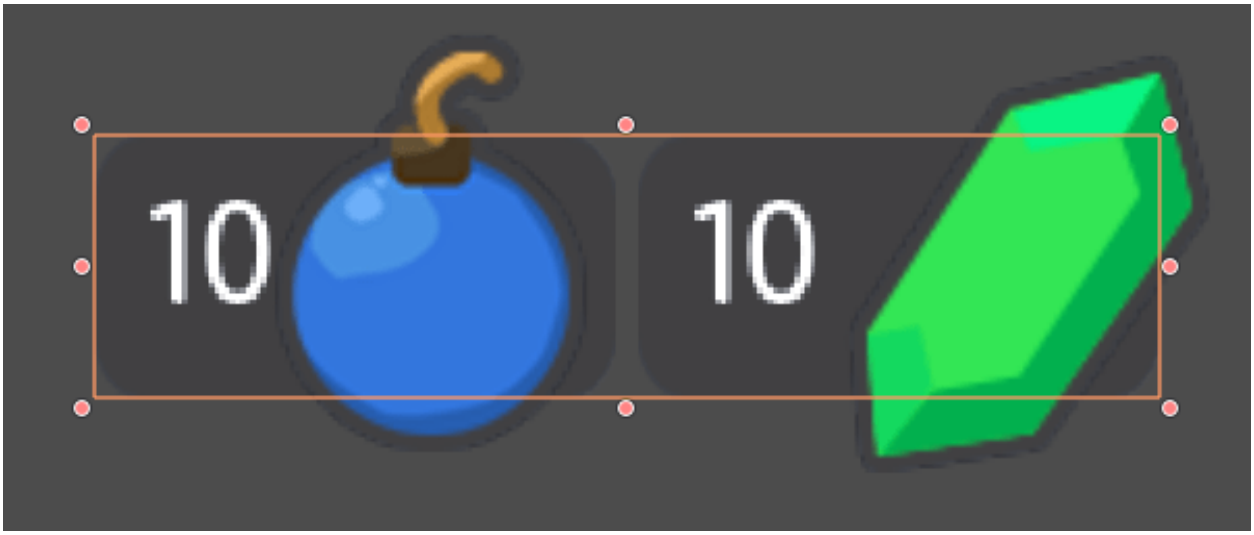


Fig. 56: Now both counters have a decent size

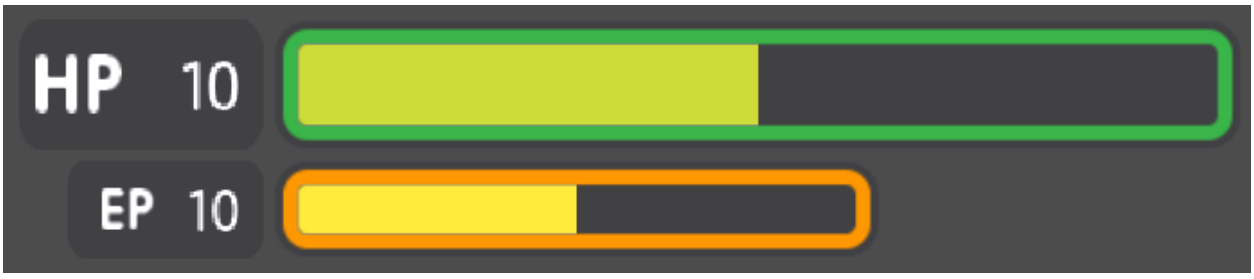


Fig. 57: The 2 bars align perfectly

2.12.7 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*.

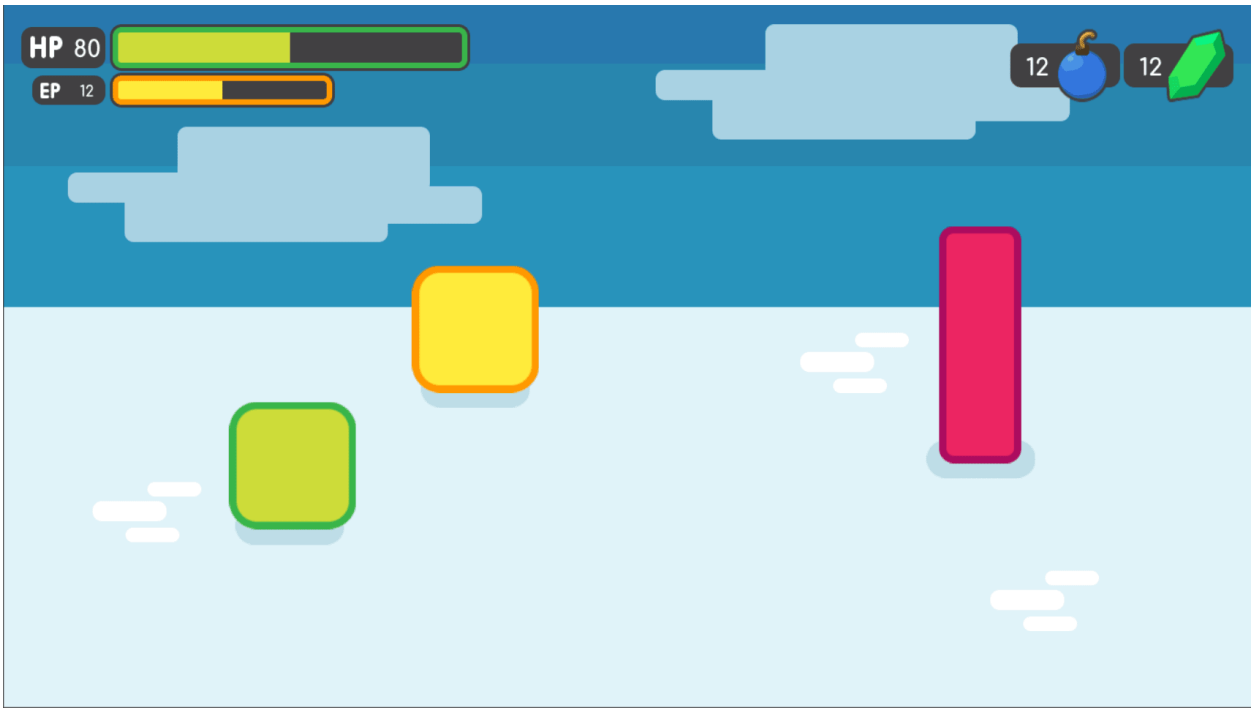


Fig. 58: The final result

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 or the other. In landscape orientation, the most common ratios range from 4:3 to 16:9. 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.

2.13 Control the game's UI with code

2.13.1 Intro

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

You will learn:

Fig. 59: Here's what you'll create: the bar and the counter animate when the character takes a hit. They fade when it dies.

- 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.

2.13.2 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:

GDScript

C#

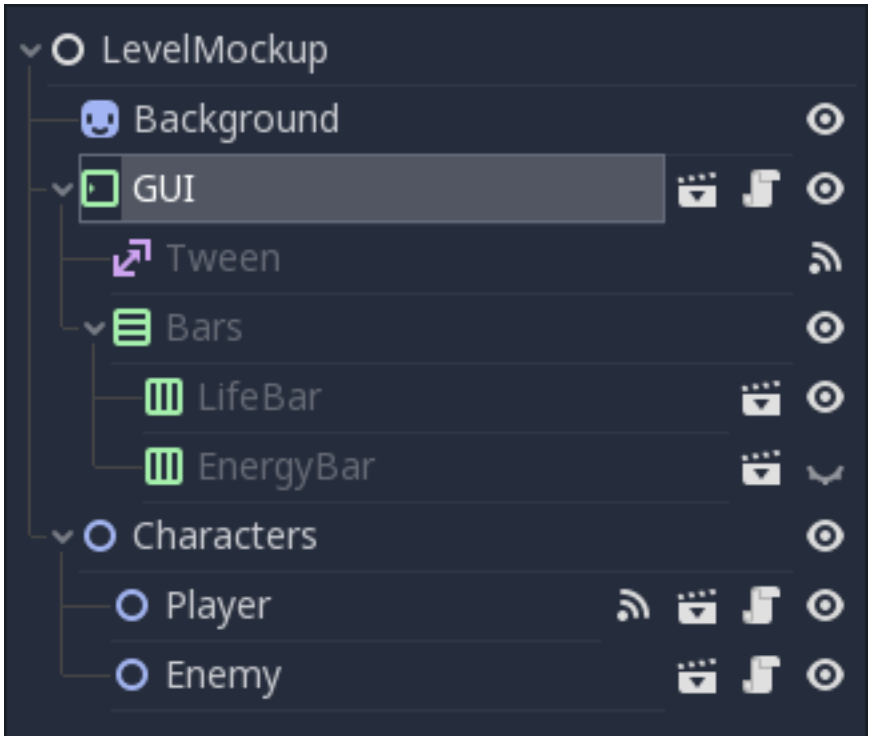


Fig. 60: The scene tree, with the GUI scene set to display its children

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

```
public class Gui : MarginContainer
{
    private Tween _tween;
    private Label _numberLabel;
    private TextureProgress _bar;

    public override void _Ready()
    {
        // C# doesn't have an onready feature, this works just the same
        _bar = (TextureProgress) GetNode("Bars/LifeBar/TextureProgress");
        _tween = (Tween) GetNode("Tween");
        _numberLabel = (Label) GetNode("Bars/LifeBar/Count/Background/Number");
    }
}
```

- `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.

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.

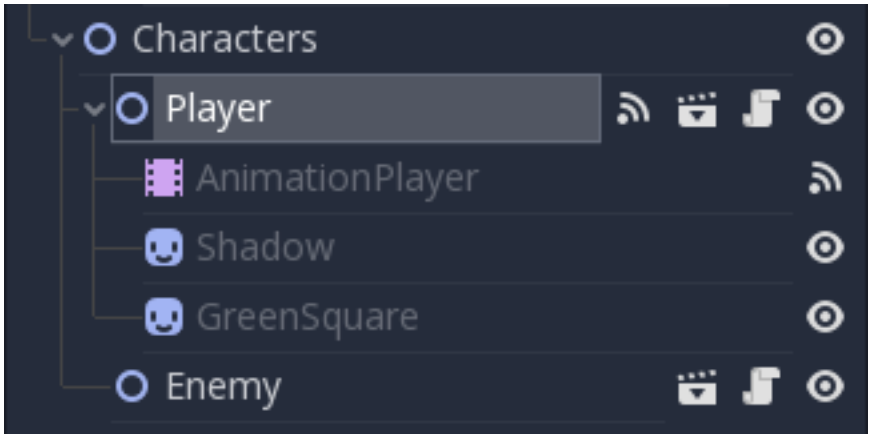


Fig. 61: The scene tree, with the Player scene set to display its children

2.13.3 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`:

GDScript

C#

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

```
public override void _Ready()
{
    // add this below _bar, _tween, and _numberLabel
    var player = (Player) GetNode("../Characters/Player");
    _bar.MaxValue = player.MaxHealth;
}
```

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)*

2.13.4 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 this way it gets complex quickly. Not only that: you need to listen to the state change constantly in the `_process` function. This 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 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 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
2. 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 it. You may replace the name with anything you'd like.

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:

GDScript

C#

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

```
public void OnPlayerHealthChanged(int playerHealth)
{
}
```

Note: The engine does not convert PascalCase to snake_case, for C# examples we'll be using PascalCase for method names & camelCase for method parameters which follows the official C# naming conventions.

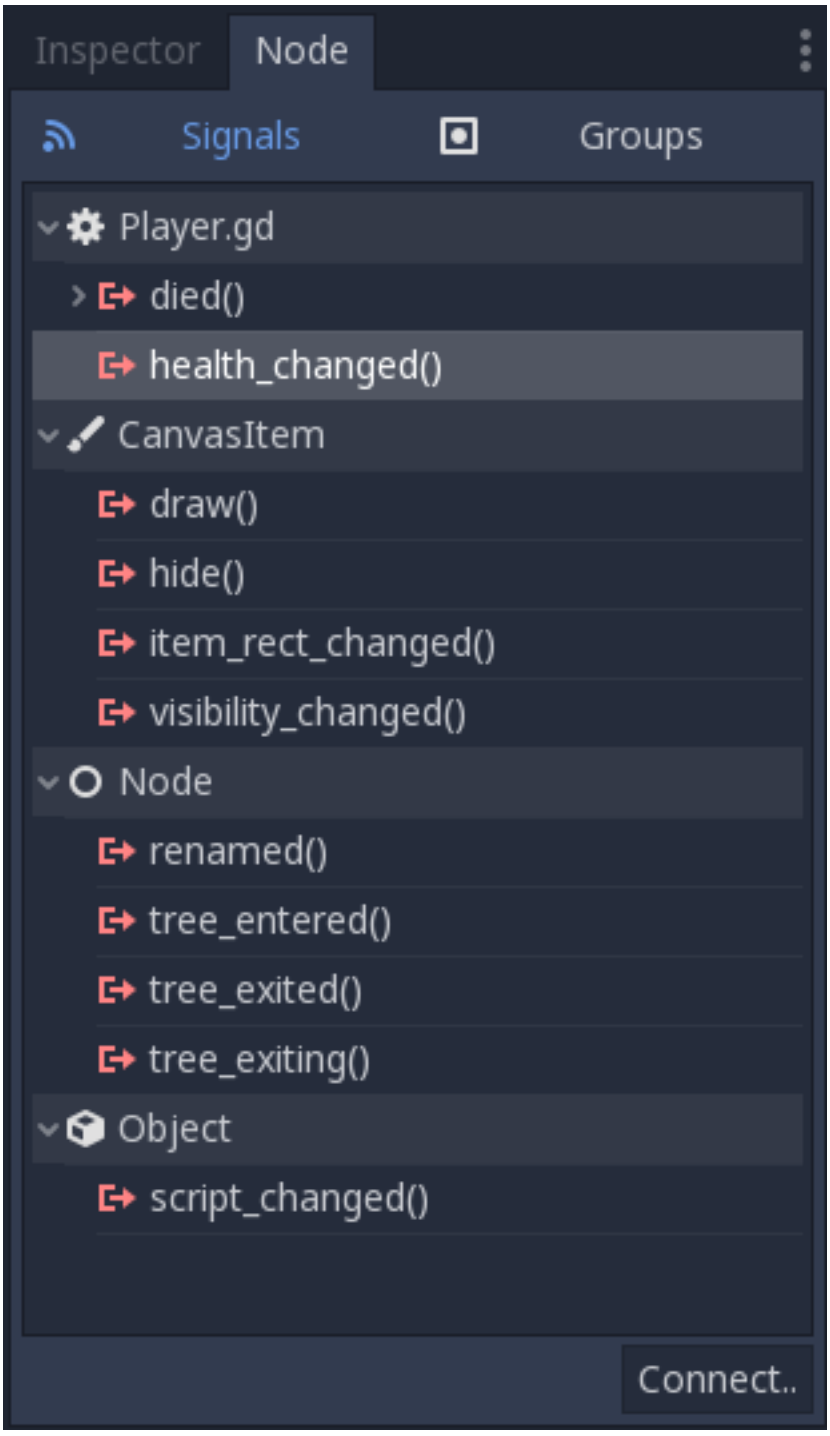


Fig. 62: We're connecting to the health_changed signal

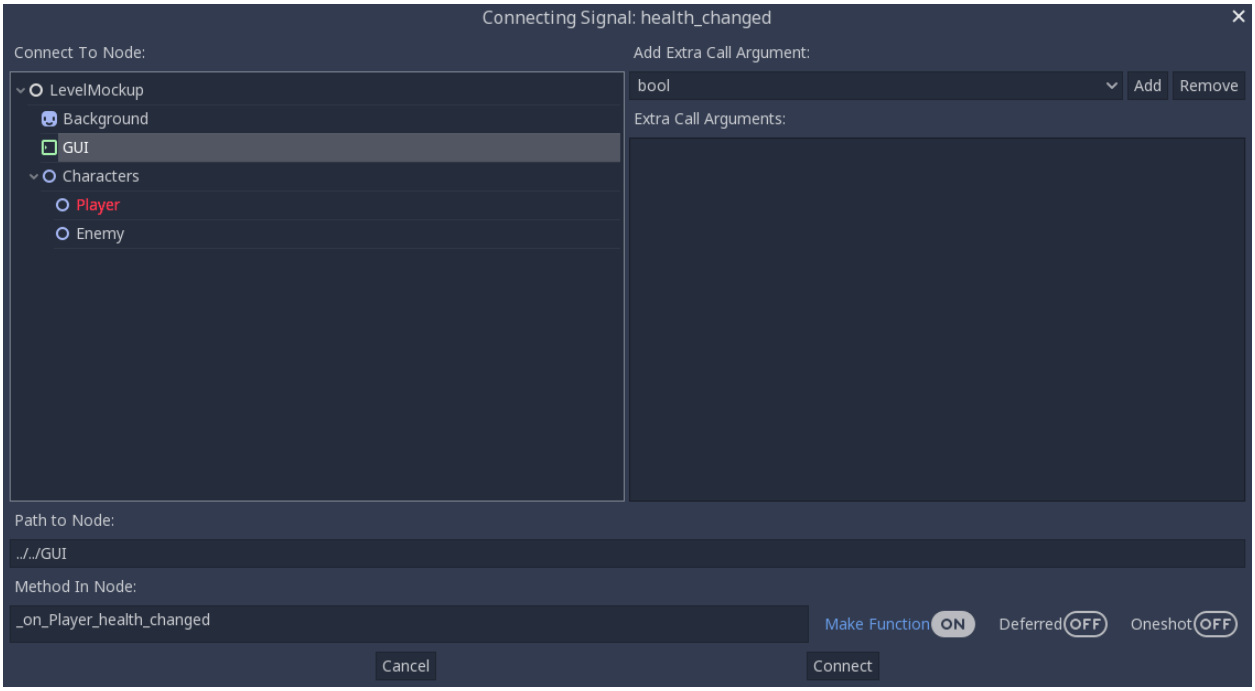


Fig. 63: The Connect Signal window with the GUI node selected

```
func _on_Player_health_changed():
    pass # replace with function body
```

Fig. 64: Godot writes the callback method for you and takes you to it

```
29      emit_signal("health_changed", health)
```

Fig. 65: In Player.gd, when the Player emits the health_changed 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 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:

GDScript

C#

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

```
public void UpdateHealth(int health)
{
}
```

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`

GDScript

C#

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

```
public void UpdateHealth(int health)
{
    _numberLabel.Text = health.ToString();
    _bar.Value = health;
}
```

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

2.13.5 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, name it `animated_health`, and set its value to 0. 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:

GDScript

C#

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

```
// add this to the top of your class
private int _animatedHealth = 0;

public void UpdateHealth(int health)
{
    _tween.InterpolateProperty(this, "_animatedHealth", _animatedHealth, health, 0.6f,
↪ Tween.TransitionType.Linear,
        Tween.EaseType.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 the `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:

GDScript

C#

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

```
if (!_tween.IsActive())
{
    _tween.Start();
}
```

Note: Although we could animate the *health* property on the *Player*, we 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*.

2.13.6 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:

GDScript

C#

```
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()
```

```
public void UpdateHealth(int health)
{
    _tween.InterpolateProperty(this, "_animatedHealth", _animatedHealth, health, 0.6f,
↪ Tween.TransitionType.Linear,
        Tween.EaseType.In);
```

(continues on next page)

(continued from previous page)

```
    if(!_tween.IsActive())
    {
        _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`:

GDScript

C#

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

```
public override void _Process(float delta)
{
    _numberLabel.Text = _animatedHealth.ToString();
    _bar.Value = _animatedHealth;
}
```

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. 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`:

GDScript

C#

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

```
public override void _Process(float delta)
{
    var roundValue = Mathf.Round(_animatedHealth);
    _numberLabel.Text = roundValue.ToString();
    _bar.Value = roundValue;
}
```

Try the game again to see a nice blocky animation.

Fig. 68: By rounding out animated_health we hit two birds with one stone

Tip: 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 a trick. It makes the GUI feel alive. If the `Player` takes 3 damage, it happens in an instant.

2.13.7 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 died. Press `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.

Fig. 69: The signal should already have the Enemy connected to it

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.

Fig. 70: You should get these values in the Connecting Signal window

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.

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.

GDScript

C#

```
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)
```

```
public void OnPlayerDied()
{
    var startColor = new Color(1.0f, 1.0f, 1.0f);
    var endColor = new Color(1.0f, 1.0f, 1.0f, 0.0f);
}
```

`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:

GDScript

C#

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

```
_tween.InterpolateProperty(this, "modulate", startColor, endColor, 1.0f, Tween.
↪TransitionType.Linear,
  Tween.EaseType.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:

GDScript

C#

```
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)
```

```
public void OnPlayerDied()
{
    var startColor = new Color(1.0f, 1.0f, 1.0f);
```

(continues on next page)

(continued from previous page)

```
    var endColor = new Color(1.0f, 1.0f, 1.0f, 0.0f);

    _tween.InterpolateProperty(this, "modulate", startColor, endColor, 1.0f, Tween.
↪TransitionType.Linear,
        Tween.EaseType.In);
}
```

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

Fig. 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.

2.14 Splash screen

2.14.1 Tutorial

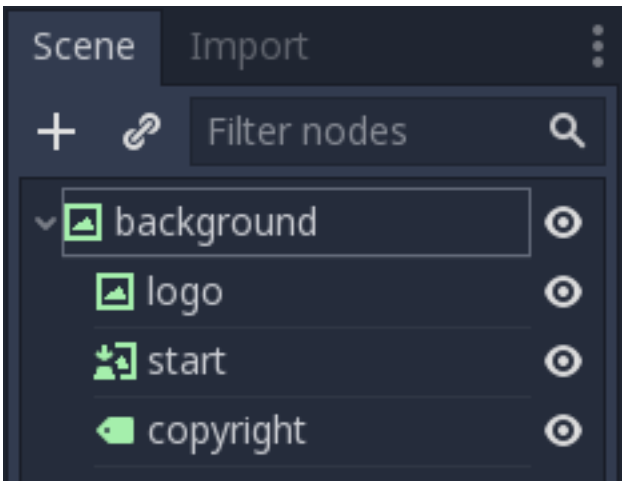
This is a simple tutorial to establish the basic idea of how the GUI subsystem works. The goal is to create a simple, static splash screen.

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.

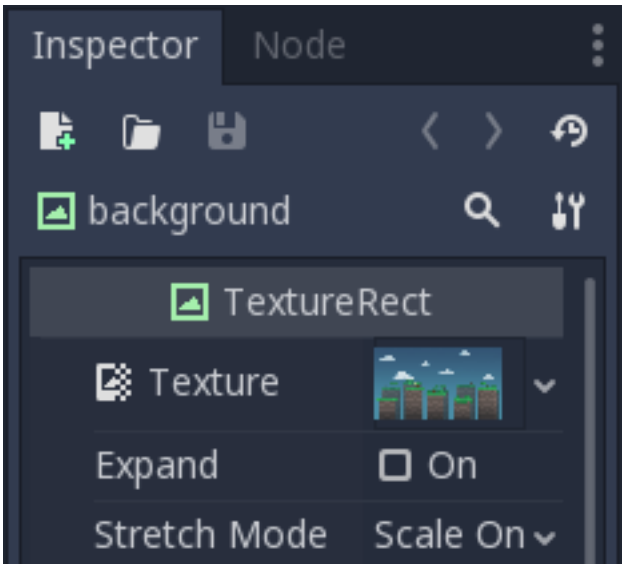
`robisplash_assets.zip`.

2.14.2 Setting up

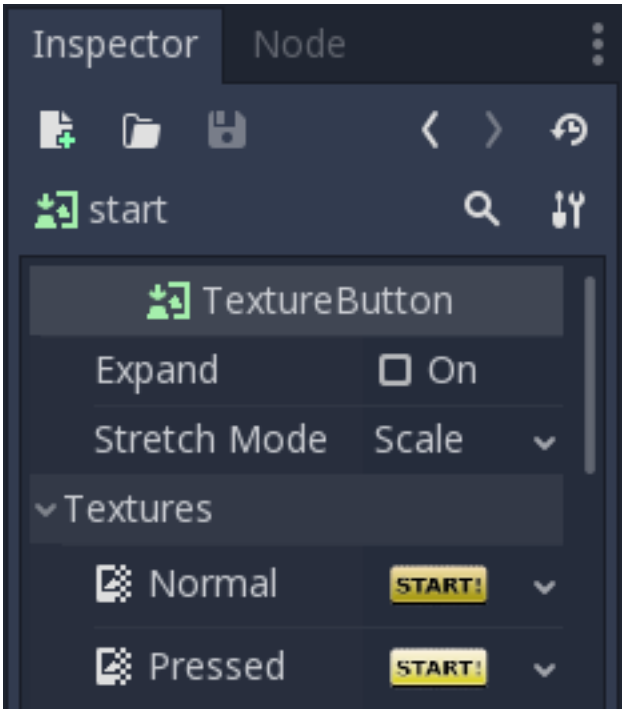
Set the display resolution to 800x450 in Project Settings, and set up a new scene like this:



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

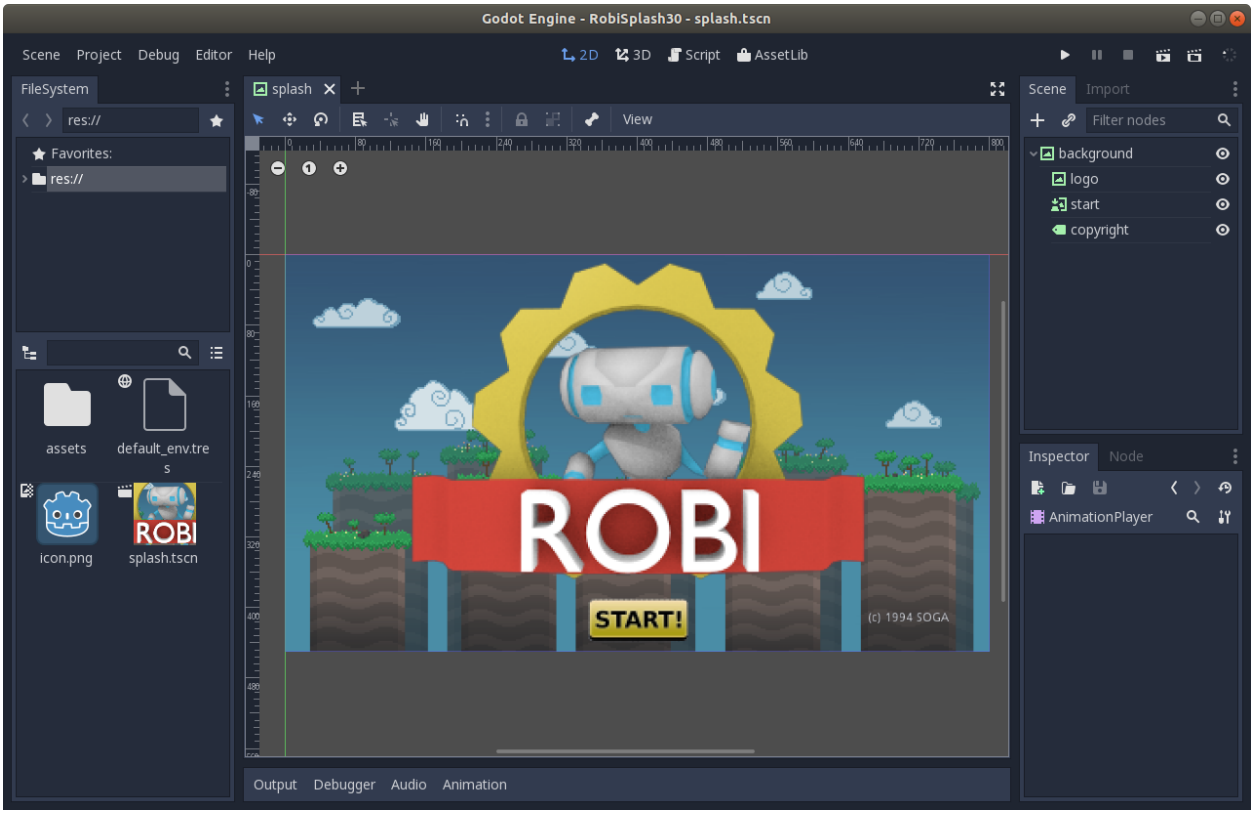


The node "start" is a *TextureButton*. It takes several images for different states, but only the normal and pressed will be supplied in this example:



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.

2.15 Animations

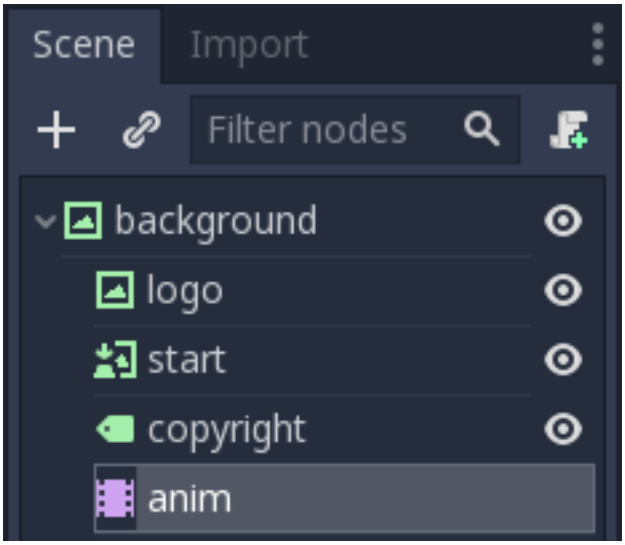
2.15.1 Introduction

Godot's animation system is extremely powerful and flexible.

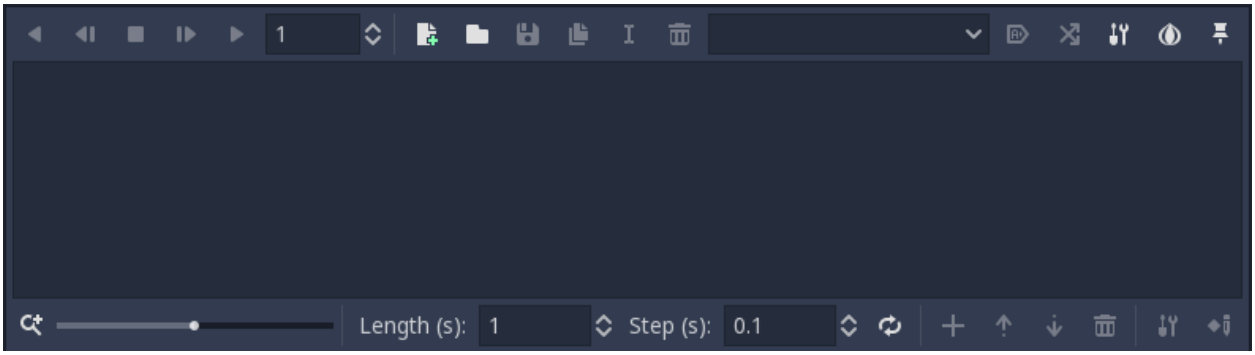
To begin, let's use the scene from the previous tutorial (*Splash screen*). The goal is to add a "fade-in" animation to the splash image. Here's a copy just in case: `robisplash.zip`.

2.15.2 Add an animation player

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



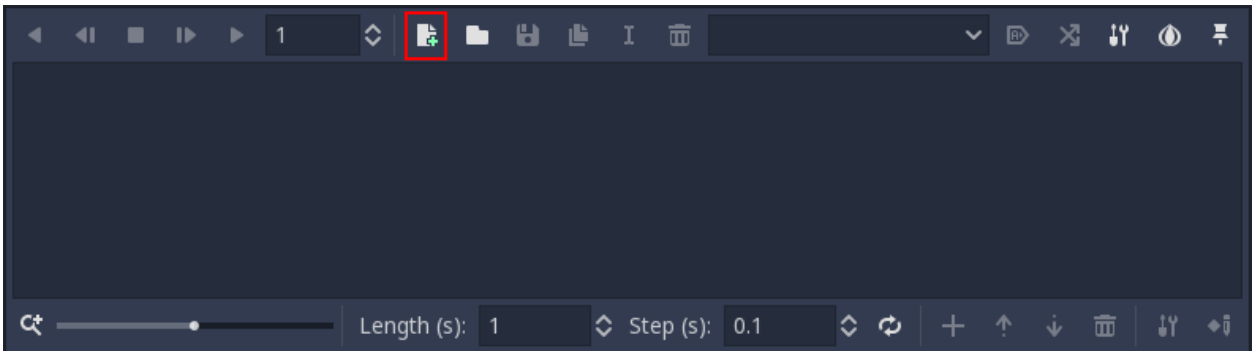
When a node of this type is selected, the animation editor panel will appear:



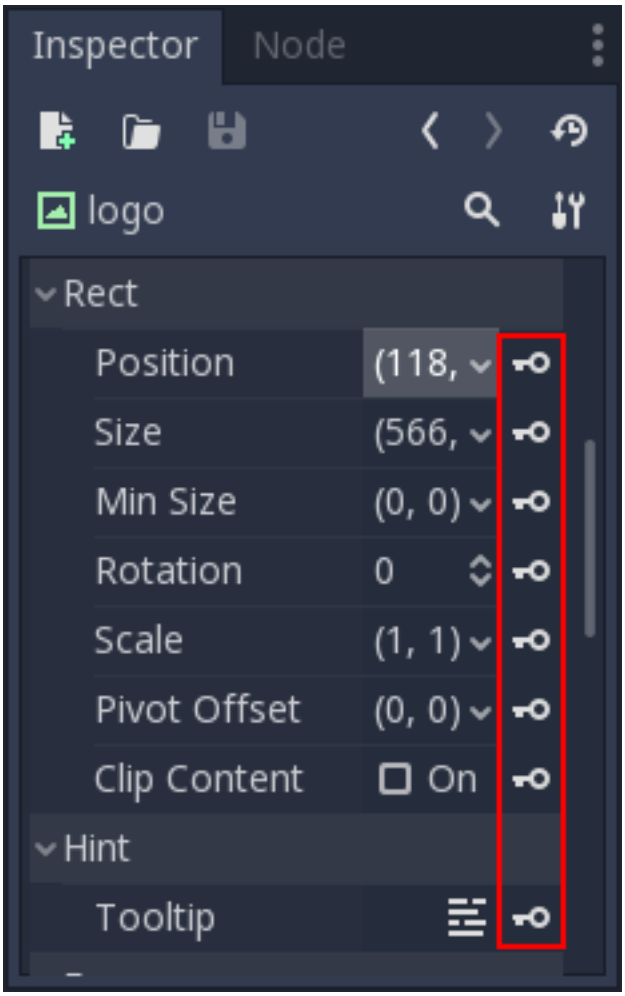
The animation editor panel stays visible until manually hidden.

2.15.3 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.



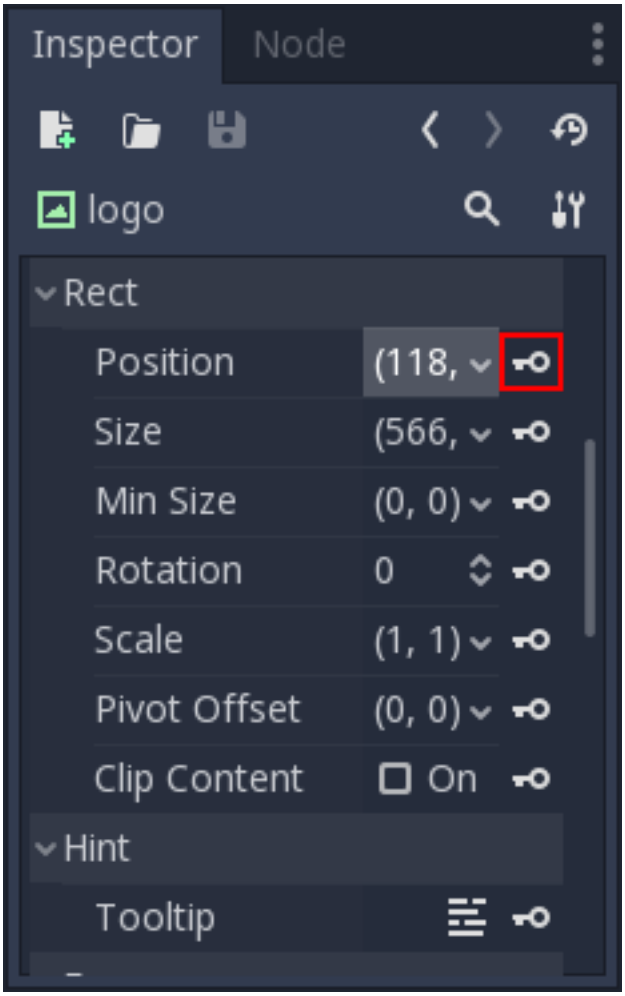
Now that we have an animation the property editor enters "animation editing" mode. In this mode, a key icon appears next to every property of the property editor. In Godot any property of an object can be animated:



2.15.4 Editing the animation

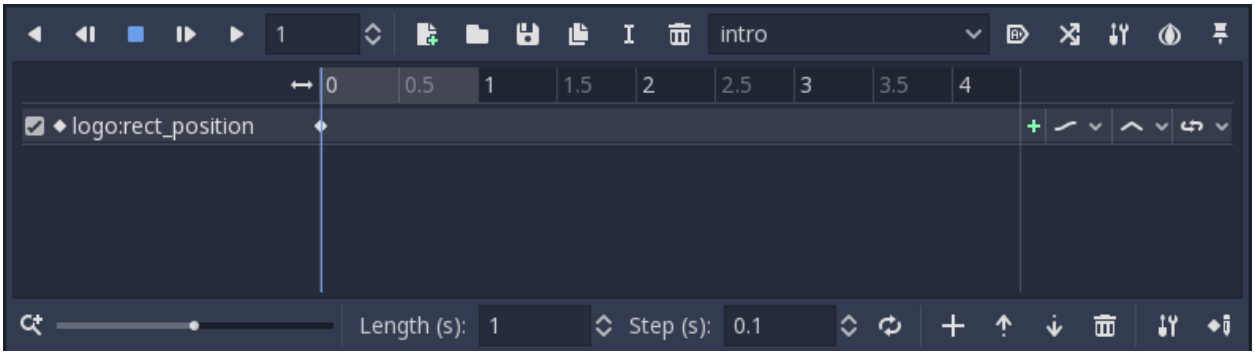
The logo will appear from the top of the screen.

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:

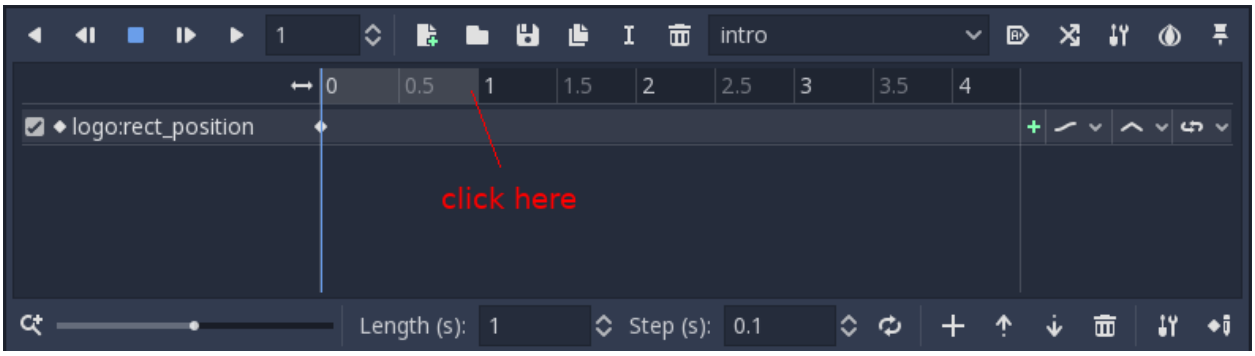


When the dialog appears, confirm that you are creating a new track.

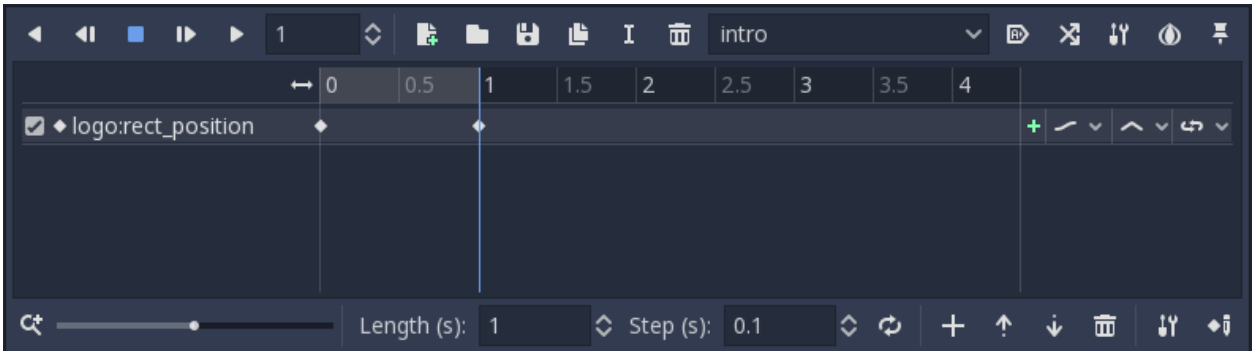
The keyframe will be added in the animation player editor:



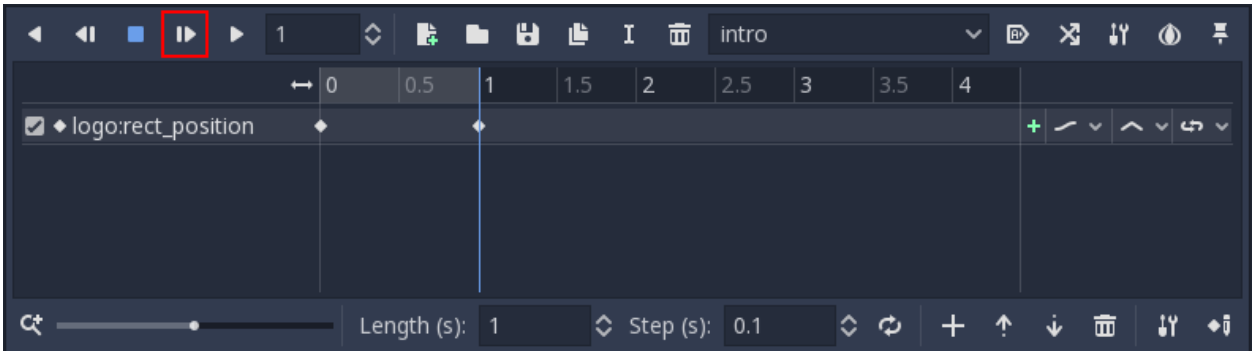
Move the editor cursor to the end, by clicking here:



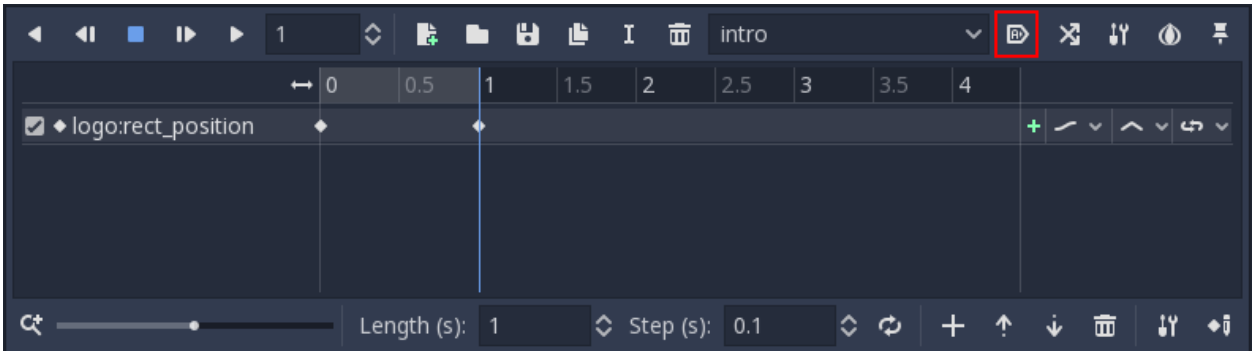
Change the logo position to `(118, 0)` and add a keyframe again. With two keyframes, the animation happens.



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:

2.16 Resources

2.16.1 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,

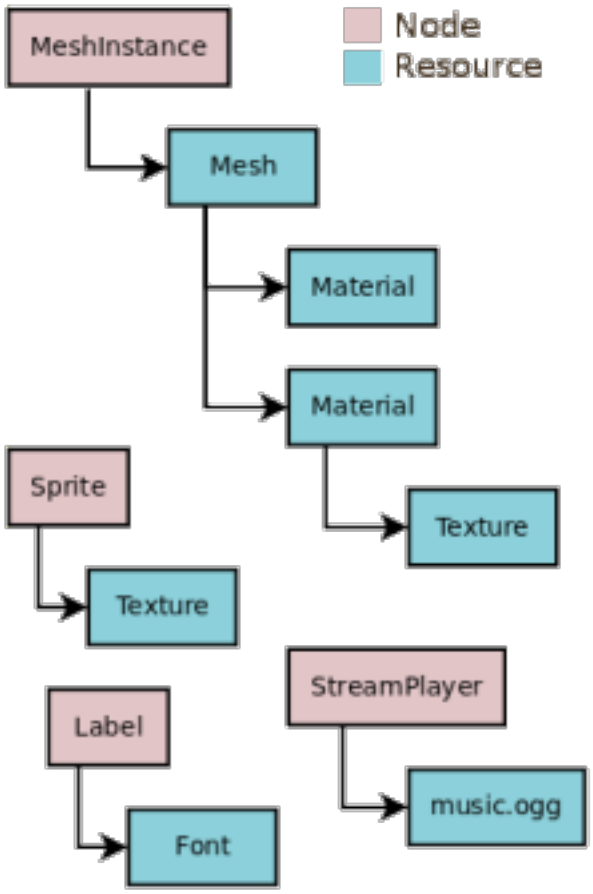
Resources are mere **data containers**. This means that they don't do any action nor process any information. Resources just contain data.

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 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.

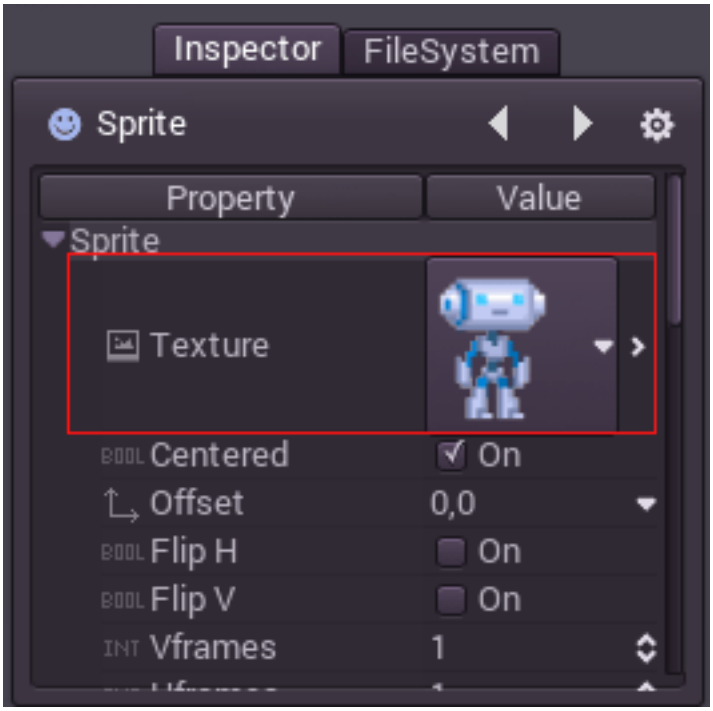
Typically, every object in Godot (Node, Resource, or anything else) can export properties, properties can be of many types (like a string, integer, Vector2, etc) and one of those types can be a resource. This means that both nodes and resources can contain resources as properties. To make it a little more visual:



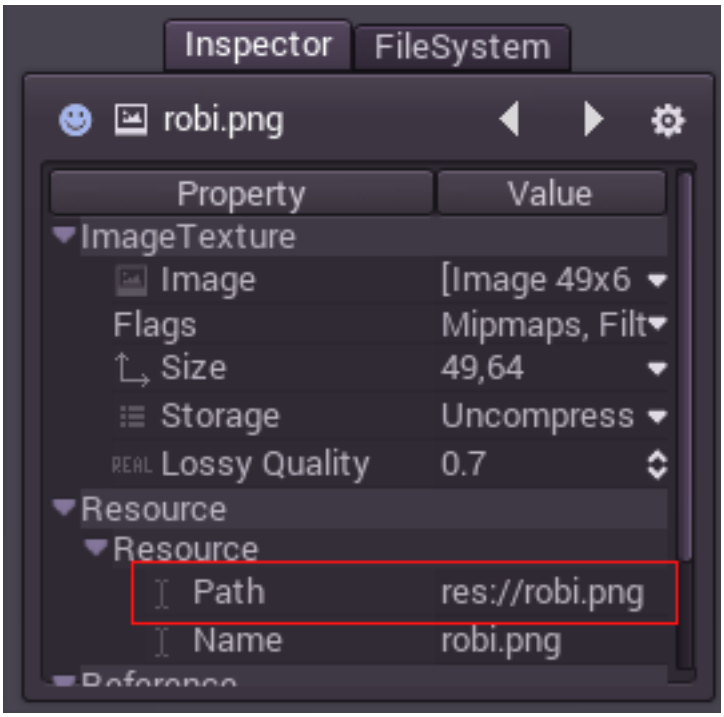
2.16.2 External vs built-in

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.

2.16.3 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:

GDScript

C#

```
func _ready():
        var res = load("res://robi.png") # resource is loaded when line is executed
        get_node("sprite").texture = res
```

```
public override void _Ready()
{
    var texture = (Texture)GD.Load("res://robi.png"); // resource is loaded when line␣
↪is executed
    var sprite = (Sprite)GetNode("sprite");
    sprite.Texture = texture;
}
```

The second way is more optimal, but only works with a string constant parameter, because it loads the resource at compile-time.

GDScript

C#

```
func _ready():
        var res = preload("res://robi.png") # resource is loaded at compile time
        get_node("sprite").texture = res
```

```
// preload() is unavailable in C Sharp
```

2.16.4 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.

GDScript

C#

```
func _on_shoot():
        var bullet = preload("res://bullet.tscn").instance()
        add_child(bullet)
```

```
private PackedScene _bulletScene = (PackedScene)GD.Load("res://bullet.tscn");

public void OnShoot()
{
    Node bullet = _bulletScene.Instance();
    AddChild(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.

2.16.5 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.

2.16.6 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.

2.17 File system

2.17.1 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.

2.17.2 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
```

2.17.3 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.

2.17.4 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`.

2.17.5 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.

2.17.6 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.

2.17.7 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 useful when writing development tools in Godot!

2.17.8 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 fine on their platorm, 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.

2.18 SceneTree

2.18.1 Introduction

This is where things start getting abstract, but don't panic. There's not 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.

2.18.2 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 low level and when making games in Godot, writing your own MainLoop seldom makes sense.

2.18.3 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()*.

2.18.4 Root viewport

The root *Viewport* is always at the top of the scene. From a node, it can be obtained in two different ways:

GDScript

C#

```
get_tree().get_root() # access via scenemainloop
get_node("/root") # access via absolute path
```

```
GetTree().GetRoot(); // access via scenemainloop
GetNode("/root"); // access via absolute path
```

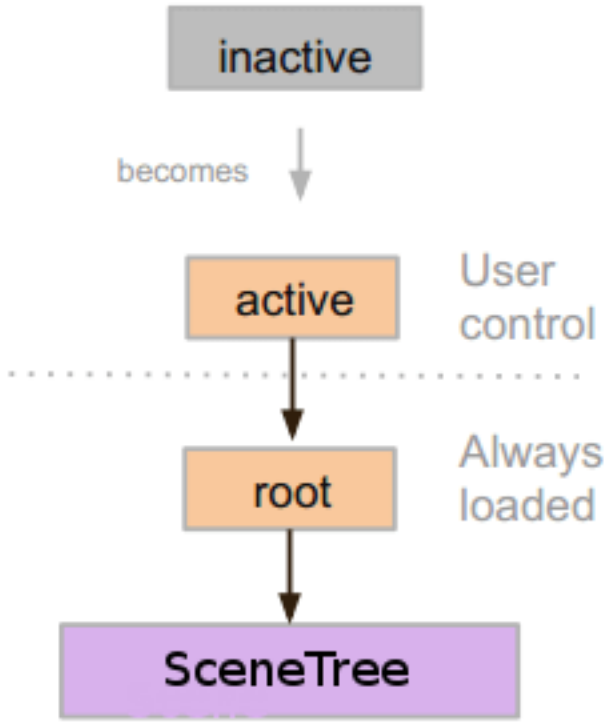
This node contains the main viewport, anything that is a child of a *Viewport* is drawn inside of it by default, so it makes sense that the top of all nodes is always a node of this type, otherwise nothing would be seen!

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.

2.18.5 Scene tree

When a node is connected, directly or indirectly, to the root viewport, it becomes part of the *scene tree*.

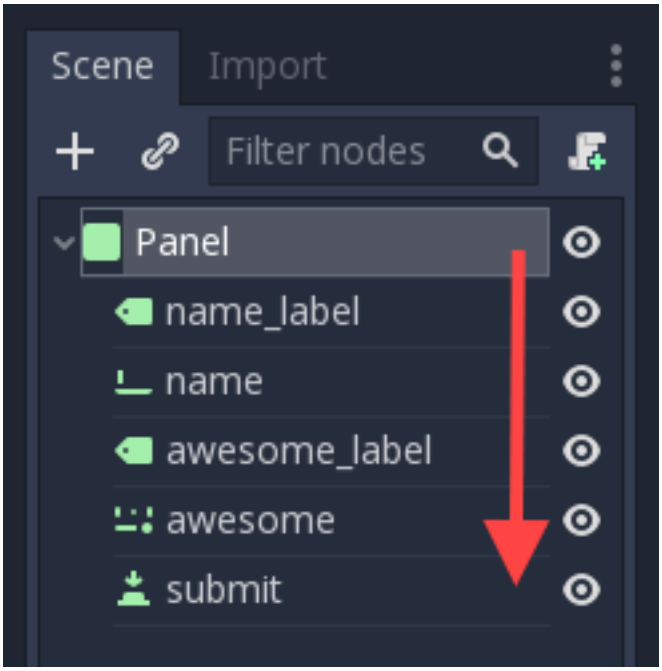
This means that, as explained in previous tutorials, it will get the _enter_tree() and _ready() callbacks (as well as _exit_tree()).



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.

2.18.6 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.



2.18.7 "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

2.18.8 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:

GDScript

C#

```
func _my_level_was_completed():
    get_tree().change_scene("res://levels/level2.tscn")
```

```
public void _MyLevelWasCompleted()
{
    GetTree().ChangeScene("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*.

2.19 Singletons (AutoLoad)

2.19.1 Introduction

Scene singletons are useful, catering to a common use case where you need to store persistent information between scenes.

Albeit 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:

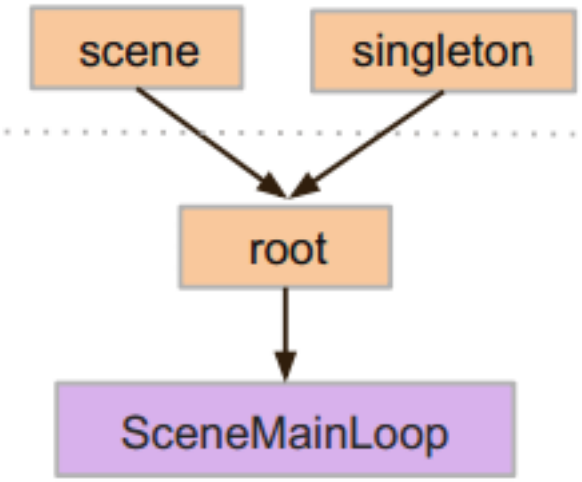
- Are always loaded, no matter which scene is opened from the editor
- Can store global variables, such as player information, items, money etc. and share information between scenes
- Can handle switching scenes and transitions
- Acts like a singleton, since GDScript does not support global variables by design.

Auto-loading nodes and scripts caters to this need.

2.19.2 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:

GDScript

C#

```
var player_vars = get_node("/root/playervariables")
player_vars.health
```

```
var playerVariables = (PlayerVariables)GetNode("/root/PlayerVariables");
playerVariables.Health -= 10; // instance field
```

Or even simpler using the name directly:

GDScript

C#

```
playervariables.health
```

```
// Static members can be accessed by using the class name
PlayerVariables.Health -= 10;
```

2.19.3 Custom scene switcher

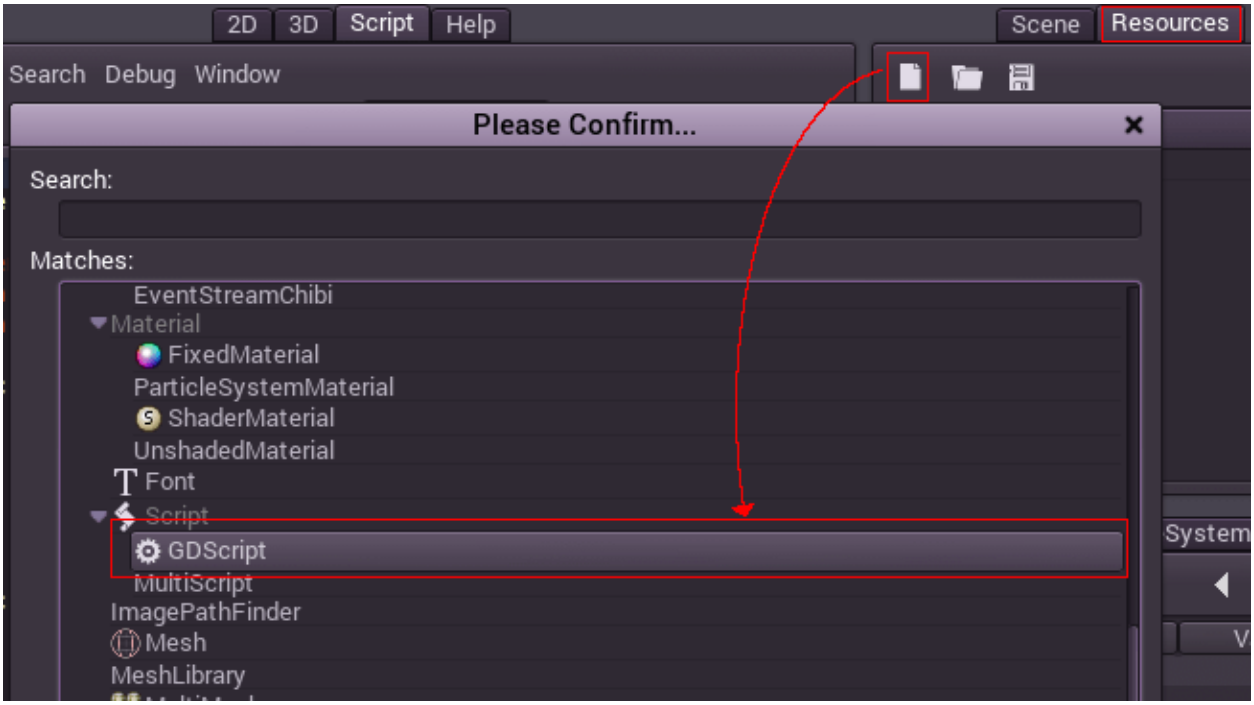
This short tutorial will explain how to make a scene switcher using autoload. For simple scene switching, the *SceneTree.change_scene()* method suffices (described in *SceneTree*), so this method is for more complex behavior when switching between scenes.

First download the template from here: `autoload.zip`, then open it.

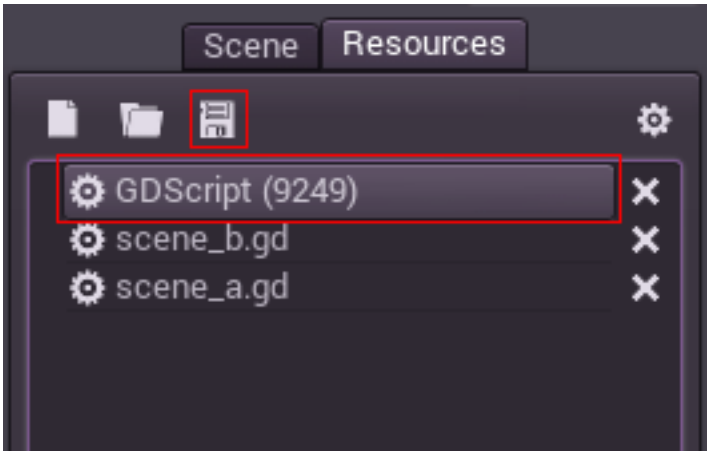
Two scenes are present, scene_a.tscn and scene_b.tscn on an otherwise empty project. Each are identical and contain a button connected to a callback for switching to the other scene. When the project runs, it starts in scene_a.tscn. However, this currently does nothing and pressing the button does not work.

2.19.4 global.gd

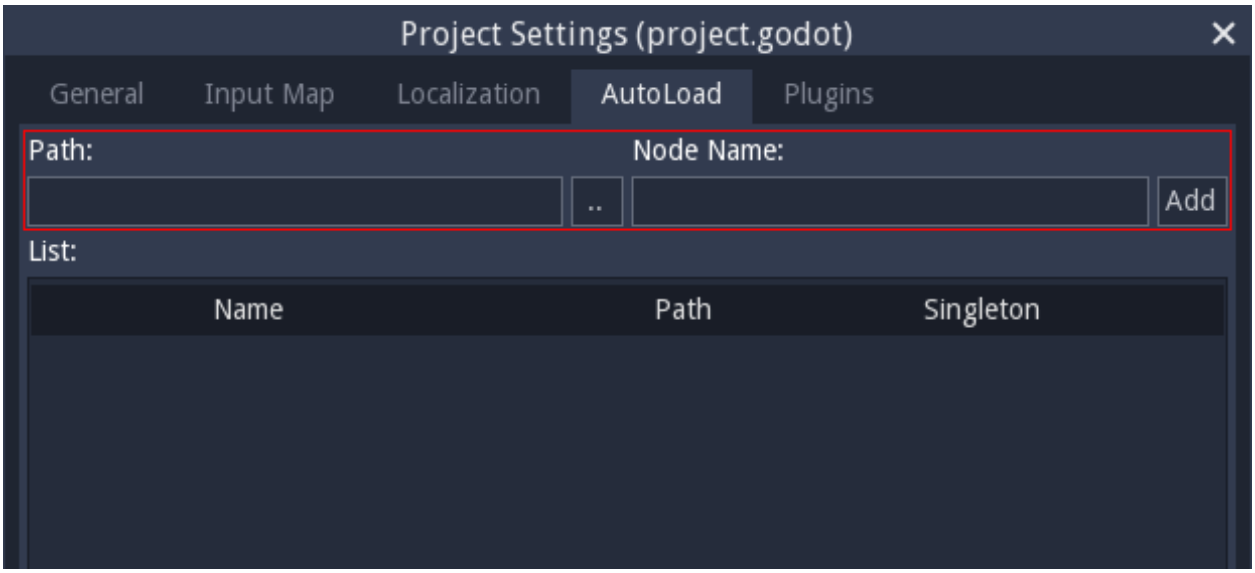
First of all, create a global.gd script. The easy way to create a resource from scratch is from the new resource button in the inspector tab:



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!

GDScript

C#

```
extends Node

var current_scene = null

func _ready():
        var root = get_tree().get_root()
        current_scene = root.get_child( root.get_child_count() -1 )
```

```
using Godot;
using System;

public class Global : Godot.Node
{
    public Node CurrentScene { get; set; }

    public override void _Ready()
    {
        Viewport root = GetTree().GetRoot();
        CurrentScene = root.GetChild(root.GetChildCount() - 1);
    }
}
```

Next up is the function for changing the scene. This function frees the current scene and replaces it with the requested one.

GDScript

C#

```
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 )
```

```
public void GotoScene(string 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:

    CallDeferred(nameof(DeferredGotoScene), path);
}

public void DeferredGotoScene(string path)
{
    // Immediately free the current scene, there is no risk here.
    CurrentScene.Free();

    // Load a new scene
    var nextScene = (PackedScene)GD.Load(path);

    // Instance the new scene
    CurrentScene = nextScene.Instance();
```

(continues on next page)

(continued from previous page)

```
    // Add it to the active scene, as child of root
    GetTree().GetRoot().AddChild(CurrentScene);

    // optional, to make it compatible with the SceneTree.change_scene() API
    GetTree().SetCurrentScene(CurrentScene);
}
```

As mentioned in the comments above, we 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:

GDScript

C#

```
#add to scene_a.gd

func _on_goto_scene_pressed():
        get_node("/root/global").goto_scene("res://scene_b.tscn")
```

```
// add to SceneA.cs

public void OnGotoScenePressed()
{
    var global = (Global)GetNode("/root/Global");
    global.GotoScene("res://scene_b.tscn");
}
```

and

GDScript

C#

```
#add to scene_b.gd

func _on_goto_scene_pressed():
        get_node("/root/global").goto_scene("res://scene_a.tscn")
```

```
// add to SceneB.cs

public void OnGotoScenePressed()
{
    var global = (Global)GetNode("/root/Global");
    global.GotoScene("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*

CHAPTER 3

Editor manual

3.1 From Unity3D to Godot Engine

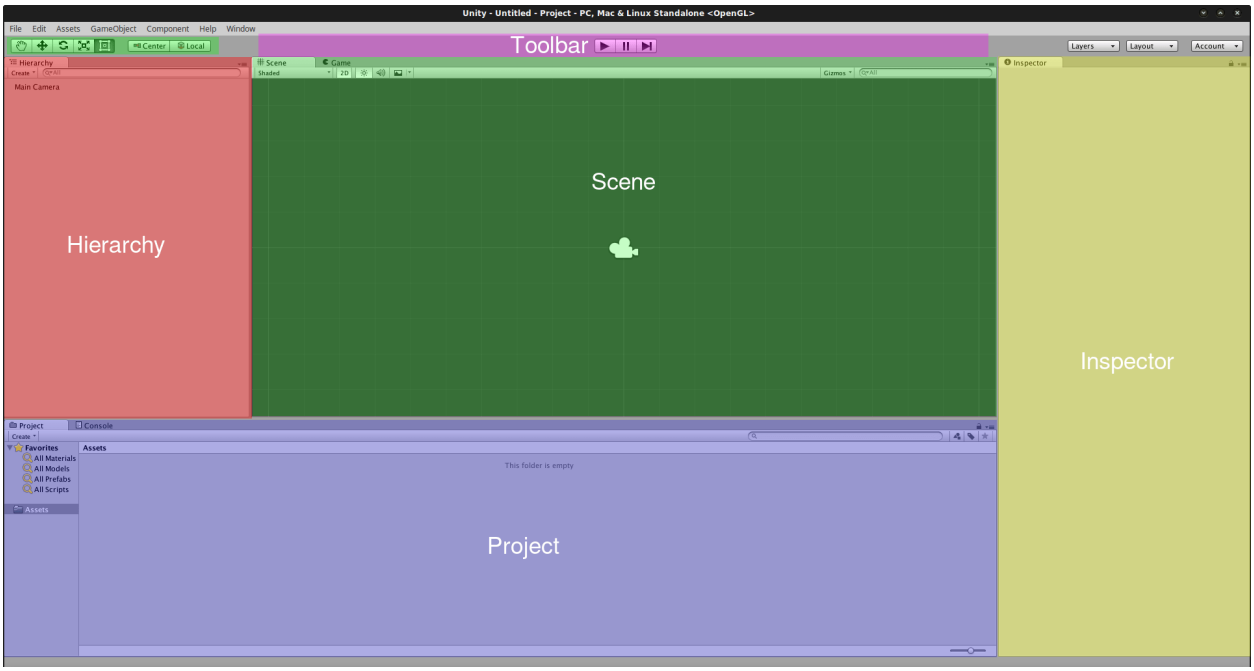
This guide provides an overview of Godot Engine from the viewpoint of a Unity user, and aims to help you migrate your existing Unity experience into the world of Godot.

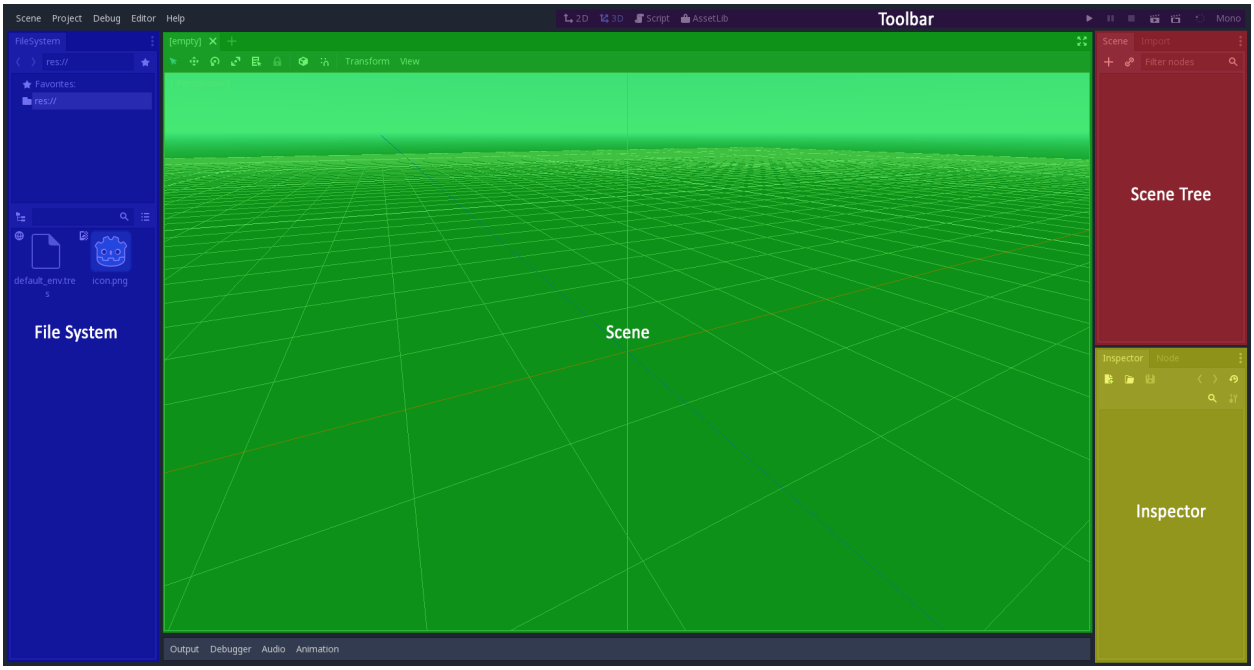
3.1.1 Differences

	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, macOS, X11 (Linux, *BSD)
OS (export)	• **Desktop:** Windows, macOS, Linux • **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, macOS, X11 • **Mobile:** Android, iOS • **Web:** WebAssembly • **Console:** See *Console Support in Godot* • **VR:** Oculus Rift, SteamVR
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 VS Code	• *External editors are possible* • *Android SDK for Android export*
Killer features	• Huge community • Large assets store	• Scene System • *Animation Pipeline* • *Easy to write Shaders* • Debug on Device

3.1.2 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 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.

3.1.3 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).

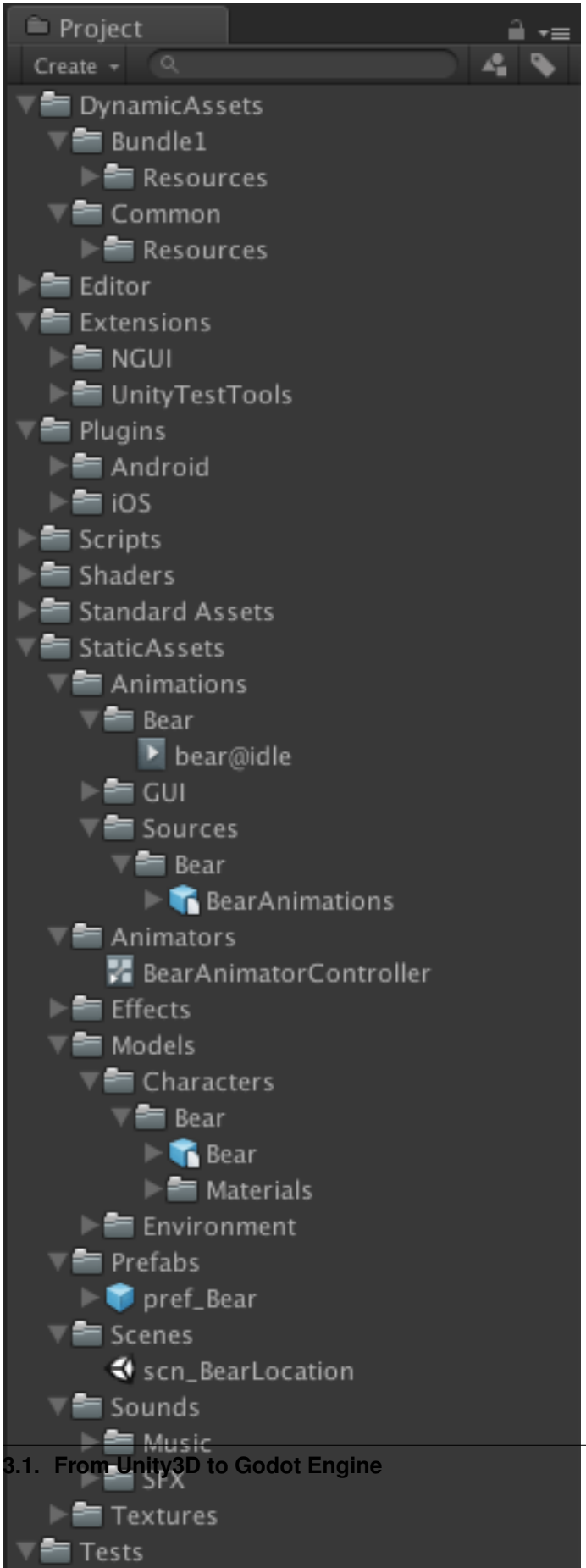
Finally, the main scene would then be composed of one root node with 2 children: a Player instance node, and a Level instance node. The root node can be anything, generally a "root" type such as "Node" which is the most global type, or "Node2D" (root type of all 2D-related nodes), "Spatial" (root type 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".

3.1.4 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.

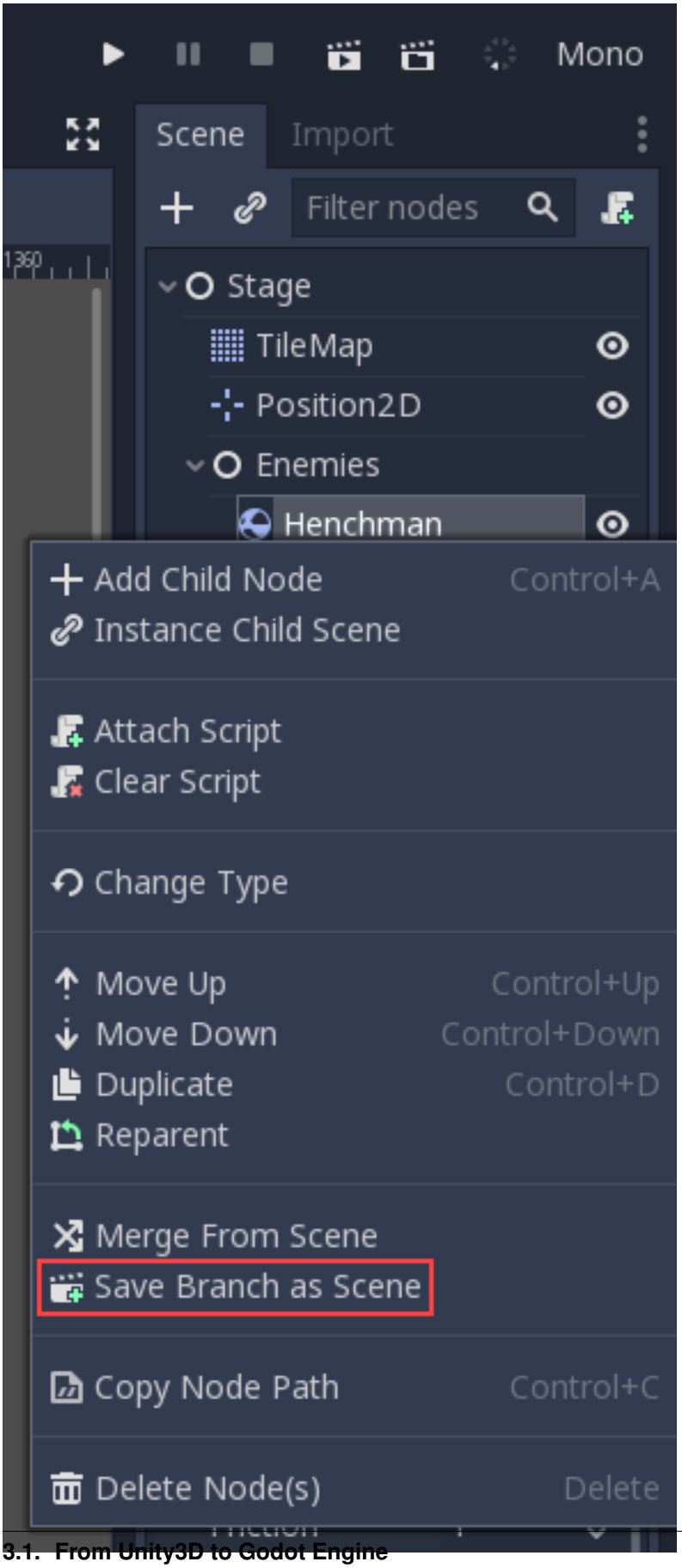
However, we often observe a common architecture for Unity projects, which consists in having one Assets folder in the root directory, that contains various folders, one per type of asset: Audio, Graphics, Models, Materials, Scripts, Scenes, etc.

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*.

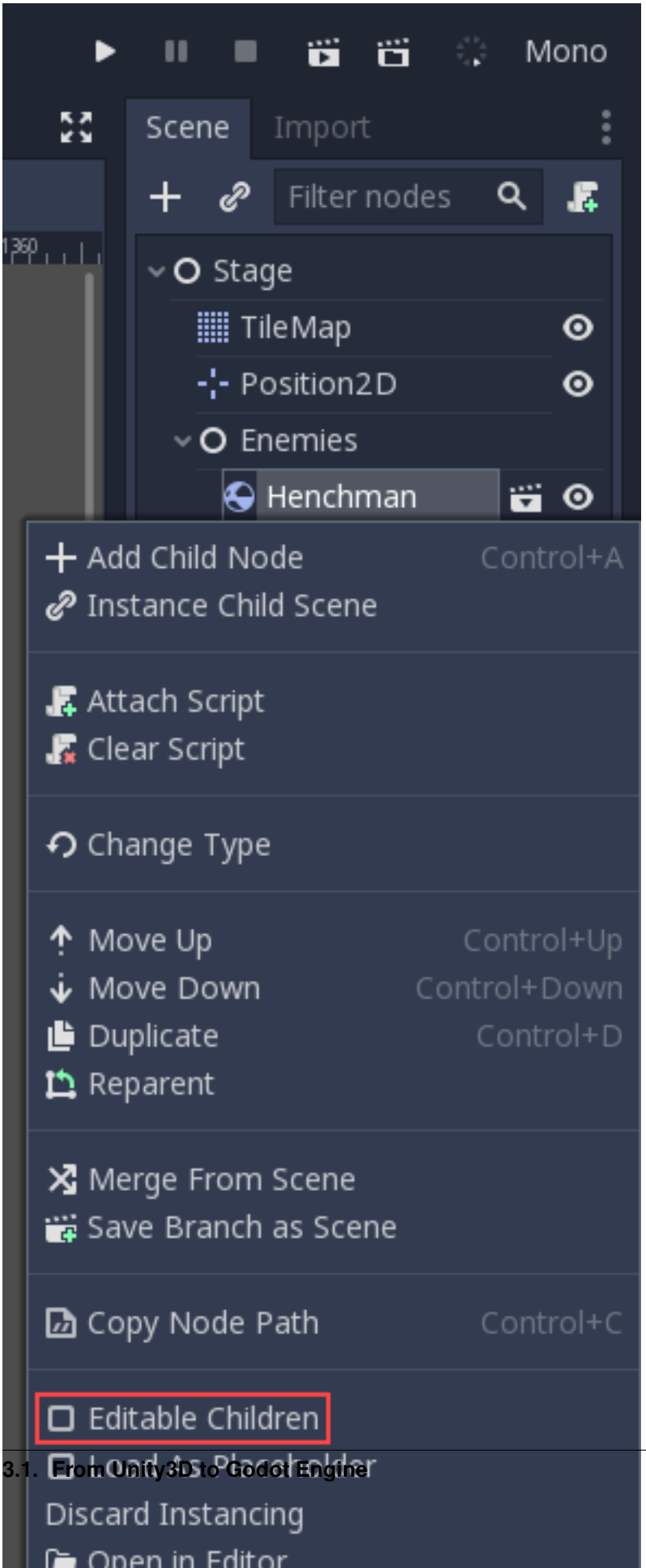
3.1.5 Where are my prefabs?

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.


Mono
Scene
Import
Filter nodes
Stage
TileMap
Position2D
Enemies
Henchman
Add Child Node
Control+A
Instance Child Scene
Attach Script
Clear Script
Change Type
Move Up
Control+Up
Move Down
Control+Down
Duplicate
Control+D
Reparent
Merge From Scene
Save Branch as Scene
Copy Node Path
Control+C
Delete Node(s)
Delete

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.


Mono
Scene
Import
Filter nodes
Stage
TileMap
Position2D
Enemies
Henchman
Add Child Node
Control+A
Instance Child Scene
Attach Script
Clear Script
Change Type
Move Up
Control+Up
Move Down
Control+Down
Duplicate
Control+D
Reparent
Merge From Scene
Save Branch as Scene
Copy Node Path
Control+C
Editable Children
Discard Instancing
Open in Editor

3.1.6 Glossary correspondence

GameObject -> Node Add a component -> Inheriting Prefab -> Externalized branch

3.1.7 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 *C#*. 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 easy to learn.

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, your 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.

3.1.8 Using Godot in C++

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.

3.2 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.

3.2.1 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`.

3.2.2 Setting the project path

Depending on where your Godot binary is located and what your current working directory is, you may need to set the path to your project for any of the following commands to work correctly.

This can be done by giving the path to the `project.godot` file of your project as either the first argument, like this:

```
user@host:~$ godot path_to_your_project/project.godot [other] [commands] [and] [args]
```

Or by using the `--path` argument:

```
user@host:~$ godot --path path_to_your_project [other] [commands] [and] [args]
```

For example, the full command for exporting your game (as explained below) might look like this:

```
user@host:~$ godot --path path_to_your_project --export my_export_preset_name game.exe
```

3.2.3 Creating a project

To create a project from the command line, navigate the to the desired place and create an empty project.godot file.

```
user@host:~$ mkdir newgame
user@host:~$ cd newgame
user@host:~/newgame$ touch project.godot
```

The project can now be opened with Godot.

3.2.4 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
```

3.2.5 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
```

3.2.6 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
```

3.2.7 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
```

3.2.8 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, 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.

3.2.9 Running a script

It is possible to run a simple .gd script from the command line. This feature is especially useful in 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()
```

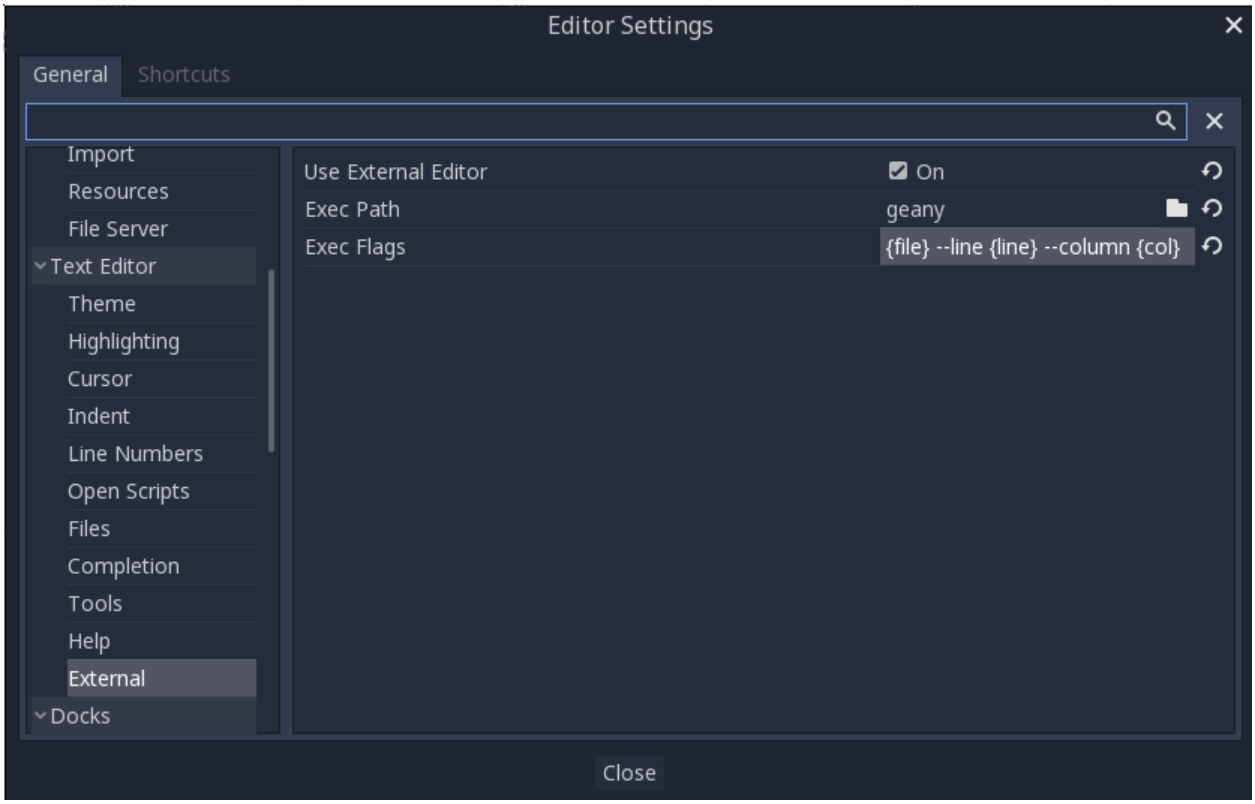
And how to run it:

```
user@host:~/newgame$ godot -s sayhello.gd
Hello!
```

If no project.godot exists at the path, current path is assumed to be the current working directory (unless `-path` is specified).

3.3 Using an external text editor

While godot has an inbuilt text editor, some developers have a tendency to want to use a text editor they are familiar with. Godot provides this option via the options under `Editor -> Editor Settings -> Text Editor -> External`



There are two fields: the executable path and command line flags. The flags allow you to better integrate the editor with godot. Godot will replace the following inside the flags parameter:

Field in Exec Flags	Is replaced with
{project}	The absolute path to the project directory
{file}	The absolute path to the file
{col}	The column number of the error
{line}	The line number of the error

Some example Exec Flags for various editors include:

Editor	Exec Flags
geany/kate	{file} –line {line} –column {col}
atom/sublime text	{file}:{line}

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 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 THE_NAME = "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
```

(continues on next page)

(continued from previous page)

```
# Functions override functions with the same name on the base/parent class.
# If you still want to call them, use '.' (like 'super' in other languages).

func something(p1, p2):
    .something(p1, p2)

# 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. Operators (like `in`, `not`, `and` or `or`) and names of built-in types as listed in the following sections are also reserved.

Keywords are defined in the GDScript tokenizer in case you want to take a look under the hood.

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.

Continued on next page

Table 1 – continued from previous page

Keyword	Description
break	Exits the execution of the current `for` or `while` loop.
continue	Immediately skips to the next iteration of the `for` or `while` 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.
self	Refers to current class instance.
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.
breakpoint	Editor helper for debugger breakpoints.
preload	Preloads a class or variable. See *Classes as resources*.
yield	Coroutine support. See *Coroutines with yield*.
assert	Asserts a condition, logs error on failure. Ignored in non-debug builds. See *Assert keyword*.
remote	Networking RPC annotation. See *high-level multiplayer docs*.
master	Networking RPC annotation. See *high-level multiplayer docs*.
slave	Networking RPC annotation. See *high-level multiplayer docs*.
sync	Networking RPC annotation. See *high-level multiplayer docs*.
PI	PI constant.
TAU	TAU constant.
INF	Infinity constant. Used for comparisons.
NAN	NAN (not a number) constant. Used for comparisons.

Operators

The following is the list of supported operators and their precedence.

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"""`	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.
```

Multi-line comments can be created using """ (three quotes in a row) at the beginning and end of a block of text. Note that this creates a string, therefore, it will not be stripped away when the script is compiled.

```
""" Everything on these
lines is considered
a comment. """
```

Built-in types

Built-in types are stack-allocated. They are passed as values. This means a copy is created on each assignment or when passing them as arguments to functions. The only exceptions are `Array`s and `Dictionaries`, which are passed by reference so they are shared. (Not `PoolArray`s like `PoolByteArray` though, those are passed as values too, so consider this when deciding which to use!)

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. 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 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",
    "some_key": 2,
    "other_key": [2, 3, 4],
    "more_key": "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",
    some_key = 2,
    other_key = [2, 3, 4],
    more_key = "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
```

(continues on next page)

(continued from previous page)

```
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 my_function(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

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.
my_node.call("my_function", args)

# Store a function reference.
var my_func = funcref(my_node, "my_function")
# Call stored function reference.
my_func.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 = [value] if [expression] else [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])

for i in range(3):
    statement # Similar to [0, 1, 2] but does not allocate an array.
```

(continues on next page)

(continued from previous page)

```
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 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 = preload("enemy.gd")

# [...]

# Use 'is' to check inheritance.
if (entity is Enemy):
    entity.apply_damage()
```

To call a function in a *base class* (i.e. one `extend`-ed in your current class), prepend `.` to the function name:

```
.basefunc(args)
```

This is especially useful because functions in extending classes replace functions with the same name in their base classes. So if you still want to call them, you can use `.` like the `super` keyword in other languages:

```
func some_func(x):
    .some_func(x) # Calls same function on the parent class.
```

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.
```

(continues on next page)

(continued from previous page)

```
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 my_class = load("myclass.gd")

# Preload the class only once at compile time.
const MyClass = preload("myclass.gd")

func _init():
    var a = MyClass.new()
    a.some_function()
```

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.
```

(continues on next page)

(continued from previous page)

```
# 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

# Named Enum Values

# Editor will enumerate as THING_1, THING_2, ANOTHER_THING.
enum NamedEnum {THING_1, THING_2, ANOTHER_THING = -1}
export (NamedEnum) var x

# 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
```

(continues on next page)

(continued from previous page)

```
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 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]

# Exported arrays can specify type (using the same hints as before).

export(Array, int) var ints = [1,2,3]
export(Array, int, "Red", "Green", "Blue") var enums = [2, 1, 0]
export(Array, Array, float) var two_dimensional = [[1, 2], [3, 4]]

# You can omit the default value, but then it would be null if not assigned.

export(Array) var b
export(Array, PackedScene) var scenes

# Typed arrays also work, only initialized empty:
```

(continues on next page)

(continued from previous page)

```
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 c = [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 my_var_set, my_var_get

func my_var_set(new_value):
    my_var = new_value

func my_var_get():
    return my_var # Getter must return a value.
```

Either of the *setter* or *getter* functions can be omitted:

```
# Only a setter.
var my_var = 5 setget myvar_set
# Only a getter (note the comma).
var my_var = 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.
    my_integer = 5
    print(my_integer)

    # Does trigger setter/getter.
    self.my_integer = 5
    print(self.my_integer)
```

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 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.

Alternatively, when not using references, the `is_instance_valid(instance)` can be used to check if an object has been freed.

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 can be connected in the editor or from code like regular signals. 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 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)
    some_instance.emit_signal("some_signal")
```

Coroutines with yield

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 my_func():
   print("Hello")
   yield()
   print("world")

func _ready():
    var y = my_func()
    # 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 my_func():
   print("Hello")
   print(yield())
   return "cheers!"

func _ready():
    var y = my_func()
    # 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")
```

Coroutines themselves use the `completed` signal when they transition into an invalid state, for example:

```
func my_func():
        yield(button_func(), "completed")
        print("All buttons were pressed, hurray!")

func button_func():
    yield($Button0, "pressed")
        yield($Button1, "pressed")
```

`my_func` will only continue execution once both the buttons are pressed.

Onready keyword

When using nodes, it's 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 my_label

func _ready():
    my_label = 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 my_label = get_node("MyLabel")
```

Assert keyword

The `assert` keyword can be used to check conditions in debug builds. These assertions are ignored in non-debug builds.

```
# Check that 'i' is 0.
assert(i == 0)
```

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 simple and easy 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 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 latter 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() {
```

(continues on next page)

(continued from previous page)

```
    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);
```

(continues on next page)

(continued from previous page)

```
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 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 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
```

(continues on next page)

(continued from previous page)

```
const SHIP_HIT = 1
const WATER_HIT = 2

var board = {}

func initialize():
    board[Vector2(1, 1)] = SHIP
    board[Vector2(1, 2)] = SHIP
    board[Vector2(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 and indexing, which makes it 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 = "Caroline" # This would work too to create a new key
```

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++)
{
```

(continues on next page)

(continued from previous page)

```
    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):
    pass

for i in range(5, 10):
    pass

for i in range(5, 10, 2):
    pass
```

And backwards looping is done through a negative counter:

```
for (int i = 10; i > 0; i--) {}
```

Becomes:

```
for i in range(10, 0, -1):
    pass
```

While

while() loops are the same everywhere:

```
var i = 0

while i < strings.size():
    print(strings[i])
    i += 1
```

Custom iterators

You can create custom iterators in case the default ones don't quite meet your needs by overriding the Variant class's `_iter_init`, `_iter_next`, and `_iter_get` functions in your script. An example implementation of a forward iterator follows:

```
class FwdIterator:
    var start, curr, end, increment

    func _init(start, stop, inc):
        self.start = start
        self.curr = start
        self.end = stop
        self.increment = inc

    func is_done():
        return (curr < end)

    func do_step():
        curr += increment
        return is_done()

    func _iter_init(arg):
        curr = start
        return is_done()

    func _iter_next(arg):
        return do_step()

    func _iter_get(arg):
        return curr
```

And it can be used like any other iterator:

```
var itr = FwdIterator.new(0, 6, 2)
for i in itr:
    print(i) # Will print 0, 2, and 4
```

Make sure to reset the state of the iterator in `_iter_init`, otherwise nested for-loops that use custom iterators will not work as expected.

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 is also another way to format text in GDScript, namely the `String.format()` method. It replaces all occurrences of a key in the string with the corresponding value. The method can handle arrays or dictionaries for the key/value pairs.

Arrays can be used as key, index, or mixed style (see below examples). Order only matters when the index or mixed style of Array is used.

A quick example in GDScript:

```
# Define a format string
var format_string = "We're waiting for {str}"

# Using the 'format' method, replace the 'str' placeholder
var actual_string = format_string.format({"str": "Godot"})

print(actual_string)
# Output: "We're waiting for Godot"
```

There are other *format specifiers*, but they are only applicable when using the `%` operator.

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.

`s`	**Simple** conversion to String by the same method as implicit String conversion.
`c`	A single **Unicode character**. Expects an unsigned 8-bit integer (0-255) for a code point or a single-character string.
`d`	A **decimal integral** number. Expects an integral or real number (will be floored).
`o`	An **octal integral** number. Expects an integral or real number (will be floored).
`x`	A **hexadecimal integral** number with **lower-case** letters. Expects an integral or real number (will be floored).
`X`	A **hexadecimal integral** number with **upper-case** letters. Expects an integral or real number (will be floored).
`f`	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.

`+`	In number specifiers, **show + sign** if positive.
Integer	Set **padding**. Padded with spaces or with zeroes if integer starts with `0` in an integer placeholder. When used after `.`, see `.`.
`.`	Before `f`, set **precision** to 0 decimal places. Can be followed up with numbers to change. Padded with zeroes.
`-`	**Pad to the right** rather than the left.
`*`	**Dynamic padding**, expect additional integral parameter to set padding or precision after `.`, see *dynamic padding*.

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%"
```

Format method examples

The following are some examples of how to use the various invocations of the `String.format` method.

Type	Style	Example	Result
Dictionary	key	“Hi, {name} v{version}!”.format({“name”:”Godette”, “version”:”3.0”})	Hi, Godette v3.0!
Dictionary	index	“Hi, {0} v{1}!”.format({“0”:”Godette”, “1”:”3.0”})	Hi, Godette v3.0!
Dictionary	mix	“Hi, {0} v{version}!”.format({“0”:”Godette”, “version”:”3.0”})	Hi, Godette v3.0!
Array	key	“Hi, {name} v{version}!”.format([[“version”:”3.0”], [“name”:”Godette”]])	Hi, Godette v3.0!
Array	index	“Hi, {0} v{1}!”.format([“Godette”,”3.0”])	Hi, Godette v3.0!
Array	mix	“Hi, {name} v{0}!”.format([3.0, [“name”:”Godette”]])	Hi, Godette v3.0!

Placeholders can also be customized when using `String.format`, here’s some examples of that functionality.

Type	Example	Result
Infix (default)	“Hi, {0} v{1}”.format([“Godette”, “3.0”], “{_}”)	Hi, Godette v3.0
Postfix	“Hi, 0% v1%”.format([“Godette”, “3.0”], “_%”)	Hi, Godette v3.0
Prefix	“Hi, %0 v%1”.format([“Godette”, “3.0”], “%_”)	Hi, Godette v3.0

Combining both the `String.format` method and the `%` operator could be useful as `String.format` does not have a way to manipulate the representation of numbers.

Example	Result
“Hi, {0} v{version}”.format({0:”Godette”, “version”:”%0.2f” % 3.114})	Hi, Godette v3.11

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:

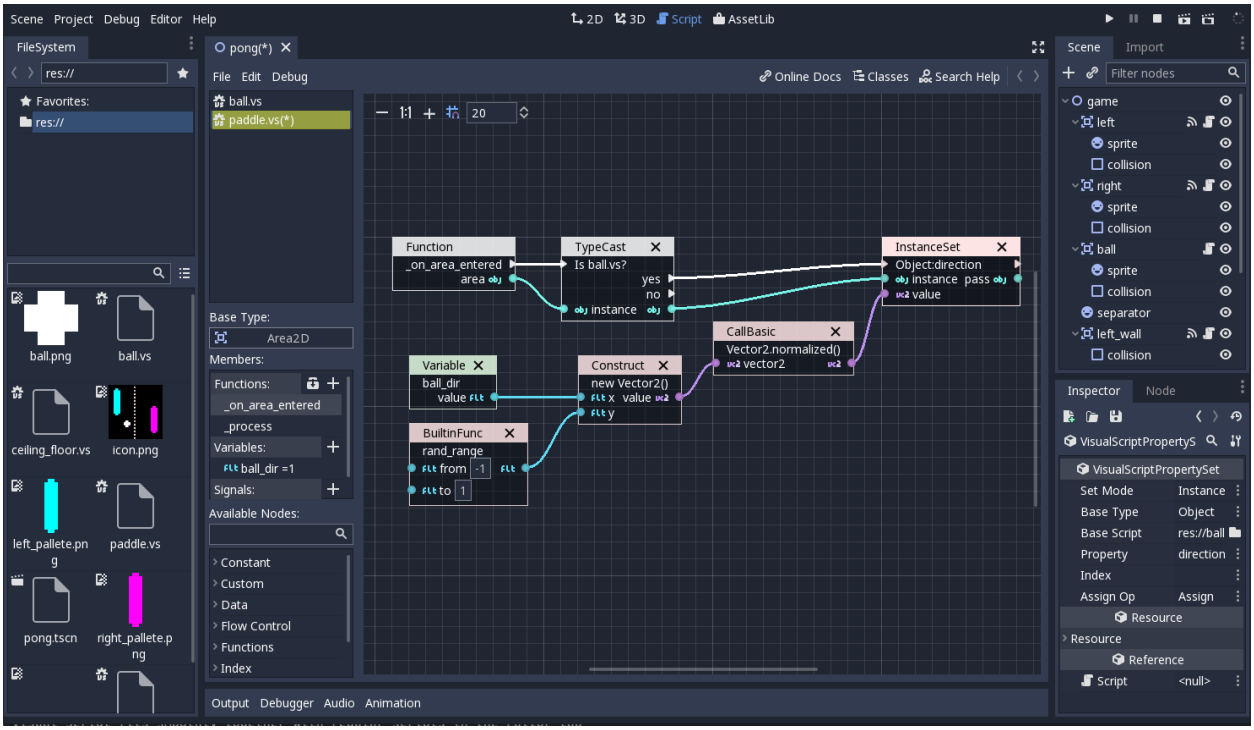
- Game development beginners who want to learn an engine but have no programming experience yet.
- Artists and Game Designers who have no experience in programming and want to create quick prototypes or simple games.
- Programmers working in a team that want to make part of the game logic available to 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.

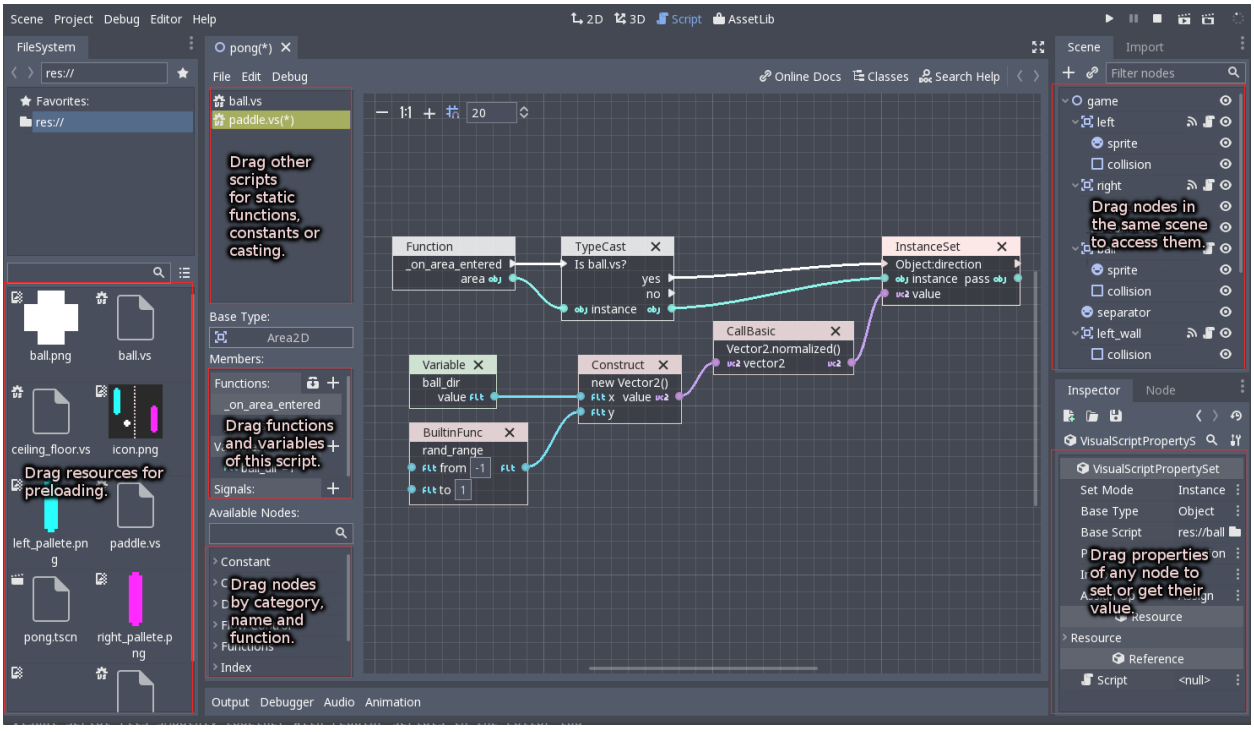
4.2.2 Getting started with Visual Scripting

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

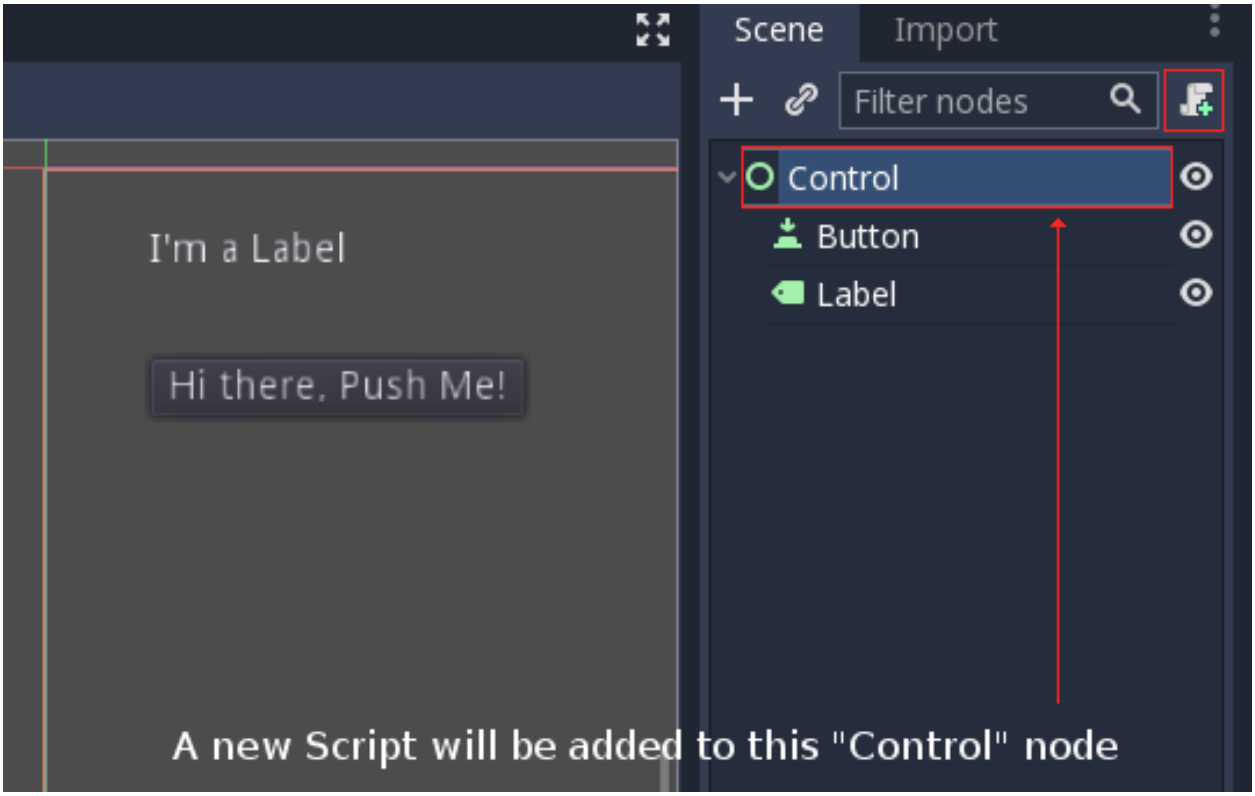


In fact, Visual Scripting integrates so well to Godot that it's hard to believe it was added only in version 3.0. This is because, when editing, the rest of Godot panels and docks act like a palette from where you can drag and drop all sorts of information to the script canvas:

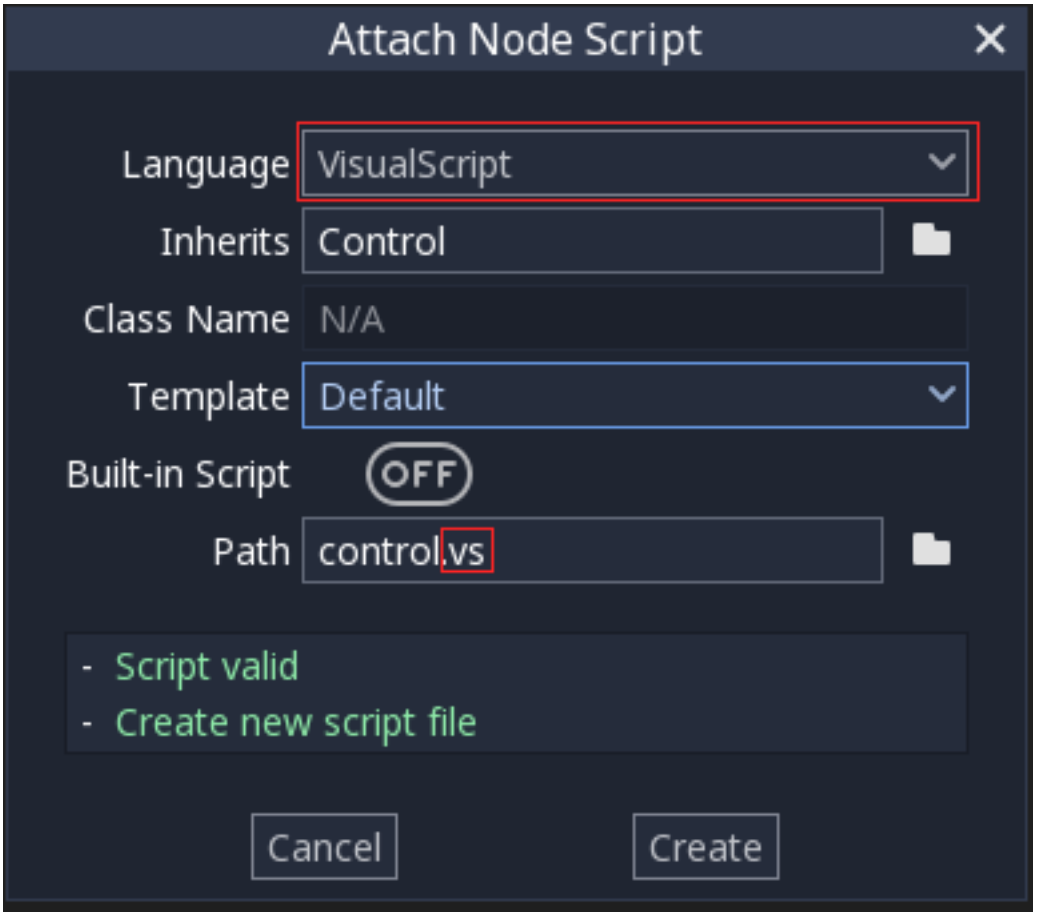


Creating a Script

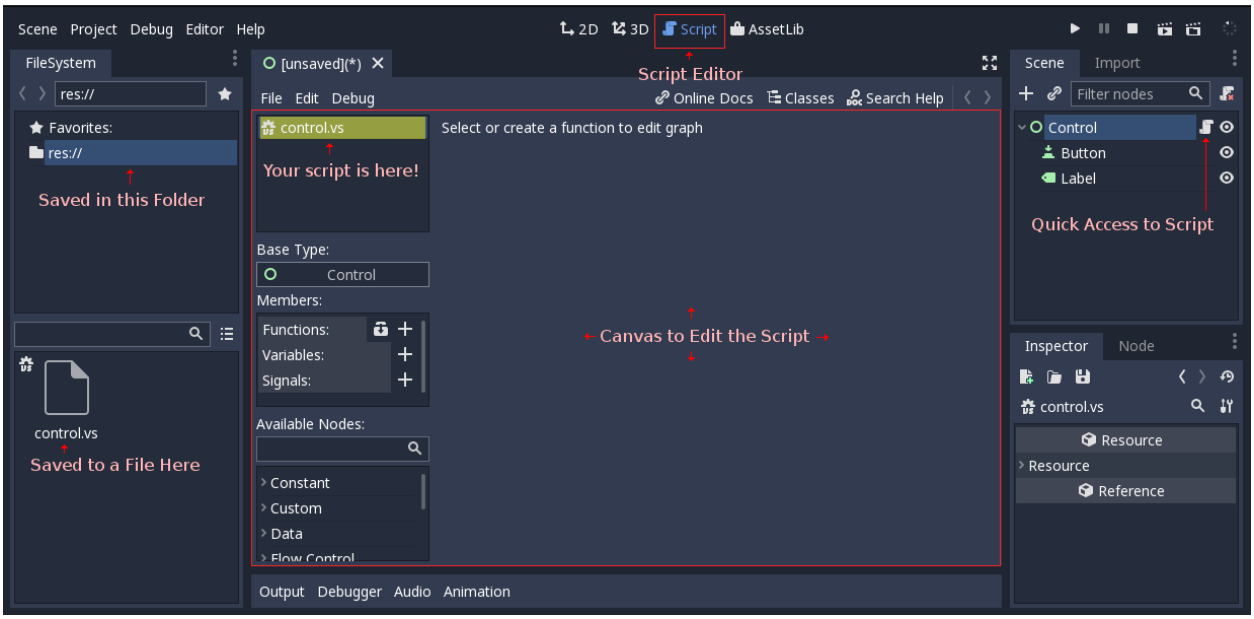
Creating scripts works the same as with other scripting languages: Select any node in the scene and push the "New Script" button at the top right corner of the Scene Tree dock:



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.

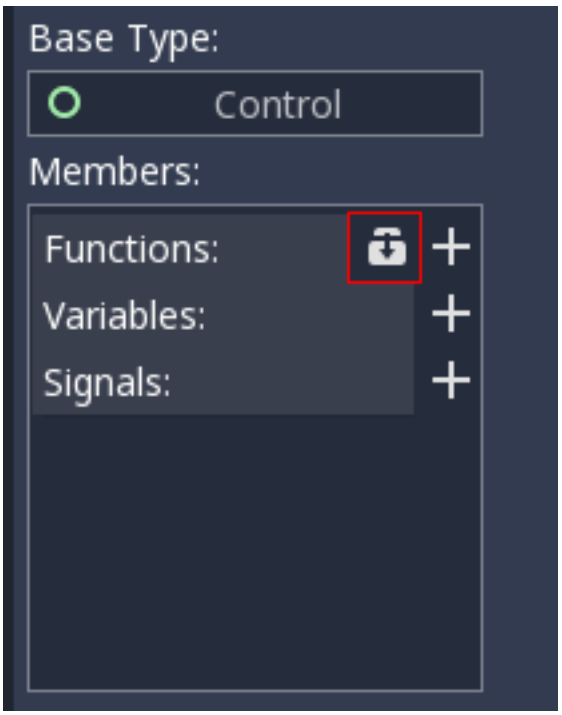
A function is an individual canvas with nodes connected.

A single script can contain many functions, each of which will have a canvas of its own, allowing for more organization.

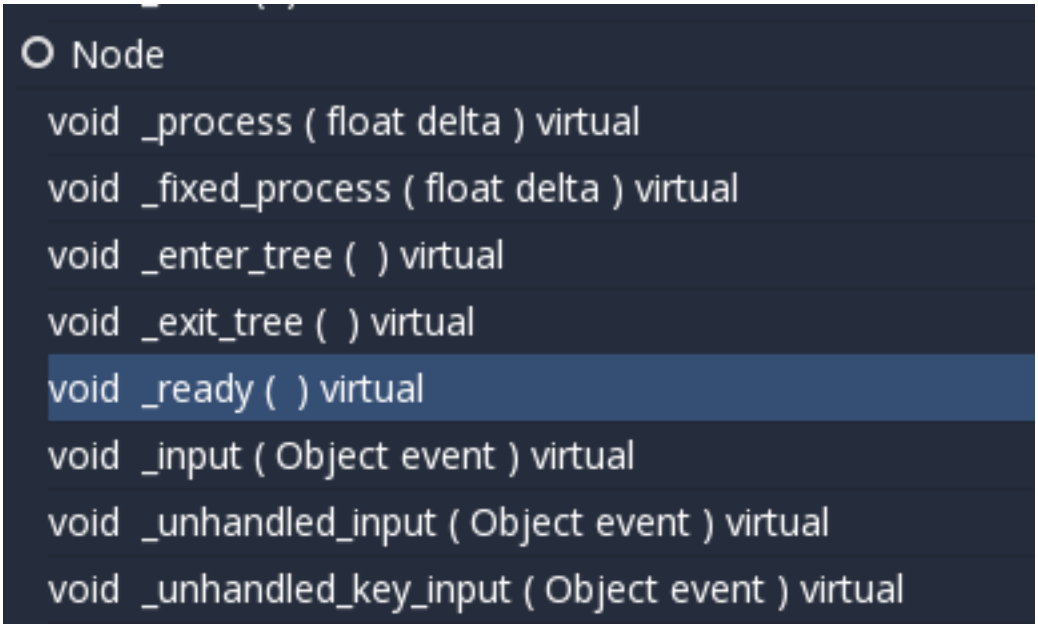
There are three main ways to add functions in a script:

Overriding a Virtual Function

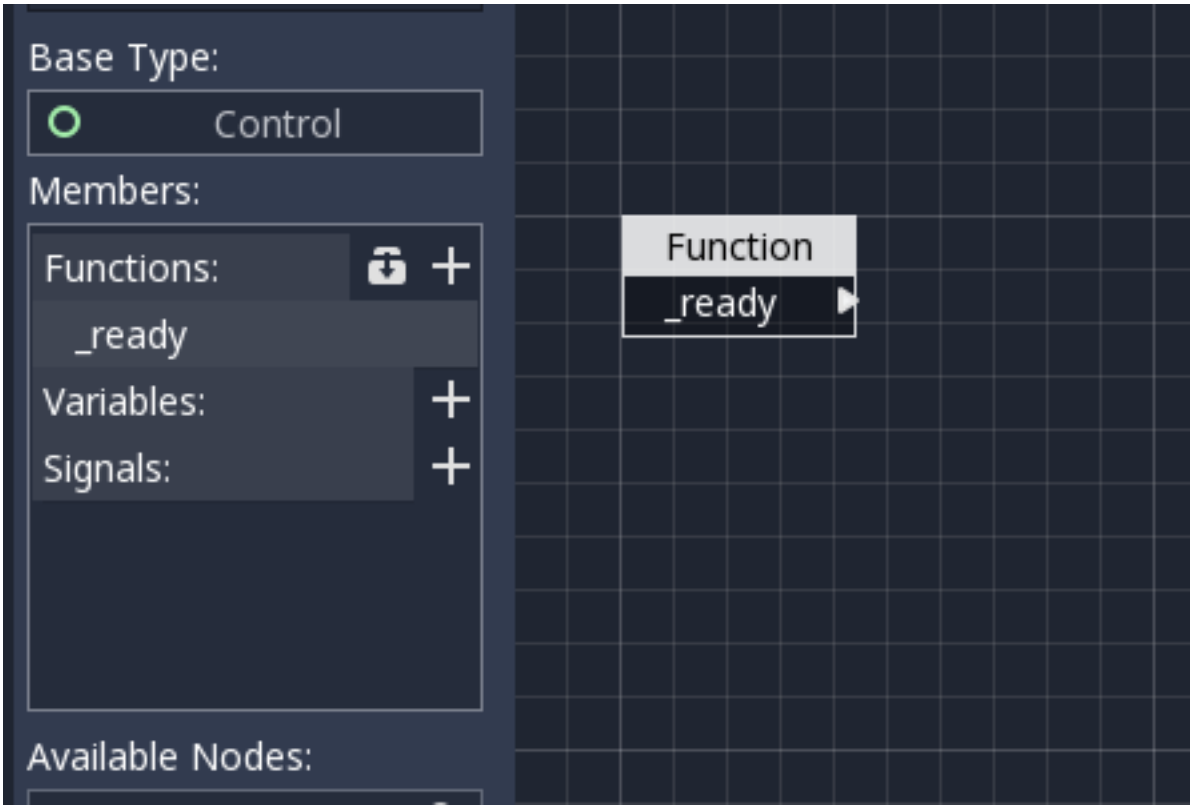
Most types of nodes and other types of objects in Godot contain virtual functions. These are functions that will be called (run your code) when something happens and can be looked up in the reference. Virtual functions are listed when pressing the "Override" icon in the member panel:



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:



Finally, a canvas appears for this function, showing the override:

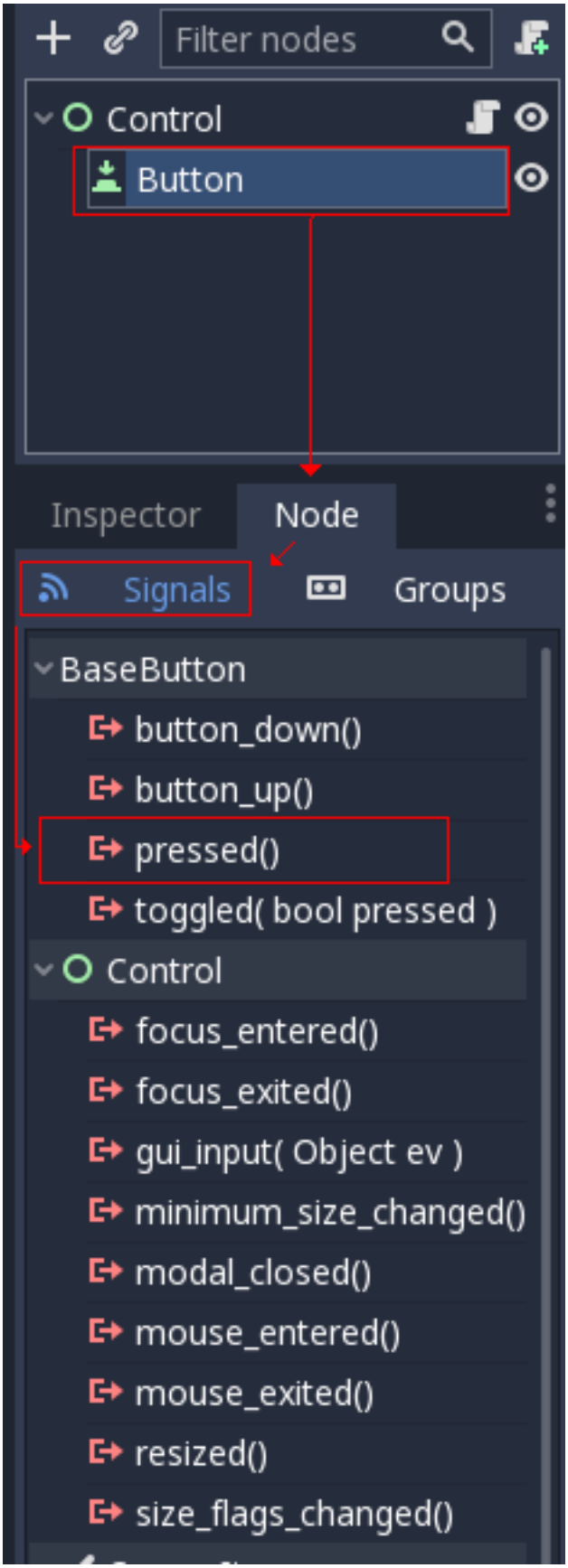


As some functions expect you to return a value, they will also add a return node where such value is supposed to be provided:

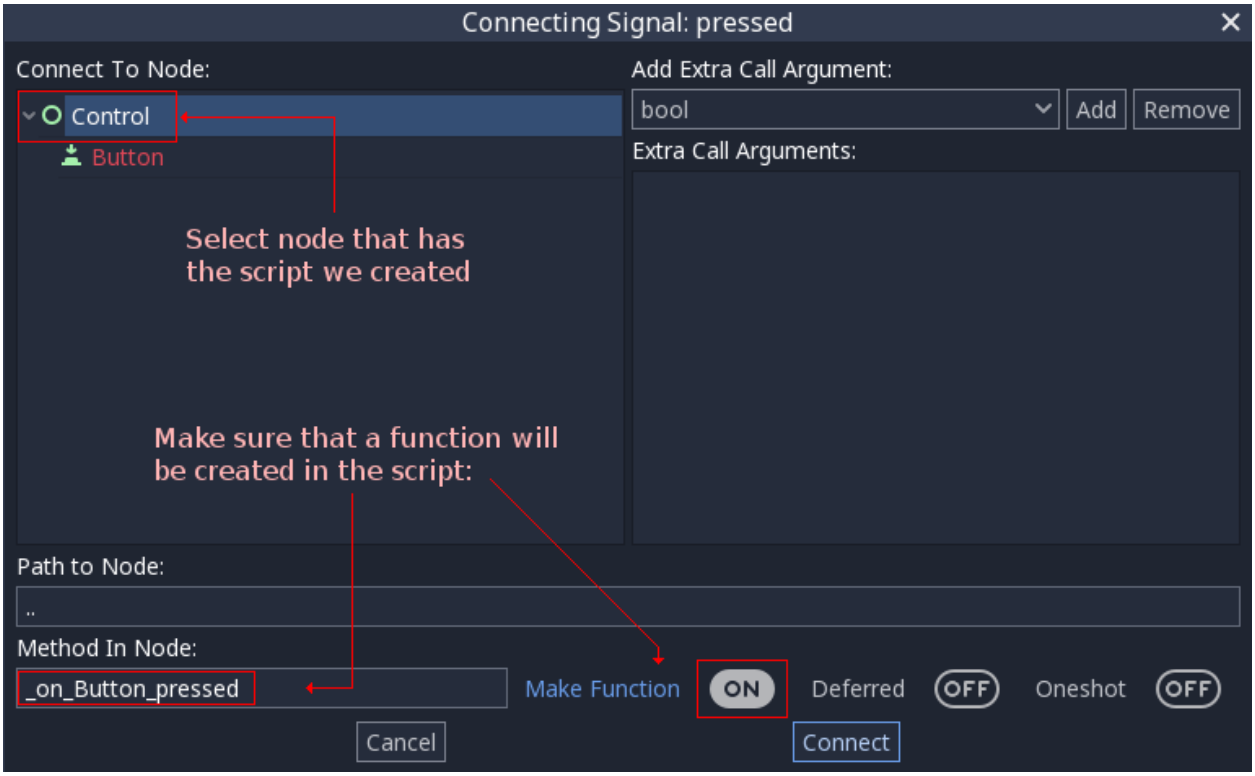
Connecting a Signal to a Function

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.

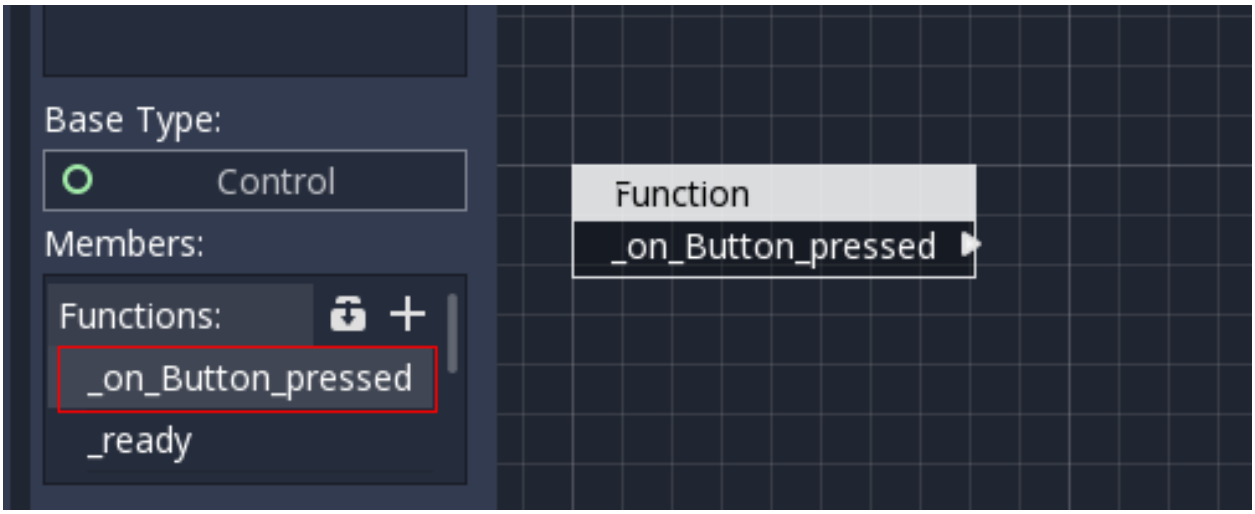
For this, a node must be selected and the Node tab opened. This will allow inspecting the signals. Once they are displayed, connect the "pressed" signal:


Filter nodes
Control
Button
Inspector
Node
Signals
Groups
BaseButton
button_down()
button_up()
pressed()
toggled(bool pressed)
Control
focus_entered()
focus_exited()
gui_input(Object ev)
minimum_size_changed()
modal_closed()
mouse_entered()
mouse_exited()
resized()
size_flags_changed()

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:



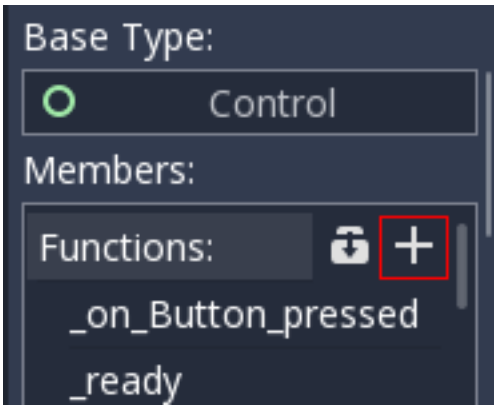
If this is done right, a new function will be created in our script and a signal will automatically be connected to it:



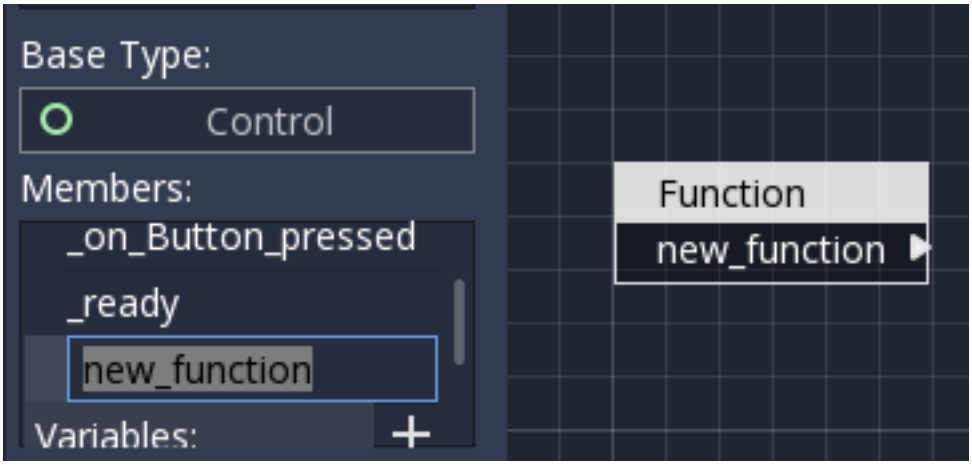
Creating a Function Manually

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.

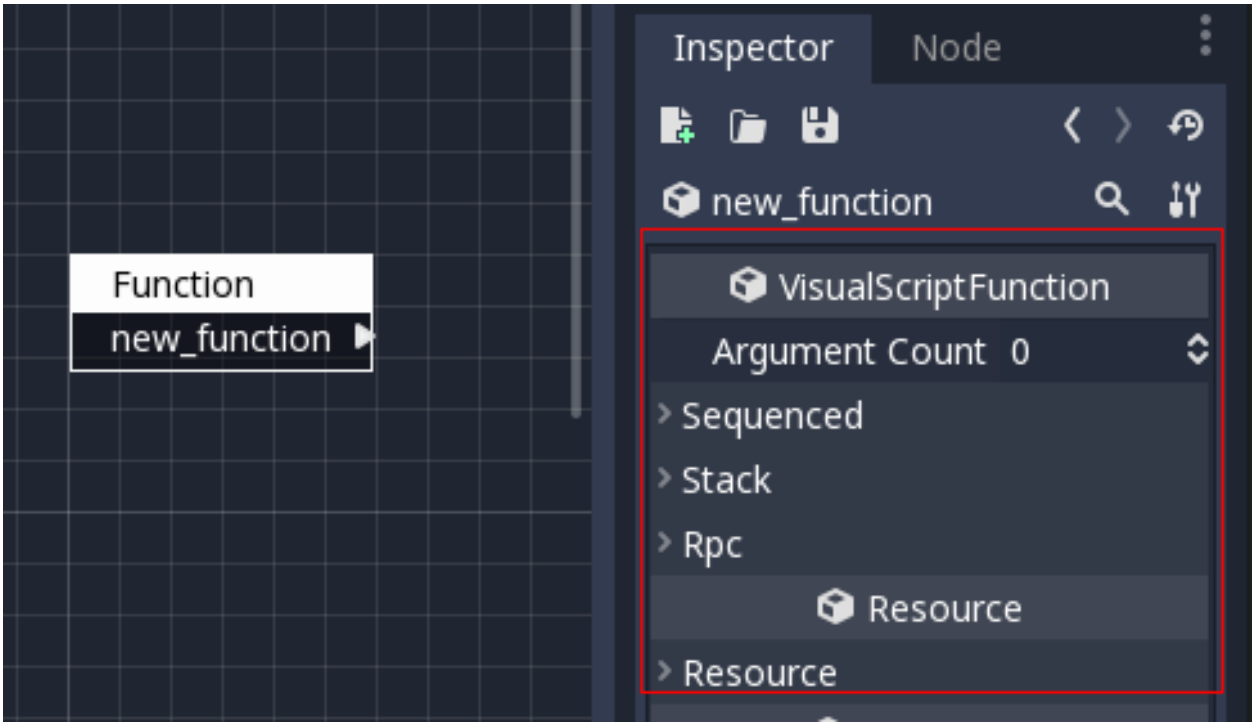
To create a function manually, push the big "Plus" button, and a new function will be added with a default name:



This will add a new function, which can be renamed by simply double clicking its name:



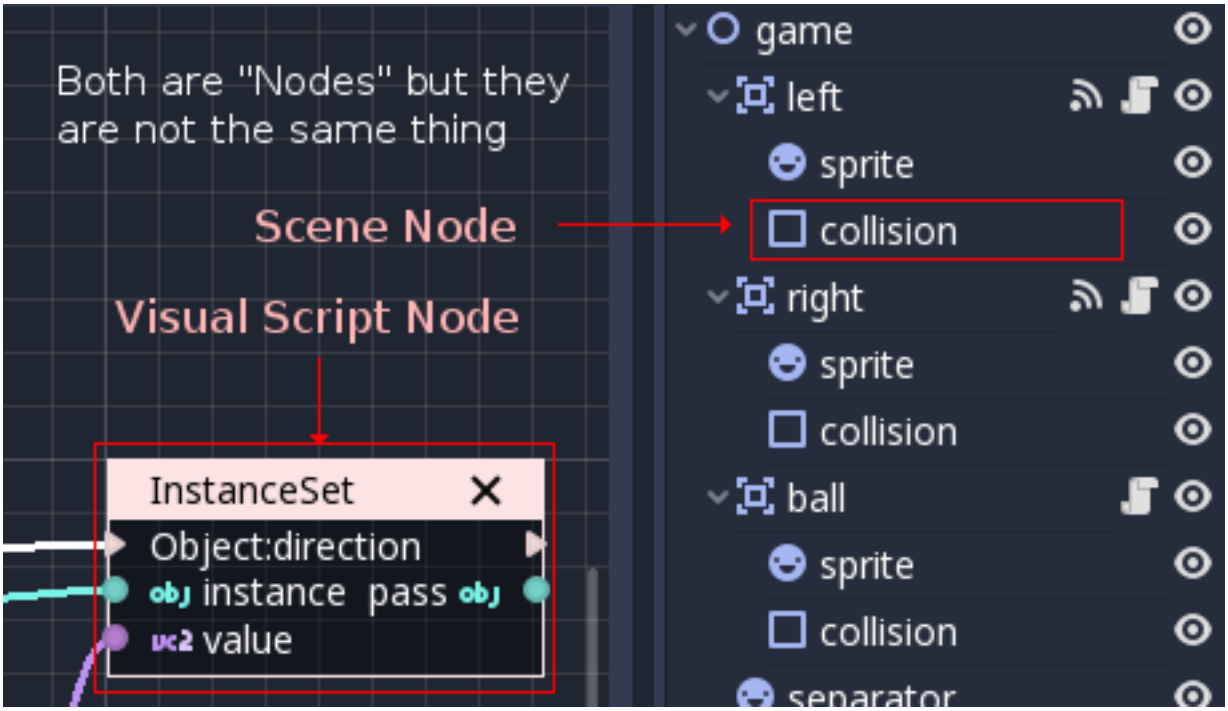
To edit the "arguments" this function can get (the values you pass to it when you call this function), simply click the Function node and check the inspector:



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.

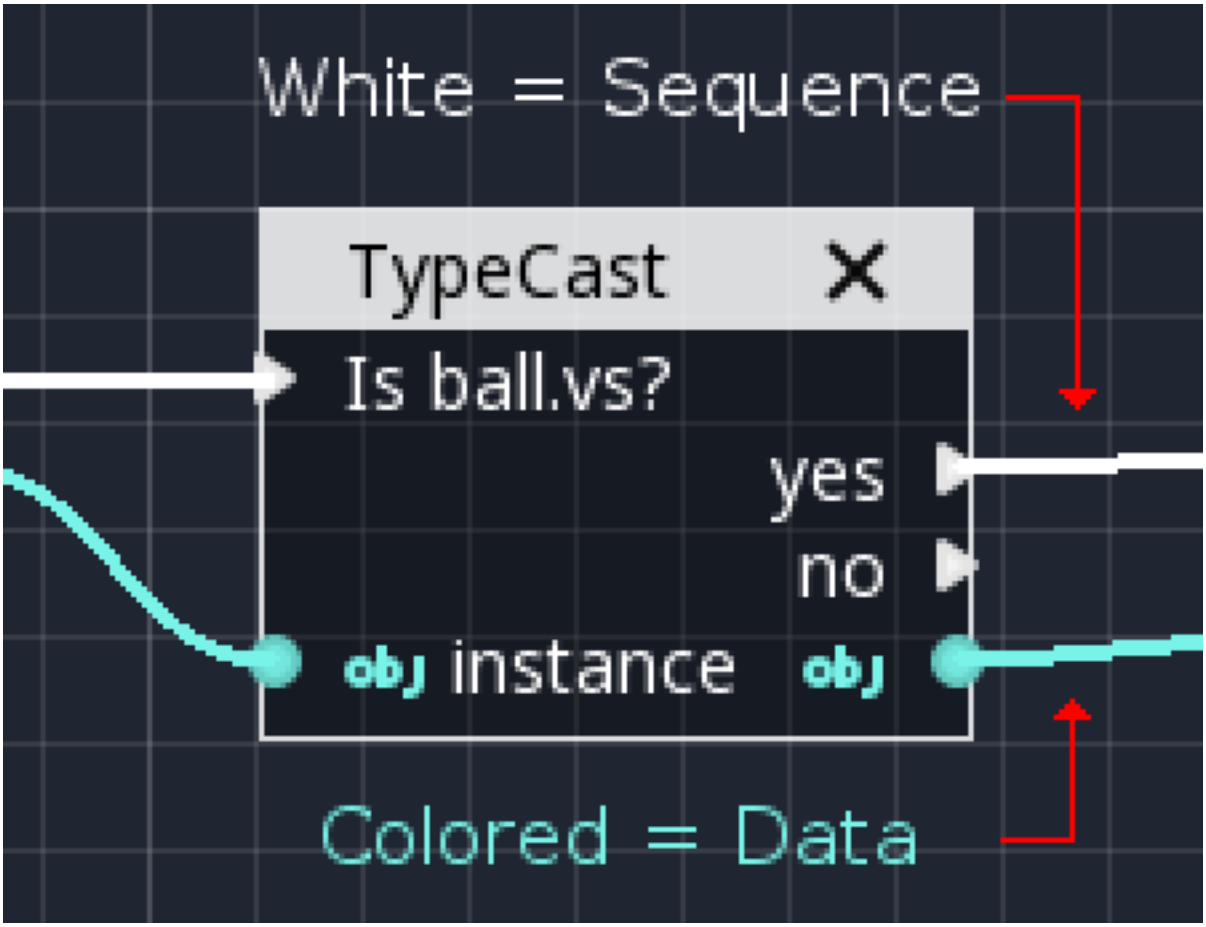
Nodes still display the required information as text, but editing is done via the *Inspector*. To edit them, select any node and edit its properties in the *Inspector*.

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, you can follow the logic flow within a function by 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:

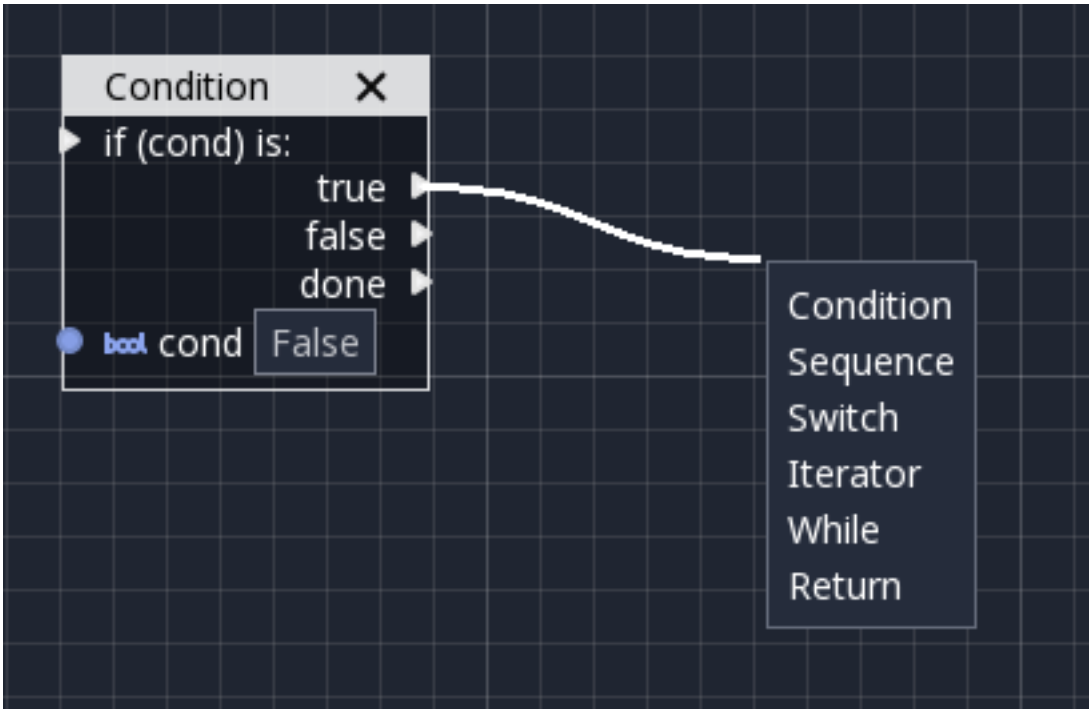
Connections

Connecting is a relatively simple process. Drag an *Output Port* towards an *Input Port*.

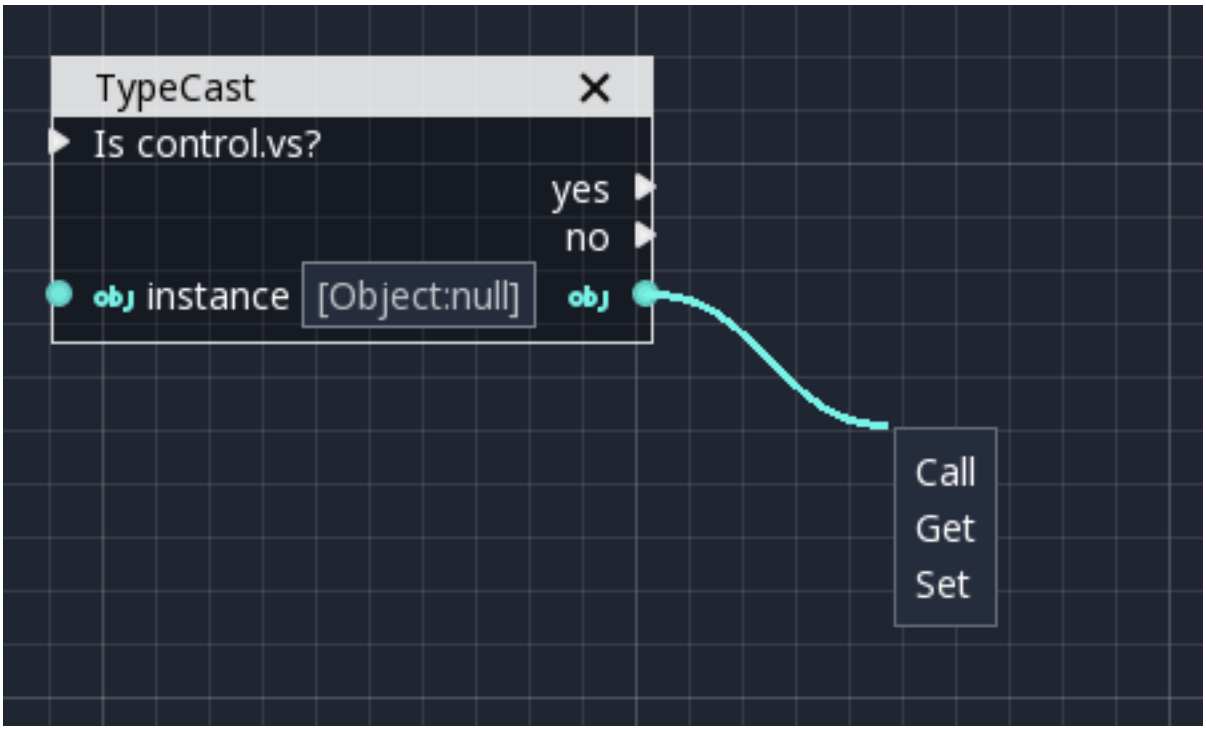
Disconnecting takes a bit more practice. Disconnecting in *Data Ports* happens by dragging the *Input* away, while for *Sequence Ports*, this happens by dragging the *Output* away.

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:

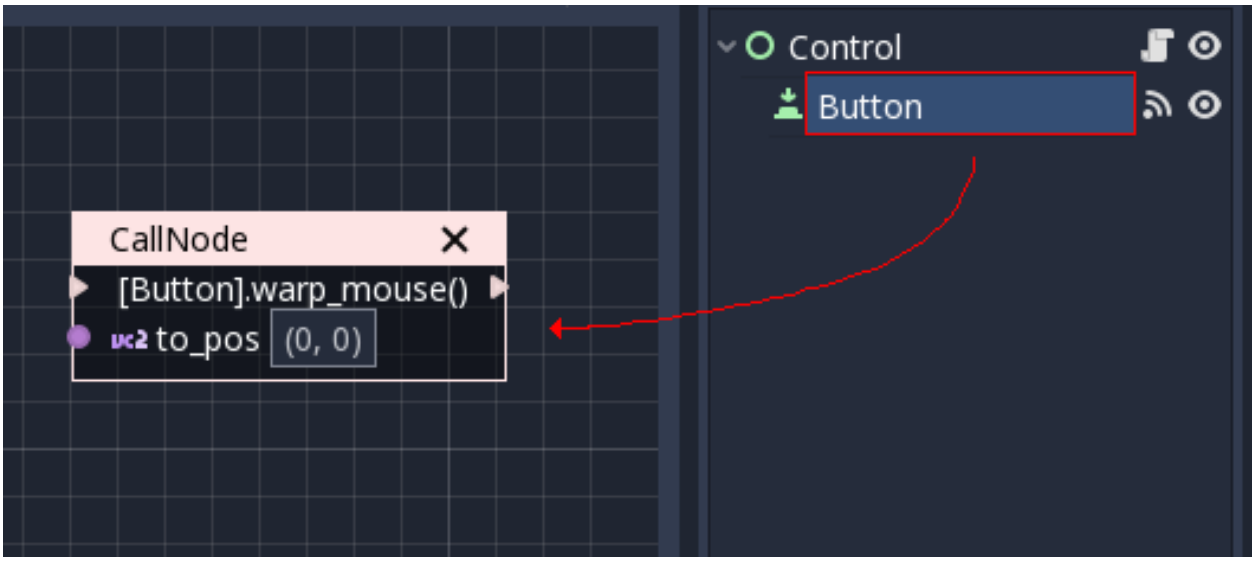


Adding Nodes

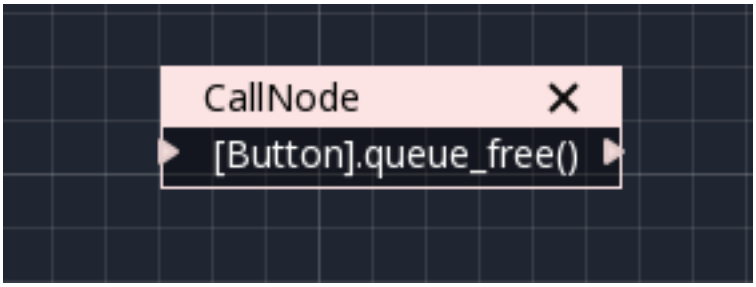
Finally! We got to the fun part! But, before explaining in more detail what each type of node does, let's take a short look at how nodes are most commonly added and dealt with.

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.



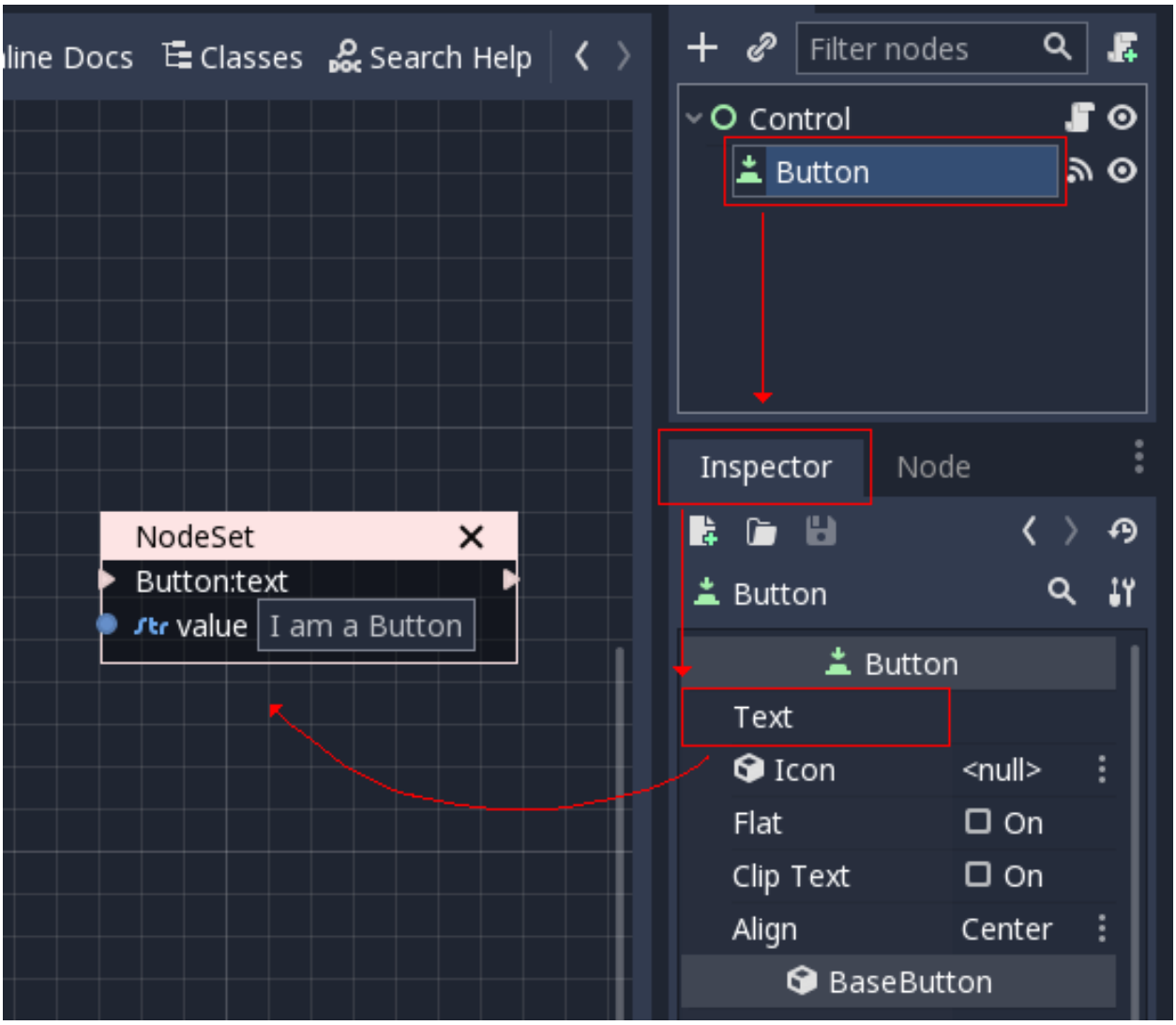
While accessing properties is desired in most cases (more on that below), sometimes *calling methods* can be useful too. Methods execute specific actions on objects. In the above case, the mouse pointer can be warped to a position in local coordinates to the control. Another common use case is queueing 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.

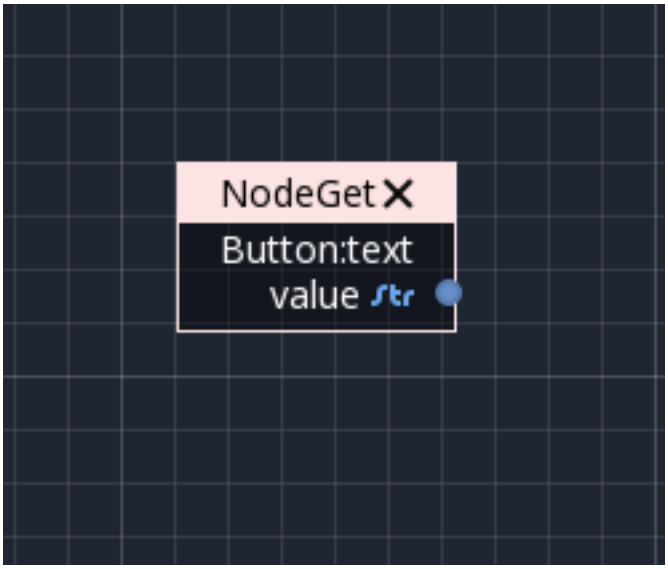
Accessing Scene Node Properties

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, drag the node again but hold the *Control* key (or Command on Mac). This will create a getter:

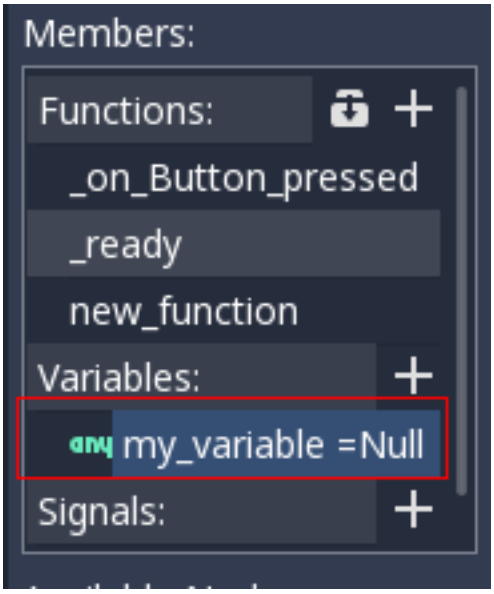


In this case, the value can be read from a *Data Port*.

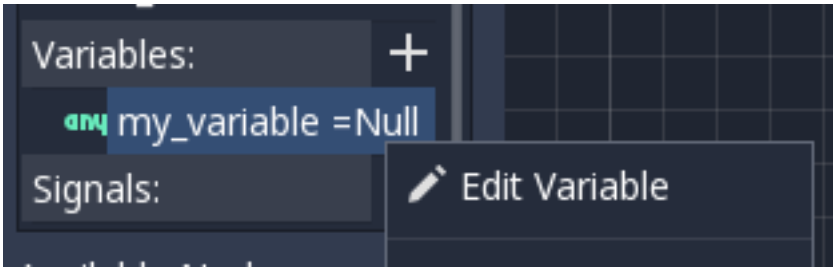
Variables

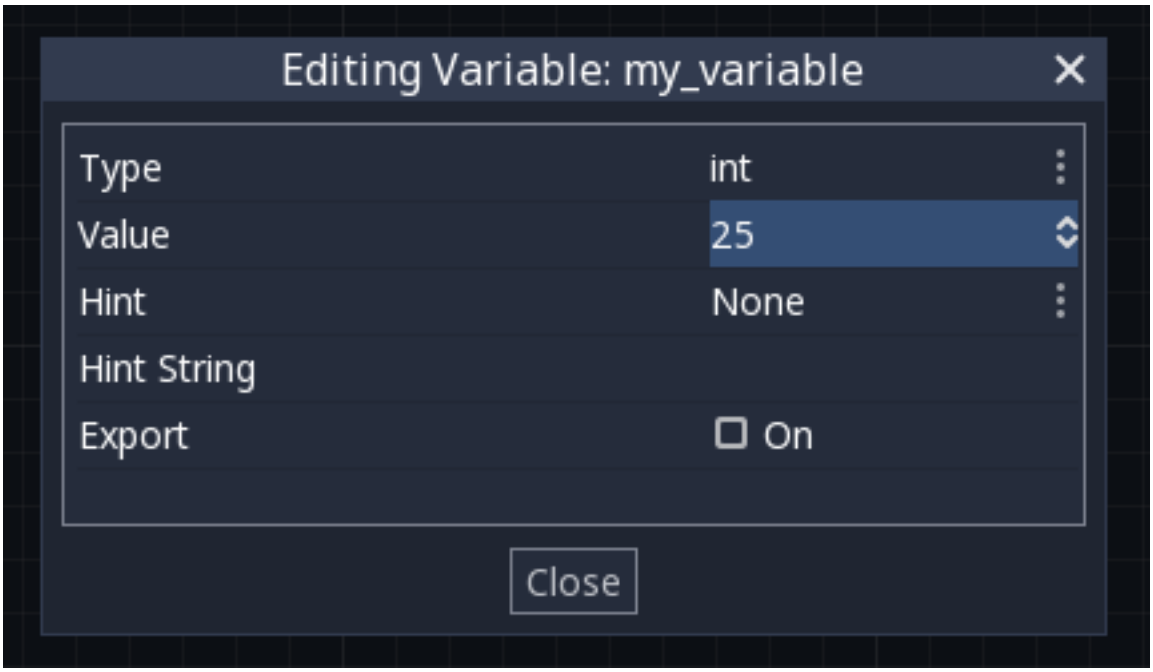
Variables are memory containers local to the script which can hold a value. This value can be read from any of the functions of the script or from other scripts via the method described in the previous step.

To add a Variable, push the "+" button on the *Variables* section of the Members panel. Double-click the new variable to rename it:

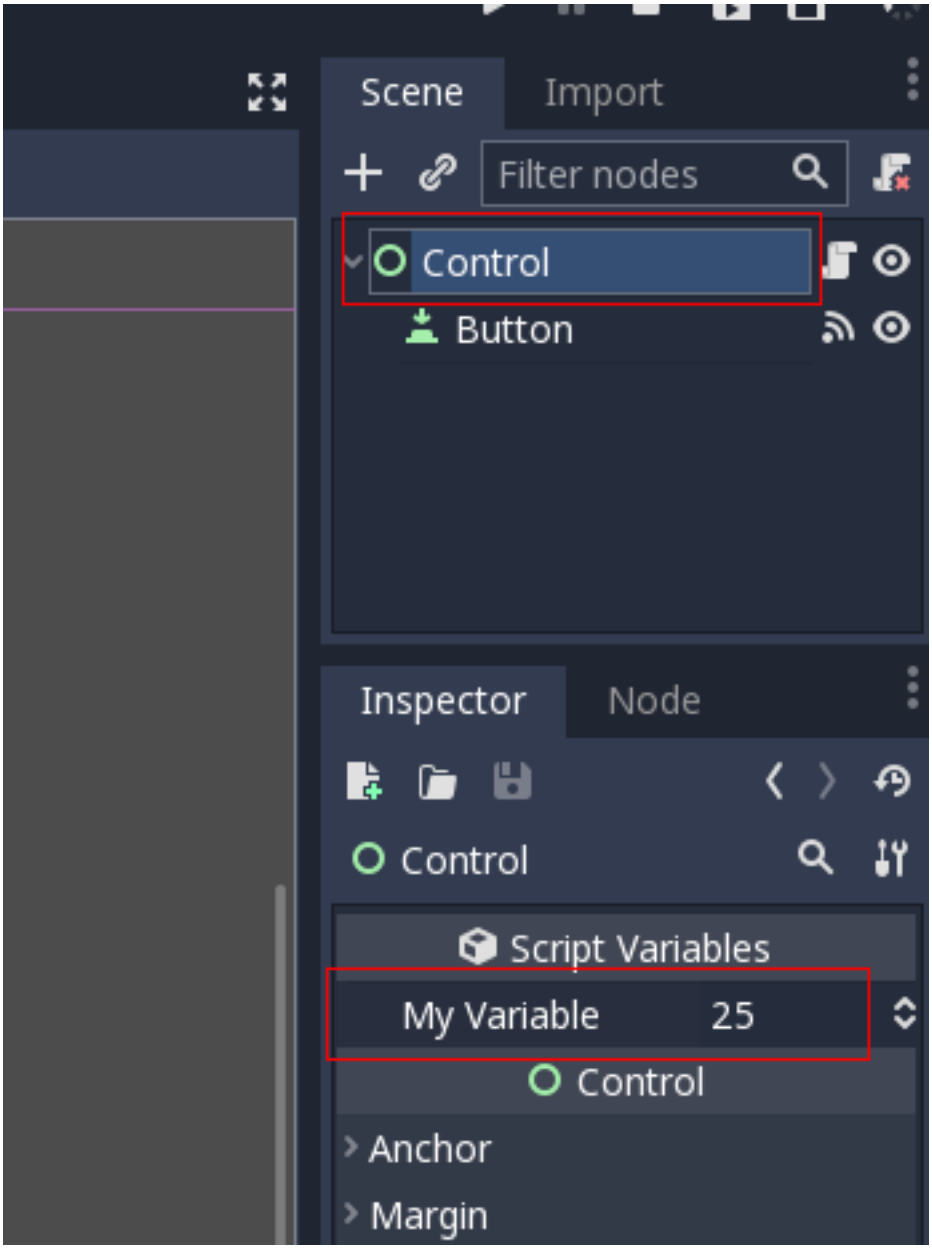


Right-clicking the variable allows you to configure its properties:

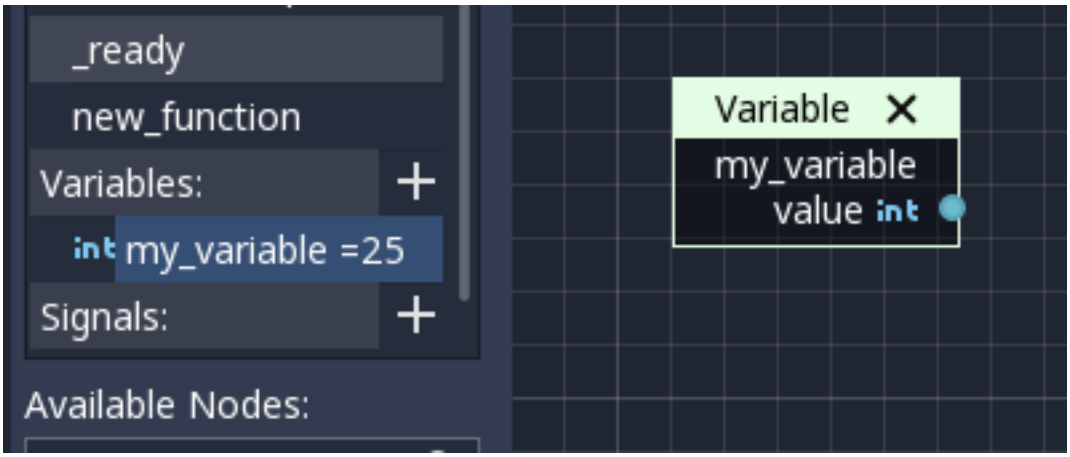




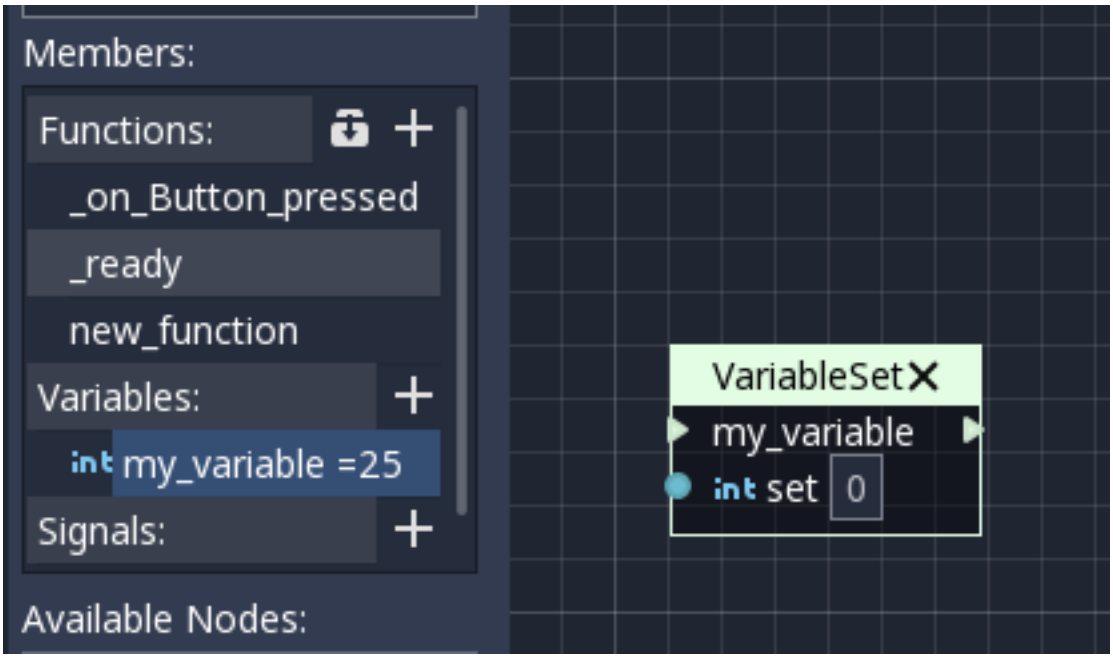
As it can be seen above, the type and initial value of the variable can be changed, as well as some property hints (@TODO, document this). Ticking the "Export" options makes the variable visible in the Inspector when selecting the node. This also makes it available to other scripts via the method described in the previous step.



To use the variable in the script, simply drag it to the canvas to create a getter:

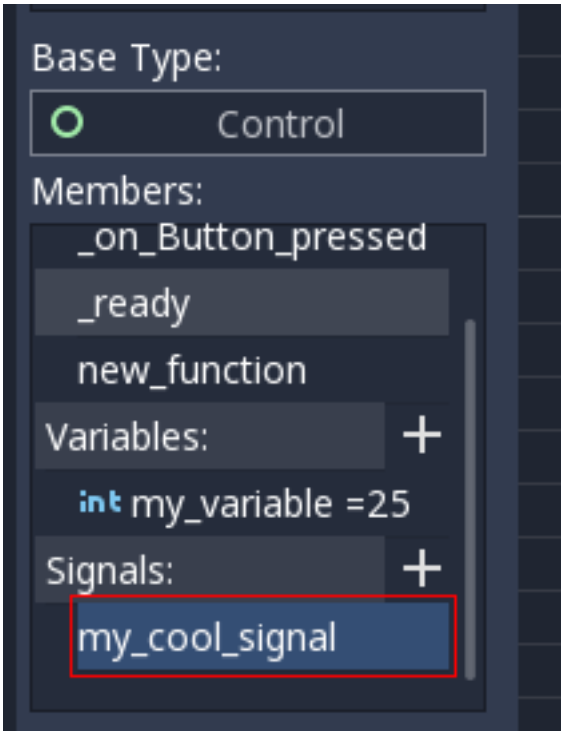


Likewise, hold *Control* (*Command* on Mac) to drop a setter:

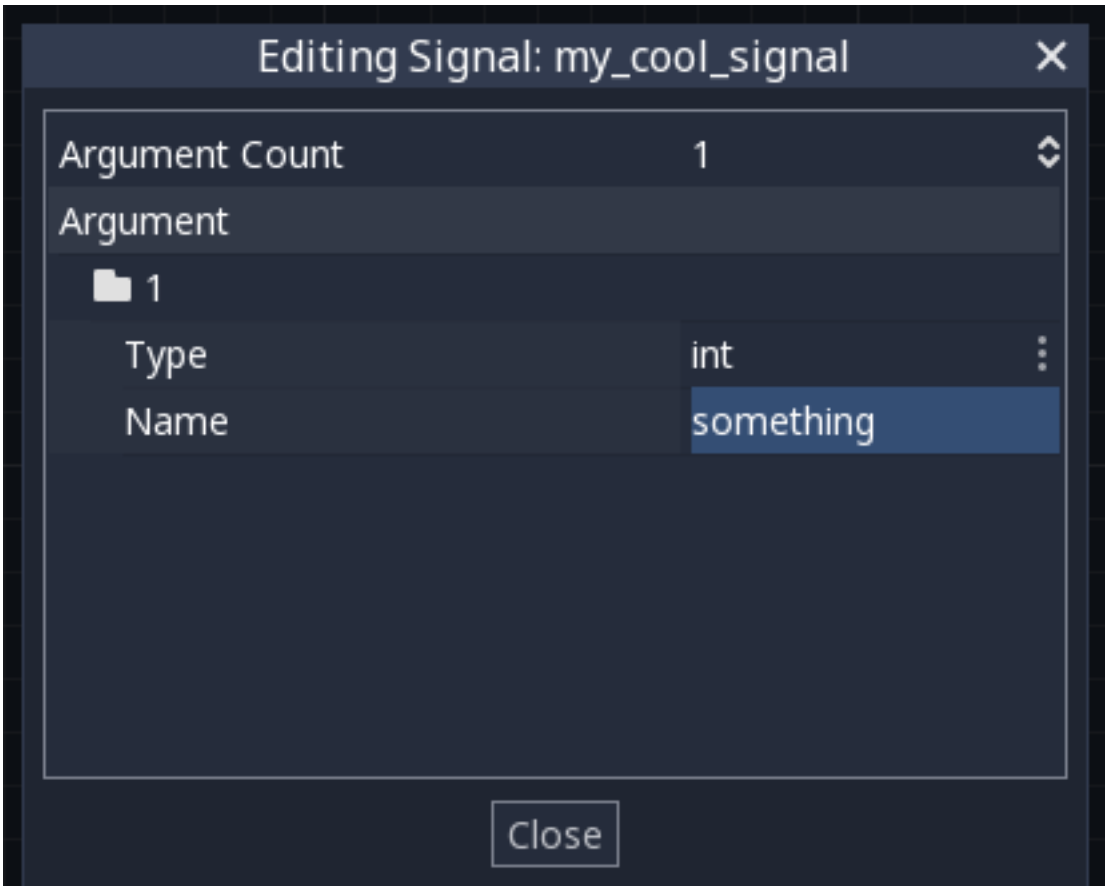


Signals

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*:

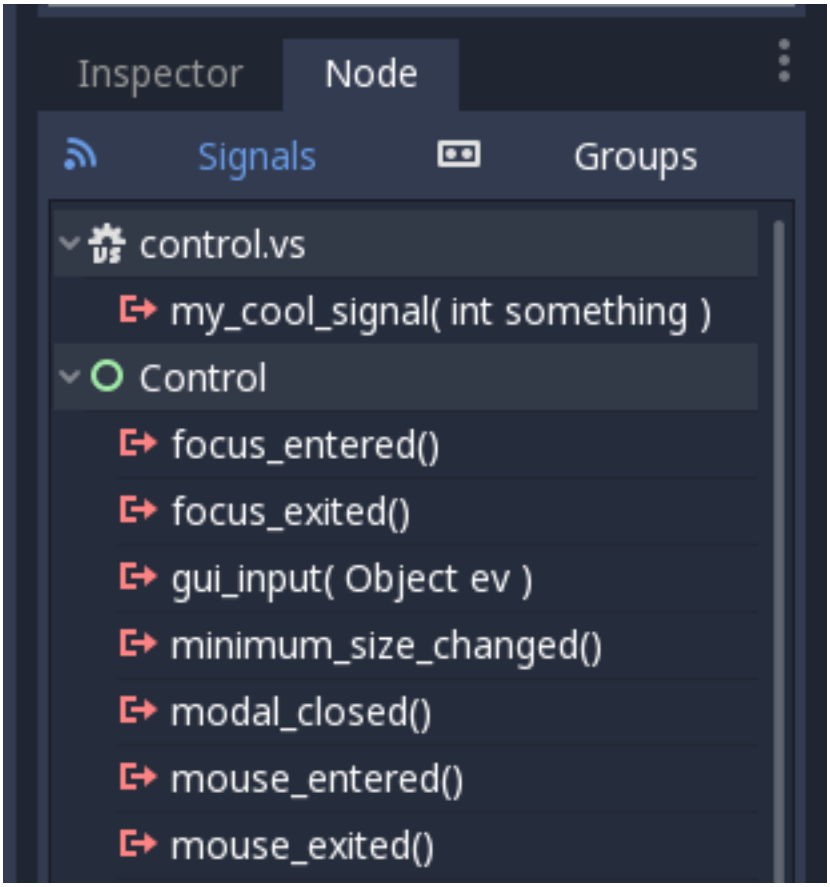


A signal can also be edited via right-click menu to customize its arguments:

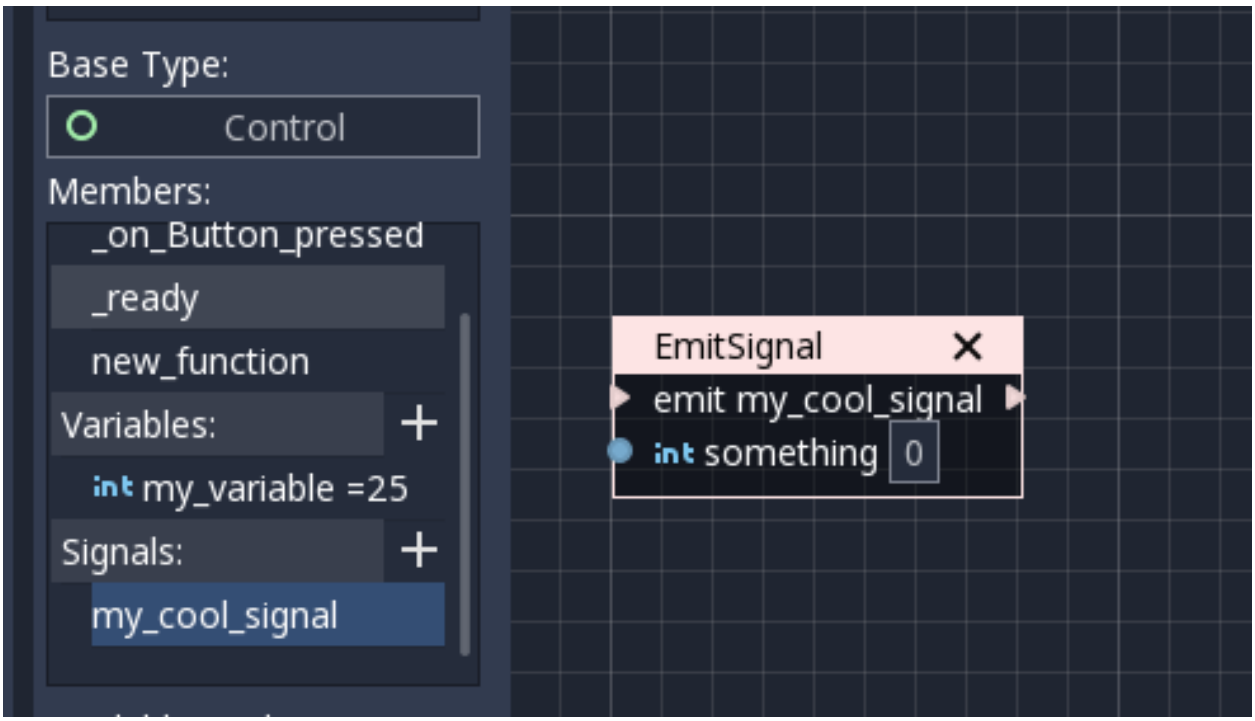


The signal you have created will appear in the Inspector along with the built-in node signals. This allows you to

connect it from another script from another *Scene Node*:



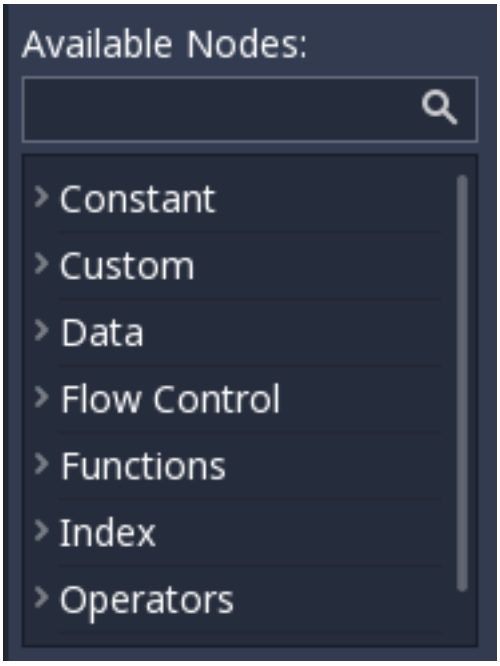
Finally, to emit the signal, simply drag it to the canvas:



Remember that emitting a signal is a sequenced operation, so it must come from a Sequence port.

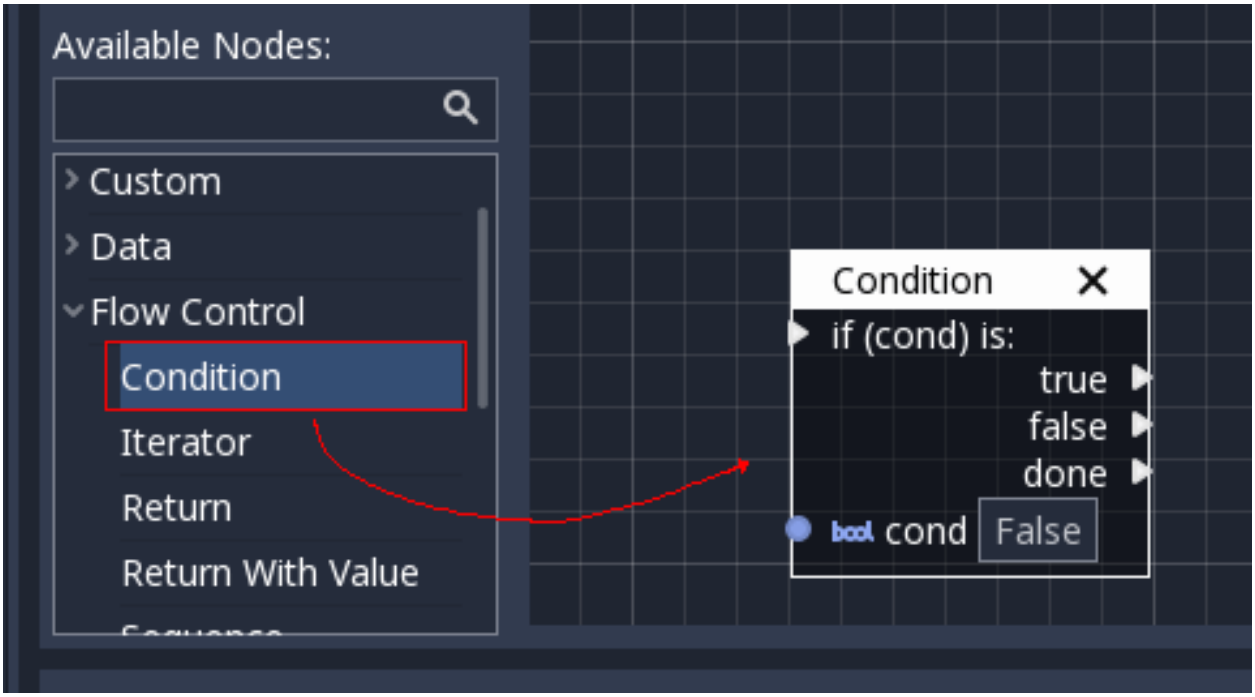
Adding More Nodes

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.

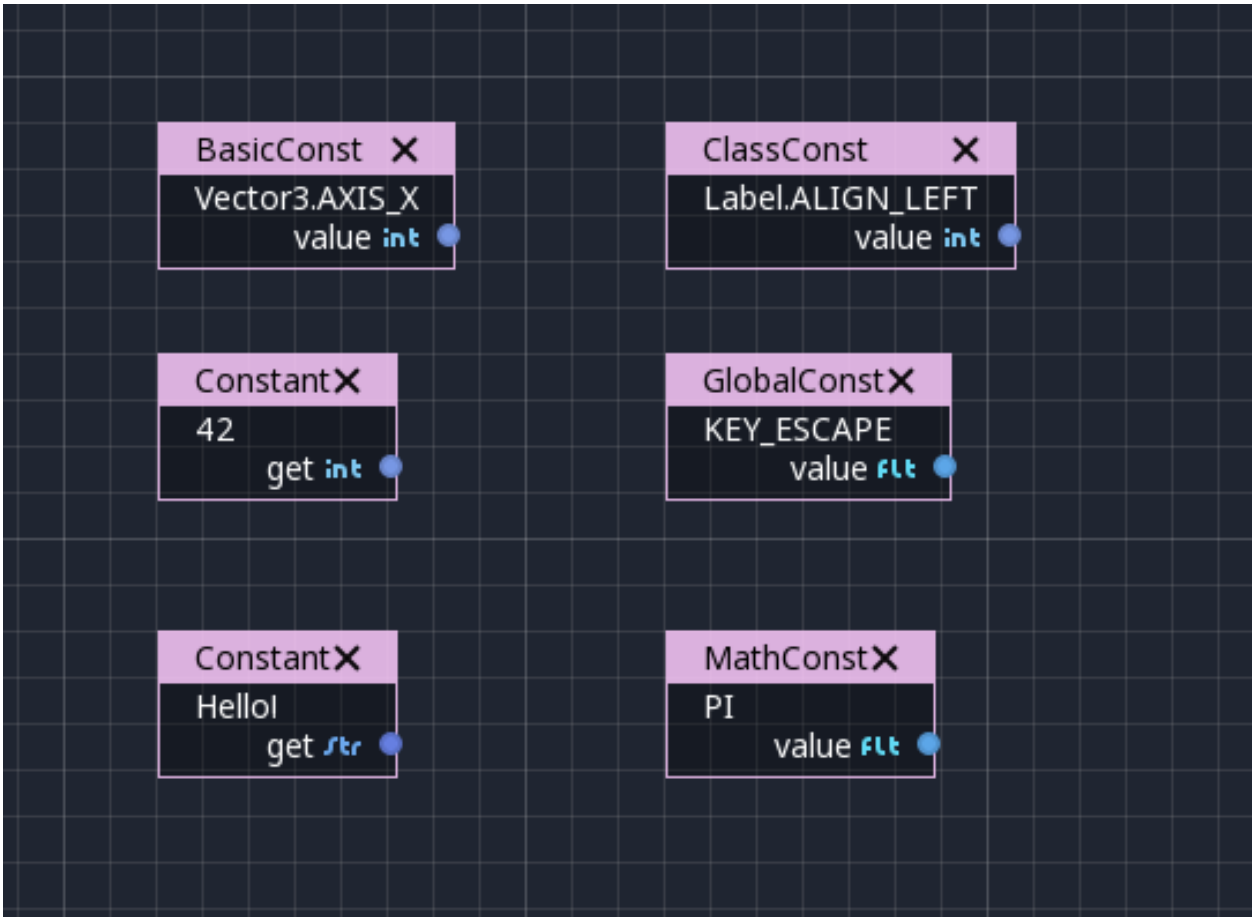
Any of them can be dragged to the scene. Unlike nodes (e.g. dragging a property from the Inspector sets the context to the node being edited automatically), these are added without any "contextual" information, so this has to be done manually.



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.



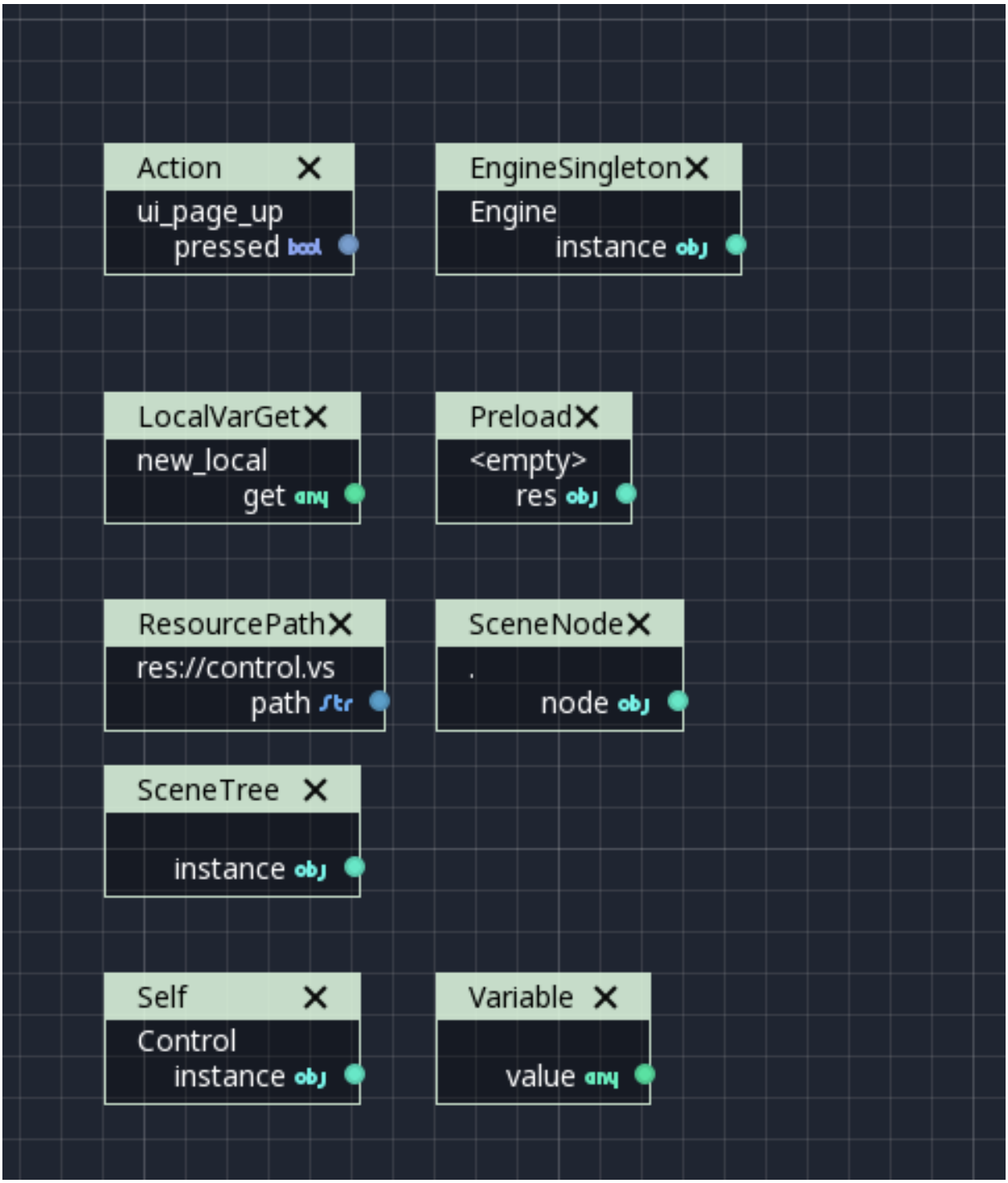
The first one is "Constant" which allows you to select any value of any type as constant, from an integer (42) to a String ("Hello!"). In general this node is not used that often because of default input values in *Data Ports*, but it's good to know it exists.

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.

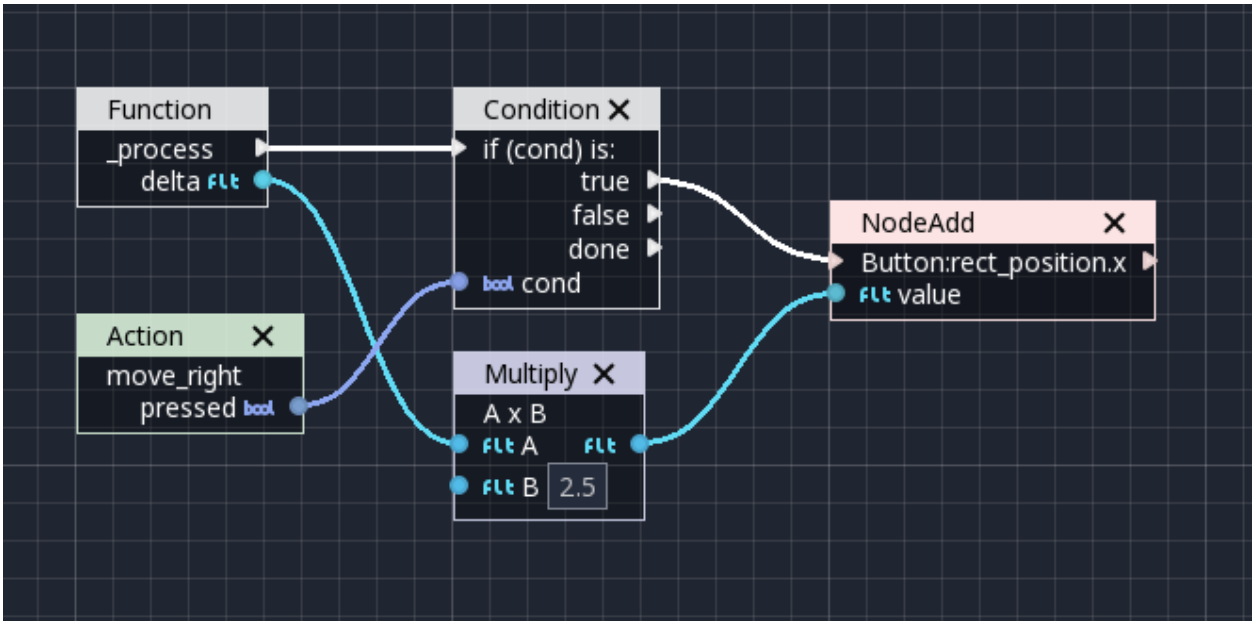


There are many types of nodes of interest here, so a short attempt to describe them will follow:

Action

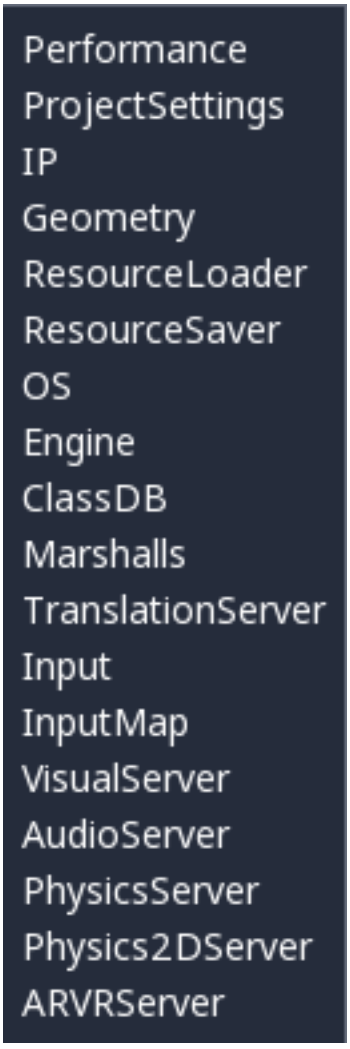
Action nodes are vital when dealing with input from a device. You can read more about actions in the (@TODO ACTION TUTE LINK). In the following example below, the control is moved to the right when the "move_right"

action is pressed.



Engine Singleton

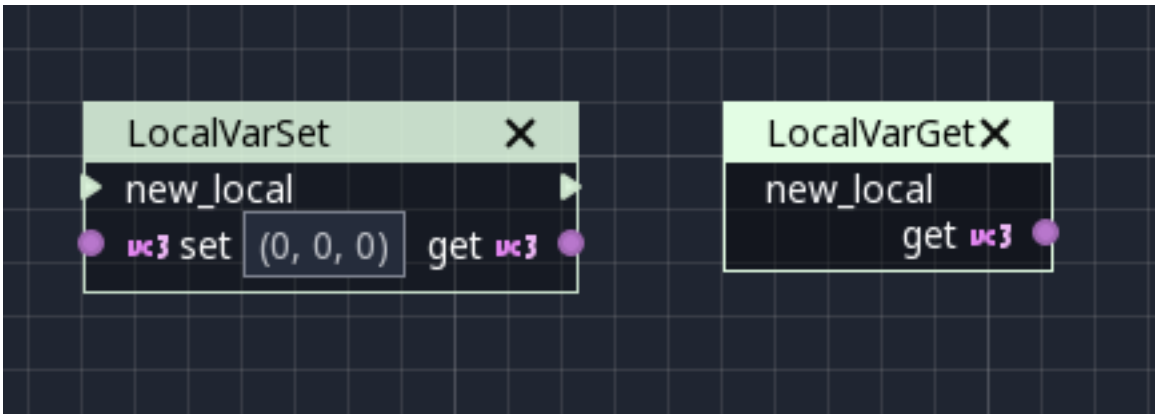
Engine singletons are global interfaces (meaning they can be accessed without a reference, unlike Scene Nodes, they are always available). They have several purposes, but in general they are useful for low level access or OS-related access.



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. 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 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 drag nodes (or event 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

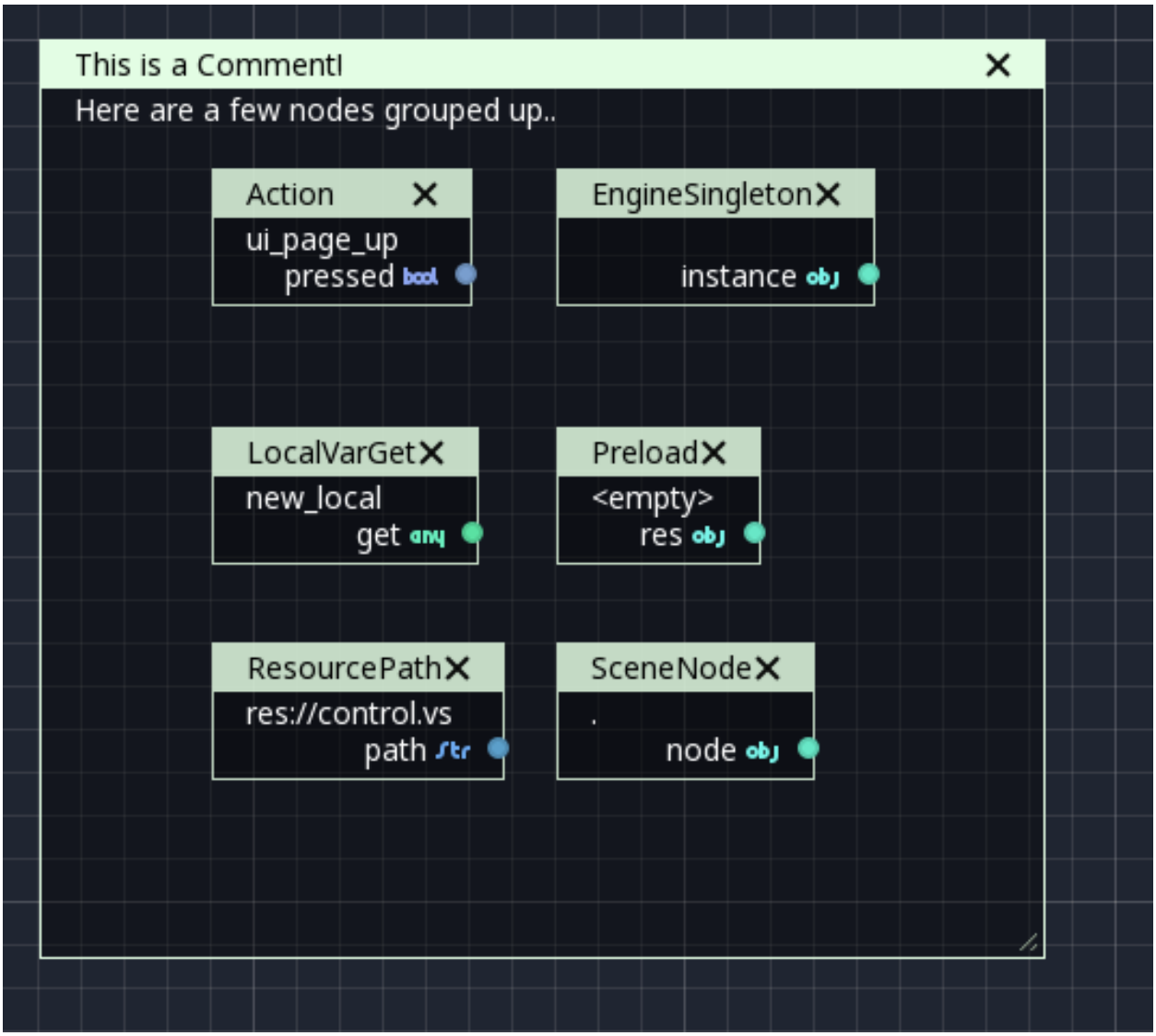
This does the same function as preload() in GDScript. It maintains this resource loaded and ready to use. Rather than instancing the node, it's simpler to drag the desired resource from the filesystem dock to the canvas.

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.

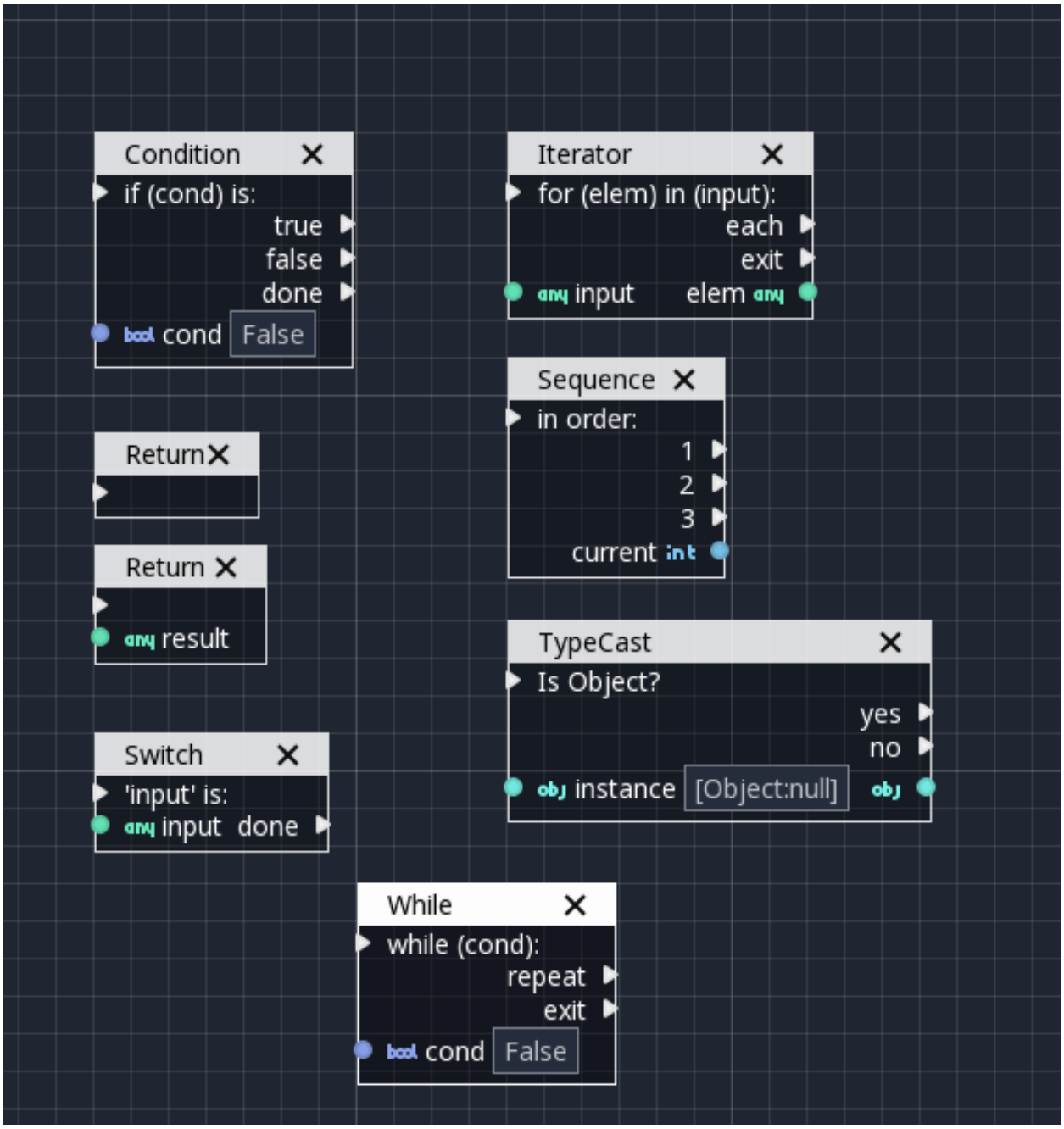
Comment

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

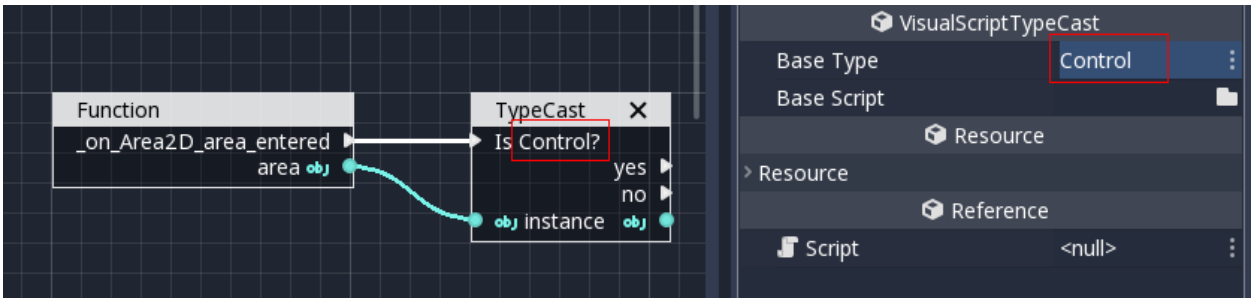
Some functions can return values. In general for virtual ones, Godot will add the Return node for you. A return node forces the function to end.

Sequence

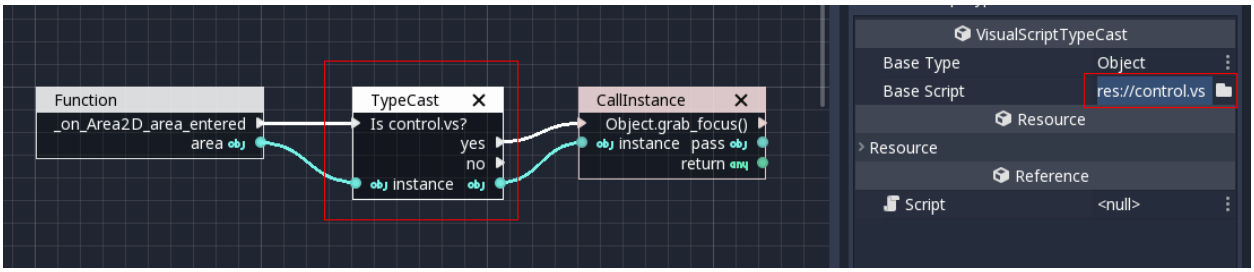
This node is useful mostly for organizing your graph. It calls its sequence ports in order.

TypeCast

This is a 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 useful helpers. Make sure to take a look at the list at some point.

By Type

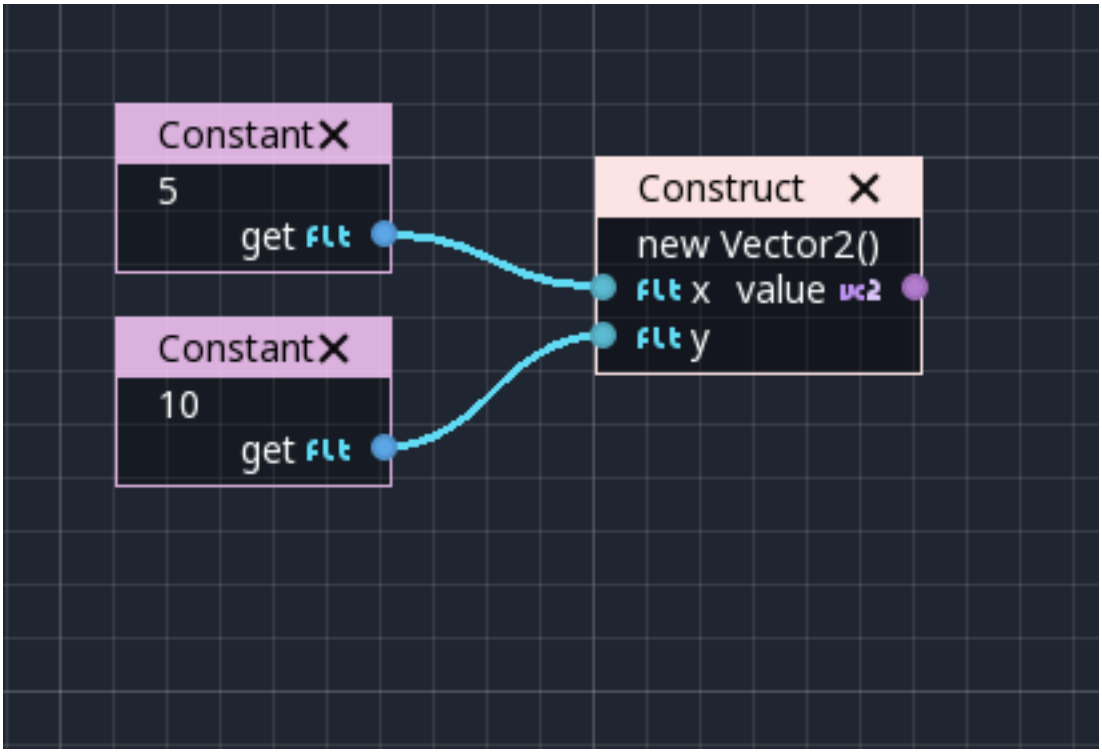
Those are the methods available to basic types. For example, if you want a dot-product, you can search for "dot" instead of the Vector3 category. In most cases just search the list of nodes, it should be faster.

Call

This is the generic calling node. It is rarely used directly but by dragging to empty space on an already configured node.

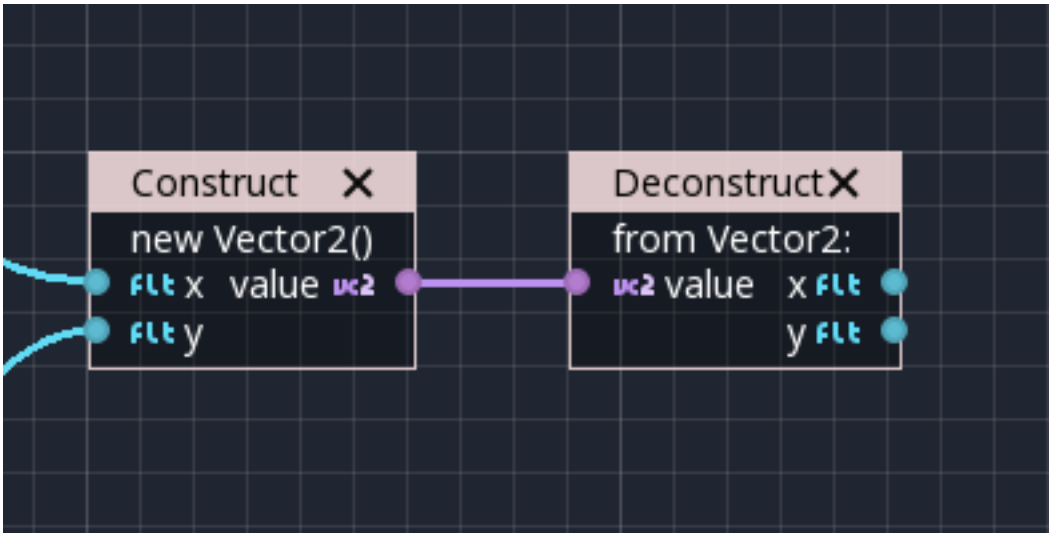
Constructors

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 that 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

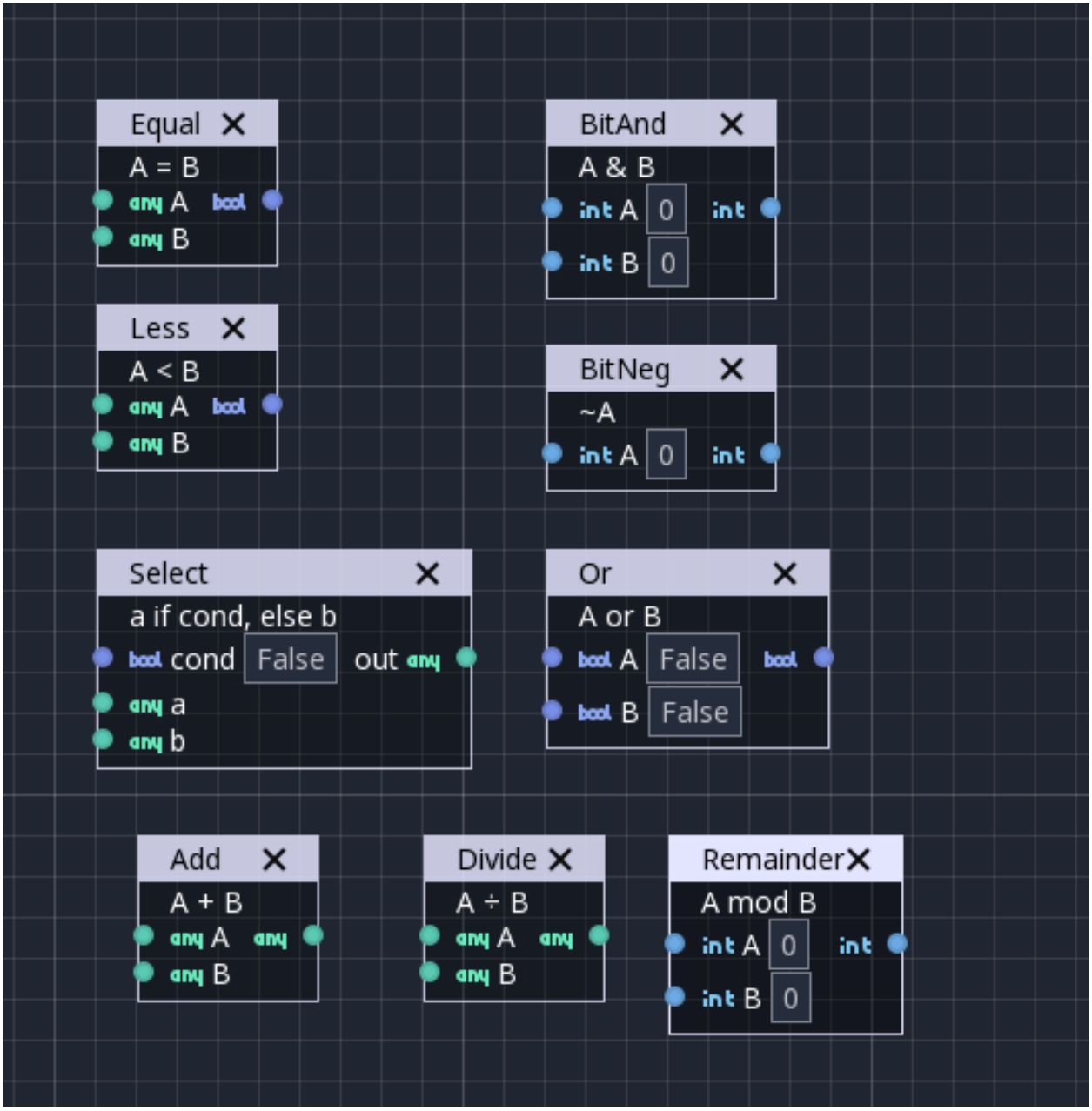
Same as Yield, but will wait until a given signal is emitted.

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.

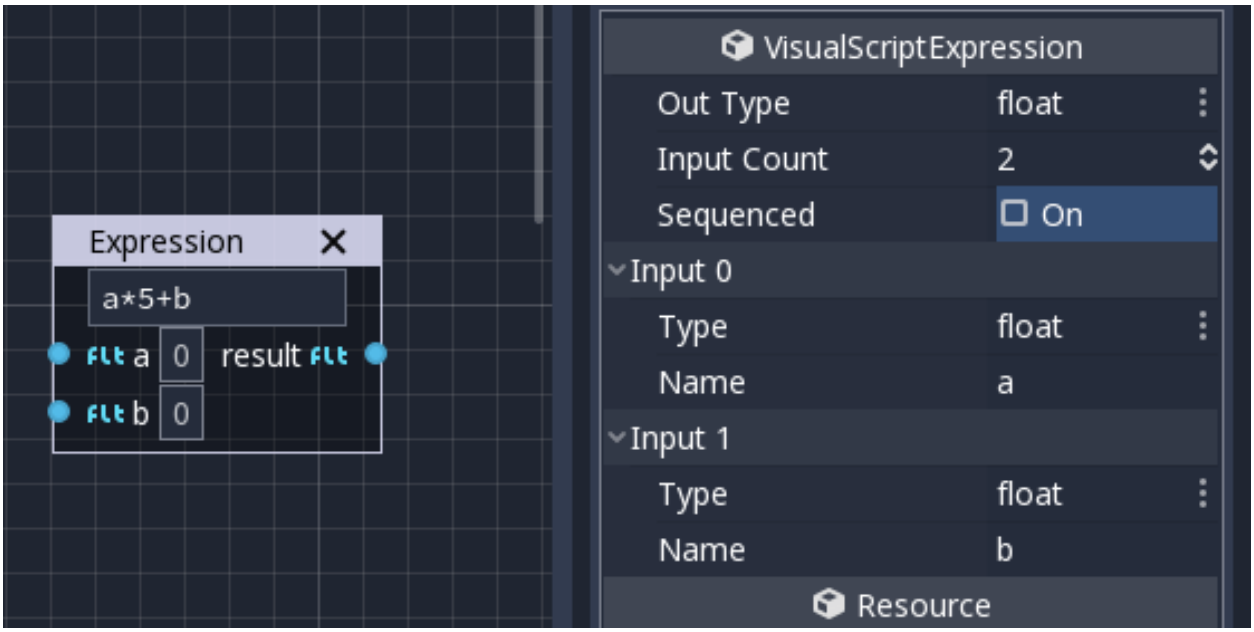


Expression Node

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. Type any expression on it and it will be executed in real-time.

Expression nodes can:

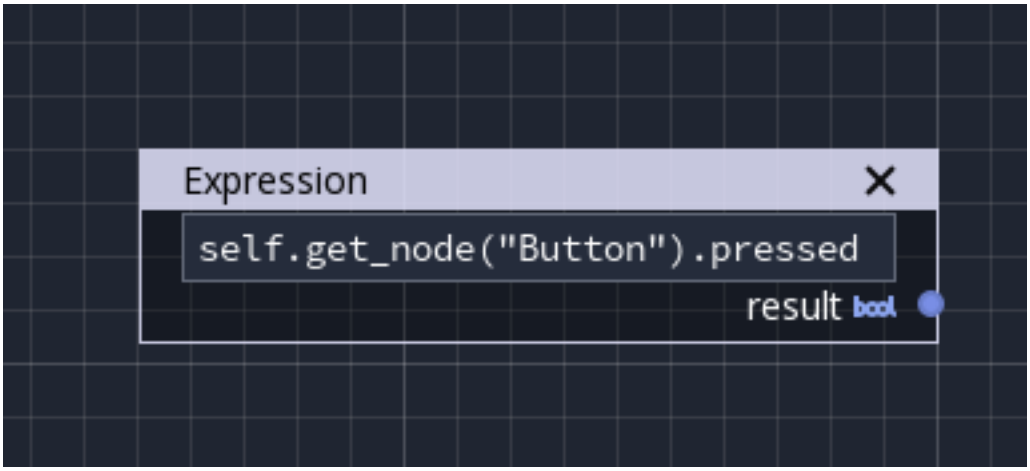
- Perform math and logic expressions based on custom inputs (eg: "a*5+b", where a and b are custom inputs):



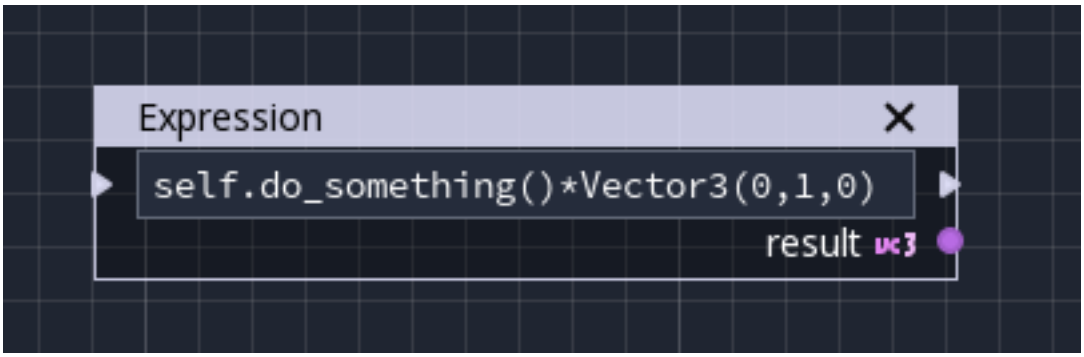
- Access local variables or properties:

- 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.x .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. Godot 3.0.2 requires Mono 5.4, 3.0.3 requires Mono 5.12 on all platforms. You also need MSBuild (at least version 15.0) which should come with the Mono installation.

Note: For instructions on installing older versions of Mono on Linux see this page. Older versions of Mono for MacOS and Windows can be found here.

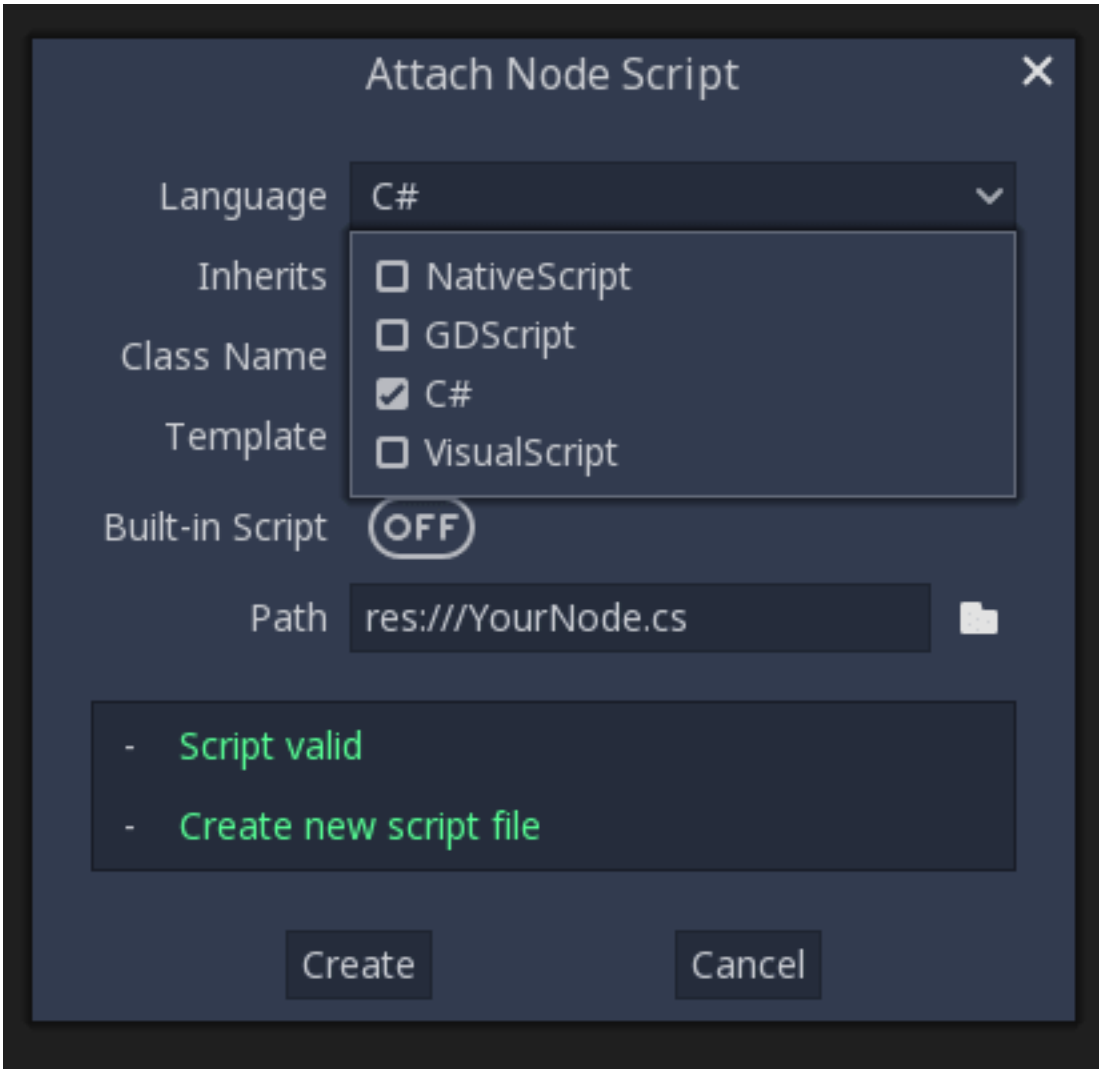
Additionally, your Godot version must have Mono support enabled, so take care to download the **Mono version** of Godot. If you are building Godot from source, make sure to follow the steps to include Mono support in your build outlined on the *Compiling with Mono* page.

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*.

Note: Keep in mind that the class you wish to attach to your node should be named as the `.cs` file. If not, you will get the following error and won't be able to run the scene: `Cannot find class XXX for script res://XXX.cs`.

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)
- Writing editor plugins and tool scripts in C# is not yet supported
- Exporting a project may not yet work (#15615)
- Signals with parameters are broken in 3.0.2-stable (#17553)

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.

Using Nuget Packages in Godot

Nuget Packages can be installed and used with Godot, as with any project. Many IDEs (such as vs code) can add packages directly. They can also be added manually by adding the package reference in the .csproj file located in the project root:

```
    <ItemGroup>
        <PackageReference Include="Newtonsoft.Json" Version="11.0.2"/>
    </ItemGroup>
    ...
</Project>
```

Then using the dotnet command line to restore package, in the project root:

```
$ dotnet restore
```

4.3.2 Features

This page provides 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 `InvalidCastException` 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.

C# Signals

For a complete C# example, see the **Handling a signal** section in the step by step *Scripting* tutorial.

Declaring a signal in C# is done with the `[Signal]` attribute on a delegate.

```
[Signal]
delegate void MySignal();

[Signal]
delegate void MySignalWithArguments(string foo, int bar);
```

These signals can then be connected either in the editor or from code with `Connect`.

```
public void MyCallback()
{
    GD.Print("My callback!");
}

public void MyCallbackWithArguments(string foo, int bar)
{
    GD.Print("My callback with: ", foo, " and ", bar, "!");
}

public void SomeFunction()
{
    instance.Connect("MySignal", this, "MyCallback");
    instance.Connect(nameof(MySignalWithArguments), this, "MyCallbackWithArguments");
}
```

Emitting signals is done with the `EmitSignal` method.

```
public void SomeFunction()
{
    EmitSignal(nameof(MySignal));
    EmitSignal("MySignalWithArguments", "hello there", 28);
}
```

Notice that you can always reference a signal name with the `nameof` keyword (applied on the delegate itself).

It is possible to bind values when establishing a connection by passing an object array.

```
public int Value { get; private set; } = 0;

private void ModifyValue(int modifier)
{
    Value += modifier;
}

public void SomeFunction()
{
    var plusButton = (Button)GetNode("PlusButton");
    var minusButton = (Button)GetNode("MinusButton");

    plusButton.Connect("pressed", this, "ModifyValue", new object[] { 1 });
    minusButton.Connect("pressed", this, "ModifyValue", new object[] { -1 });
}
```

Signals support parameters and bound values of all the built-in types and Classes derived from *Godot.Object*. Consequently any `Node` or `Reference` will be compatible automatically but custom data objects will need to extend from *Godot.Object* or one of its subclasses.

```
public class DataObject : Godot.Object
{
    public string Field1 { get; set; }
    public string Field2 { get; set; }
}
```

Finally, signals can be created by calling `AddUserSignal`, but be aware that it should be executed before any use of said signals (with `Connect` or `EmitSignal`).

```
public void SomeFunction()
{
    AddUserSignal("MyOtherSignal");
    EmitSignal("MyOtherSignal");
}
```

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.RandRange` instead of `rand_range`.

Export keyword

Use the `[Export]` attribute instead of the GDScript `export` keyword.

Signal keyword

Use the `[Signal]` attribute instead of the GDScript `signal` keyword. This attribute should be used on a *delegate*, whose name signature will be used to define the signal.

```
[Signal]
delegate void MySignal(string willSendsAString);
```

See also: *C# Signals*

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

4.3.4 Style Guide

Having well-defined and consistent coding conventions is important for every project, and Godot is no exception to this rule.

This page contains a coding style guide which is followed by developers and contributors of Godot itself. As such, it is mainly intended for those who want to contribute to the project, but since the conventions and guidelines mentioned in this article are those most widely adopted by the users of the language, we encourage you to do the same, especially if you do not have such a guide yet.

Note: This article is by no means an exhaustive guide on how to follow the standard coding conventions or best practices. If you feel unsure of an aspect which is not covered here, please refer to more comprehensive documentation, such as C# Coding Conventions or Framework Design Guidelines.

Language Specification

Currently, Godot uses C# version 6.0 in its engine and example source code. So, before we move to a newer version, care must be taken to avoid mixing language features only available in C# 7.0 or later, such as pattern matching or expression-bodied members inside get/set accessors.

For detailed information of C# features in different versions, please see What's New in C#.

Formatting Conventions

- If you create a new file, make sure that it uses linefeed (*LF*) characters to break lines, not *CRLF* or *CR*.
- Use UTF-8 encoding without a byte order mark (BOM <https://en.wikipedia.org/wiki/Byte_order_mark>).
- Use 4 spaces instead of tabs for indentation (which is referred to as 'soft tabs').

Line Breaks and Blank Lines

For a general indentation rule, follow The 'Allman Style' which recommends placing the brace associated with a control statement on the next line, indented to the same level:

```
// Use this style:
if (x > 0)
{
    DoSomething();
}

// NOT this:
if (x > 0) {
    DoSomething();
}
```

However, you may choose to omit line breaks inside brackets,

- For simple property accessors.
- For simple object, array, or collection initializers.
- For abstract auto property, indexer, or event declarations.

```
// You may put the brackets in a single line in following cases:
public interface MyInterface
{
    int MyProperty { get; set; }
}

public class MyClass : ParentClass
{
    public int Value
    {
        get { return 0; }
        set
        {
            ArrayValue = new [] {value};
        }
    }
}
```

Insert a blank line,

- After *using* statement list.
- Between method, properties, and inner type declarations.

Field and constant declarations can be grouped together according to relevance. In that case, consider inserting a blank line between the groups for easier reading.

Avoid inserting a blank line,

- After an opening bracket (‘{‘).
- Before a closing bracket (‘}’).
- After a comment block, or a single line comment.
- Adjacent to another blank line.

```
using System;
using Godot;
                                                    // Blank line after using list.
public class MyClass
{                                                   // No blank line after '{'.
```

(continues on next page)

(continued from previous page)

```
    public enum MyEnum
    {
        Value,
        AnotherValue                            // No blank line before '}'.
    }
                                                // Blank line around inner types.
    public const int SomeConstant = 1;
    public const int AnotherConstant = 2;

    private Vector3 _x;
    private Vector3 _y;                         // Related constants or fields can be
                                                // grouped together.
    private float _width;
    private float _height;

    public int MyProperty { get; set; }
                                                // Blank line around properties.
    public void MyMethod()
    {
        // Some comment.
        AnotherMethod();                        // No blank line after a comment.
    }
                                                // Blank line around methods.
    public void AnotherMethod()
    {
    }
}
```

Consider breaking a line when it's longer than 100 characters. And it's also a good practice to insert a line feed (LF) character at the end of a file because some utilities have trouble recognizing the last line without it (i.e. Linux's *cat* command).

Using Spaces

Insert a space,

- Around a binary and tertiary operator.
- Between an opening parenthesis and *if*, *for*, *foreach*, *catch*, *while*, *lock* or *using* keywords.
- Before and within a single line accessor block.
- Between accessors in a single line accessor block.
- After a comma.
- After a semi-colon in a *for* statement.
- After a colon in a single line *case* statement.
- Around a colon in a type declaration.
- Around a lambda arrow.
- After a single line comment symbol ('//'), and before it if used at the end of a line.

Do not use a space,

- After a type cast parentheses.
- Within single line initializer braces.

The following example shows a proper use of spaces, according to some of the the above mentioned conventions:

```
public class MyClass<A, B> : Parent<A, B>
{
    public float MyProperty { get; set; }

    public float AnotherProperty
    {
        get { return MyProperty; }
    }

    public void MyMethod()
    {
        int[] values = {1, 2, 3, 4}; // No space within initializer brackets.
        int sum = 0;

        // Single line comment.
        for (int i = 0; i < values.Length; i++)
        {
            switch (i)
            {
                case 3: return;
                default:
                    sum += i > 2 ? 0 : 1;
                    break;
            }
        }

        i += (int)MyProperty; // No space after a type cast.
    }
}
```

Naming Conventions

Use *PascalCase* for all namespaces, type names and member level identifiers (i.e. methods, properties, constants, events), except for private fields:

```
namespace ExampleProject
{
    public class PlayerCharacter
    {
        public const float DefaultSpeed = 10f;

        public float CurrentSpeed { get; set; }

        protected int HitPoints;

        private void CalculateWeaponDamage()
        {
        }
    }
}
```

Use *camelCase* for all other identifiers (i.e. local variables, method arguments), and use underscore('_') as a prefix for private fields (but not for methods or properties, as explained above):

```
private Vector3 _aimingAt; // Use '_' prefix for private fields.

private void Attack(float attackStrength)
{
    Enemy targetFound = FindTarget(_aimingAt);

    targetFound?.Hit(attackStrength);
}
```

There's an exception with acronyms which consist of two letters like *'UI'* which should be written in upper case letters when used where Pascal case would be expected, and in lower case letters otherwise.

Note that *'id'* is **not** an acronym, so it should be treated as a normal identifier:

```
public string Id { get; }

public UIManager UI
{
    get { return uiManager; }
}
```

It is generally discouraged to use a type name as a prefix of an identifier like *'string strText'* or *'float fPower'*, for example. However, there's an exception about interfaces, in which case they **should** be named using an upper case *'I'* as a prefix, like *'IInventoryHolder'* or *'IDamageable'*.

Lastly, consider choosing descriptive names and do not try to shorten them too much if it affects readability.

For instance, if you want to write a code to find a nearby enemy and hit with an weapon, prefer

```
FindNearbyEnemy()?.Damage(weaponDamage);
```

Rather than,

```
FindNode()?.Change(wpnDmg);
```

Implicitly Typed Local Variables

Consider using implicitly typing (*'var'*) for declaration of a local variable, but do so **only when the type is evident** from the right side of the assignment:

```
// You can use `var` for these cases:

var direction = new Vector2(1, 0);

var value = (int)speed;

var text = "Some value";

for (var i = 0; i < 10; i++)
{
}

// But not for these:

var value = GetValue();
```

(continues on next page)

(continued from previous page)

```
var velocity = direction * 1.5;

// It's generally a better idea to use explicit typing for numeric values, especially␣
↪with
// the existence of 'real_t' alias in Godot, which can either be double or float␣
↪depending
// on the build configuration.

var value = 1.5;
```

Other Considerations

- Use explicit access modifiers.
- Use properties instead of non-private fields.
- Use modifiers in this order: *'public/protected/private/internal virtual/override/abstract/new static readonly'*.
- Avoid using fully qualified names or *'this.'* prefix for members when it's not necessary.
- Remove unused *'using'* statements and unnecessary parentheses.
- Consider omitting default initial value for a type.
- Consider using null-conditional operators or type initializers to make the code more compact.
- Use safe cast when there is a possibility of the value being a different type, and use direct cast otherwise.

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
```

(continues on next page)

(continued from previous page)

```
/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 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+

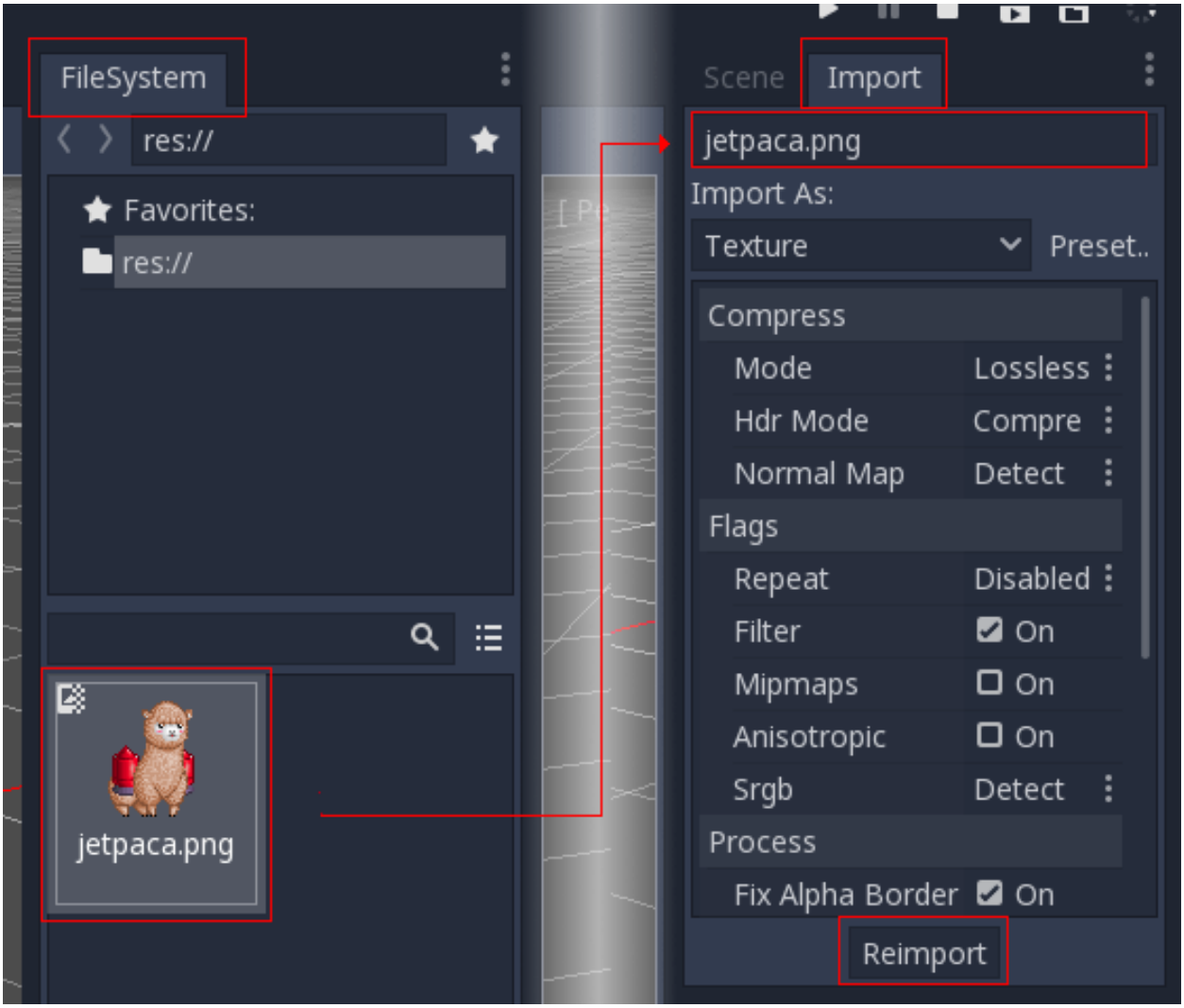
Previously, importing assets in Godot 2.x required manual maintenance of a separate directory with source assets. Without doing this, it was impossible to specify how to convert and change import flags for textures, audios, scenes, etc.

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. Select the relevant resource in the filesystem dock:



Then, after adjusting the parameters, press "Reimport". The parameters used will be only for this asset and will be used on future reimports.

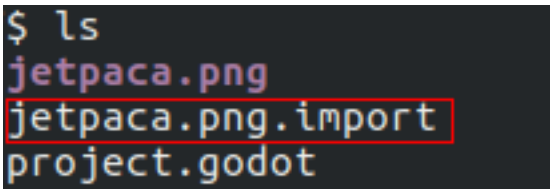
Changing import parameters of several assets at the same time is also possible. Simply select all of them together in the resources dock and the exposed parameters will apply to all of them when reimporting.

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!



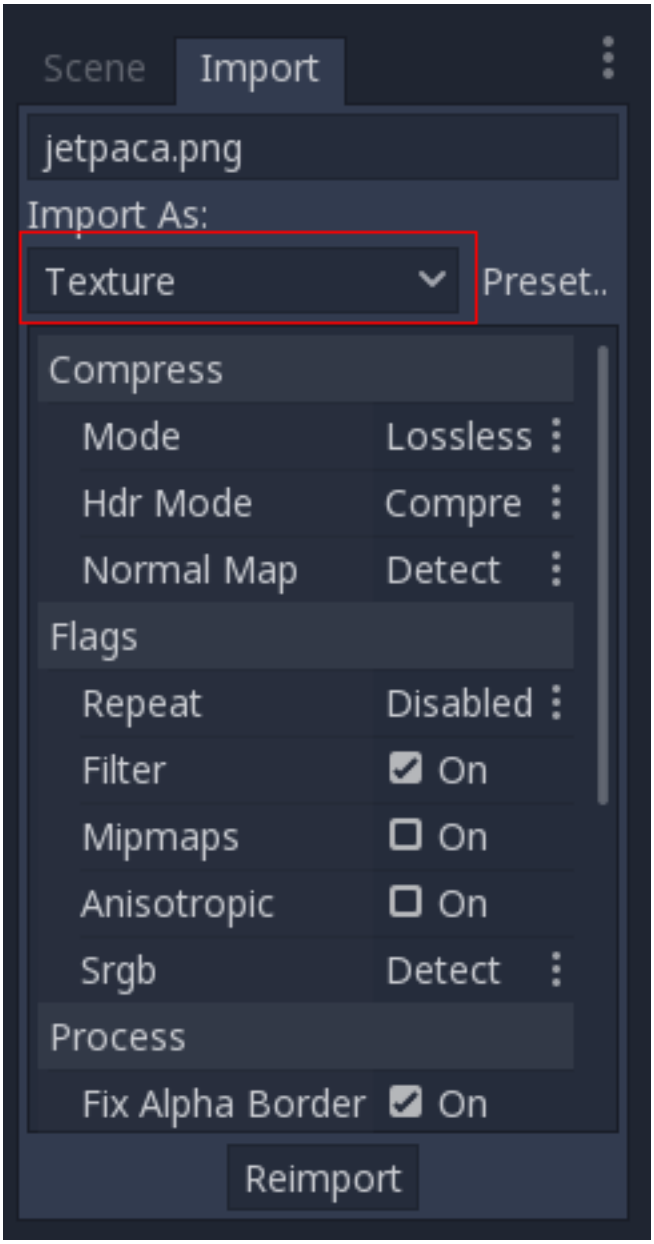
Additionally, extra assets will be preset in the hidden res://.import folder:

```
$ 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!

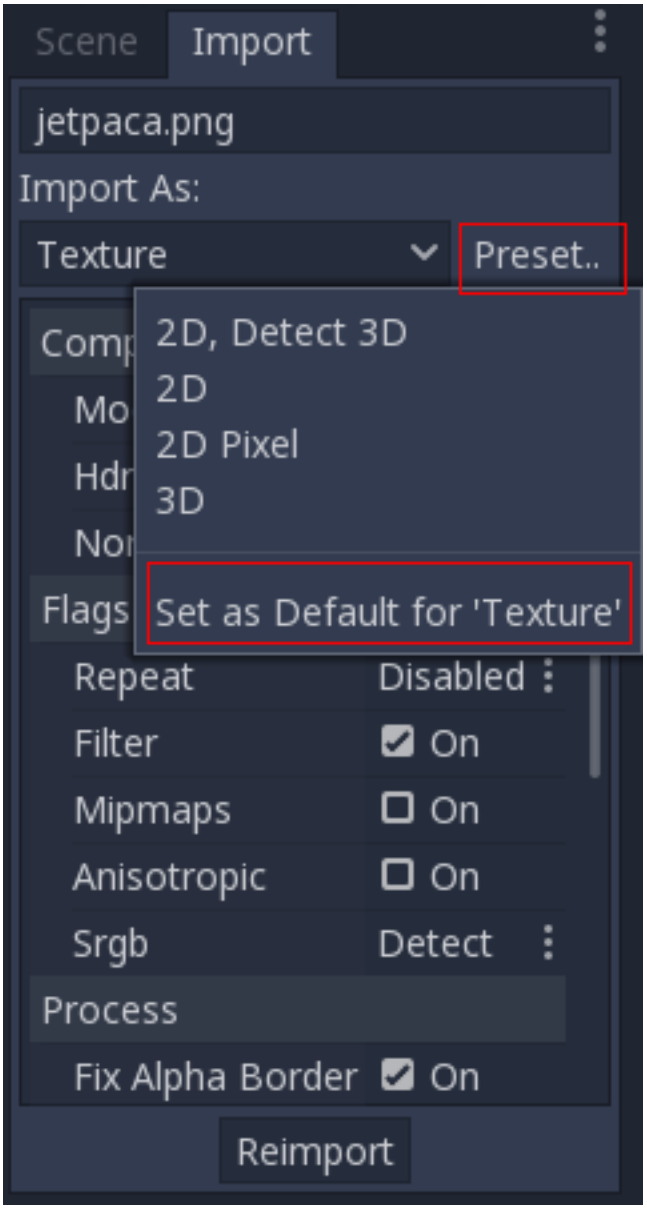
Changing import resource type

Some source assets can be imported as different types of resources. For this, 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?

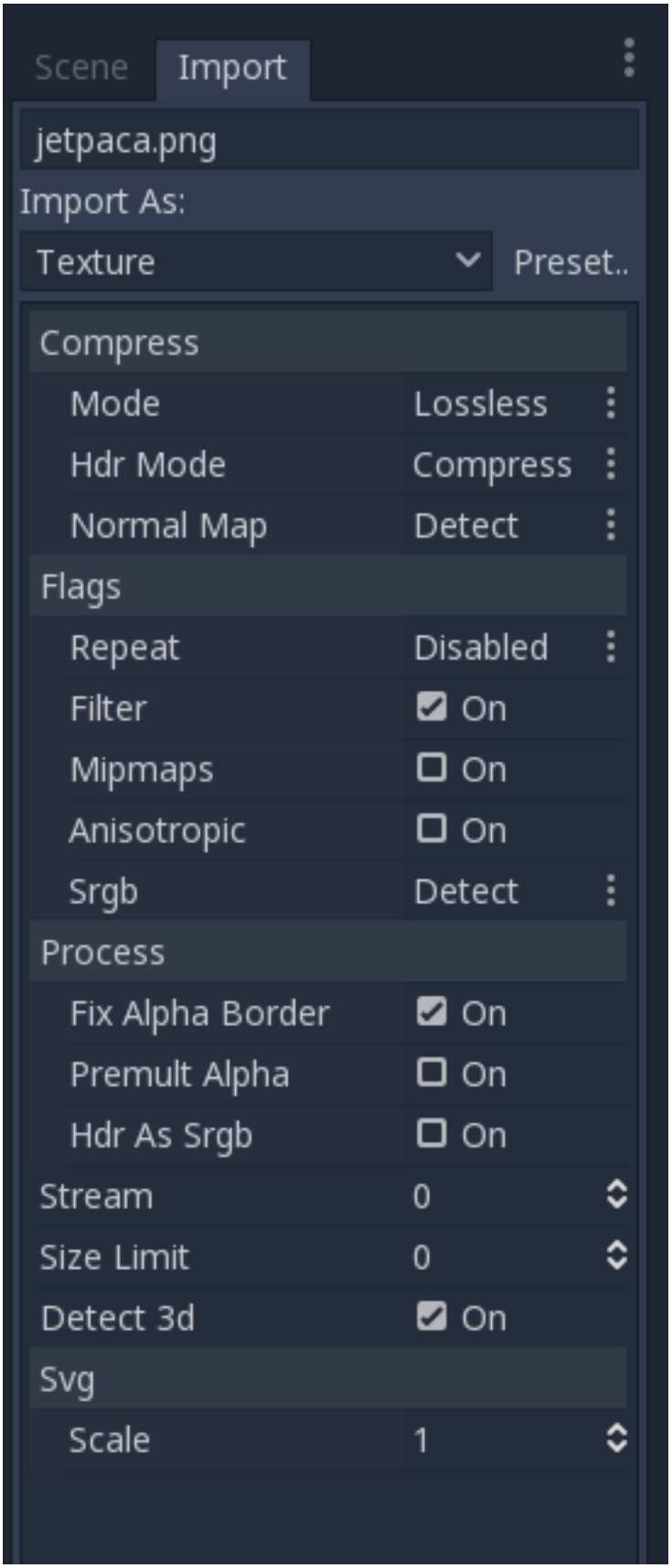
In Godot 3+, image files are no longer native resources and they must be imported. The reason behind this is the large amount of configuration parameters that image files can be imported with.

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:


Scene
Import
jetpaca.png
Import As:
Texture
Preset..
Compress
Mode
Lossless
Hdr Mode
Compress
Normal Map
Detect
Flags
Repeat
Disabled
Filter
On
Mipmaps
On
Anisotropic
On
Srgb
Detect
Process
Fix Alpha Border
On
Premult Alpha
On
Hdr As Srgb
On
Stream
0
Size Limit
0
Detect 3d
On
Svg
Scale
1

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):

	Uncompressed	Compress Lossless (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 equirectancular 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 mosty 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 blurryness.

SRGB

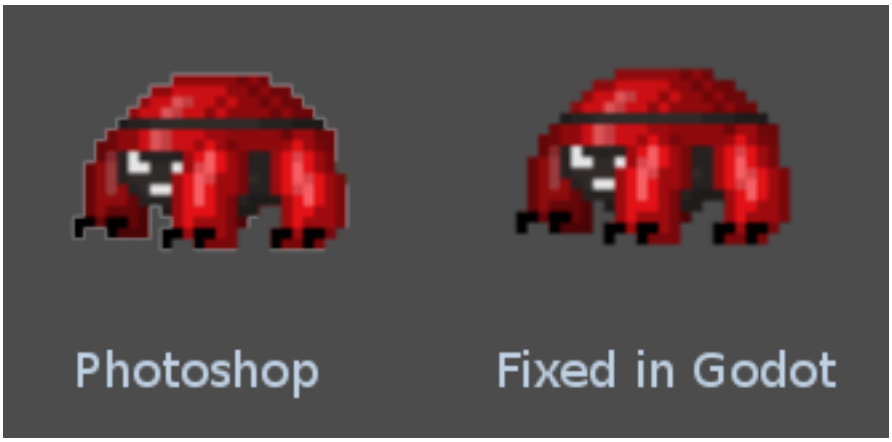
Godot uses Linear colorspace when rendering 3D. Textures mapped to albedo or detail channels need to have this option turned on in order for colors to look correct. When set to "Detect" mode, the texture will be marked as SRGB when used in albedo channels.

Process

Some special processes can be applied to images when importe 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

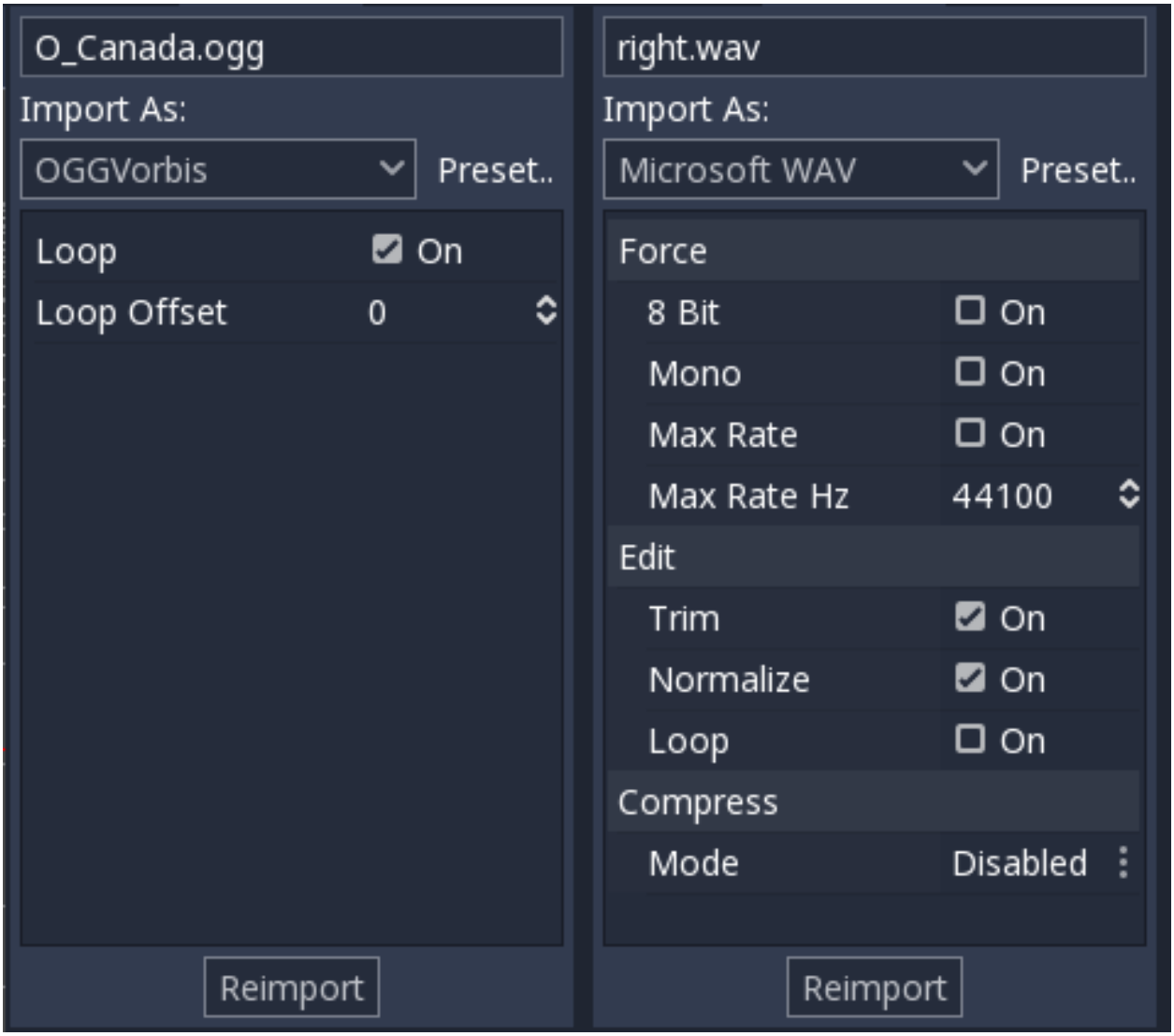
This option makes Godot be aware of when a texture (which is imported for 2D as default) is used in 3D. If this happens, setting are changed so the texture flags are friendlier to 3D (mipmaps, filter and repeat become enabled and compression is changed to VRAM). Texture is also reimported automatically.

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.



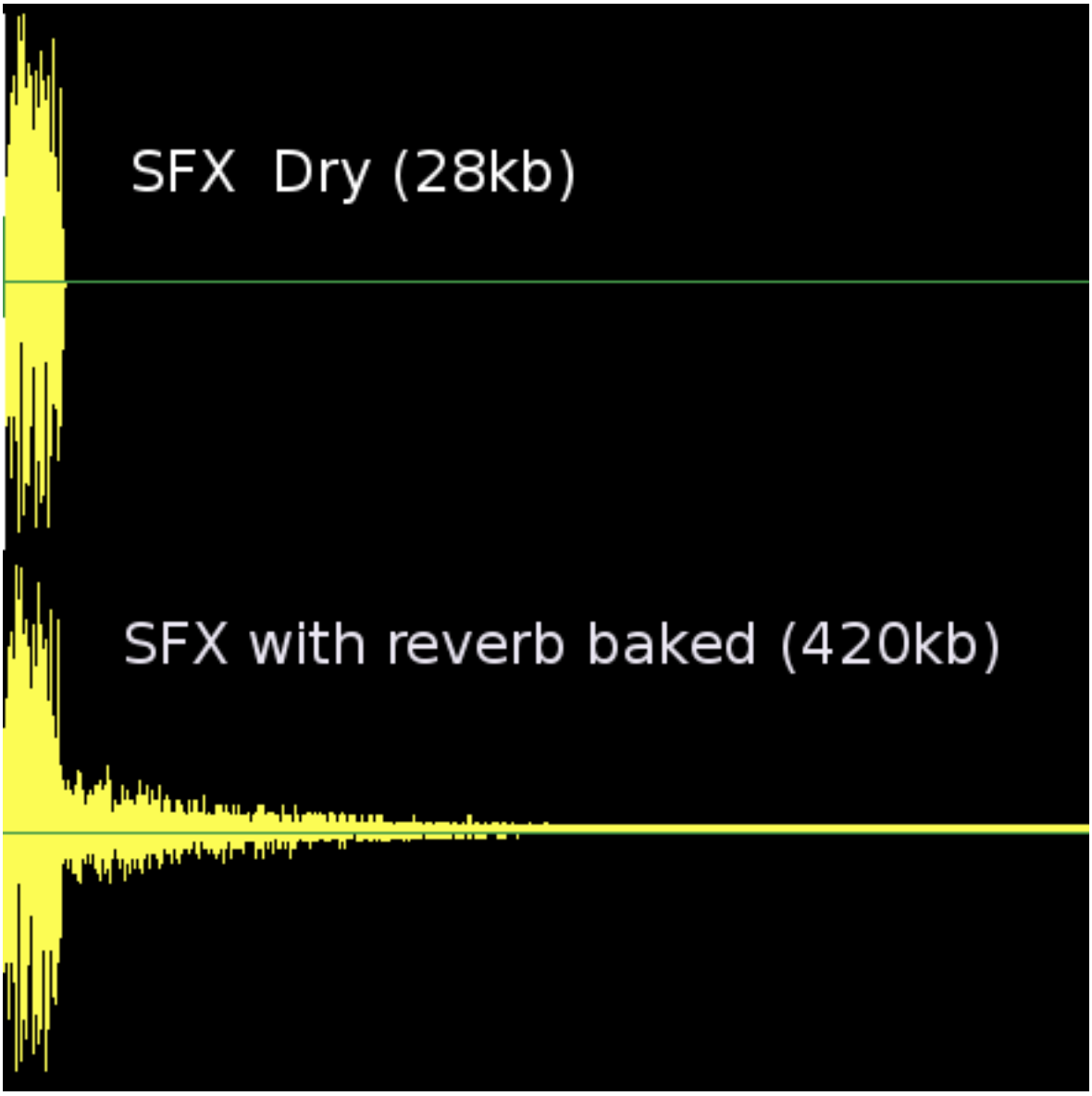
Here is a comparative chart.

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 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 fomats*:

- 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 well, although they are not always up-to date with the latest version of the software.

Exporting DAE files from Blender

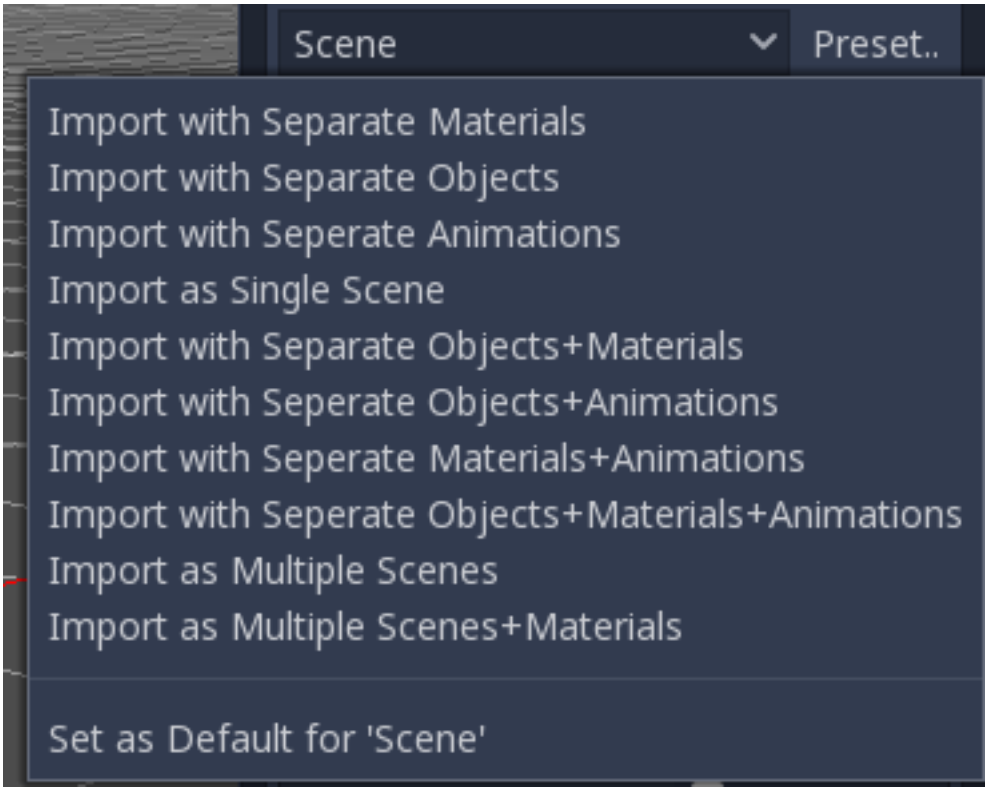
Blender has built-in collada support too, but it's also broken and should not be used.

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:

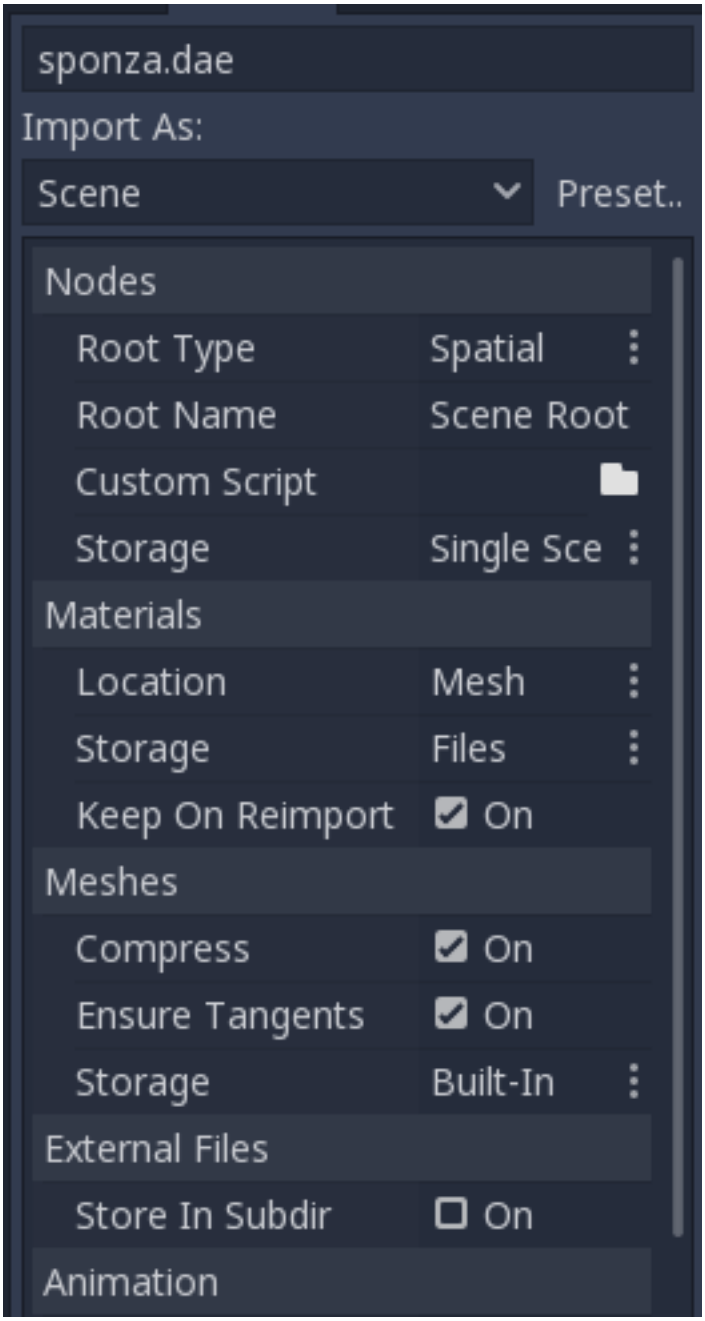
- External materials (default): Where each material is saved to a file resource. Modifications to them are kept.
- External meshes: Where each mesh is saved to a different file. Many users prefer to deal with meshes directly.
- External animations: Allowing saved animations to be modified and merged 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 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 vertices 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.

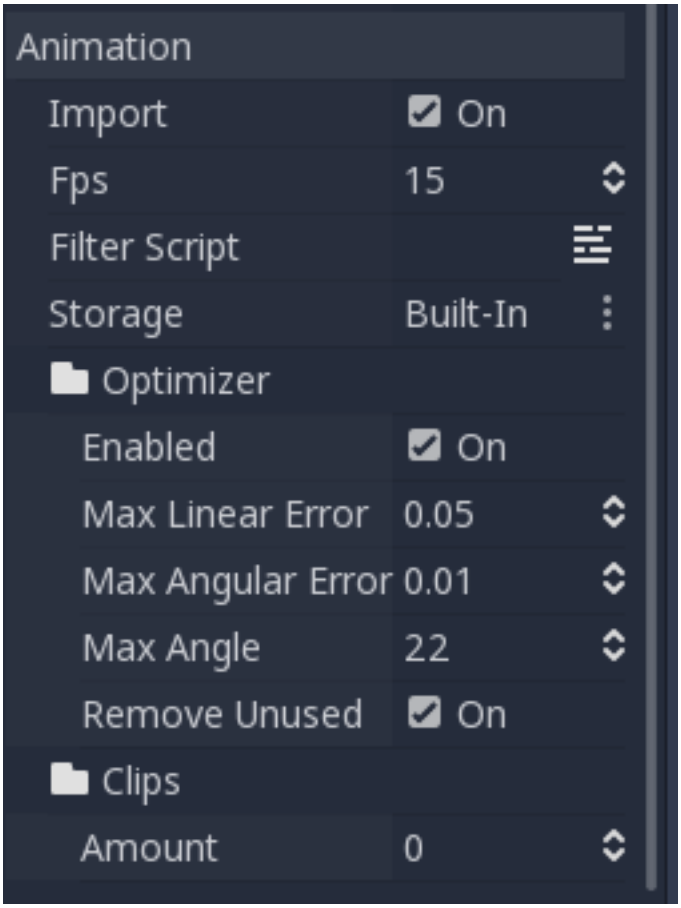
This option is provided to help those who prefer working directly with meshes instead of scenes.

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.

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.

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)

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.

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.

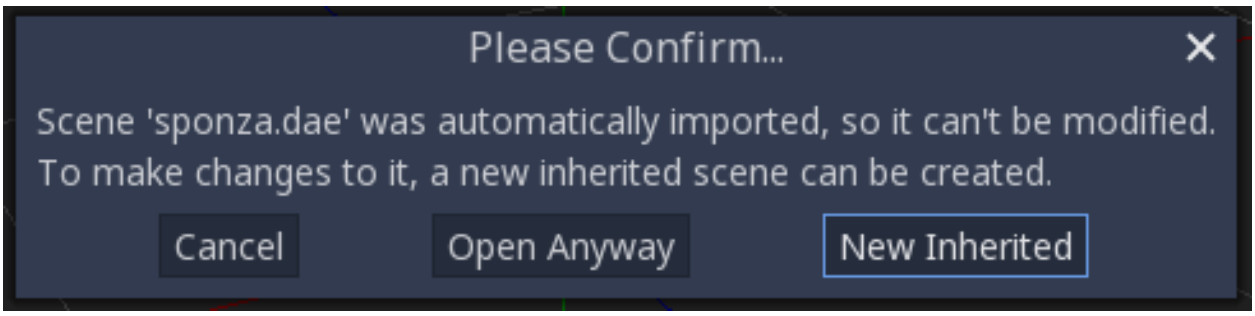
Clips

It is possible to specify multiple animations from a single timeline as clips. 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 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*. 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)

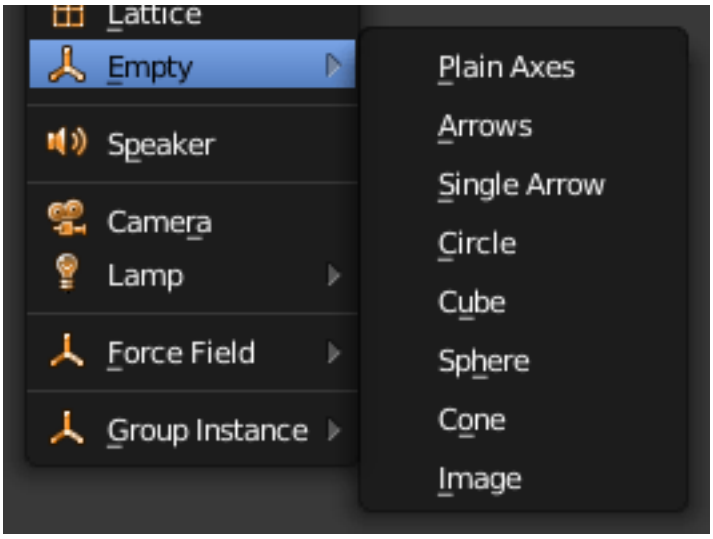
Option "-col" will work only for Mesh nodes. If it is detected, a child static collision node will be added, using the same geometry as the mesh.

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 navigatopm (-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. 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 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.

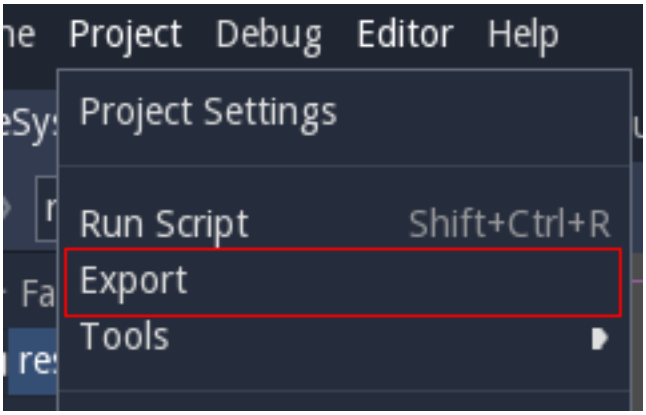
This can be troublesome because it means that the developer must be familiarized with the SDK of each platform before even being able to export. While learning each SDK is always encouraged, it can be frustrating to be forced to do it at an undesired time.

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).

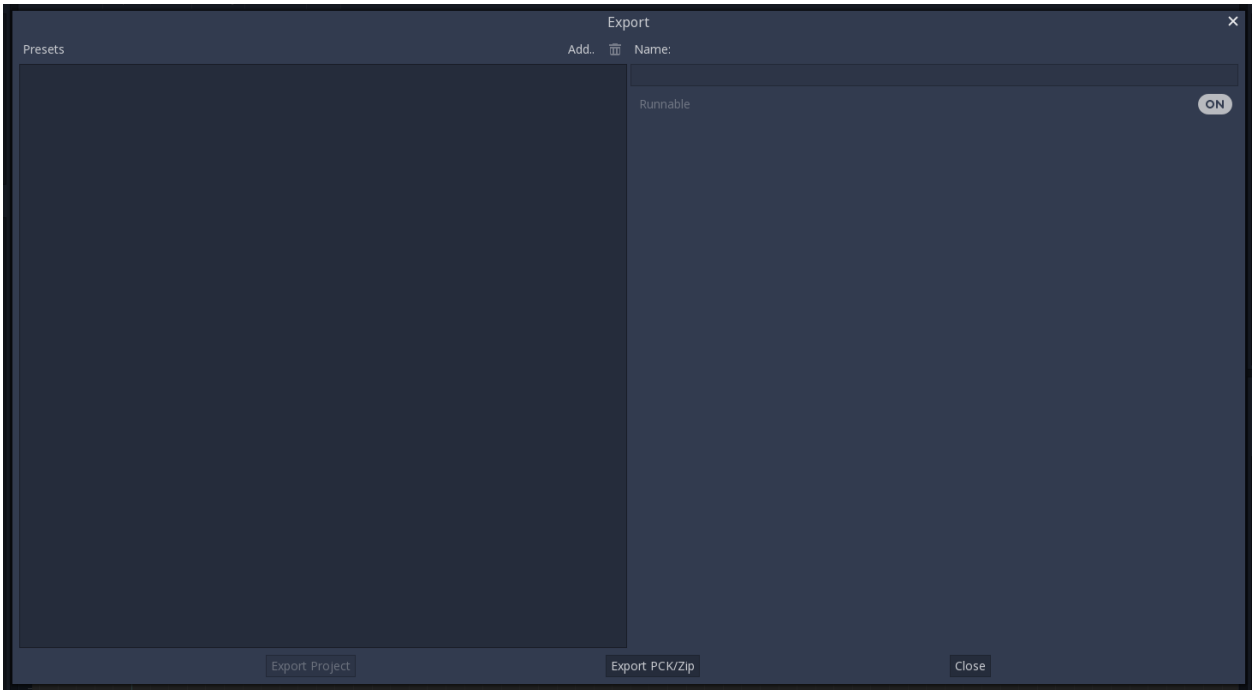
Export menu

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.

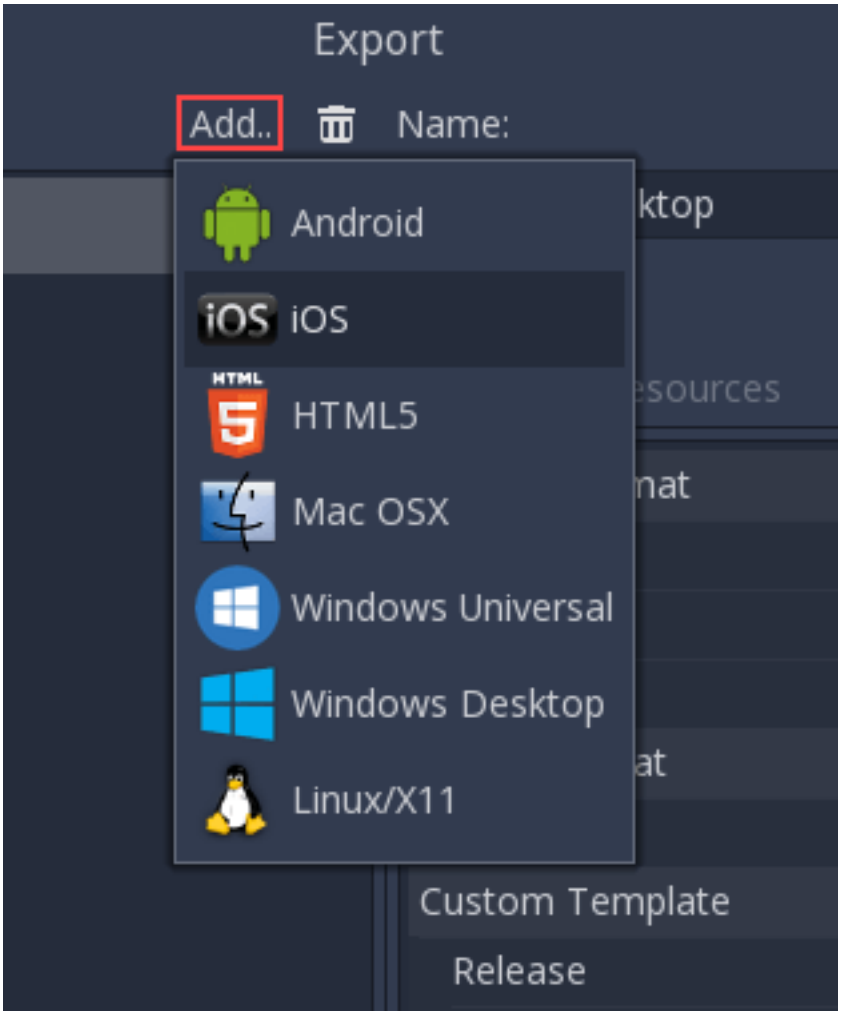
To open the export menu, click the "Export" button:



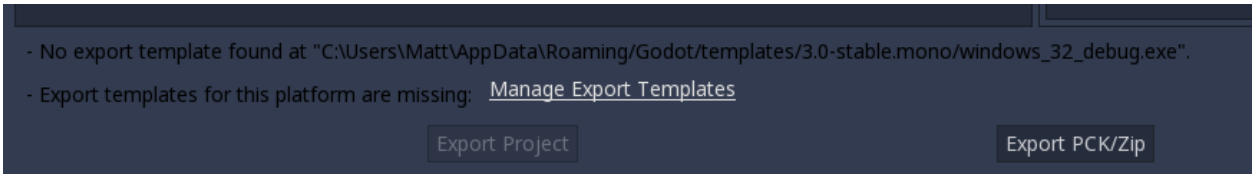
The export menu will open, however it will be completely empty.



That is because we need to add an export preset. To do that click the *Add..* button at the top of the export menu. This will open a drop down list of platforms to choose from for an export preset.



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 menu will complain when something is missing and will not allow the user to export for that platform until they resolve it:

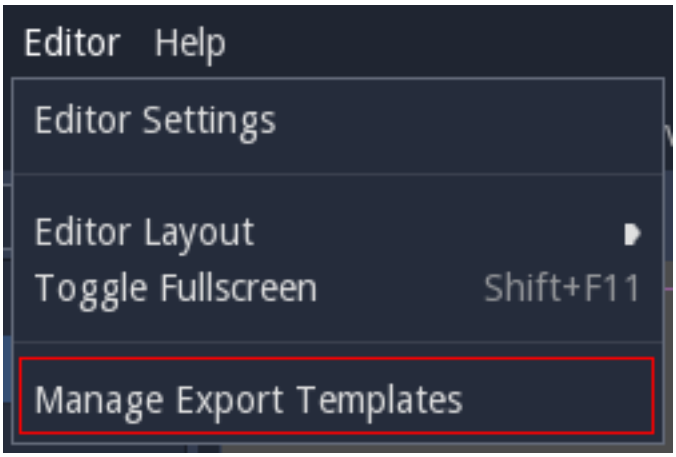


At that time, the user is expected to come back to the documentation and follow instructions on how to properly set up that platform.

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.

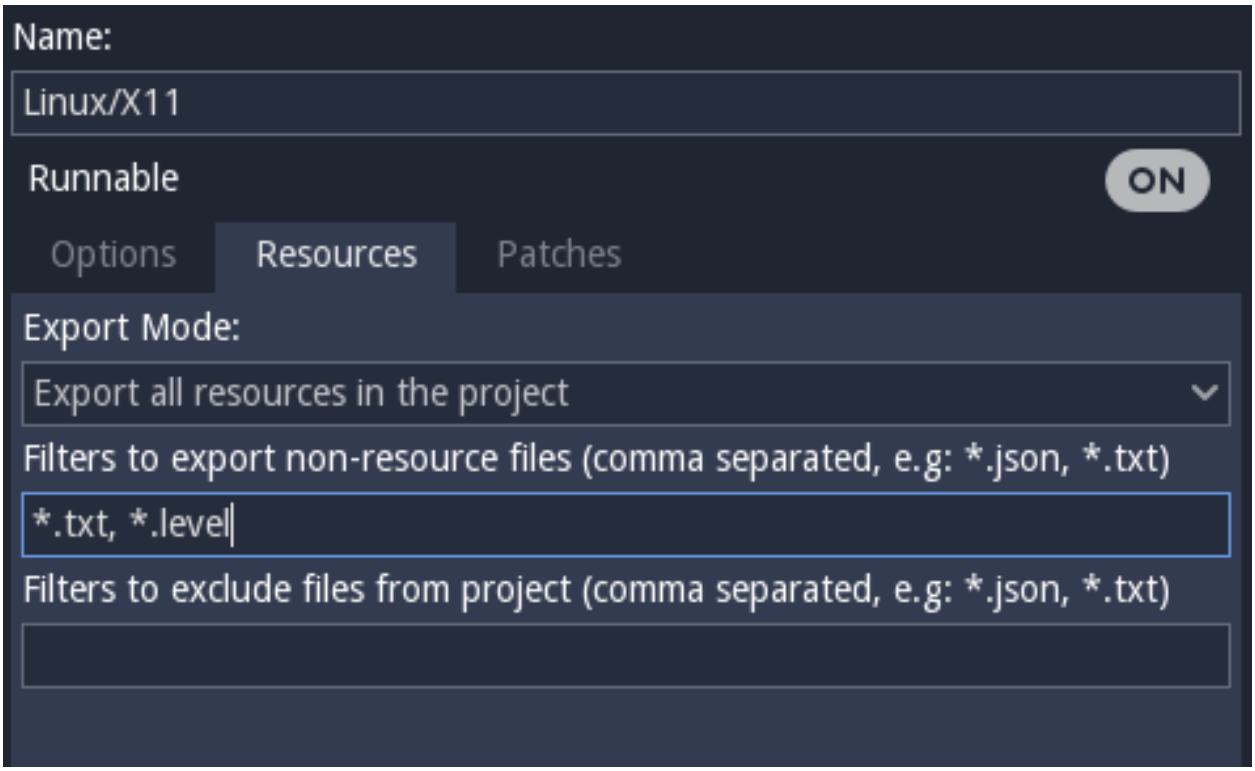
Once downloaded, they can be installed using the "Install Export Templates" option in the editor:



Export mode

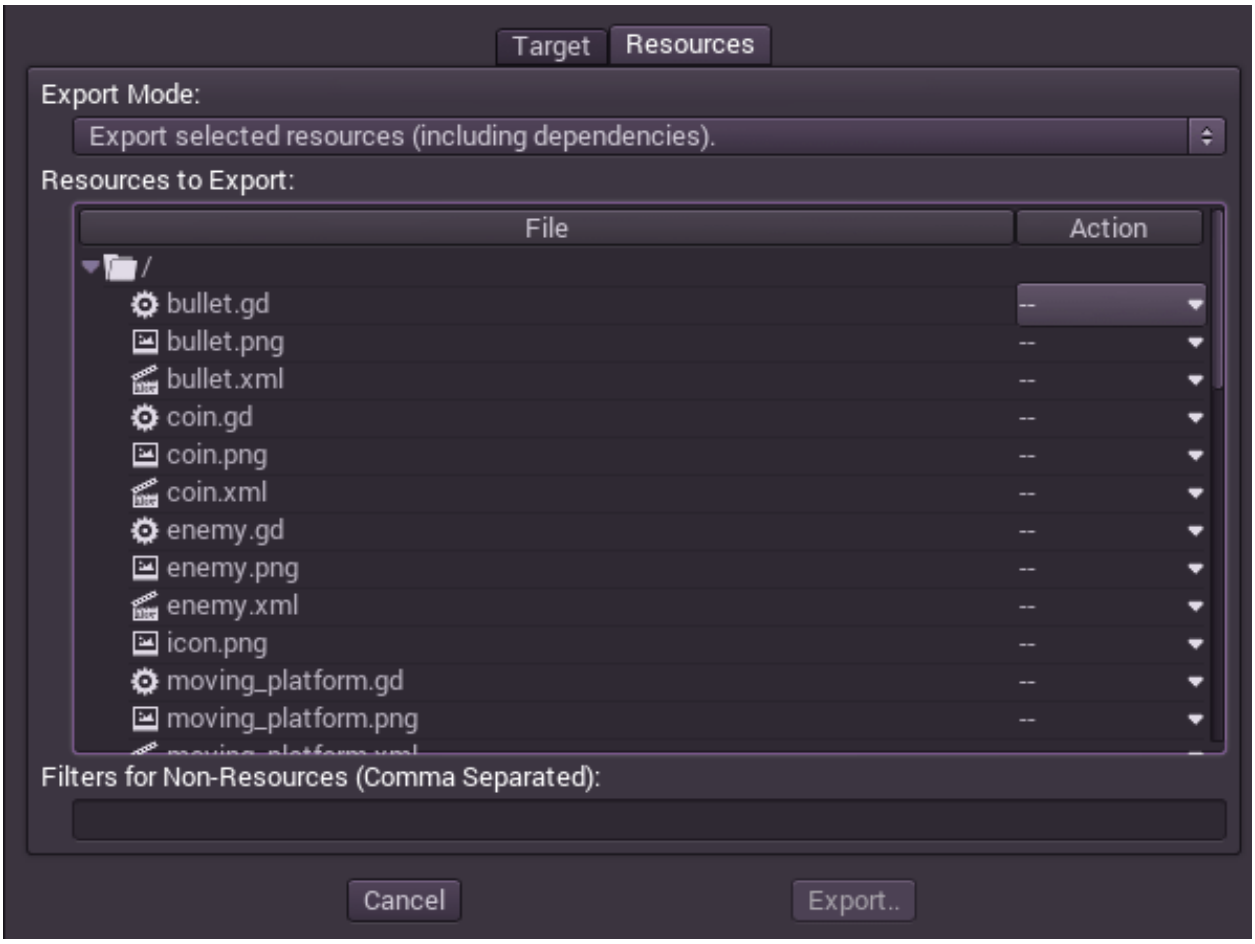
When exporting, Godot makes a list of all the files to export and then creates the package. There are 3 different modes for exporting:

- 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.



Export from Command Line

In production it is useful to automate builds, and Godot supports this with the `--export` and `--export-debug` command line parameters. Exporting from command line still requires an export template to define the export parameters. A basic invocation of the export would be `godot --export "Windows Desktop" some_name`

Which, assuming there is a preset called "Windows Desktop" and the template can be found, will export to `some_name.exe`. The output path is relative to the project path or absolute. It does not respect the directory the command was invoked from.

You can also configure it to export only the .pck or .zip file (allowing a single export to be used with multiple Godot executables). This takes place if:

- The export preset is not marked as runnable
- The target name ends with *.pck* or with *.zip*

It is often useful to combine the `--export` flag with the `--path` flag, and to create a dedicated export template for automated export: `godot --path path/to/project --export "pck" game_name.pck`

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)
```

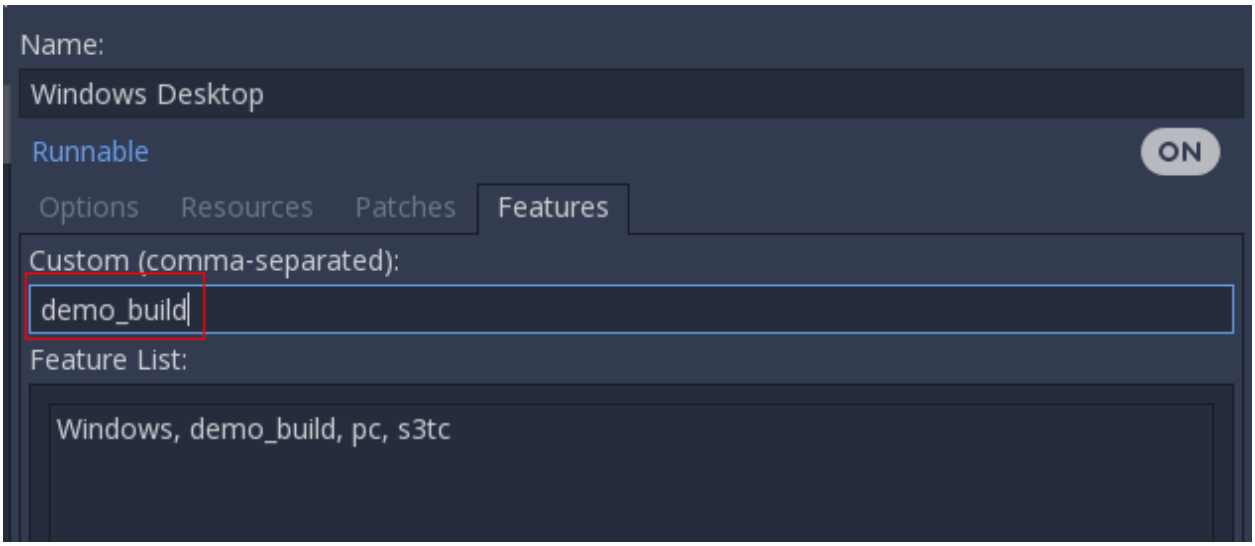
Default features

Here is a list of most feature tags in Godot. Keep in mind they are *case sensitive*:

Feature Tag	Description
Android	Running on Android
HTML5	Running on HTML5
JavaScript	*JavaScript singleton* is available
OSX	Running on macOS
iOS	Running on iOS
UWP	Running on UWP
Windows	Running on Windows
X11	Running on X11
debug	Running on a debug build
release	Running on a release build
32	Running on a 32-bit build
64	Running on a 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
pvrtc	Textures using PVRTC compression are supported

Custom features

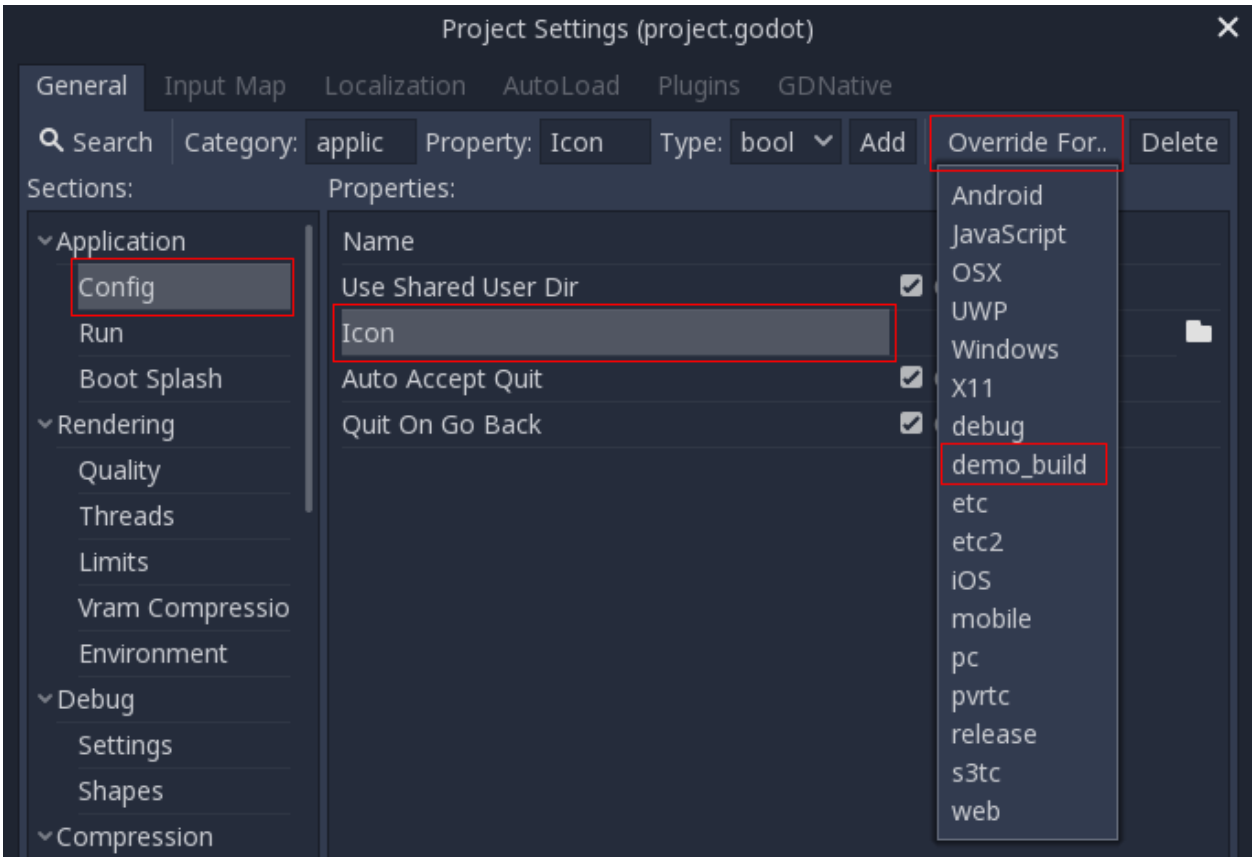
It is possible to add custom features to a build, use the relevant field in the *export preset* used to generate it:



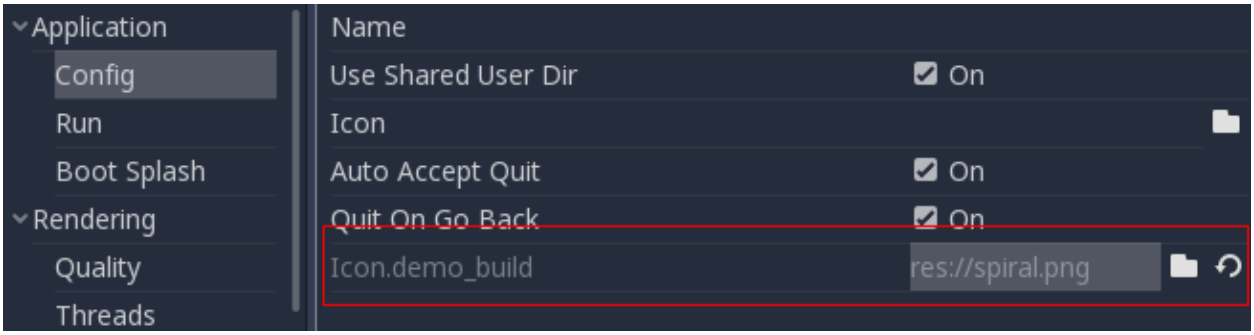
Overriding project settings

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).

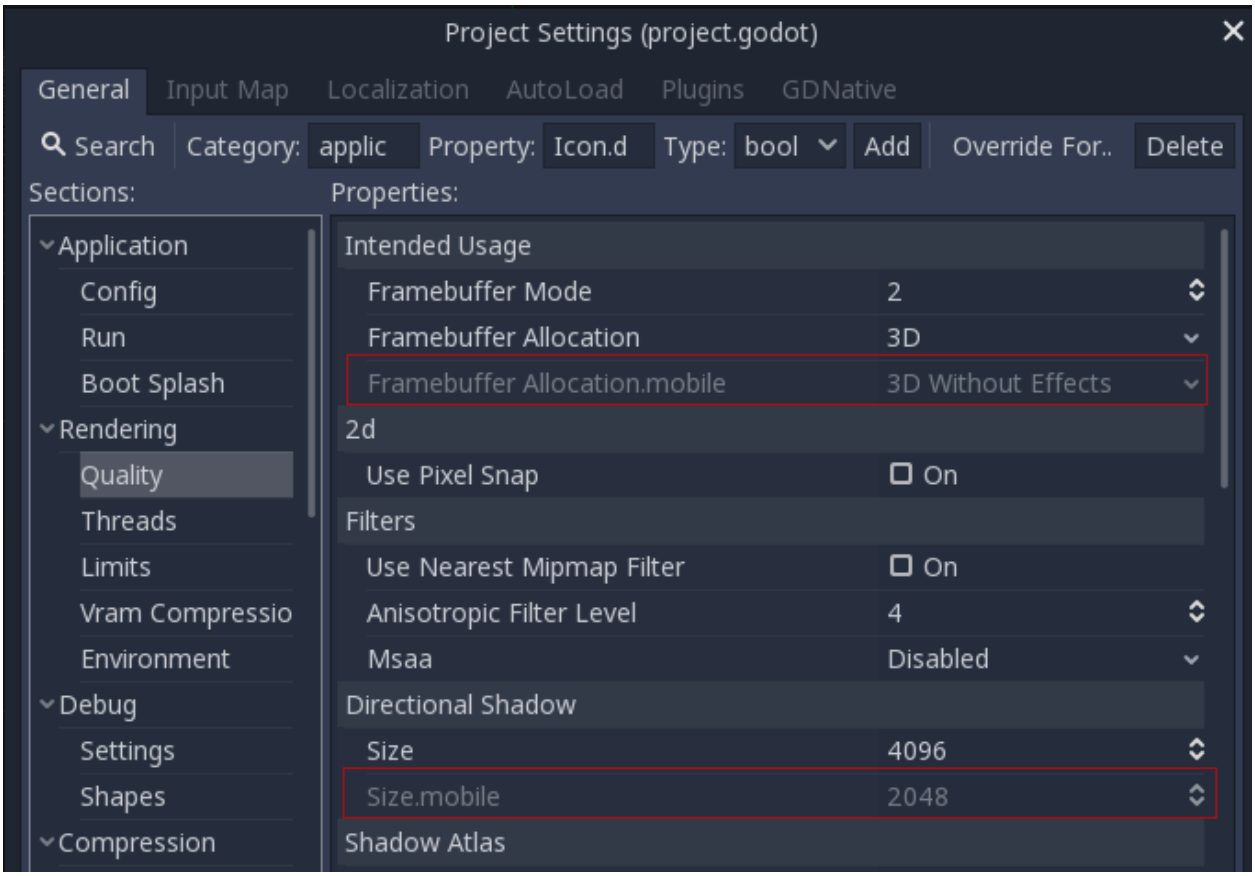


After overriding, a new field is added for this specific configuration:



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 works properly.

5.3.4 Exporting for iOS

These are the steps to load a Godot project in Xcode. This allows you to build and deploy to an iOS device, build a release for the App Store, and do everything else you can normally do with Xcode.

Requirements

- You must export for iOS from a computer running macOS with Xcode installed.
- Download the Godot export templates. Use the Godot menu: Editor > Manage Export Templates

Export a Godot project to Xcode

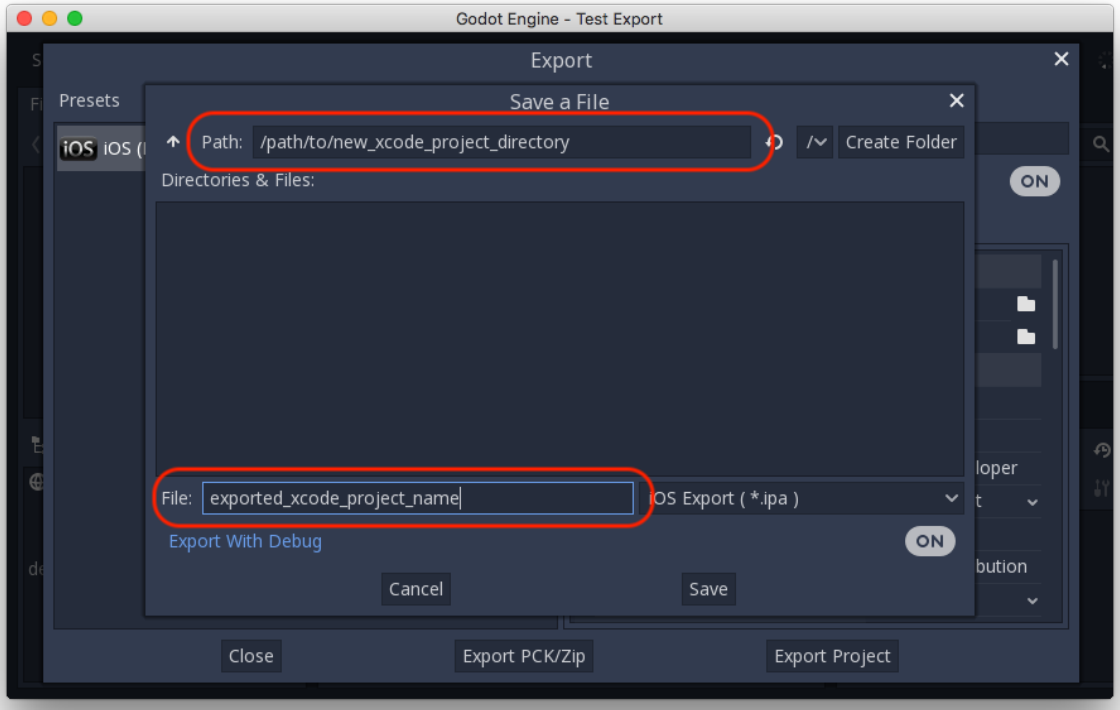
In the Godot editor, open the **Export** window from the **Project** menu. When the Export window opens, click **Add..** and select **iOS**.

The following export options are required. Leaving any blank with cause the exporter to throw an error:

- In the **Application** category * **App Store Team ID**
- Everything in the **Required Icons** category
- Everything in the **Landscape Launch Screens** category
- Everything in the **Portrait Launch Screens** category

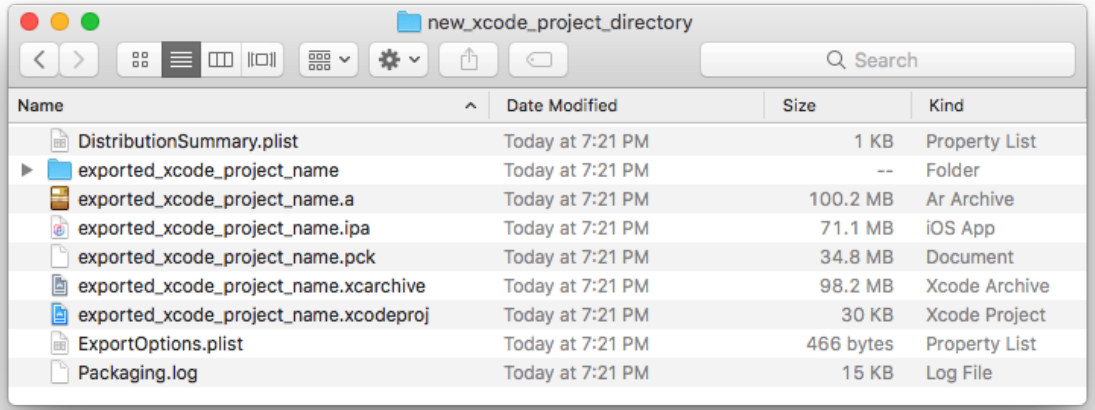
After you click **Export Project**, there are still two important options left:

- **Path** is an empty folder that will contain the exported Xcode project files.
- **File** will be the name of the Xcode project and several project specific files and directories.



Note: This tutorial uses **exported_xcode_project_name**, but you will use your project's name. When you see **exported_xcode_project_name** in the following steps, replace it with the name you used instead.

When the export completes, the output folder should look like this:



Opening **exported_xcode_project_name.xcodeproj** lets you build and deploy like any other iOS app.

Active development considerations

The above method creates an exported project that you can build for release, but you have to re-export every time you make a change in Godot.

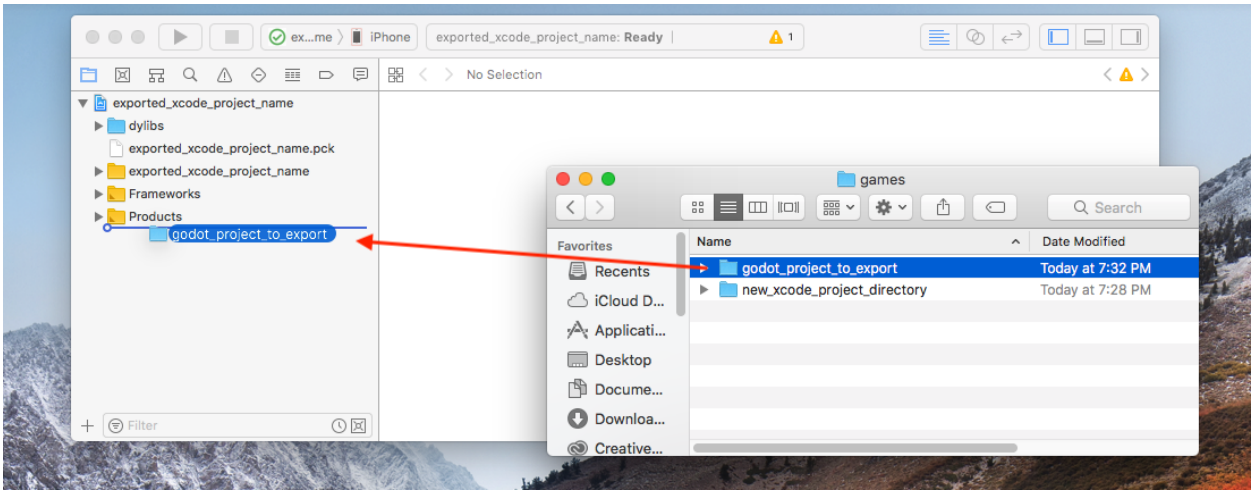
While developing, you can speed this process up by linking your Godot project files directly into your app.

In the following example:

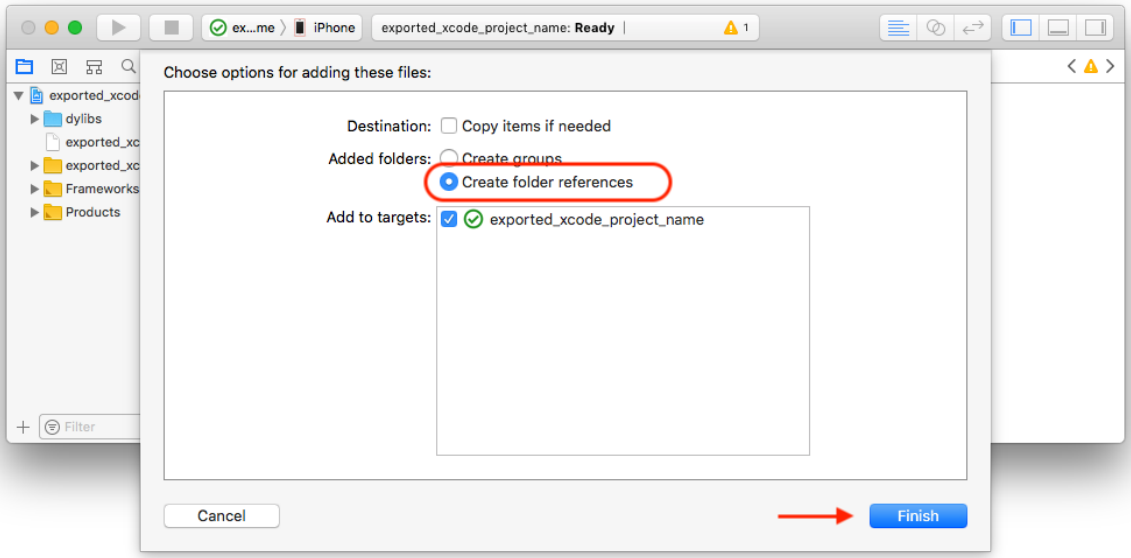
- **exported_xcode_project_name** is the name of the exported iOS application (as above).
- **godot_project_to_export** is the name of the Godot project.

Steps to link an Godot project folder to Xcode

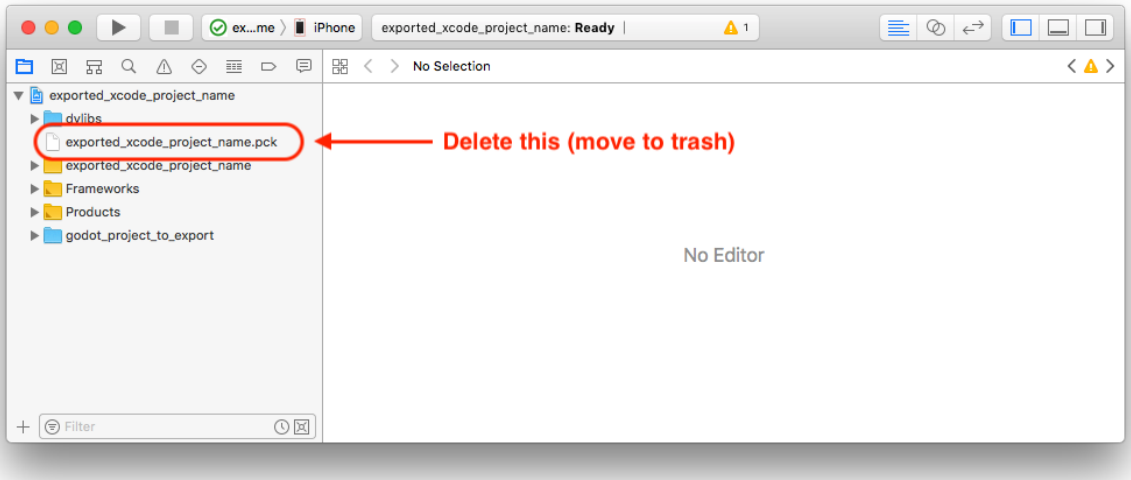
1. Start from an exported iOS project (follow the steps above).
2. In Finder, drag the Godot project folder into the Xcode file browser.



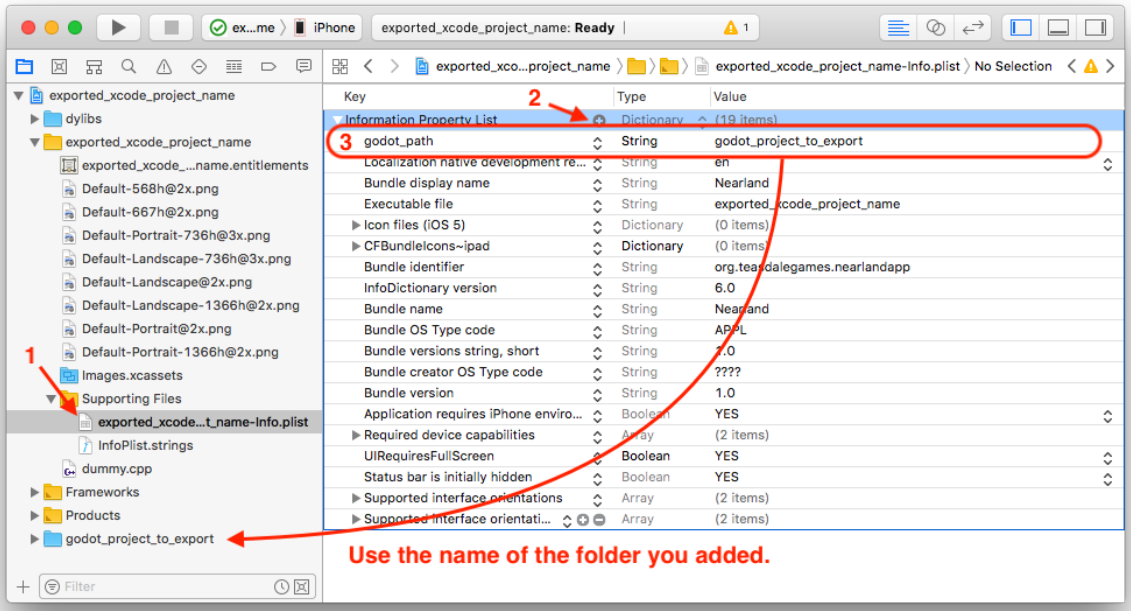
3. In the dialog, make sure **Create folder references** is selected. This means you will be able to continue to edit your Godot project in its current location.



4. See the **godot_project_to_export** folder in the Xcode file browser.
5. Delete **exported_xcode_project_name.pck** from the Xcode project.



6. Open **exported_xcode_project_name-Info.plist** and add a string property named **godot_path** (this is the real key name) with a value **godot_project_to_export** (this is the name of your project)



That's it! You can now edit your project in the Godot editor and build it in Xcode when you want to run it on a device.

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 submited to the Windows Store. Exporting UWP packages also works from any platform, not only from Windows.

However, if you want to install and run the app, you need to sign it with a trusted signature. Currently, Godot does not support signing of packages, so you need to use external tools to do so.

Also, make sure the Publisher Name you set when exporting the package matches the name used on the certificate.

Limitations on Xbox One

As described in UWP documentation:

- **Submitted as an "App"**
  - available memory is 1GB
  - share of 2-4 CPU cores
  - shared access of GPU power (45%)
- **Submitted as a "Game" (through Xbox Live Creators Program)**
  - available memory is 5GB
  - 4 exclusive CPU cores and 2 shared CPU cores

  - exclusive access to GPU power (100%)
- Exceeding these memory limitations will cause allocation failures and the application will crash.

Creating a signing certificate

This requires the `MakeCert.exe` and `Pvk2Pfx.exe` tools, which come with the Windows SDK. If you use Visual Studio, you can open one of its Developer Prompts, since it comes with these tools and they can be located in the path.

You can get more detailed instructions from Microsoft's documentation.

First, run `MakeCert` to create a private key:

```
MakeCert /n publisherName /r /h 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 will also need to trust this certificate in order to be able to install your app. Open the Command Prompt as Administrator and run the following command:

```
Certutil -addStore TrustedPeople MyKey.cer
```

Signing the package

Finally, use `SignTool.exe` from the Windows SDK or Visual Studio:

```
SignTool sign /fd SHA256 /a /f MyKey.pfx /p pfxPassword package.appx
```

Installing the package

As of the Windows 10 Anniversary Update, you are able to install packages simply by double clicking the `.appx` file from Windows Explorer.

It's also possible to install by using the `Add-AppxPackage` PowerShell cmdlet.

Note: If you want to update your already installed app, you must update the version number on the new package or first uninstall the previous package.

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 https://developer.android.com/studio/

Install OpenJDK or Oracle JDK

Download and install OpenJDK or Oracle JDK. Versions below JDK 8 may not 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:

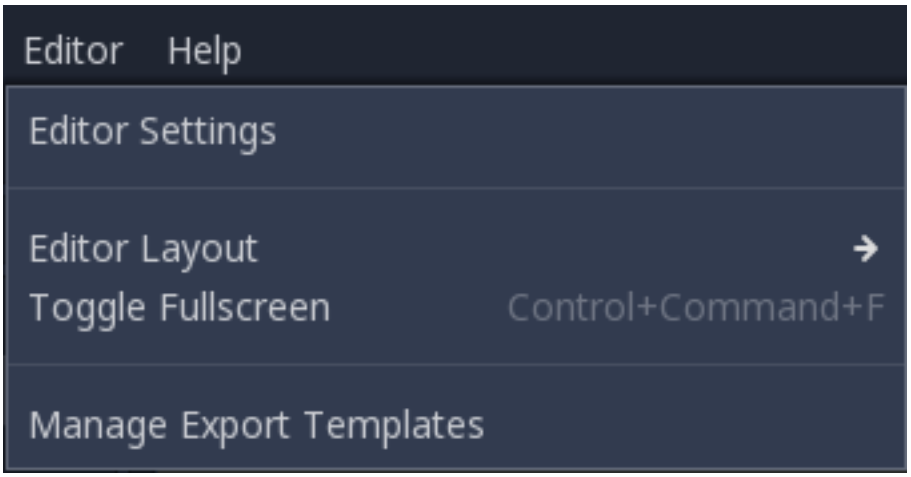
```
keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android -keystore␣
↪debug.keystore -storepass android -dname "CN=Android Debug,O=Android,C=US" -
↪validity 9999
```

Make sure you have adb

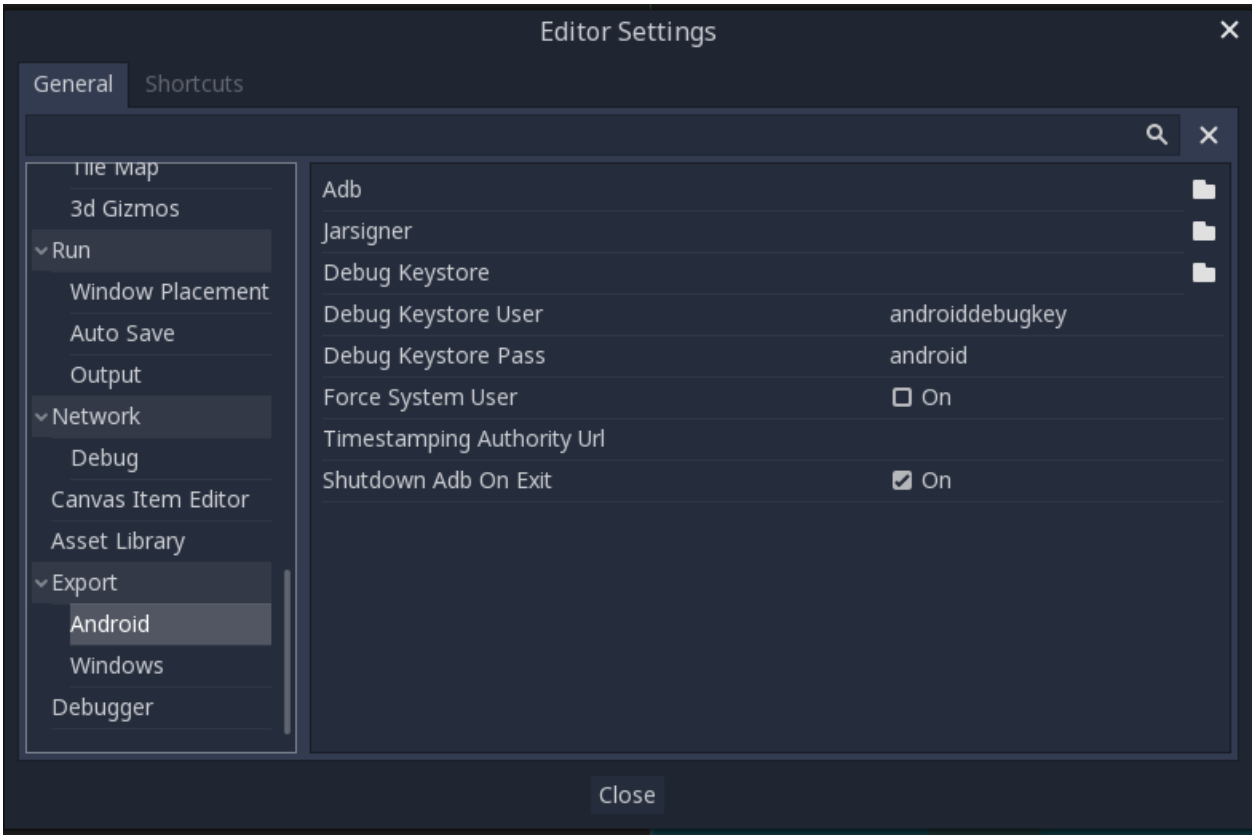
Android Debug Bridge (adb) is the command line tool used to communicate with Android devices. It's installed with the SDK, but you may need to install one (any) of the Android API levels for it to be installed in the SDK directory.

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. On **iOS**, all browsers must be based on WebKit (i.e. Safari), so they will also not work.

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 browsing 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
- Cannot progress more than once per frame, so polling in a loop will freeze
- 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, Chromium-based browsers specifically, 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 compression.

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. If offers a single method called `eval` that works similarly to the JavaScript 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`.

HTML5 export templates may be built without support for the singleton. With such templates, and on platforms other than HTML5, calling `JavaScript.eval` will also return `null`. The availability of the singleton can be checked with the `JavaScript` *feature tag*:

```
func my_func3():
    if OS.has_feature('JavaScript'):
        JavaScript.eval("console.log('The JavaScript singleton is available')")
    else:
        print("The JavaScript singleton is NOT available")
```

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_func4():
    # 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 the 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. Some simple use-cases where customizing the default page is useful include:

- Loading files from a different directory
- Loading a `.zip` file instead of a `.pck` file as main pack
- Loading engine files from a different directory than the main pack file
- Loading some extra files before the engine starts, so they are available in the file system later
- Passing custom "command line" arguments, e.g. `-s` to start a MainLoop script

Placeholder substitution

When exporting the game, several placeholders in the HTML page are substituted by values dependening on the export:

Placeholder	substituted by
`$GODOT_BASENAME`	Basename of exported files without suffixes, e.g. `game` when exporting `game.html`
`$GODOT_DEBUG_ENABLED`	`true` if debugging, `false` otherwise
`$GODOT_HEAD_INCLUDE`	Custom string to include just before the end of the HTML `<head>` 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 `<img />` 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.

`Engine.isWebGLAvailable(majorVersion = 1)`

Returns `true` if the given major version of WebGL is available, `false` otherwise. Defaults to `1` for WebGL 1.0.

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, path)`

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. If `file` is an `ArrayBuffer` or a view on one, the buffer will used as content of the file.

If `path` is a string, it specifies the path by which the file will be available. This is mandatory if `file` is not a string. Otherwise, the path is derived from the URL of the loaded file.

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.setCanvasResizedOnStart(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.setStderrFunc(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 printStderr(text) {
    console.warn("Error: " + text);
}
engine.setStderrFunc(printStderr);
```

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

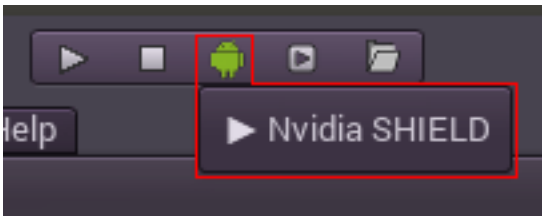
Sounds good, what is it?

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, likes the computer, usb is recognized, usb cable is plugged, etc.).
3. Connect the device..
4. And voila!



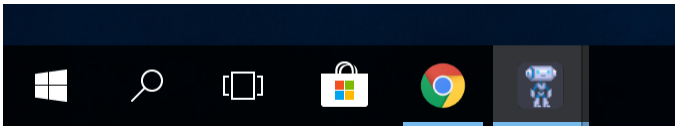
Click once.. and deploy!

5.3.10 Changing application icon for windows

By default, the exported game icon will be the Godot icon. Most likely you will want to change that for your game. There are two types of icons that can be changed: the file icon and the taskbar icon.

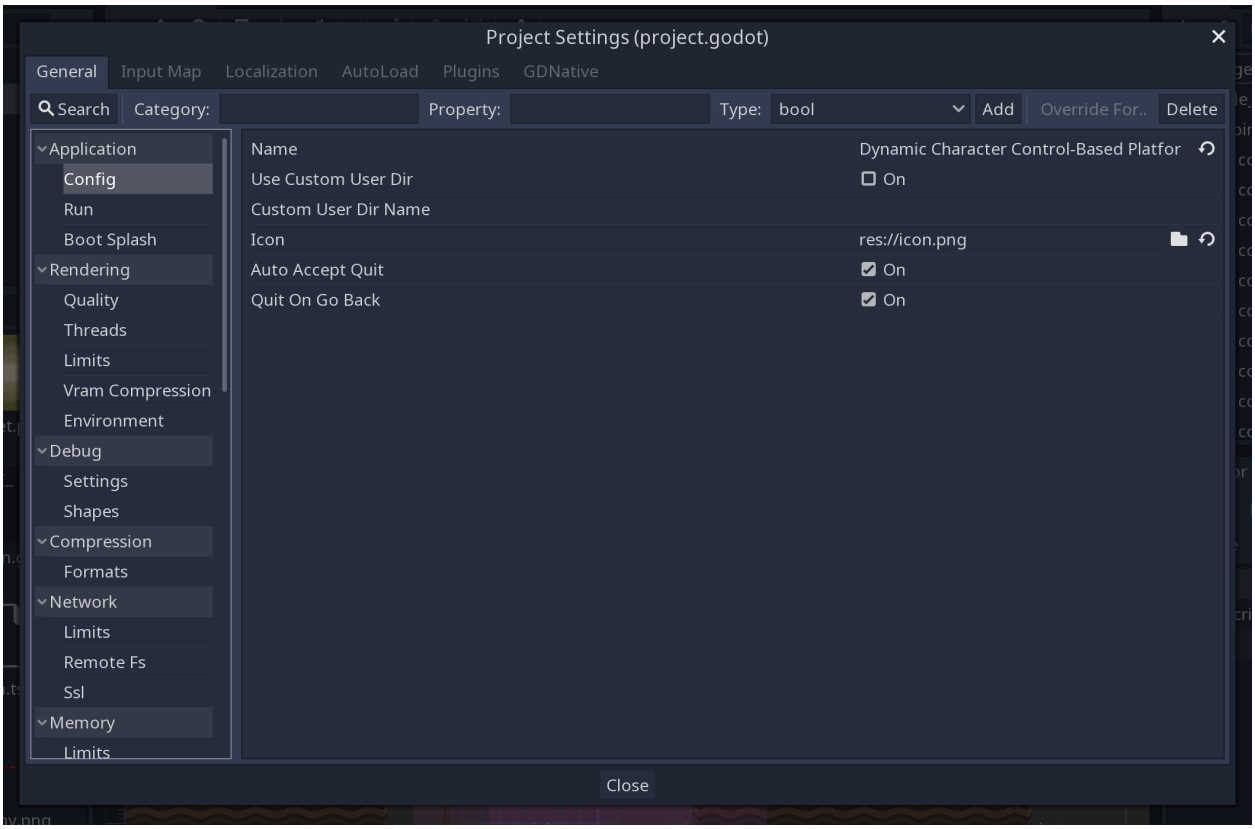
Changing the taskbar icon

The taskbar icon is the icon that shows up on the taskbar when your game is running.



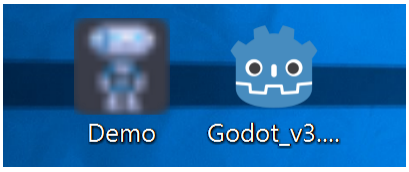
To change the taskbar icon, go to Project>Project Settings>Application>Config>Icon. Click on the folder icon and select your desired icon.

Note: This is also the icon that gets displayed in the Godot project list.



Changing the file icon

The file icon is the icon of the executable that you click on to start the game.



Before selecting it in the export options, you will need to install an extra tool called **rcedit**. You can download it here: https://github.com/electron/rcedit/releases

After downloading, you need to tell Godot the path to the **rcedit** executable on your computer. Go to Editor>Editor Settings>Export>Windows. Click on the folder icon for the **rcedit** entry. Navigate to and select the **rcedit** executable.

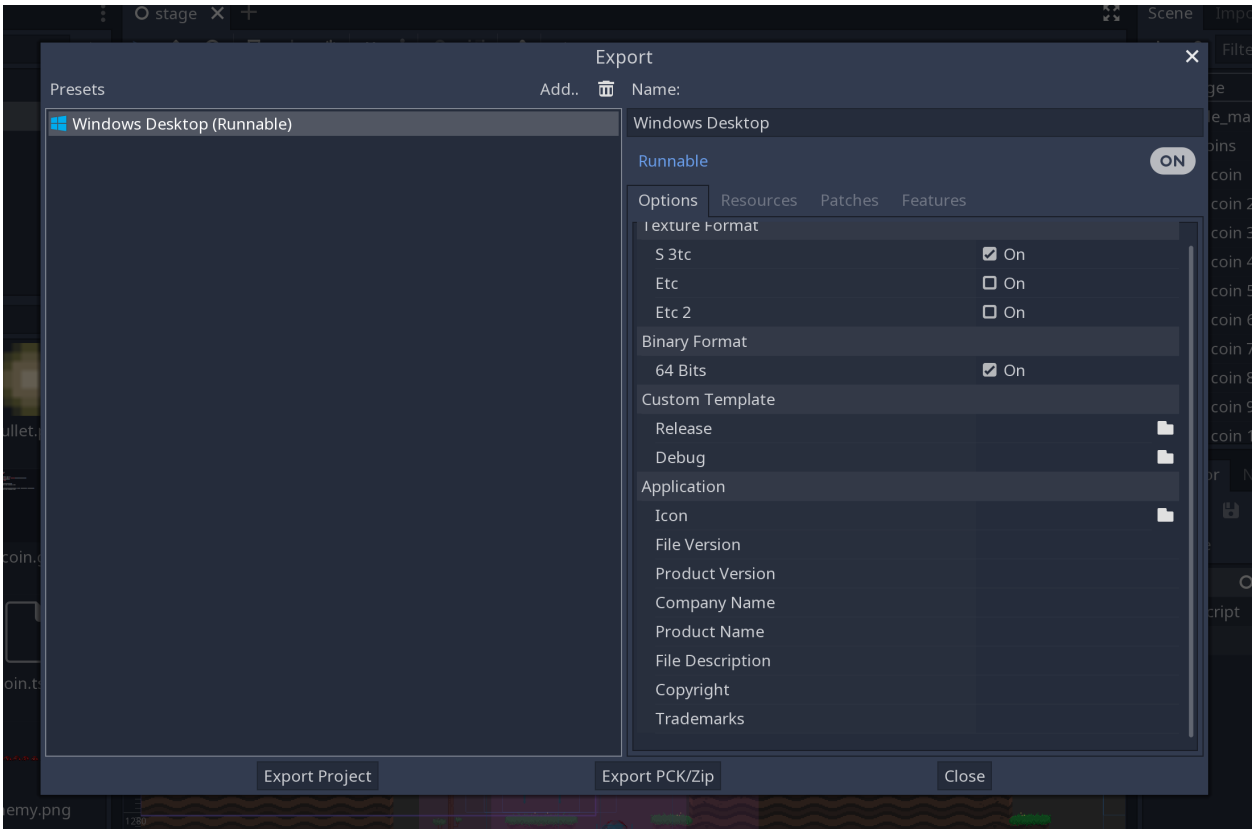
Note: For Linux users, you will also need to install wine in order to use rcedit. For more information, check https://www.winehq.org/

Now you have everything ready for changing the file icon. To do that, you will need to specify the icon when exporting. Go to Project>Export. Assuming you have a windows desktop preset ready, in the options, under Application, you will find Icon, select your desired image in ICO format as your file icon.

Note: To export an ICO image, you can use GIMP. For more details, please refer to this tutorial: http://skyboygames.com/easily-create-a-windows-app-icon-with-gimp/

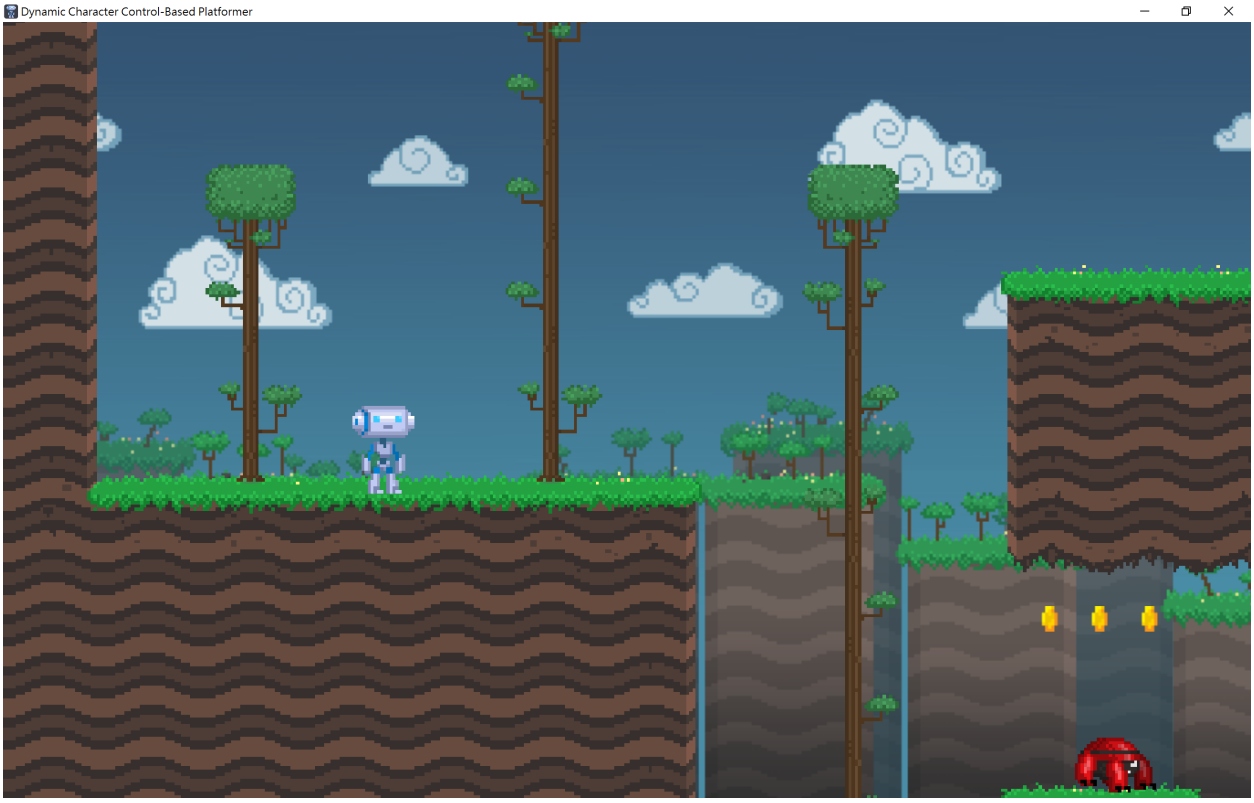
See also:

Check the documentation for more info about exporting.



Testing the result

You can now export the game and see whether you have change the icons successfully or not. If everything works fine, you will see this.



Icon (ICO) file requirements

Regardless of which program you use to create your ICO file, there are some requirements to ensure the icon (and your executable) works on Windows.

This is a bit tricky, as can be seen in the following StackOverflow threads: one, two.

Your ICO file should at least contain icons in the following resolutions: 16x16, 48x48 and 256x256. They should also be uncompressed. The 256x256 icon *can* be compressed, but this breaks backwards compatibility with Windows XP.

If you want to fully support high-DPI screens, this is the full list of supported icon sizes on Windows 10: 16, 20, 24, 28, 30, 31, 32, 40, 42, 47, 48, 56, 60, 63, 84 and one larger than 255px. (I.e. 256 or 512 or 1024)

Note that for high-DPI compression may be used, also they should be using 24bpp mode in contrast to the lower resolutions.

CHAPTER 6

2D

6.1 Canvas layers

6.1.1 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.

6.1.2 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.

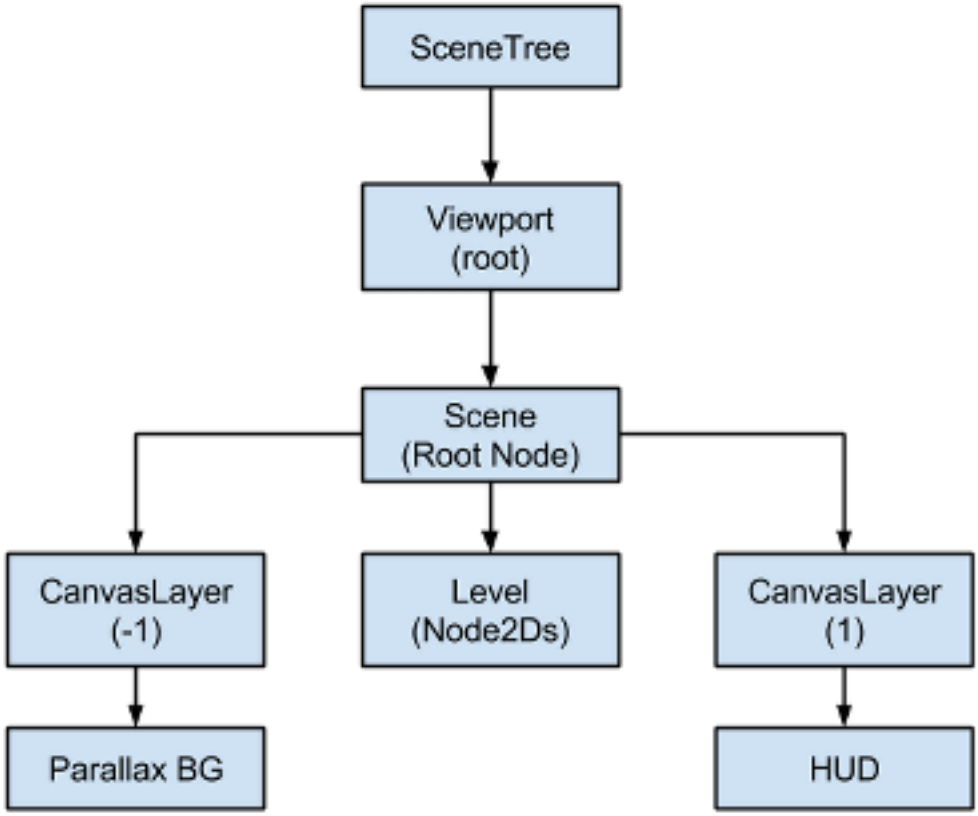
How can these problems be solved in a single scene tree?

6.1.3 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.

6.1.4 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*).

6.2 Viewport and canvas transforms

6.2.1 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.

6.2.2 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.

6.2.3 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.

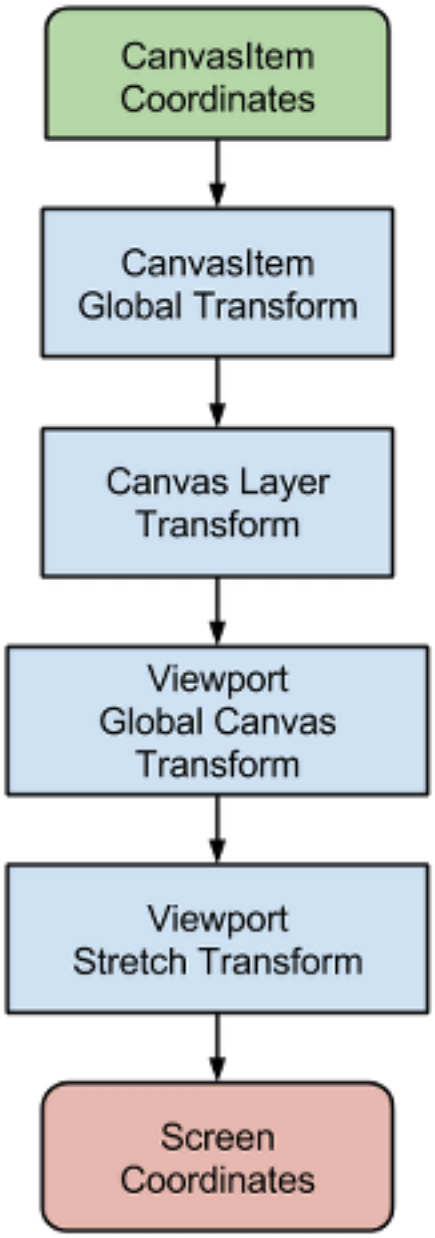
6.2.4 Stretch transform

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.

6.2.5 Transform order

For a coordinate in CanvasItem local properties to become an actual screen coordinate, the following chain of transforms must be applied:



6.2.6 Transform functions

Obtaining each transform can be achieved with the following functions:

Type	Transform
CanvasItem	*CanvasItem.get_global_transform()*
CanvasLayer	*CanvasItem.get_canvas_transform()*
CanvasLayer+GlobalCanvas+Stretch	*CanvasItem.get_viewport_transform()*

Finally then, to convert a CanvasItem local coordinates to screen coordinates, just multiply in the following order:

GDScript

C#

```
var screen_coord = get_viewport_transform() * ( get_global_transform() * local_pos )
```

```
var screenCord = (GetViewportTransform() * GetGlobalTransform()).Xform(localPos);
```

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.

6.2.7 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:

GDScript

C#

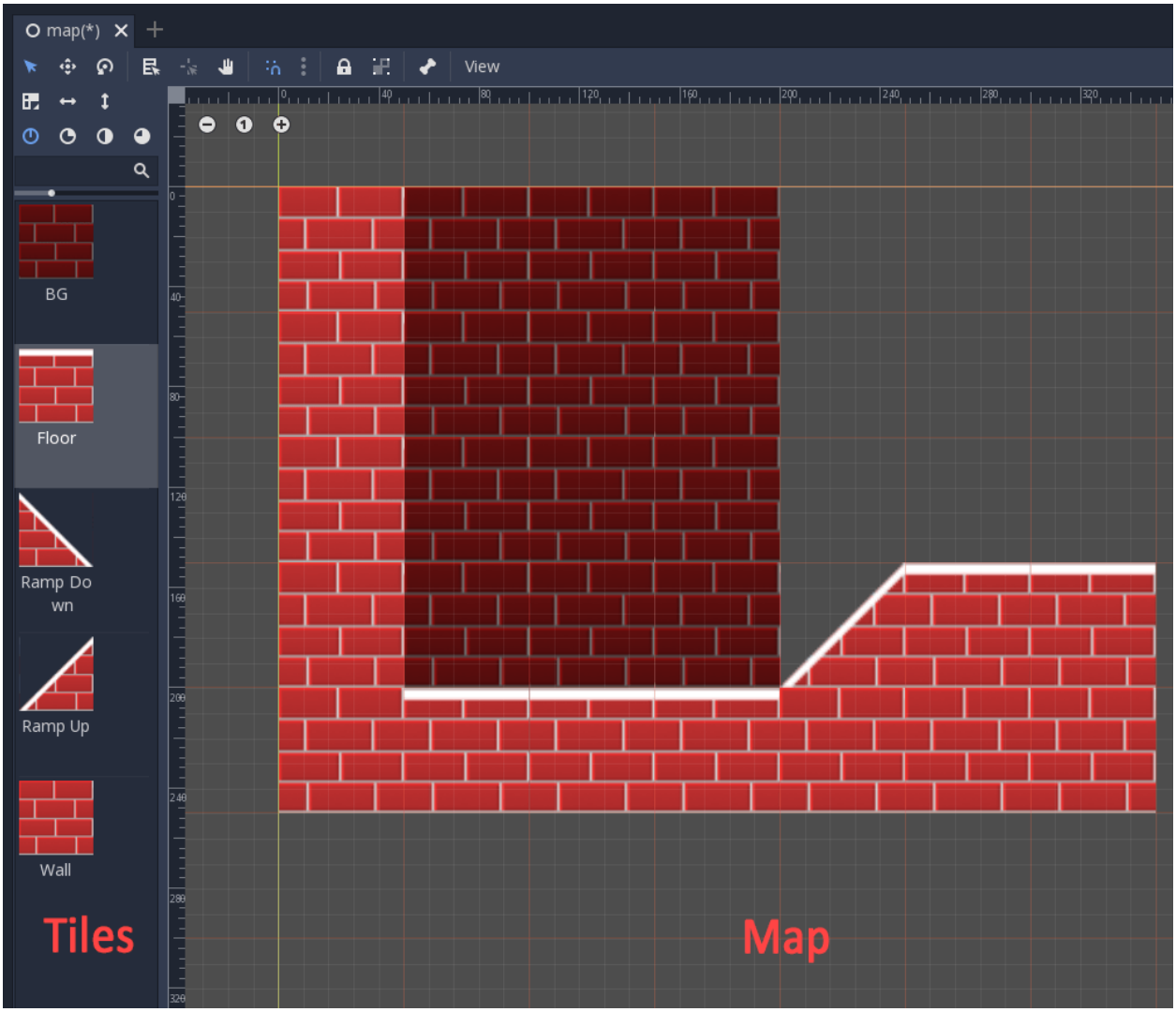
```
var local_pos = Vector2(10, 20) # local to Control/Node2D
var ie = InputEventMouseButton.new()
ie.button_index = BUTTON_LEFT
ie.position = get_viewport_transform() * (get_global_transform() * local_pos)
get_tree().input_event(ie)
```

```
var localPos = new Vector2(10,20); // local to Control/Node2D
var ie = new InputEventMouseButton();
ie.ButtonIndex = (int)ButtonList.Left;
ie.Position = (GetViewportTransform() * GetGlobalTransform()).Xform(localPos);
GetTree().InputEvent(ie);
```

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 on a grid, as many times each as desired - think of it like a map editor:



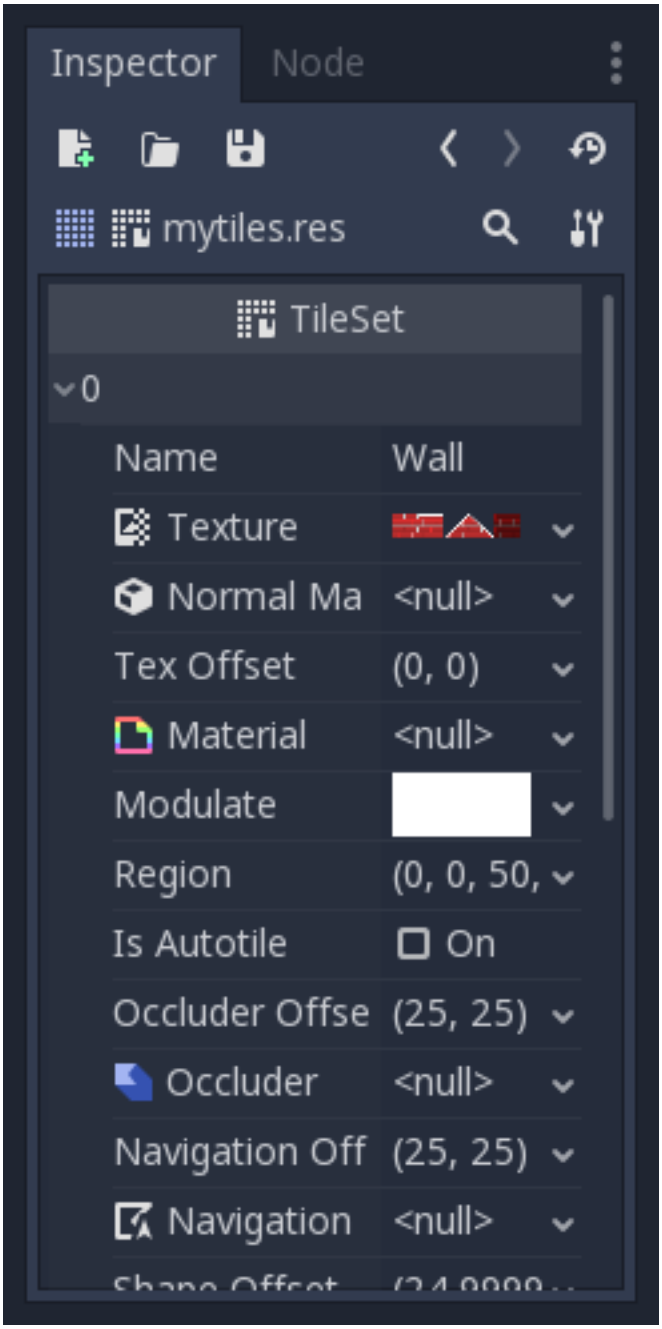
Collisions can also be added to the tiles, allowing for both 2D side scrolling and top down games.

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 for optimization reasons. There are so-called *texture packers* that will generate these spritesheets out of your separate texture files. 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's import settings and turn off `Filter`, keeping it on will cause issues later. `Mipmaps` should already be disabled, if not, disable this too.

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!

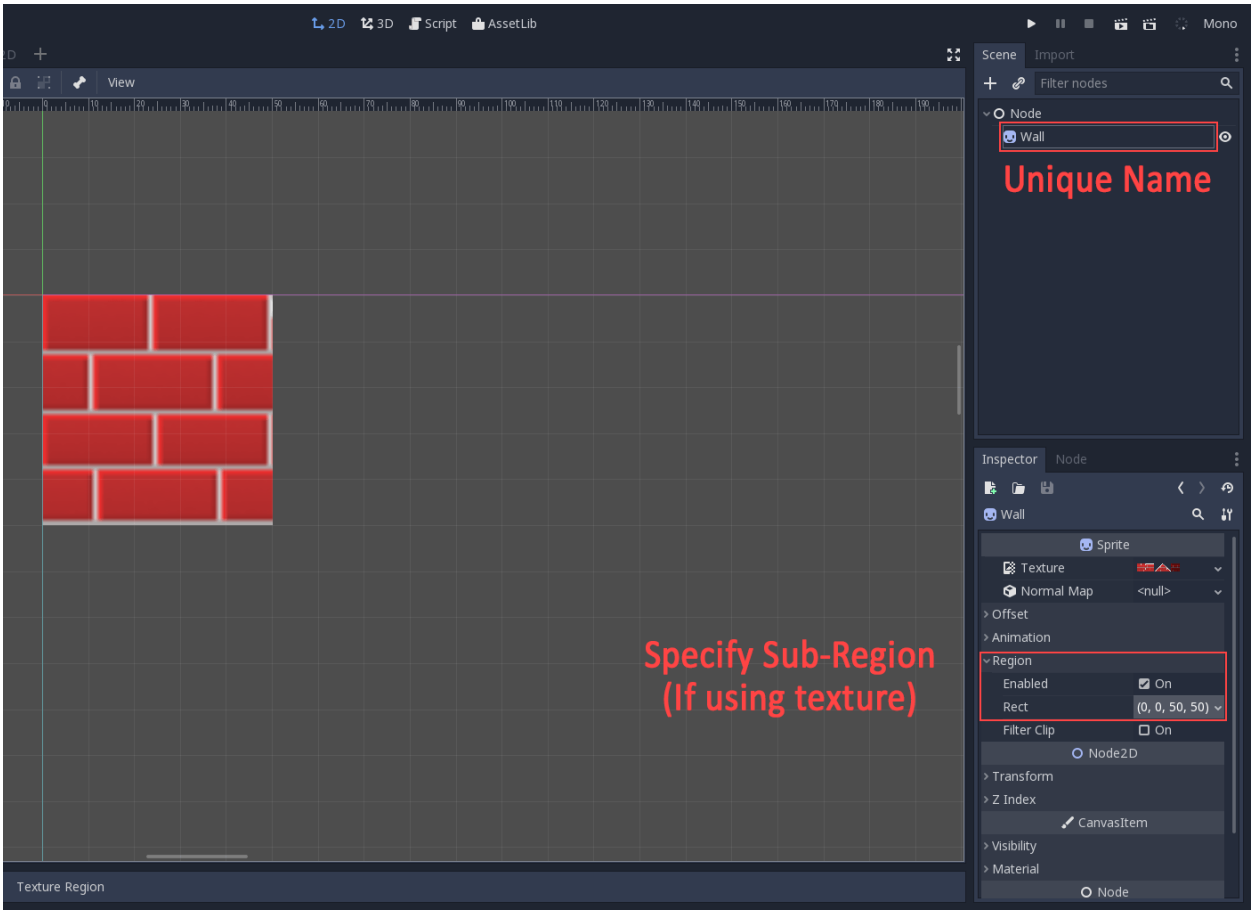
6.3.3 TileSet scene

Create a new scene with a regular Node or Node2D as root. For each tile you want to define, add a sprite node as a child. Since tiles here are 50x50, you should turn on the grid (`View -> Show Grid` or `G` key) and enable snap (`Use Snap` icon or `S` key). Moving tiles with the mouse might still be a inaccurate so use your arrow keys as well.

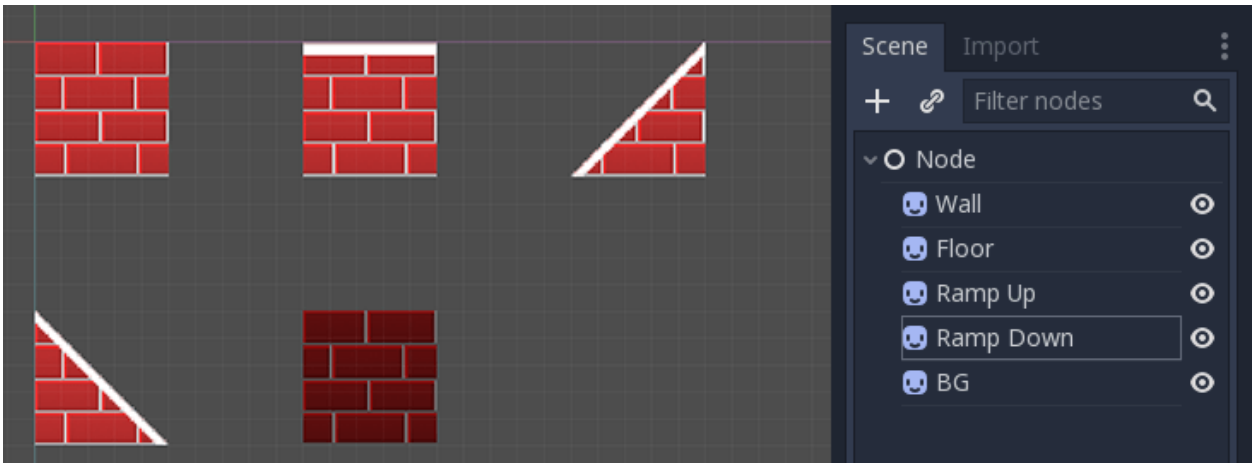
If more than one tile is present in the source image, make sure to use the region property of the sprite to adjust which part of the texture is being used.

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:

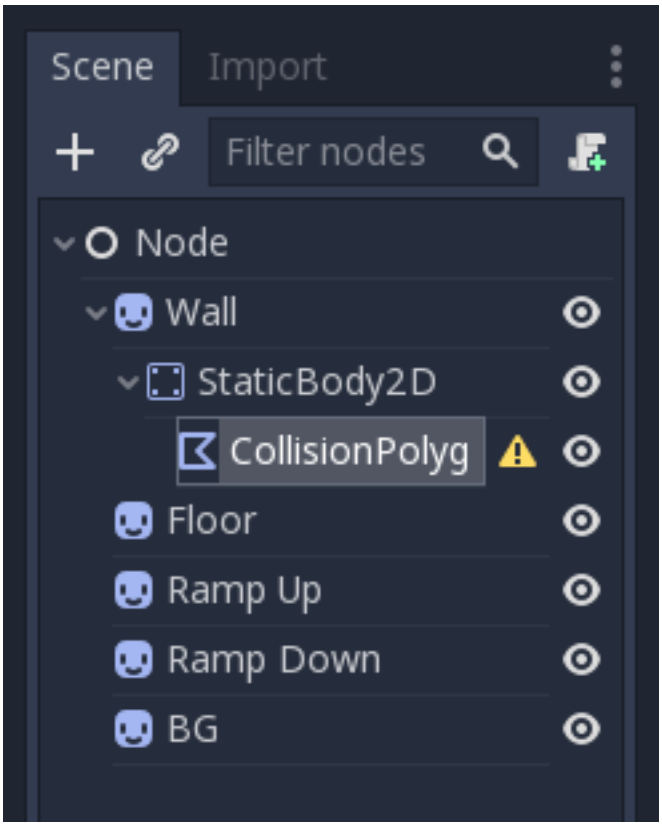


Continue adding all the tiles, adjusting the offsets if needed (that is, if you have multiple tiles in a single source image). Again, *remember that their names must be unique*.

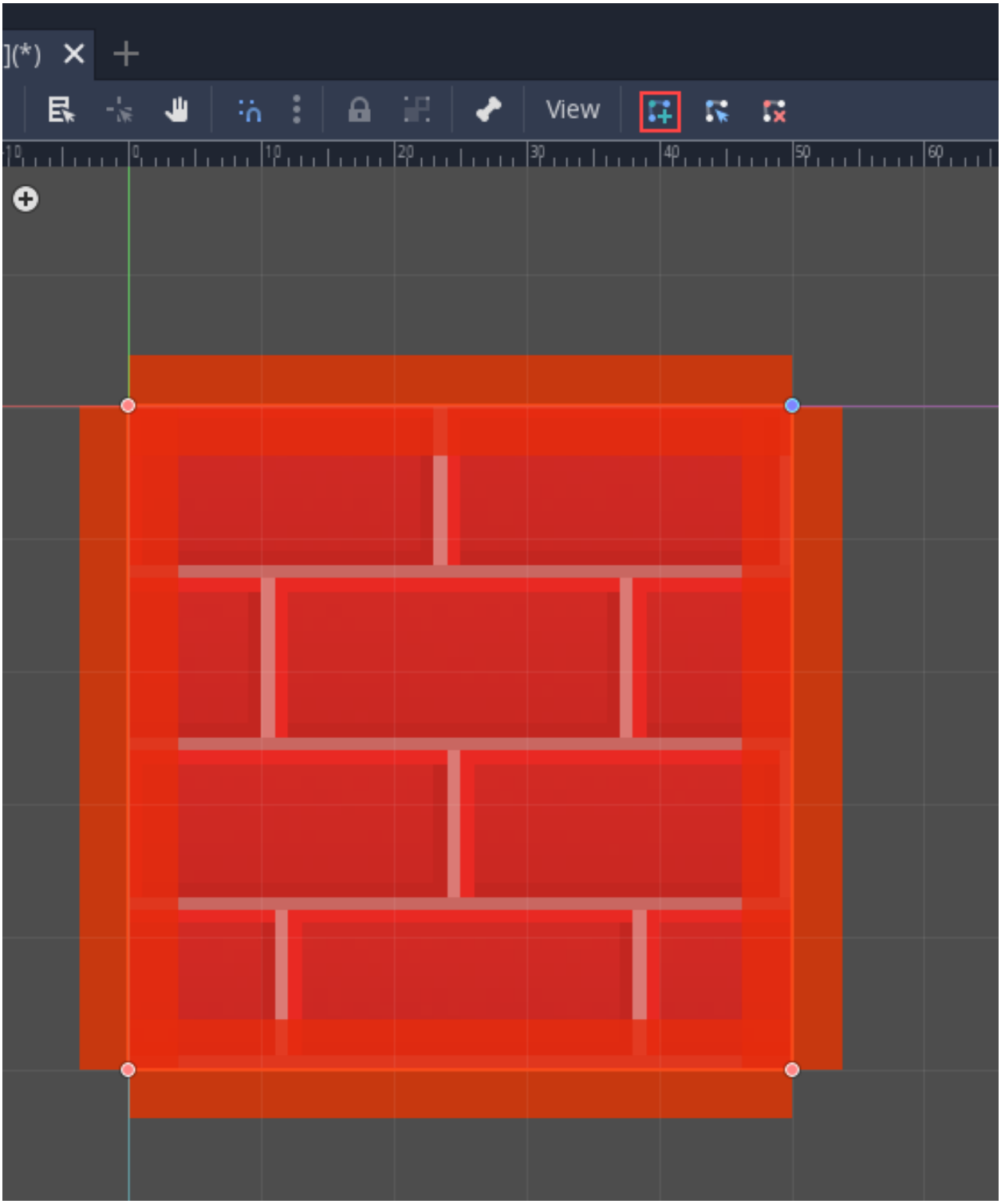


6.3.4 Collision

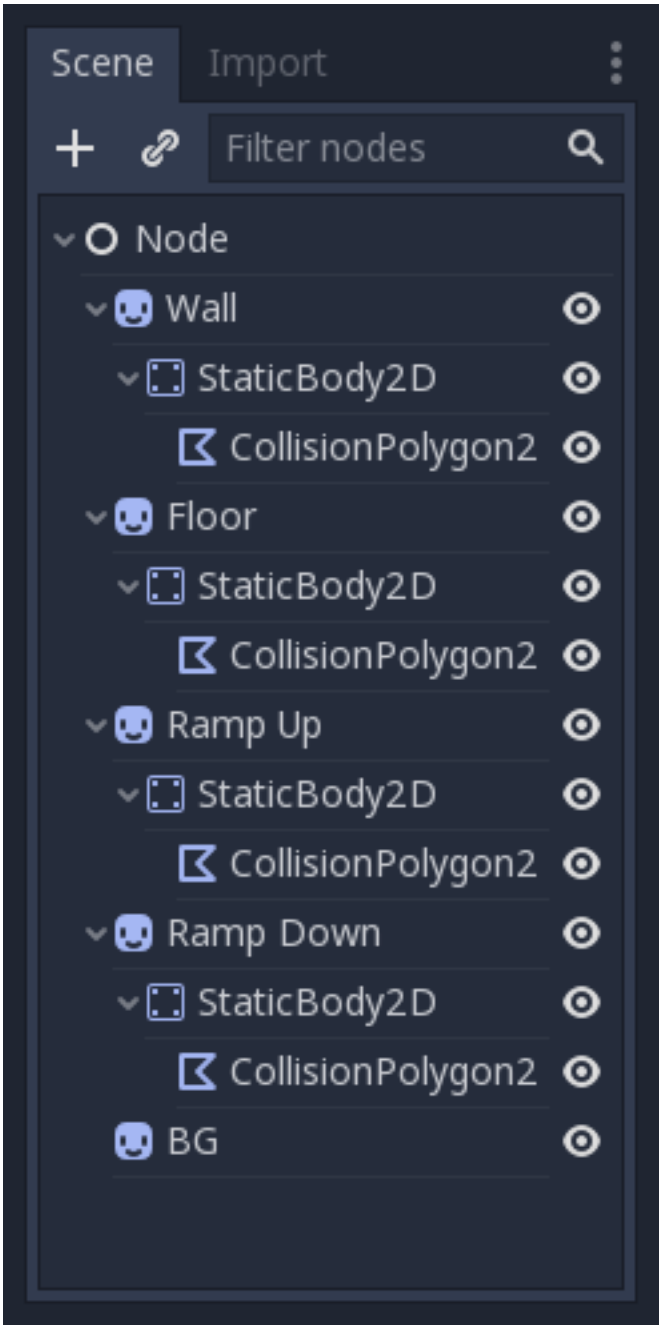
To add collision to a tile, create a StaticBody2D child for each sprite. This is a static collision node. Then create a CollisionShape2D or CollisionPolygon as a child of the StaticBody2D. The CollisionPolygon is recommended because it is easier to edit.



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.



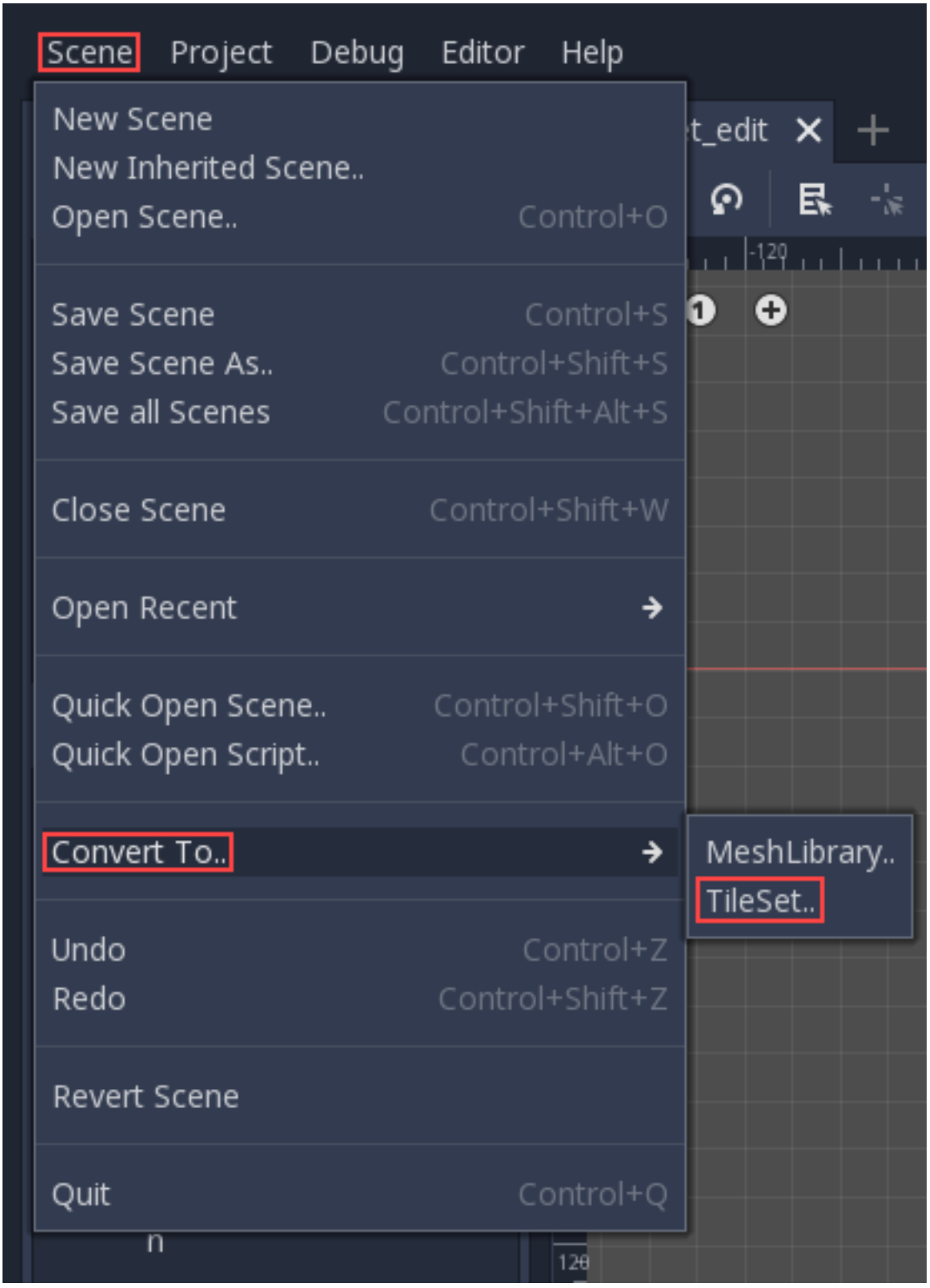
Keep adding collisions to the tiles until we are done. Note that BG is just a background, so it should not have a collision.



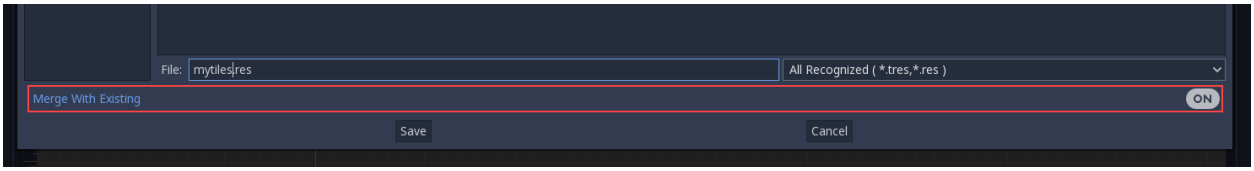
OK! We're done! Remember to save this scene for future edit. Name it "tileset_edit.scn" or something like that.

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:

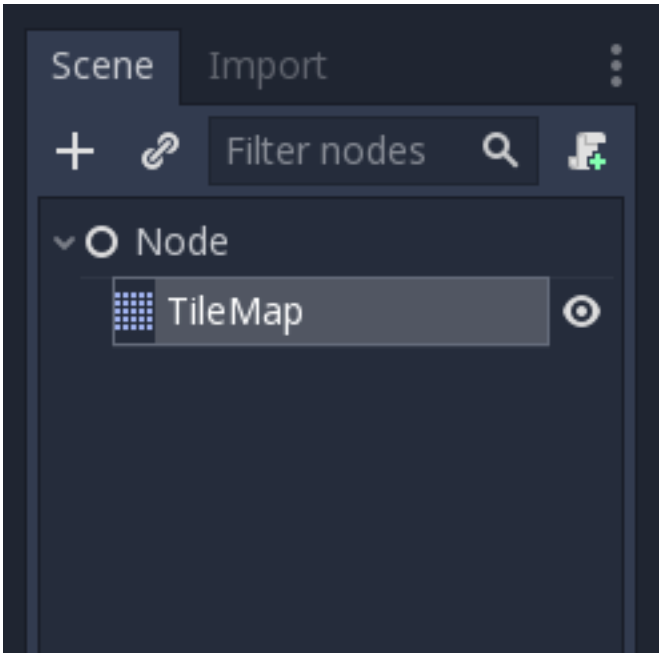


Then choose a filename, like "mytiles.tres". Make sure the "Merge With Existing" option is toggled on. This way, every time the tileset resource file is overwritten, existing tiles are merged and updated (they are referenced by their unique name, so again, **name your tiles properly**).

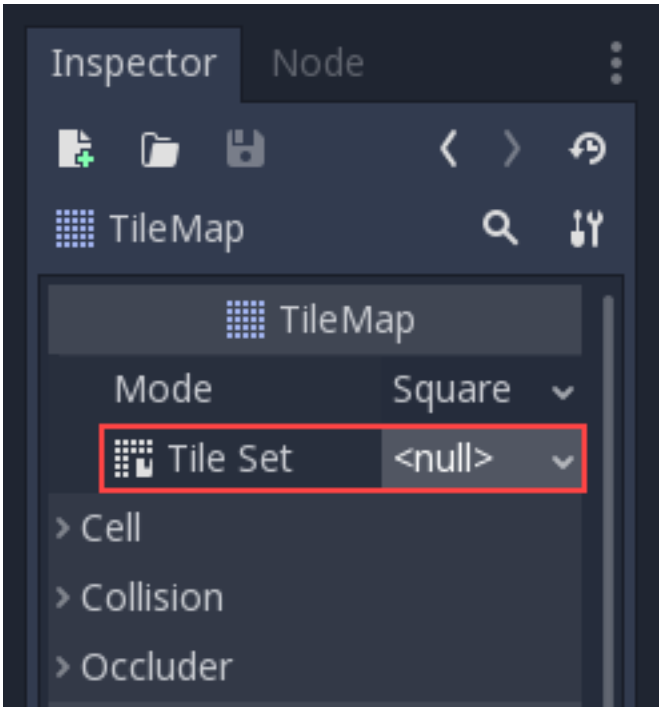


6.3.6 Using the TileSet in a TileMap

Create a new scene, using any node or node2d as root, and then create a *TileMap* as a child.



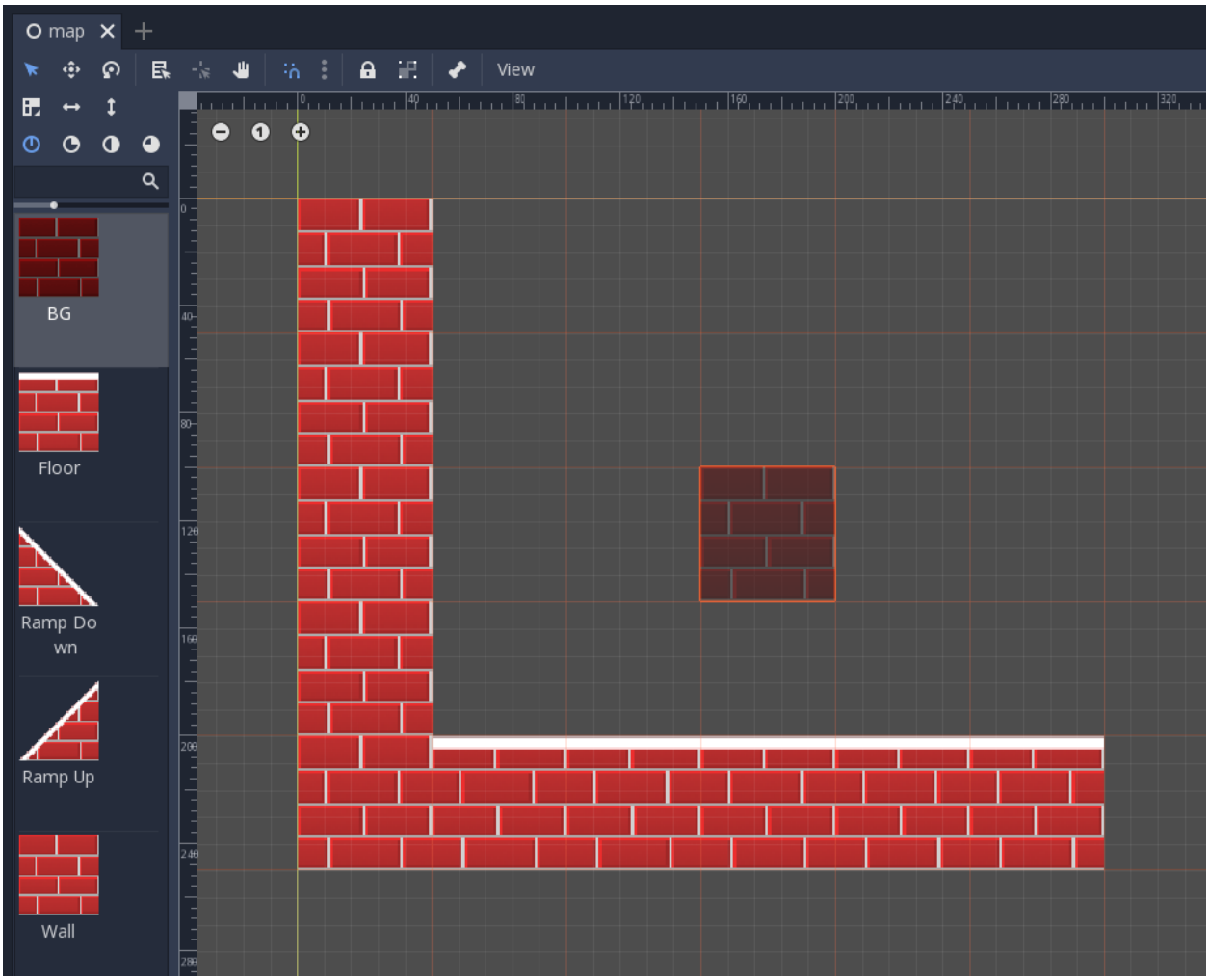
Go to the TileSet property of this node and assign the one created in previous steps:



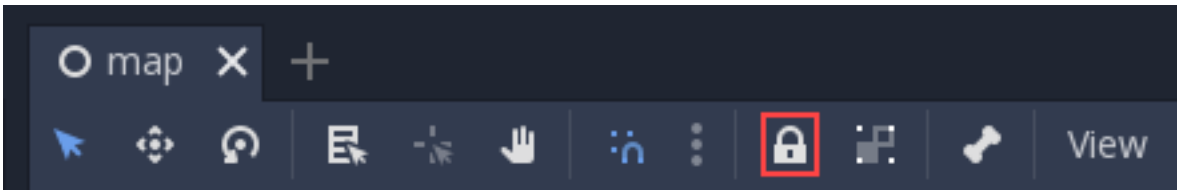
Also set the cell size to '50', since that is the size used by the tiles. Quadrant size is a tuning value, which means that the engine will draw and cull the tilemap in blocks of 16x16 tiles. This value is usually fine and does not need to be changed, but can be used to fine tune performance in specific 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:

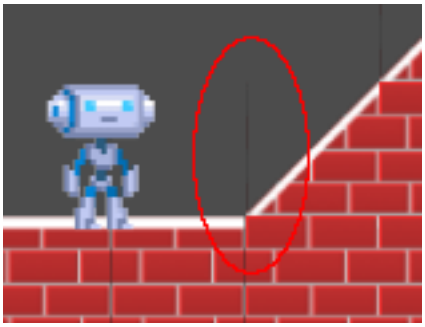


If you accidentally place a tile somewhere you don't want it to be, you can delete it with `RMB` while in the tilemap editor.

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 and mipmaps for either the tileset texture or all tile textures if using separate images (see the *Importing Images* asset pipeline tutorial).
- Enable pixel snap (Set `Project > Project Settings > Rendering > Quality > 2d > Use Pixel Snap` to true, you can also search for `Pixel Snap`).
- Viewport Scaling can often help with shrinking the map (see the *Viewports* tutorial). Simply adding a camera, setting it to `Current` and playing around with it's `Zoom` may be a good starting point.
- You can use a single, separate image for each tile. This will remove all artifacts, but can be more cumbersome to implement and is less optimized.

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.

GDScript

C#

```
extends Node2D

func _draw():
    # Your draw commands here
    pass
```

```
public override void _Draw()
{
    // Your draw commands here
}
```

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:

GDScript

C#

```
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())
```

```
public class CustomNode2D : Node2D
{
    private Texture _texture;
    public Texture Texture
    {
        get
        {
            return _texture;
        }
```

(continues on next page)

(continued from previous page)

```
        set
        {
            _texture = value;
            Update();
        }
    }

    public override void _Draw()
    {
        DrawTexture(_texture, new Vector2());
    }
}
```

In some cases, it may be desired to draw every frame. For this, just call update() from the _process() callback, like this:

GDScript

C#

```
extends Node2D

func _draw():
    # Your draw commands here
    pass

func _process(delta):
    update()
```

```
public class CustomNode2D : Node2D
{
    public override _Draw()
    {
        // Your draw commands here
    }

    public override _Process(delta)
    {
        Update();
    }
}
```

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.

GDScript

C#

```
func draw_circle_arc(center, radius, angle_from, angle_to, color):
    var nb_points = 32
    var points_arc = PoolVector2Array()

    for i in range(nb_points+1):
        var angle_point = deg2rad(angle_from + i * (angle_to-angle_from) / nb_points -
↪ 90)
        points_arc.push_back(center + Vector2(cos(angle_point), sin(angle_point)) *␣
↪radius)

    for index_point in range(nb_points):
        draw_line(points_arc[index_point], points_arc[index_point + 1], color)
```

```
public void DrawCircleArc(Vector2 center, float radius, float angleFrom, float␣
↪angleTo, Color color)
{
    int nbPoints = 32;
    var pointsArc = new Vector2[nbPoints];

    for (int i = 0; i < nbPoints; ++i)
    {
        float anglePoint = Mathf.Deg2Rad(angleFrom + i * (angleTo - angleFrom) /␣
↪nbPoints - 90f);
        pointsArc[i] = center + new Vector2(Mathf.Cos(anglePoint), Mathf.
↪Sin(anglePoint)) * radius;
    }

    for (int i = 0; i < nbPoints - 1; ++i)
        DrawLine(pointsArc[i], pointsArc[i + 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 PoolVector2Array, 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 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 PoolVector2Array 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 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.

GDScript

C#

```
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)
```

```
public override void _Draw()
{
    var center = new Vector2(200, 200);
    float radius = 80;
    float angleFrom = 75;
    float angleTo = 195;
    var color = new Color(1, 0, 0);
    DrawCircleArc(center, radius, angleFrom, angleTo, 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:

GDScript

C#

```
func draw_circle_arc_poly(center, radius, angle_from, angle_to, color):
    var nb_points = 32
    var points_arc = PoolVector2Array()
    points_arc.push_back(center)
    var colors = PoolColorArray([color])

    for i in range(nb_points+1):
        var angle_point = deg2rad(angle_from + i * (angle_to - angle_from) / nb_
↪points - 90)
        points_arc.push_back(center + Vector2(cos(angle_point), sin(angle_point)) *␣
↪radius)
    draw_polygon(points_arc, colors)
```

```
public void DrawCircleArcPoly(Vector2 center, float radius, float angleFrom, float␣
↪angleTo, Color color)
{
    int nbPoints = 32;
    var pointsArc = new Vector2[nbPoints + 1];
    pointsArc[0] = center;
    var colors = new Color[] { color };
```

(continues on next page)

(continued from previous page)

```
    for (int i = 0; i < nbPoints; ++i)
    {
        float anglePoint = Mathf.Deg2Rad(angleFrom + i * (angleTo - angleFrom) /␣
↪nbPoints - 90);
        pointsArc[i + 1] = center + new Vector2(Mathf.Cos(anglePoint), Mathf.
↪Sin(anglePoint)) * radius;
    }

    DrawPolygon(pointsArc, colors);
}
```

Dynamic custom drawing

Alright, we are now able to draw custom stuff on screen. However, it is 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().

GDScript

C#

```
extends Node2D

var rotation_angle = 50
```

(continues on next page)

(continued from previous page)

```
var angle_from = 75
var angle_to = 195
```

```
public class CustomNode2D : Node2D
{
    private float _rotationAngle = 50;
    private float _angleFrom = 75;
    private float _angleTo = 195;
}
```

We make these values change in the _process(delta) 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.

GDScript

C#

```
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 and angle_to > 360:
        angle_from = wrap(angle_from, 0, 360)
        angle_to = wrap(angle_to, 0, 360)
    update()
```

```
private float Wrap(float value, float minVal, float maxVal)
{
    float f1 = value - minVal;
    float f2 = maxVal - minVal;
    return (f1 % f2) + minVal;
}

public override void _Process(float delta)
{
    _angleFrom += _rotationAngle;
    _angleTo += _rotationAngle;

    // We only wrap angles if both of them are bigger than 360
    if (_angleFrom > 360 && _angleTo > 360)
    {
        _angleFrom = Wrap(_angleFrom, 0, 360);
        _angleTo = Wrap(_angleTo, 0, 360);
```

(continues on next page)

(continued from previous page)

```
    }
    Update();
}
```

Also, don't forget to modify the _draw() function to make use of these variables:

GDScript

C#

```
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 )
```

```
public override void _Draw()
{
    var center = new Vector2(200, 200);
    float radius = 80;
    var color = new Color(1, 0, 0);

    DrawCircleArc(center, radius, _angleFrom, _angleTo, 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.

GDScript

C#

```
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 and angle_to > 360:
        angle_from = wrap(angle_from, 0, 360)
        angle_to = wrap(angle_to, 0, 360)
    update()
```

```
public override void _Process(float delta)
{
    _angleFrom += _rotationAngle * delta;
    _angleTo += _rotationAngle * delta;

    // We only wrap angles if both of them are bigger than 360
```

(continues on next page)

(continued from previous page)

```
    if (_angleFrom > 360 && _angleTo > 360)
    {
        _angleFrom = Wrap(_angleFrom, 0, 360);
        _angleTo = Wrap(_angleTo, 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 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.

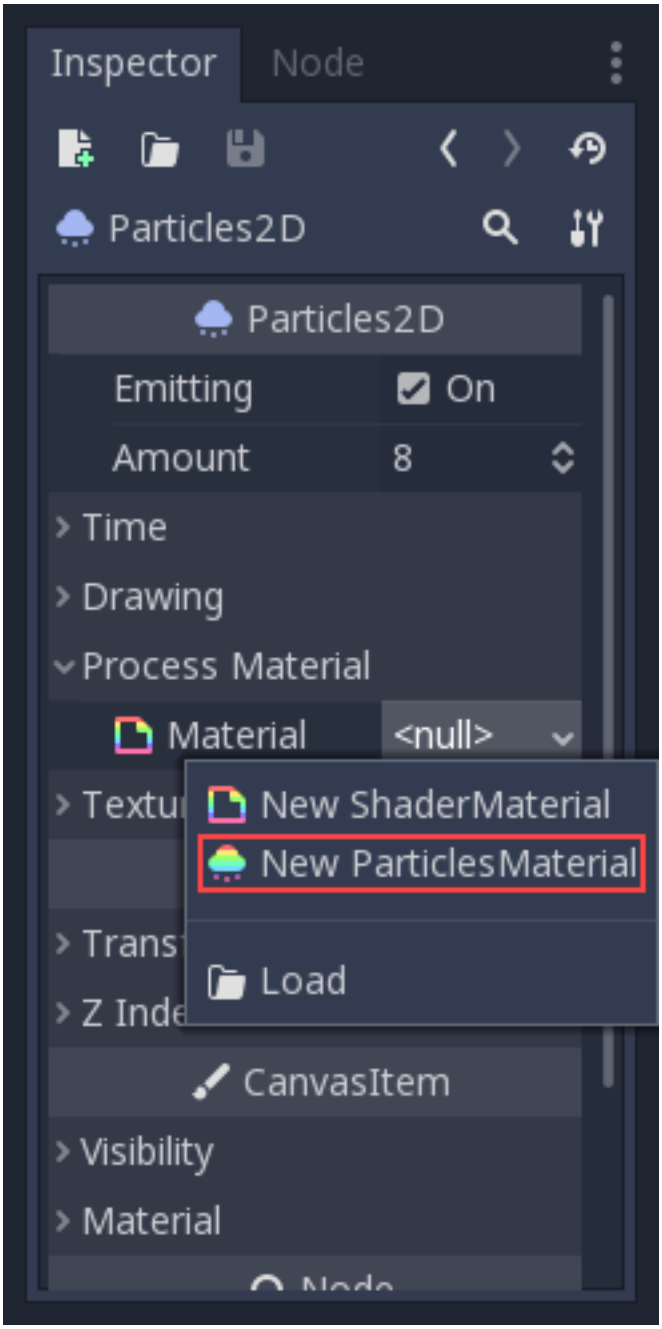
The idea is that a "particle" is emitted at a fixed interval and with a fixed lifetime. During its lifetime, every particle will have the same base behavior. What makes every particle different and provides a more organic look is the "randomness" associated to each parameter. In essence, creating a particle 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

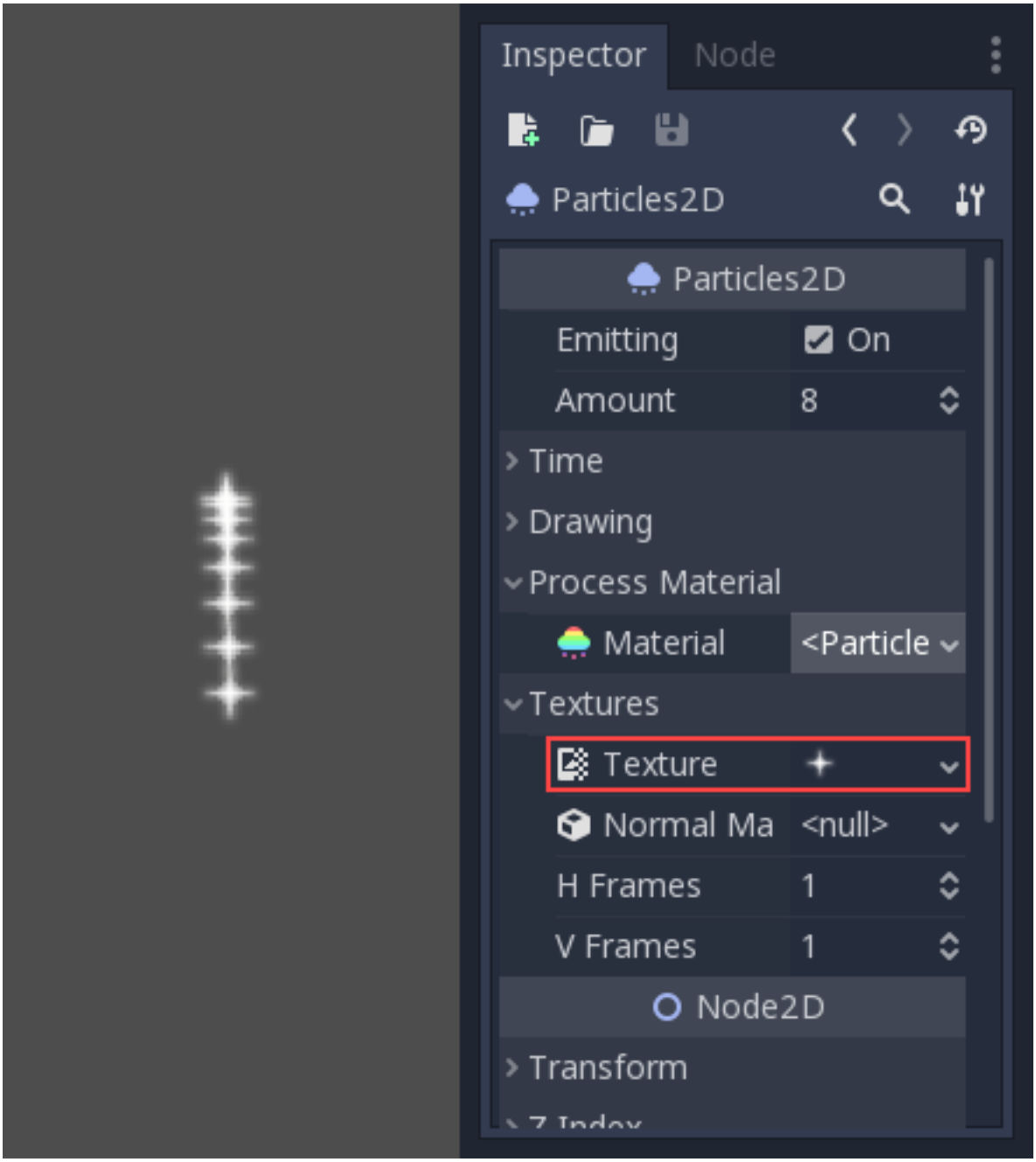
To add a process material to your particles node, go to Process Material in your inspector panel. Click on the box next to material, and from the dropdown menu select New Particles Material.



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 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

Commented out as not implemented in 3.x for now.

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 degress). 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.

6.6 2D Movement Overview

6.6.1 Introduction

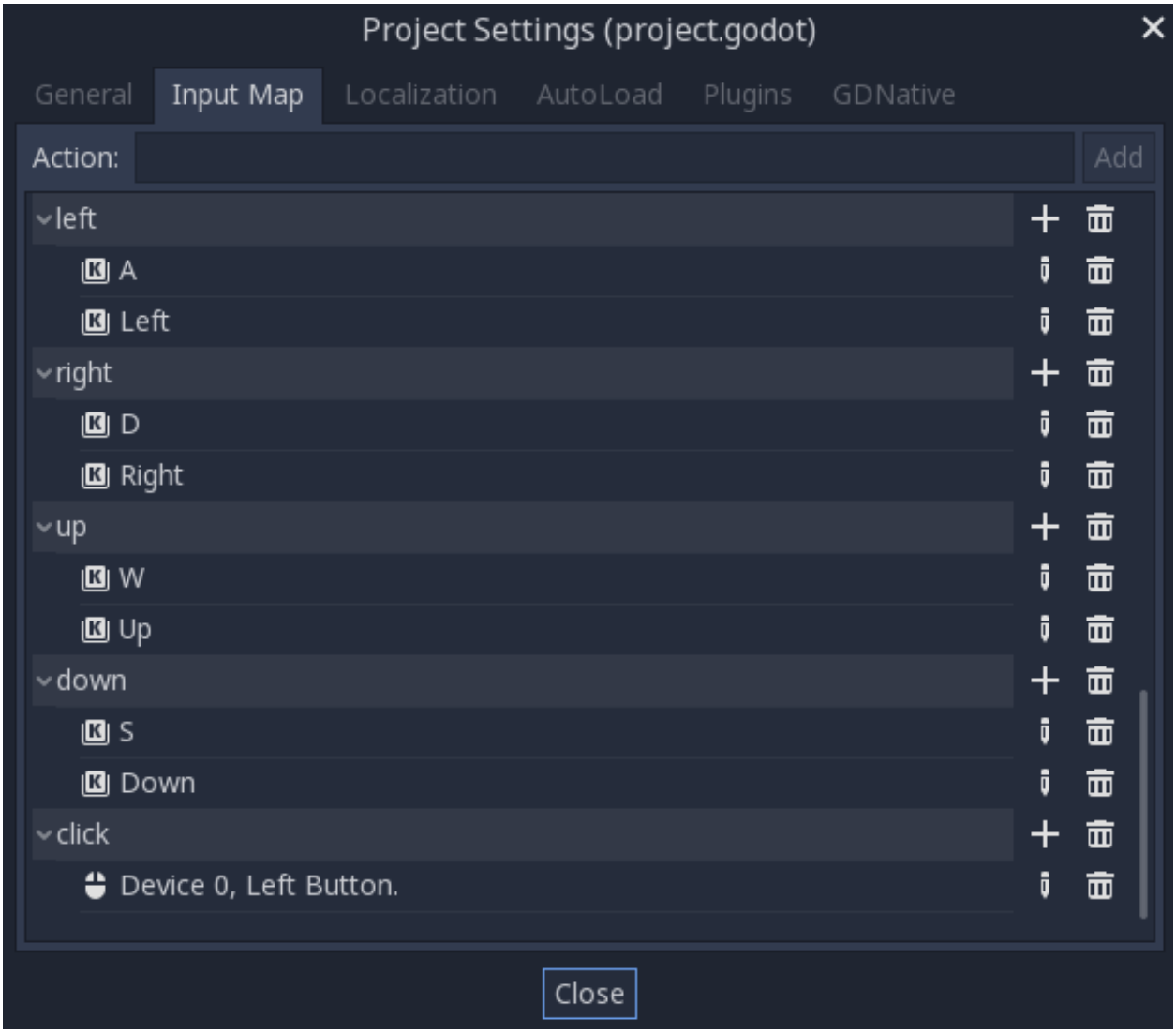
Every beginner has been there: "How do I move my character?" Depending on the style of game you're making, you may have special requirements, but in general the movement in most 2D games is based on a small number of designs.

We'll use *KinematicBody2D* for these examples, but the principles will apply to other node types (Area2D, RigidBody2D) as well.

6.6.2 Setup

Each example below uses the same scene setup. Start with a `KinematicBody2D` with two children: `Sprite` and `CollisionShape2D`. You can use the Godot icon ("icon.png") for the Sprite's texture or use any other 2D image you have.

Open `Project -> Project Settings` and select the "Input Map" tab. Add the following input actions (see *InputEvent* for details):



6.6.3 8-Way Movement

In this scenario, you want the user to press the four directional keys (up/left/down/right or W/A/S/D) and move in the selected direction. The name "8-way movement" comes from the fact that the player can move diagonally by pressing two keys at the same time.

Add a script to the kinematic body and add the following code:

GDScript

C#

```
extends KinematicBody2D

export (int) var speed = 200

var velocity = Vector2()
```

(continues on next page)

(continued from previous page)

```
func get_input():
    velocity = Vector2()
    if Input.is_action_pressed('right'):
        velocity.x += 1
    if Input.is_action_pressed('left'):
        velocity.x -= 1
    if Input.is_action_pressed('down'):
        velocity.y += 1
    if Input.is_action_pressed('up'):
        velocity.y -= 1
    velocity = velocity.normalized() * speed

func _physics_process(delta):
    get_input()
    move_and_slide(velocity)
```

```
using Godot;
using System;

public class Movement : KinematicBody2D
{
    [Export] public int Speed = 200;

    Vector2 velocity = new Vector2();

    public void GetInput()
    {
        velocity = new Vector2();
        if (Input.IsActionPressed("right"))
        {
            velocity.x += 1;
        }
        if (Input.IsActionPressed("left"))
        {
            velocity.x -= 1;
        }
        if (Input.IsActionPressed("down"))
        {
            velocity.y += 1;
        }
        if (Input.IsActionPressed("up"))
        {
            velocity.y -= 1;
        }
        velocity = velocity.Normalized() * Speed;
    }

    public override void _PhysicsProcess(float delta)
    {
        GetInput();
        MoveAndSlide(velocity);
    }
}
```

In the `get_input()` function we check for the four key events and sum them up to get the velocity vector. This has the benefit of making two opposite keys cancel each other out, but will also result in diagonal movement being faster

due to the two directions being added together.

We can prevent that if we *normalize* the velocity, which means we set its *length* to `1`, and multiply by the desired speed.

Tip: If you've never used vector math before, or need a refresher, you can see an explanation of vector usage in Godot at *Vector math*.

6.6.4 Rotation + Movement

This type of movement is sometimes called "Asteroids-style" because it resembles how that classic arcade game worked. Pressing left/right rotates the character, while up/down moves it forward or backward in whatever direction it's facing.

GDScript

C#

```
extends KinematicBody2D

export (int) var speed = 200
export (float) var rotation_speed = 1.5

var velocity = Vector2()
var rotation_dir = 0

func get_input():
    rotation_dir = 0
    velocity = Vector2()
    if Input.is_action_pressed('right'):
        rotation_dir += 1
    if Input.is_action_pressed('left'):
        rotation_dir -= 1
    if Input.is_action_pressed('down'):
        velocity = Vector2(-speed, 0).rotated(rotation)
    if Input.is_action_pressed('up'):
        velocity = Vector2(speed, 0).rotated(rotation)

func _physics_process(delta):
    get_input()
    rotation += rotation_dir * rotation_speed * delta
    move_and_slide(velocity)
```

```
using Godot;
using System;

public class Movement : KinematicBody2D
{
    [Export] public int Speed = 200;
    [Export] public float RotationSpeed = 1.5f;

    Vector2 velocity = new Vector2();
    int rotationDir = 0;
```

(continues on next page)

(continued from previous page)

```
    public void GetInput()
    {
        rotationDir = 0;
        velocity = new Vector2();
        if (Input.IsActionPressed("right"))
        {
            rotationDir += 1;
        }
        if (Input.IsActionPressed("left"))
        {
            rotationDir -= 1;
        }
        if (Input.IsActionPressed("down"))
        {
            velocity = new Vector2(-Speed, 0).Rotated(Rotation);
        }
        if (Input.IsActionPressed("up"))
        {
            velocity = new Vector2(Speed, 0).Rotated(Rotation);
        }
        velocity = velocity.Normalized() * Speed;
    }

    public override void _PhysicsProcess(float delta)
    {
        GetInput();
        Rotation += rotationDir * RotationSpeed * delta;
        MoveAndSlide(velocity);
    }
}
```

Here we've added two new variables to track our rotation direction and speed. Again, pressing both keys at once will cancel out and result in no rotation. The rotation is applied directly to the body's `rotation` property.

To set the velocity, we use the `Vector2.rotated()` method so that it points in the same direction as the body. `rotated()` is a useful vector function that you can use in many circumstances where you would otherwise need to apply trigonometric functions.

6.6.5 Rotation + Movement (mouse)

This style of movement is a variation of the previous one. This time, the direction is set by the mouse position instead of the keyboard. The character will always "look at" the mouse pointer. The forward/back inputs remain the same, however.

GDScript

C#

```
extends KinematicBody2D

export (int) var speed = 200

var velocity = Vector2()
```

(continues on next page)

(continued from previous page)

```
func get_input():
    look_at(get_global_mouse_position())
    velocity = Vector2()
    if Input.is_action_pressed('down'):
        velocity = Vector2(-speed, 0).rotated(rotation)
    if Input.is_action_pressed('up'):
        velocity = Vector2(speed, 0).rotated(rotation)

func _physics_process(delta):
    get_input()
    move_and_slide(velocity)
```

```
using Godot;
using System;

public class Movement : KinematicBody2D
{
    [Export] public int Speed = 200;

    Vector2 velocity = new Vector2();

    public void GetInput()
    {
        LookAt(GetGlobalMousePosition());
        velocity = new Vector2();
        if (Input.IsActionPressed("down"))
        {
            velocity = new Vector2(-Speed, 0).Rotated(Rotation);
        }
        if (Input.IsActionPressed("up"))
        {
            velocity = new Vector2(Speed, 0).Rotated(Rotation);
        }
        velocity = velocity.Normalized() * Speed;
    }

    public override void _PhysicsProcess(float delta)
    {
        GetInput();
        MoveAndSlide(velocity);
    }
}
```

Here we're using the *Node2D* `look_at()` method to point the player towards a given position. Without this function, you could get the same effect by setting the angle like this:

GDScript

C#

```
rotation = get_global_mouse_position().angle_to_point(position)
```

```
var rotation = GetGlobalMousePosition().AngleToPoint(Position);
```

6.6.6 Click-and-Move

This last example uses only the mouse to control the character. Clicking on the screen will cause the player to move to the target location.

GDScript

C#

```
extends KinematicBody2D

export (int) var speed = 200

var target = Vector2()
var velocity = Vector2()

func _input(event):
    if event.is_action_pressed('click'):
        target = get_global_mouse_position()

func _physics_process(delta):
    velocity = (target - position).normalized() * speed
    # rotation = velocity.angle()
    if (target - position).length() > 5:
        move_and_slide(velocity)
```

```
using Godot;
using System;

public class Movement : KinematicBody2D
{
    [Export] public int Speed = 200;

    Vector2 target = new Vector2();
    Vector2 velocity = new Vector2();

    public override void _Input(InputEvent @event)
    {
        if (@event.IsActionPressed("click"))
        {
            target = GetGlobalMousePosition();
        }
    }

    public override void _PhysicsProcess(float delta)
    {
        velocity = (target - Position).Normalized() * Speed;
        // Rotation = velocity.Angle();
        if ((target - Position).Length() > 5)
        {
            MoveAndSlide(velocity);
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Note the `length()` check we make prior to movement. Without this test, the body would "jitter" upon reaching the target position, as it moves slightly past the position and tries to move back, only to move too far and repeat.

Uncommenting the `rotation` line will also turn the body to point in its direction of motion if you prefer.

Tip: This technique can also be used as the basis of a "following" character. The `target` position can be that of any object you want to move to.

6.6.7 Summary

You may find these code samples useful as starting points for your own projects. Feel free to use them and experiment with them to see what you can make.

You can download this sample project here: `2D_movement_demo.zip`

CHAPTER 7

3D

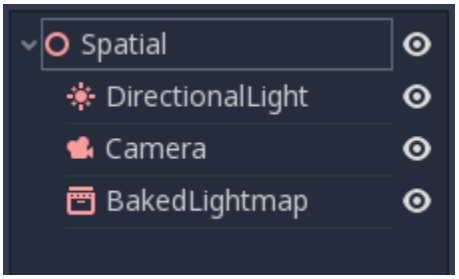
7.1 Introduction to 3D

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 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.

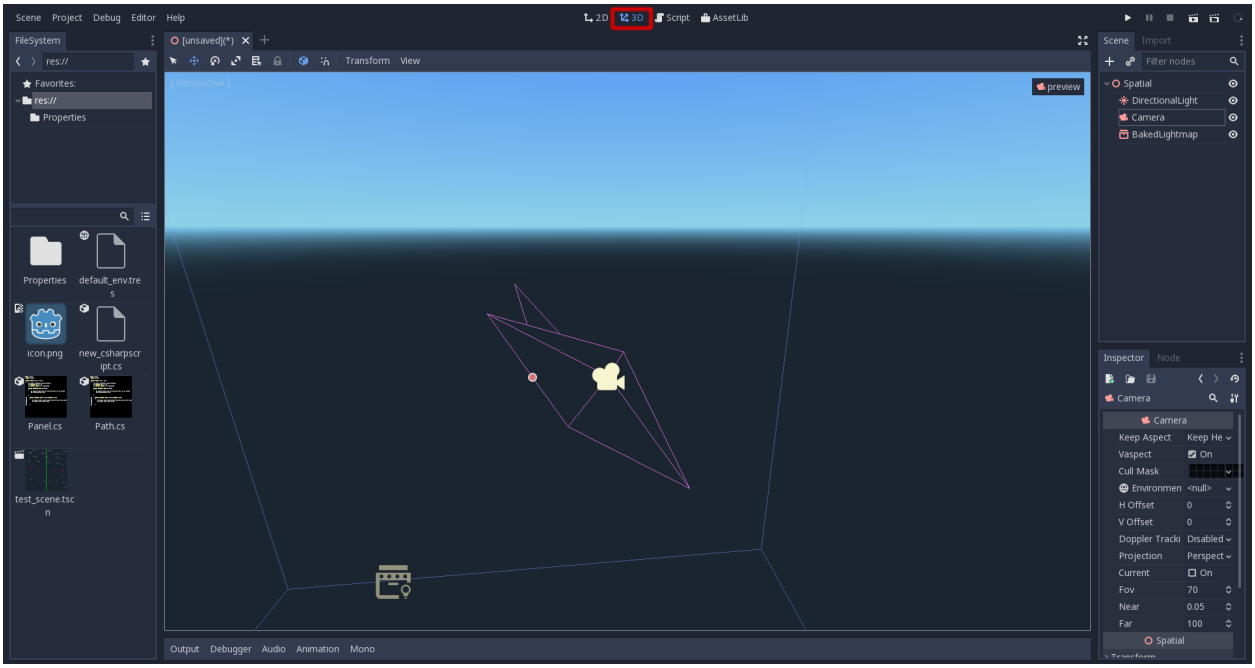
The disadvantage is, of course, that added complexity and reduced performance in comparison to plain 2D, as well as the lack of reference of working in pixels.

7.1.3 Environment

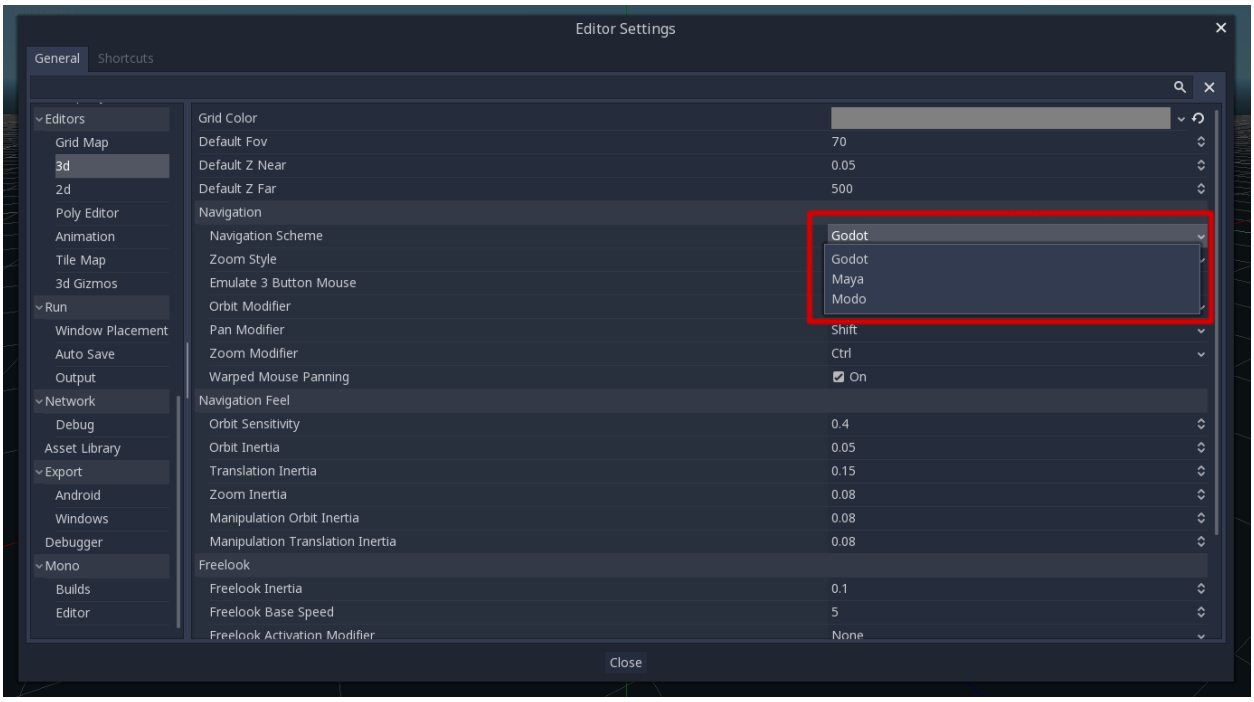
Besides editing a scene, it is often common to edit the environment. Godot provides a *WorldEnvironment* node that allows changing the background color, mode (as in, put a skybox), and applying several types of built-in post-processing effects. Environments can also be overridden in the Camera.

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).

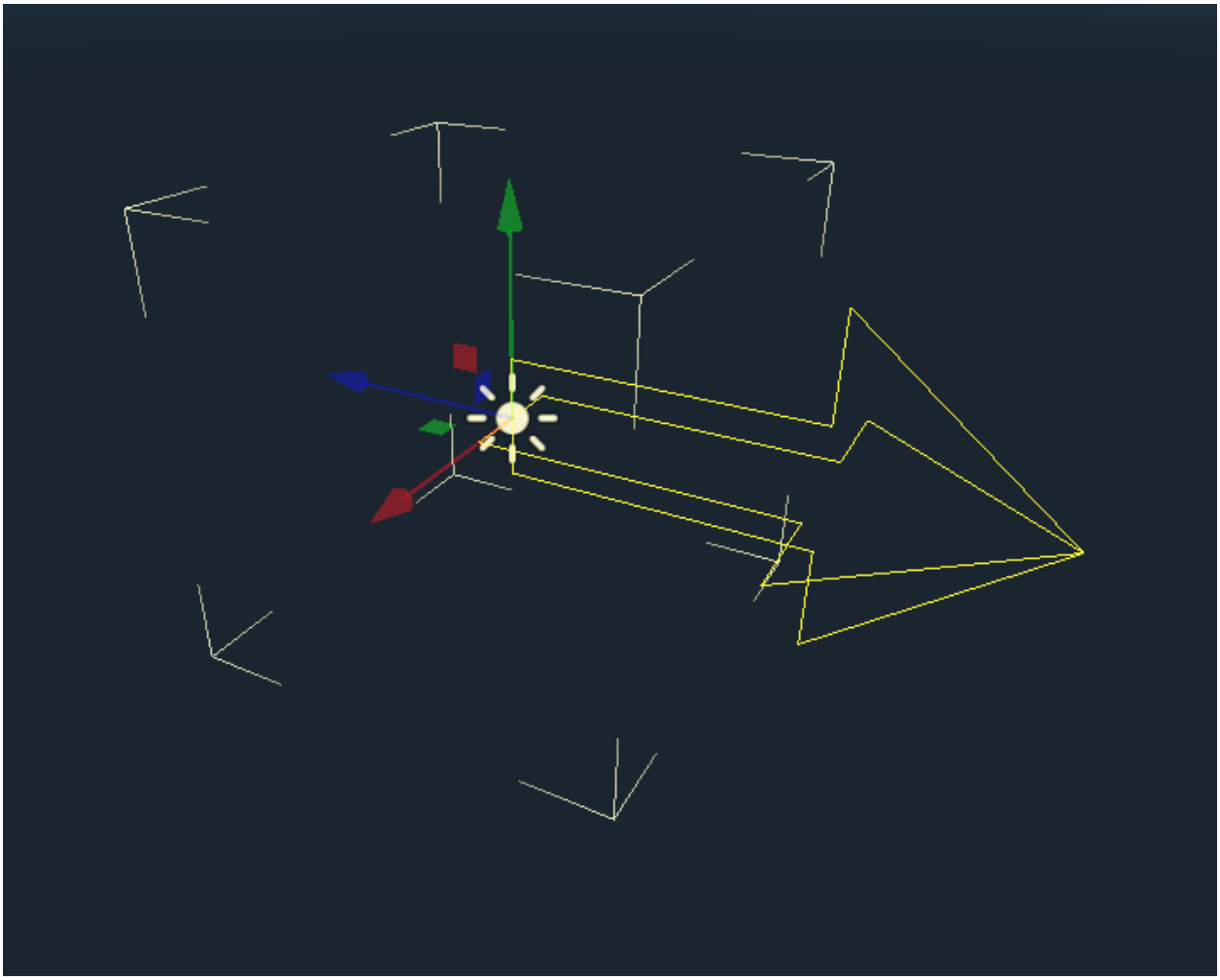
When working with 3D assets, it's always best to work in the correct scale (set your DCC to metric). Godot allows scaling post-import and, while this works in most cases, in rare situations it may introduce floating point precision issues (and thus, glitches or artifacts) in delicate areas such as rendering 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.).

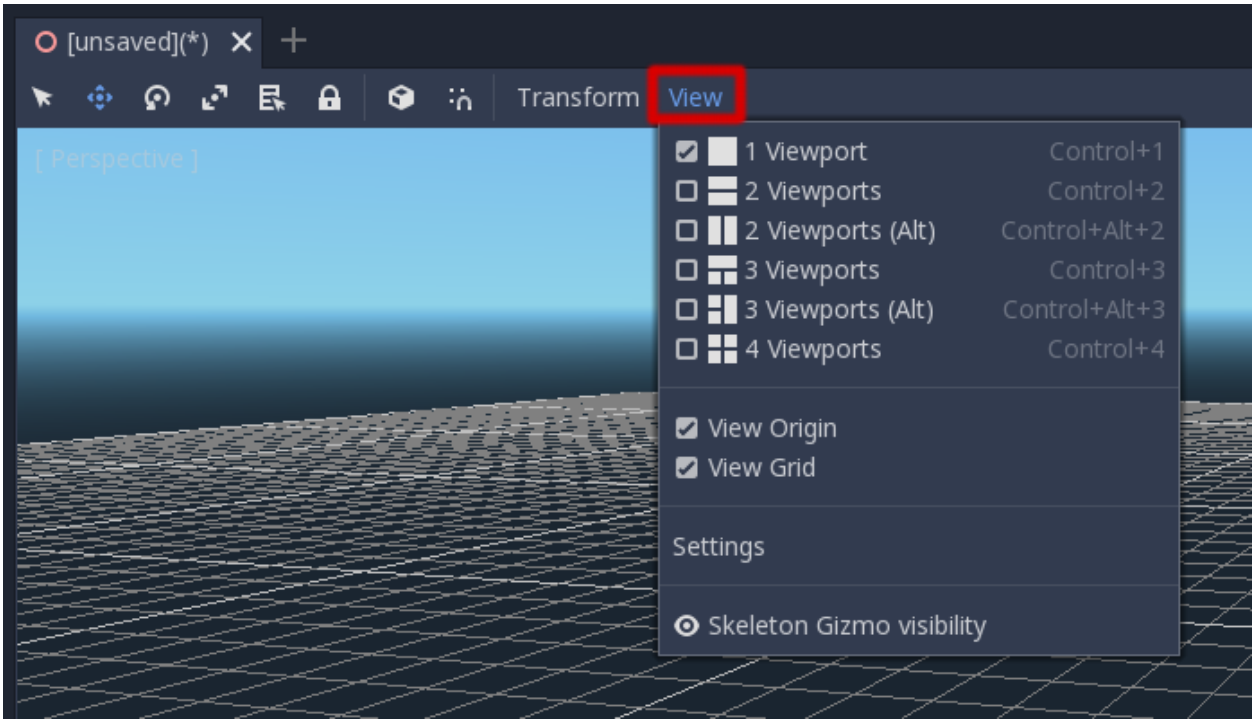


Some useful keybindings:

- 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.

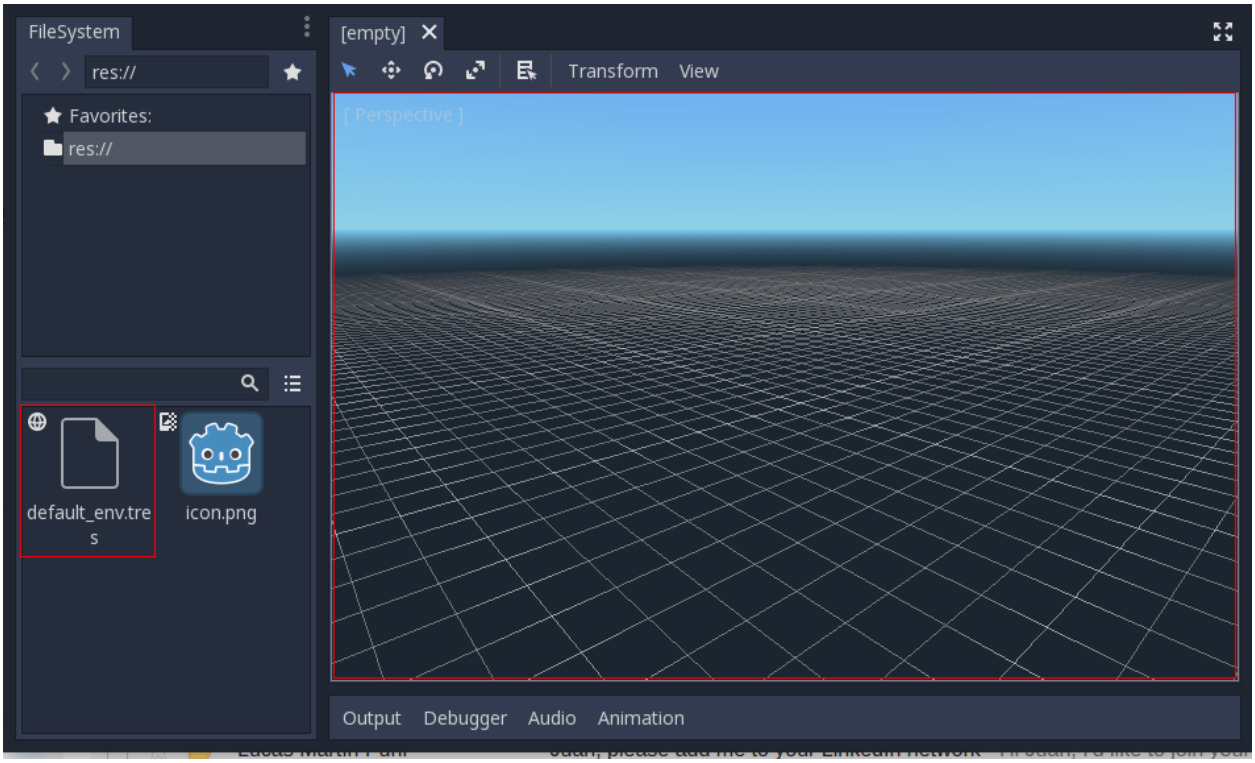
View menu

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

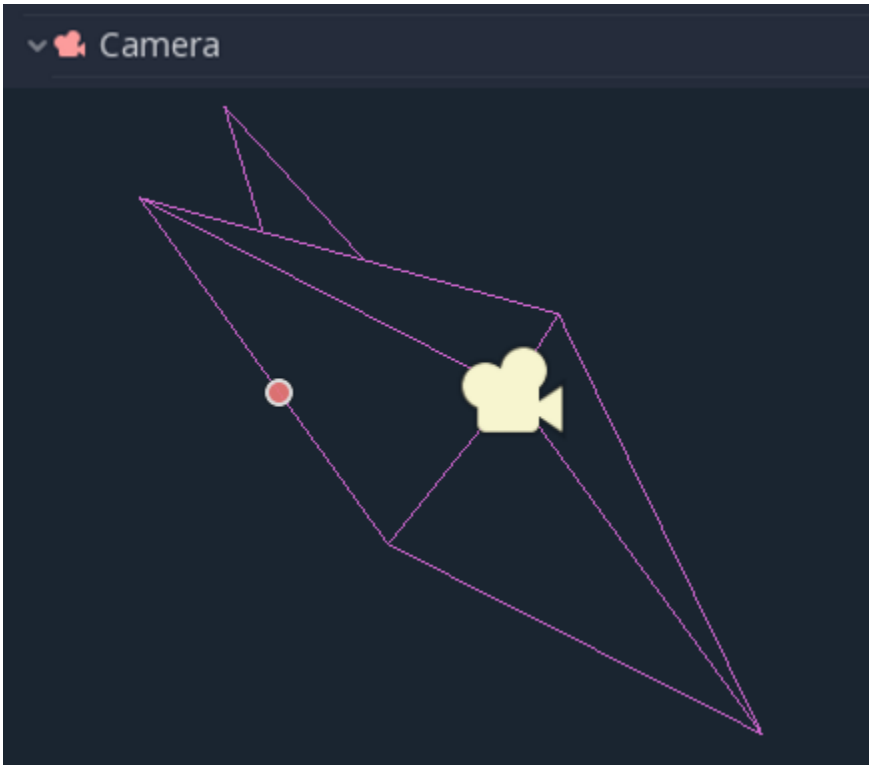
When created from the Project Manager, the 3D environment has a default sky.



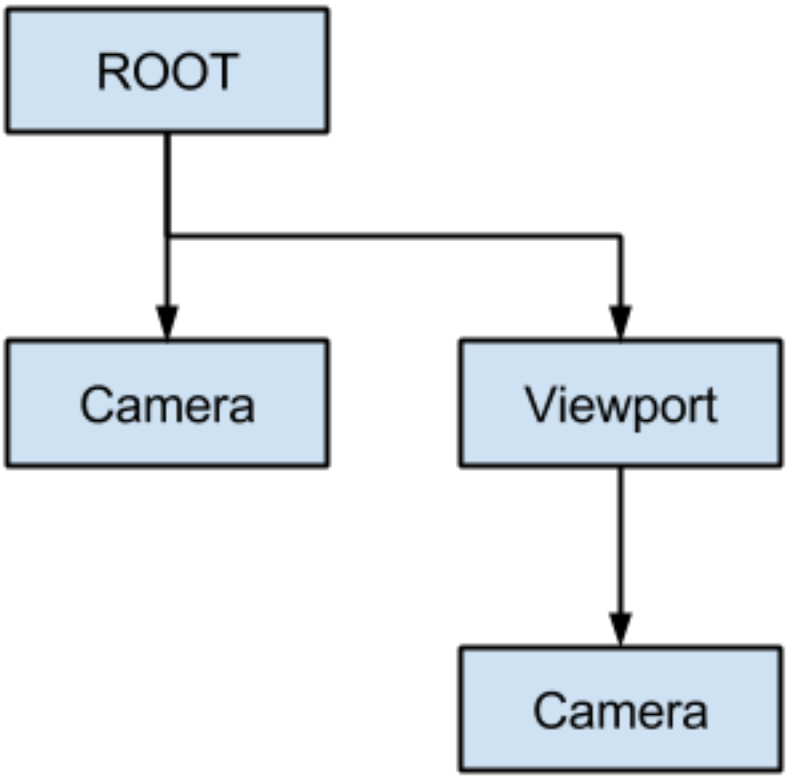
Given how physically based rendering works, it is advised to always try to work with a default environment in order to provide indirect and reflected light to your objects.

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).

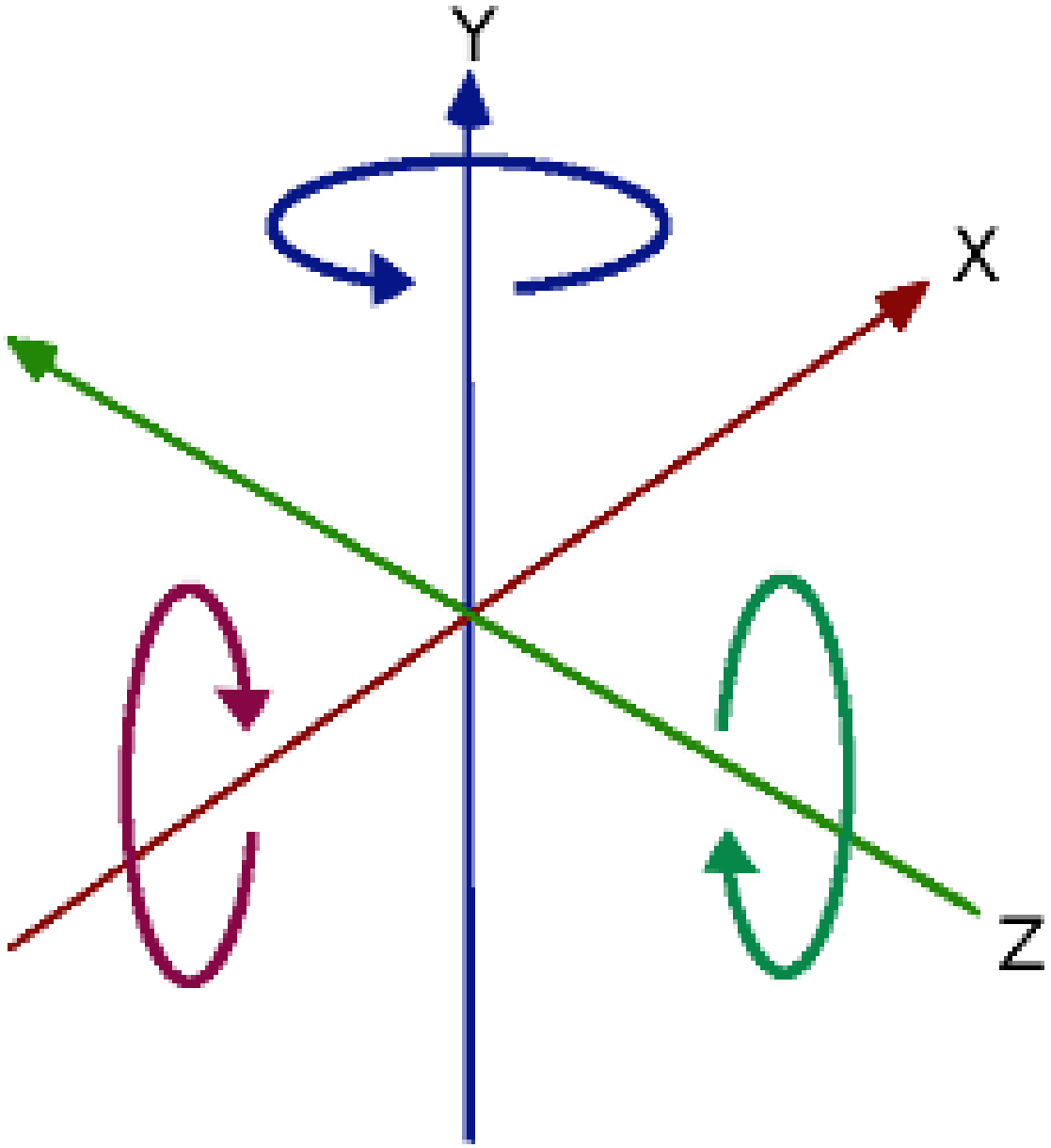
7.2 Using 3D transforms in Godot

7.2.1 Introduction

If you have never made 3D games before, working with rotations in three dimensions can be confusing at first. Coming from 2D, the natural way of thinking is along the lines of *"Oh, it's just like rotating in 2D, except now rotations happen in X, Y and Z"*.

At first this seems easy and for simple games, this way of thinking may even be enough. Unfortunately, it's often incorrect.

Angles in three dimensions are most commonly referred to as “Euler Angles”.



Euler angles were introduced by mathematician Leonhard Euler in the early 1700s.

This way of representing 3D rotations was groundbreaking at the time, but it has several shortcomings when used in game development (which is to be expected from a guy with a funny hat). The idea of this document is to explain why,

as well as outlining best practices for dealing with transforms when programming 3D games.

7.2.2 Problems of Euler Angles

While it may seem intuitive that each axis has a rotation, the truth is that it's just not practical.

Axis Order

The main reason for this is that there isn't a *unique* way to construct an orientation from the angles. There isn't a standard mathematical function that takes all the angles together and produces an actual 3D rotation. The only way an orientation can be produced from angles is to rotate the object angle by angle, in an *arbitrary order*.

This could be done by first rotating in *X*, then *Y* and then in *Z*. Alternatively, you could first rotate in *Y*, then in *Z* and finally in *X*. Anything works, but depending on the order, the final orientation of the object will *not necessarily be the same*. Indeed, this means that there are several ways to construct an orientation from 3 different angles, depending on *the order of the rotations*.

Following is a visualization of rotation axes (in X,Y,Z order) in a gimbal (from Wikipedia). As you can see, the orientation of each axis depends on the rotation of the previous one:

You may be wondering how this affects you. Let's look at a practical example:

Imagine you are working on a first-person controller (FPS game). Moving the mouse left and right controls your view angle parallel to the ground, while moving it up and down moves the player's view up and down.

In this case to achieve the desired effect, rotation must be applied first in the *Y* axis ("up" in this case, since Godot uses a "Y-Up" orientation), followed by rotation in the *X* axis.

If we were to apply rotation in the *X* axis first, and then in *Y*, the effect would be undesired:

Depending on the type of game or effect desired, the order in which you want axis rotations to be applied may differ. Therefore, applying rotations in X, Y, and Z is not enough: you also need a *rotation order*.

Interpolation

Another problem with using Euler angles is interpolation. Imagine you want to transition between two different camera or enemy positions (including rotations). One logical way to approach this is to interpolate the angles from one position to the next. One would expect it to look like this:

But this does not always have the expected effect when using angles:

The camera actually rotated the opposite direction!

There are a few reasons this may happen:

- Rotations don't map linearly to orientation, so interpolating them does not always result in the shortest path (i.e., to go from `270` to `0` degrees is not the same as going from `270` to `360`, even though the angles are equivalent).
- Gimbal lock is at play (first and last rotated axis align, so a degree of freedom is lost). See Wikipedia's page on Gimbal Lock for a detailed explanation of this problem.

Say no to Euler Angles

The result of all this is that you should **not use** the `rotation` property of *Spatial* nodes in Godot for games. It's there to be used mainly in the editor, for coherence with the 2D engine, and for simple rotations (generally just one axis, or even two in limited cases). As much as you may be tempted, don't use it.

Instead, there is a better way to solve your rotation problems.

7.2.3 Introducing Transforms

Godot uses the *Transform* datatype for orientations. Each *Spatial* node contains a `transform` property which is relative to the parent's transform, if the parent is a Spatial-derived type.

It is also possible to access the world coordinate transform via the `global_transform` property.

A transform has a *Basis* (transform.basis sub-property), which consists of three *Vector3* vectors. These are accessed via the `transform.basis` property and can be accessed directly by `transform.basis.x`, `transform.basis.y`, and `transform.basis.z`. Each vector points in the direction its axis has been rotated, so they effectively describe the node's total rotation. The scale (as long as it's uniform) can be also be inferred from the length of the axes. A *basis* can also be interpreted as a 3x3 matrix and used as `transform.basis[x][y]`.

A default basis (unmodified) is akin to:

GDScript

C#

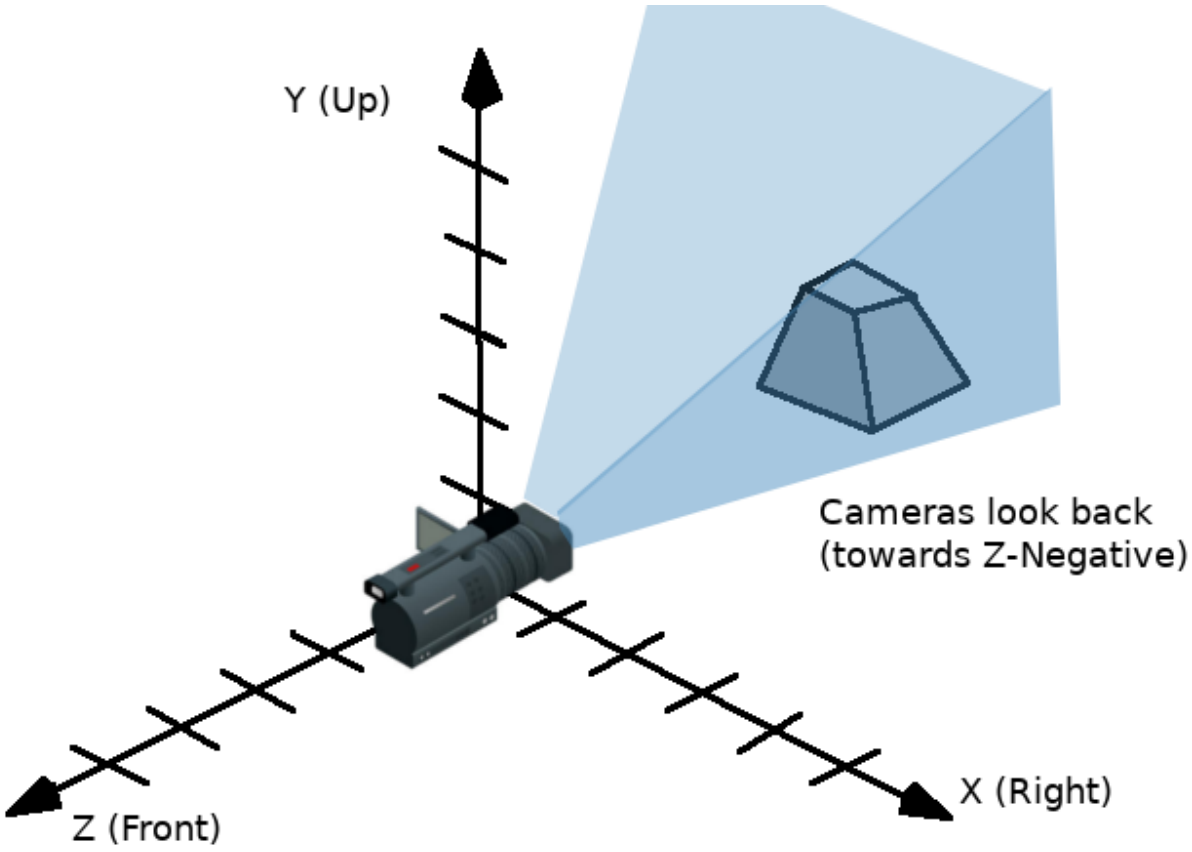
```
var basis = Basis()
# Contains the following default values:
basis.x = Vector3(1, 0, 0) # Vector pointing along the X axis
basis.y = Vector3(0, 1, 0) # Vector pointing along the Y axis
basis.z = Vector3(0, 0, 1) # Vector pointing along the Z axis
```

```
// Due to technical limitations on structs in C# the default
// constructor will contain zero values for all fields.
var defaultBasis = new Basis();
GD.Print(defaultBasis); // prints: ((0, 0, 0), (0, 0, 0), (0, 0, 0))

// Instead we can use the Identity property.
var identityBasis = Basis.Identity;
GD.Print(identityBasis.x); // prints: (1, 0, 0)
GD.Print(identityBasis.y); // prints: (0, 1, 0)
GD.Print(identityBasis.z); // prints: (0, 0, 1)

// The Identity basis is equivalent to:
var basis = new Basis(Vector3.Right, Vector3.Up, Vector3.Back);
GD.Print(basis); // prints: ((1, 0, 0), (0, 1, 0), (0, 0, 1))
```

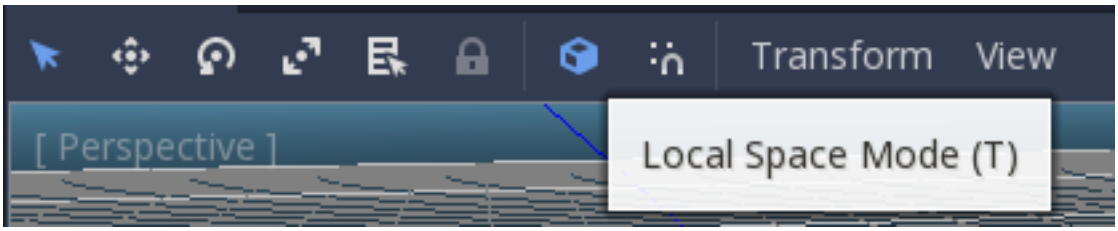
This is also an analog to a 3x3 identity matrix.

Following the OpenGL convention, `X` is the *Right* axis, `Y` is the *Up* axis and `Z` is the *Forward* axis.

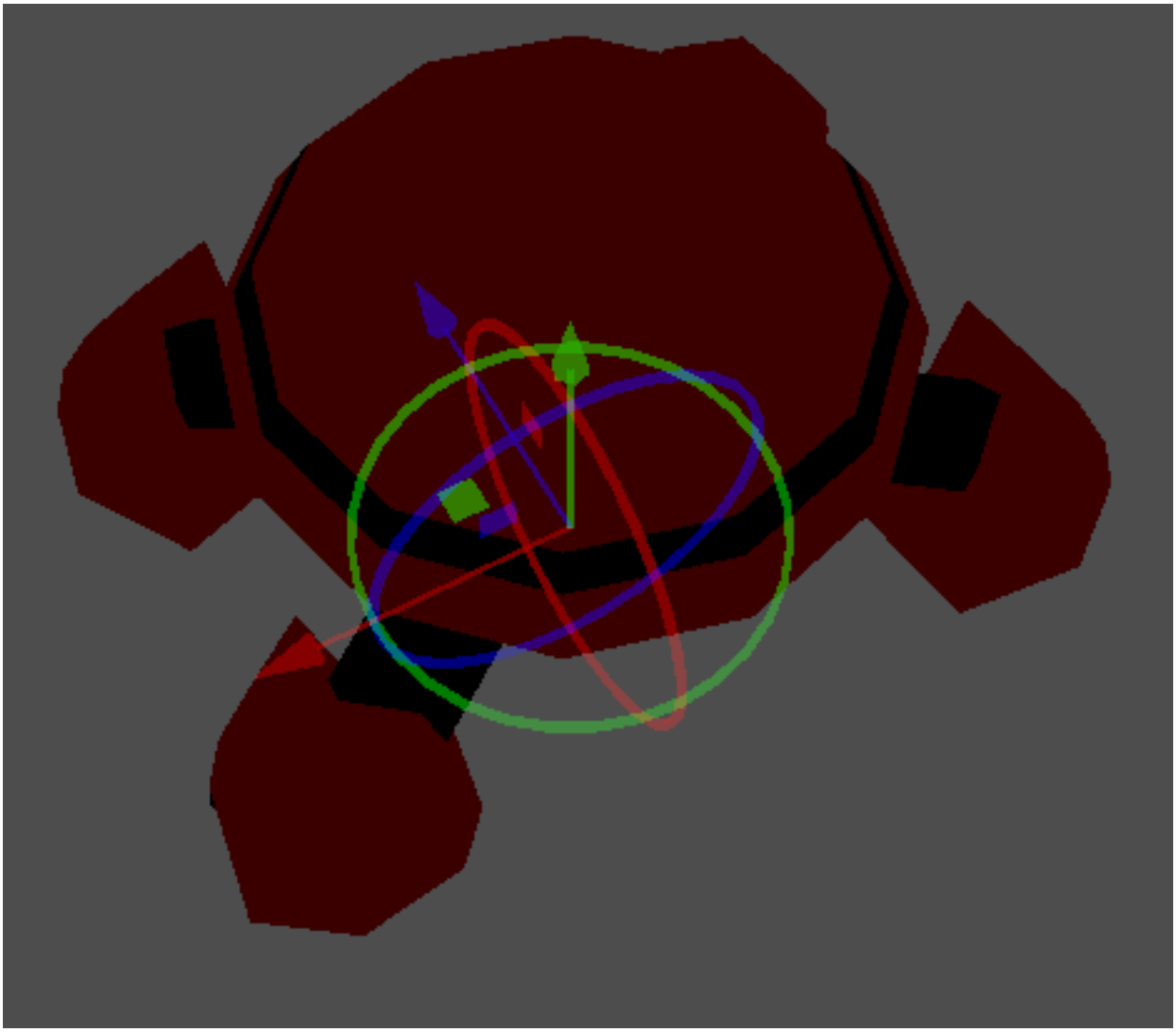
Together with the *basis*, a transform also has an *origin*. This is a *Vector3* specifying how far away from the actual origin `(0, 0, 0)` this transform is. Combining the *basis* with the *origin*, a *transform* efficiently represents a unique translation, rotation, and scale in space.



One way to visualize a transform is to look at an object's 3D gizmo while in "local space" mode.



The gizmo's arrows show the `X`, `Y`, and `Z` axes (in red, green, and blue respectively) of the basis, while gizmo's center is at the object's origin.



For more information on the mathematics of vectors and transforms, please read the *Vector math* tutorials.

Manipulating Transforms

Of course, transforms are not as straightforward to manipulate as angles and have problems of their own.

It is possible to rotate a transform, either by multiplying its basis by another (this is called accumulation), or by using the rotation methods.

GDScript

C#

```
# Rotate the transform about the X axis
transform.basis = Basis(Vector3(1, 0, 0), PI) * transform.basis
# shortened
transform.basis = transform.basis.rotated(Vector3(1, 0, 0), PI)
```

```
// rotate the transform about the X axis
transform.basis = new Basis(Vector3.Right, Mathf.Pi) * transform.basis;
// shortened
transform.basis = transform.basis.Rotated(Vector3.Right, Mathf.Pi);
```

A method in Spatial simplifies this:

GDScript

C#

```
# Rotate the transform in X axis
rotate(Vector3(1, 0, 0), PI)
# shortened
rotate_x(PI)
```

```
// Rotate the transform about the X axis
Rotate(Vector3.Right, Mathf.Pi);
// shortened
RotateX(Mathf.Pi);
```

This rotates the node relative to the parent node.

To rotate relative to object space (the node's own transform) use the following:

GDScript

C#

```
# Rotate locally
rotate_object_local(Vector3(1, 0, 0), PI)
```

```
// Rotate locally
RotateObjectLocal(Vector3.Right, Mathf.Pi);
```

Precision Errors

Doing successive operations on transforms will result in a loss of precision due to floating-point error. This means the scale of each axis may no longer be exactly `1.0`, and they may not be exactly `90` degrees from each other.

If a transform is rotated every frame, it will eventually start deforming over time. This is unavoidable.

There are two different ways to handle this. The first is to *orthonormalize* the transform after some time (maybe once per frame if you modify it every frame):

GDScript

C#

```
transform = transform.orthonormalized()
```

```
transform = transform.Orthonormalized();
```

This will make all axes have `1.0` length again and be `90` degrees from each other. However, any scale applied to the transform will be lost.

It is recommended you don't scale nodes that are going to be manipulated. Scale their children nodes instead (such as MeshInstance). If you absolutely must scale the node, then re-apply it at the end:

GDScript

C#

```
transform = transform.orthonormalized()
transform = transform.scaled(scale)
```

```
transform = transform.Orthonormalized();
transform = transform.Scaled(scale);
```

Obtaining Information

You might be thinking at this point: **"Ok, but how do I get angles from a transform?"**. The answer again is: you don't. You must do your best to stop thinking in angles.

Imagine you need to shoot a bullet in the direction your player is facing. Just use the forward axis (commonly `Z` or `-Z`).

GDScript

C#

```
bullet.transform = transform
bullet.speed = transform.basis.z * BULLET_SPEED
```

```
bullet.Transform = transform;
bullet.LinearVelocity = transform.basis.z * BulletSpeed;
```

Is the enemy looking at the player? Use the dot product for this (see the *Vector math* tutorial for an explanation of the dot product):

GDScript

C#

```
# Get the direction vector from player to enemy
var direction = enemy.transform.origin - player.transform.origin
if direction.dot(enemy.transform.basis.z) > 0:
    enemy.im_watching_you(player)
```

```
// Get the direction vector from player to enemy
Vector3 direction = enemy.Transform.origin - player.Transform.origin;
if (direction.Dot(enemy.Transform.basis.z) > 0)
{
    enemy.ImWatchingYou(player);
}
```

Strafe left:

GDScript

C#

```
# Remember that +X is right
if Input.is_action_pressed("strafe_left"):
    translate_object_local(-transform.basis.x)
```

```
// Remember that +X is right
if (Input.IsActionPressed("strafe_left"))
{
    TranslateObjectLocal(-Transform.basis.x);
}
```

Jump:

GDScript

C#

```
# Keep in mind Y is up-axis
if Input.is_action_just_pressed("jump"):
    velocity.y = JUMP_SPEED

velocity = move_and_slide(velocity)
```

```
// Keep in mind Y is up-axis
if (Input.IsActionJustPressed("jump"))
{
    velocity.y = JumpSpeed;
}
velocity = MoveAndSlide(velocity);
```

All common behaviors and logic can be done with just vectors.

Setting Information

There are, of course, cases where you want to set information to a transform. Imagine a first person controller or orbiting camera. Those are definitely done using angles, because you *do want* the transforms to happen in a specific order.

For such cases, keep the angles and rotations *outside* the transform and set them every frame. Don't try retrieve them and re-use them because the transform is not meant to be used this way.

Example of looking around, FPS style:

GDScript

C#

```
# accumulators
var rot_x = 0
var rot_y = 0

func _input(event):
    if event is InputEventMouseMotion and event.button_mask & 1:
        # modify accumulated mouse rotation
        rot_x += event.relative.x * LOOKAROUND_SPEED
        rot_y += event.relative.y * LOOKAROUND_SPEED
        transform.basis = Basis() # reset rotation
        rotate_object_local(Vector3(0, 1, 0), rot_x) # first rotate in Y
        rotate_object_local(Vector3(1, 0, 0), rot_y) # then rotate in X
```

```
// accumulators
private float _rotationX = 0f;
```

(continues on next page)

(continued from previous page)

```
private float _rotationY = 0f;

public override void _Input(InputEvent @event)
{
    var mouseMotion = @event as InputEventMouseMotion;
    if (mouseMotion != null)
    {
        // modify accumulated mouse rotation
        _rotationX += mouseMotion.Relative.x * LookAroundSpeed;
        _rotationY += mouseMotion.Relative.y * LookAroundSpeed;

        // reset rotation
        Transform transform = Transform;
        transform.basis = Basis.Identity;
        Transform = transform;

        RotateObjectLocal(Vector3.Up, _rotationX); // first rotate about Y
        RotateObjectLocal(Vector3.Right, _rotationY); // then rotate about X
    }
}
```

As you can see, in such cases it's even simpler to keep the rotation outside, then use the transform as the *final* orientation.

Interpolating with Quaternions

Interpolating between two transforms can efficiently be done with quaternions. More information about how quaternions work can be found in other places around the Internet. For practical use, it's enough to understand that pretty much their main use is doing a closest path interpolation. As in, if you have two rotations, a quaternion will smoothly allow interpolation between them using the closest axis.

Converting a rotation to quaternion is straightforward.

GDScript

C#

```
# Convert basis to quaternion, keep in mind scale is lost
var a = Quat(transform.basis)
var b = Quat(transform2.basis)
# Interpolate using spherical-linear interpolation (SLERP).
var c = a.slerp(b,0.5) # find halfway point between a and b
# Apply back
transform.basis = Basis(c)
```

```
// Convert basis to quaternion, keep in mind scale is lost
var a = transform.basis.Quat();
var b = transform.basis.Quat();
// Interpolate using spherical-linear interpolation (SLERP).
var c = a.Slerp(b, 0.5f); // find halfway point between a and b
// Apply back
transform.basis = new Basis(c);
```

The *Quat* type reference has more information on the datatype (it can also do transform accumulation, transform points, etc. though this is used less often). If you interpolate or apply operations to quaternions many times, keep in mind they need to be eventually normalized or they also may suffer from numerical precision errors.

Quaternions are useful when doing camera/path/etc. interpolations, as the result will be always correct and smooth.

7.2.4 Transforms are your friend

For most beginners, getting used to working with transforms can take some time. However, once you get used to them, you will appreciate their simplicity and power.

Don't hesitate to ask for help on this topic in any of Godot's online communities and, once you become confident enough, please help others!

7.3 3D performance and limitations

7.3.1 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 rendered 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 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.

7.3.2 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

The 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 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 number of polygons in a model, the faster it will be rendered. This is *really* relative and depends on many factors.

On a modern PC and console, vertex cost is 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 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 simple system to change 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 the 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.

7.4 Spatial Material

7.4.1 Introduction

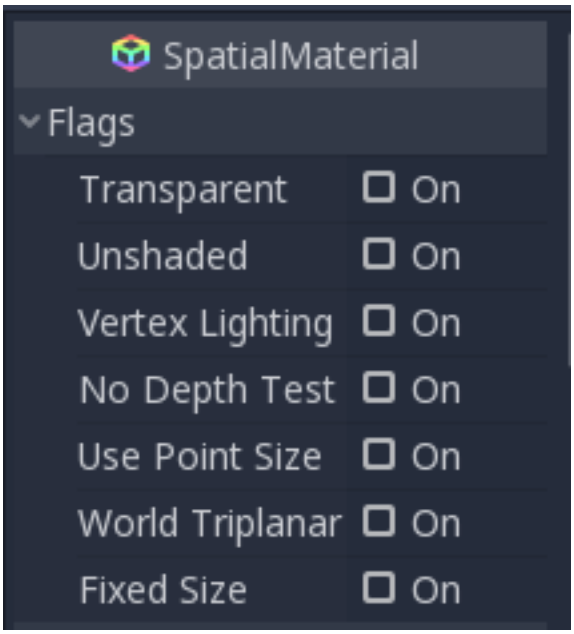
For Godot 3, instead of following the trend and focusing on shader graphs, we put in most of the work offering a default material that covers far more use cases. This replaces the old "FixedMaterial" in Godot 2.x

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.

7.4.2 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.

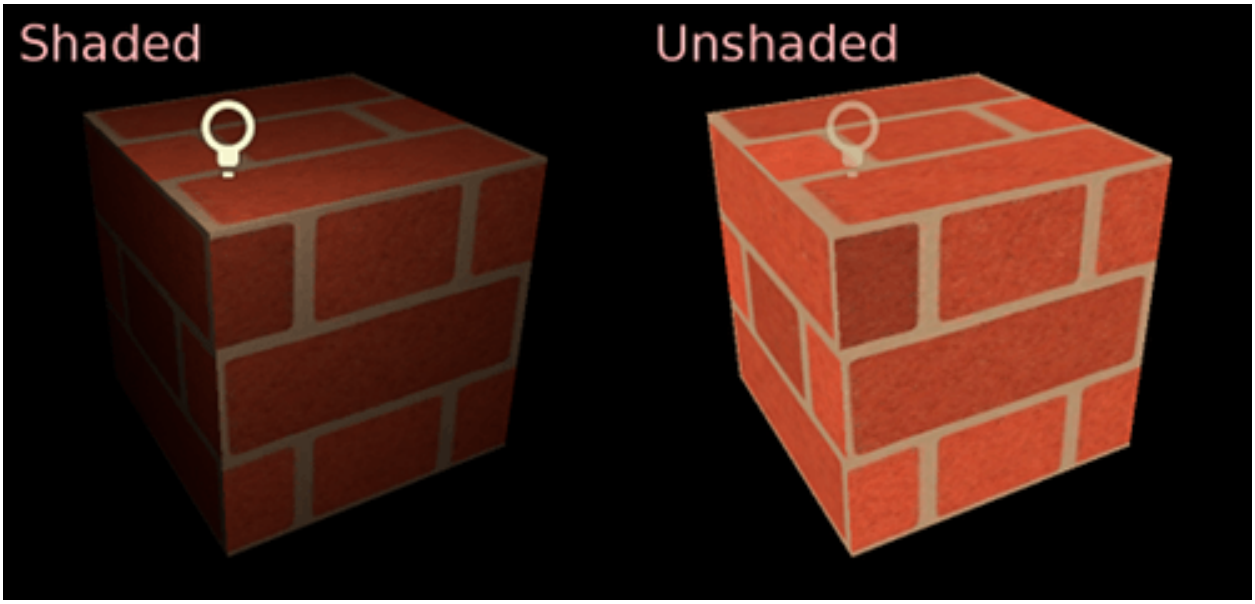
For this reason, materials in Godot are assumed opaque unless specified otherwise. The main settings that enable transparency are:

- 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.

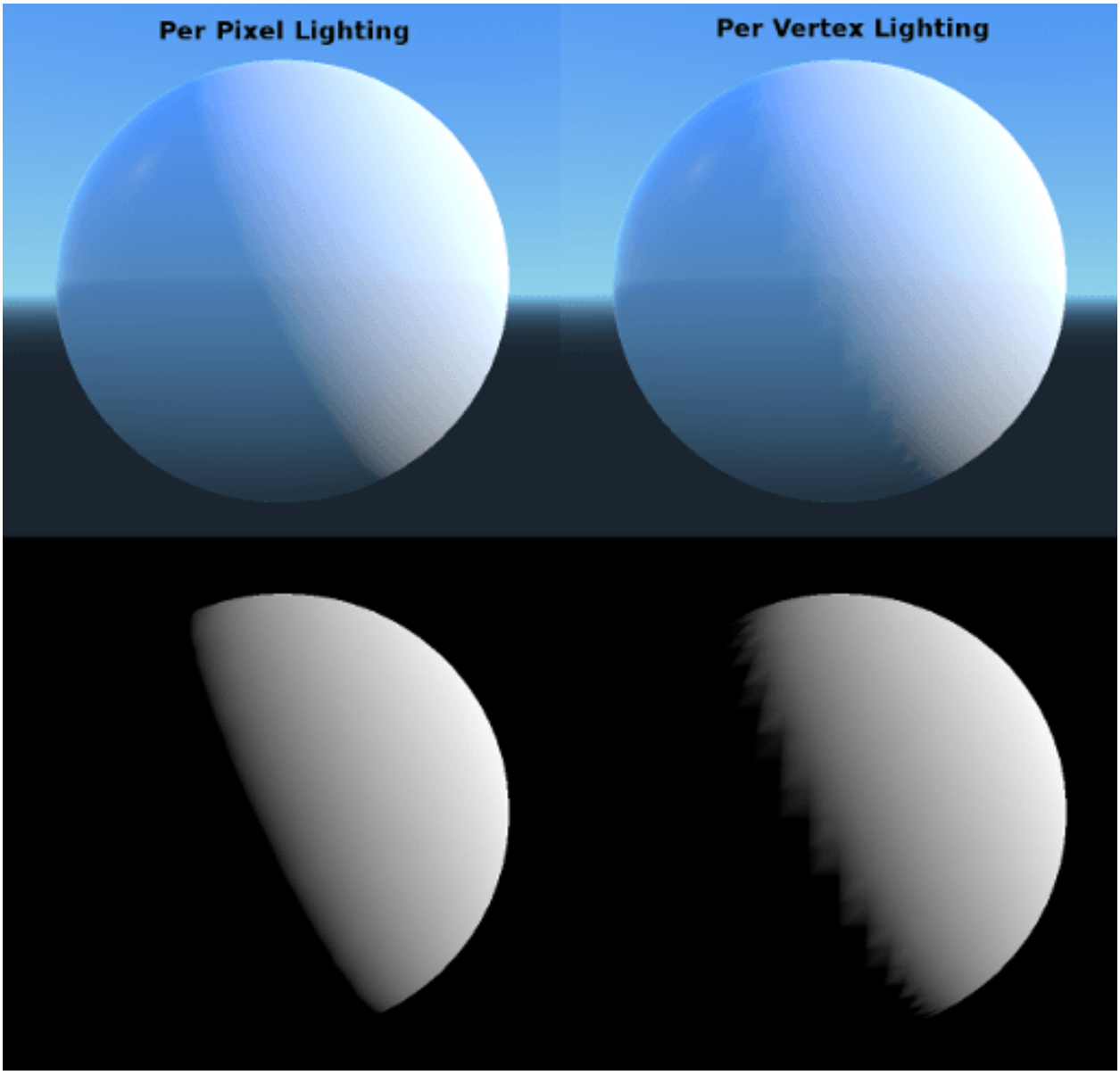


Vertex Lighting

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 low end or mobile devices, switching to vertex lighting can considerably increase rendering performance.

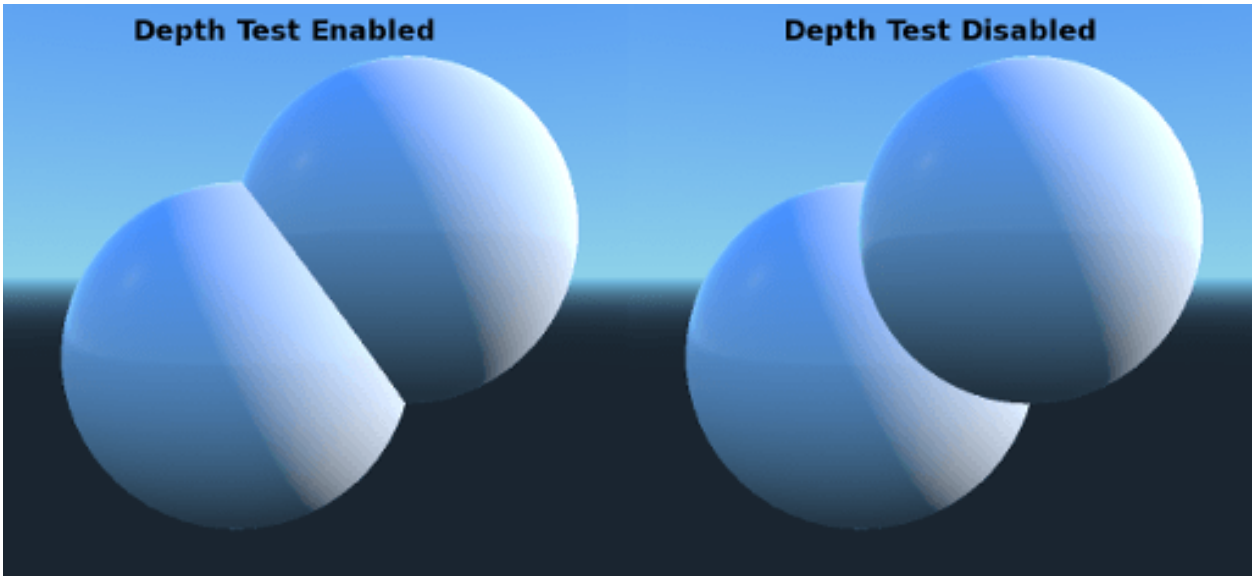


Keep in mind that, when vertex lighting is enabled, only directional lighting can produce shadows (for performance reasons).

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

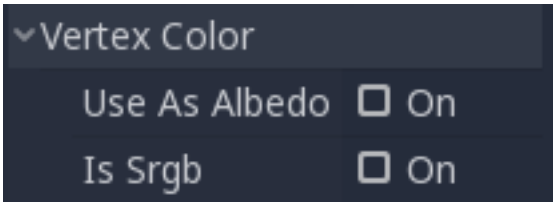
Makes the object rendered at the same size no matter the distance. This is, again, useful mostly for indicators (no depth test and high render priority) and some types of billboards.

Do Not Receive Shadows

Makes the object not receive any kind of shadow that would otherwise be cast onto it.

7.4.3 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

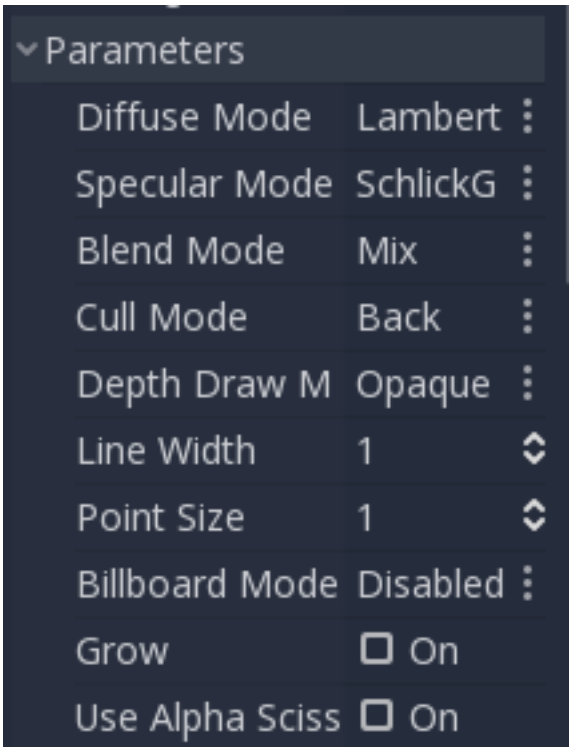
Vertex color is used as albedo color.

Is SRGB

Most 3D DCCs will likely export vertex colors as SRGB, so toggling this option on will help them look correct.

7.4.4 Parameters

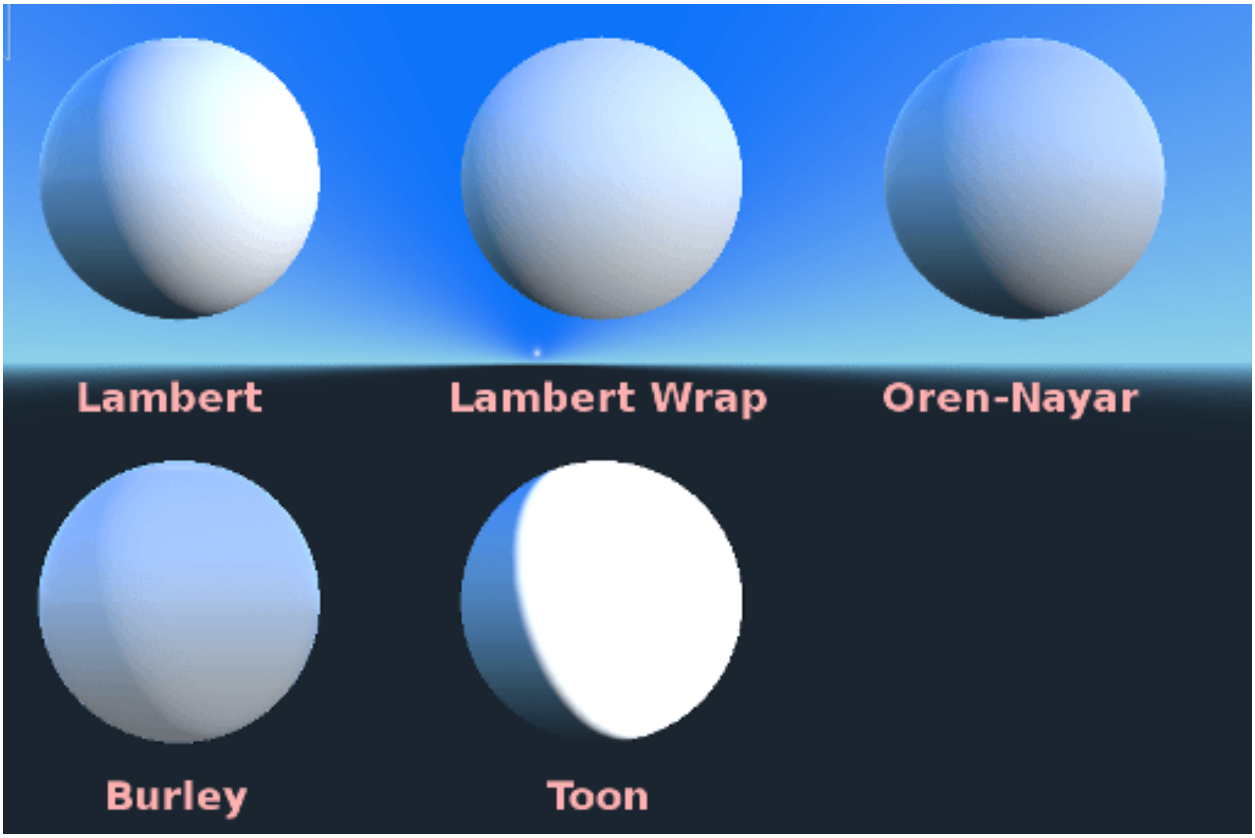
SpatialMaterial also has several configurable parameters to tweak many aspects of the rendering:



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:

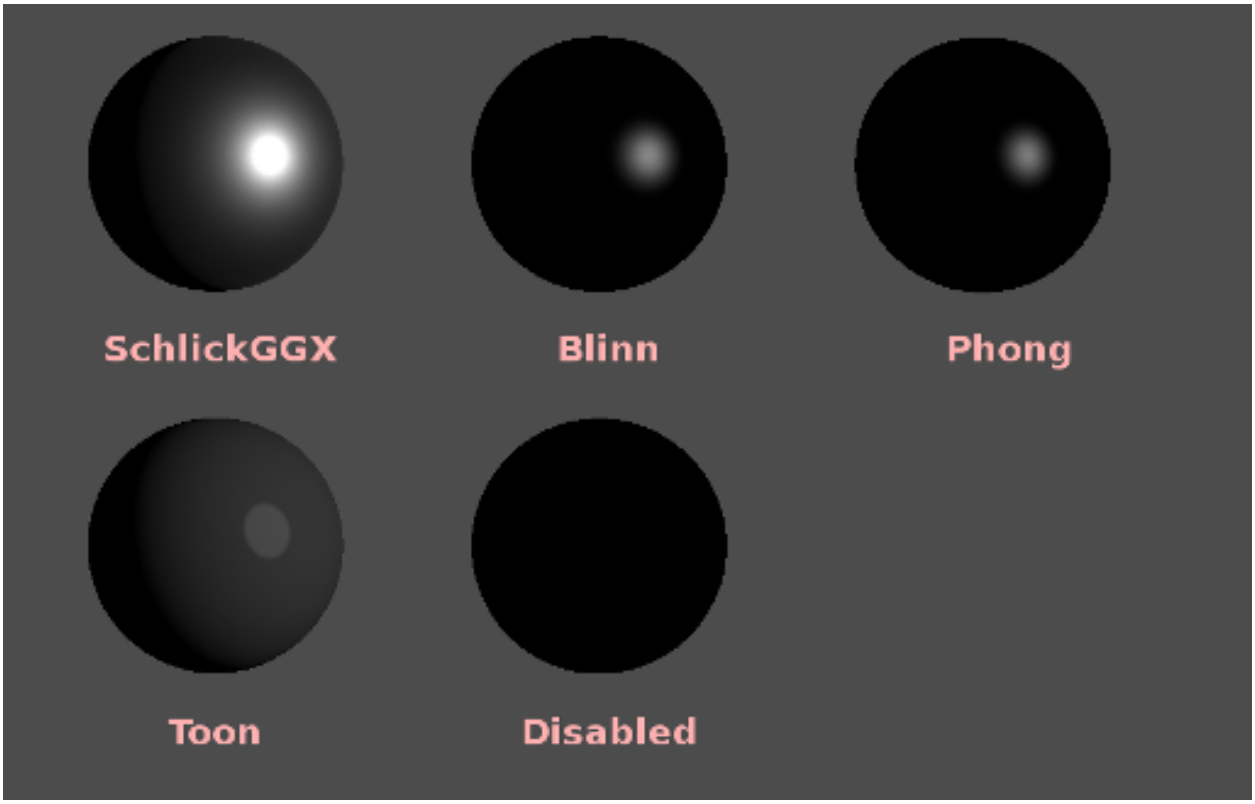
- **Burley:** Default mode, the original Disney Principled PBS diffuse algorithm.
- **Lambert:** Is not affected by roughness.
- **Lambert Wrap:** Extends Lambert to cover more than 90 degrees when roughness increases. Works great for hair and simulating cheap subsurface scattering. This implementation is energy conserving.
- **Oren Nayar:** This implementation aims to take microsurfacing into account (via roughness). Works 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.

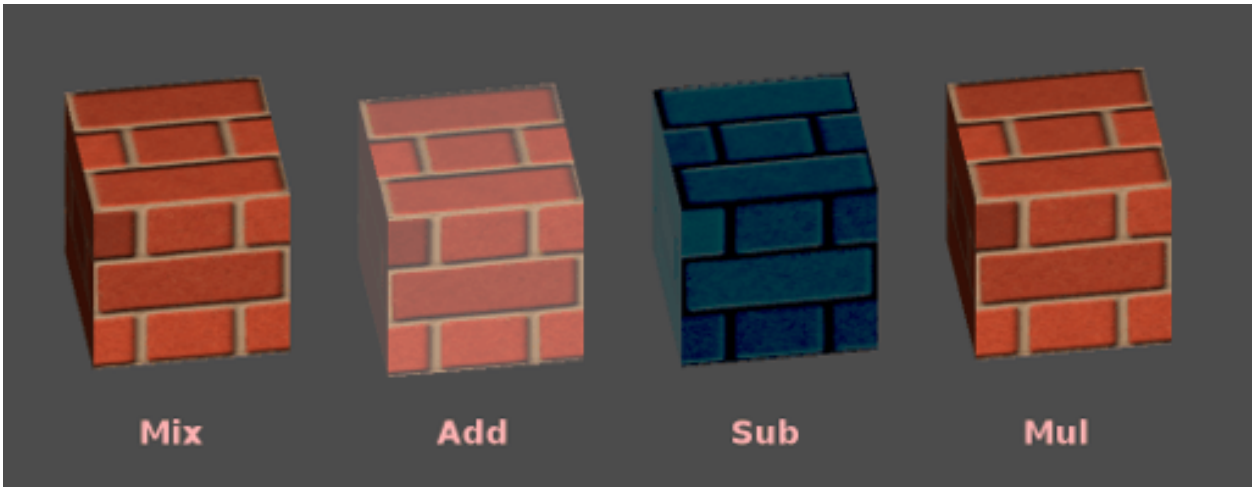
- **ShlickGGX:** The most common blob used by PBR 3D engines nowadays.
- **Blinn:** Common in previous-generation engines. Not worth using nowadays, but left here for the sake of compatibility.
- **Phong:** Same as above.
- **Toon:** Creates a toon blob, which changes size depending on roughness.
- **Disabled:** 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

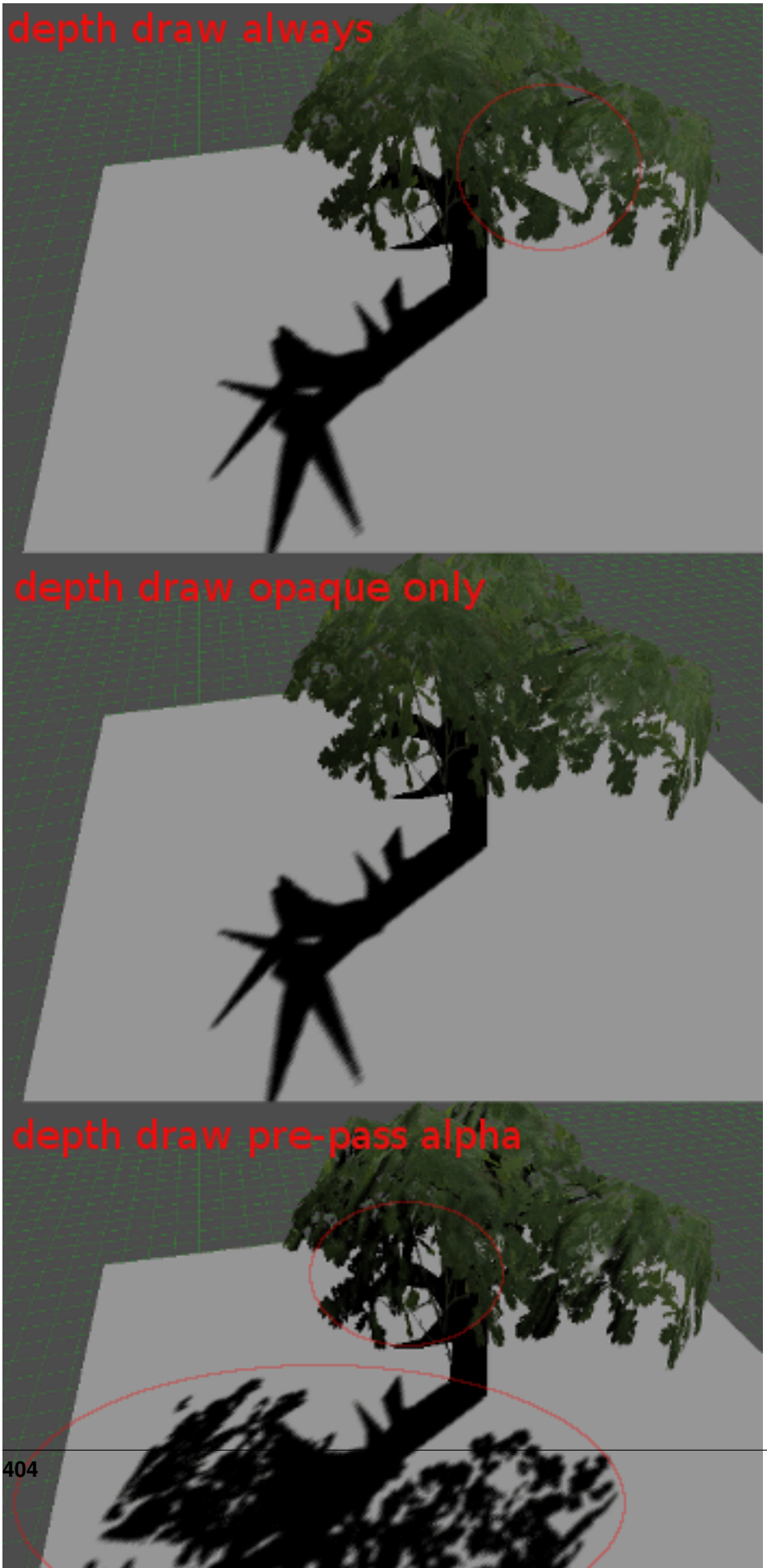
Determines which side of the object is not drawn when back-faces are rendered:

- 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.

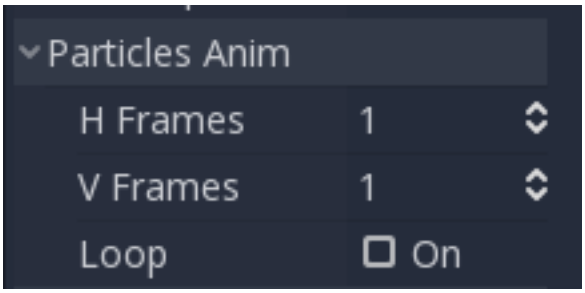
Point Size

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:

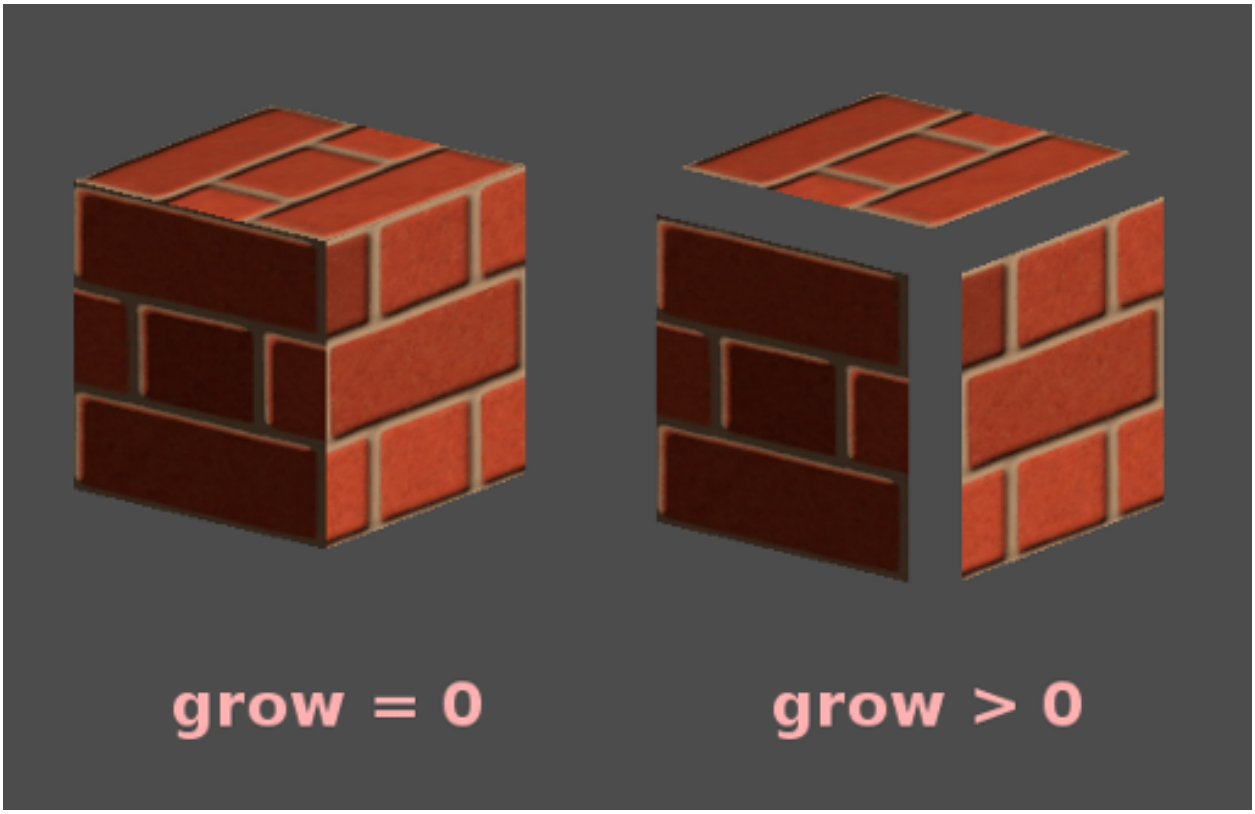
- Disabled: Billboard mode is disabled
- Enabled: Billboard mode is enabled, object -Z axis will always face the camera.
- Y-Billboard: Object X axis will always be aligned with the camera
- Particles: When using particle systems, this type of billboard is best, because it allows specifying animation options.



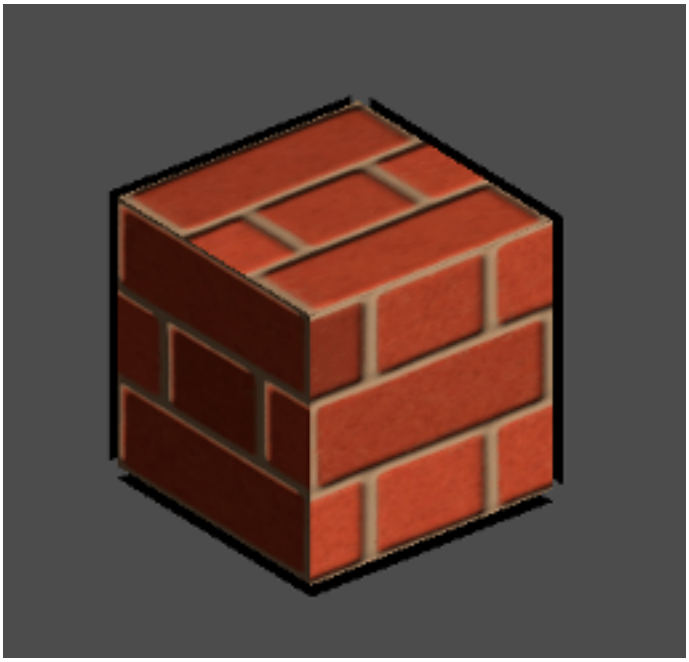
Above options are only enabled for Particle Billboard.

Grow

Grows the object vertices in the direction pointed by their normals:

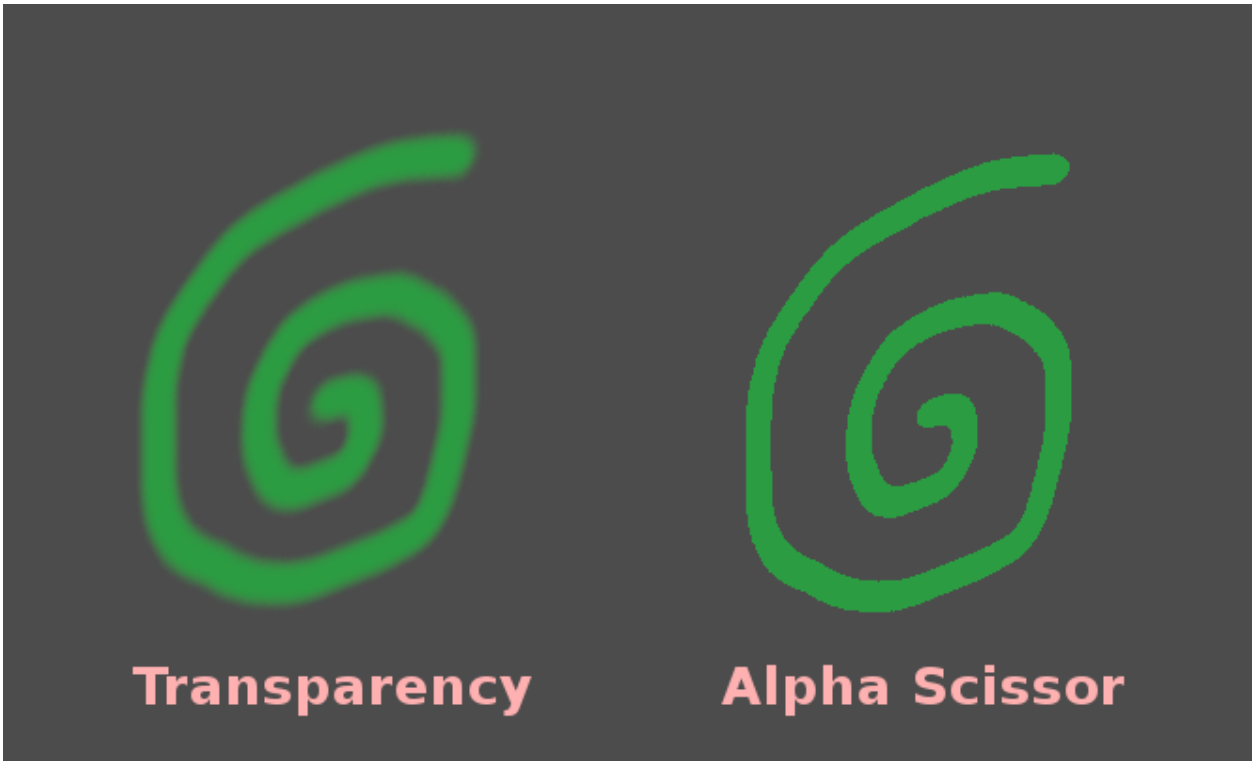


This is commonly used to create cheap outlines. Add a second material pass, make it black an unshaded, reverse culling (Cull Front), and add some grow:



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.

7.4.5 Material colors, maps and channels

Besides the parameters, what defines materials themselves are the colors, textures and channels. Godot supports a extensive list of them. 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.

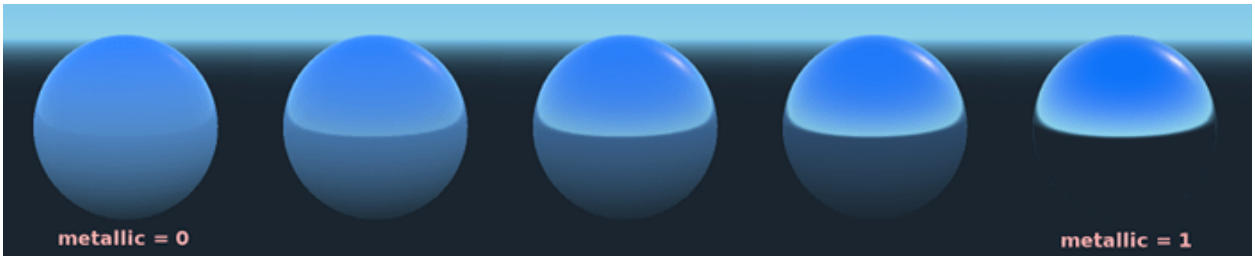
Alpha channel in albedo color and texture is also used for the object transparency. If you use a color or texture with *alpha channel*, make sure to either enable transparency or *alpha scissoring* for it to work.

Metallic

Godot uses a Metallic model over competing models due to it's 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".

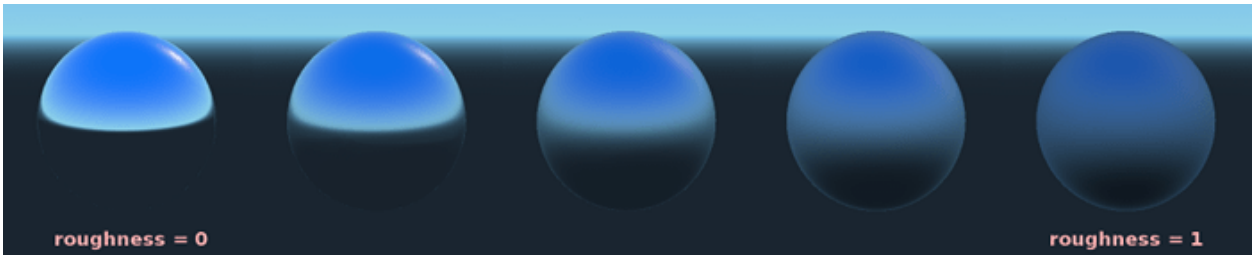
The "specular" parameter here is just a general amount of for the reflectivity (unlike *metallic*, this one is not energy conserving, so simply leave it as 0.5 and don't touch it unless you need to).

The minimum internal reflectivity is 0.04, so (just like in real life) it's impossible to make a material completely unreflective.



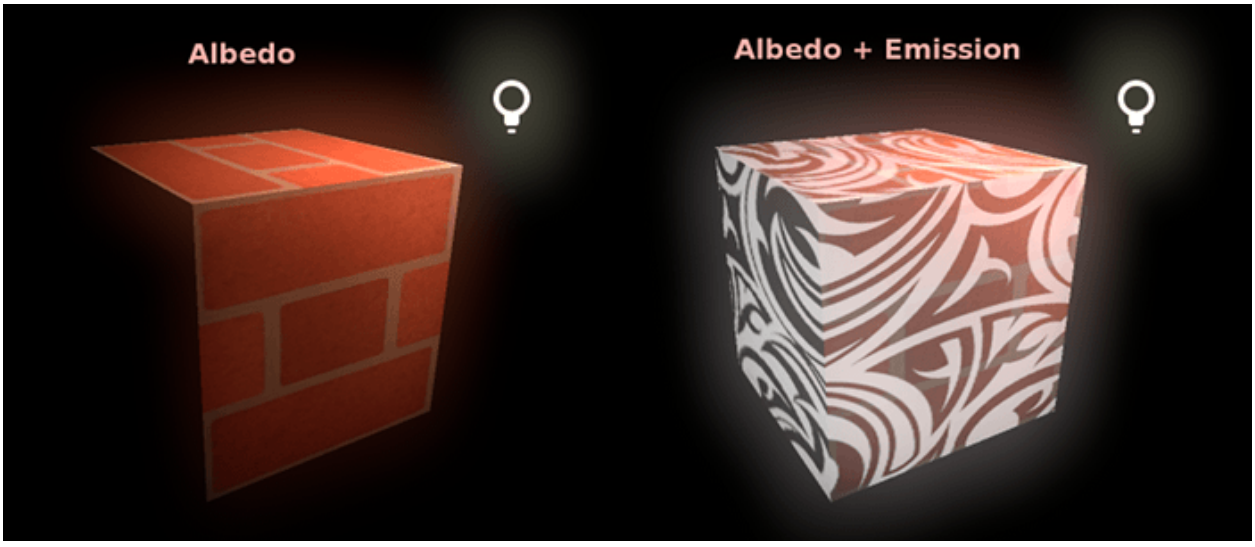
Roughness

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*.



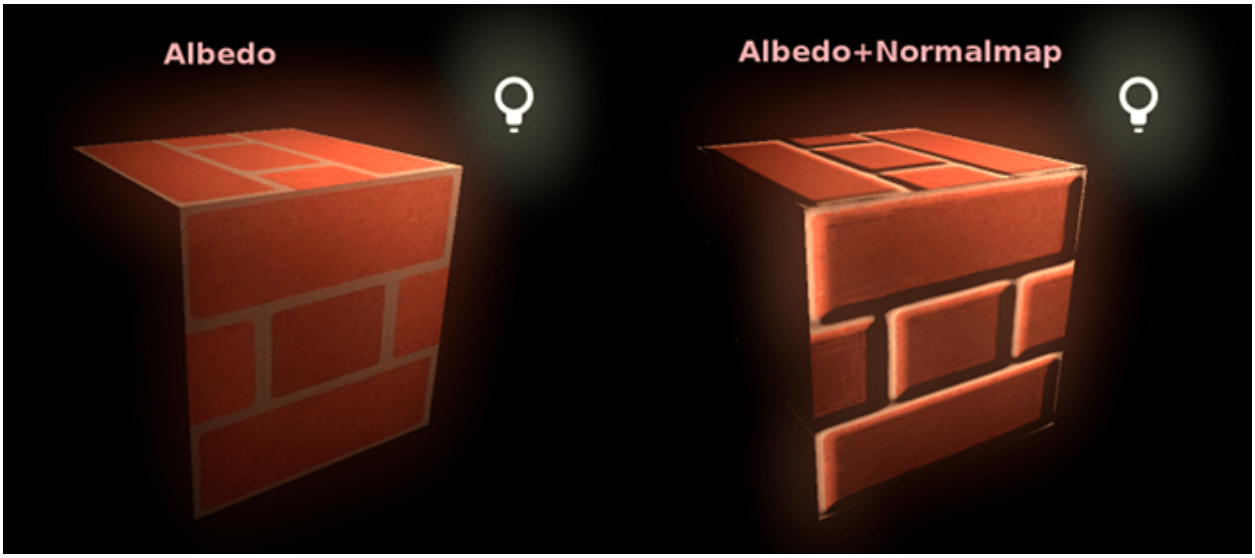
Emission

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.



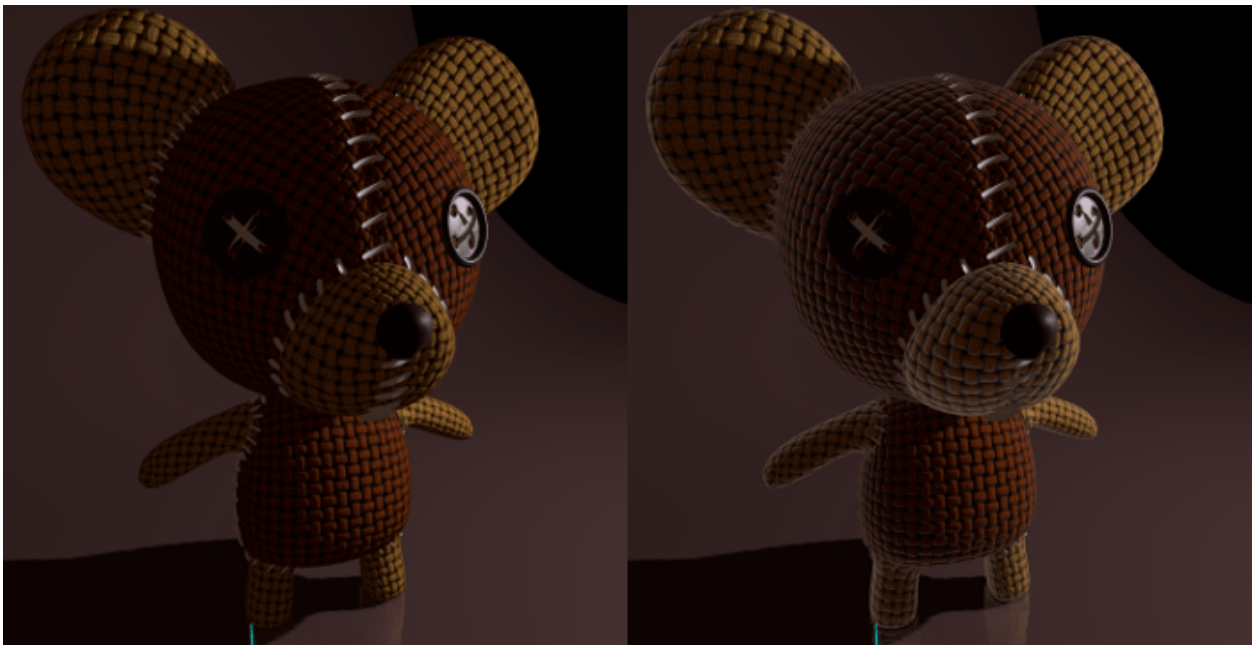
Normalmap

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.



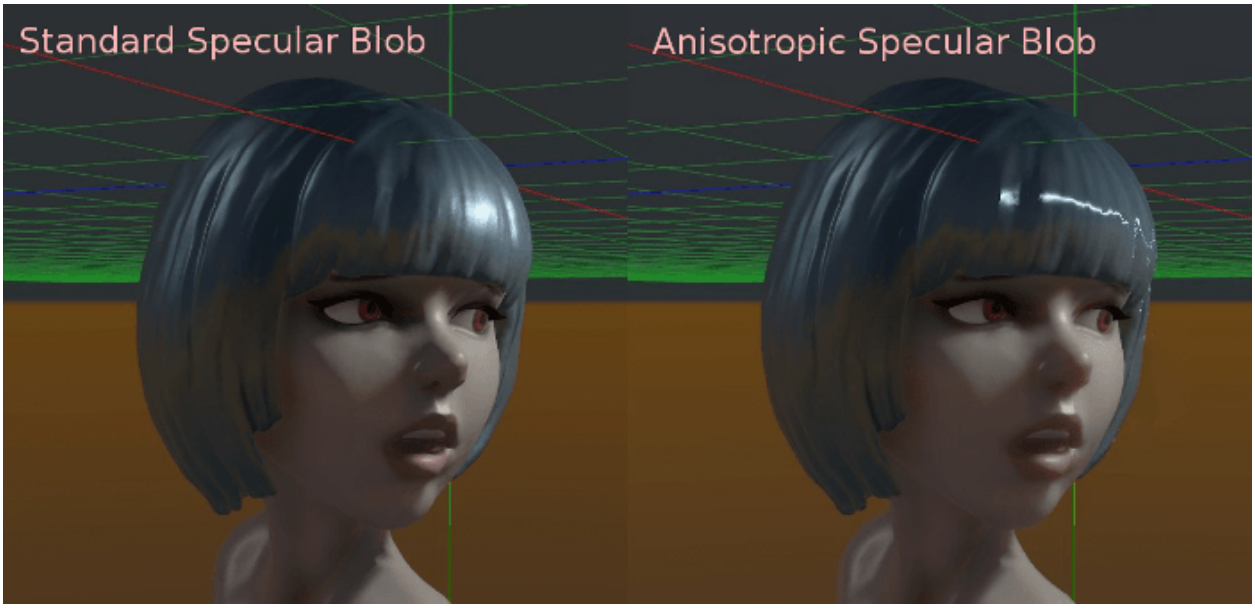
Rim size depends on roughness and there is a special parameter to specify how it must be colored. If *tint* is 0, the color of the light is used for the rim. If *tint* is 1, then the albedo of the material is used. Using intermediate values generally works best.

Clearcoat

The *clearcoat* parameter is used mostly to add a *secondary* pass of transparent coat to the material. This is common in car paint and toys. In practice, it's a smaller specular blob added on top of the existing material.

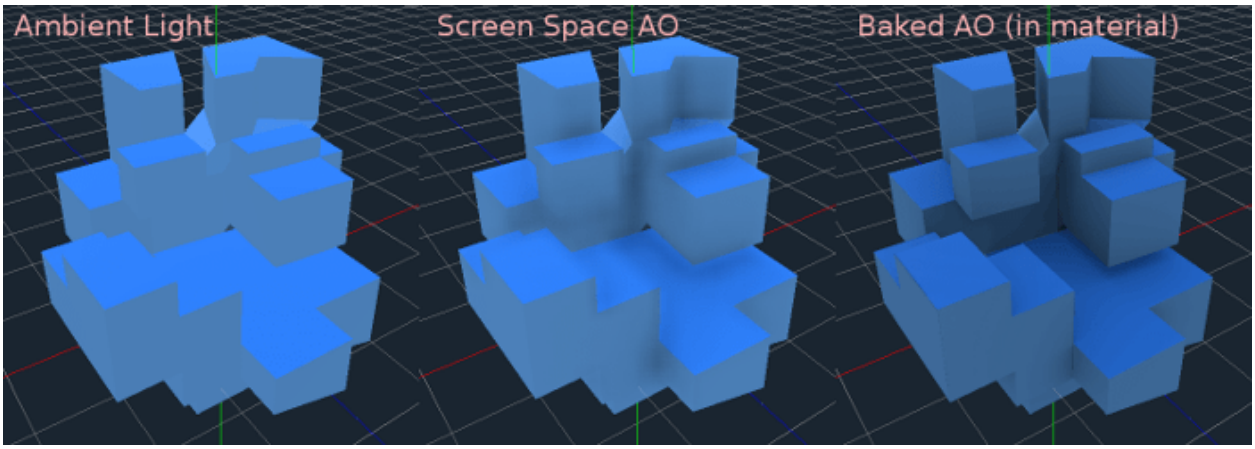
Anisotropy

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 aluminium more realistic. It works especially well when combined with flowmaps.



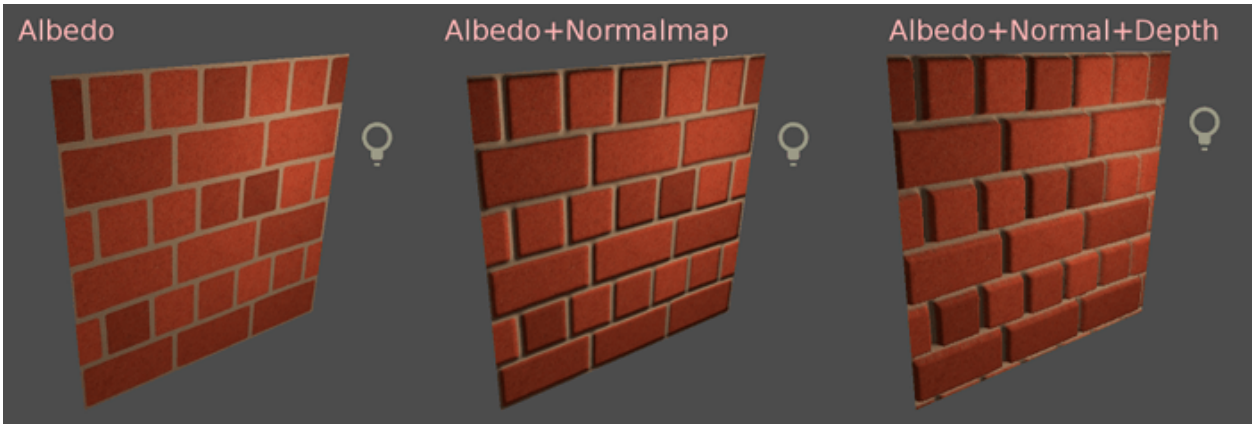
Ambient Occlusion

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.



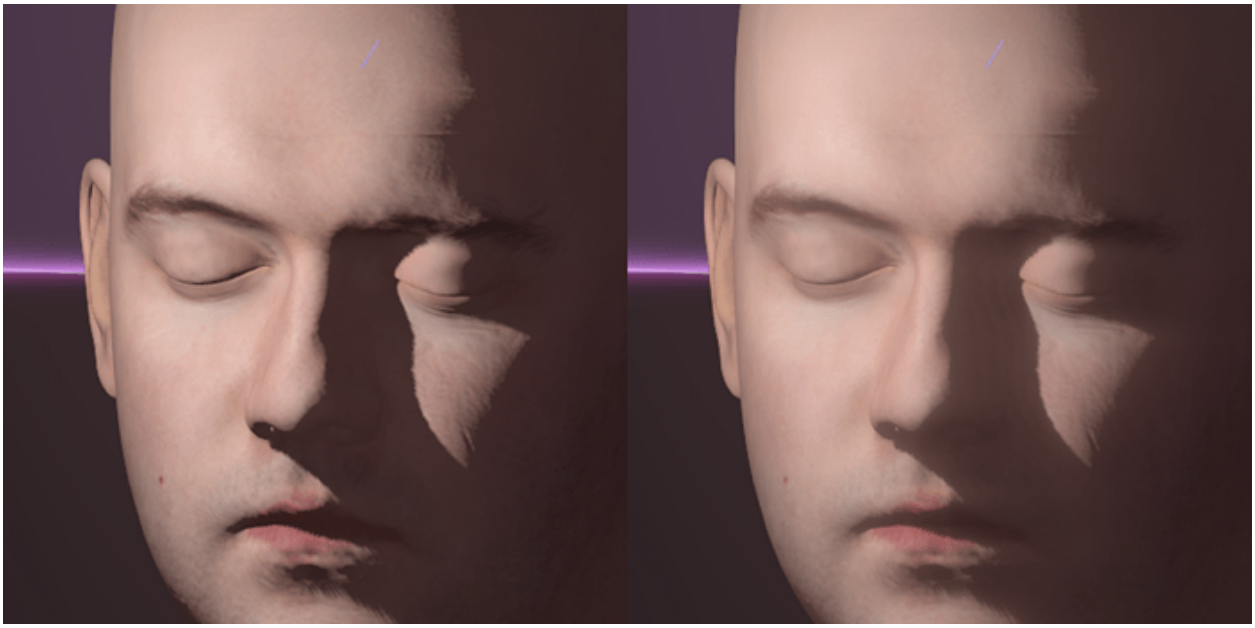
Depth

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 objets, but it produces a realistic depth effect for textues. 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 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 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 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.

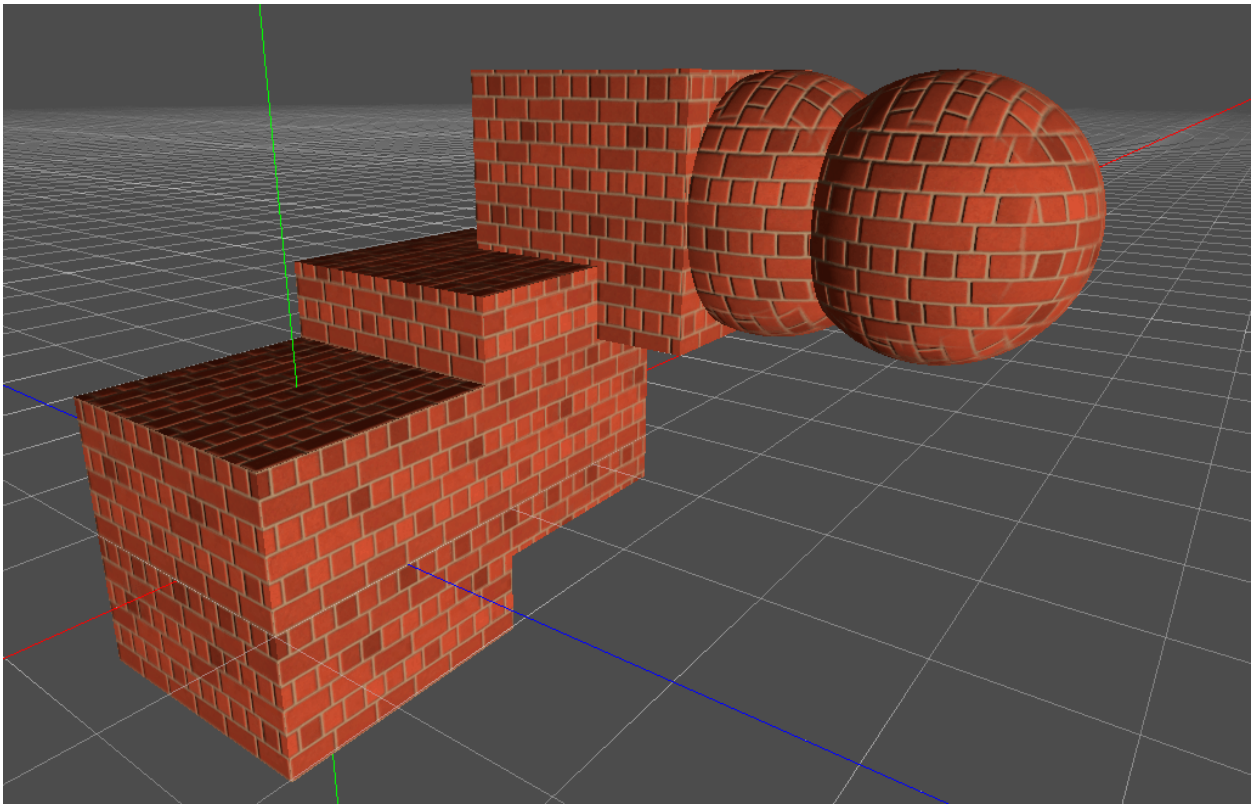
UV1 and UV2

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

Triplanar 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.



7.4.6 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 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.

7.4.7 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).

7.5 Lights And Shadows

7.5.1 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:

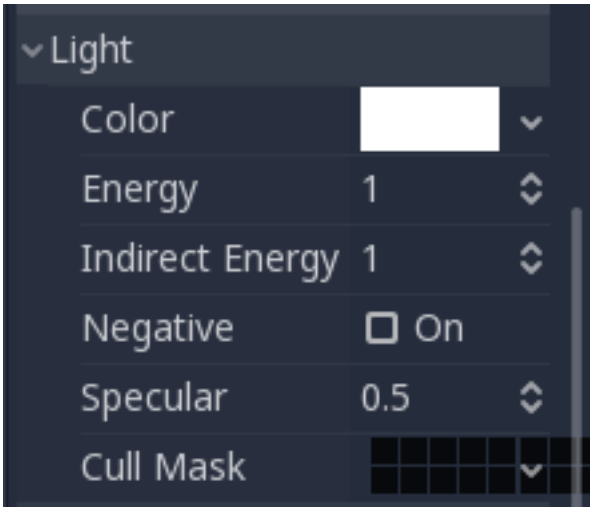
- From the Material itself, in the form of the emission color (though it does not affect nearby objects unless baked).
- Light Nodes: Directional, Omni and Spot.

- 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.

7.5.2 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:

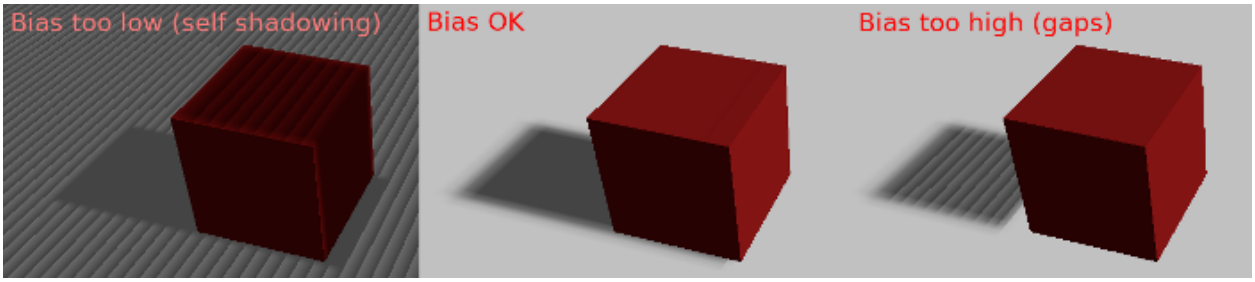
- **Color**: Base color for emitted light.
- **Energy**: Energy multiplier. This is useful to saturate lights or working with *High dynamic range*.
- **Indirect Energy**: Secondary multiplier used with indirect light (light bounces). This works in baked light or GIProbe.
- **Negative**: Light becomes substractive 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

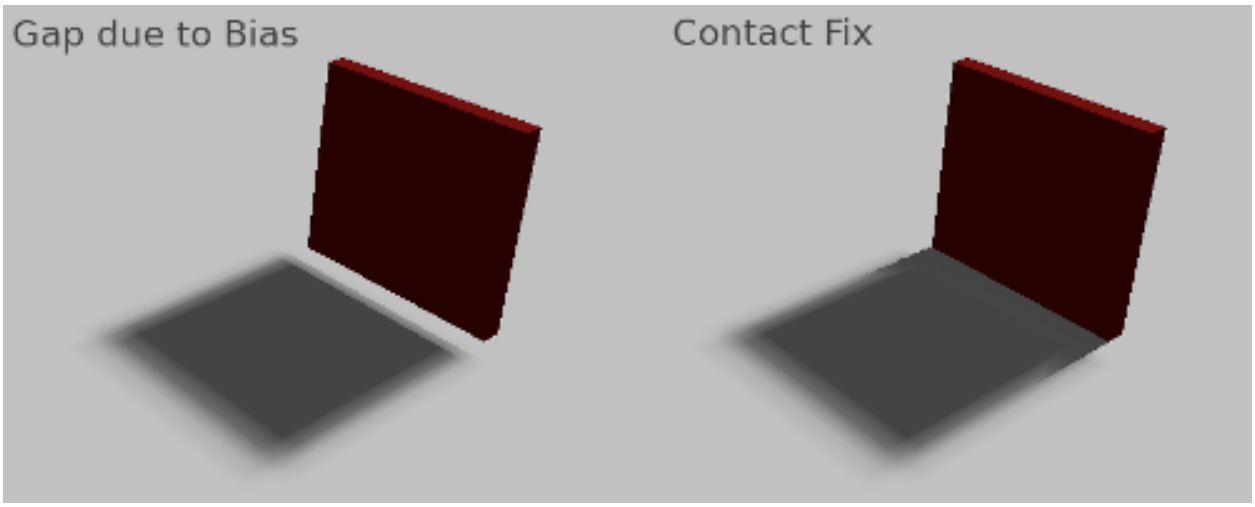
Lights can optionally cast shadows. This gives them greater realism (light does not reach occluded areas), but it can incur a bigger performance cost. There is a list of generic shadow parameters, each also has a specific function:

- **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:

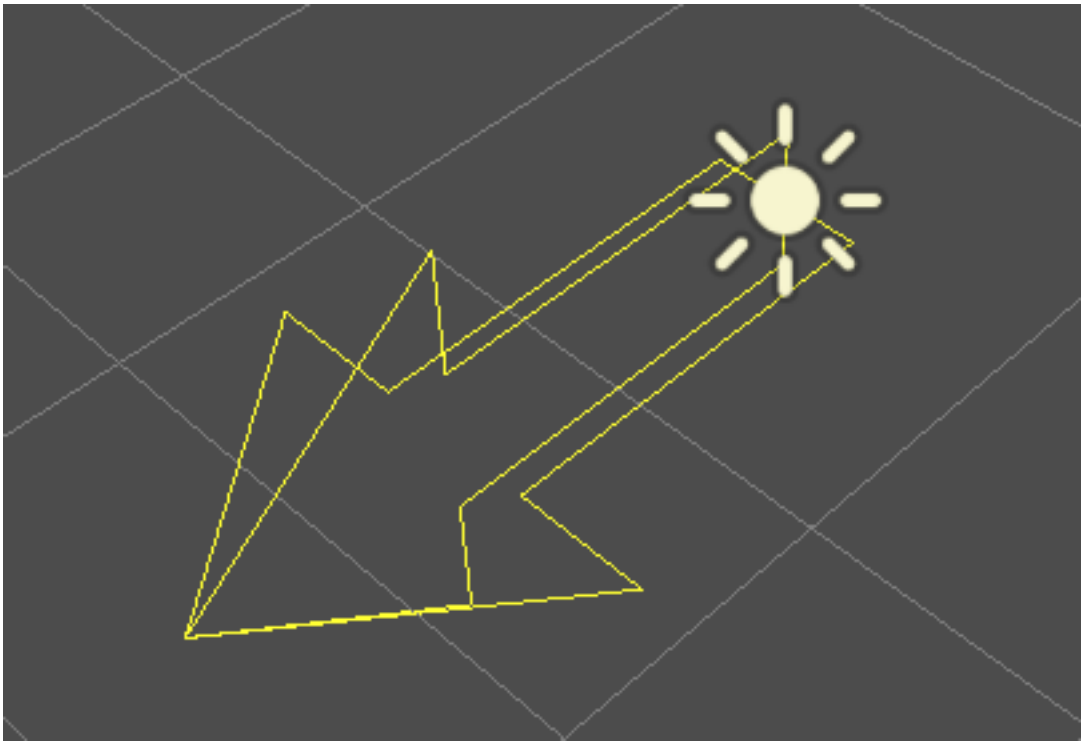


Any sort of bias issues can always be fixed by increasing the shadow map resolution, although that may lead to decreased peformance on low-end hardware.

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).

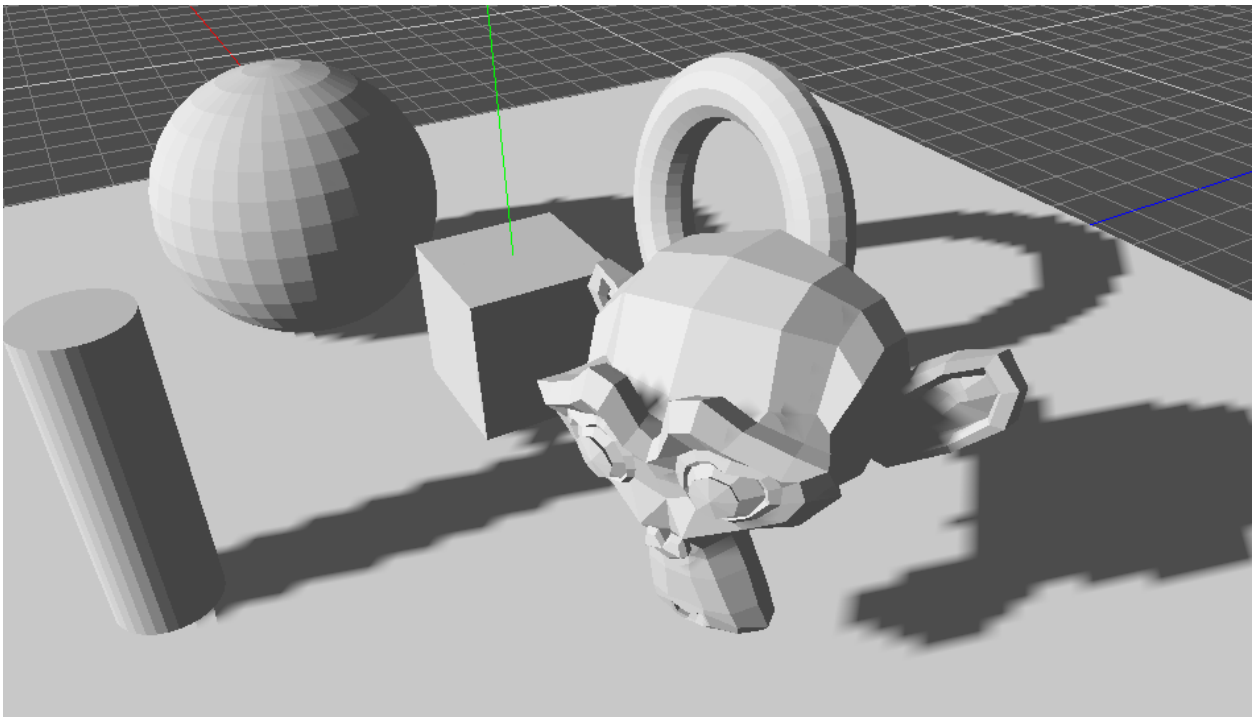
Directional light models an infinite number of parallel light rays covering the whole scene. The directional light node is represented by a big arrow, which indicates the direction of the light rays. However, the position of the node does not affect the lighting at all, and can be anywhere.



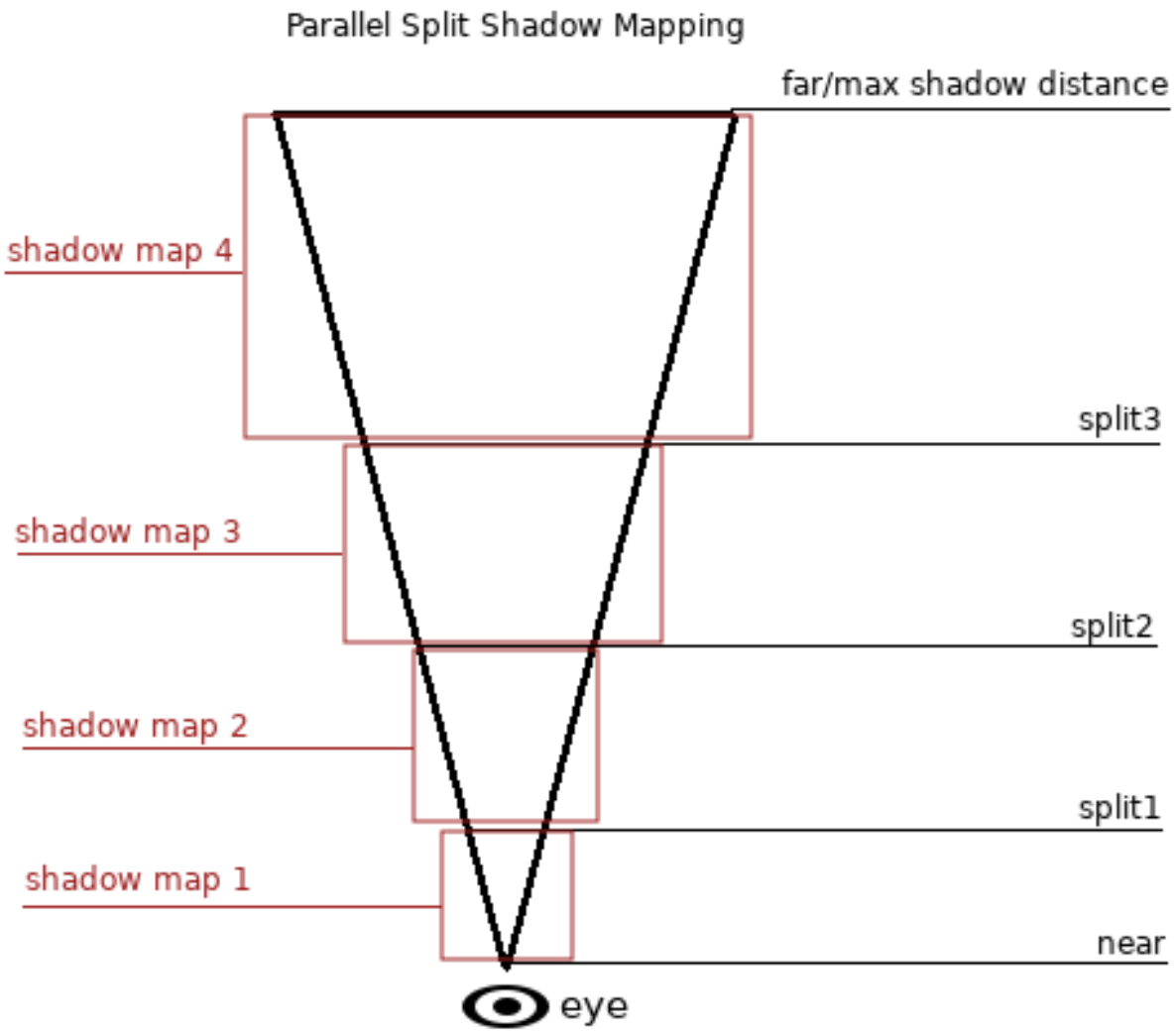
Every face whose front-side is hit by the light rays is lit, the others stay dark. Most light types have specific parameters but directional lights are pretty simple in nature so they don't.

Directional Shadow Mapping

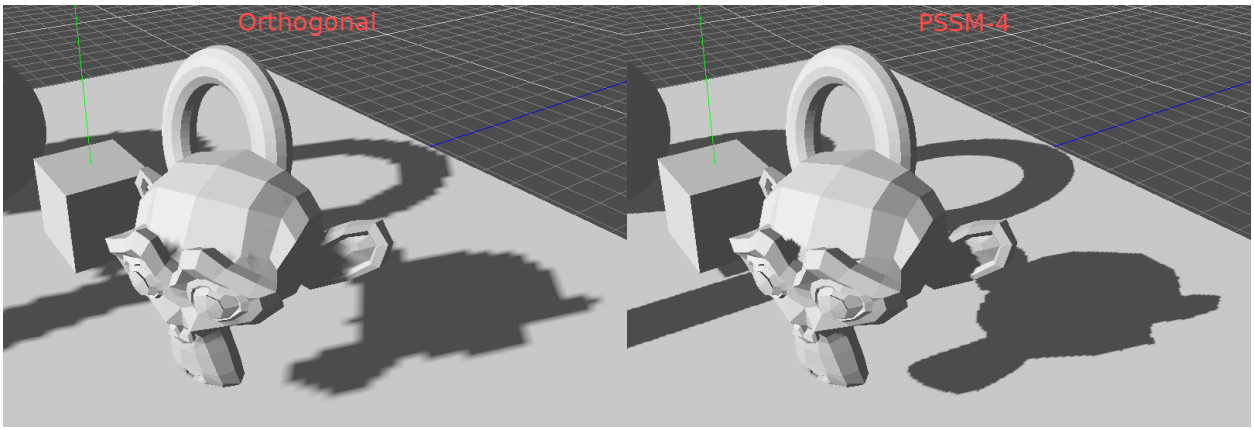
To compute shadow maps, the scene is rendered (only depth) from an orthogonal point of view that covers the whole scene (or up to the max distance). There is, however, a problem with this approach because objects closer to the camera receive blocky shadows.



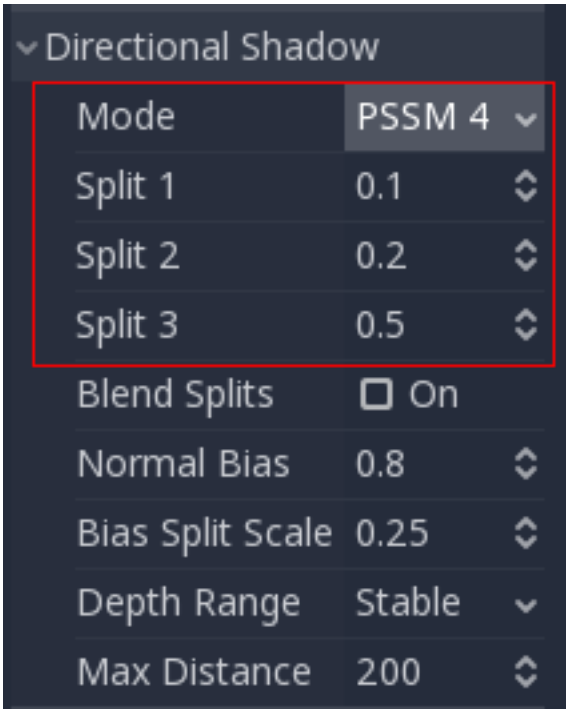
To fix this, a technique named "Parallel Split Shadow Maps" (or PSSM) is used. This splits the view frustum in 2 or 4 areas. Each area gets it's own shadow map. This allows small, close areas to the viewer to have the same shadow resolution as a huge, far-away area.



With this, shadows become more detailed:



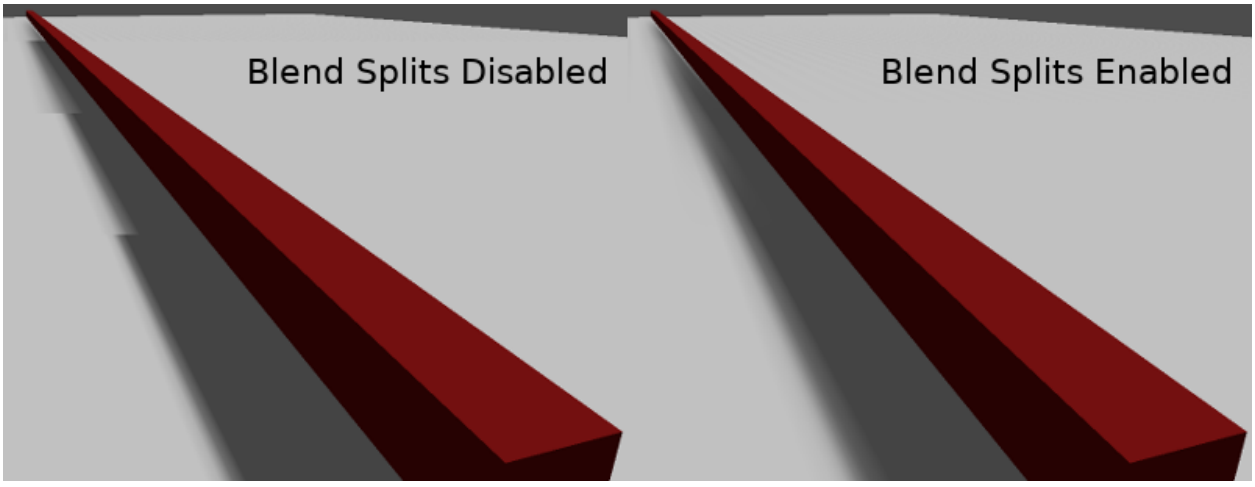
To control PSSM, a number of parameters are exposed:



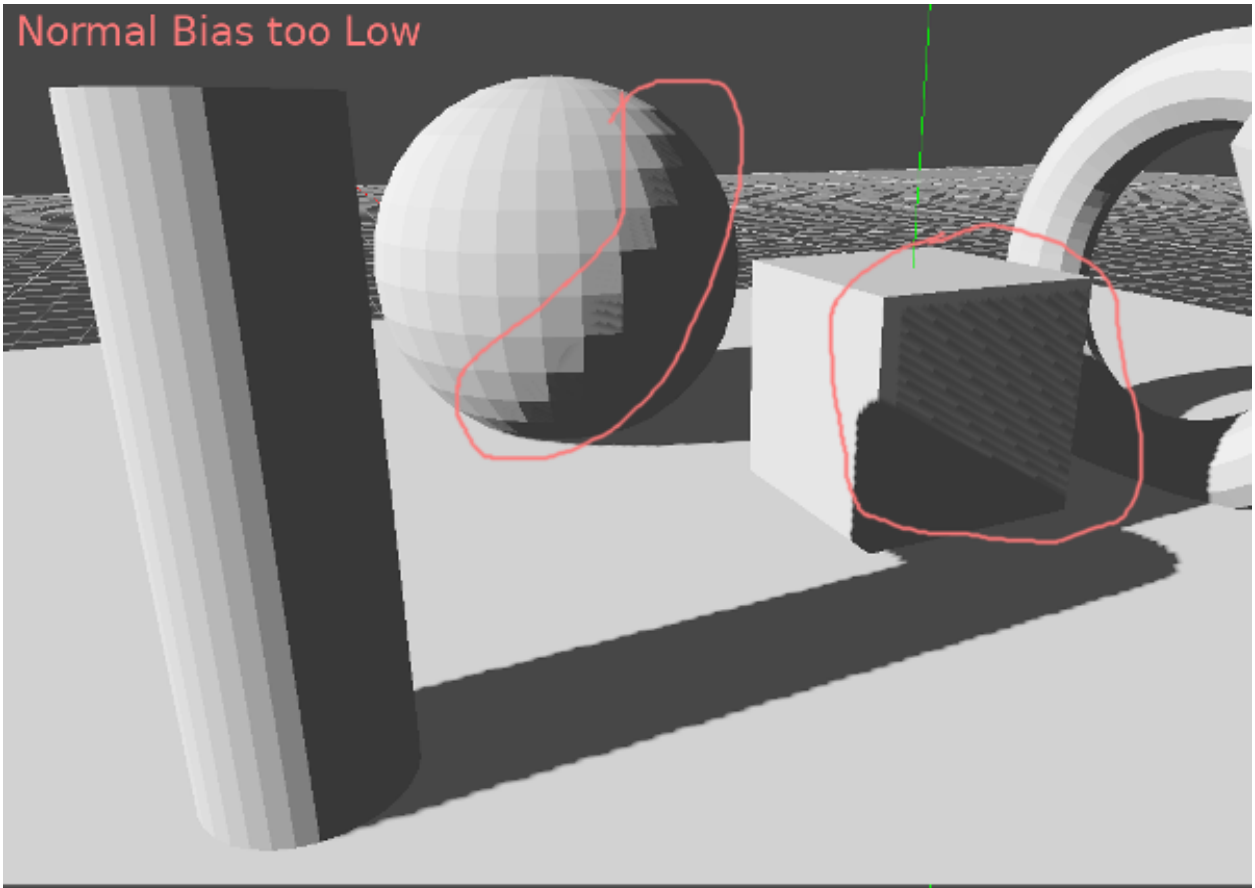
Each split distance is controlled relative to the camera far (or shadow **Max Distance** if greater than zero), so *0.0* is the eye position and *1.0* is where the shadow ends at a distance. Splits are in-between. Default values generally work well, but tweaking the first split a bit is common to give more detail to close objects (like a character in a third person game).

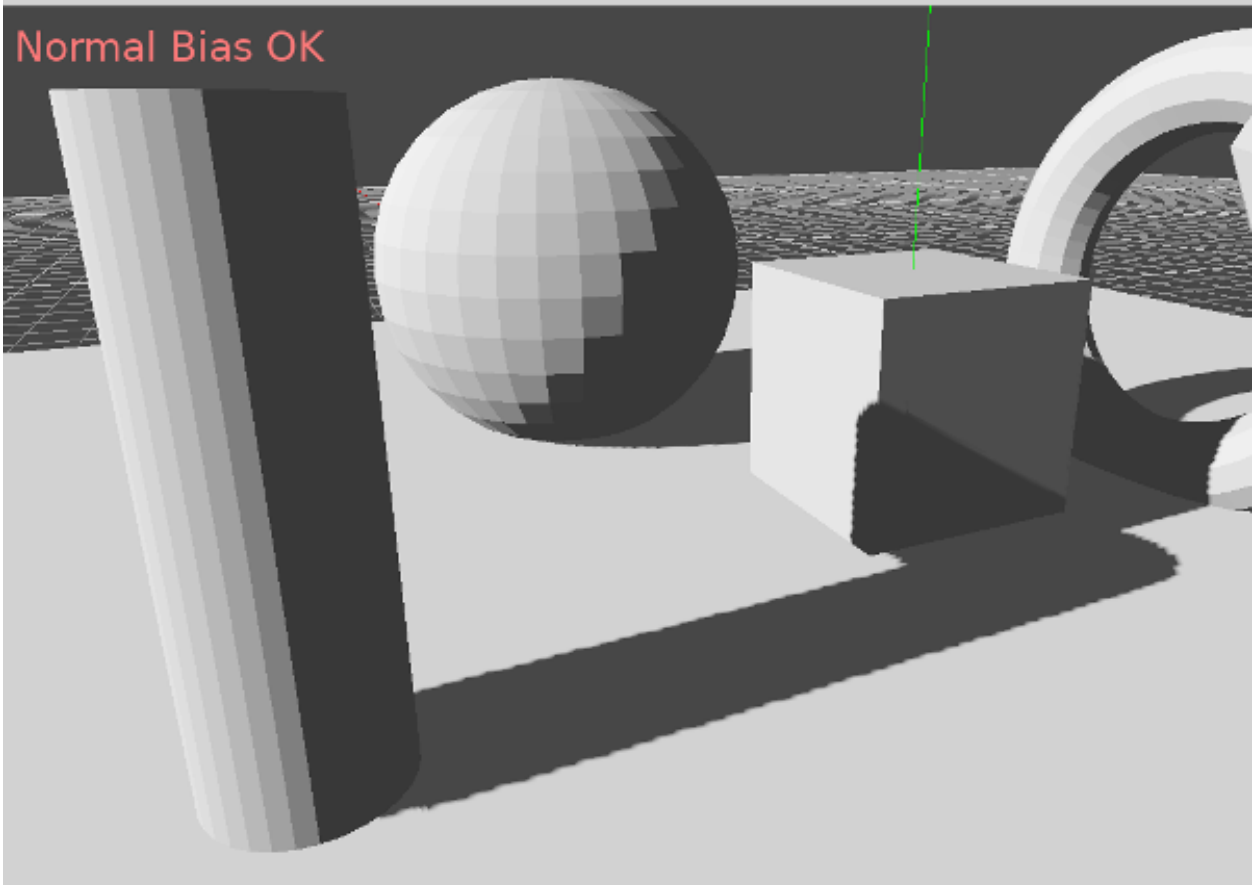
Always make sure to set a shadow *Max Distance* according to what the scene needs. The closer the max distance, the higher quality they shadows will have.

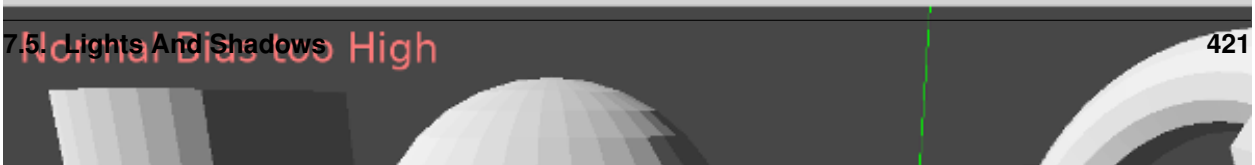
Sometimes, the transition between a split and the next can look bad. To fix this, the **"Blend Splits"** option can be turned on, which sacrifices detail in exchange for smoother transitions:



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.


Normal Bias too Low


Normal Bias OK


Normal Bias too High

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 two 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**: Triest 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).

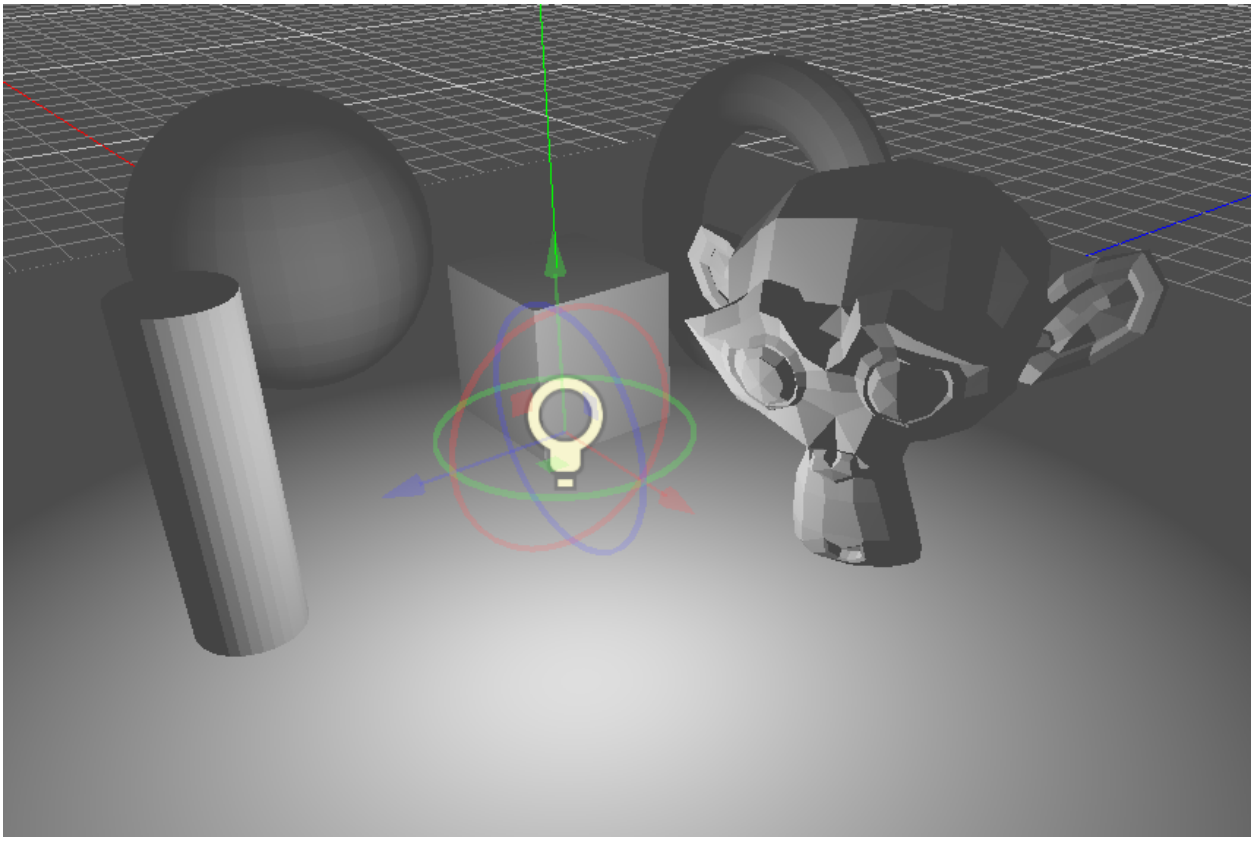
Just experiment which setting works better for your scene.

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.

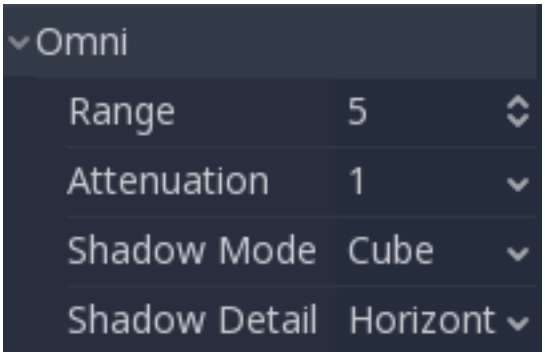
Omni light

Omni light is a point source that emits light spherically in all directions up to a given radius .

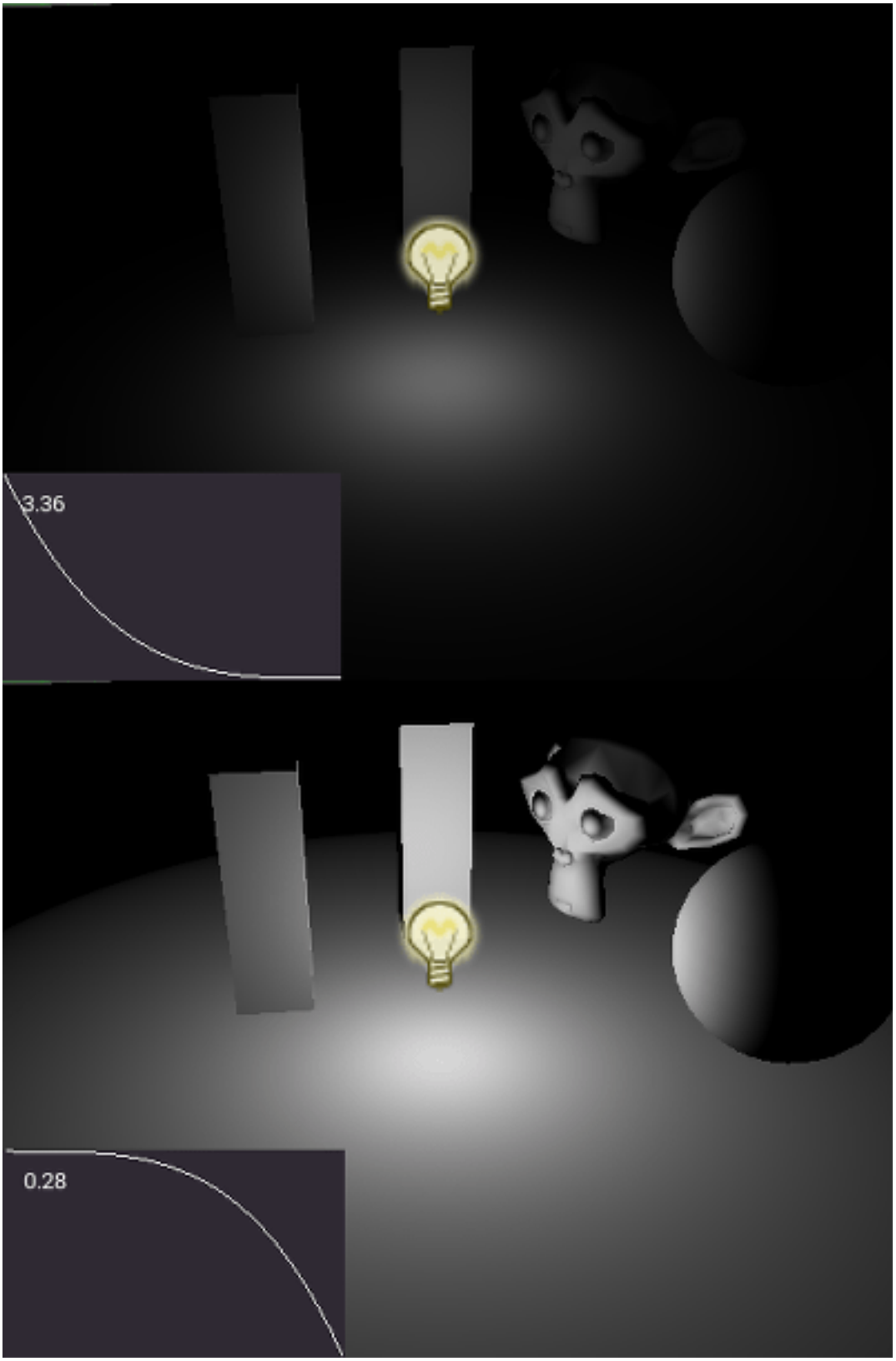


In real life, light attenuation is an inverse function, which means omni lights don't have a radius. This is a problem, because it means computing several omni lights would become demanding.

To solve this, a *Range* is introduced, together with an attenuation function.



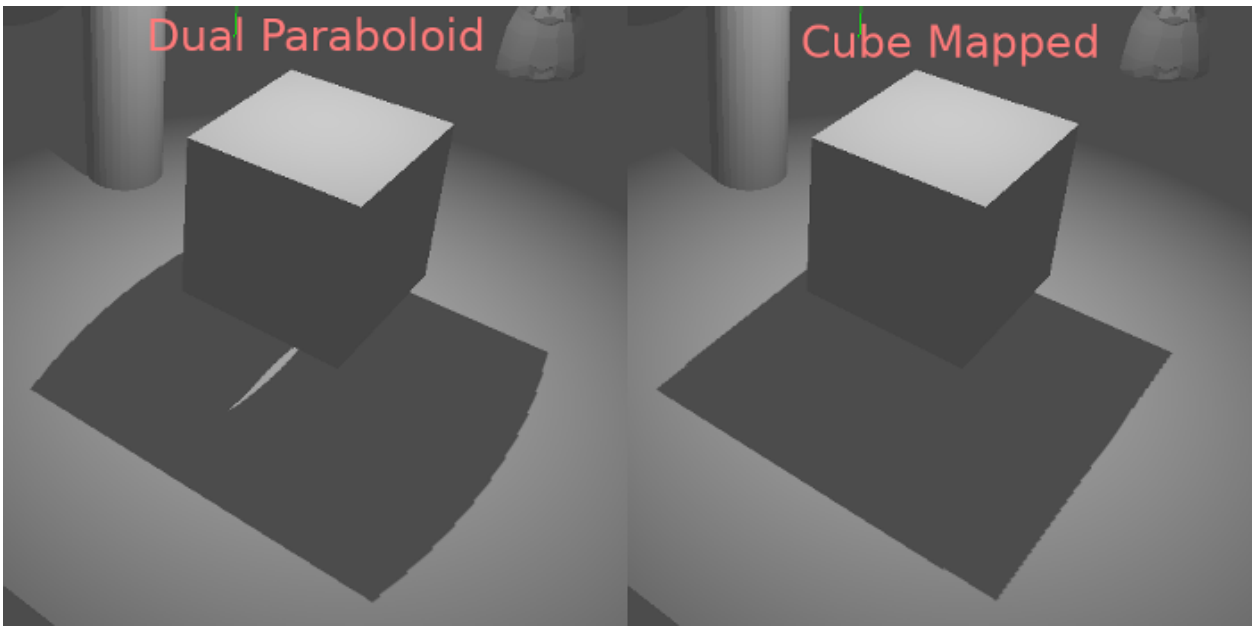
These two parameters allow tweaking how this works visually, in order to find aesthetically pleasing results.


3.36
0.28

Omni Shadow Mapping

Omni light shadow mapping is relatively straightforward. The main issue that needs to be considered is the algorithm used to render it.

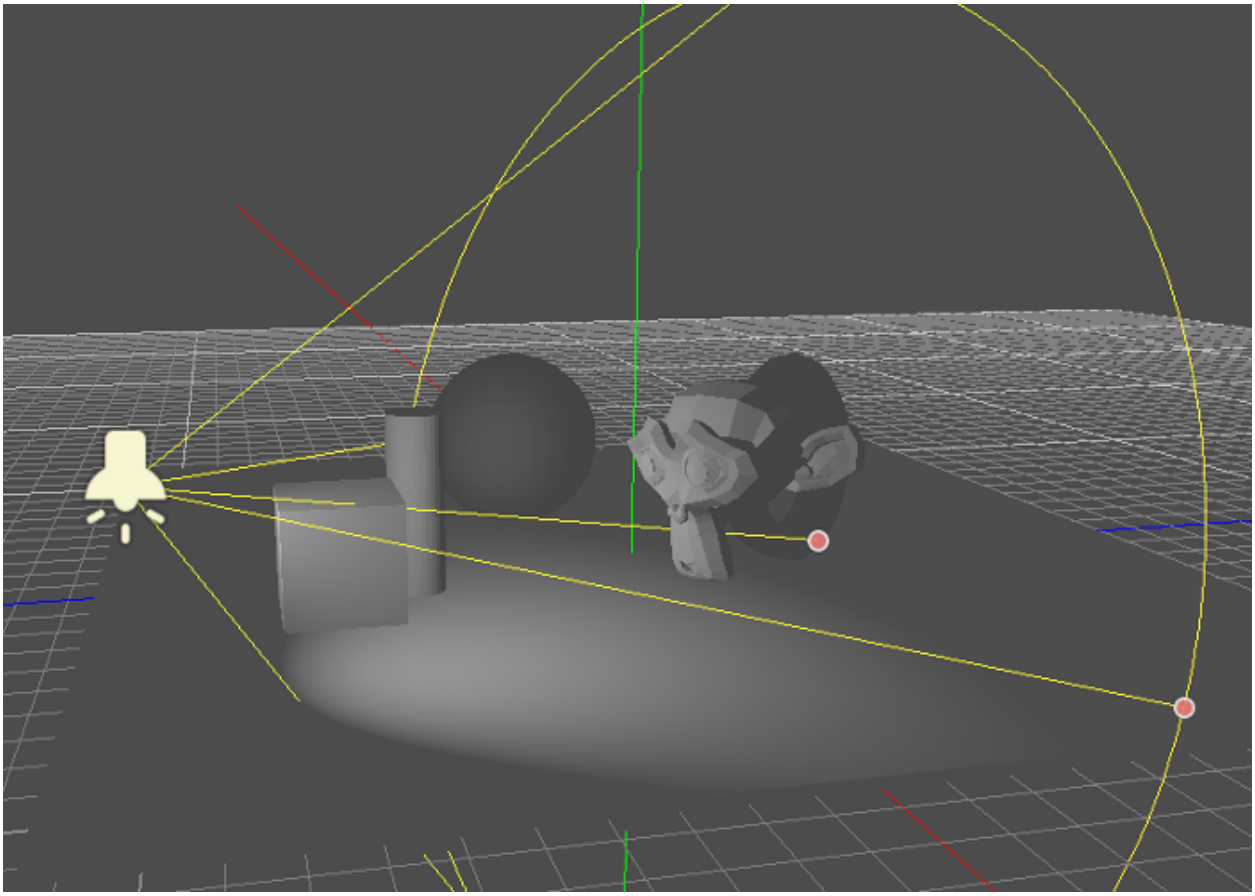
Omni Shadows can be rendered as either **"Dual Paraboloid" or "Cube Mapped"**. The former renders quickly but can cause deformations, while the later is more correct but more costly.



If the objects being renderer 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.

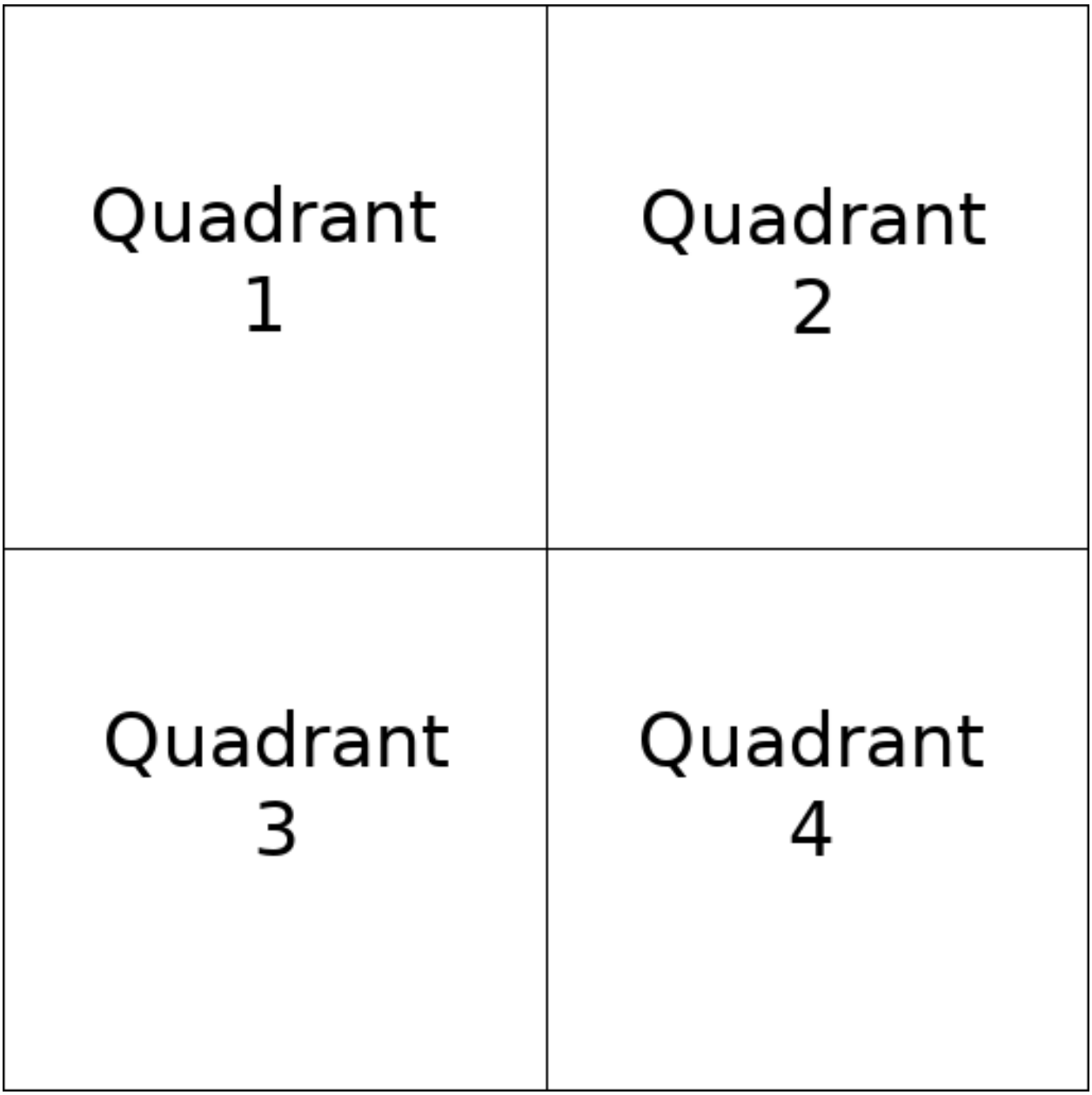
Spot Shadow Mapping

Spots don't need any parameters for shadow mapping. Keep in mind that, at more than 89 degrees of aperture, shadows stop functioning for spots, and you should consider using an Omni light.

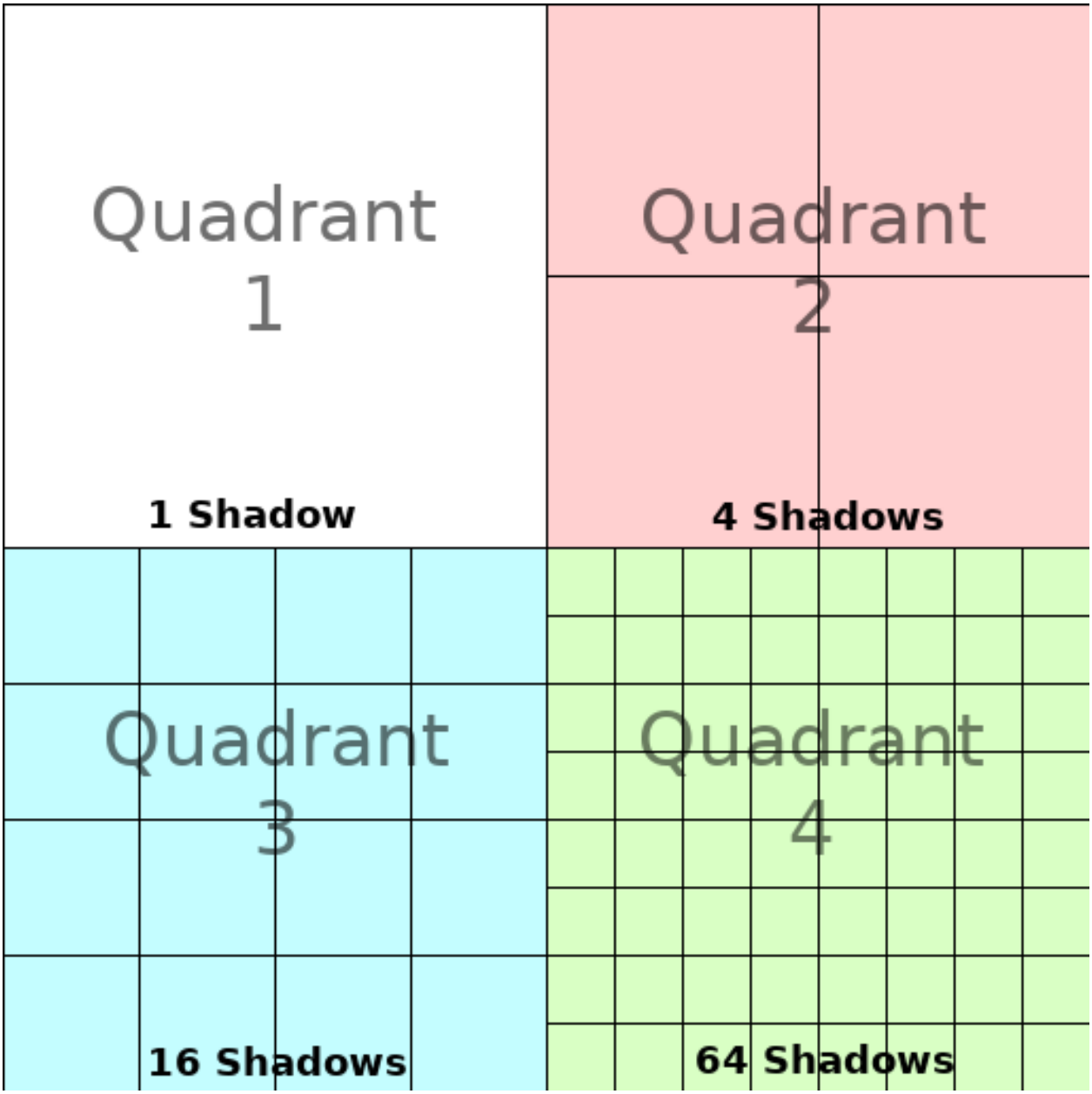
Shadow Atlas

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.

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.

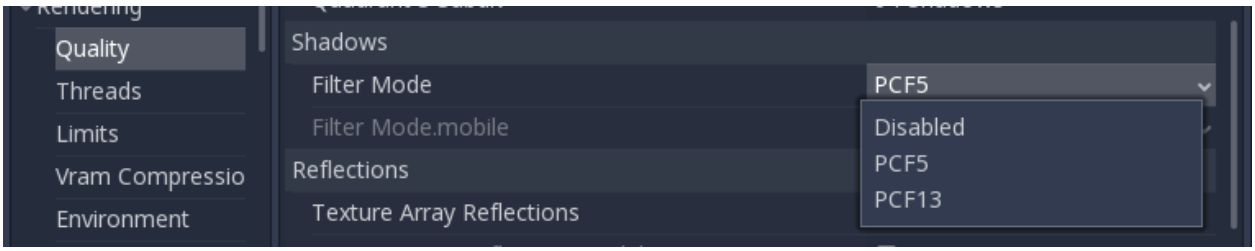
If the slots in a quadrant are full, lights are pushed back to smaller slots depending on size and distance.

This allocation strategy works for most games, but you may to use a separate one in some cases (as example, a

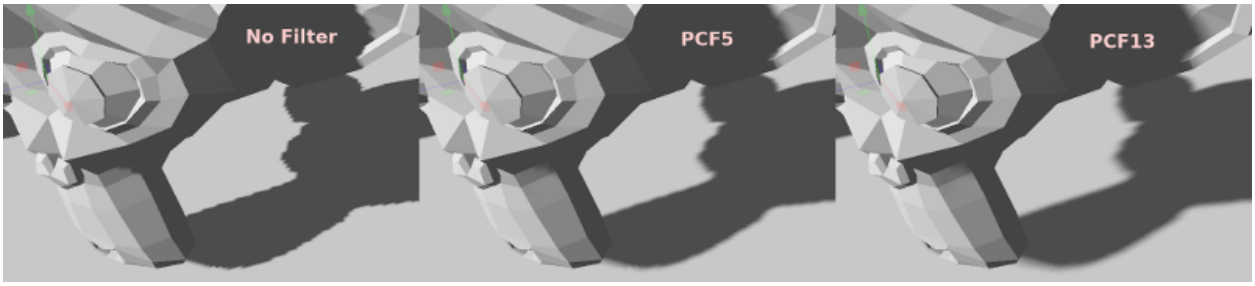
top-down game where all lights are around the same size and quadrands may have all the same subdivision).

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:



7.6 Reflection Probes

7.6.1 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.

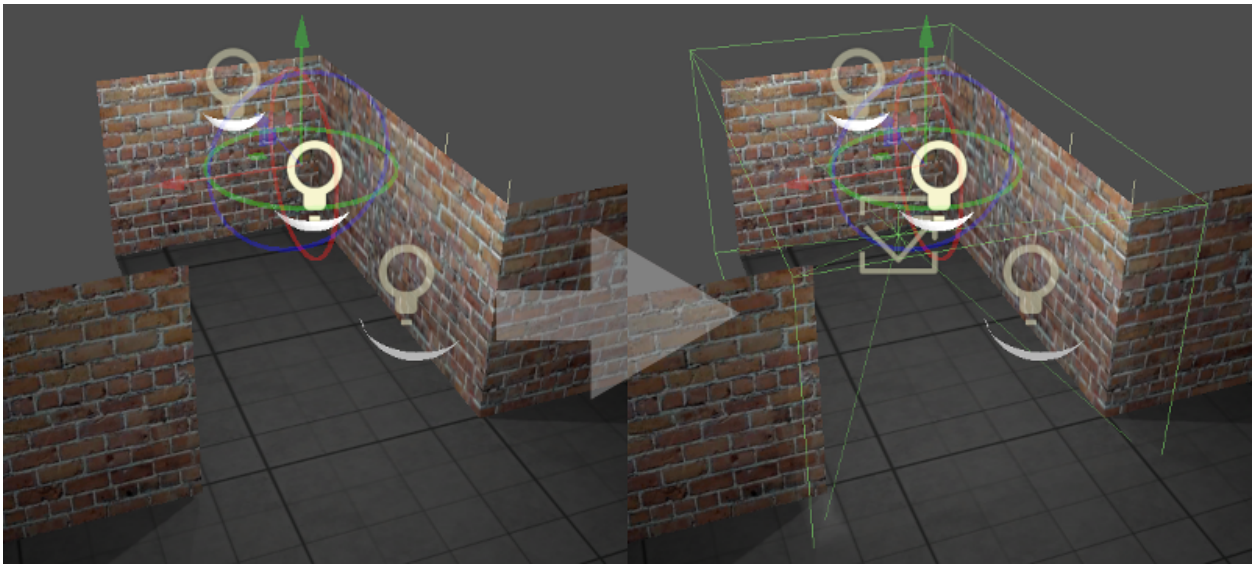
A probe of this type captures the surroundings (as a sort of 360 degrees image), and stores versions of it with increasing levels of *blur*. This is used to simulate roughness in materials, as well as ambient lighting.

While these probes are a 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 (especially when roughness is 0).

7.6.2 Setting Up

Create a ReflectionProbe node, and wrap it around the area where you want to have reflections:



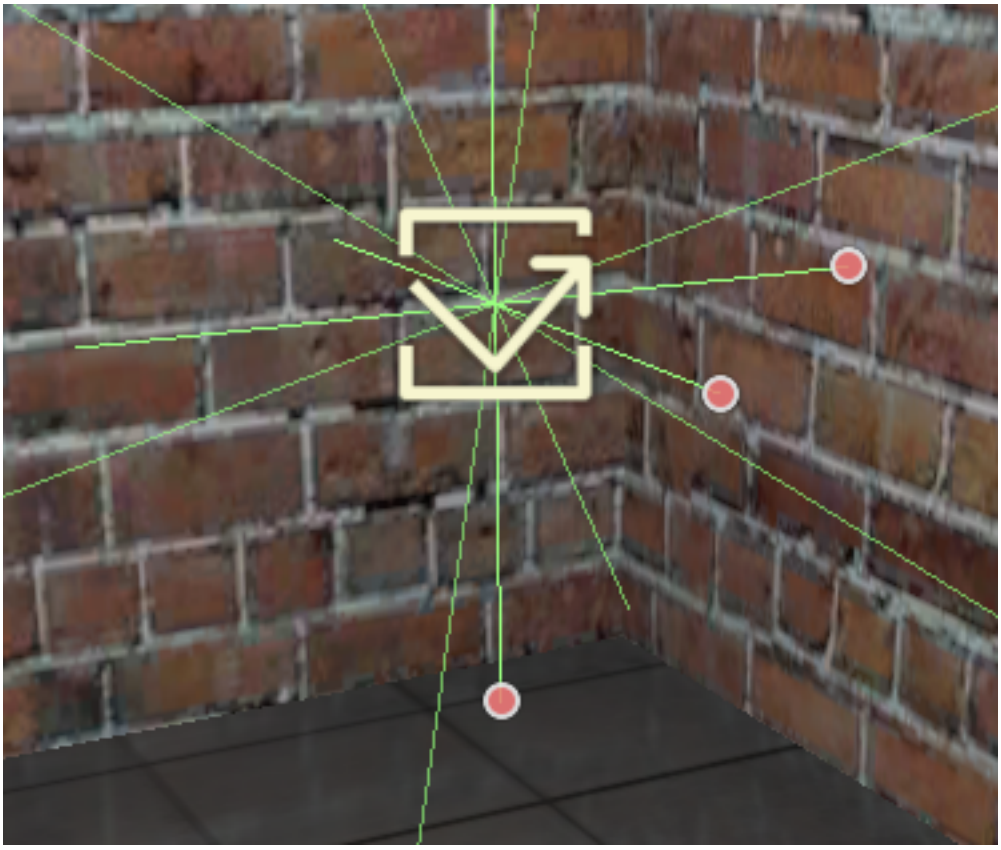
This should result in immediate local reflections. If you are using a Sky texture, reflections are by default blended with it.

By default, on interiors, reflections may appear to not have much consistence. In this scenario, make sure to tick the *"Box Correct"* property.

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:

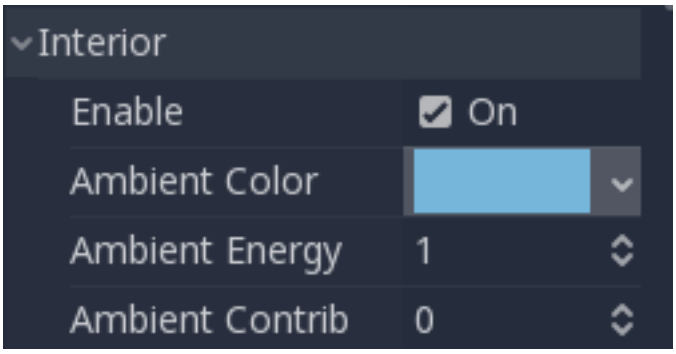


By default, shadow mapping is disabled when rendering probes (only in the rendered image inside the probe, not the actual scene). This is a simple way to save on performance and memory. If you want shadows in the probe, they can be toggled on/off with the *Enable Shadow* setting:

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:

7.6.3 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.

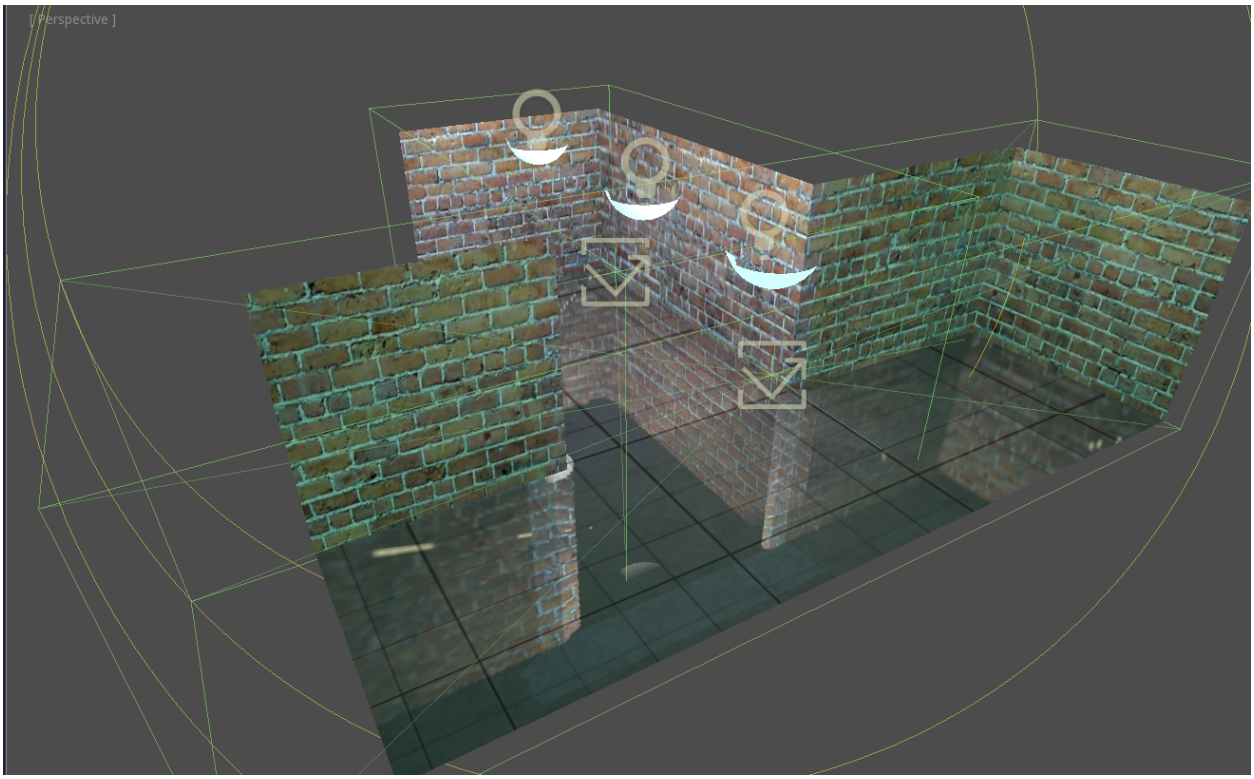


When probes are set to **Interior**, custom constant ambient lighting can be specified per probe. Just choose a color and an energy.

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).

7.6.4 Blending

Multiple reflection probes can be used and Godot will blend them where they overlap using a smart algorithm:



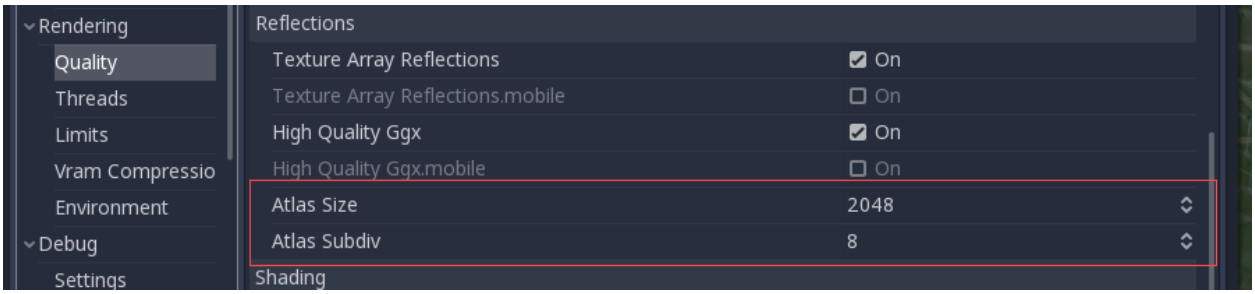
As you can see, this blending is never perfect (after all, these are box reflections, not real reflections), but these artifacts are only visible when using perfectly mirrored reflections. Normally, scenes have normal mapping and varying levels of roughness which can hide this.

Alternatively, Reflection Probes work 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.

7.6.5 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



7.7 GI Probes

7.7.1 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.

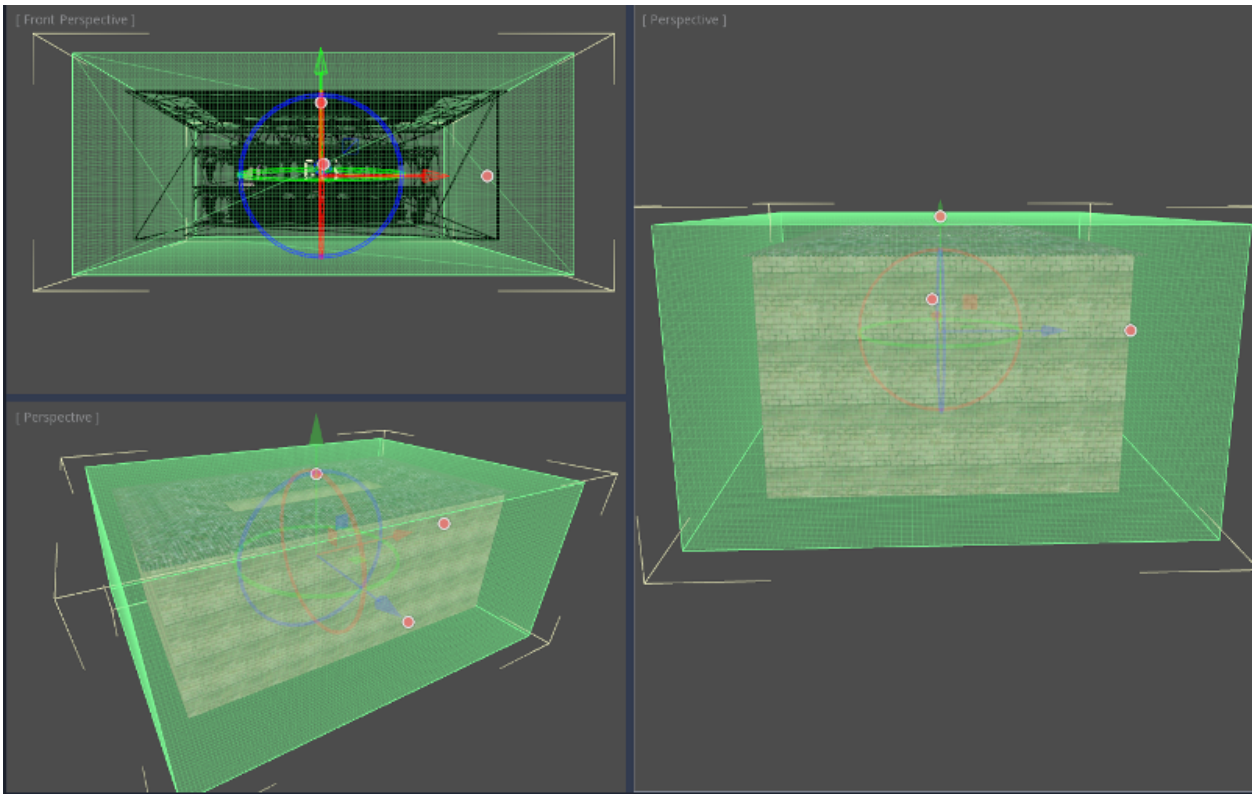
The main downside of GIProbes are:

- 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).

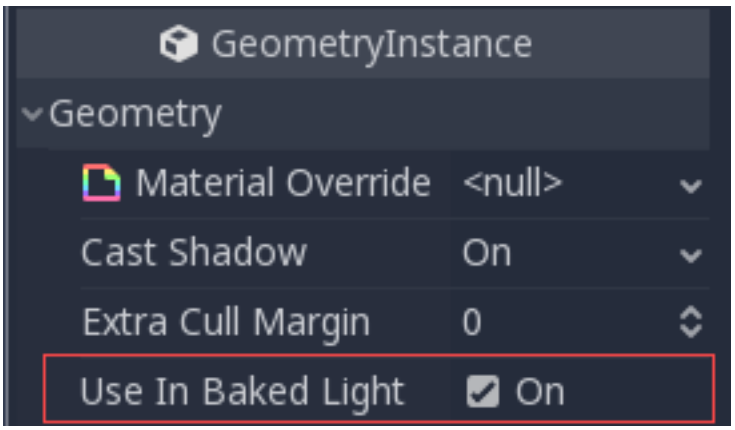
- Reflections are voxelized, so they don't look as sharp as with ReflectionProbe, but in exchange they are volumetric so any room size or shape works for them. Mixing them with Screen Space Reflection also works well.
- They consume considerably more video memory than Reflection Probes, so they must be used by care in the right subdivision sizes.

7.7.2 Setting Up

Just like a ReflectionProbe, simply set up the GIProbe by wrapping it around the geometry that will be affected.

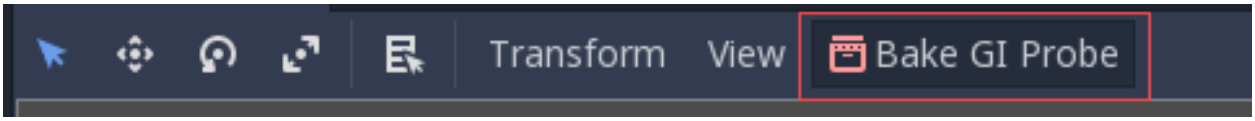


Afterwards, make sure to enable the geometry will be baked. This is important in order for GIPRobe to recognize objects, otherwise they will be ignored:



Once the geometry is set-up, push the Bake button that appears on the 3D editor toolbar to begin the pre-baking

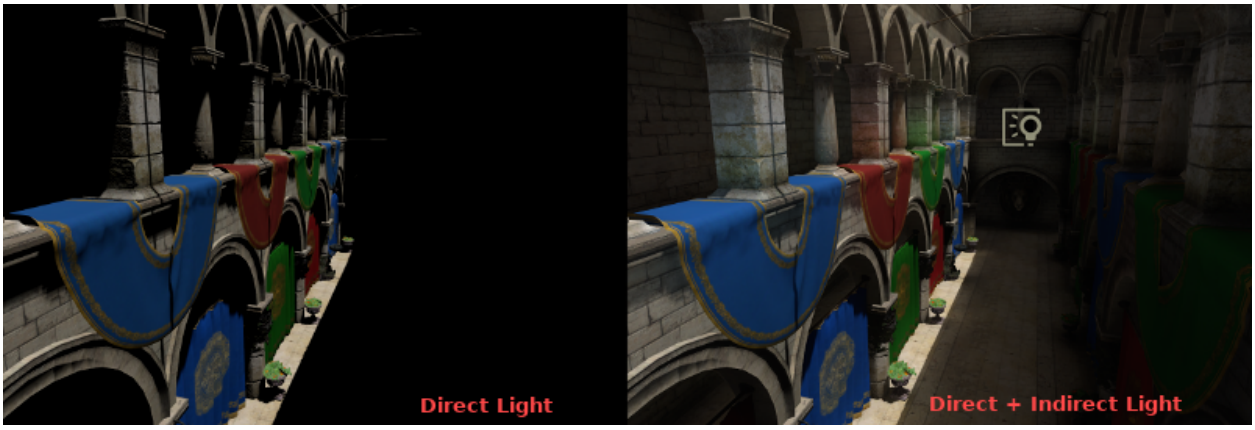
process:



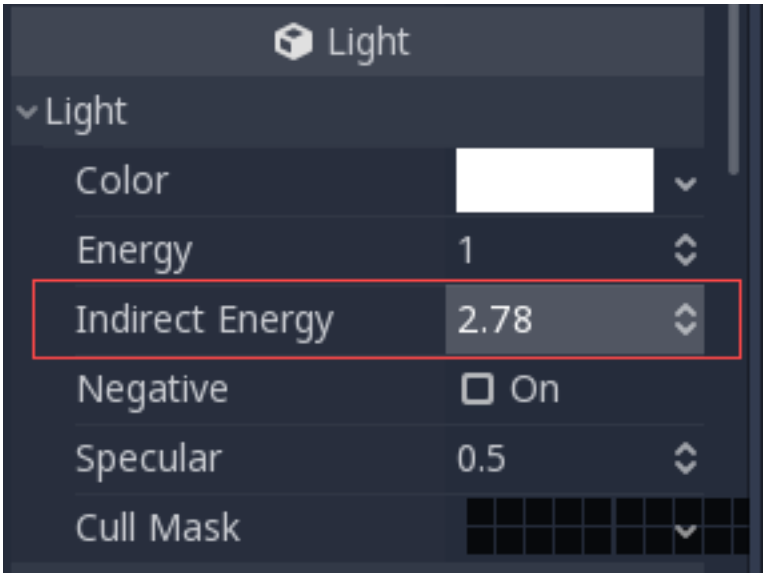
7.7.3 Adding Lights

Unless there are materials with emission, GIProbe does nothing by default. Lights need to be added to the scene to have an effect.

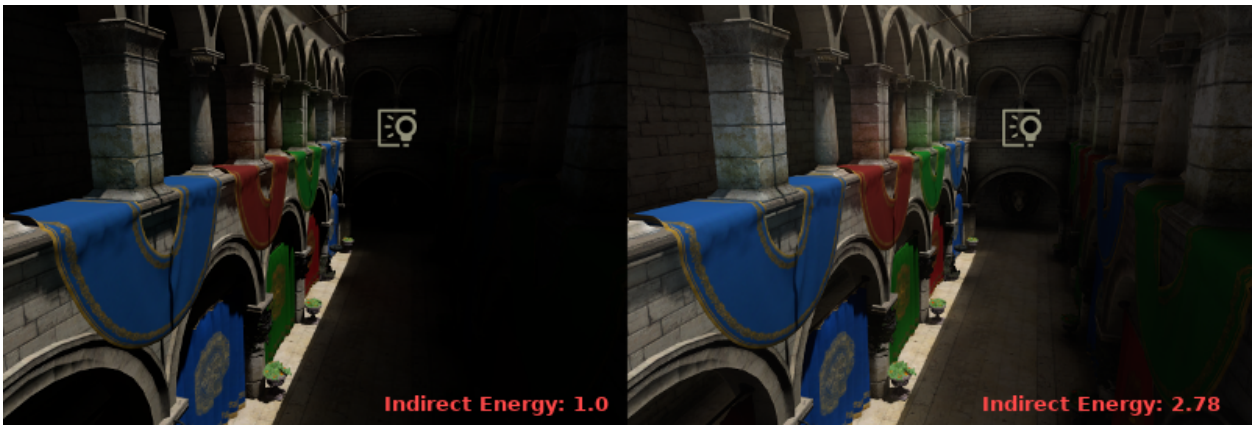
The effect of indirect light can be viewed quickly (it is recommended you turn off all ambient/sky lighting to tweak this, though as in the picture):



In some situations, though, indirect light may be too weak. Lights have an indirect multiplier to tweak this:

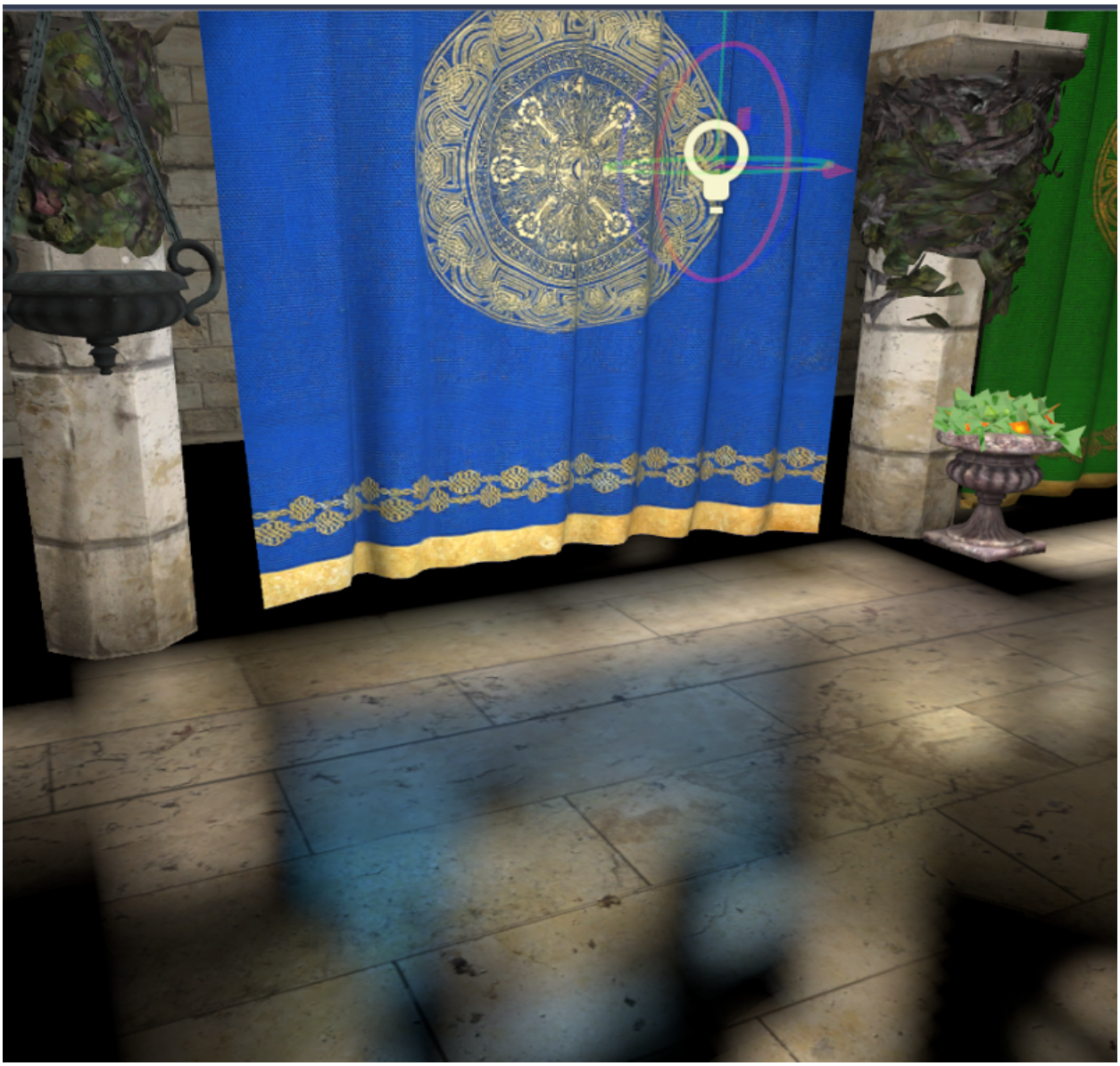


And, as GIPRobe lighting updates in real-time, this effect is immediate:

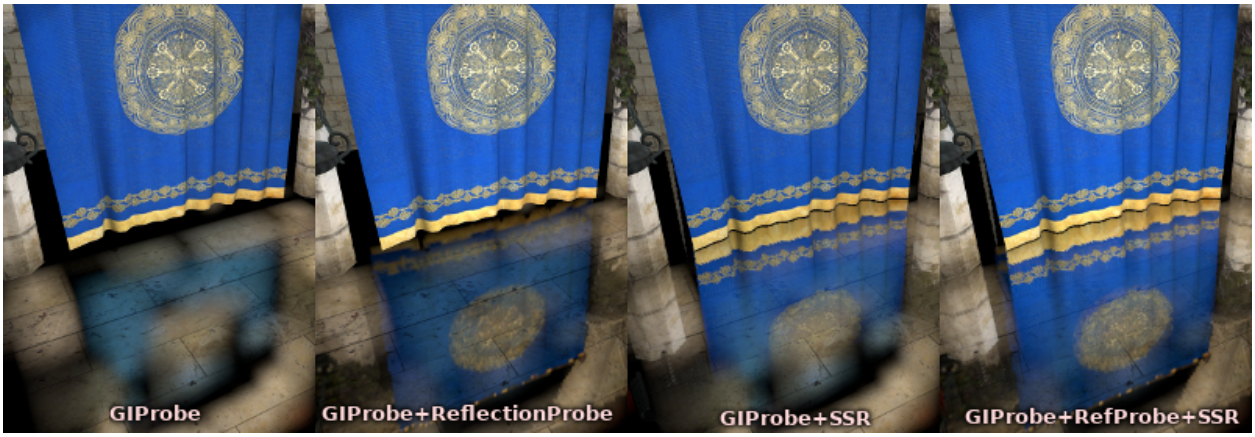


7.7.4 Reflections

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.



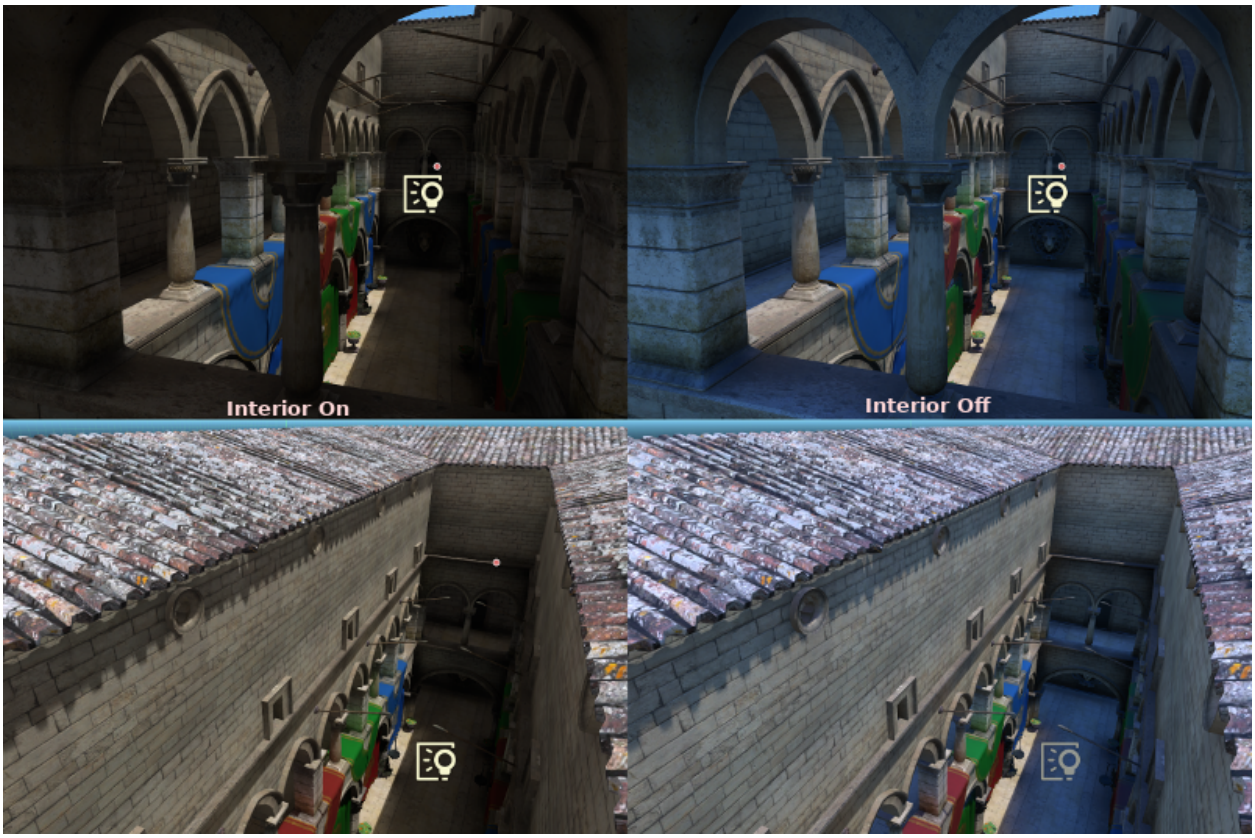
GIProbes can easily be mixed with Reflection Probes and Screen Space Reflections, as a full 3-stage fallback-chain. This allows to have precise reflections where needed:



7.7.5 Interior vs Exterior

GI Probes normally allow mixing with lighting from the sky. This can be disabled when turning on the *Interior* setting.

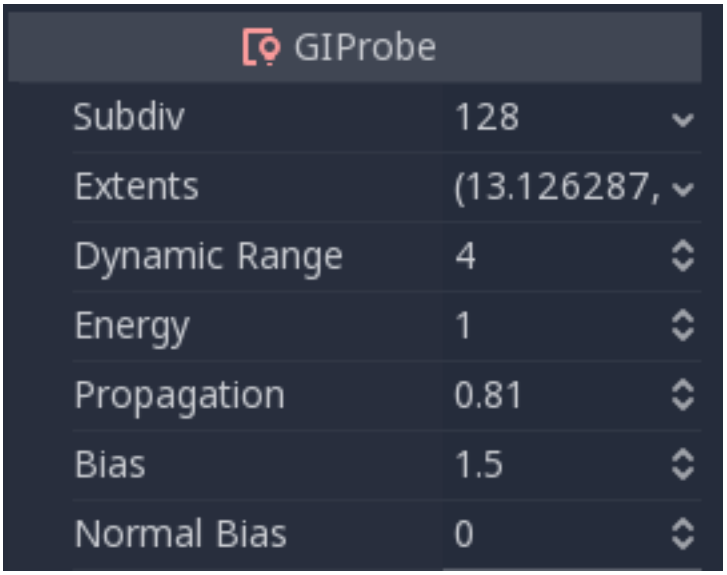
The difference becomes clear in the image below, where light from the sky goes from spreading inside to being ignored.



As complex buildings may mix interiors with exteriors, combining GIProbes for both parts works well.

7.7.6 Tweaking

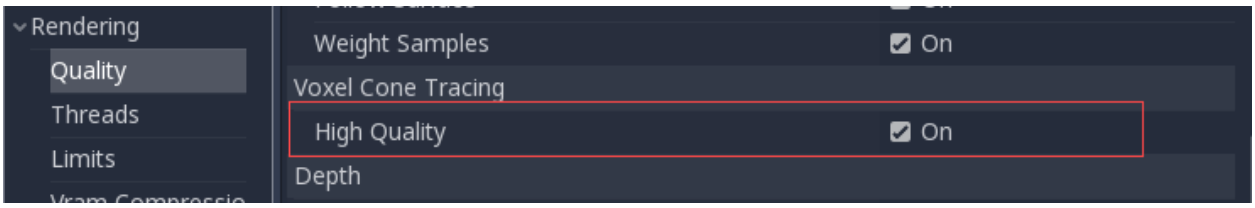
GI Probes support a few parameters for tweaking:



- **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.

7.7.7 Quality

GIProbes are quite demanding. It is possible to use lower quality voxel cone tracing in exchange of more performance.



7.8 Baked Lightmaps

7.8.1 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 low end PCs and mobile devices as they consume almost no resources in

run-time.

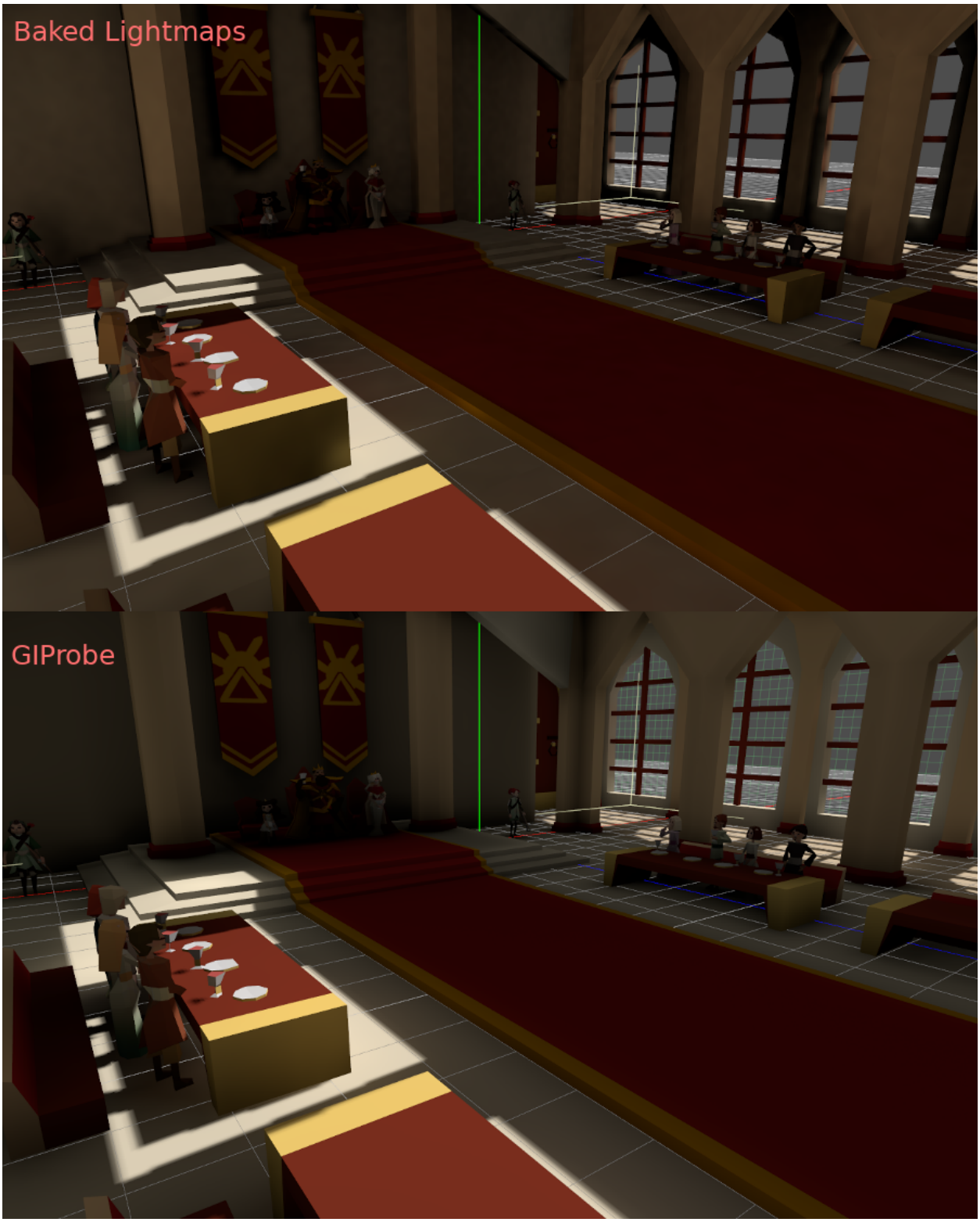
Unlike GIProbes, Baked Lightmaps are completely static, once baked they can't be modified at all. They also don't provide the scene with reflections, so using *Reflection Probes* together with it on interiors (or using a Sky on exteriors) is a requirement to get good quality.

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 upt. For low end compatibility or mobile, though, Baked Lightmaps are your only choice.

7.8.2 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.


Baked Lightmaps
GIProbe

7.8.3 Setting Up

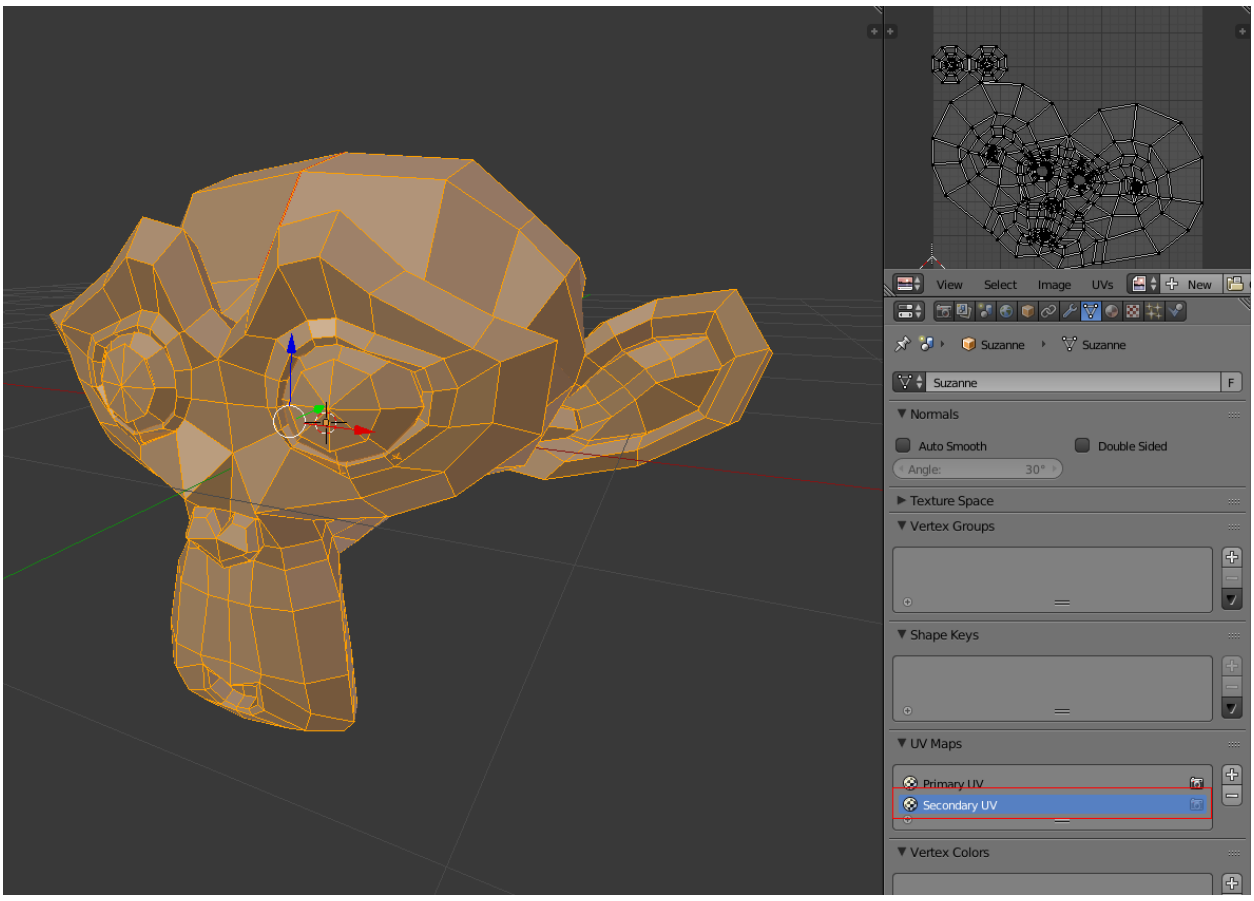
First of all, before the lightmapper can do anything, objects to be baked need an UV2 layer and a texture size. An UV2 layer is a set of secondary texture coordinates that ensures any face in the object has it's own place in the UV map. Faces must not share pixels in the texture.

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 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.

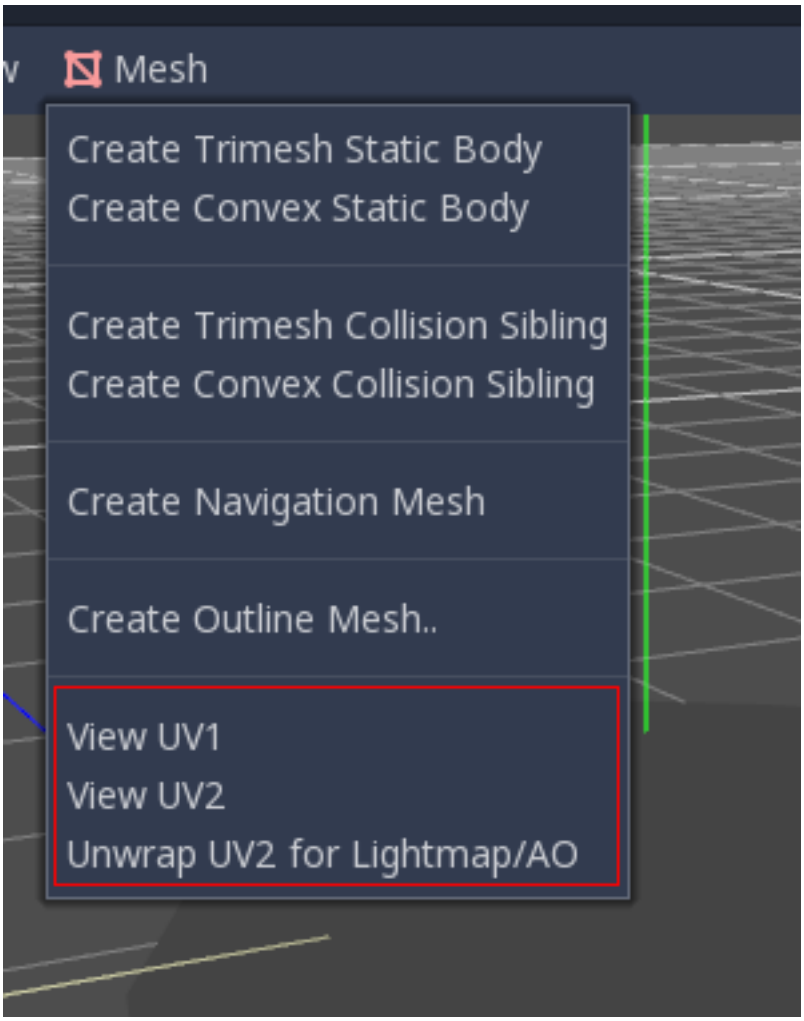


And import normally. Remember you will need to set the texture size on the mesh after import.

If you use external meshes on import, the size will be kept. Be wary that most unwrappers in 3D DCCs are not 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

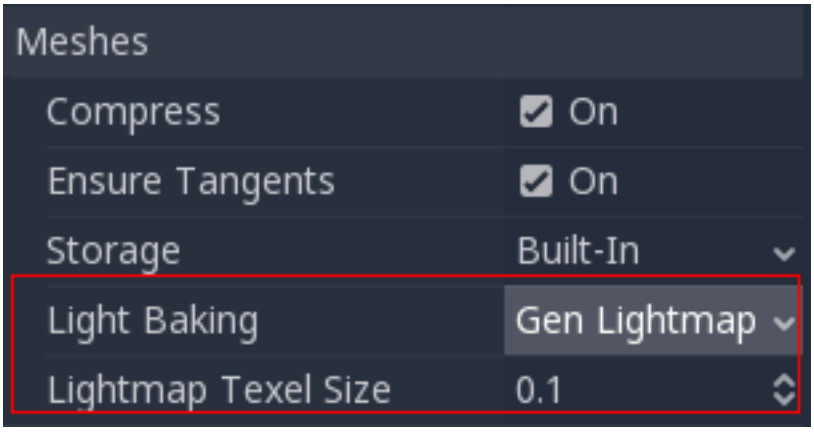
Godot has an option to unwrap meshes and visualize the UV channels. It can be found in the Mesh menu:



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:



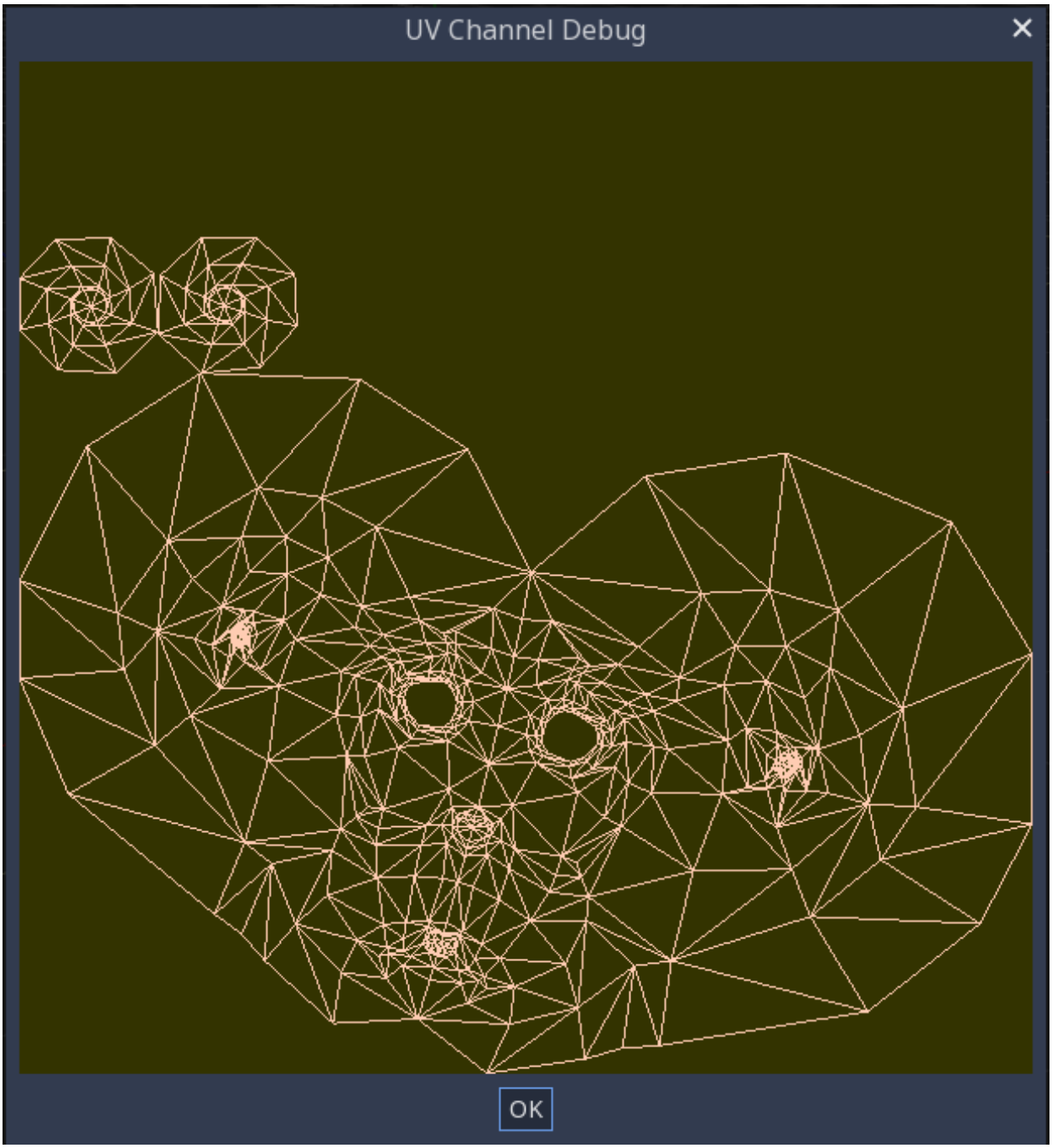
The **Light Baking** mode needs to be set to **"Gen Lightmaps"**. A texel size in world units must also be provided, as this will determine the final size of the lightmap texture (and, in consequence, the UV padding in the map).

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 different scales if you are planning to use lightmapping.

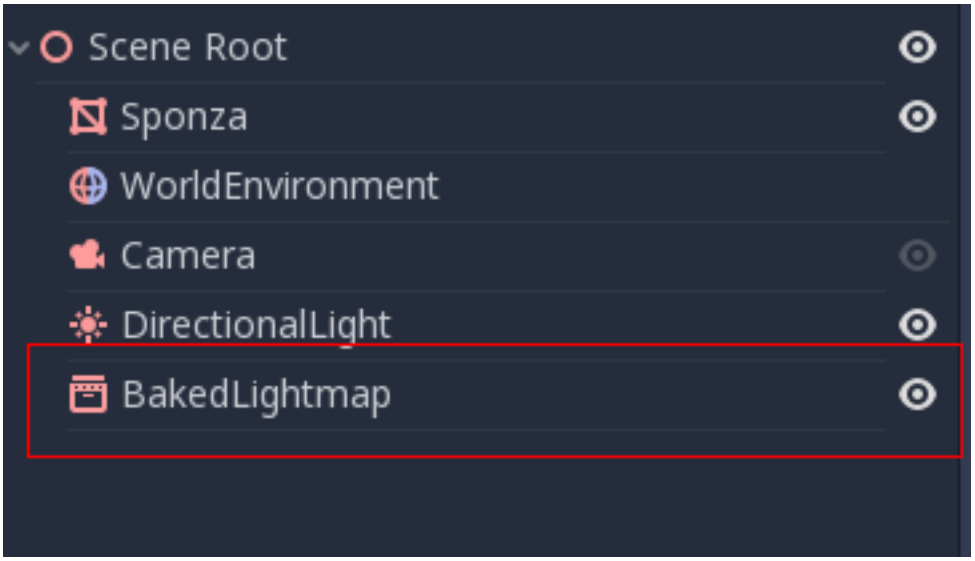
Checking UV2

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:



7.8.4 Setting up the Scene

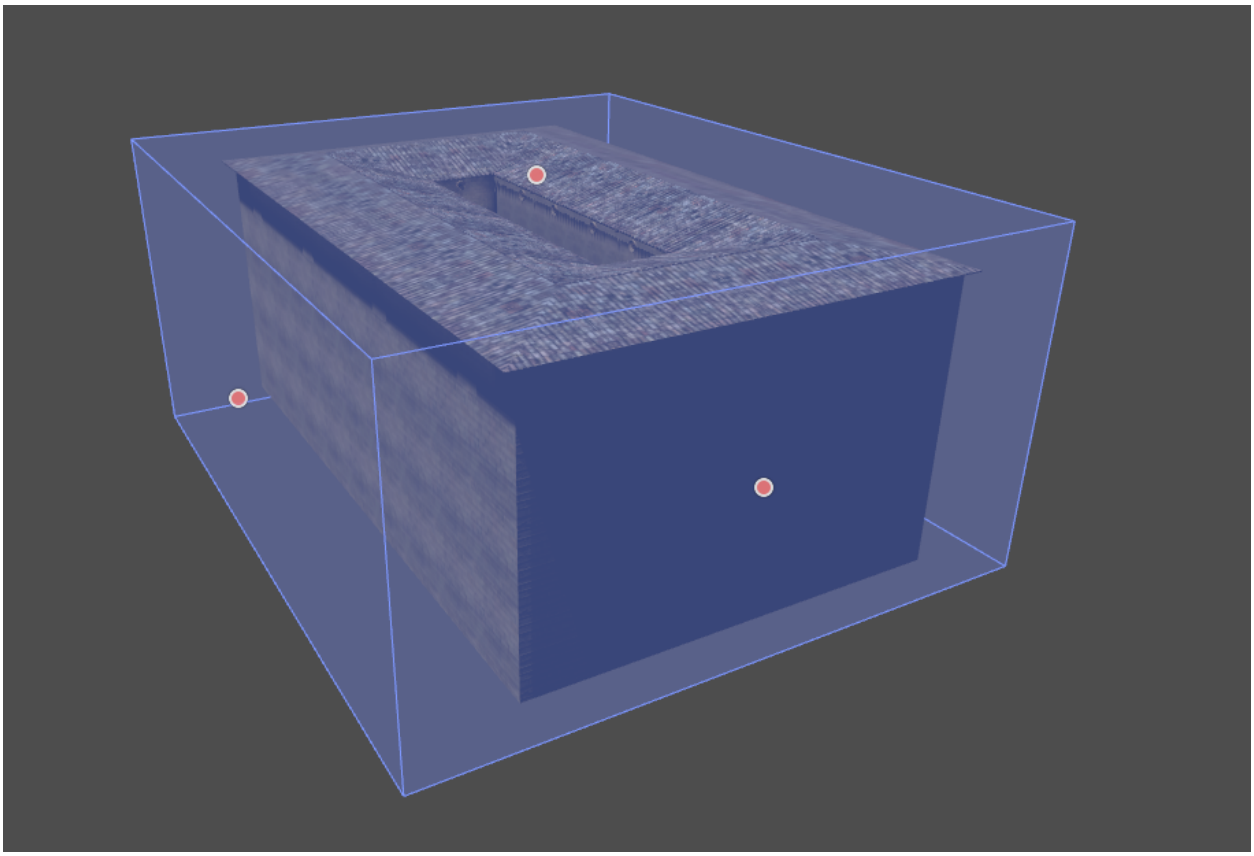
Before anything is done, a **BakedLight** Node needs to be added to a scene. This will enable light baking on all nodes (and sub-nodes) in that scene, even on instanced scenes.



A sub-scene can be instanced several times, as this is supported by the baker and each will be assigned a lightmap of it's own (just make sure to respect the rule about scaling mentioned before):

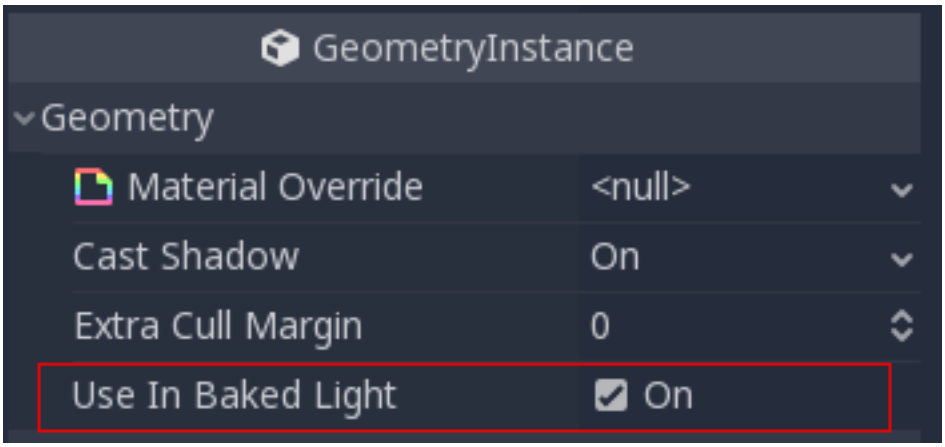
Configure Bounds

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.

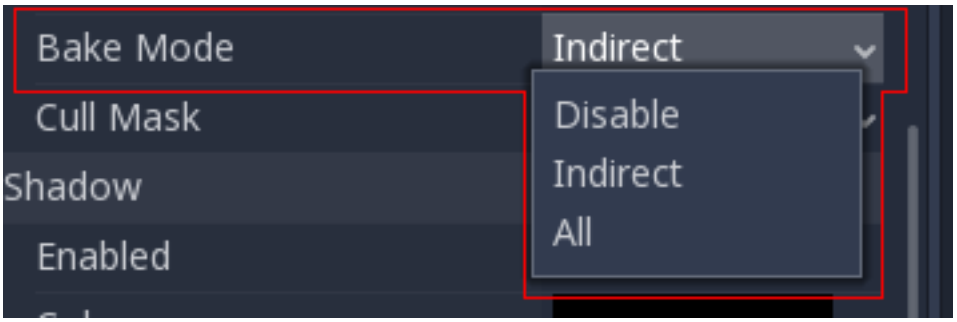


When auto-generating lightmaps on scene import, this is enabled automatically.

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:



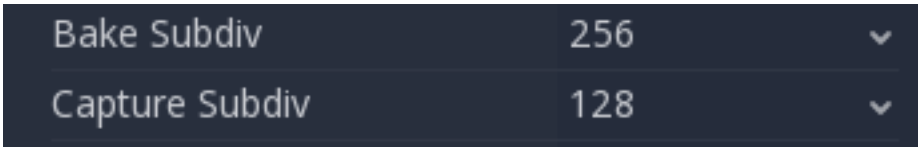
The modes are :

- **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.

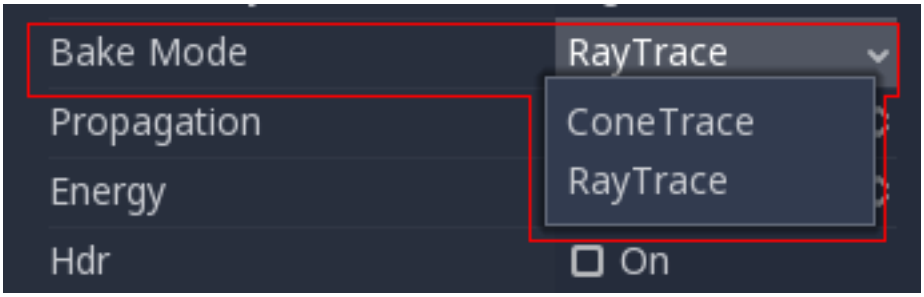
Baking Quality

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). It's default value is also good enough for more cases.



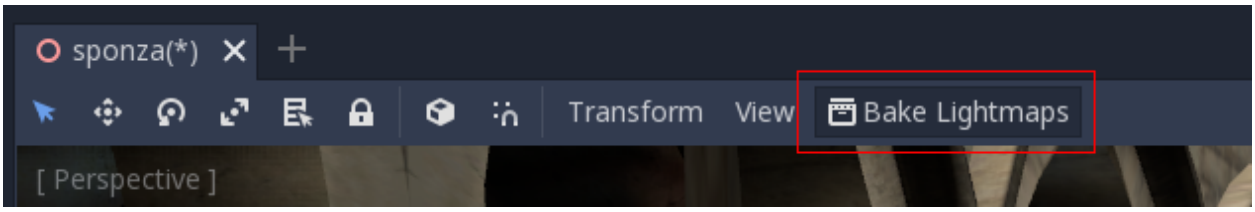
Besides the capture size, quality can be modified by setting the **Bake Mode**. Two modes of capturing indirect are provided:



- **Voxel Cone**: Trace: Is the default one, it's less precise but 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.

7.8.5 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 small details or your scene is 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.

7.8.6 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.

7.9 Environment and Post-Processing

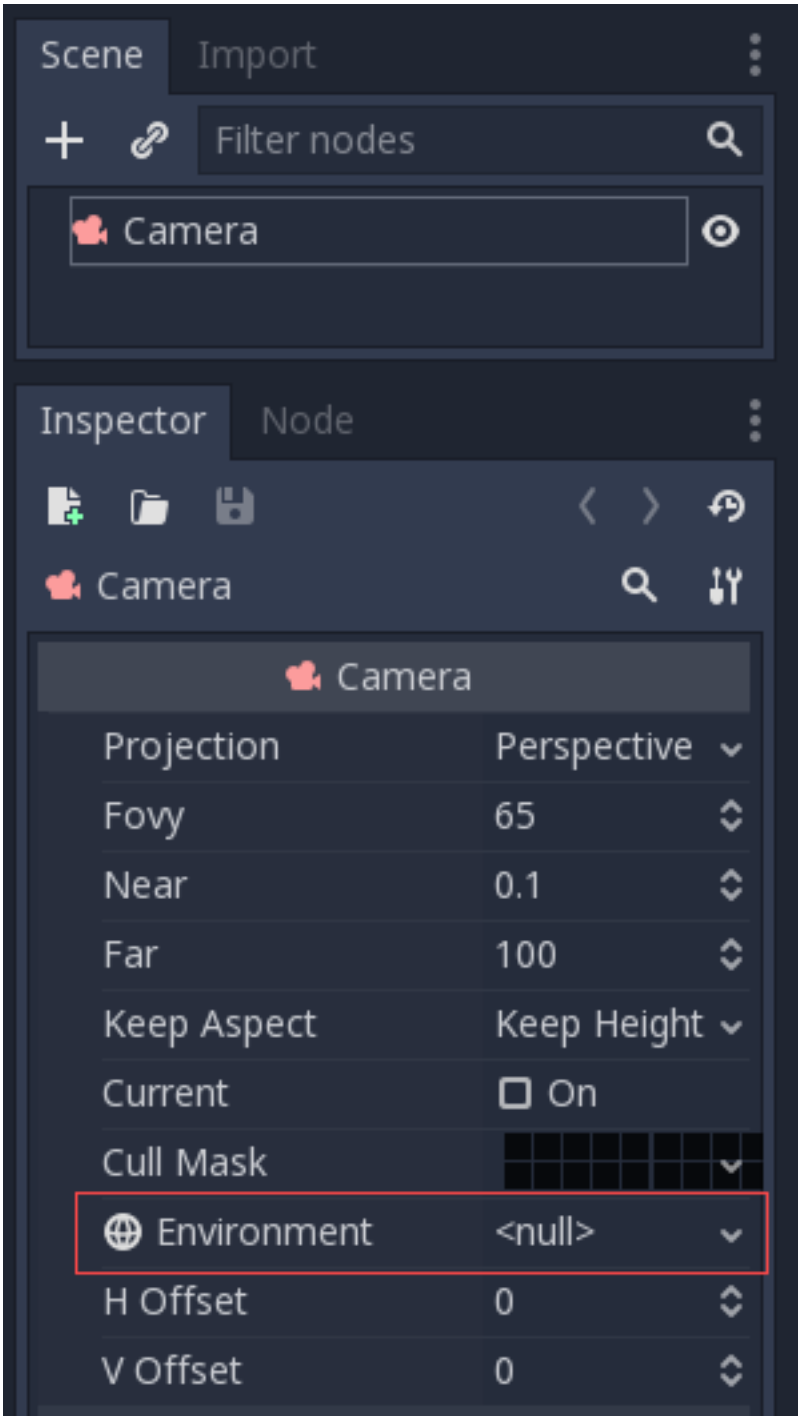
Godot 3 provides a redesigned Environment resource, as well as a brand new post-processing system with many available effects right out of the box.

7.9.1 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

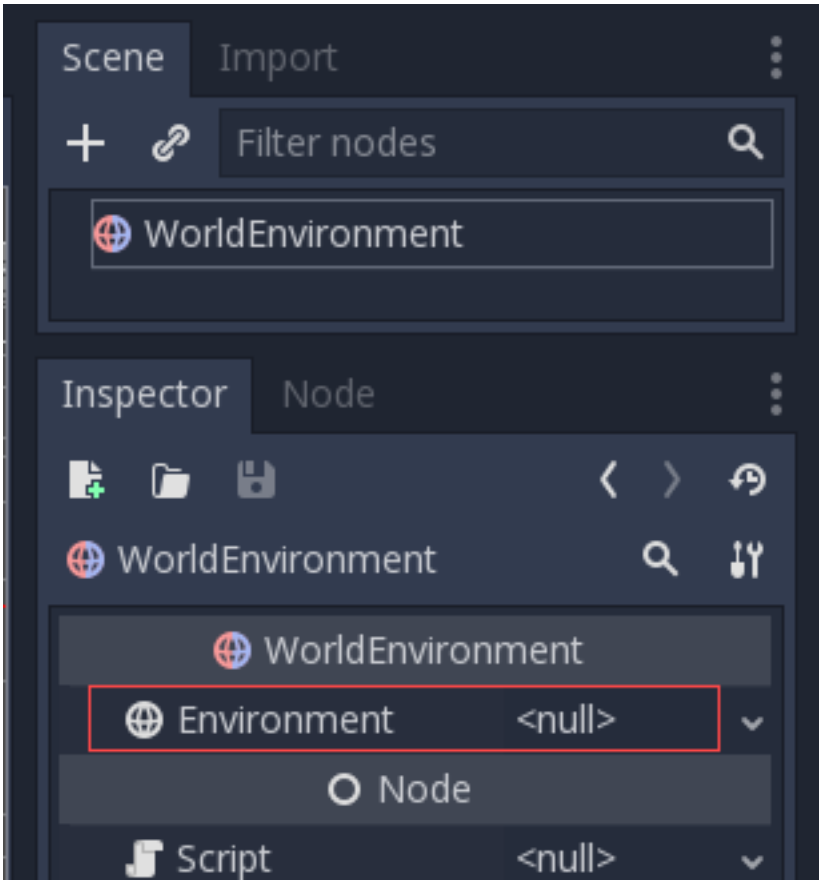
An Environment can be set to a camera. It will have priority over any other setting.



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:

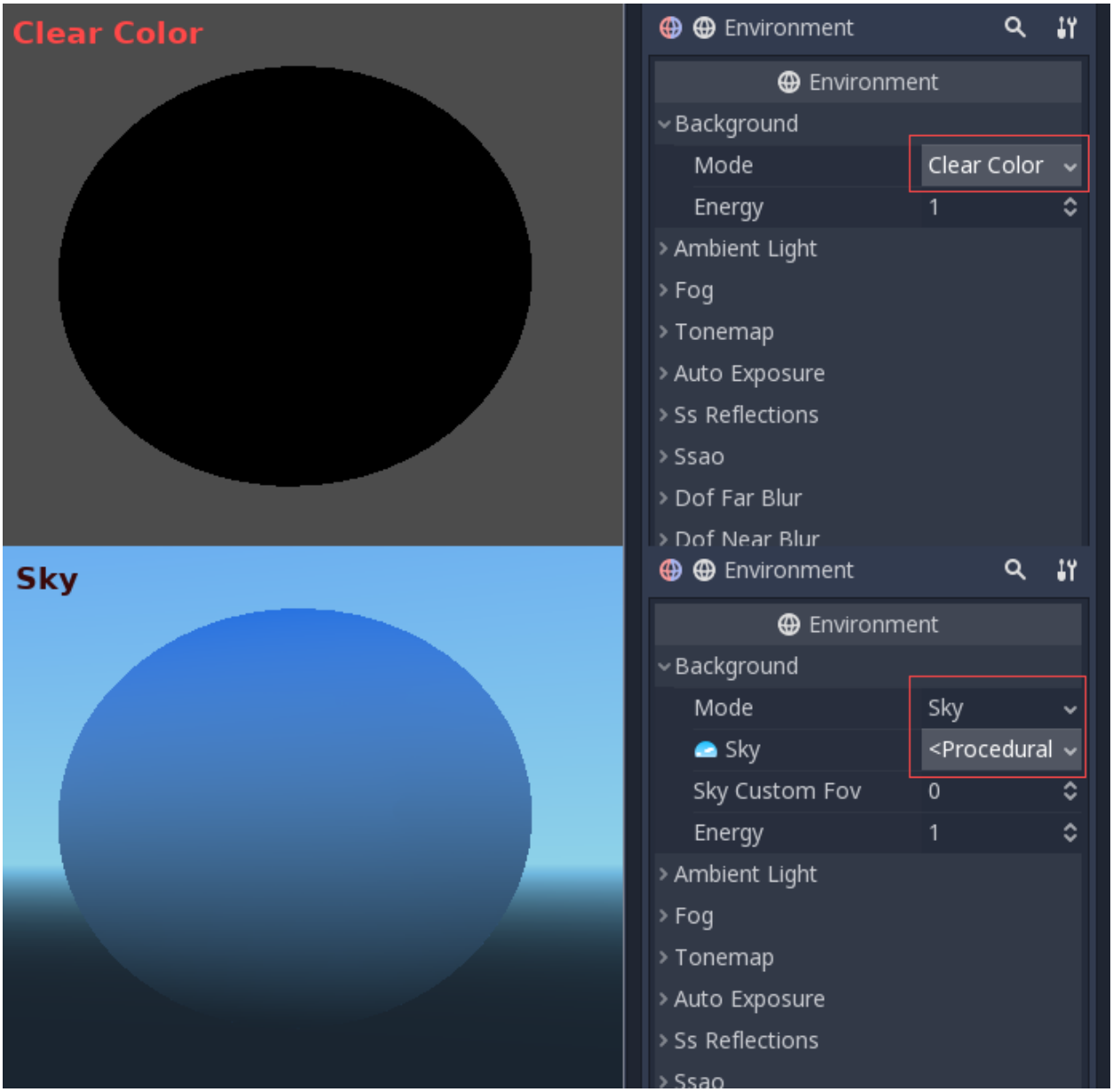
New projects created from the Project Manager come with a default environment (`default_env.tres`). If one needs to be created, save it to disk before referencing it here.

7.9.2 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 where 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.

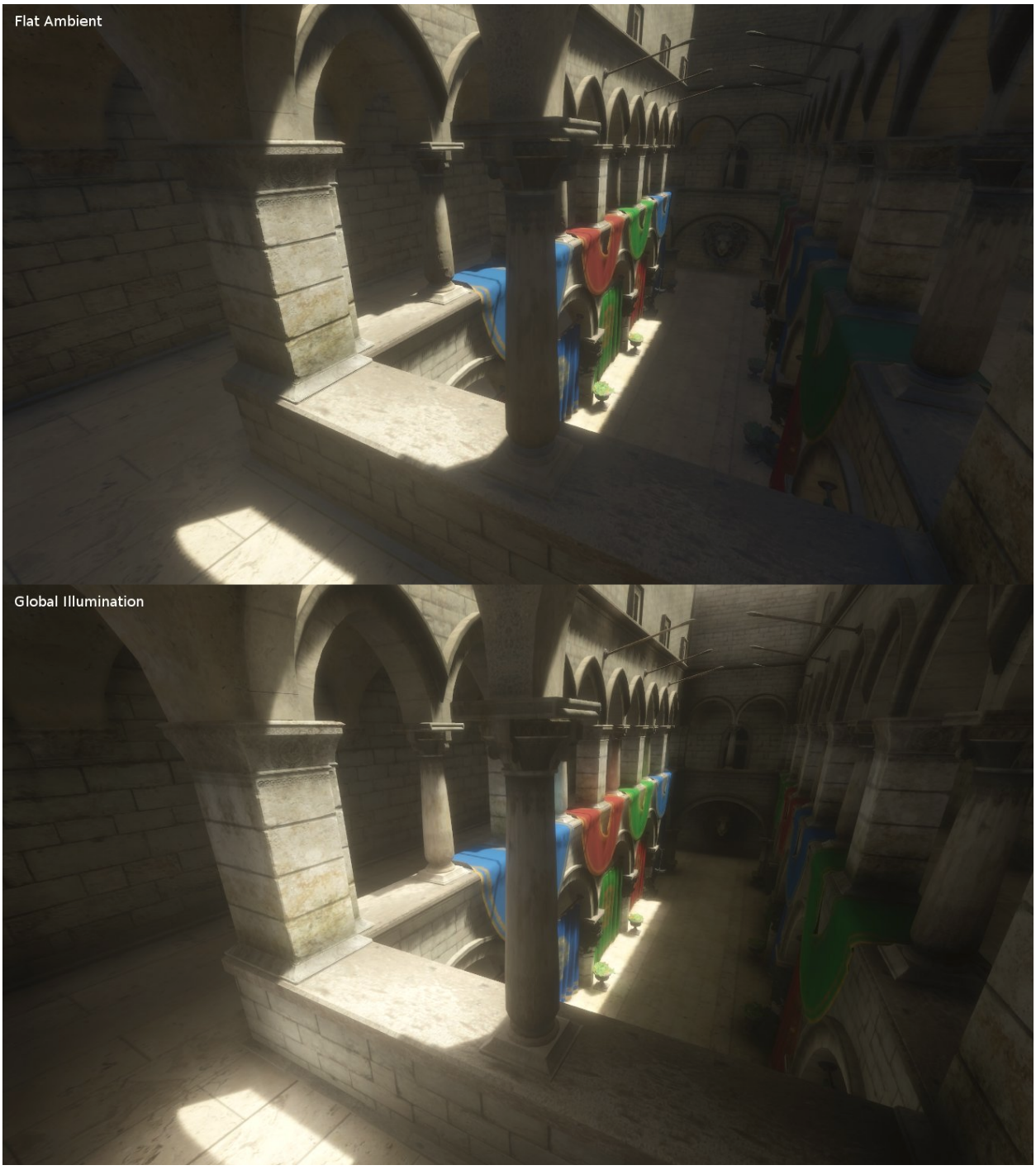
There are two types of ambient light, the *Ambient Color* (which is a constant color multiplied by the material albedo), and then one obtained from the *Sky* (as described before, but a sky needs to be set as background for this to be enabled).

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 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, 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, Reindhart, 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).

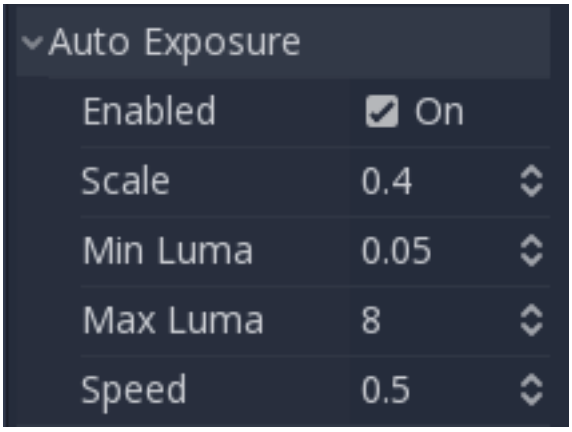
Auto Exposure (HDR)

Even though, in most cases, lighting and texturing are heavily artist controlled, Godot suports 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 expure 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 the with the directional light. Normally, values between 3.0 and 6.0 are enough to simulate indoor-oudoor 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 then:



- **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.

7.9.3 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 scren-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.

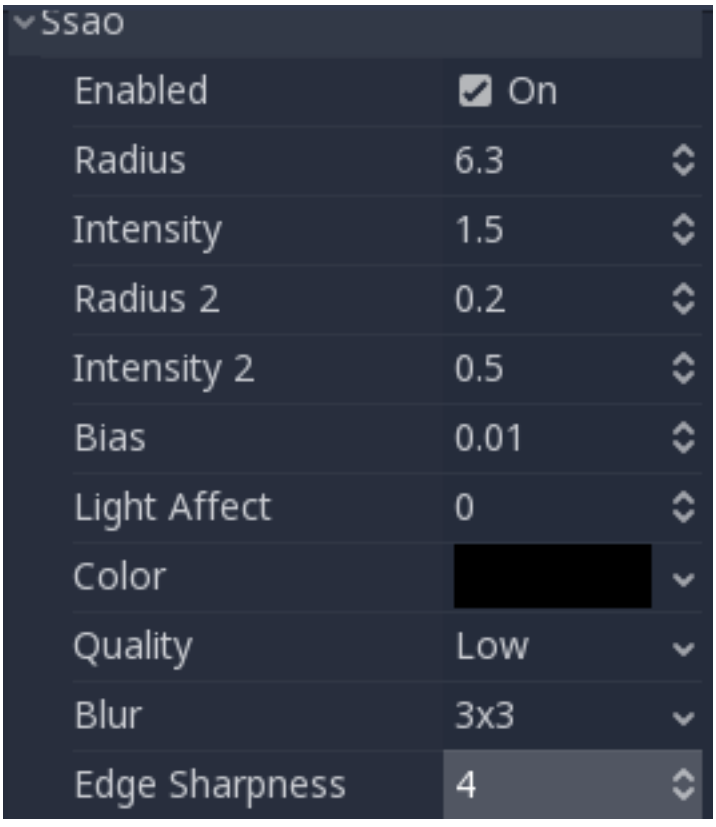
This can be simulated with Screen Space Ambient Occlusion. As you can see in the image below, the goal of it is to make sure concave areas are darker, simulating a narrower path for the light to enter:

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, 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:



- **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).

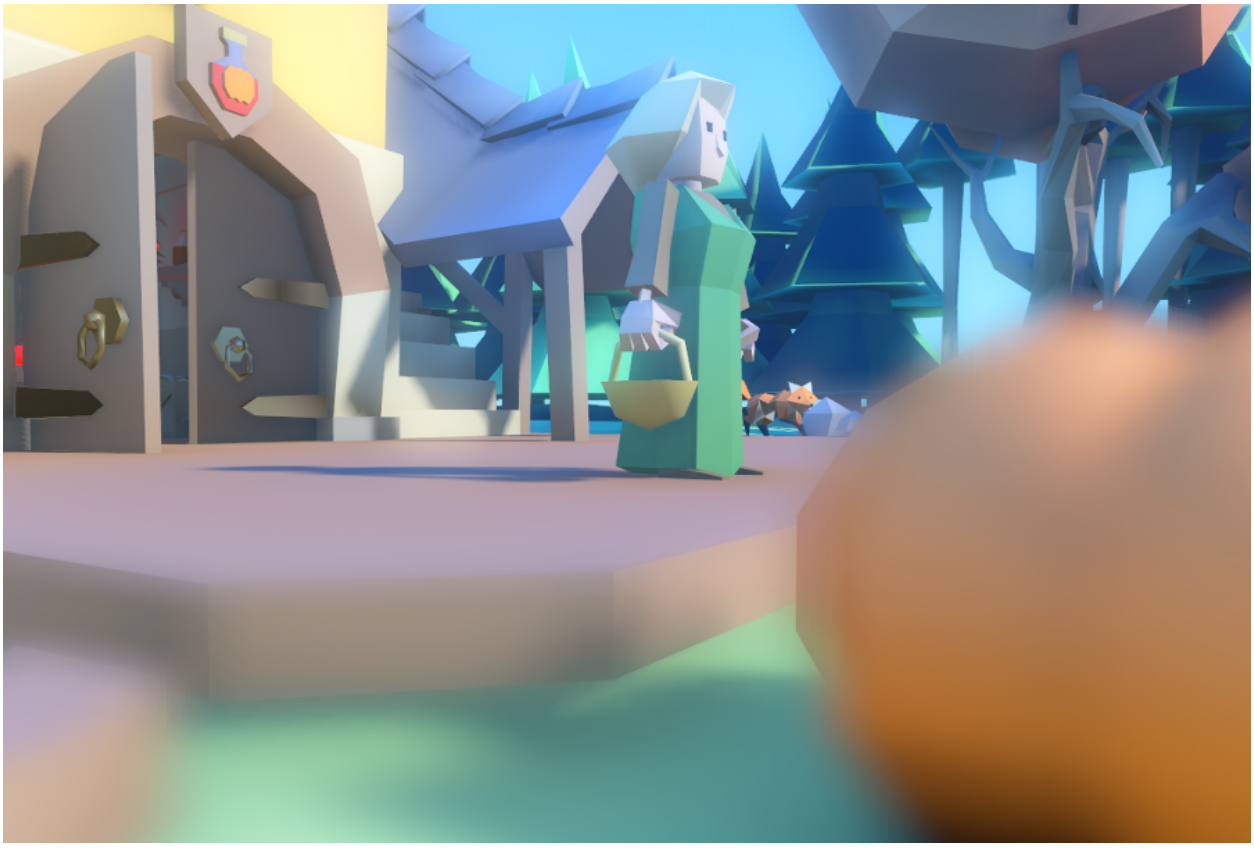
Depth of Field / Far Blur

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 arctifacts.

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 arctifacts.

It is common to use both blurs together to focus the viewer's attention on a given object:

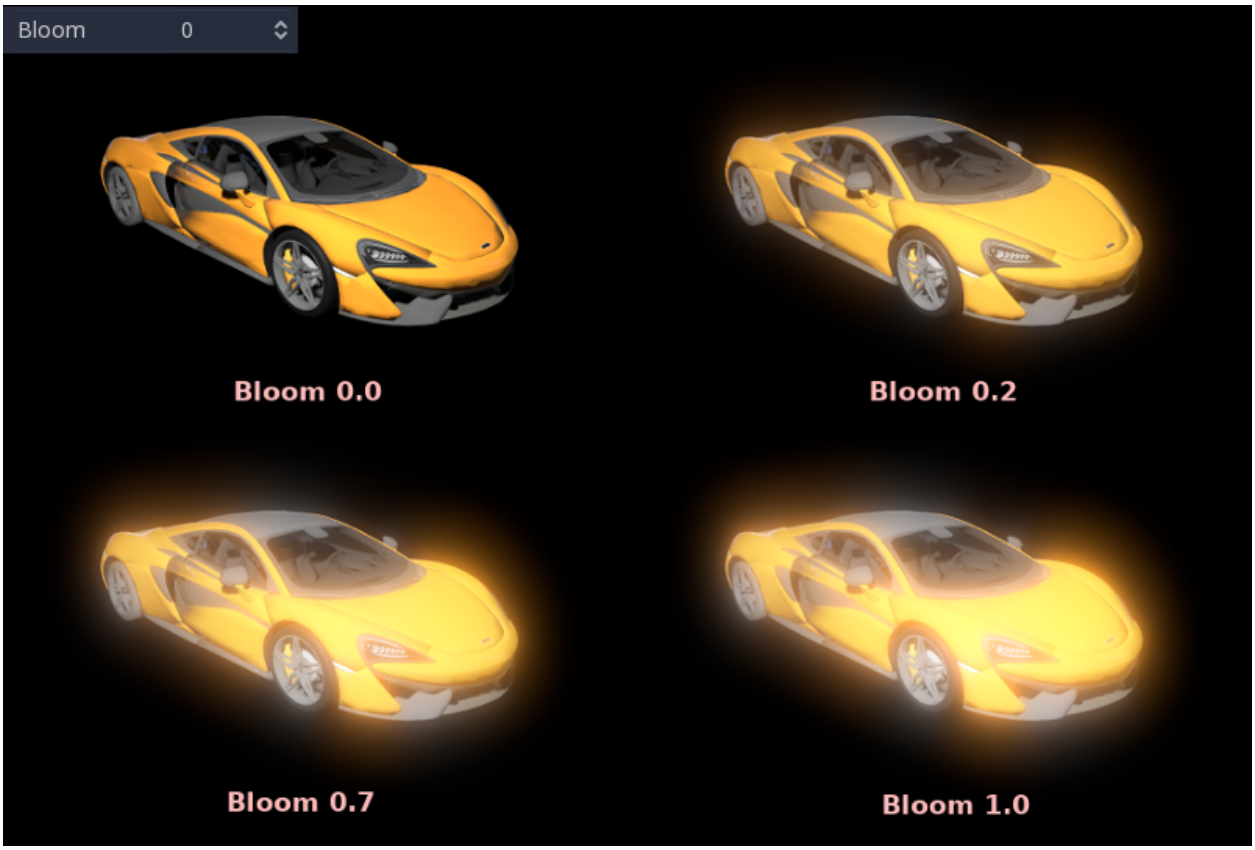
Glow

In photography and film, when light amount exceeds the maxium 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 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 surpasing 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.

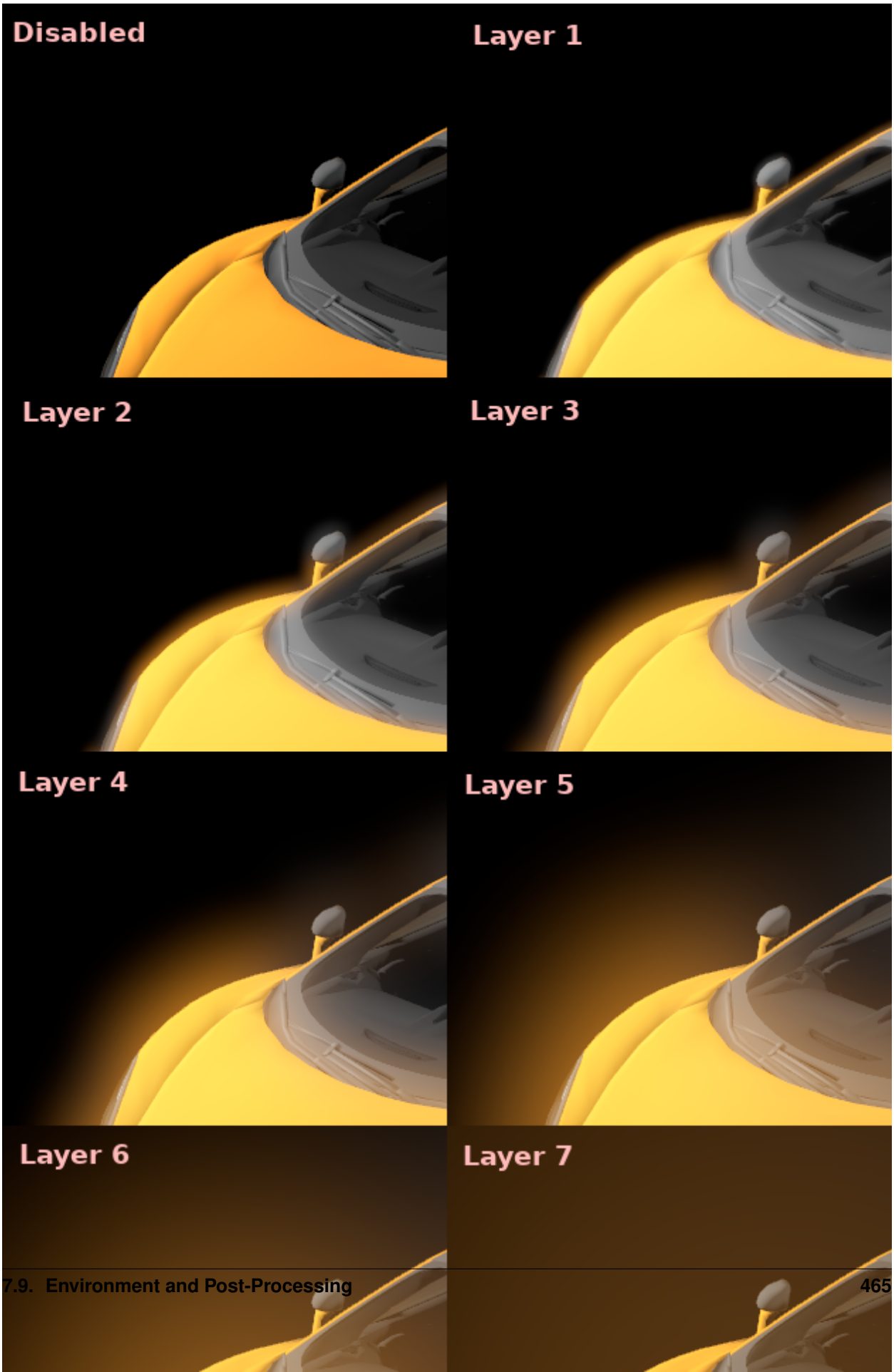
Once glow is visible, it can be controlled with a few extra parameters:

- **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:


Disabled
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Layer 6
Layer 7

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 rids of it, at a minimal performance cost.

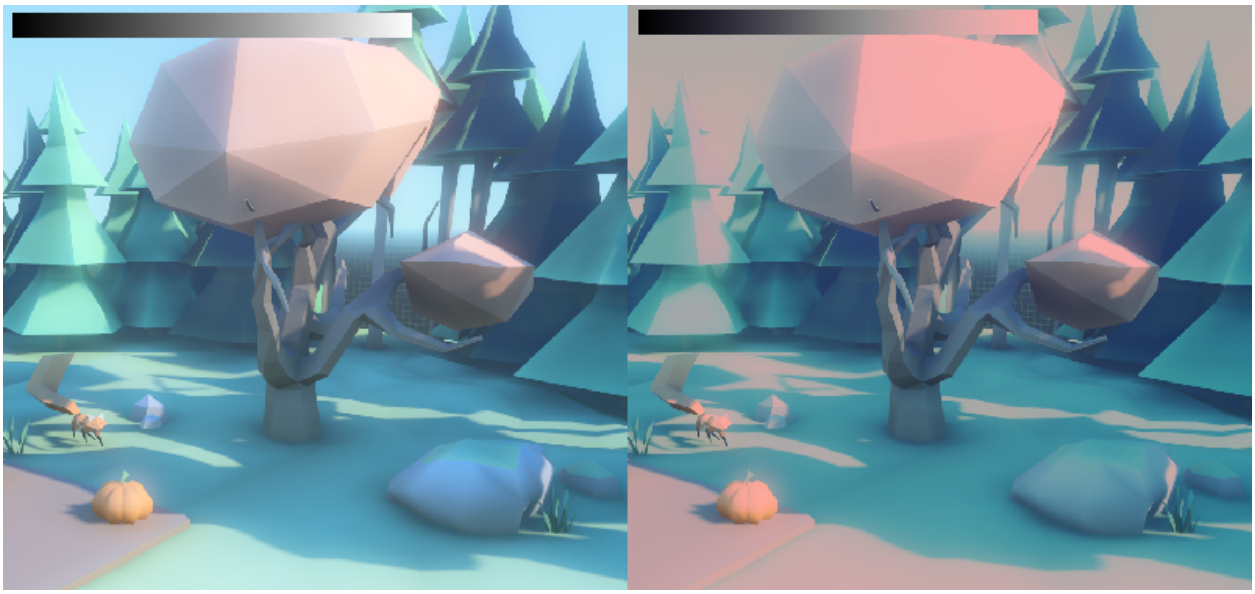
Adjustments

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:



7.10 High dynamic range

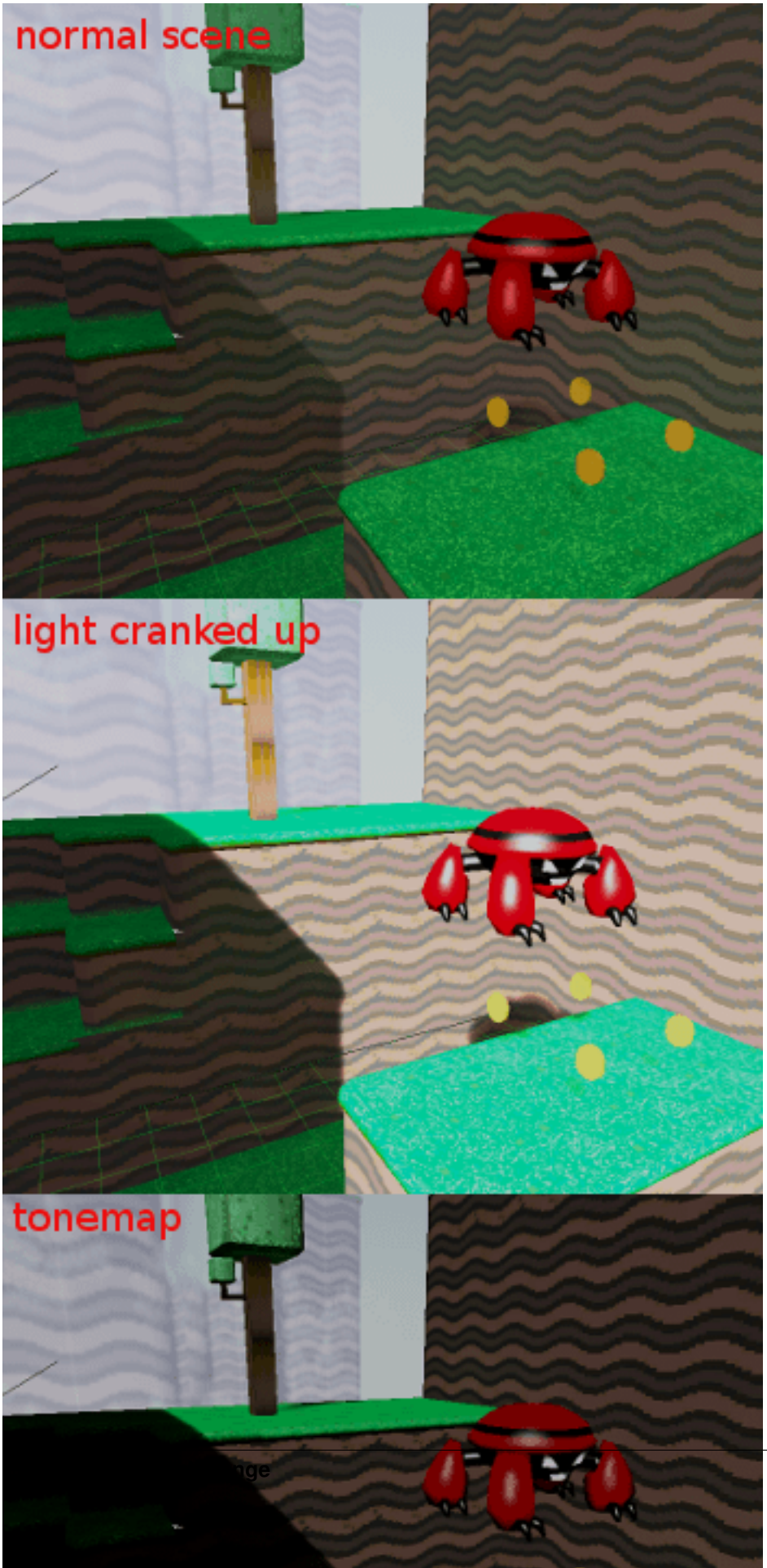
7.10.1 Introduction

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:


normal scene
light cranked up
tonemap

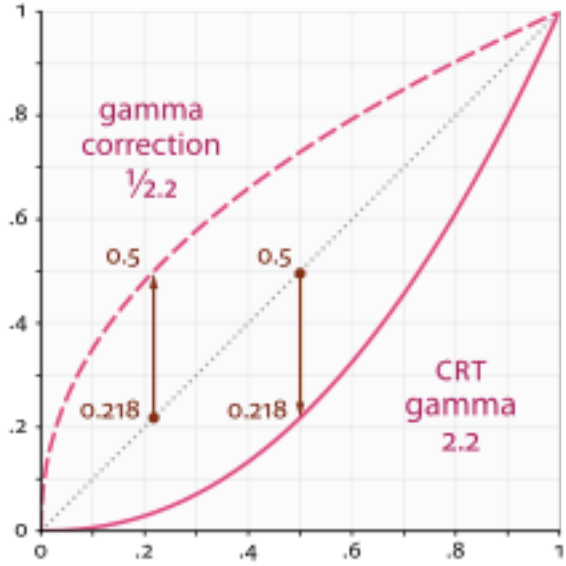
Except the scene is more contrasted, because there is a higher light range in play. What is this all useful for? The idea is that the scene luminance will change while you move through the world, allowing situations like this to happen:

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.

7.10.2 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.

7.10.3 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 no 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.

7.10.4 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.

7.11 Using gridmaps

7.11.1 Introduction

Gridmaps are a tool for creating 3D game levels, similar to the way *TileMap* works in 2D. You start with a predefined collection of 3D meshes (a *MeshLibrary*) that can be placed on a grid, as if you were building a level with an unlimited amount of Lego blocks.

Collisions and navigation can also be added to the meshes, just like you would do with the tiles of a tilemap.

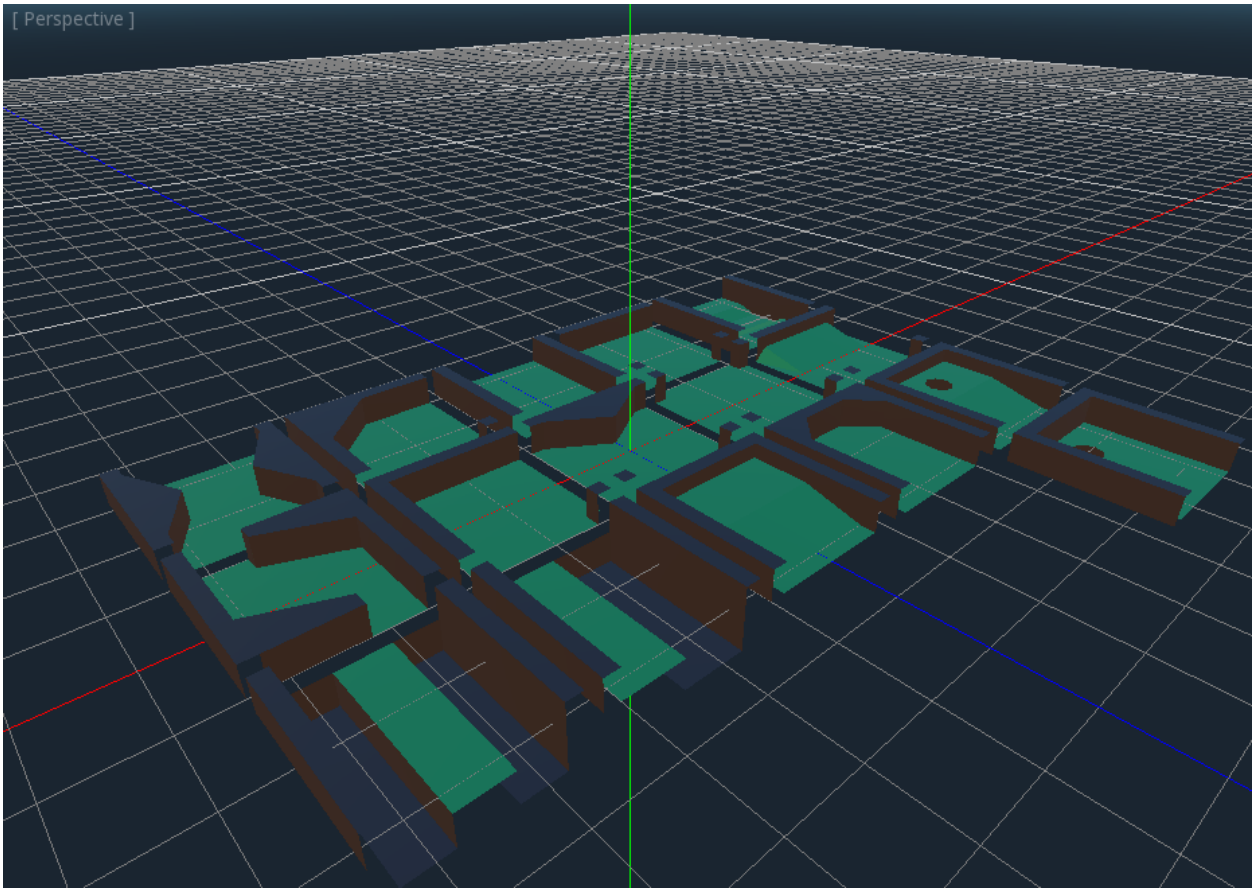
7.11.2 Example Project

To learn how GridMaps work, start by downloading the sample project: `gridmap_demo.zip`.

Unzip this project and add it to the Project Manager using the "Import" button.

7.11.3 Creating a MeshLibrary

To begin, you need a *MeshLibrary*, which is a collection of individual meshes that can be used in the gridmap. Open the "MeshLibrary_Source.tscn" scene to see an example of how to set up the mesh library.

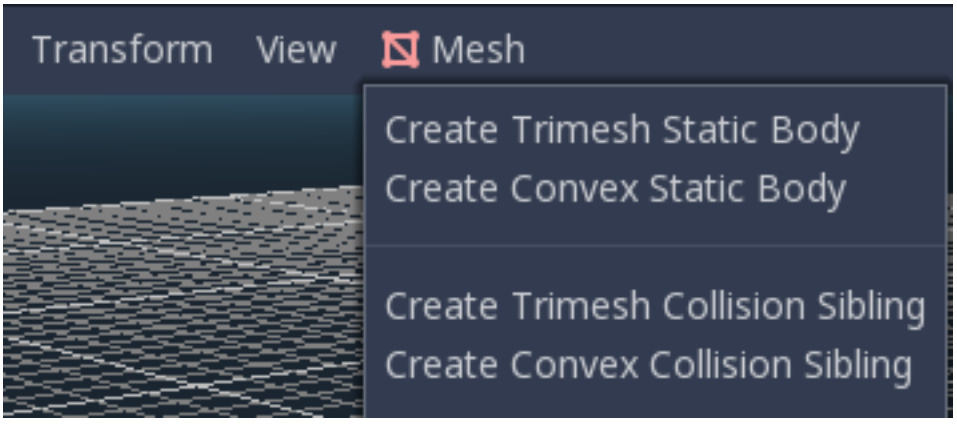


As you can see, this scene has a *Spatial* node as its root, and a number of *MeshInstance* node children.

If you don't need any physics in your scene, then you're done. However, in most cases you'll want to assign collision bodies to the meshes.

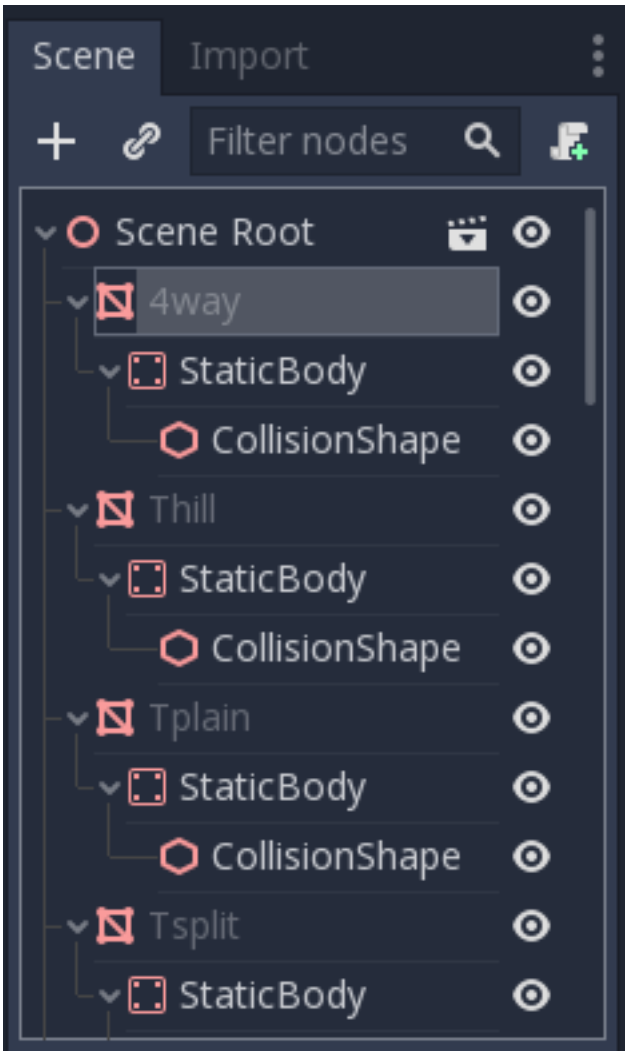
7.11.4 Collisions

You can manually assign a *StaticBody* and *CollisionShape* to each mesh. Alternatively, you can use the "Mesh" menu to automatically create the collision body based on the mesh data.



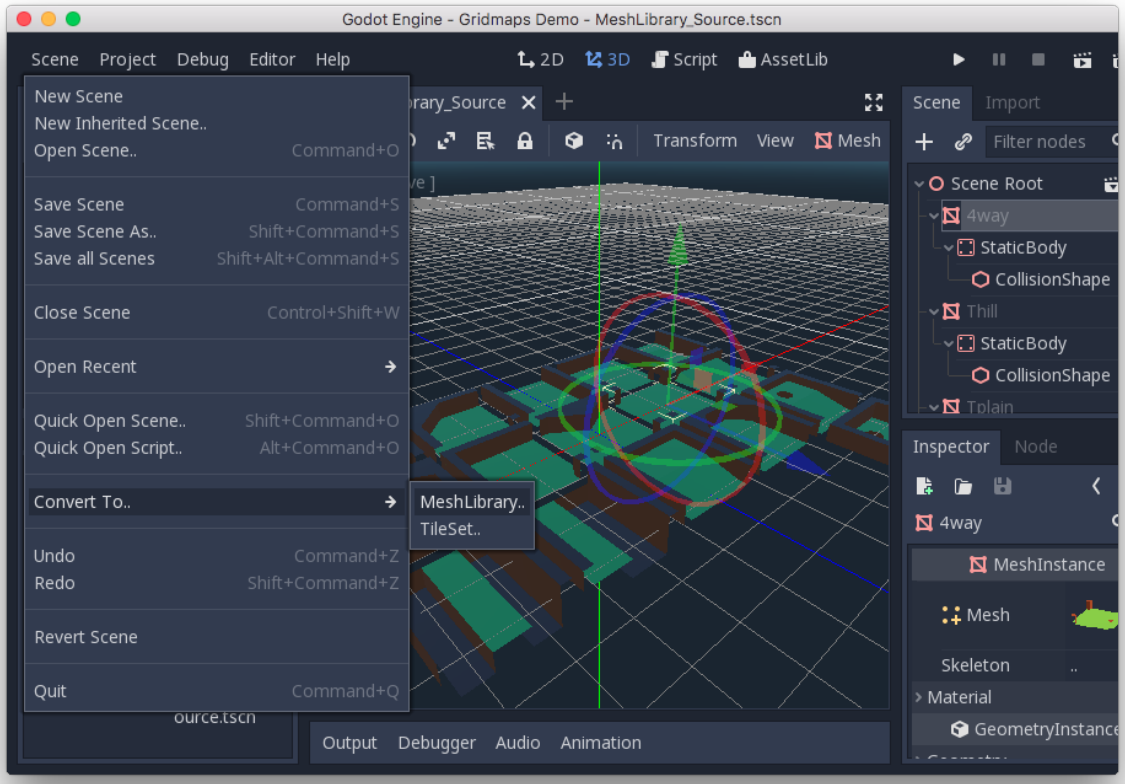
Note that a "Convex" collision body will work better for simple meshes. For more complex shapes, select "Create Trimesh Static Body". Once each mesh has a physics body and collision shape assigned, your mesh library is ready to

be used.



7.11.5 Exporting the MeshLibrary

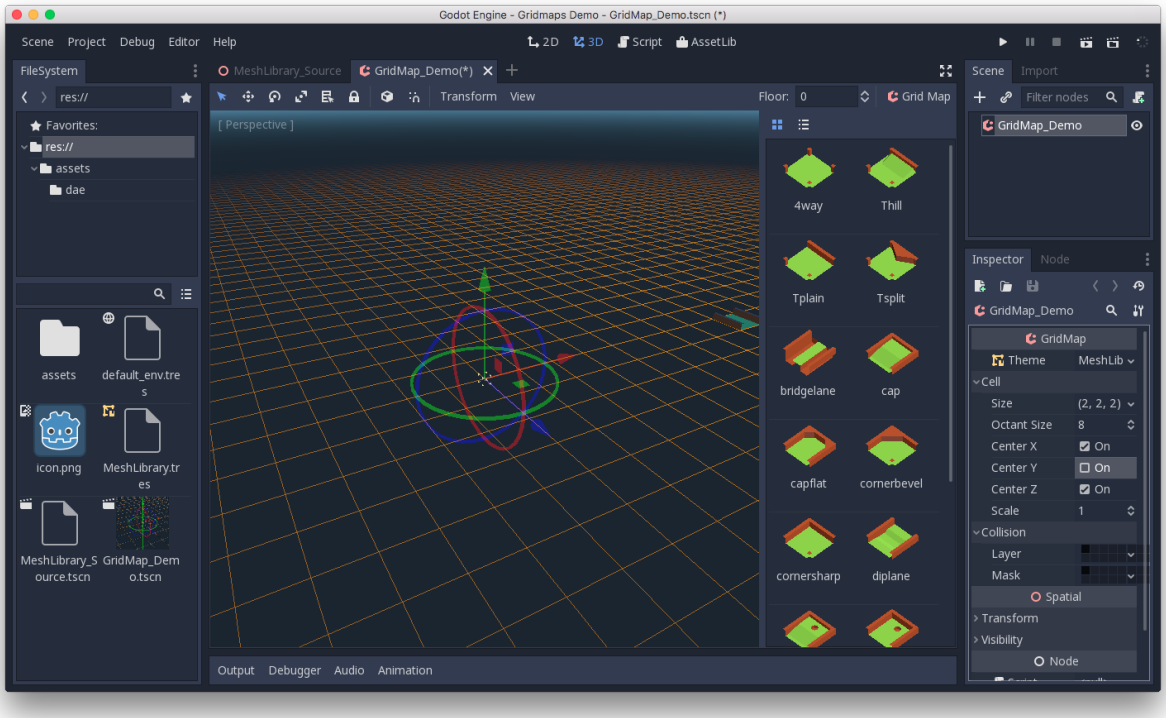
To export the library, click on Scene -> Convert To.. -> MeshLibrary.., and save it as a resource.



You can find an already exported MeshLibrary in the project named “MeshLibrary.tscn”.

7.11.6 Using GridMap

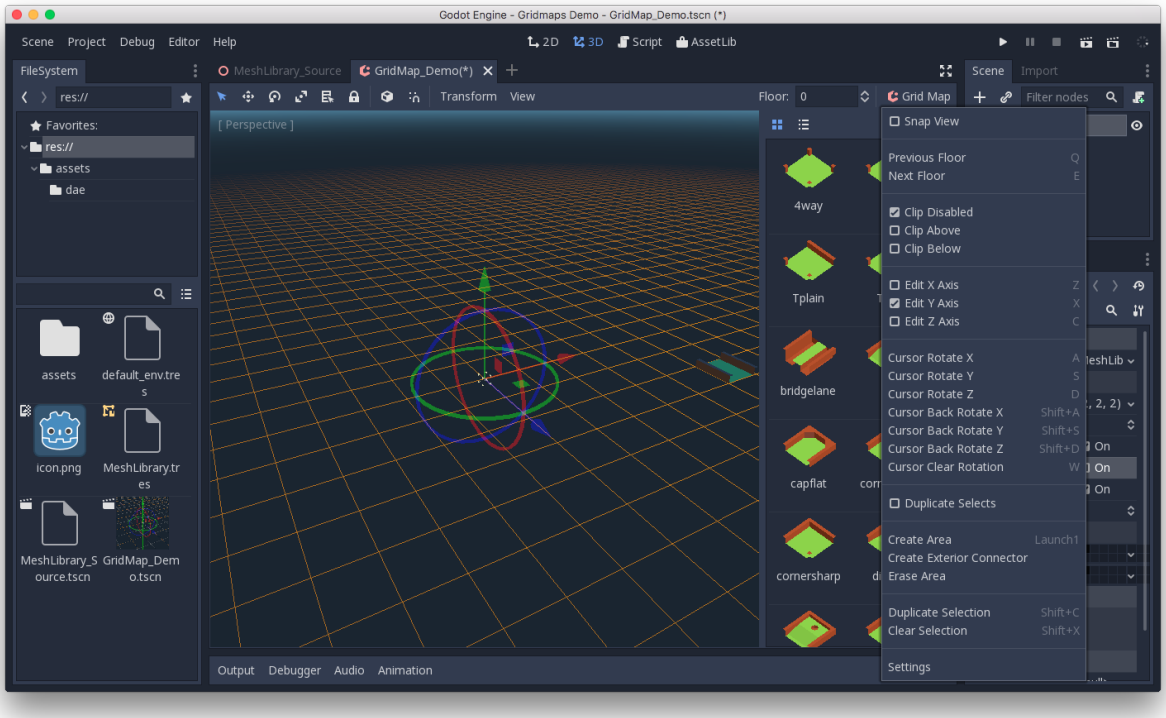
Create a new scene and add a GridMap node. Add the mesh library by dragging the resource file from the FileSystem dock and dropping it in the “Theme” property in the Inspector.



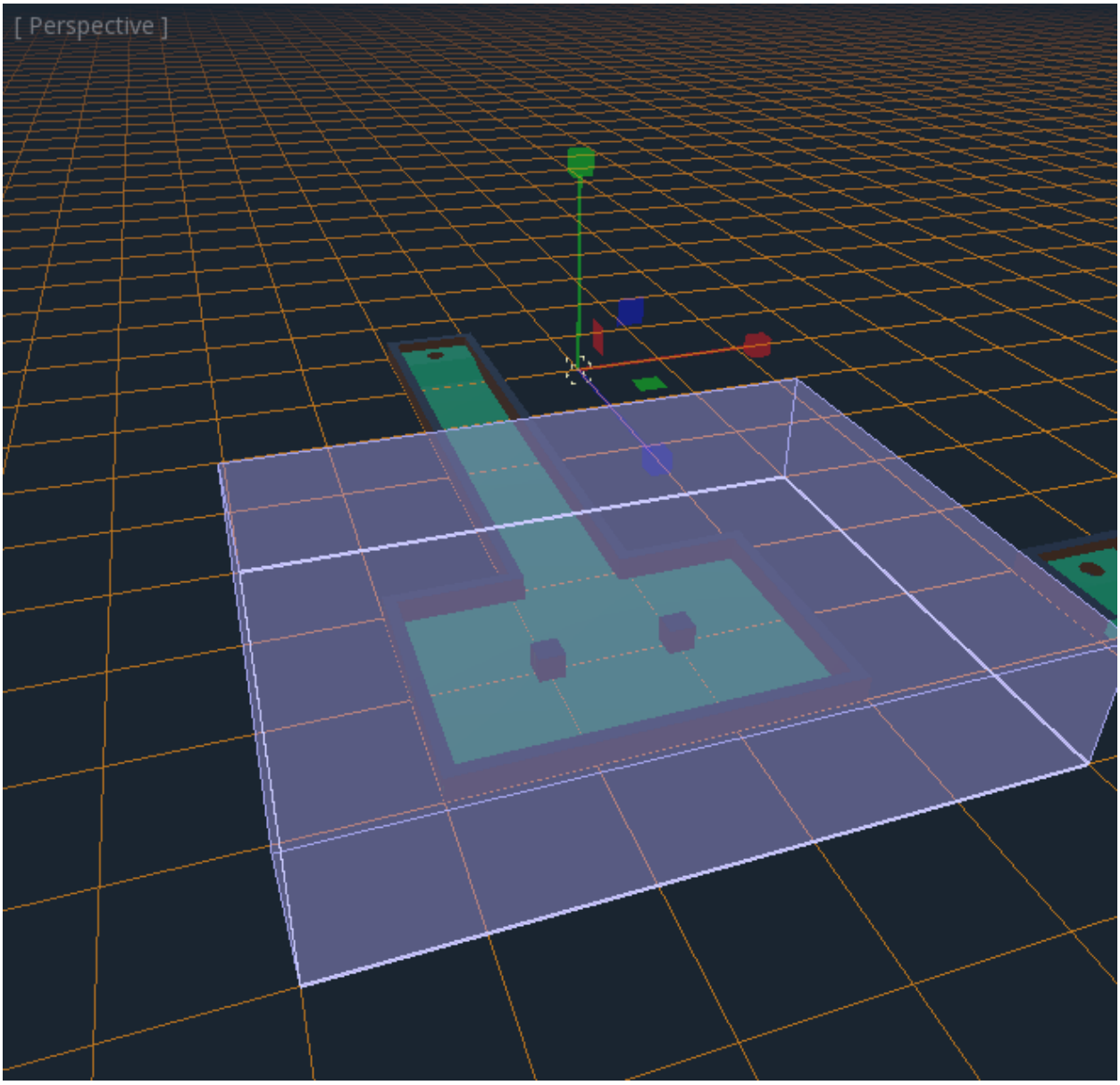
The “Cell/Size” property should be set to the size of your meshes. You can leave it at the default value for the demo. Set the “Center Y” property to “Off”.

Now you can start designing the level by choosing a tile from the palette and placing it with Left-Click in the editor window. To remove a tile, use Shift+Right-click.

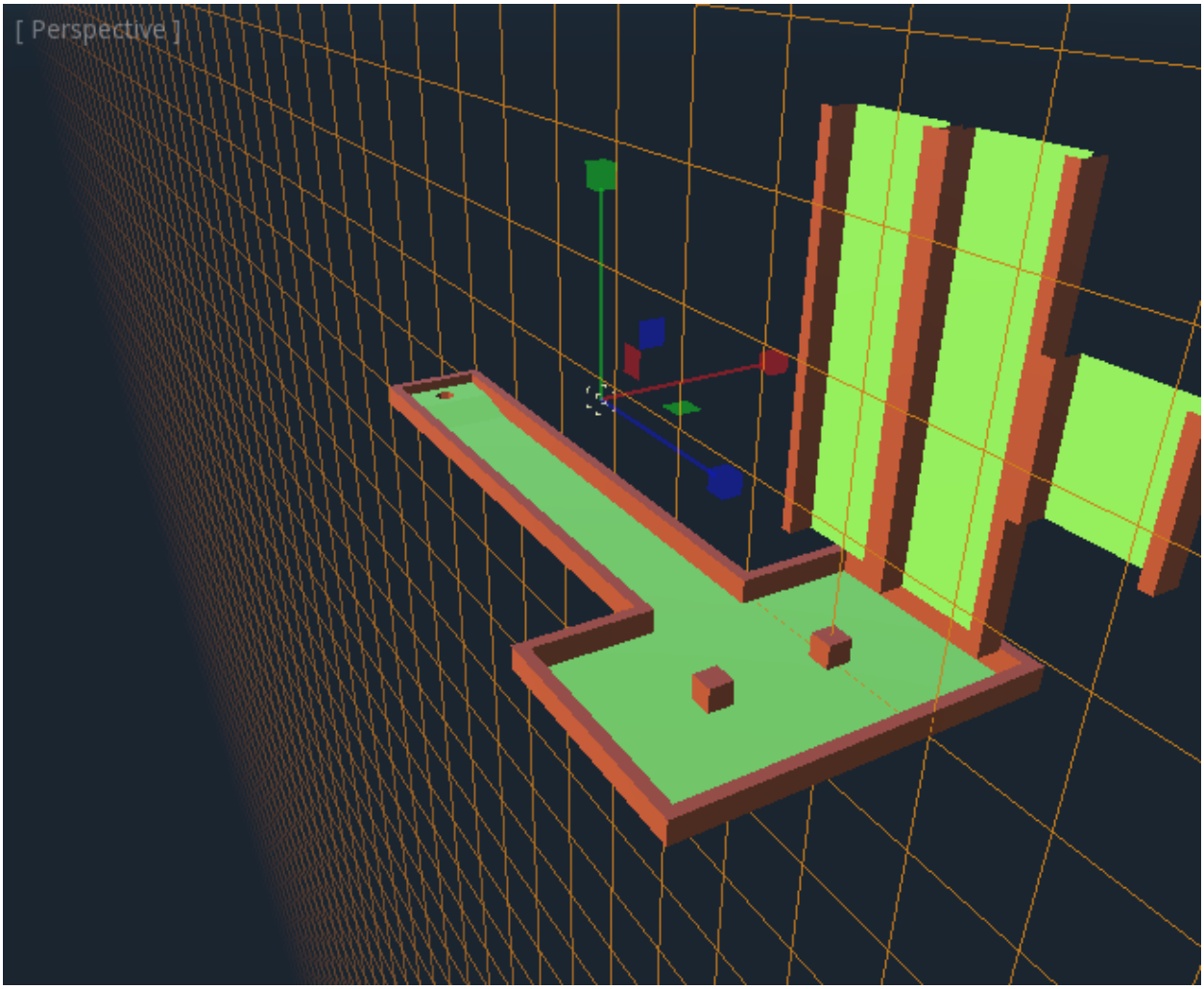
Click on the “GridMap” menu to see options and shortcuts. For example, pressing “S” rotates a tile around the y-axis.



Holding <Shift> and dragging with the left mouse button will draw a selection box. You can duplicate or clear the selected area using the respective menu options.



In the menu, you can also change the axis you're drawing on as well as shift the drawing plane higher or lower on its axis.



7.11.7 Using gridmap in code

See *GridMap* for details on the node's methods and member variables.

7.12 Using MultiMeshInstance

7.12.1 Introduction

In a normal scenario you would use a *MeshInstance* node to display a 3D mesh like a human model for the main character. But in some cases you would like to create multiple instances of the same mesh in a scene. You *could* duplicate the same node multiple times and adjust the transforms manually. This may be a tedious process and the result may look mechanical. Also, this method is not favourable to rapid iterations. *MultiMeshInstance* is one of the possible solutions to this problem.

MultiMeshInstance, as the name suggests, creates multiple copies of a MeshInstance over a surface of a specific mesh. An example would be having a tree mesh populate a landscape mesh with random scales and orientations.

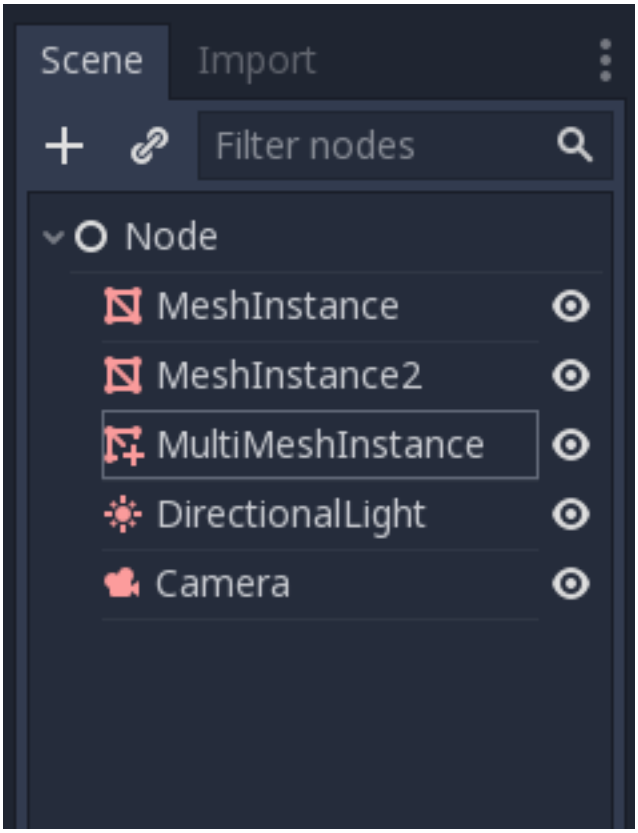
7.12.2 Setting up the nodes

The basic setup requires three nodes. Firstly, the MultiMeshInstance node. Then, two MeshInstance nodes.

One node is used as the target, the mesh that you want to place multiple meshes on. In the tree example, this would be the landscape.

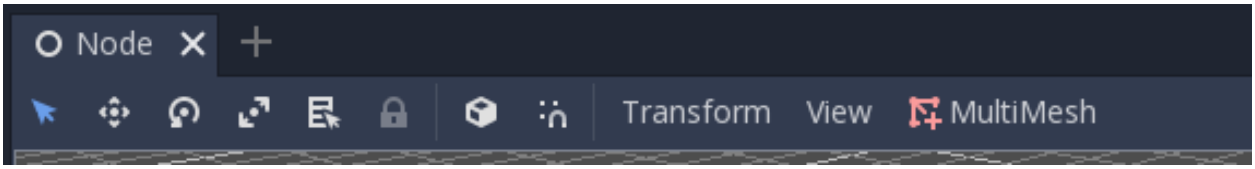
Another node is used as the source, the mesh that you want to have duplicated. In the tree case, this would be the tree.

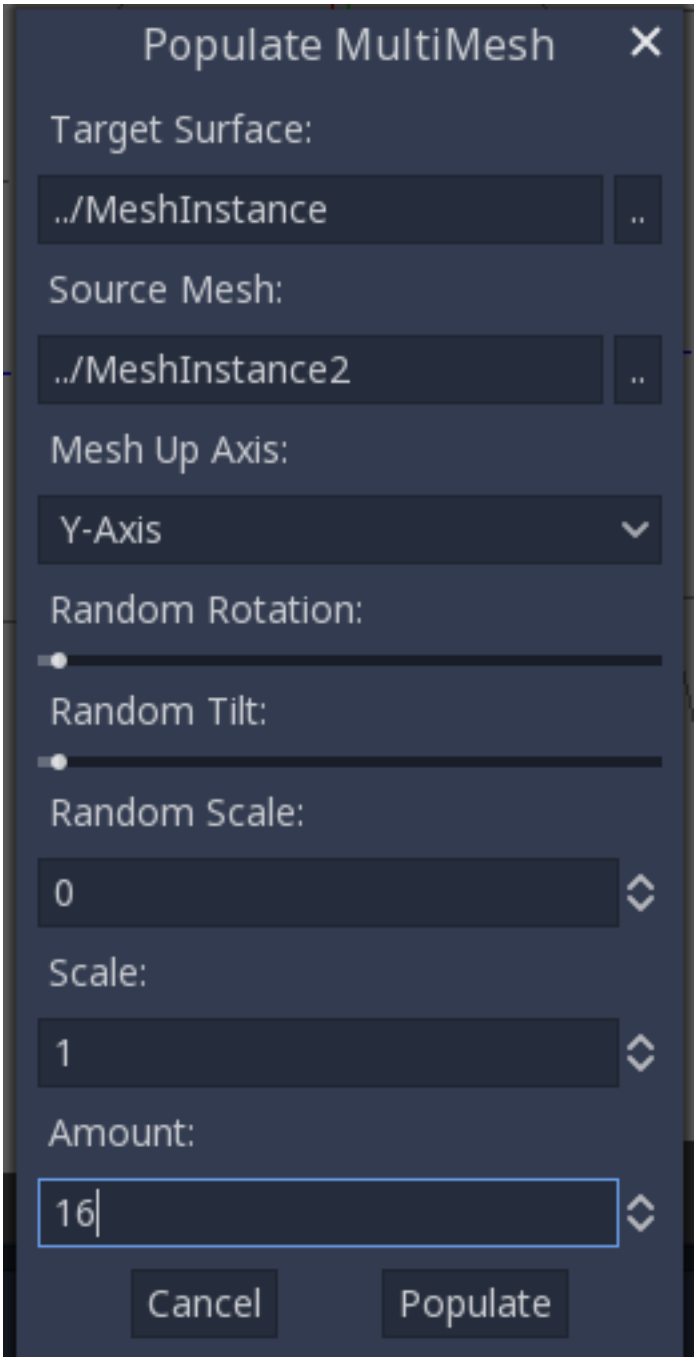
In our example, we would use a *Node* as the root node of the scene. Your scene tree would look like this:



Note: For simplification purposes, this tutorial uses built-in primitives.

Now you have everything ready. Select the MultiMeshInstance node and look at the toolbar, you should see an extra button called `MultiMesh` next to `View`. Click it and select *Populate surface* in the dropdown menu. A new window titled *Populate MultiMesh* will pop up.





7.12.3 MultiMesh Settings

Below are descriptions of the options.

Target Surface

The mesh you would be using as the target surface for placing copies of you source mesh on.

Source Mesh

The mesh you want duplicated on the target surface.

Mesh Up Axis

The axis used as the up axis of the source mesh.

Random Rotation

Randomizing the rotation around the mesh up axis of the source mesh.

Random Tilt

Randomizing the overall rotation of the source mesh.

Random Scale

Randomizing the scale of the source mesh.

Scale

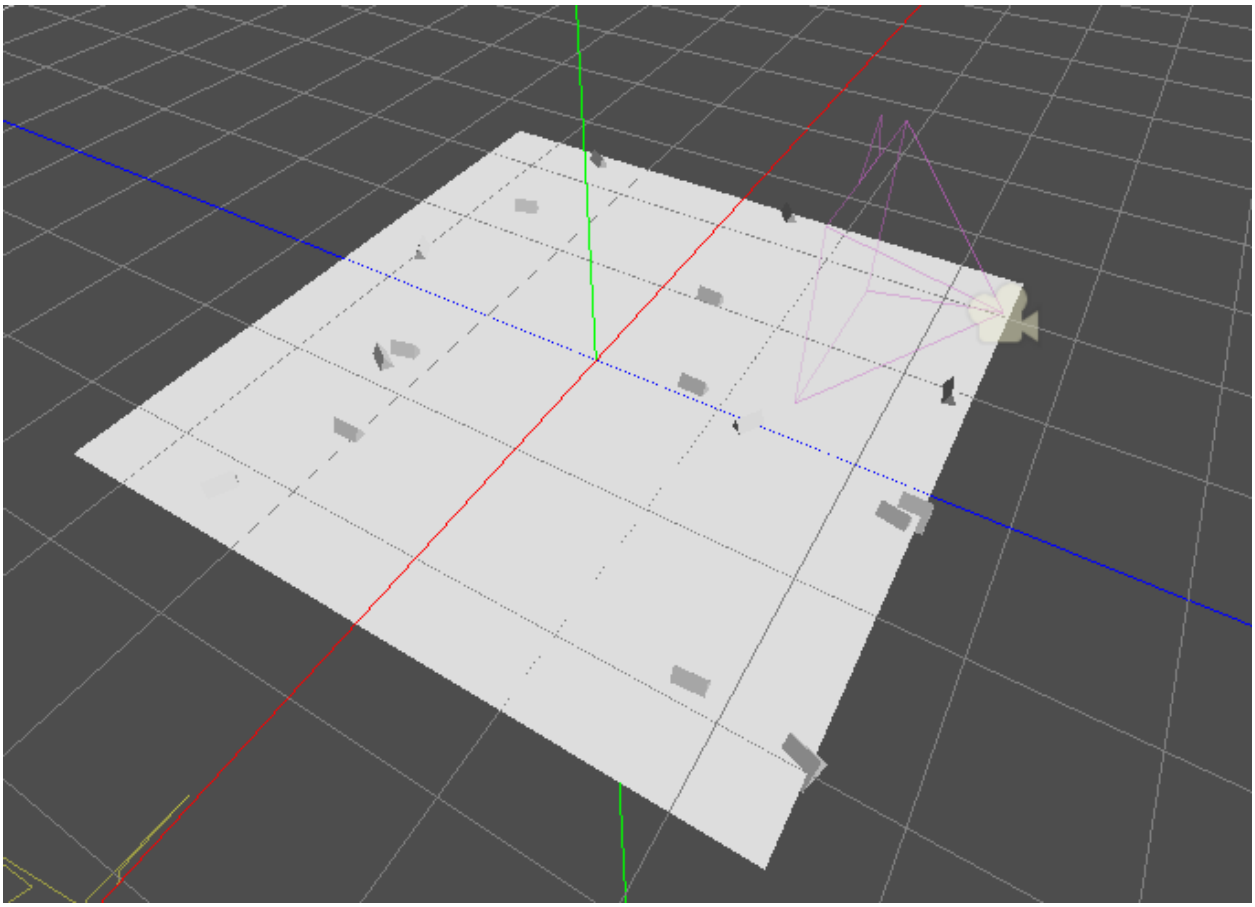
The scale of the source mesh that will be placed over the target surface.

Amount

The amount of mesh instances placed over the target surface.

Select the target surface, in the tree case, this should be the landscape node. And the source mesh should be the tree node. Adjust the other parameters according to your preference. Press `Populate` and multiple copies of the source mesh will be placed over the target mesh. If you are satisfied with the result, you can delete the mesh instance used as the source mesh.

The end result should look like this:



To change the result, repeat the same step with different parameters.

7.13 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.

7.13.1 Skeleton node

The Skeleton node can be directly added anywhere you want on a scene. Usually mesh is a child of Skeleton, as it easier to manipulate this way, as Transforms within a skeleton are relative to where the Skeleton is. But you can specify a Skeleton node in every MeshInstance.

Being obvious, Skeleton is intended to deform meshes, and consists of structures called "bones". Each "bone" is represented as a 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 the *MeshDataTool* class and its method *set_vertex_bones*.

The "bones" are organized hierarchically, every bone, except for root bone(s) have a parent. Every bone has an associated name you can use to refer to it (e.g. "root" or "hand.L", etc.). Also all bones are 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 an arm mesh with 2 bones - upperarm and lowerarm, with the lowerarm bone parented to the upperarm.

7.13.2 Skeleton class

You can view Godots internal help for descriptions of all functions. Basically all operations on bones are done using their numeric ID. You can convert from a name to a numeric ID and vice versa.

To find the number of bones in a skeleton we use the 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 the ID of a bone, use the 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 the ID, not just printing it. Also, we might need additional information - finding bone parents to complete chains, etc. This is done with the get/set_bone_* functions.

To find the parent of a bone we use the 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)
```

The bone transforms are the things of our interest here. There are 3 kind of transforms - local, global, custom.

To find the local Transform of a bone we use get_bone_pose(id) function:

```
extends Spatial
var skel

func _ready():
```

(continues on next page)

(continued from previous page)

```
    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 get a 3x4 matrix there, with the first column filled with 1s. What can we do with this matrix? It is a Transform, so we can do everything we can do with Transforms, basically translate, rotate and scale. We could also multiply transforms to have more complex transforms. Remember, "bones" in Godot are just Transforms over a group of vertices. We could also 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(delta):
    var t = skel.get_bone_pose(id)
    t = t.rotated(Vector3(0.0, 1.0, 0.0), 0.1 * delta)
    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 check the results.

What we used here was the local pose. By default all bones are not modified. But this Transform tells us nothing about the relationship between bones. This information is needed for quite a number of tasks. How can we get it? Here the global transform comes into play:

To find the bone global Transform we use get_bone_global_pose(id) function:

Let's find the global Transform for the 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 can see, this transform is not zeroed. While being called global, it is actually relative to the Skeleton origin. For a root bone, origin is always at 0 if not modified. Lets print the 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 the 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 the bone tip? We can't know things like the bone length, which we need for many things, without knowing the tip location. For all bones in a chain except for the last one we can calculate the tip location - it is simply a child bone's 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. A 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 an extra bone to the chain or just calculating the length of the leaf bone in Blender and storing the value in your script.

7.13.3 Using 3D "bones" for mesh control

Now as you know the 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 the Camera so that the arm is properly visible. Rotate DirectionLight so that the arm is properly lit while in scene play mode.

Now we need to create a new script under main:

First we define the setup parameters:

```
var lowerarm_angle = Vector3()
var 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 -> Input Map:

- **selext_x** - bind to X key
- **selext_y** - bind to Y key
- **selext_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(delta):
    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(delta):
    if Input.is_action_pressed("select_x"):
        coordinate = 0
    elif Input.is_action_pressed("select_y"):
```

(continues on next page)

(continued from previous page)

```
        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 selects upperarm/lowerarm, select the 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 the "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 the web-interface:

https://github.com/slapin/godot-skel3d

7.13.4 Using 3D "bones" to implement Inverse Kinematics

See *Inverse kinematics*.

7.13.5 Using 3D "bones" to implement ragdoll-like physics

TODO.

7.14 Inverse kinematics

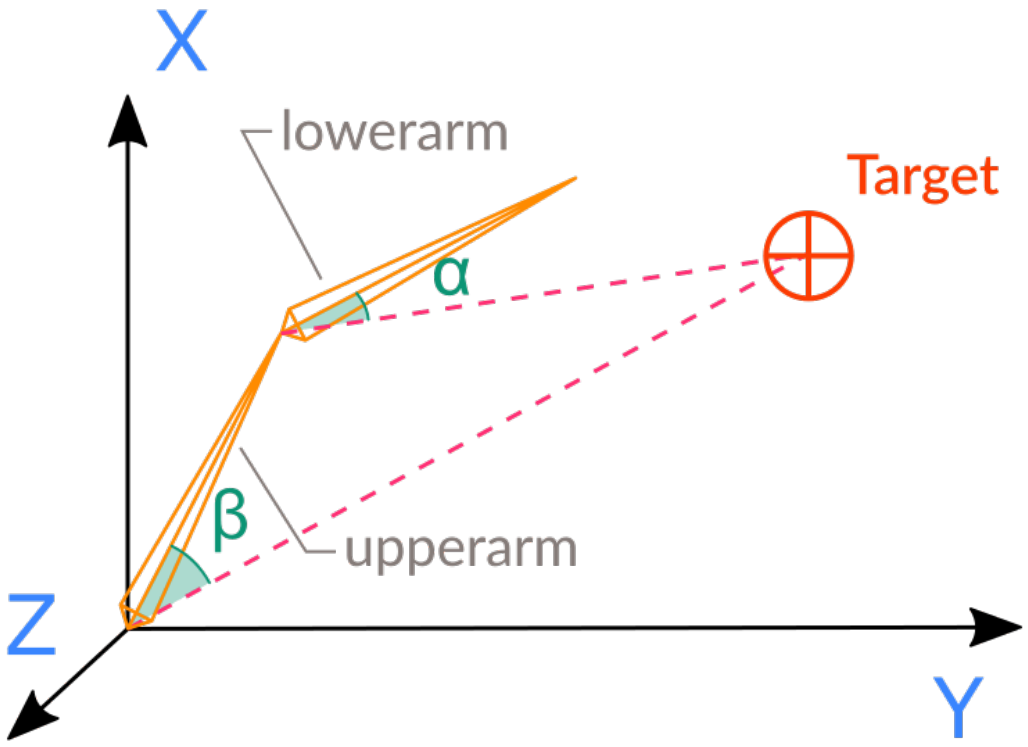
This tutorial is a follow-up of *Working with 3D skeletons*.

Before continuing on, I'd recommend reading some theory, the simplest article I could find is this:

http://freespace.virgin.net/hugo.elias/models/m_ik2.htm

7.14.1 Initial problem

Talking in Godot terminology, the task we want to solve here is to position our 2 angles we talked about above so, that the tip of the lowerarm bone is as close to the target point (which is set by the target Vector3()) as possible using only rotations. This task is calculation-intensive and never resolved by analytical equation solving. So, it is an underconstrained problem, which means there is an unlimited number of solutions to the equation.



For easy calculation, in this chapter we consider the target being a child of Skeleton. If this is not the case for your setup you can always reparent it in your script, as you will save on calculations if you do so.

In the picture you see the 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 a plane which is defined by bone base, bone tip and target.

The rotation axis is easily calculated using the cross-product of the bone vector and the target vector. The rotation in this case will be always in positive direction. If `t` is the Transform which we get from the get_bone_global_pose() function, the bone vector is

```
t.basis[2]
```

So we have all the information we need 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 the vector from the bone base to the 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 the 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 at the tip of the IK bone and calculate using that.

7.14.2 Implementation

We will use an exported variable for the bone length to make it 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 the IK bone to the base of the chain. So we apply a rotation to the IK bone then move from our IK bone up to its parent, apply rotation again, then move to the parent of the current bone again, etc. So we need to limit our chain somewhat.

```
export var ik_limit = 2
```

For the `_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(delta):
    pass_chain(delta)
```

Executing this script will pass through the 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)
```

(continues on next page)

(continued from previous page)

```
func pass_chain(delta):
    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(delta):
    pass_chain(delta)
```

Now we need to actually work with the target. The target should be placed somewhere accessible. Since "arm" is an imported scene, we better place the target node within our top level scene. But for us to work with target easily its Transform should be on the same level as the Skeleton.

To cope with this problem we create a "target" node under our scene root node and at runtime we will reparent it copying the global transform, which will achieve the desired effect.

Create a new Spatial node under the root node and rename it to "target". Then modify the `_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)
```

7.15 Mesh generation with heightmap and shaders

7.15.1 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:

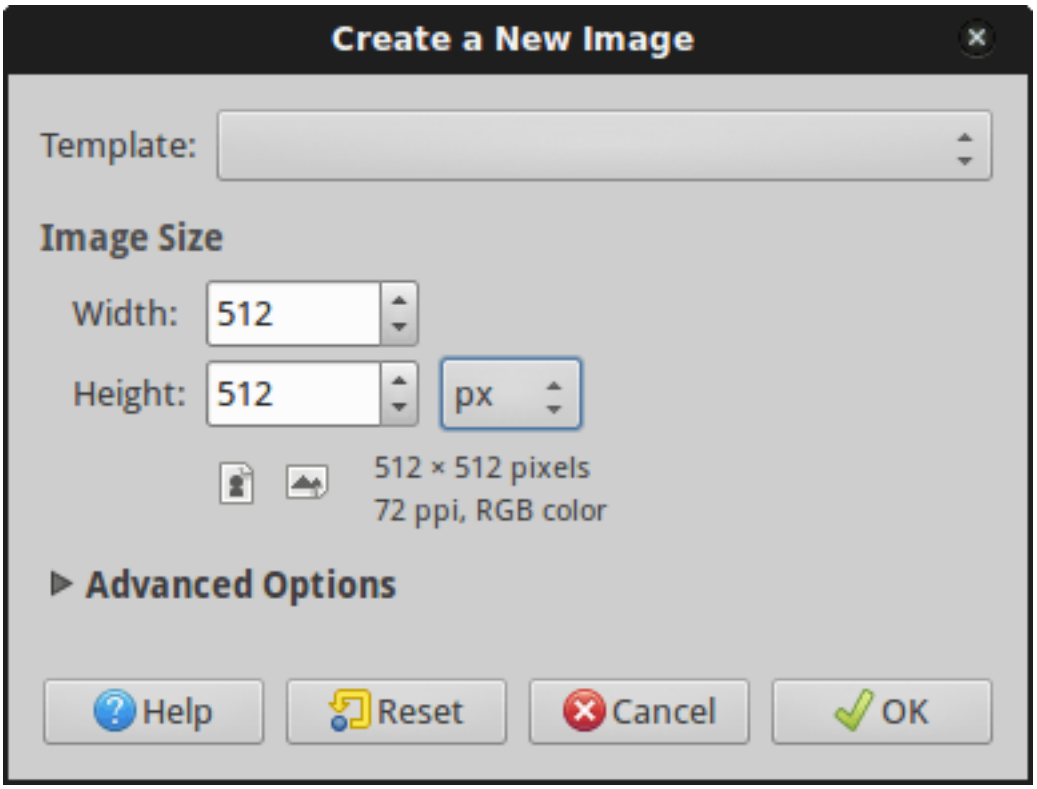
- The Vertex Shader can't re-compute normals of the faces. Thus, if your mesh is not static, this method will **not** work with shaded materials.
- This tutorial uses a plane mesh imported from Blender to Godot Engine. Godot is able to create meshes as well.

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.

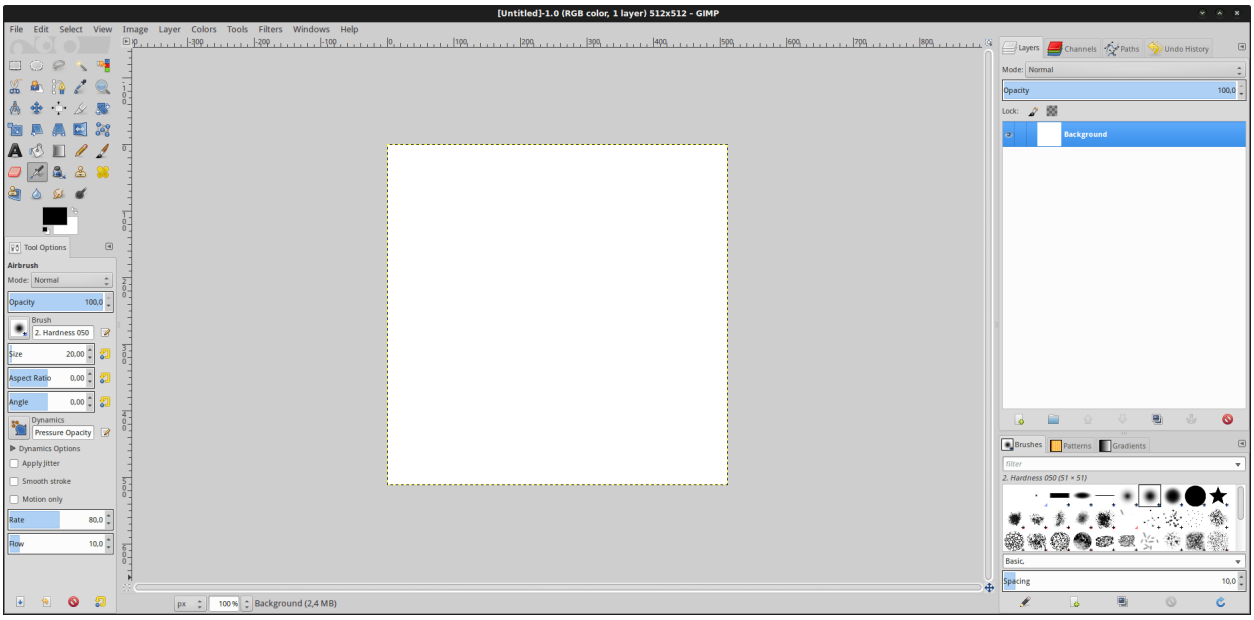
However, let's first create a heightmap,or a 2D representation of the terrain. To do this, I'll use GIMP, but you can use any image editor you like.

7.15.2 The heightmap

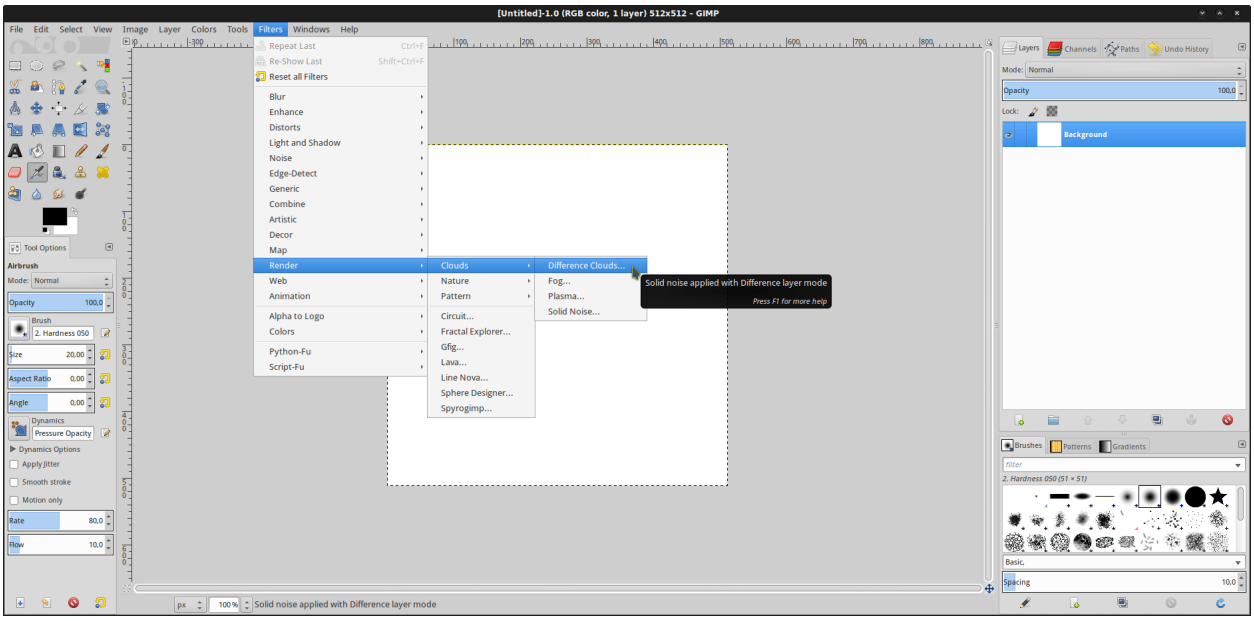
We will use a few functions of GIMP image editor to produce a simple heightmap. Start GIMP and create a square image of 512x512 pixels.



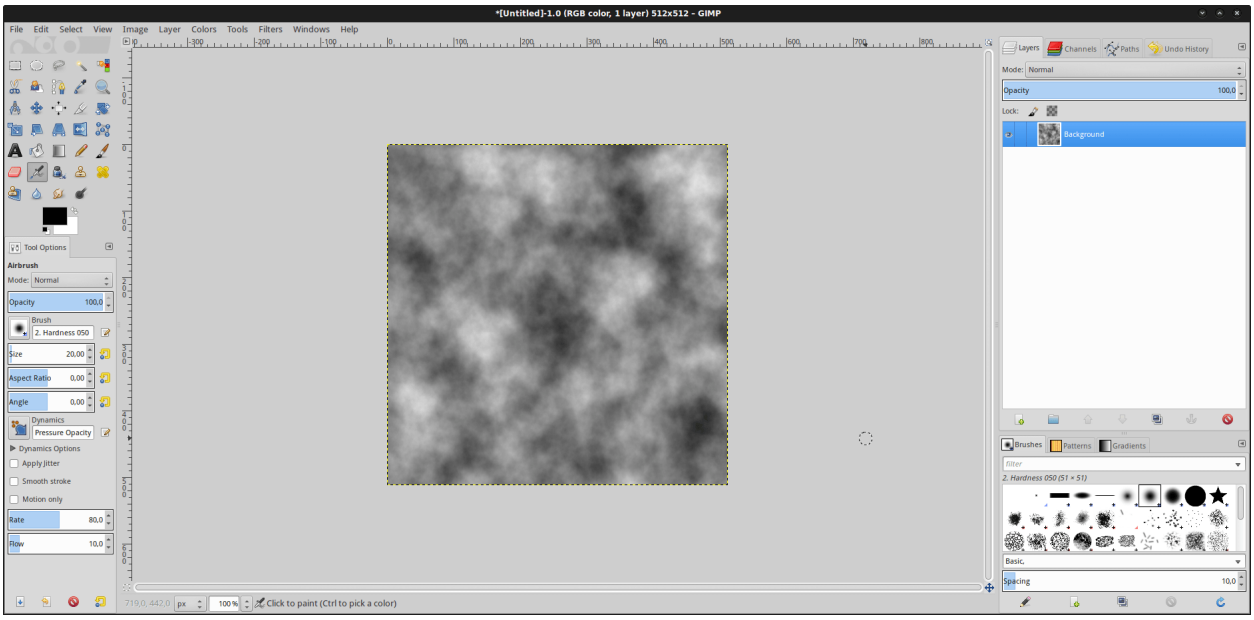
You are now in front of a new, blank, square image.



Then, use a filter to render some clouds on this new image.



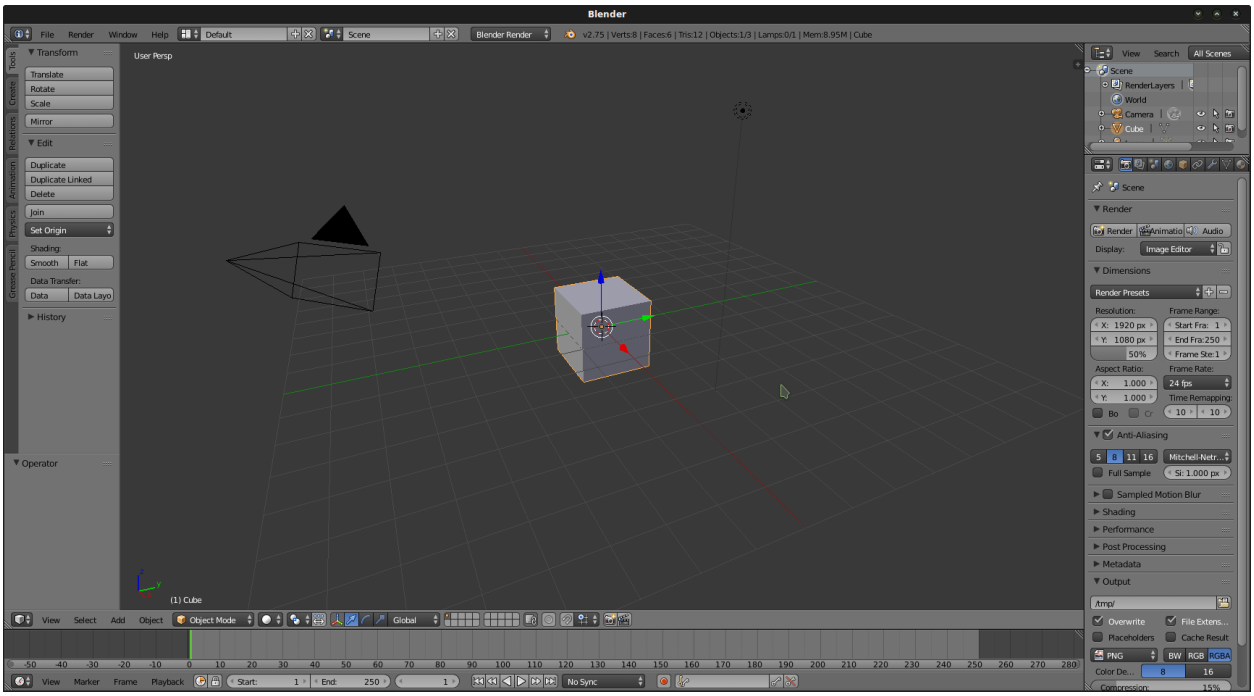
Parameter this filter to whatever you want. A white pixel corresponds to the highest point of the heightmap, a black pixel corresponds to the lowest one. So, darker regions are valleys and brighter are mountains. If you want, you can check “tileable” to render a heightmap that can be cloned and tiled 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”.



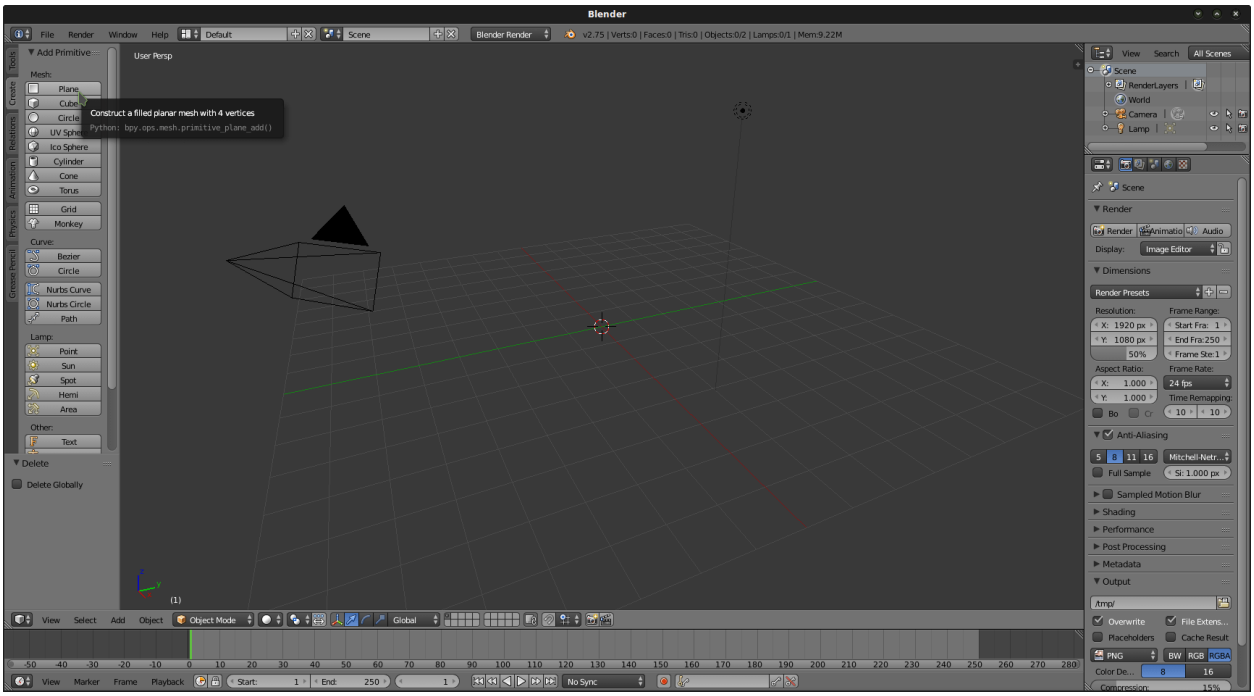
You can continue to edit your image if you wish. For our example, let's keep the heightmap as is, and let's export it to a PNG file, say "heightmap.png". Save it in your Godot project folder.

7.15.3 The plane mesh

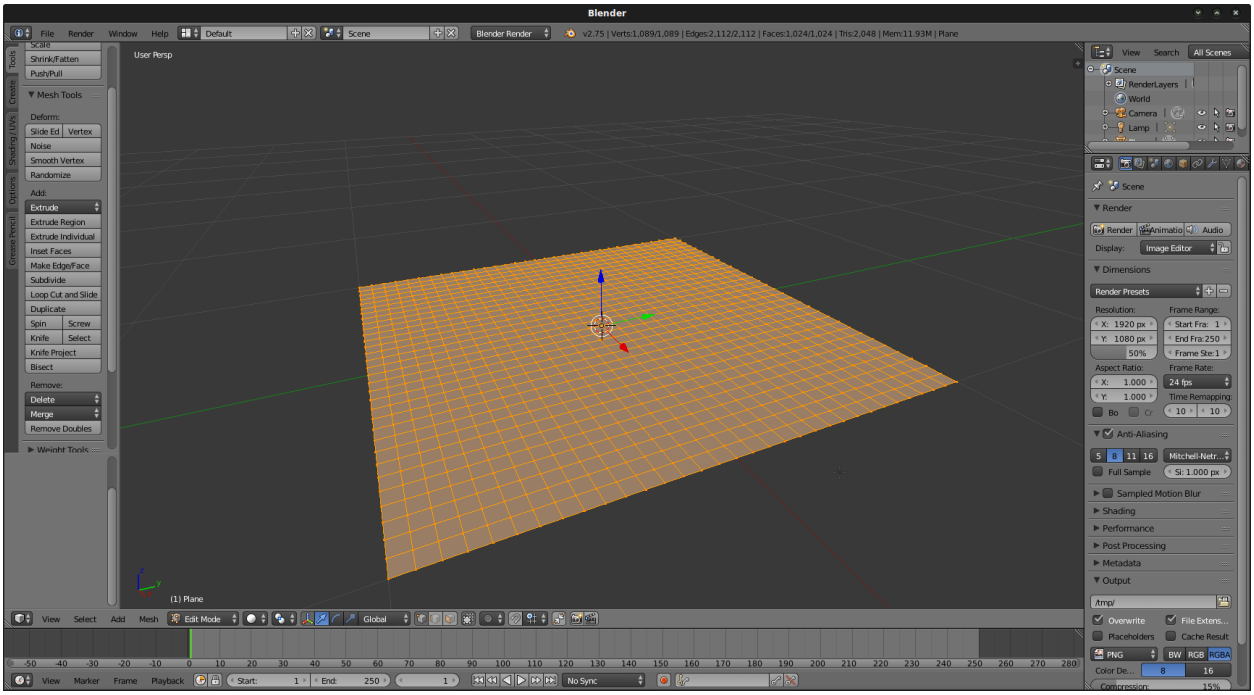
Now, we will need a plane mesh to import in Godot. Let's run Blender.



Remove the start cube mesh, then add a new plane to the scene.

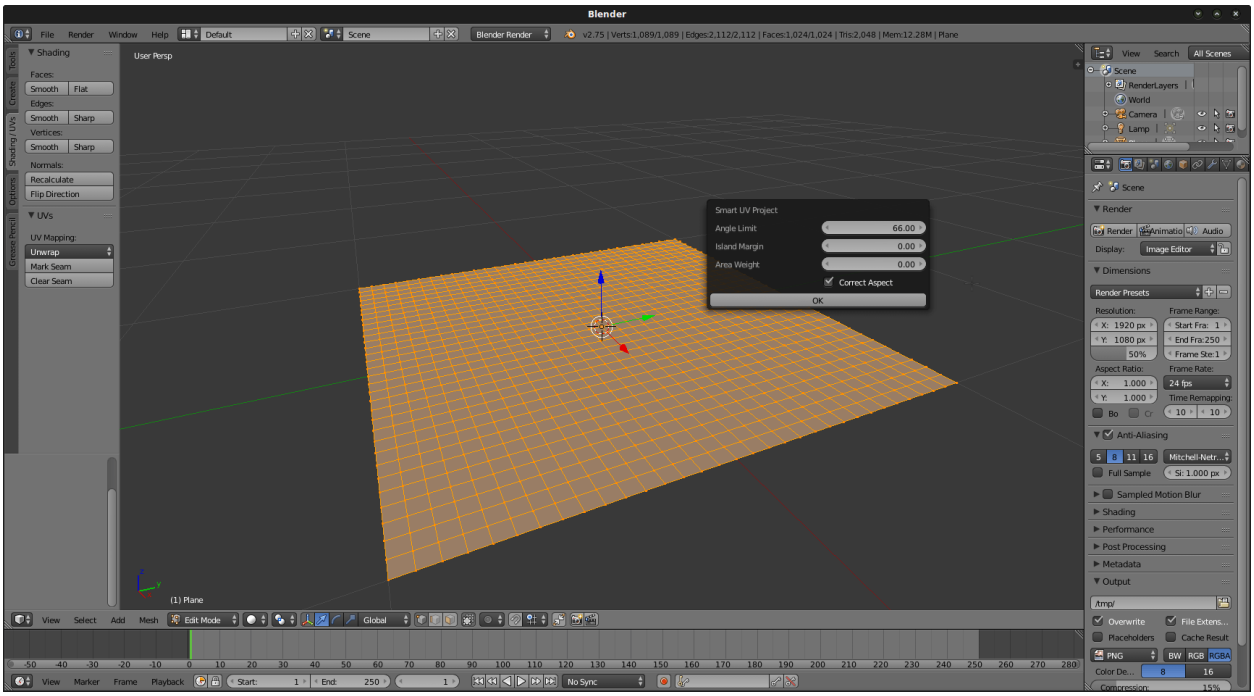


Zoom a bit, then switch to Edit mode (Tab key) and in the Tools buttongroup at the left, hit "Subdivide" 5 or 6 times.

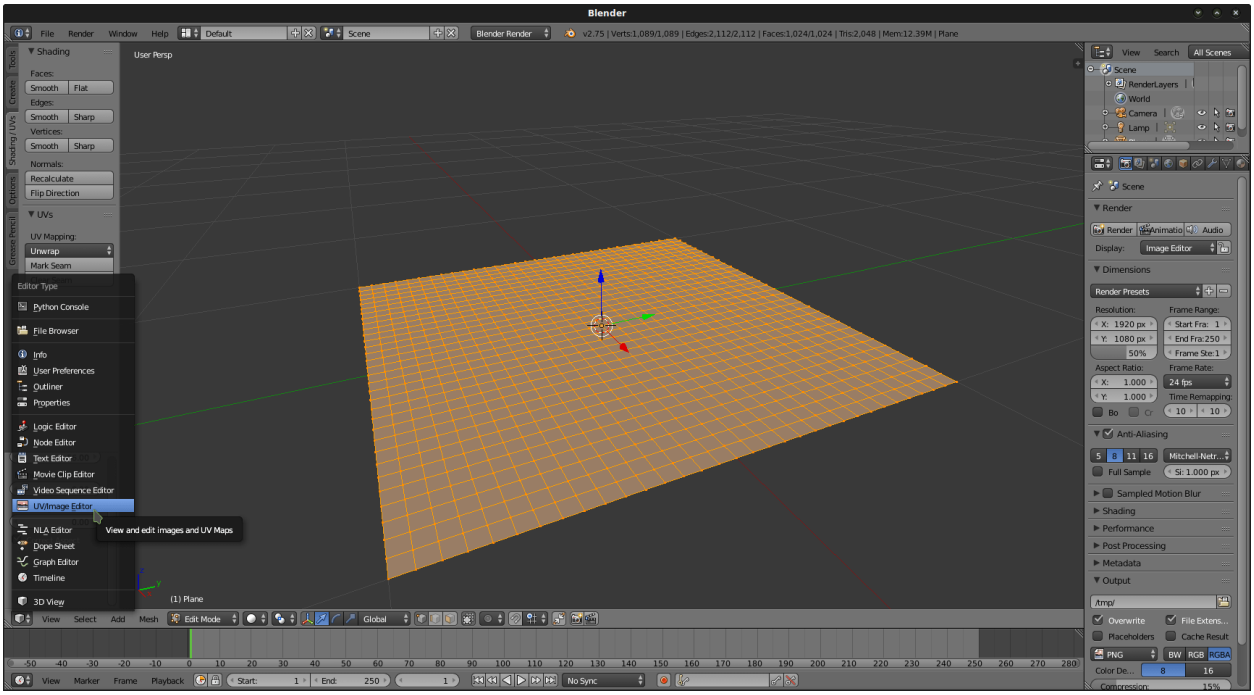


Your mesh is now subdivided, which means we added vertices to the plane mesh that we will later be able to move. Job's not finished yet: in order to texture this mesh a proper UV map is necessary. Currently, the default UV map contains only the 4 corner vertices we had at the beginning. However, we now have more, and we want to be able to texture over the whole mesh correctly.

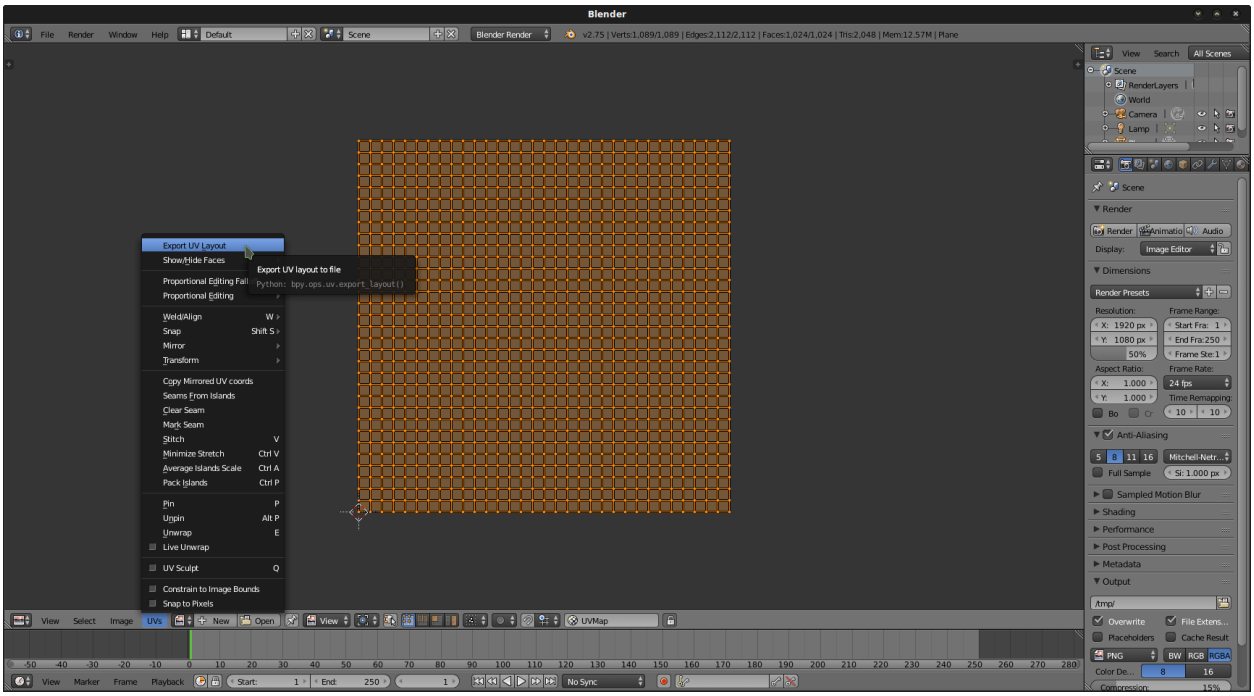
If all the vertices of your mesh are not selected, select them all (hit "A"). They must appear orange, not black. Then, in the Shading/UVs button group to the left, click the "Unwrap" button (or simply hit "U") and select "Smart UV Project". Keep the default options and hit "Ok".



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”.

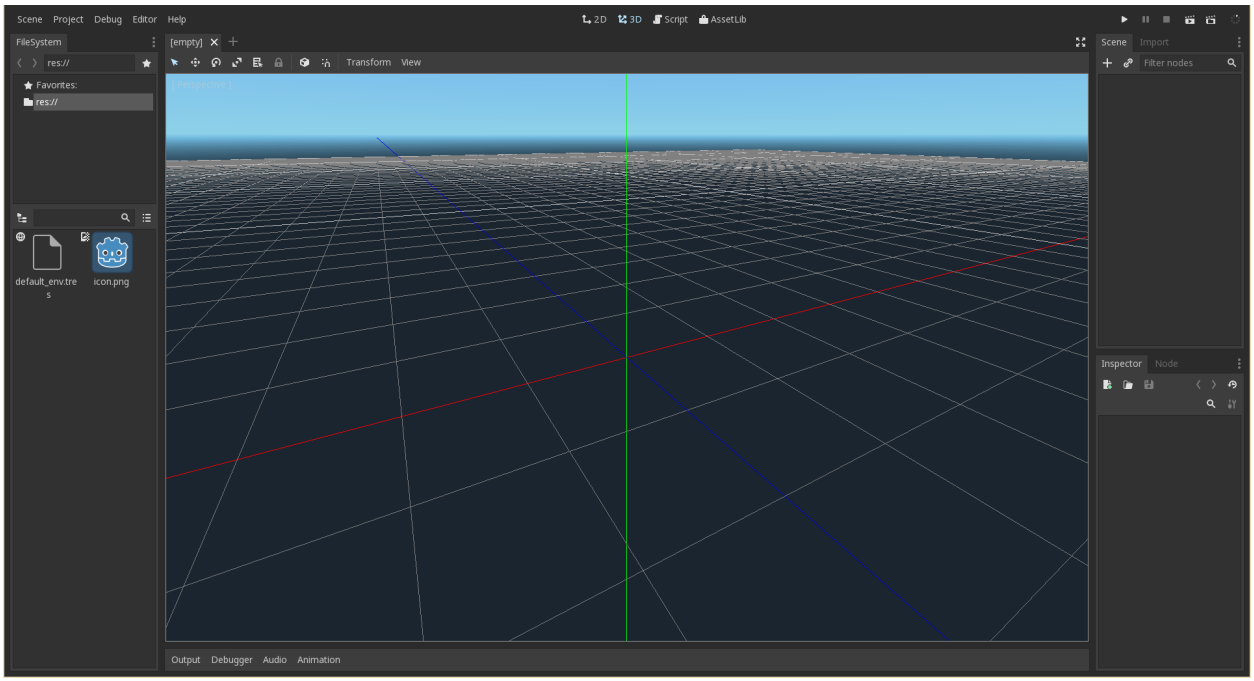


Export the layout as a PNG file. Name it "plane.png" and save it in your Godot project folder. Now, let's export our mesh as an OBJ file. Top of the screen, click "File/Export/Wavefront (obj)". Save your object as "plane.obj" in your Godot project folder.

7.15.4 Shader magic

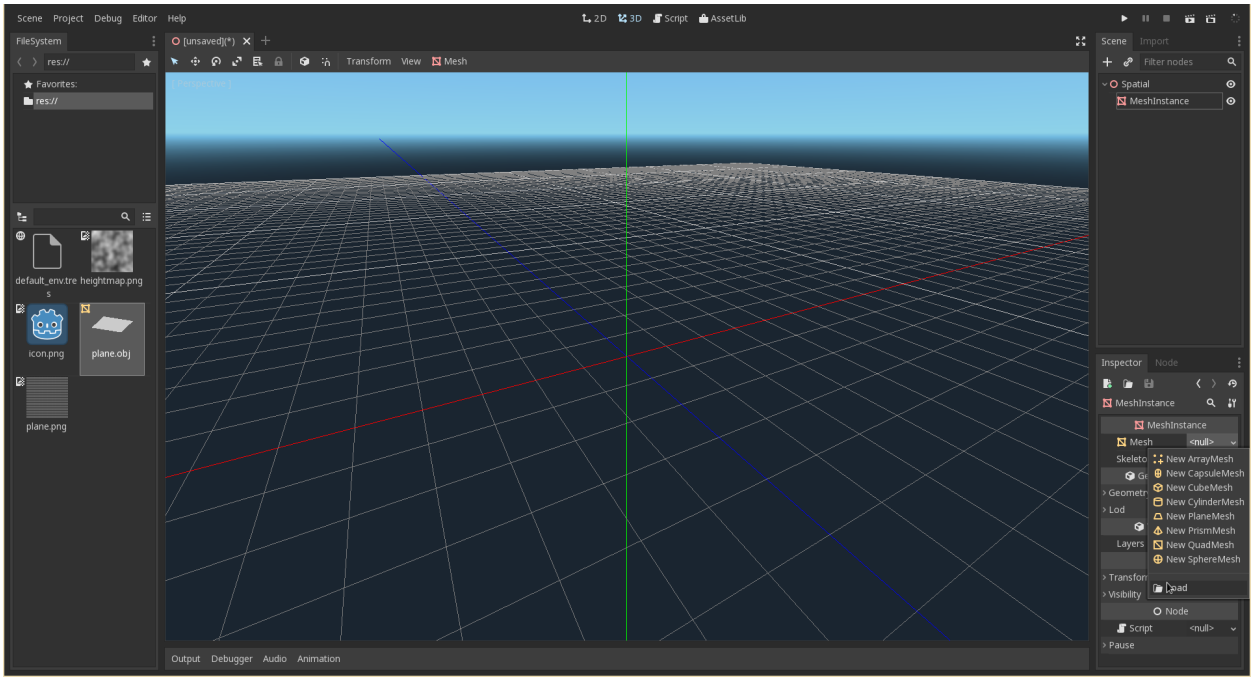
Let's now open Godot Editor.

Create a new project in the folder you previously created and name it what you want.

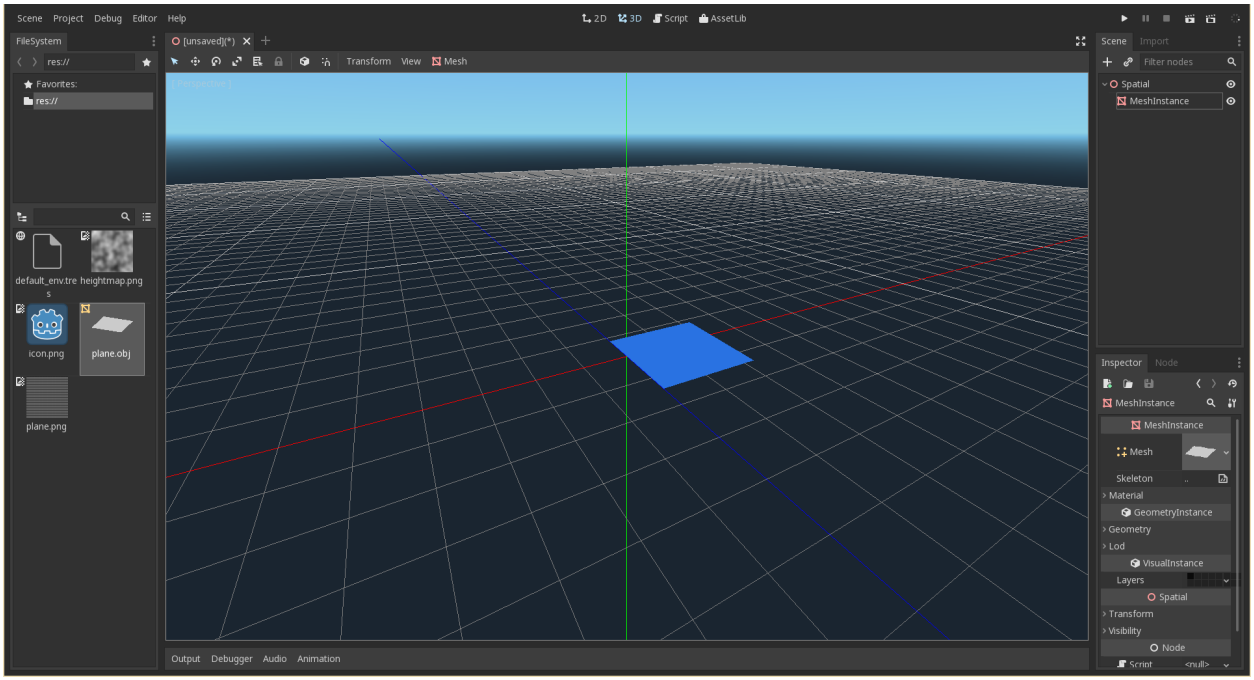


In our default scene (3D), create a root node “Spatial”.

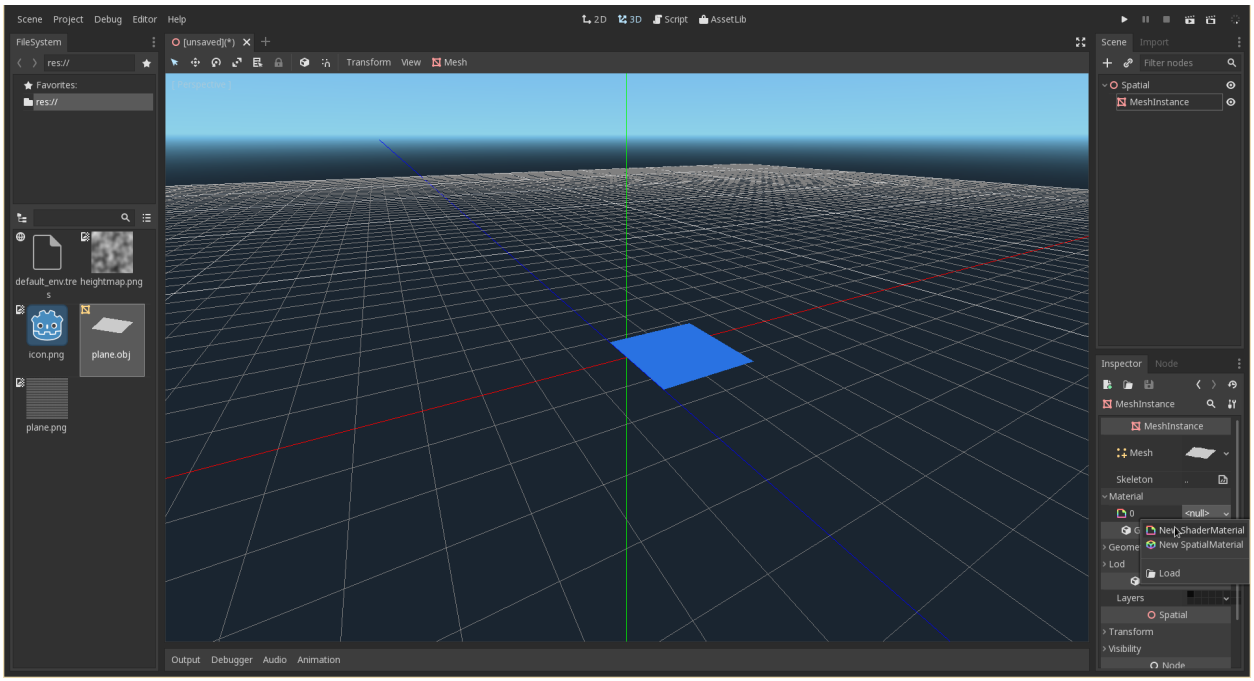
Create a MeshInstance node as a child of the node we just created. Then, load the Mesh selecting “Load” and then our “plane.obj” file.



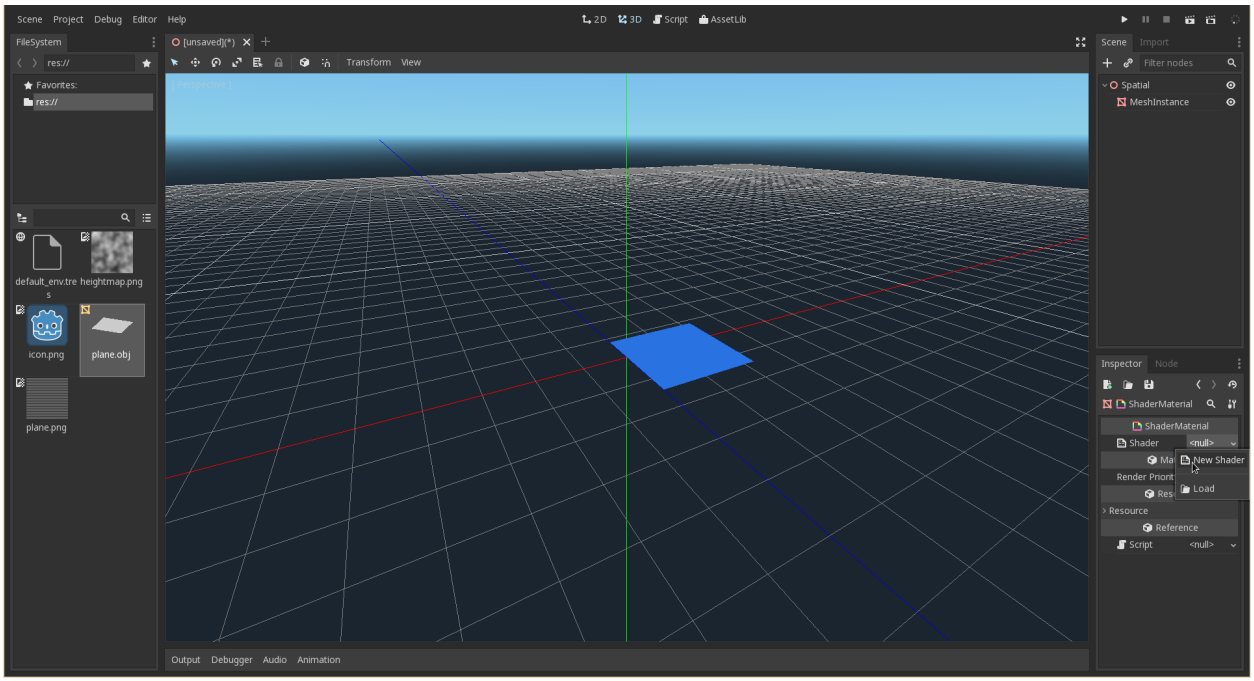
Great! Our plane is now rendered in the 3D view.



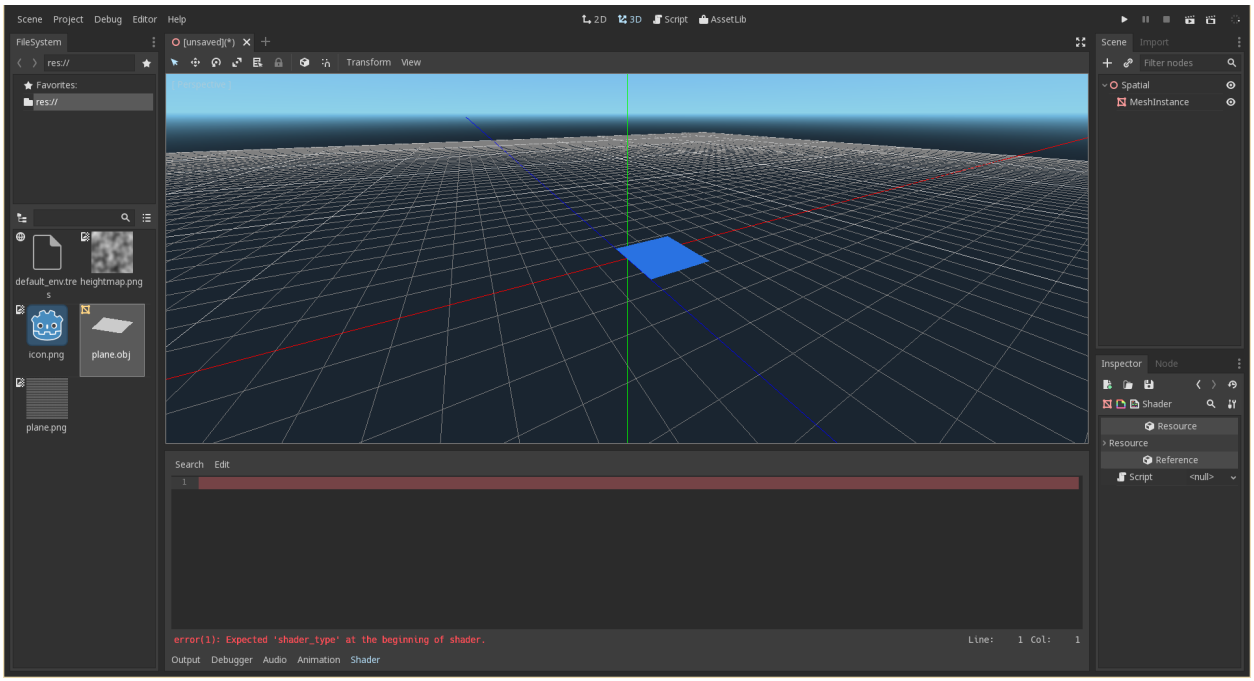
It is time to add some shader stuff. In the Inspector, in the “Material” line, add a “New ShaderMaterial”. Edit it by clicking it and then selecting the “Edit” option.



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.

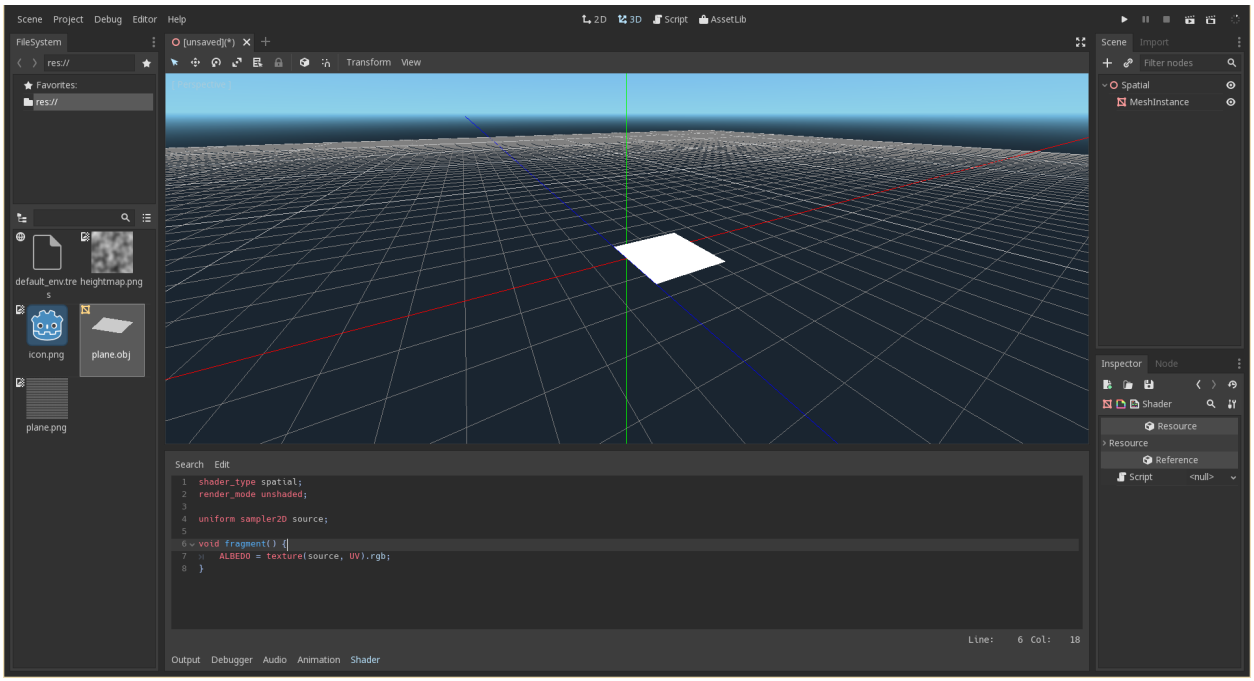
Let's start with the Fragment part. This one is used to texture the plane using an image. For this example, we will texture it with the heightmap image itself, so we'll actually see mountains as brighter regions and canyons as darker regions. Use this code:

```
shader_type spatial;
render_mode unshaded;

uniform sampler2D source;

void fragment() {
    ALBEDO = texture(source, UV).rgb;
}
```

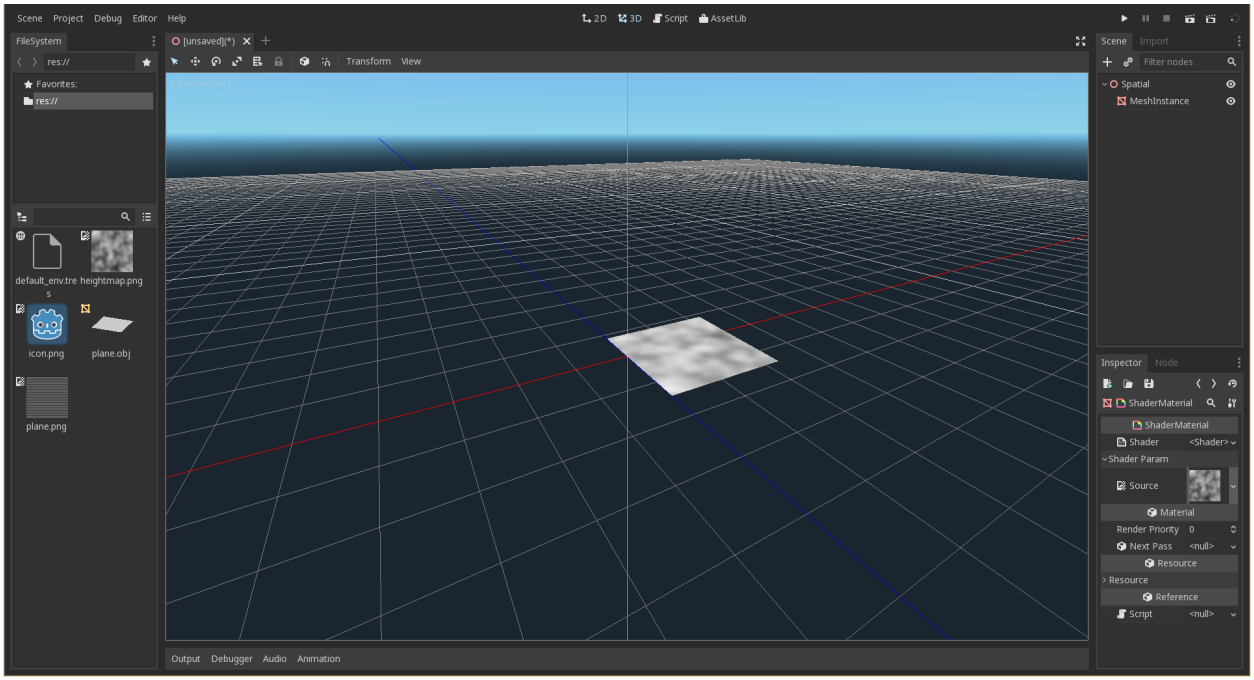
First, we set the shader type as `spatial` (for 3D). The `render_mode unshaded` line makes our MeshInstance be unaffected by the lighting in our world. It doesn't matter since it is a greyscale image. We take a parameter (`uniform`) as a `sampler2D`, which will be the texture of our heightmap.

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.

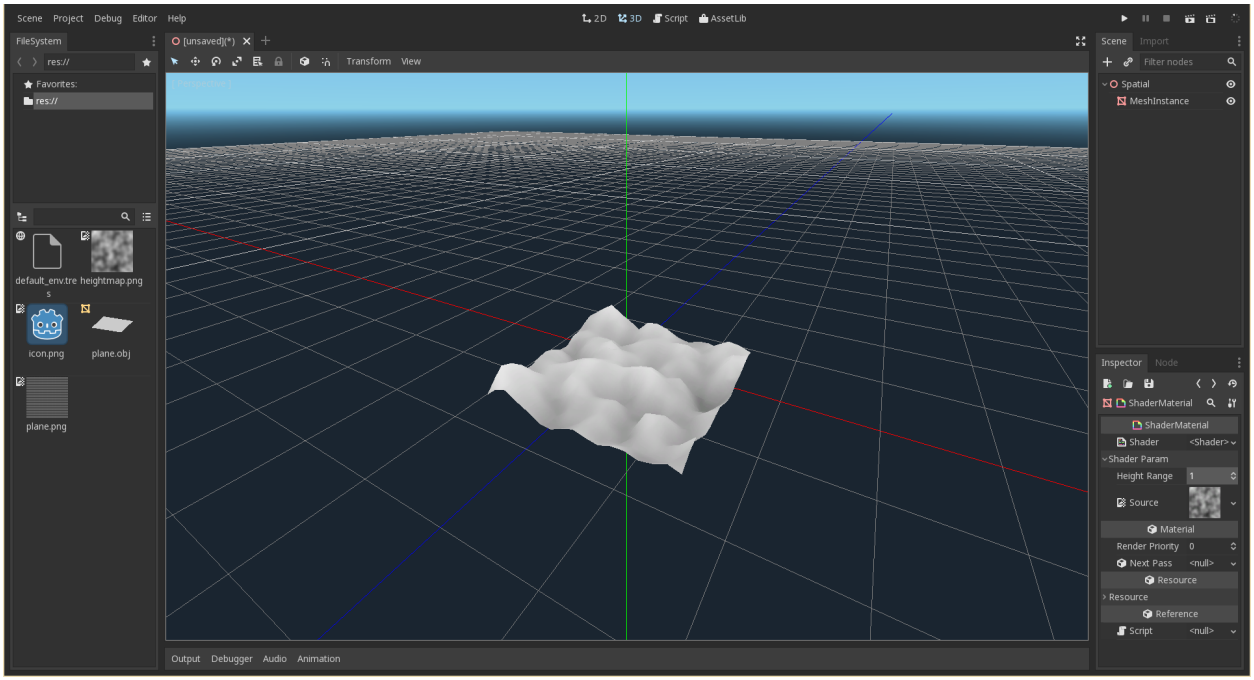
```
void vertex() {
    vec2 xz = VERTEX.xz;
    float h = texture(source, UV).g * height_range;
    VERTEX = vec3(xz.x, h, xz.y);
}
```

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.

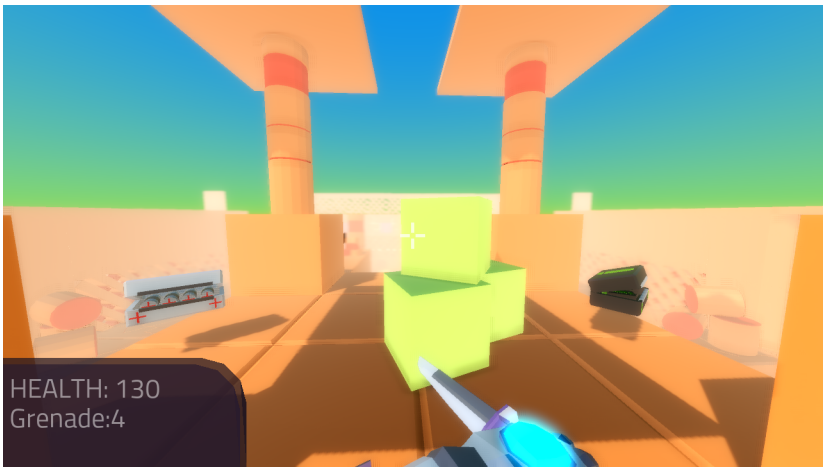
That's all good, but our plane remains flat. This is because the `height_range` value is 0. Increase this value to observe the mesh distort and take to form of the terrain we set before:



7.16 FPS tutorial

7.16.1 Part 1

Tutorial introduction



This tutorial series will show you how to make a single player FPS game.

Throughout the course of this tutorial series, we will cover how:

- To make a first person character that can move, sprint, and jump.
- To make a simple animation state machine for handling animation transitions.
- To add three weapons to the first person character, each using a different way to handle bullet collisions:
  - A knife (using an *Area*)
  - A pistol (Bullet scenes)
  - A rifle (using a *Raycast*)
- To add two different types of grenades to the first person character:
  - A normal grenade
  - A sticky grenade
- To add the ability to grab and throw *RigidBody* nodes
- To add joypad input for the player
- To add ammo and reloading for all weapons that consume ammo.
- To add ammo and health pick ups
  - In two sizes: big and small
- To add an automatic turret
  - That can fire using bullet objects or a *Raycast*
- To add targets that break when they've taken enough damage
- To add sounds that play when the guns fire.
- To add a simple main menu:

- – With an options menu for changing how the game runs
- – With a level select screen
- To add a universal pause menu we can access anywhere

Note: While this tutorial can be completed by beginners, it is highly advised to complete *Your First Game*, if you are new to Godot and/or game development **before** going through this tutorial series.

Remember: Making 3D games is much harder than making 2D games. If you do not know how to make 2D games you will likely struggle making 3D games.

This tutorial assumes you know have experience working with the Godot editor, have basic programming experience in GDScript, and have basic experience in game development.

You can find the start assets for this tutorial here: `Godot_FPS_Starter.zip`

The provided starter assets contain an animated 3D model, a bunch of 3D models for making levels, and a few scenes already configured for this tutorial.

All assets provided (unless otherwise noted) were originally created by TwistedTwigleg, with changes/additions by the Godot community. All original assets provided for this tutorial are released under the `MIT` license.

Feel free to use these assets however you want! All original assets belong to the Godot community, with the other assets belonging to those listed below:

Note: The skybox is created by **StumpyStrust** on OpenGameArt. The skybox used is licensed under `CC0`.

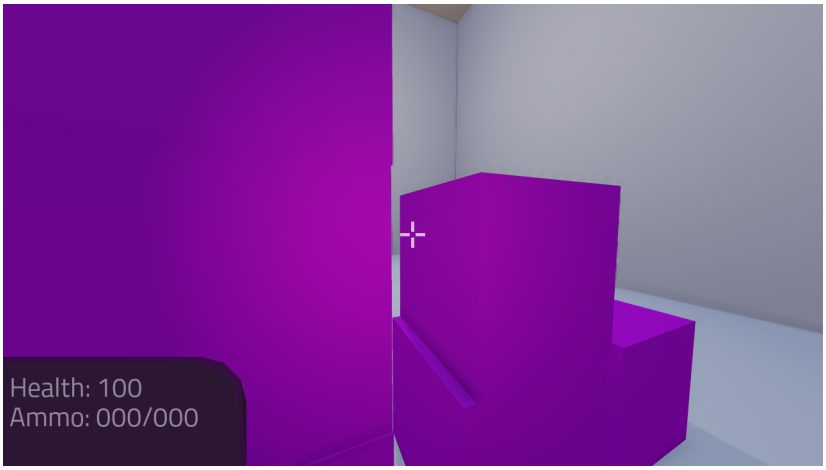
The font used is **Titillium-Regular**, and is licensed under the `SIL Open Font License, Version 1.1`.

Tip: You can find the finished project for each part at the bottom of each part's page

---

Part Overview

In this part we will be making a first person player that can move around the environment.



By the end of this part you will have a working first person character who can move around the game environment, look around with a mouse based first person camera, that can jump into the air, turn on and off a flash light, and sprint.

Getting everything ready

Launch Godot and open up the project included in the starter assets.

Note: While these assets are not necessarily required to use the scripts provided in this tutorial, they will make the tutorial much easier to follow as there are several pre-setup scenes we will be using throughout the tutorial series.

First, go open the project settings and go to the "Input Map" tab. You'll find several actions have already been defined. We will be using these actions for our player. Feel free to change the keys bound to these actions if you want.

Let's take a second to see what we have in the starter assets.

Included in the starter assets are several scenes. For example, in *res://* we have 14 scenes, most of which we will be visiting as we go through this tutorial series.

For now let's open up `Player.tscn`.

Note: There are a bunch of scenes and a few textures in the `Assets` folder. You can look at these if you want, but we will not be exploring through `Assets` in this tutorial series. `Assets` contains all of the models used for each of the levels, as well as some textures and materials.

Making the FPS movement logic

Once you have `Player.tscn` open, let's take a quick look at how it is set up

First, notice how the player's collision shapes are set up. Using a vertical pointing capsule as the collision shape for the player is fairly common in most first person games.

We are adding a small square to the 'feet' of the player so the player does not feel like they are balancing on a single point.

We do want the 'feet' slightly higher than the bottom of the capsule so we can roll over slight edges. Where to place the 'feet' is dependent on your levels and how you want your player to feel.

Note: Many times player will notice how the collision shape being circular when they walk to an edge and slide off. We are adding the small square at the bottom of the capsule to reduce sliding on, and around, edges.

Another thing to notice is how many nodes are children of `Rotation_Helper`. This is because `Rotation_Helper` contains all of the nodes we want to rotate on the `X` axis (up and down). The reason behind this is so we can rotate `Player` on the `Y` axis, and `Rotation_helper` on the `X` axis.

Note: If we did not use `Rotation_helper` then we'd likely have cases where we are rotating both the `X` and `Y` axes at the same time. This can lead to undesirable results, as we then could rotate on all three axes in some cases.

See *using transforms* for more information

Attach a new script to the `Player` node and call it `Player.gd`.

Let's program our player by adding the ability to move around, look around with the mouse, and jump. Add the following code to `Player.gd`:

```
extends KinematicBody

const GRAVITY = -24.8
var vel = Vector3()
const MAX_SPEED = 20
const JUMP_SPEED = 18
const ACCEL= 4.5

var dir = Vector3()

const DEACCEL= 16
const MAX_SLOPE_ANGLE = 40

var camera
var rotation_helper

var MOUSE_SENSITIVITY = 0.05

func _ready():
    camera = $Rotation_Helper/Camera
    rotation_helper = $Rotation_Helper

    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)

func _physics_process(delta):
    process_input(delta)
    process_movement(delta)

func process_input(delta):

    # ----------------------------------
    # Walking
    dir = Vector3()
    var cam_xform = camera.get_global_transform()

    var input_movement_vector = Vector2()

    if Input.is_action_pressed("movement_forward"):
        input_movement_vector.y += 1
    if Input.is_action_pressed("movement_backward"):
        input_movement_vector.y -= 1
    if Input.is_action_pressed("movement_left"):
        input_movement_vector.x -= 1
    if Input.is_action_pressed("movement_right"):
        input_movement_vector.x += 1
```

(continues on next page)

(continued from previous page)

```
    input_movement_vector = input_movement_vector.normalized()

    dir += -cam_xform.basis.z.normalized() * input_movement_vector.y
    dir += cam_xform.basis.x.normalized() * input_movement_vector.x
    # ----------------------------------

    # ----------------------------------
    # Jumping
    if is_on_floor():
        if Input.is_action_just_pressed("movement_jump"):
            vel.y = JUMP_SPEED
    # ----------------------------------

    # ----------------------------------
    # Capturing/Freeing the cursor
    if Input.is_action_just_pressed("ui_cancel"):
        if Input.get_mouse_mode() == Input.MOUSE_MODE_VISIBLE:
            Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)
        else:
            Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
    # ----------------------------------

func process_movement(delta):
    dir.y = 0
    dir = dir.normalized()

    vel.y += delta*GRAVITY

    var hvel = vel
    hvel.y = 0

    var target = dir
    target *= MAX_SPEED

    var accel
    if dir.dot(hvel) > 0:
        accel = ACCEL
    else:
        accel = DEACCEL

    hvel = hvel.linear_interpolate(target, accel*delta)
    vel.x = hvel.x
    vel.z = hvel.z
    vel = move_and_slide(vel,Vector3(0,1,0), true, 0.05, 4, deg2rad(MAX_SLOPE_ANGLE))

func _input(event):
    if event is InputEventMouseMotion and Input.get_mouse_mode() == Input.MOUSE_MODE_
↪CAPTURED:
        rotation_helper.rotate_x(deg2rad(event.relative.y * MOUSE_SENSITIVITY))
        self.rotate_y(deg2rad(event.relative.x * MOUSE_SENSITIVITY * -1))

        var camera_rot = rotation_helper.rotation_degrees
        camera_rot.x = clamp(camera_rot.x, -70, 70)
        rotation_helper.rotation_degrees = camera_rot
```

This is a lot of code, so let's break it down function by function:

Tip: While copy and pasting code is ill advised, as you can learn a lot from manually typing the code in, you can copy and paste the code from this page directly into the script editor.

If you do this, all of the code copied will be using spaces instead of tabs.

To convert the spaces to tabs in the script editor, click the "edit" menu and select "Convert Indent To Tabs". This will convert all of the spaces into tabs. You can select "Convert Indent To Spaces" to convert t back into spaces.

First, we define some global variables to dictate how our player will move about the world.

Note: Throughout this tutorial, **variables defined outside functions will be referred to as "global variables"**. This is because we can access any of these variables from any place in the script. We can "globally" access them, hence the name.

Let's go through each of the global variables:

- `GRAV`: How strong gravity pulls us down.
- `vel`: Our *KinematicBody*'s velocity.
- `MAX_SPEED`: The fastest speed we can reach. Once we hit this speed, we will not go any faster.
- `JUMP_SPEED`: How high we can jump.
- `ACCEL`: How fast we accelerate. The higher the value, the faster we get to max speed.
- `DEACCEL`: How fast we are going to decelerate. The higher the value, the faster we will come to a complete stop.
- `MAX_SLOPE_ANGLE`: The steepest angle our *KinematicBody* will consider as a 'floor'.
- `camera`: The *Camera* node.
- `rotation_helper`: A *Spatial* node holding everything we want to rotate on the X axis (up and down).
- `MOUSE_SENSITIVITY`: How sensitive the mouse is. I find a value of `0.05` works well for my mouse, but you may need to change it based on how sensitive your mouse is.

You can tweak many of these variables to get different results. For example, by lowering `GRAVITY` and/or increasing `JUMP_SPEED` you can get a more 'floaty' feeling character. Feel free to experiment!

Note: You may have noticed that `MOUSE_SENSITIVITY` is written in all caps like the other constants, but is `MOUSE_SENSITIVITY` is not a constant.

The reason behind this is we want to treat it like a constant variable (a variable that cannot change) throughout our script, but we want to be able to change the value later when we add customizable settings. So, in an effort to remind ourselves to treat it like a constant, it's named in all caps.

Now let's look at the `_ready` function:

First we get the `camera` and `rotation_helper` nodes and store them into their variables.

Then we need to set the mouse mode to captured so the mouse cannot leave the game window.

This will hide the mouse and keep it at the center of the screen. We do this for two reasons: The first reason being we do not want to the player to see their mouse cursor as they play.

The second reason is because we do not want the cursor to leave the game window. If the cursor leaves the game window there could be instances where the player clicks outside the window, and then the game would lose focus. To assure neither of these issues happen, we capture the mouse cursor.

Note: see *Input documentation* for the various mouse modes. We will only be using `MOUSE_MODE_CAPTURED` and `MOUSE_MODE_VISIBLE` in this tutorial series.

Next let's take a look at `_physics_process`:

All we're doing in `_physics_process` is calling two functions: `process_input` and `process_movement`.

`process_input` will be where we store all of the code relating to player input. We want to call it first before anything else so we have fresh player input to work with.

`process_movement` is where we'll send all of the date necessary to the *KinematicBody* so it can move through the game world.

Let's look is `process_movement` next:

First we set `dir` to an empty *Vector3*.

`dir` will be used for storing the direction the player intends to move towards. Because we do not want the player's previous input to effect the player beyond a single `process_movement` call, we reset `dir`.

Next we get the camera's global transform and store it as well, into the `cam_xform` variable.

The reason we need the camera's global transform is so we can use it's directional vectors. Many have found directional vectors confusing, so let's take a second to explain how they work:

World space can be defined as: The space in which all objects are placed in, relative to a constant origin point. Every object, no matter if it is 2D or 3D, has a position in world space.

To put it another way: world space is the space in a universe where every object's position, rotation, and scale can be measured by a known, fixed point called the origin.

In Godot, the origin is at position `(0, 0, 0)` with a rotation of `(0, 0, 0)` and a scale of `(1, 1, 1)`.

Note: When you open up the Godot editor and select a *Spatial* based node, a gizmo pops up. Each of the arrows points using world space directions by default.

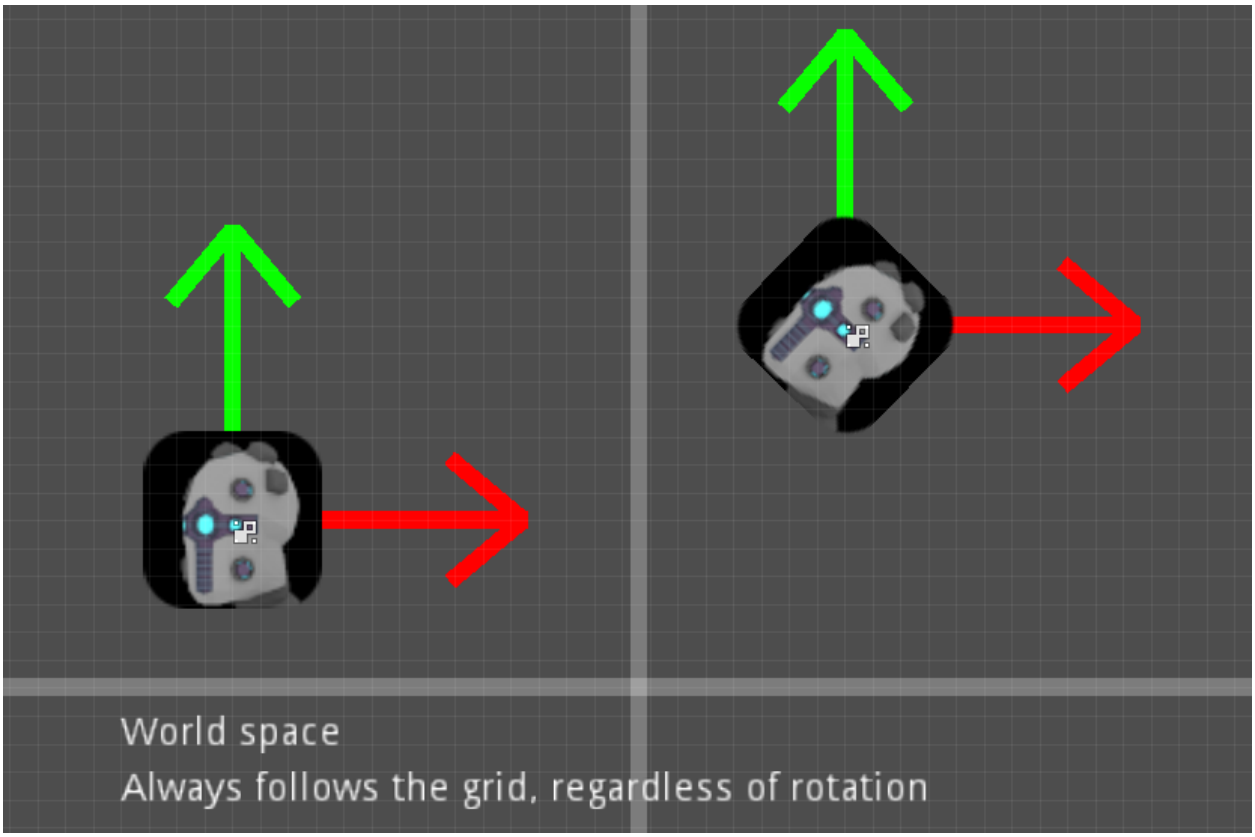
If you want to move using the world space directional vectors, you'd do something like this:

```
if Input.is_action_pressed("movement_forward"):
    node.translate(Vector3(0, 0, 1))
if Input.is_action_pressed("movement_backward"):
    node.translate(Vector3(0, 0, -1))
if Input.is_action_pressed("movement_left"):
    node.translate(Vector3(1, 0, 0))
if Input.is_action_pressed("movement_right"):
    node.translate(Vector3(-1, 0, 0))
```

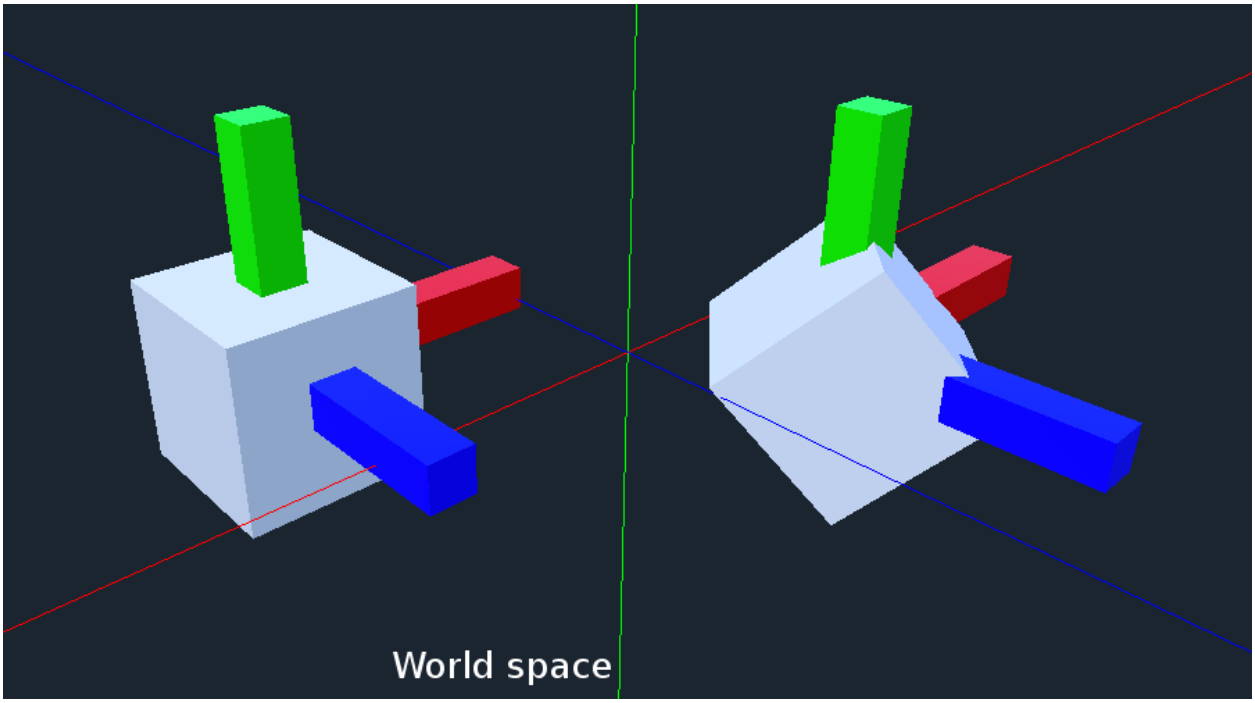
Note: Notice how we do not need to do any calculations to get world space directional vectors. We can define a few *Vector3* variables and input the values pointing in each direction.

Here is what world space looks like in 2D:

Note: The following images are just examples. Each arrow/rectangle represents a directional vector



And here is what it looks like for 3D:



Notice how in both examples, the rotation of the node does not change the directional arrows. This is because world space is a constant. No matter how you translate, rotate, or scale an object, world space will *always point in the same direction*.

Local space is different, because it takes the rotation of the object into account.

Local space can be defined as follows: The space in which a object's position is the origin of the universe. Because the position of the origin can be at `N` many locations, the values derived from local space change with the position of the origin.

Note: This stack overflow question has a much better explanation of world space and local space.

https://gamedev.stackexchange.com/questions/65783/what-are-world-space-and-eye-space-in-game-development (Local space and eye space are essentially the same thing in this context)

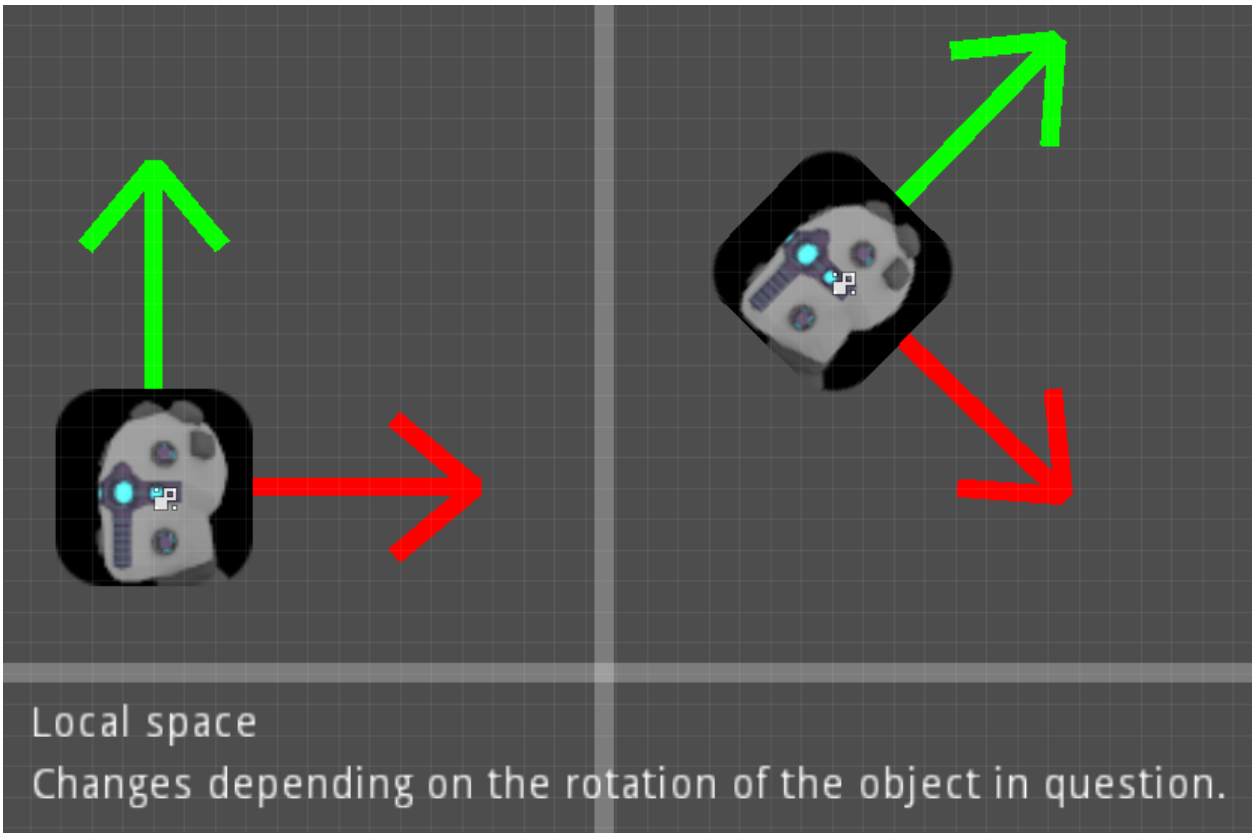
To get a *Spatial* node's local space, we need to get its *Transform*, so then we can get the *Basis* from the *Transform*.

Each *Basis* has three vectors: `X`, `Y`, and `Z`. Each of those vectors point towards each of the local space vectors coming from that object.

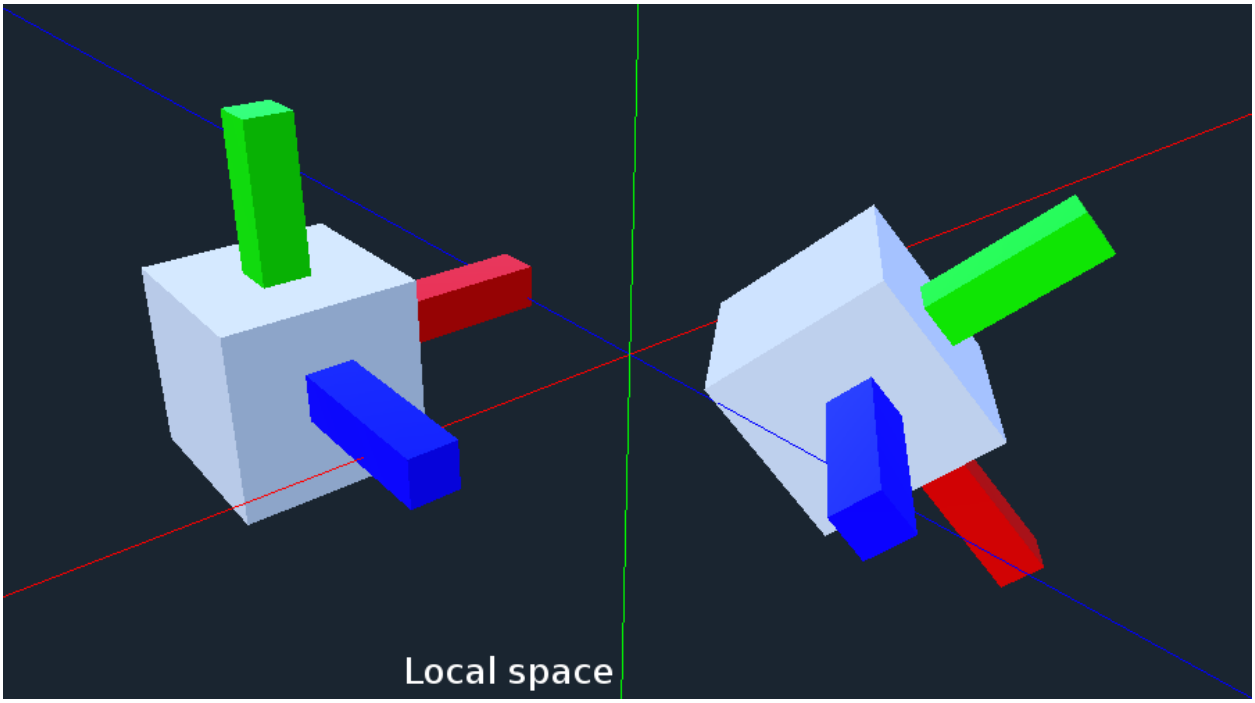
To use the a *Spatial* node's local directional vectors, we use this code:

```
if Input.is_action_pressed("movement_forward"):
    node.translate(node.global_transform.basis.z.normalized())
if Input.is_action_pressed("movement_backward"):
    node.translate(-node.global_transform.basis.z.normalized())
if Input.is_action_pressed("movement_left"):
    node.translate(node.global_transform.basis.x.normalized())
if Input.is_action_pressed("movement_right"):
    node.translate(-node.global_transform.basis.x.normalized())
```

Here is what local space looks like in 2D:

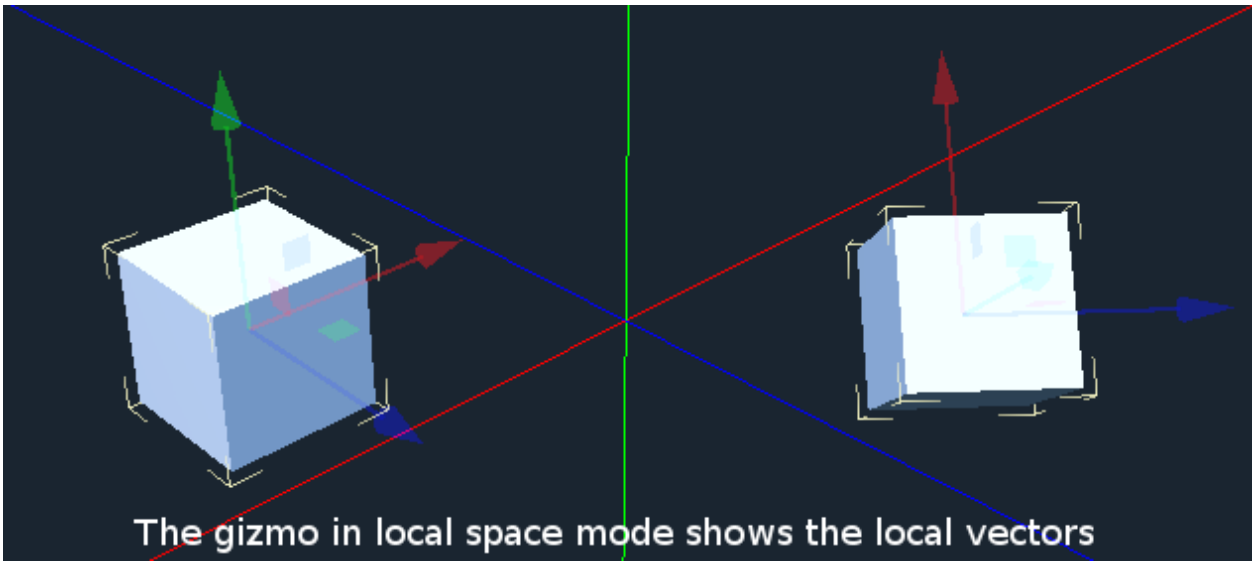


And here is what it looks like for 3D:



Here is what the *Spatial* gizmo shows when you are using local space mode. Notice how the arrows follow the rotation of the object on the left, which looks exactly the same as the 3D example for local space.

Note: You can change between local and world space modes by pressing the little cube button when you have a *Spatial* based node selected.



Local vectors are confusing even for more experienced game developers, so do not worry if this all doesn't make a lot of sense. The key thing to remember about local vectors is that we are using local coordinates to get direction from the object's point of view, as opposed to using world vectors which give direction from the world's point of view.

Okay, back to `process_input`:

Next we make a new variable called `input_movement_vector` and assign it to an empty *Vector2*. We will use this to make a virtual axis of sorts so map the player's input to movement.

Note: This may seem overkill for just the keyboard, but this will make sense later when we add joypad input.

Based on which directional movement action is pressed, we add or remove from `input_movement_vector`.

After we've checked each of the directional movement actions, we normalize `input_movement_vector`. This makes it where `input_movement_vector`'s values are within a `1` radius unit circle.

Next we add the camera's local `Z` vector times `input_movement_vector.y` to `dir`. This where when we pressed forward or backwards we add the camera's local `Z` axis, so we move forward in relation to the camera.

Note: Because the camera is rotated by `-180` degrees, we have to flip the `Z` directional vector. Normally forward would be the positive Z axis, so using `basis.z.normalized()` would work, but we are using `-basis.z.normalized()` because our camera's Z axis faces backwards in relation to the rest of the player.

We do the same thing for the camera's local `X` vector, and instead of using `input_movement_vector.y` we instead use `input_movement_vector.x`. This makes it where when we press left or right, we move left/right in relation to the camera.

Next we check if the player is on the floor using *KinematicBody*'s `is_on_floor` function. If it is, then we check to see if the "movement_jump" action has just been pressed. If it has, then we set our `Y` velocity to `JUMP_SPEED`.

Because we're setting the Y velocity, we will jump into the air.

Then we check for the `ui_cancel` action. This is so we can free/capture the mouse cursor when the `escape` button is pressed. We do this because otherwise we'd have no way to free the cursor, meaning it would be stuck until you terminate the runtime.

To free/capture the cursor, we check to see if the mouse is visible (freed) or not. If it is, then we capture it, and if it's not we make it visible (free it).

That's all we're doing right now for `process_input`. We'll come back several times to this function as we add more complexities to our player.

Now let's look at `process_movement`:

First we assure that `dir` does not have any movement on the `Y` axis by setting it's `Y` value to zero.

Next we normalize `dir` to assure we're within a `1` radius unit circle. This makes it where we're moving at a constant speed regardless of whether we've moving straight, or moving diagonal. If we did not normalize, we would move faster on the diagonal than when we're going straight.

Next we add gravity to our player by adding `GRAVITY * delta` to our `Y` velocity.

After that we assign our velocity to a new variable (called `hvel`) and remove any movement on the `Y` axis.

Next we set a new variable (`target`) to our direction vector. Then we multiply that by our max speed so we know how far we will can move in the direction provided by `dir`.

After that we make a new variable for our acceleration, named `accel`.

We then take the dot product of `hvel` to see if we are moving according to `hvel`. Remember, `hvel` does not have any `Y` velocity, meaning we are only checking if we are moving forwards, backwards, left, or right.

If we are moving according to `hvel`, then we set `accel` to our `ACCEL` constant so we accelerate, otherwise we set `accel` to our `DEACCEL` constant so we decelerate.

Then we interpolate our horizontal velocity, set our `X` and `Z` velocity to the interpolated horizontal velocity, and call `move_and_slide` to let the *KinematicBody* handle moving through the physics world.

Tip: All of the code in `process_movement` is exactly the same as the movement code from the Kinematic Character demo!

The final function we have is the `_input` function, and thankfully it's fairly short:

First we make sure that the event we are dealing with is a *InputEventMouseMotion* event. We also want to check if the cursor is captured, as we do not want to rotate if it is not.

Note: See *Mouse and input coordinates* for a list of possible input events.

If the event is indeed a mouse motion event and the cursor is captured, we rotate based on the relative mouse motion provided by *InputEventMouseMotion*.

First we rotate the `rotation_helper` node on the `X` axis, using the relative mouse motion's `Y` value, provided by *InputEventMouseMotion*.

Then we rotate the entire *KinematicBody* on the `Y` axis by the relative mouse motion's `X` value.

Tip: Godot converts relative mouse motion into a *Vector2* where mouse movement going up and down is `1` and `-1` respectively. Right and Left movement is `1` and `-1` respectively.

Because of how we are rotating the player, we multiply the relative mouse motion's `X` value by `-1` so mouse motion going left and right rotates the player left and right in the same direction.

Finally, we clamp the `rotation_helper`'s `X` rotation to be between `-70` and `70` degrees so we cannot rotate ourselves upside down.

Tip: see *using transforms* for more information on rotating transforms.

To test the code open up the scene named `Testing_Area.tscn`, if it's not already opened up. We will be using this scene as we go through the tutorial, so be sure to keep it open in one of your scene tabs.

Go ahead and test your code either by pressing `F4` with `Testing_Area.tscn` as the open tab, by pressing the play button in the top right corner, or by pressing `F6`. You should now be able to walk around, jump in the air, and look around using the mouse.

Giving the player a flash light and the option to sprint

Before we get to making the weapons work, there is a couple more things we should add.

Many FPS games have an option to sprint and a flash light. We can easily add these to our player, so let's do that!

First we need a few more global variables in our player script:

```
const MAX_SPRINT_SPEED = 30
const SPRINT_ACCEL = 18
var is_sprinting = false

var flashlight
```

All of the sprinting variables work exactly the same as the non sprinting variables with similar names.

`is_sprinting` is a boolean to track whether the player is currently sprinting, and `flashlight` is a variable we will be using to hold our flash light node.

Now we need to add a few lines of code, starting in `_ready`. Add the following to `_ready`:

```
flashlight = $Rotation_Helper/Flashlight
```

This gets our flash light node and assigns it to the `flashlight` variable.

Now we need to change some of the code in `process_input`. Add the following somewhere in `process_input`:

```
# ----------------------------------
# Sprinting
if Input.is_action_pressed("movement_sprint"):
    is_sprinting = true
else:
    is_sprinting = false
# ----------------------------------

# ----------------------------------
# Turning the flashlight on/off
if Input.is_action_just_pressed("flashlight"):
```

(continues on next page)

(continued from previous page)

```
    if flashlight.is_visible_in_tree():
        flashlight.hide()
    else:
        flashlight.show()
# ----------------------------------
```

Let's go over the additions:

We set `is_sprinting` to true when we are holding down the `movement_sprint` action, and false when the `movement_sprint` action is released. In `process_movement` we'll add the code that makes the player faster when they sprint. Here in `process_input` we're going to change the `is_sprinting` variable.

We do something similar freeing/capturing the cursor for handling the flash light. We first check to see if the `flashlight` action was just pressed. If it was, we then check to see if `flashlight` is visible in the scene tree. If it is, then we hide it, and if it's not we show it.

Now we need to change a couple things in `process_movement`. First, replace `target *= MAX_SPEED` with the following:

```
if is_sprinting:
    target *= MAX_SPRINT_SPEED
else:
    target *= MAX_SPEED
```

Now instead of always multiplying `target` by `MAX_SPEED`, we first check to see if we are sprinting or not. If we are sprinting, we instead multiply `target` by `MAX_SPRINT_SPEED`.

Now all that's left is changing the accleration when sprinting. Change `accel = ACCEL` to the following:

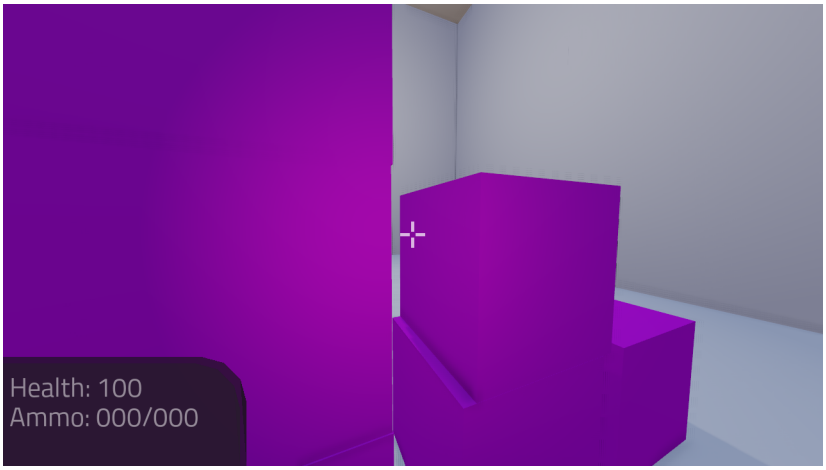
```
if is_sprinting:
    accel = SPRINT_ACCEL
else:
    accel = ACCEL
```

Now when we are sprinting we'll use `SPRINT_ACCEL` instead of `ACCEL`, which will accelerate us faster.

You should now be able to sprint if you press the `shift` button, and can toggle the flash light on and off by pressing the `F` button!

Go give it a whirl! You can change the sprint related global variables to make the player faster or slower when sprinting!

Final notes



Phew! That was a lot of work. Now you have a fully working first person character!

In *Part 2* we will add some guns to our player character.

Note: At this point we've recreated the Kinematic character demo from first person perspective with sprinting and a flash light!

Tip: Currently the player script would be at an ideal state for making all sorts of first person games. For example: Horror games, platformer games, adventure games, and more!

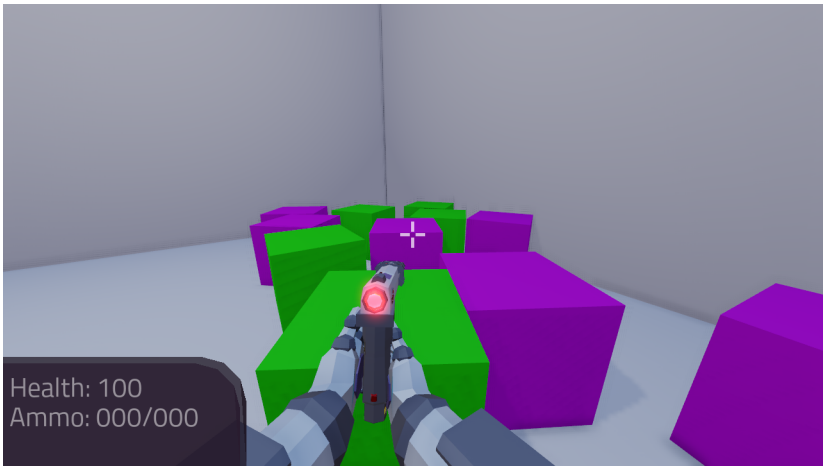
Warning: If you ever get lost, be sure to read over the code again!

You can download the finished project for this part here: `Godot_FPS_Part_1.zip`

7.16.2 Part 2

Part Overview

In this part we will be giving our player weapons to play with.



By the end of this part, you will have a player that can fire a pistol, rifle, and attack using a knife. The player will also now have animations with transitions, and the weapons will interact with objects in the environment.

Note: You are assumed to have finished *Part 1* before moving on to this part of the tutorial.

The finished project from *Part 1* will be the starting project for part 2

Let's get started!

Making a system to handle animations

First we need a way to handle changing animations. Open up `Player.tscn` and select the *AnimationPlayer* Node (`Player` -> `Rotation_Helper` -> `Model` -> `Animation_Player`).

Create a new script called `AnimationPlayer_Manager.gd` and attach that to the *AnimationPlayer*.

Add the following code to `AnimationPlayer_Manager.gd`:

```
extends AnimationPlayer

# Structure -> Animation name :[Connecting Animation states]
var states = {
    "Idle_unarmed":["Knife_equip", "Pistol_equip", "Rifle_equip", "Idle_unarmed"],

    "Pistol_equip":["Pistol_idle"],
    "Pistol_fire":["Pistol_idle"],
    "Pistol_idle":["Pistol_fire", "Pistol_reload", "Pistol_unequip", "Pistol_idle"],
    "Pistol_reload":["Pistol_idle"],
    "Pistol_unequip":["Idle_unarmed"],

    "Rifle_equip":["Rifle_idle"],
    "Rifle_fire":["Rifle_idle"],
    "Rifle_idle":["Rifle_fire", "Rifle_reload", "Rifle_unequip", "Rifle_idle"],
    "Rifle_reload":["Rifle_idle"],
    "Rifle_unequip":["Idle_unarmed"],

    "Knife_equip":["Knife_idle"],
    "Knife_fire":["Knife_idle"],
    "Knife_idle":["Knife_fire", "Knife_unequip", "Knife_idle"],
    "Knife_unequip":["Idle_unarmed"],
```

(continues on next page)

(continued from previous page)

```
}

var animation_speeds = {
    "Idle_unarmed":1,

    "Pistol_equip":1.4,
    "Pistol_fire":1.8,
    "Pistol_idle":1,
    "Pistol_reload":1,
    "Pistol_unequip":1.4,

    "Rifle_equip":2,
    "Rifle_fire":6,
    "Rifle_idle":1,
    "Rifle_reload":1.45,
    "Rifle_unequip":2,

    "Knife_equip":1,
    "Knife_fire":1.35,
    "Knife_idle":1,
    "Knife_unequip":1,
}

var current_state = null
var callback_function = null

func _ready():
    set_animation("Idle_unarmed")
    connect("animation_finished", self, "animation_ended")

func set_animation(animation_name):
    if animation_name == current_state:
        print ("AnimationPlayer_Manager.gd -- WARNING: animation is already ",␣
↪animation_name)
        return true


    if has_animation(animation_name) == true:
        if current_state != null:
            var possible_animations = states[current_state]
            if animation_name in possible_animations:
                current_state = animation_name
                play(animation_name, -1, animation_speeds[animation_name])
                return true
            else:
                print ("AnimationPlayer_Manager.gd -- WARNING: Cannot change to ",␣
↪animation_name, " from ", current_state)
                return false
        else:
            current_state = animation_name
            play(animation_name, -1, animation_speeds[animation_name])
            return true
    return false


func animation_ended(anim_name):
```

(continues on next page)

(continued from previous page)

```
    # UNARMED transitions
    if current_state == "Idle_unarmed":
        pass
    # KNIFE transitions
    elif current_state == "Knife_equip":
        set_animation("Knife_idle")
    elif current_state == "Knife_idle":
        pass
    elif current_state == "Knife_fire":
        set_animation("Knife_idle")
    elif current_state == "Knife_unequip":
        set_animation("Idle_unarmed")
    # PISTOL transitions
    elif current_state == "Pistol_equip":
        set_animation("Pistol_idle")
    elif current_state == "Pistol_idle":
        pass
    elif current_state == "Pistol_fire":
        set_animation("Pistol_idle")
    elif current_state == "Pistol_unequip":
        set_animation("Idle_unarmed")
    elif current_state == "Pistol_reload":
        set_animation("Pistol_idle")
    # RIFLE transitions
    elif current_state == "Rifle_equip":
        set_animation("Rifle_idle")
    elif current_state == "Rifle_idle":
        pass;
    elif current_state == "Rifle_fire":
        set_animation("Rifle_idle")
    elif current_state == "Rifle_unequip":
        set_animation("Idle_unarmed")
    elif current_state == "Rifle_reload":
        set_animation("Rifle_idle")

func animation_callback():
    if callback_function == null:
        print ("AnimationPlayer_Manager.gd -- WARNING: No callback function for the␣
↪animation to call!")
    else:
        callback_function.call_func()
```

Lets go over what this script is doing:

Lets start with this script's global variables:

- `states`: A dictionary for holding our animation states. (Further explanation below)
- `animation_speeds`: A dictionary for holding all of the speeds we want to play our animations at.
- `current_state`: A variable for holding the name of the animation state we are currently in.
- `callback_function`: A variable for holding the callback function. (Further explanation below)

If you are familiar with state machines, then you may have noticed that `states` is structured like a basic state machine. Here is roughly how `states` is set up:

`states` is a dictionary with the key being the name of the current state, and the value being an array holding all of the states we can transition to. For example, if we are in currently in state `Idle_unarmed`, we can only transition to `Knife_equip`, `Pistol_equip`, `Rifle_equip`, and `Idle_unarmed`.

If we try to transition to a state that is not included in our possible transitions states, then we get a warning message and the animation does not change. We can also automatically transition from some states into others, as will be explained further below in `animation_ended`

Note: For the sake of keeping this tutorial simple we are not using a 'proper' state machine. If you are interested to know more about state machines, see the following articles:

- (Python example) https://dev.to/karn/building-a-simple-state-machine-in-python
- (C# example) https://www.codeproject.com/Articles/489136/UnderstandingplusandplusImplementingplusStateplusP
- (Wiki article) https://en.wikipedia.org/wiki/Finite-state_machine

---

`animation_speeds` is how fast each animation will play. Some of the animations are a little slow and in an effort to make everything look smooth, we need to play them at faster speeds.

Tip: Notice that all of the firing animations are faster than their normal speed. Remember this for later!

`current_state` will hold the name of the animation state we are currently in.

Finally, `callback_function` will be a *FuncRef* passed in by our player for spawning bullets at the proper frame of animation. A *FuncRef* allows us to pass in a function as an argument, effectively allowing us to call a function from another script, which is how we will use it later.

Now lets look at `_ready`.

First we are setting our animation to `Idle_unarmed` using the `set_animation` function, so we for sure start in that animation.

Next we connect the `animation_finished` signal to this script and assign it to call `animation_ended`. This means whenever an animation is finished, `animation_ended` will be called.

Lets look at `set_animation` next.

`set_animation` changes the animation to the the animation named `animation_name` *if* we can transition to it. In other words, if the animation state we are currently in has the passed in animation state name in `states`, then we will change to that animation.

To start we check if the passed in animation name is the same as name as the animation currently playing. If they are the same, then we write a warning to the console and return `true`.

Next we see if *AnimationPlayer* has the a animation with the name `animation_name` using `has_animation`. If it does not, we return `false`.

Then we check if `current_state` is set or not. If `current_state` is *not* currently set, we set `current_state` to the passed in animation name and tell *AnimationPlayer* to start playing the animation with a blend time of `-1` at the speed set in `animation_speeds` and then we return `true`.

Note: Blend time is how long to blend/mix the two animations together.

By putting in a value of `-1`, the new animation instantly plays, overriding whatever animation is already playing.

If you put in a value of `1`, for one second the new animation will play with increasing strength, blending the two animations together for one second before playing only the new animation. This leads to a smooth transition between animations, which is looks great when you are changing from a walking animation to a running animation.

We set the blend time to `-1` because we want to instantly change animations.

If we have a state in `current_state`, then we get all of the possible states we can transition to.

If the animation name is in the list of possible transitions, we set `current_state` to the passed in animation (`animation_name`), tell *AnimationPlayer* to play the animation with a blend time of `-1` at the speed set in `animation_speeds` and return `true`.

Now lets look at `animation_ended`.

`animation_ended` is the function that will be called by *AnimationPlayer* when it's done playing a animation.

For certain animation states, we may need to transition into another state when its finished. To handle this, we check for every possible animation state. If we need to, we transition into another state.

Warning: If you are using your own animated models, make sure that none of the animations are set to loop. Looping animations do not send the `animation_finished` signal when they reach the end of the animation and are about to loop again.

Note: the transitions in `animation_ended` ideally would be part of the data in `states`, but in an effort to make the tutorial easier to understand, we'll hard code each state transition in `animation_ended`.

Finally we have `animation_callback`. This function will be called by a function track in our animations. If we have a *FuncRef* assigned to `callback_function`, then we call that passed in function. If we do not have a *FuncRef* assigned to `callback_function`, we print out a warning to the console.

Tip: Try running `Testing_Area.tscn` to make sure there is no runtime issues. If the game runs but nothing seems to have changed, then everything is working correctly.

Getting the animations ready

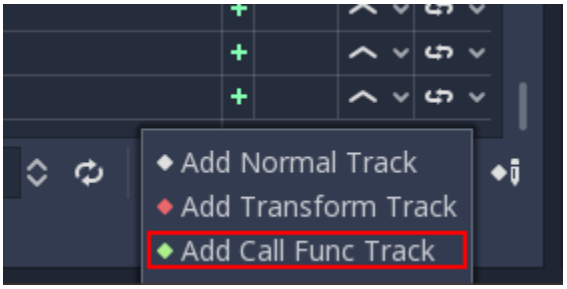
Now that we have a working animation manager, we need to call it from our player script. Before that though, we need to set some animation callback tracks in our firing animations.

Open up `Player.tscn` if you don't have it open and navigate to the *AnimationPlayer* node (`Player` -> `Rotation_Helper` -> `Model` -> `Animation_Player`).

We need to attach a function track to three of our animations: The firing animation for the pistol, rifle, and knife. Let's start with the pistol. Click the animation drop down list and select "Pistol_fire".

Now scroll down to the bottom of the list of animation tracks. The final item in the list should read `Armature/Skeleton:Left_UpperPointer`. Now at the bottom of the list, click the plus icon on the bottom bar of animation window, right next to the loop button and the up arrow.

This will bring up a window with three choices. We're wanting to add a function callback track, so click the option that reads "Add Call Func Track". This will open a window showing the entire node tree. Navigate to the *AnimationPlayer* node, select it, and press OK.



Now at the bottom of list of animation tracks you will have a green track that reads "AnimationPlayer". Now we need to add the point where we want to call our callback function. Scrub the timeline until you reach the point where the muzzle starts to flash.

Note: The timeline is the window where all of the points in our animation are stored. Each of the little points represents a point of animation data.

Scrubbing the timeline means moving ourselves through the animation. So when we say "scrub the timeline until you reach a point", what we mean is move through the animation window until you reach the a point on the timeline.

Also, the muzzle of a gun is the end point where the bullet comes out. The muzzle flash is the flash of light that escapes the muzzle when a bullet is fired. The muzzle is also sometimes referred to as the barrel of the gun.

Tip: For finer control when scrubbing the timeline, press `control` and scroll forwards with the mouse wheel to zoom in. Scrolling backwards will zoom out.

You can also change how the timeline scrubbing snaps by changing the value in `Step (s)` to a lower/higher value.

Once you get to a point you like, press the little green plus symbol on the far right side of the `AnimationPlayer` track. This will place a little green point at the position you are currently at in the animation on your `AnimationPlayer` track.

Now we have one more step before we are done with the pistol. Select the "enable editing of individual keys" button on the far right corner of the animation window. It looks like a pencil with a little point beside it.

Once you've click that, a new window will open on the right side. Now click the green point on the `AnimationPlayer` track. This will bring up the information associated with that point in the timeline. In the empty name field, enter `animation_callback` and press `enter`.

Now when we are playing this animation the callback function will be triggered at that specific point of the animation.

Warning: Be sure to press the "enable editing of individual keys" button again to turn off the ability to edit individual keys so you cannot change one of the transform tracks by accident!

Let's repeat the process for the rifle and knife firing animations!

Note: Because the process is exactly the same as the pistol, the process is going to explained in a little less depth. Follow the steps in the above if you get lost! It is exactly the same, just on a different animation.

Go to the "Rifle_fire" animation from the animation drop down. Add the function callback track once you reach the bottom of the animation track list by clicking the little plus icon at the bottom of the screen. Find the point where the muzzle starts to flash and click the little green plus symbol to add a function callback point at that position on the track.

Next, click the "enable editing of individual keys" button. Select the newly created function callback point, put "animation_callback" into the name field and press `enter`. Click the "enable editing of individual keys" button again to turn off individual key editing. so we cannot change one of the transform tracks by accident.

Now we need to apply the callback function track to the knife animation. Select the "Knife_fire" animation and scroll to the bottom of the animation tracks. Click the plus symbol at the bottom of the animation window and add a function callback track. Next find a point around the first third of the animation to place the animation callback function point at.

Note: We will not actually be firing the knife, and the animation is a stabbing animation rather than a firing one. For this tutorial we are reusing the gun firing logic for our knife, so the animation has been named in a style that is consistent with the other animations.

From there click the little green plus to add a function callback point at the current position. Then click the "enable editing of individual keys" button, the button with a plus at the bottom right side of the animation window. Select the newly created function callback point, put "animation_callback" into the name field and press `enter`. Click the "enable editing of individual keys" button again to turn off individual key editing. so we cannot change one of the transform tracks by accident.

Tip: Be sure to save your work!

With that done, we are almost ready to start adding the ability to fire to our player script! We need to setup one last scene: The scene for our bullet object.

Creating the bullet scene

There are several ways to handle a gun's bullets in video games. In this tutorial series, we will be exploring two of the more common ways: Objects, and raycasts.

One of the two ways is using a bullet object. This will be an object that travels through the world and handles its own collision code. This method we create/spawn a bullet object in the direction our gun is facing, and then it sends itself forward.

There are several advantages to this method. The first being we do not have to store the bullets in our player. We can simply create the bullet and then move on, and the bullet itself with handle checking for collisions, sending the proper signal(s) to the object it collides with, and destroying itself.

Another advantage is we can have more complex bullet movement. If we want to make the bullet fall ever so slightly as time goes on, we can make the bullet controlling script slowly push the bullet towards the ground. Using a object also makes the bullet take time to reach its target, it doesn't instantly hit whatever its pointed at. This feels more realistic because nothing in real life moves instantly from one point to another.

One of the huge disadvantages performance. While having each bullet calculate their own paths and handle their own collision allows for a lot of flexibility, it comes at the cost of performance. With this method we are calculating every bullet's movement every step, and while this may not be a problem for a few dozen bullets, it can become a huge problem when you potentially have several hundred bullets.

Despite the performance hit, many first person shooters include some form of object bullets. Rocket launchers are a prime example because in many first person shooters, rockets do not just instantly explode at their target position. You can also find bullets as object many times with grenades because they generally bounce around the world before exploding.

Note: While I cannot say for sure this is the case, these games *probably* use bullet objects in some form or another: (These are entirely from my observations. **They may be entirely wrong**. I have never worked on **any** of the following games)

- Halo (Rocket launchers, fragment grenades, sniper rifles, brute shot, and more)
- Destiny (Rocket launchers, grenades, fusion rifles, sniper rifles, super moves, and more)
- Call of Duty (Rocket launchers, grenades, ballistic knives, crossbows, and more)
- Battlefield (Rocket launchers, grenades, claymores, mortars, and more)

Another disadvantage with bullet objects is networking. Bullet objects have to sync the positions (at least) with however many clients are connected to the server.

While we are not implementing any form of networking (as that would be it's own entire tutorial series), it is a consideration to keep in mind when creating your first person shooter, especially if you plan on adding some form of networking in the future.

The other way of handling bullet collisions we will be looking at, is raycasting.

This method is extremely common in guns that have fast moving bullets that rarely change trajectory change over time.

Instead of creating a bullet object and sending it through space, we instead send a ray starting from the barrel/muzzle of the gun forwards. We set the raycast's origin to the starting position of the bullet, and based on the length we can adjust how far the bullet 'travels' through space.

Note: While I cannot say for sure this is the case, these games *probably* use raycasts in some form or another: (These are entirely from my observations. **They may be entirely wrong**. I have never worked on **any** of the following games)

- Halo (Assault rifles, DMRs, battle rifles, covenant carbine, spartan laser, and more)
- Destiny (Auto rifles, pulse rifles, scout rifles, hand cannons, machine guns, and more)
- Call of Duty (Assault rifles, light machine guns, sub machine guns, pistols, and more)
- Battlefield (Assault rifles, SMGs, carbines, pistols, and more)

One huge advantage for this method is it's light on performance. Sending a couple hundred rays through space is *way* easier for the computer to calculate than sending a couple hundred bullet objects.

Another advantage is we can instantly know if we've hit something or not exactly when we call for it. For networking this is important because we do not need to sync the bullet movements over the Internet, we only need to send whether or not the raycast hit.

Raycasting does have some disadvantages though. One major disadvantage is we cannot easily cast a ray in anything but a linear line. This means we can only fire in a straight line for however long our ray length is. You can create the illusion of bullet movement by casting multiple rays at different positions, but not only is this hard to implement in code, it is also is heavier on performance.

Another disadvantage is we cannot see the bullet. With bullet objects we can actually see the bullet travel through space if we attach a mesh to it, but because raycasts happen instantly, we do not have a decent way of showing the bullets. You could draw a line from the origin of the raycast to the point where the raycast collided, and that is one popular way of showing raycasts. Another way is simply not drawing the raycast at all, because theoretically the bullets move so fast our eyes could not see it anyway.

Lets get the bullet object setup. This is what our pistol will create when the "Pistol_fire" animation callback function is called.

Open up `Bullet_Scene.tscn`. The scene contains *Spatial* node called bullet, with a *MeshInstance* and an *Area* with a *CollisionShape* childed to it.

Create a new script called `Bullet_script.gd` and attach it to the `Bullet` *Spatial*.

We are going to move the entire bullet object at the root (`Bullet`). We will be using the *Area* to check whether or not we've collided with something

---

Note: Why are we using a *Area* and not a *RigidBody*? The mean reason we're not using a *RigidBody* is because we do not want the bullet to interact with other *RigidBody* nodes. By using an *Area* we are assuring that none of the other *RigidBody* nodes, including other bullets, will be effected.

Another reason is simply because it is easier to detect collisions with a *Area*!

Here's the script that will control our bullet:

```
extends Spatial

var BULLET_SPEED = 70
var BULLET_DAMAGE = 15

const KILL_TIMER = 4
var timer = 0

var hit_something = false

func _ready():
    $Area.connect("body_entered", self, "collided")


func _physics_process(delta):
    var forward_dir = global_transform.basis.z.normalized()
    global_translate(forward_dir * BULLET_SPEED * delta)

    timer += delta
    if timer >= KILL_TIMER:
        queue_free()


func collided(body):
    if hit_something == false:
        if body.has_method("bullet_hit"):
            body.bullet_hit(BULLET_DAMAGE, self.global_transform.origin)

    hit_something = true
    queue_free()
```

Lets go through the script:

First we define a few global variables:

- `BULLET_SPEED`: The speed the bullet travels at.
- `BULLET_DAMAGE`: The damage the bullet will cause to whatever it collides with.
- `KILL_TIMER`: How long the bullet can last without hitting anything.
- `timer`: A float for tracking how long we've been alive.
- `hit_something`: A boolean for tracking whether or not we've hit something.

With the exception of `timer` and `hit_something`, all of these variables change how the bullet interacts with the world.

Note: The reason we are using a kill timer is so we do not have a case where we get a bullet travelling forever. By using a kill timer, we can assure that no bullets will travel forever and consume resources.

Tip: As in *Part 1*, we have a couple all uppercase global variables. The reason behind this is the same as the reason given in *Part 1*: We want to treat these variables like constants, but we want to be able to change them. In this case we will later need to change the damage and speed of these bullets, so we need them to be variables and not constants.

In `_ready` we set the area's `body_entered` signal to ourself so that it calls the `collided` function when a body enters the area.

`_physics_process` gets the bullet's local `Z` axis. If you look in at the scene in local mode, you will find that the bullet faces the positive local `Z` axis.

Next we translate the entire bullet by that forward direction, multiplying in our speed and delta time.

After that we add delta time to our timer and check if the timer has as long or longer than our `KILL_TIME` constant. If it has, we use `queue_free` to free ourselves.

In `collided` we check if we've hit something yet or not.

Remember that `collided` is only called when a body has entered the *Area* node. If we have not already collided with something, we then proceed to check if the body we've collided with has a function/method called `bullet_hit`. If it does, we call it and pass in our damage and our position.

Note: in `collided`, the passed in body can be a *StaticBody*, *RigidBody*, or *KinematicBody*

We set `hit_something` to `true` because regardless of whether or not the body the bullet collided with has the `bullet_hit` function/method, it has hit something and so we need to not hit anything else.

Then we free the bullet using `queue_free`.

Tip: You may be wondering why we even have a `hit_something` variable if we free the bullet using `queue_free` as soon as it hits something.

The reason we need to track whether we've hit something or not is because `queue_free` does not immediately free the node, so the bullet could collide with another body before Godot has a chance to free it. By tracking if the bullet has hit something we can make sure that the bullet will only hit one object.

Before we start programming the player again, let's take a quick look at `Player.tscn`. Open up `Player.tscn` again.

Expand `Rotation_Helper` and notice how it has two nodes: `Gun_Fire_Points` and `Gun_Aim_Point`.

`Gun_aim_point` is the point that the bullets will be aiming at. Notice how it is lined up with the center of the screen and pulled a distance forward on the Z axis. `Gun_aim_point` will serve as the point where the bullets will for sure collide with as it goes along.

Note: There is a invisible mesh instance for debugging purposes. The mesh is a small sphere that visually shows where the bullets will be aiming at.

Open up `Gun_Fire_Points` and you'll find three more *Spatial* nodes, one for each weapon.

Open up `Rifle_Point` and you'll find a *Raycast* node. This is where we will be sending the raycasts for our rifle's bullets. The length of the raycast will dictate how far our the bullets will travel.

We are using a *Raycast* node to handle the rifle's bullet because we want to fire lots of bullets quickly. If we use bullet objects, it is quite possible we could run into performance issues on older machines.

Note: If you are wondering where the positions of the points came from, they are the rough positions of the ends of each weapon. You can see this by going to `AnimationPlayer`, selecting one of the firing animations and scrubbing through the timeline. The point for each weapon should mostly line up with the end of each weapon.

Open up `Knife_Point` and you'll find a *Area* node. We are using a *Area* for the knife because we only care for all of the bodies close to us, and because our knife does not fire into space. If we were making a throwing knife, we would likely spawn a bullet object that looks like a knife.

Finally, we have `Pistol_Point`. This is the point where we will be creating/instancing our bullet objects. We do not need any additional nodes here, as the bullet handles all of its own collision detection.

Now that we've seen how we will handle our other weapons, and where we will spawn the bullets, let's start working on making them work.

Note: You can also look at the HUD nodes if you want. There is nothing fancy there and other than using a single *Label*, we will not be touching any of those nodes. Check *Design interfaces with the Control nodes* for a tutorial on using GUI nodes.

Creating the first weapon

Lets write the code for each of our weapons, starting with the pistol.

Select `Pistol_Point` (`Player` -> `Rotation_Helper` -> `Gun_Fire_Points` -> `Pistol_Point`) and create a new script called `Weapon_Pistol.gd`.

Add the following code to `Weapon_Pistol.gd`:

```
extends Spatial

const DAMAGE = 15

const IDLE_ANIM_NAME = "Pistol_idle"
const FIRE_ANIM_NAME = "Pistol_fire"

var is_weapon_enabled = false

var bullet_scene = preload("Bullet_Scene.tscn")

var player_node = null

func _ready():
    pass

func fire_weapon():
    var clone = bullet_scene.instance()
    var scene_root = get_tree().root.get_children()[0]
    scene_root.add_child(clone)

    clone.global_transform = self.global_transform
    clone.scale = Vector3(4, 4, 4)
    clone.BULLET_DAMAGE = DAMAGE

func equip_weapon():
    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
```

(continues on next page)

(continued from previous page)

```
        is_weapon_enabled = true
        return true

    if player_node.animation_manager.current_state == "Idle_unarmed":
        player_node.animation_manager.set_animation("Pistol_equip")

    return false

func unequip_weapon():
    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        if player_node.animation_manager.current_state != "Pistol_unequip":
            player_node.animation_manager.set_animation("Pistol_unequip")

    if player_node.animation_manager.current_state == "Idle_unarmed":
        is_weapon_enabled = false
        return true
    else:
        return false
```

Let's go over how the script works.

First we define some global variables we'll need in the script:

- `DAMAGE`: The amount of damage a single bullet does.
- `IDLE_ANIM_NAME`: The name of the pistol's idle animation.
- `FIRE_ANIM_NAME`: The name of the pistol's fire animation.
- `is_weapon_enabled`: A variable for checking whether this weapon is in use/enabled.
- `bullet_scene`: The bullet scene we worked on earlier.
- `player_node`: A variable to hold `Player.gd`.

The reason we define most of these variables is so we can use them in `Player.gd`.

All of the weapons we'll make will have all of these variables (minus `bullet_scene`) so we have a consistent interface to interact with in `Player.gd`. By using the same variables/functions in each weapon, we can interact with them without having to know which weapon we are using, which makes our code much more modular because we can add weapons without having to change much of the code in `Player.gd` and it will just work.

If we could write all of the code in `Player.gd`, but then `Player.gd` will get increasingly harder to manage as we add weapons. By using a modular design with a consistent interface, we can keep `Player.gd` nice and neat, while also making it easier to add/remove/modify weapons.

In `_ready` we simply pass over it.

There is one thing of note though, an assumption we're assuming we'll fill in `Player.gd`.

We are going to assume that `Player.gd` will pass themselves in before calling any of the functions in `Weapon_Pistol.gd`.

While this can lead to situations where the player does not pass themselves in (because we forget), we would have to have a long string of `get_parent` calls to traverse up the scene tree to retrieve the player. This does not look pretty (`get_parent().get_parent().get_parent()` and so on) and it is relatively safe to assume we will remember to pass ourselves to each weapon in `Player.gd`.

Next let's look at `fire_weapon`:

The first thing we do is instance the bullet scene we made earlier.

Tip: By instancing the scene, we are creating a new node holding all of the node(s) in the scene we instanced, effectively cloning that scene.

Then we add `clone` to the first child node of the root of the scene we are currently in. By doing this we're making it at a child of the root node of the currently loaded scene.

In other words, we are adding `clone` as a child of the first node (whatever is at the top of the scene tree) in the currently loaded/opened scene. If the currently loaded/open scene is `Testing_Area.tscn`, we'd be adding our `clone` as a child of `Testing_Area`, the root node in that scene.

Warning: As mentioned later below in the section on adding sounds, this method makes a assumption. This will be explained later in the section on adding sounds in *Part 3*

Next we set the global transform of the clone to the `Pistol_Aim_Point`'s global transform. The reason we do this is so the bullet is spawned at the end of the pistol.

You can see that `Pistol_Aim_Point` is positioned right at the end of the pistol by clicking the *AnimationPlayer* and scrolling through `Pistol_fire`. You'll find the position more or less is at the end of the pistol when it fires.

Next we scale it up by a factor of `4` because the bullet scene is a little too small by default.

Then we set the bullet's damage (`BULLET_DAMAGE`) to the amount of damage a single pistol bullet does (`DAMAGE`)

Now let's look at `equip_weapon`:

The first thing we do is check to see if the animation manager is in the pistol's idle animation. If we are in the pistol's idle animation, we set `is_weapon_enabled` to `true` and return `true` because we have successfully been equipped.

Because we know our pistol's `equip` animation automatically transitions to the pistol's idle animation, if we are in the pistol's idle animation we most have finished playing the equip animation.

Note: We know these animations will transition because we wrote to the code to make them transition in `Animation_Manager.gd`

Next we check to see if we are in the `Idle_unarmed` animation state. Because all unequipping animations go to this state, and because any weapon can be equipped from this state, we change animations to `Pistol_equip` if we are in `Idle_unarmed`.

Since we know `Pistol_equip` will transition to `Pistol_idle`, we do not need to do any more additional processing for equipping weapons, but since we were not able to equip the pistol yet, we return `false`.

Finally, let's look at `unequip_weapon`:

`unequip_weapon` is similar to `equip_weapon`, but instead we're checking things in reverse.

First we check to see if we are in our idle animation. Then check to make sure we are not in the `Pistol_unequip` animation. If we are not in the `Pistol_unequip` animation, we want to play `pistol_unequip`.

Note: You may be wondering why we are checking to see if we are the pistol's idle animation, and then making sure we are not unequipping right after. The reason behind the additional check is because we could (in rare cases) call `unequip_weapon` twice before we've had a chance to process `set_animation`, so we add this additional check to make sure the unequip animation plays.

Next we check to see if we are in `Idle_unarmed`, which is the animation state we will transition into from `Pistol_unequip`. If we are, then we set `is_weapon_enabled` to false since we are no longer using this weapon, and return `true` because we have successfully unequipped the pistol.

If we are not in `Idle_unarmed`, we return `false` because we have not yet successfully unequipped the pistol.

Creating the other two weapons

Now that we all of the code we'll need for the pistol, let's add the code for the rifle and knife next.

Select `Rifle_Point` (`Player` -> `Rotation_Helper` -> `Gun_Fire_Points` -> `Rifle_Point`) and create a new script called `Weapon_Rifle.gd`, then add the following:

```
extends Spatial

const DAMAGE = 4

const IDLE_ANIM_NAME = "Rifle_idle"
const FIRE_ANIM_NAME = "Rifle_fire"

var is_weapon_enabled = false

var player_node = null

func _ready():
    pass

func fire_weapon():
    var ray = $Ray_Cast
    ray.force_raycast_update()

    if ray.is_colliding():
        var body = ray.get_collider()

        if body == player_node:
            pass
        elif body.has_method("bullet_hit"):
            body.bullet_hit(DAMAGE, ray.get_collision_point())

func equip_weapon():
    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        is_weapon_enabled = true
        return true

    if player_node.animation_manager.current_state == "Idle_unarmed":
        player_node.animation_manager.set_animation("Rifle_equip")

    return false

func unequip_weapon():
```

(continues on next page)

(continued from previous page)

```
    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        if player_node.animation_manager.current_state != "Rifle_unequip":
            player_node.animation_manager.set_animation("Rifle_unequip")

    if player_node.animation_manager.current_state == "Idle_unarmed":
        is_weapon_enabled = false
        return true

    return false
```

Most of this is exactly the same as `Weapon_Pistol.gd`, so we're only going to look at what's changed: `fire_weapon`.

The first thing we do is get the *Raycast* node, which is a child of `Rifle_Point`.

Next we force the raycast to update using `force_raycast_update`. This will force the raycast to detect collisions when we call it, meaning we get a frame perfect collision check with the 3D physics world.

Then we check to see if the raycast collided with something.

If the raycast has collided with something, we first get the collision body it collided with. This can be a *StaticBody*, *RigidBody*, or a *KinematicBody*.

Next we want to make sure the body we've collided with is not the player, since we (probably) do not want to give the player the ability to shoot themselves in the foot.

If the body is not the player, we then check to see if they have a function/method called `bullet_hit`. If they do, we call it and pass in the amount of damage this bullet does (`DAMAGE`), and the point where the raycast collided with the body.

Now all we need to do is write the code for the knife.

Select `Knife_Point` (`Player` -> `Rotation_Helper` -> `Gun_Fire_Points` -> `Knife_Point`) and create a new script called `Weapon_Knife.gd`, then add the following:

```
extends Spatial

const DAMAGE = 40

const IDLE_ANIM_NAME = "Knife_idle"
const FIRE_ANIM_NAME = "Knife_fire"

var is_weapon_enabled = false

var player_node = null

func _ready():
    pass

func fire_weapon():
    var area = $Area
    var bodies = area.get_overlapping_bodies()

    for body in bodies:
        if body == player_node:
            continue
```

(continues on next page)

(continued from previous page)

```
        if body.has_method("bullet_hit"):
            body.bullet_hit(DAMAGE, area.global_transform.origin)

func equip_weapon():
    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        is_weapon_enabled = true
        return true

    if player_node.animation_manager.current_state == "Idle_unarmed":
        player_node.animation_manager.set_animation("Knife_equip")

    return false

func unequip_weapon():

    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        player_node.animation_manager.set_animation("Knife_unequip")

    if player_node.animation_manager.current_state == "Idle_unarmed":
        is_weapon_enabled = false
        return true

    return false
```

As with `Weapon_Rifle.gd`, the only differences are in `fire_weapon`, so let's look at that:

The first thing we do is get the *Area* child node of `Knife_Point`.

Next we want to get all of the collision bodies inside the area using `get_overlapping_bodies`. This will return a list of every body that touches the area.

We next want to go through each of those bodies.

First we check to make sure the body is not the player, because we do not want to be able to stab ourselves. If the body is the player, we use `continue` so we jump to looking at the next body in `bodies`.

If we have not jumped to the next body, we then check to see if the body has the `bullet_hit` function/method. If it does, we call it, passing in the amount of damage a single knife swipe does (`DAMAGE`) and the position of the *Area*.

Note: While we could attempt to calculate a rough location for where the knife hit, we do not bother because using the area's position works well enough and the extra time needed to calculate a rough position for each body is not worth the effort.

Making the weapons work

Lets start making the weapons work in `Player.gd`.

First lets start by adding some global variables we'll need for the weapons:

```
# Place before _ready
var animation_manager

var current_weapon_name = "UNARMED"
var weapons = {"UNARMED":null, "KNIFE":null, "PISTOL":null, "RIFLE":null}
```

(continues on next page)

(continued from previous page)

```
const WEAPON_NUMBER_TO_NAME = {0:"UNARMED", 1:"KNIFE", 2:"PISTOL", 3:"RIFLE"}
const WEAPON_NAME_TO_NUMBER = {"UNARMED":0, "KNIFE":1, "PISTOL":2, "RIFLE":3}
var changing_weapon = false
var changing_weapon_name = "UNARMED"

var health = 100

var UI_status_label
```

Lets go over what these new variables will do:

- `animation_manager`: This will hold the *AnimationPlayer* node and its script, which we wrote previously.
- `current_weapon_name`: The name of the weapon we are currently using. It has four possible values: `UNARMED`, `KNIFE`, `PISTOL`, and `RIFLE`.
- `weapons`: A dictionary that will hold all of the weapon nodes.
- `WEAPON_NUMBER_TO_NAME`: A dictionary allowing us to convert from a weapon's number to its name. We'll use this for changing weapons.
- `WEAPON_NAME_TO_NUMBER`: A dictionary allowing us to convert from a weapon's name to its number. We'll use this for changing weapons.
- `changing_weapon`: A boolean to track whether or not we are changing guns/weapons.
- `changing_weapon_name`: The name of the weapon we want to change to.
- `health`: How much health our player has. In this part of the tutorial we will not be using it.
- `UI_status_label`: A label to show how much health we have, and how much ammo we have both in our gun and in reserves.

Next we need to add a few things in `_ready`. Here's the new `_ready` function:

```
func _ready():
    camera = $Rotation_Helper/Camera
    rotation_helper = $Rotation_Helper

    animation_manager = $Rotation_Helper/Model/Animation_Player
    animation_manager.callback_function = funcref(self, "fire_bullet")

    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)

    weapons["KNIFE"] = $Rotation_Helper/Gun_Fire_Points/Knife_Point
    weapons["PISTOL"] = $Rotation_Helper/Gun_Fire_Points/Pistol_Point
    weapons["RIFLE"] = $Rotation_Helper/Gun_Fire_Points/Rifle_Point

    var gun_aim_point_pos = $Rotation_Helper/Gun_Aim_Point.global_transform.origin

    for weapon in weapons:
        var weapon_node = weapons[weapon]
        if weapon_node != null:
            weapon_node.player_node = self
            weapon_node.look_at(gun_aim_point_pos, Vector3(0, 1, 0))
            weapon_node.rotate_object_local(Vector3(0, 1, 0), deg2rad(180))

    current_weapon_name = "UNARMED"
```

(continues on next page)

(continued from previous page)

```
    changing_weapon_name = "UNARMED"

    UI_status_label = $HUD/Panel/Gun_label
    flashlight = $Rotation_Helper/Flashlight
```

Let's go over what's changed.

First we get the *AnimationPlayer* node and assign it to our animation_manager variable. Then we set the callback function to a *FuncRef* that will call the player's `fire_bullet` function. Right now we haven't written our fire_bullet function, but we'll get there soon.

Next we get all of the weapon nodes and assign them to `weapons`. This will allow us to access the weapon nodes only with their name (`KNIFE`, `PISTOL`, or `RIFLE`).

We then get `Gun_Aim_Point`'s global position so we can rotate our weapons to aim at it.

Then we go through each weapon in `weapons`.

We first get the weapon node. If the weapon node is not `null`, we then set it's `player_node` variable to ourself. Then we have it look at `gun_aim_point_pos`, and then rotate it by `180` degrees on the `Y` axis.

Note: We rotate all of those weapon points by `180` degrees on their `Y` axis because our camera is pointing backwards. If we did not rotate all of these weapon points by `180` degrees, all of the weapons would fire backwards.

Then we set `current_weapon_name` and `changing_weapon_name` to `UNARMED`.

Finally, we get the UI *Label* from our HUD.

Lets add a new function call to `_physics_process` so we can change weapons. Here's the new code:

```
func _physics_process(delta):
    process_input(delta)
    process_movement(delta)
    process_changing_weapons(delta)
```

Now we will call `process_changing_weapons`.

Now lets add all of the player input code for the weapons in `process_input`. Add the following code:

```
# ----------------------------------
# Changing weapons.
var weapon_change_number = WEAPON_NAME_TO_NUMBER[current_weapon_name]

if Input.is_key_pressed(KEY_1):
    weapon_change_number = 0
if Input.is_key_pressed(KEY_2):
    weapon_change_number = 1
if Input.is_key_pressed(KEY_3):
    weapon_change_number = 2
if Input.is_key_pressed(KEY_4):
    weapon_change_number = 3

if Input.is_action_just_pressed("shift_weapon_positive"):
    weapon_change_number += 1
```

(continues on next page)

(continued from previous page)

```
if Input.is_action_just_pressed("shift_weapon_negative"):
    weapon_change_number -= 1

weapon_change_number = clamp(weapon_change_number, 0, WEAPON_NUMBER_TO_NAME.size()-1)

if changing_weapon == false:
    if WEAPON_NUMBER_TO_NAME[weapon_change_number] != current_weapon_name:
        changing_weapon_name = WEAPON_NUMBER_TO_NAME[weapon_change_number]
        changing_weapon = true
# ---------------------------------

# ---------------------------------
# Firing the weapons
if Input.is_action_pressed("fire"):
    if changing_weapon == false:
        var current_weapon = weapons[current_weapon_name]
        if current_weapon != null:
            if animation_manager.current_state == current_weapon.IDLE_ANIM_NAME:
                animation_manager.set_animation(current_weapon.FIRE_ANIM_NAME)
    # ---------------------------------
```

Lets go over the additions, starting with how we're changing weapons.

First we get the current weapon's number and assign it to `weapon_change_number`.

Then we check to see if any of the number keys (keys 1-4) are pressed. If they are, we set `weapon_change_number` to the value mapped at that key.

Note: The reason key 1 is mapped to `0` is because the first element in a list is mapped to zero, not one. Most list/array accessors in most programming languages start at `0` instead of `1`. See https://en.wikipedia.org/wiki/Zero-based_numbering for more information.

Next we check to see if `shift_weapon_positive` or `shift_weapon_negative` is pressed. If one of them are, we add/subtract `1` from `weapon_change_number`.

Because we may have shifted `weapon_change_number` outside of the number of weapons we have, we clamp it so it cannot exceed the maximum number of weapons we have and has to be `0` or more.

Then we check to make sure we are not already changing weapons. If we are not, we then check to see if the weapon we want to change to is a new weapon and not the one we are currently using. If the weapon we're wanting to change to is a new weapon, we then set `changing_weapon_name` to the weapon at `weapon_change_number` and set `changing_weapon` to true.

For firing the weapon we first check to see if the `fire` action is pressed. Then we check to make sure we are not changing weapons. Next we get the weapon node for the current weapon.

If the current weapon node does not equal null, and we are in it's `IDLE_ANIM_NAME` state, we set our animation to the current weapon's `FIRE_ANIM_NAME`.

Lets add `process_changing_weapons` next.

Add the following code:

```
func process_changing_weapons(delta):
    if changing_weapon == true:
```

(continues on next page)

(continued from previous page)

```
    var weapon_unequipped = false
    var current_weapon = weapons[current_weapon_name]

    if current_weapon == null:
        weapon_unequipped = true
    else:
        if current_weapon.is_weapon_enabled == true:
            weapon_unequipped = current_weapon.unequip_weapon()
        else:
            weapon_unequipped = true

    if weapon_unequipped == true:

        var weapon_equiped = false
        var weapon_to_equip = weapons[changing_weapon_name]

        if weapon_to_equip == null:
            weapon_equiped = true
        else:
            if weapon_to_equip.is_weapon_enabled == false:
                weapon_equiped = weapon_to_equip.equip_weapon()
            else:
                weapon_equiped = true

        if weapon_equiped == true:
            changing_weapon = false
            current_weapon_name = changing_weapon_name
            changing_weapon_name = ""
```

Lets go over what's happening here:

The first thing we do is make sure we've recived input to change weapons. We do this by making sure `changing_weapons` is `true`.

Next we define a variable (`weapon_unequipped`) so we can check whether the current weapon has been successfully unequipped or not.

Then we get the current weapon from `weapons`.

If the current weapon is not `null`, then we have need to check to see if the weapon is enabled or not. If the weapon is enabled, we call it's `unequip_weapon` function so it will start the unequip animation. If the weapon is not enabled, we set `weapon_unequippped` to `true`, because we the weapon has successfully been unequipped.

If the current weapon is `null`, then we can simply set `weapon_unequipped` to `true`. The reason we do this check is because there is no weapon script/node for `UNARMED`, but there is also no animations for `UNARMED`, so we can just start equipping the weapon we want to change to.

If we have successfully unequipped the current weapon (`weapon_unequipped == true`), we need to equip the new weapon.

First we define a new variable (`weapon_equipped`) for tracking whether we have successfully equipped the new weapon or not.

Then we get the weapon we want to change to. If the weapon we want to change to is not `null`, we then check to see whether or not it's enabled. If it is not enabled, we call it's `equip_weapon` function so it starts to equip the weapon. If the weapon is enabled, we set `weapon_equipped` to `true`.

If the weapon we want to change to is `null`, we simply set `weapon_equipped` to `true` because we do not have any node/script for `UNARMED`, nor do we have any animations.

Finally, we check to see if we have successfully equipped the new weapon. If we have, we set `changing_weapon` to false because we are no longer changing weapons. We also set `current_weapon_name` to `changing_weapon_name`, since the current weapon has changed, and then we set `changing_weapon_name` to a empty string.

Now, we need to add one more function to the player, and then the player is ready to start the weapons!

We need to add `fire_bullet`, which will be called when by the *AnimationPlayer* at those points we set earlier in the *AnimationPlayer* function track:

```
func fire_bullet():
    if changing_weapon == true:
        return

    weapons[current_weapon_name].fire_weapon()
```

Lets go over what this function is doing:

First we check if we are changing weapons or not. If we are changing weapons, we do not want shoot so we `return`.

Tip: Calling `return` stops the rest of the function from being called. In this case, we are not returning a variable because we are only interested in not running the rest of the code, and because we are not looking for a returned variable either when we call this function.

Then we tell the current weapon we are using to fire by calling its `fire_weapon` function.

Tip: Remember how we mentioned the speed of the animations for firing was faster than the other animations? By changing the firing animation speeds, you can change how fast the weapon fires bullets!

Before we are ready to test our new weapons, we still have a little bit of work to do.

Creating some test subjects

Create a new script by going to the scripting window, clicking "file", and selecting new. Name this script `RigidBody_hit_test` and make sure it extends *RigidBody*.

Now we need to add this code:

```
extends RigidBody

func _ready():
    pass

func bullet_hit(damage, bullet_hit_pos):
    var direction_vect = global_transform.origin - bullet_hit_pos
    direction_vect = direction_vect.normalized()

    apply_impulse(bullet_hit_pos, direction_vect * damage)
```

Lets go over how `bullet_hit` works:

First we get the direction from the bullet pointing towards our global *Transform*. We do this by subtracting the bullet's hit position from the *RigidBody*'s position. This results in a *Vector3* that we can use to tell the direction the bullet collided into the *RigidBody* at.

We then normalize it so we do not get crazy results from collisions on the extremes of the collision shape attached to the *RigidBody*. Without normalizing shots farther away from the center of the *RigidBody* would cause a more noticeable reaction than those closer to the center.

Finally, we apply an impulse at the passed in bullet collision position. With the force being the directional vector times the damage the bullet is supposed to cause. This makes the *RigidBody* seem to move in response to the bullet colliding into it.

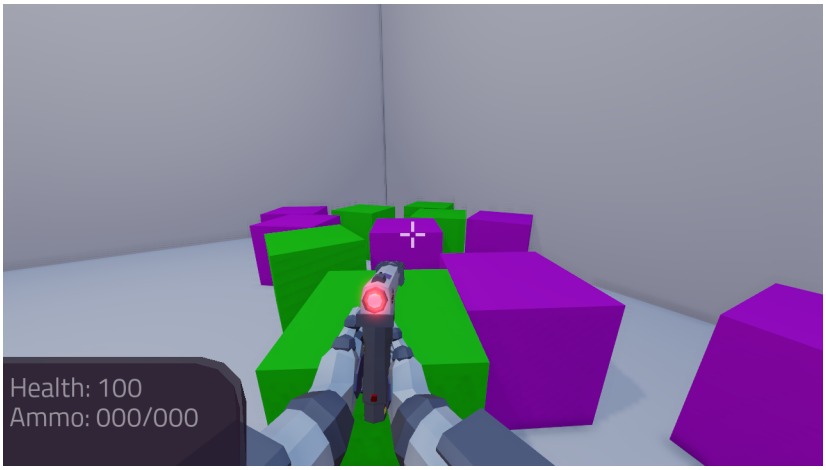
Now we need to attach this script to all of the *RigidBody* nodes we want to effect.

Open up `Testing_Area.tscn` and select all of the cubes parented to the `Cubes` node.

Tip: If you select the top cube, and then hold down `shift` and select the last cube, Godot will select all of the cubes in between!

Once you have all of the cubes selected, scroll down in the inspector until you get to the the "scripts" section. Click the drop down and select "Load". Open your newly created `RigidBody_hit_test.gd` script.

Final notes



That was a lot of code! But now with all that done you can go give your weapons a test!

You should now be able to fire as many bullets as you want on the cubes and they will move in response to the bullets colliding into them.

In *Part 3*, we will add ammo to the weapons, as well as some sounds!

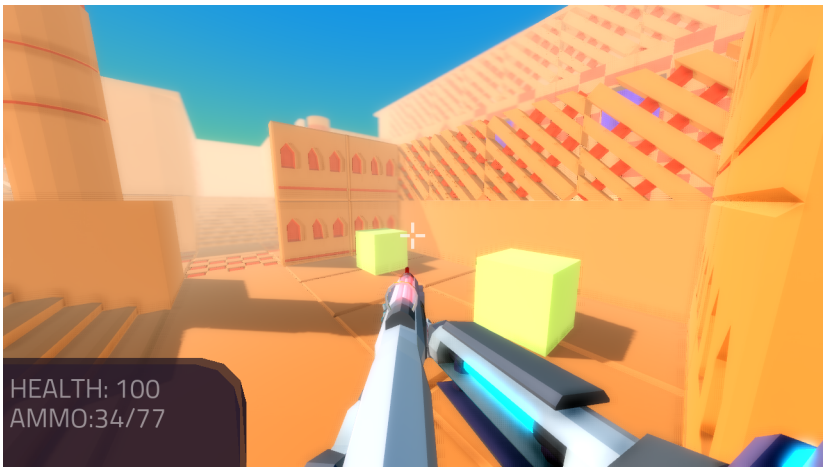
Warning: If you ever get lost, be sure to read over the code again!

You can download the finished project for this part here: `Godot_FPS_Part_2.zip`

7.16.3 Part 3

Part Overview

In this part we will be limiting our weapons by giving them ammo. We will also be giving the player the ability to reload, and we will be adding sounds when the weapons fire.



By the end of this part, the player will have limited ammo, the ability to reload, and sounds will play when the player fires and changes weapons.

Note: You are assumed to have finished *Part 2* before moving on to this part of the tutorial.

The finished project from *Part 2* will be the starting project for part 3

Let's get started!

Changing levels

Now that we have a fully working FPS, let's move to a more FPS like level.

Open up `Space_Level.tscn` (`assets/Space_Level_Objects/Space_Level.tscn`) and/or `Ruins_Level.tscn` (`assets/Ruin_Level_Objects/Ruins_Level.tscn`).

`Space_Level.tscn` and `Ruins_Level.tscn` are complete custom FPS levels created for the purpose of this tutorial. Press `F6` to play the open scene, or press the `play current scene button`, and give it them a whirl.

Warning: `Space_Level.tscn` is more graphically demanding of the GPU than `Ruins_Level.tscn`. If your computer is struggling to render `Space_Level.tscn`, try using `Ruins_Level.tscn` instead.

You might have noticed there are several *RigidBody* nodes placed throughout the level. We can place `RigidBody_hit_test.gd` on them and then they will react to being hit with bullets, so let's do that!

Follow the instructions below for either (or both) of the scenes you want to use

Space_Level

Ruins_Level

```
Expand "Other_Objects" and then expand "Turrets_And_Physics_Objects".

Expand one of the "Barrel_Group" nodes and then select "Barrel_Rigid_Body" and open␣
↪it using
the "Open in Editor" button.
This will bring you to the "Barrel_Rigid_Body" scene. From there select the root node␣
↪and
scroll the inspector down to the bottom.
Select the drop down arrow under the "Node" tab, and then select "Load". Navigate to
"RigidBody_hit_test.gd" and select "Open".

Return back to "Space_Level.tscn".

Expand one of the "Box_Group" nodes and then select "Box_Rigid_Body" and open it␣
↪using the
"Open in Editor" button.
This will bring you to the "Box_Rigid_Body" scene. From there select the root node and
scroll the inspector down to the bottom.
Select the drop down arrow under the "Node" tab, and then select "Load". Navigate to
"RigidBody_hit_test.gd" and select "Open".

Return back to "Space_Level.tscn".
```

```
Expand "Misc_Objects" and then expand "Physics_Objects".

Select all of the "Stone_Cube" RigidBodies and then in the inspector scroll down to␣
↪the bottom.
Select the drop down arrow under the "Node" tab, and then select "Load". Navigate to
"RigidBody_hit_test.gd" and select "Open".

Return back to "Ruins_Level.tscn".
```

Now you can fire at all of the rigid bodies in either level!

Adding ammo

Now that we've got working guns, let's give them a limited amount of ammo.

First we need to define a few variables in each of our weapon scripts.

Open up `Weapon_Pistol.gd` and add the following global variables:

```
var ammo_in_weapon = 10
var spare_ammo = 20
const AMMO_IN_MAG = 10
```

- `ammo_in_weapon`: The amount of ammo currently in the pistol
- `spare_ammo`: The amount of ammo we have left in reserve for the pistol
- `AMMO_IN_MAG`: The amount of ammo in a fully reload weapon/magazine

Now all we need to do is add a single line of code to `fire_weapon`.

Add the following right under `Clone.BULLET_DAMAGE = DAMAGE`: `ammo_in_weapon -= 1`

This will remove one from `ammo_in_weapon` every time we fire. Notice we're not checking to see if we have ammo count of `0` or greater in `fire_weapon`. Instead we're going to check that the ammo count in `Player.gd`.

Now we need to add ammo for both the rifle and the knife.

Note: You may be wondering why we are adding ammo for the knife given it does not consume any ammunition. The reason we want to add ammo to the knife is so we have a consistent interface for all of our weapons.

If we did not add ammo variables for the knife, we would have to add checks for the knife. By adding the ammo variables to the knife, we don't need to worry about whether or all our weapons have the same variables.

Add the following global variables to `Weapon_Rifle.gd`:

```
var ammo_in_weapon = 50
var spare_ammo = 100
const AMMO_IN_MAG = 50
```

And then add the following to `fire_weapon`: `ammo_in_weapon -= 1`. Make sure that `ammo_in_weapon -= 1` is outside of the `if ray.is_colliding()` check so we lost ammo regardless of whether we've hit something or not.

Now all that's left is the knife. Add the following to `Weapon_Knife.gd`:

```
var ammo_in_weapon = 1
var spare_ammo = 1
const AMMO_IN_MAG = 1
```

And because our knife does not consume ammo, that is all we need to add.

Now all we need to do is change a one thing in `Player.gd`.

All we need to change how we're firing our weapons in `process_input`. Change the code for firing weapons to the following:

```
# ----------------------------------
# Firing the weapons
if Input.is_action_pressed("fire"):
    if changing_weapon == false:
        var current_weapon = weapons[current_weapon_name]
        if current_weapon != null:
            if current_weapon.ammo_in_weapon > 0:
                if animation_manager.current_state == current_weapon.IDLE_ANIM_NAME:
                    animation_manager.set_animation(current_weapon.FIRE_ANIM_NAME)
# ----------------------------------
```

Now our weapons have a limited amount of ammo, and will stop firing when we run out.

Ideally we'd like to be able to see how much ammo we have left. Let's make a new function called `process_ui`.

First, add `process_UI(delta)` to `_physics_process`.

Now add the following to `Player.gd`:

```
func process_UI(delta):
    if current_weapon_name == "UNARMED" or current_weapon_name == "KNIFE":
        UI_status_label.text = "HEALTH: " + str(health)
    else:
        var current_weapon = weapons[current_weapon_name]
```

(continues on next page)

(continued from previous page)

```
        UI_status_label.text = "HEALTH: " + str(health) + \
        "\nAMMO:" + str(current_weapon.ammo_in_weapon) + "/" + str(current_weapon.
↪spare_ammo)
```

Let's go over what's happening:

First we check to see if the current weapon is either `UNARMED` or `KNIFE`. If it is, we change the `UI_status_label`'s text to only show our health, since `UNARMED` and `KNIFE` do not consume ammo.

If we are using a weapon that does consume ammo, we first get the weapon node.

Then change `UI_status_label`'s text to show our health, how much ammo we have in the weapon, along with how much spare ammo we have for that weapon.

Now we can see how much ammo we have through the HUD.

Adding reloading to the weapons

Now that we can run our weapons out of ammo, we need a way to fill them back up. Let's add reloading next!

For reloading we need to add a few more variables and a function to every weapon.

Open up `Weapon_Pistol.gd` and add the following global variables:

```
const CAN_RELOAD = true
const CAN_REFILL = true

const RELOADING_ANIM_NAME = "Pistol_reload"
```

- `CAN_RELOAD`: A boolean to track whether this weapon has the ability to reload
- `CAN_REFIL`: A boolean to track whether we can refill this weapon's spare ammo. We will not be using `CAN_REFIL` in this part, but we will in the next part!
- `RELOADING_ANIM_NAME`: The name of the reloading animation for this weapon.

Now we need to add a function for handling reloading. Add the following function to `Weapon_Pistol.gd`:

```
func reload_weapon():
    var can_reload = false

    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        can_reload = true

    if spare_ammo <= 0 or ammo_in_weapon == AMMO_IN_MAG:
        can_reload = false

    if can_reload == true:
        var ammo_needed = AMMO_IN_MAG - ammo_in_weapon

        if spare_ammo >= ammo_needed:
            spare_ammo -= ammo_needed
            ammo_in_weapon = AMMO_IN_MAG
        else:
            ammo_in_weapon += spare_ammo
            spare_ammo = 0

        player_node.animation_manager.set_animation(RELOADING_ANIM_NAME)
```

(continues on next page)

(continued from previous page)

```
        return true

    return false
```

Let's go over what's happening:

First we define a variable to see whether or not we can reload.

We first check to see if we are in this weapon's idle animation state because we only want to be able to reload when we are not firing. equipping, or unequipping.

Next we check to see if we have spare ammo, and if the ammo already in our weapon is equal to a fully reloaded weapon. This way we can assure we're not going to reload when we have no ammo or when the weapon is already full of ammo.

If we still can reload, then we calculate the amount of ammo needed to reload the weapon.

If we have enough ammo to fill the weapon, we remove the ammo needed from `spare_ammo` and then set `ammo_in_weapon` to a full weapon/magazine.

If we do not have enough ammo, we add all of the ammo left in `spare_ammo`, then set `spare_ammo` to `0`.

Next we play the reloading animation for this weapon, and then return `true`.

If we could not reload, we return `false`.

Now we need to add reloading to the rifle. Open up `Weapon_Rifle.gd` and add the following global variables:

```
const CAN_RELOAD = true
const CAN_REFILL = true

const RELOADING_ANIM_NAME = "Rifle_reload"
```

These variables are exactly the same as the pistol, just with `RELOADING_ANIM_NAME` changed to the rifle's reloading animation.

Now we need to add `reload_weapon` to `Weapon_Rifle.gd`:

```
func reload_weapon():
    var can_reload = false

    if player_node.animation_manager.current_state == IDLE_ANIM_NAME:
        can_reload = true

    if spare_ammo <= 0 or ammo_in_weapon == AMMO_IN_MAG:
        can_reload = false

    if can_reload == true:
        var ammo_needed = AMMO_IN_MAG - ammo_in_weapon

        if spare_ammo >= ammo_needed:
            spare_ammo -= ammo_needed
            ammo_in_weapon = AMMO_IN_MAG
        else:
            ammo_in_weapon += spare_ammo
            spare_ammo = 0
```

(continues on next page)

(continued from previous page)

```
        player_node.animation_manager.set_animation(RELOADING_ANIM_NAME)

        return true

    return false
```

This code is exactly the same as the pistol.

The last bit we need to do for the weapons is add 'reloading' to the knife. Add the following global variables to `Weapon_Knife.gd`:

```
const CAN_RELOAD = false
const CAN_REFILL = false

const RELOADING_ANIM_NAME = ""
```

Since we both cannot reload or refill a knife, we set both constants to `false`. We also define `RELOADING_ANIM_NAME` as an empty string, since the knife has no reloading animation.

Now we need to add `reloading_weapon`:

```
func reload_weapon():
    return false
```

Since we cannot reload a knife, we always return `false`.

Adding reloading to the player

Now we need to add a few things to `Player.gd`. First we need to define a new global variable:

```
var reloading_weapon = false
```

- `reloading_weapon`: A variable to track whether or not we are currently trying to reload.

Next we need to add another function call to `_physics_process`.

Add `process_reloading(delta)` to `_physics_process`. Now `_physics_process` should look something like this:

```
func _physics_process(delta):
    process_input(delta)
    process_movement(delta)
    process_changing_weapons(delta)
    process_reloading(delta)
    process_UI(delta)
```

Now we need to add `process_reloading`. Add the following function to `Player.gd`:

```
func process_reloading(delta):
    if reloading_weapon == true:
        var current_weapon = weapons[current_weapon_name]
        if current_weapon != null:
            current_weapon.reload_weapon()
        reloading_weapon = false
```

Let's go over what's happening here.

First we check to make sure we are trying to reload.

If we are, we then get the current weapon. If the current weapon does not equal `null`, we call its `reload_weapon` function.

Note: If the current weapon is equal to `null`, then the current weapon is `UNARMED`.

Finally, we set `reloading_weapon` to `false`, because regardless of whether we've successfully reloaded, we've tried reloading and no longer need to keep trying.

Before we can reload, we need to change a few things in `process_input`.

The first thing we need to change is in the code for changing weapons. We need to add a additional check (`if reloading_weapon == false:`) to see if we are reloading:

```
if changing_weapon == false:
    # New line of code here!
    if reloading_weapon == false:
        if WEAPON_NUMBER_TO_NAME[weapon_change_number] != current_weapon_name:
            changing_weapon_name = WEAPON_NUMBER_TO_NAME[weapon_change_number]
            changing_weapon = true
```

This makes it where we cannot change weapons if we are reloading.

Now we need to add the code to trigger a reload when the player pushes the `reload` action. Add the following code to `process_input`:

```
# ---------------------------------
# Reloading
if reloading_weapon == false:
    if changing_weapon == false:
        if Input.is_action_just_pressed("reload"):
            var current_weapon = weapons[current_weapon_name]
            if current_weapon != null:
                if current_weapon.CAN_RELOAD == true:
                    var current_anim_state = animation_manager.current_state
                    var is_reloading = false
                    for weapon in weapons:
                        var weapon_node = weapons[weapon]
                        if weapon_node != null:
                            if current_anim_state == weapon_node.RELOADING_ANIM_NAME:
                                is_reloading = true
                    if is_reloading == false:
                        reloading_weapon = true
# ---------------------------------
```

Let's go over what's happening here.

First we make sure we're not reloading already, nor are we trying to change weapons.

Then we check to see if the `reload` action has been pressed.

If we have pressed `reload`, we then get the current weapon and check to make sure it is not `null`. Then we check to see if the weapon can reload or not using its `CAN_RELOAD` constant.

If the weapon can reload, we then get the current animation state, and make a variable for tracking whether we are already reloading or not.

We then go through every weapon to make sure we're not already playing that weapon's reloading animation.

If we are not reloading with any weapon, we set `reloading_weapon` to `true`.

One thing I like to add is where the weapon will reload itself if you try to fire it when it's out of ammo.

We also need to add a additional if check (`is_reloading_weapon == false:`) so we cannot fire the current weapon while reloading.

Let's change our firing code in `process_input` so it reloads when trying to fire an empty weapon:

```
# ----------------------------------
# Firing the weapons
if Input.is_action_pressed("fire"):
    if reloading_weapon == false:
        if changing_weapon == false:
            var current_weapon = weapons[current_weapon_name]
            if current_weapon != null:
                if current_weapon.ammo_in_weapon > 0:
                    if animation_manager.current_state == current_weapon.IDLE_ANIM_
↪NAME:
                        animation_manager.set_animation(current_weapon.FIRE_ANIM_NAME)
                else:
                    reloading_weapon = true
# ----------------------------------
```

Now we check to make sure we're not reloading before we fire out weapon, and when we have `0` or less ammo in our weapon we set `reloading_weapon` to `true` if we try to fire.

This will make it where we will try to reload when we try to fire a empty weapon.

With that we can reload our weapons! Give it a try! Now you can fire all of the spare ammo for each weapon.

Adding sounds

Finally, let's add some sounds that play when we are reloading, changing guns, and when we are firing them.

Tip: There are no game sounds provided in this tutorial (for legal reasons). https://gamesounds.xyz/ is a collection of **"royalty free or public domain music and sounds suitable for games"**. I used Gamemaster's Gun Sound Pack, which can be found in the Sonniss.com GDC 2017 Game Audio Bundle.

Open up `SimpleAudioPlayer.tscn`. It is simply a *Spatial* with a *AudioStreamPlayer* as it's child.

Note: The reason this is called a 'simple' audio player is because we are not taking performance into account and because the code is designed to provide sound in the simplest way possible.

If you want to use 3D audio, so it sounds like it's coming from a location in 3D space, right click the *AudioStreamPlayer* and select "Change type".

This will open the node browser. Navigate to *AudioStreamPlayer3D* and select "change". In the source for this tutorial, we will be using *AudioStreamPlayer*, but you can optionally use *AudioStreamPlayer3D* if you desire, and the code provided below will work regardless of which one you chose.

Create a new script and call it `SimpleAudioPlayer.gd`. Attach it to the *Spatial* in `SimpleAudioPlayer.tscn` and insert the following code:

```
extends Spatial

# All of the audio files.
# You will need to provide your own sound files.
var audio_pistol_shot = preload("res://path_to_your_audio_here")
var audio_gun_cock = preload("res://path_to_your_audio_here")
var audio_rifle_shot = preload("res://path_to_your_audio_here")

var audio_node = null

func _ready():
    audio_node = $Audio_Stream_Player
    audio_node.connect("finished", self, "destroy_self")
    audio_node.stop()


func play_sound(sound_name, position=null):

    if audio_pistol_shot == null or audio_rifle_shot == null or audio_gun_cock ==␣
↪null:
        print ("Audio not set!")
        queue_free()
        return

    if sound_name == "Pistol_shot":
        audio_node.stream = audio_pistol_shot
    elif sound_name == "Rifle_shot":
        audio_node.stream = audio_rifle_shot
    elif sound_name == "Gun_cock":
        audio_node.stream = audio_gun_cock
    else:
        print ("UNKNOWN STREAM")
        queue_free()
        return

    # If you are using a AudioPlayer3D, then uncomment these lines to set the␣
↪position.
    # if position != null:
    #       audio_node.global_transform.origin = position

    audio_node.play()


func destroy_self():
    audio_node.stop()
    queue_free()
```

Tip: By setting `position` to `null` by default in `play_sound`, we are making it an optional argument, meaning `position` doesn't necessarily have to be passed in to call the `play_sound`.

Let's go over what's happening here:

In `_ready` we get the *AudioStreamPlayer* and connect its `finished` signal to ourselves. It doesn't matter if it's

a *AudioStreamPlayer* or *AudioStreamPlayer3D* node, as they both have the finished signal. To make sure it is not playing any sounds, we call `stop` on the *AudioStreamPlayer*.

> **Warning:** Make sure your sound files are **not** set to loop! If it is set to loop the sounds will continue to play infinitely and the script will not work!

The `play_sound` function is what we will be calling from `Player.gd`. We check if the sound is one of the three possible sounds, and if it is we set the audio stream for our *AudioStreamPlayer* to the correct sound.

If it is an unknown sound, we print an error message to the console and free ourselves.

If you are using a *AudioStreamPlayer3D*, remove the `#` to set the position of the audio player node so it plays at the correct position.

Finally, we tell the *AudioStreamPlayer* to play.

When the *AudioStreamPlayer* is finished playing the sound, it will call `destroy_self` because we connected the `finished` signal in `_ready`. We stop the *AudioStreamPlayer* and free ourself to save on resources.

Note: This system is extremely simple and has some major flaws:

One flaw is we have to pass in a string value to play a sound. While it is relatively simple to remember the names of the three sounds, it can be increasingly complex when you have more sounds. Ideally we'd place these sounds in some sort of container with exposed variables so we do not have to remember the name(s) of each sound effect we want to play.

Another flaw is we cannot play looping sounds effects, nor background music easily with this system. Because we cannot play looping sounds, certain effects like footstep sounds are harder to accomplish because we then have to keep track of whether or not there is a sound effect and whether or not we need to continue playing it.

One of the biggest flaws with this system is we can only play sounds from `Player.gd`. Ideally we'd like to be able to play sounds from any script at any time.

With that done, let's open up `Player.gd` again. First we need to load the `SimpleAudioPlayer.tscn`. Place the following code in your global variables:

```
var simple_audio_player = preload("res://Simple_Audio_Player.tscn")
```

Now we need to instance the simple audio player when we need it, and then call its `play_sound` function and pass the name of the sound we want to play. To make the process simpler, let's create a `create_sound` function:

```
func create_sound(sound_name, position=null):
    var audio_clone = simple_audio_player.instance()
    var scene_root = get_tree().root.get_children()[0]
    scene_root.add_child(audio_clone)
    audio_clone.play_sound(sound_name, position)
```

Lets walk through what this function does:

The first line instances the `Simple_Audio_Player.tscn` scene and assigns it to a variable, named `audio_clone`.

The second line gets the scene root, using one large assumption. We first get this node's *SceneTree*, and then access the root node, which in this case is the *Viewport* this entire game is running under. Then we get the first child of the

Viewport, which in our case happens to be the root node in `Test_Area.tscn` or any of the other provided levels. We are making a huge assumption that the first child of the root is the root node that our player is under, which could not always be the case.

If this doesn't make sense to you, don't worry too much about it. The second line of code only doesn't work reliably if you have multiple scenes loaded as children to the root node at a time, which will rarely happen for most projects. This is only potentially a issue depending on how you handle scene loading.

The third line adds our newly created `SimpleAudioPlayer` scene to be a child of the scene root. This works exactly the same as when we are spawning bullets.

Finally, we call the `play_sound` function and pass in the arguments we're given. This will call `SimpleAudioPlayer.gd`'s `play_sound` function with the passed in arguments.

Now all that is left is playing the sounds when we want to. Let's add sound to the pistol first!

Open up `Weapon_Pistol.gd`.

Now, we want to make a noise when we fire the pistol, so add the following to the end of the `fire_weapon` function:

```
player_node.create_sound("pistol_shot", self.global_transform.origin)
```

Now when we fire our pistol, we'll play the `pistol_shot` sound.

To make a sound when we reload, we need to add the following right under `player_node.animation_manager.set_animation(RELOADING_ANIM_NAME)` in the `reload_weapon` function:

```
player_node.create_sound("gun_cock", player_node.camera.global_transform.origin)
```

Now when we reload we'll play the `gun_cock` sound.

Now let's add sounds to the rifle. Open up `Weapon_Rifle.gd`.

To play sounds when the rifle is fired, add the following to the end of the `fire_weapon` function:

```
player_node.create_sound("rifle_shot", ray.global_transform.origin)
```

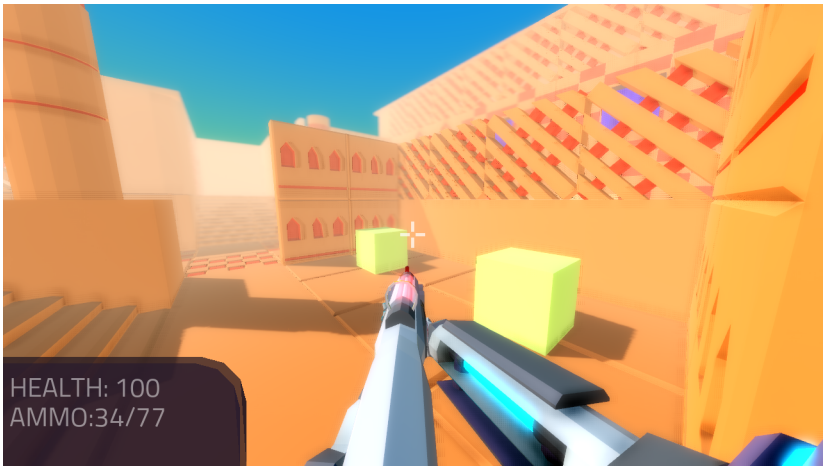
Now when we fire our rifle, we'll play the `rifle_shot` sound.

To make a sound when we reload, we need to add the following right under `player_node.animation_manager.set_animation(RELOADING_ANIM_NAME)` in the `reload_weapon` function:

```
player_node.create_sound("gun_cock", player_node.camera.global_transform.origin)
```

Now when we reload we'll play the `gun_cock` sound.

Final notes



Now you have weapons with limited ammo that play sounds when you fire them!

At this point we have all of the basics of a FPS working. There's still a few things that would be nice to add, and we're going to add them in the next three parts!

For example, right now we have no way to add ammo to our spares, so we'll eventually run out. Also, we don't have anything to shoot at outside of the *RigidBody* nodes.

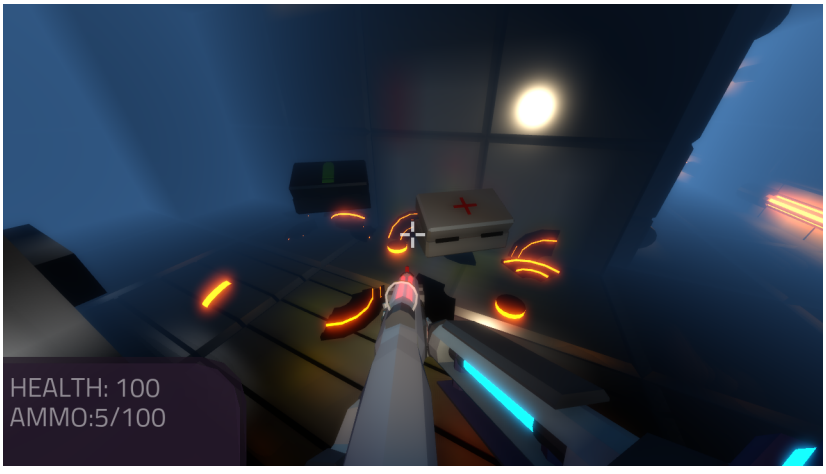
In In *Part 4* we'll add some targets to shoot at, along with some health and ammo pick ups! We're also going to add joypad support, so we can play with wired Xbox 360 controllers!

> **Warning:** If you ever get lost, be sure to read over the code again!
>
> You can download the finished project for this part here: `Godot_FPS_Part_3.zip`

7.16.4 Part 4

Part Overview

In this part we will be adding health pick ups, ammo pick ups, targets we can destroy, add support for joypads, and add the ability to change weapons with the scroll wheel.



Note: You are assumed to have finished *Part 3* before moving on to this part of the tutorial.

The finished project from *Part 3* will be the starting project for part 4

Let's get started!

Adding joypad input

Note: In Godot any game controller is referred to as a joypad. This includes: Console controllers, Joysticks (like for flight simulators), Wheels (like for driving simulators), VR Controllers, and more.

First we need to change a few things in our project's input map. Open up the project settings and select the `Input Map` tab.

Now we need to add some joypad buttons to our various actions. Click the plus icon and select `Joy Button`.

Feel free to use whatever button layout you want. Make sure that the device selected is set to `0`. In the finished project, we will be using the following:

- movement_sprint: `Device 0, Button 4 (L, L1)`
- fire: `Device 0, Button 0 (PS Cross, XBox A, Nintendo B)`
- reload: `Device 0, Button 0 (PS Square, XBox X, Nintendo Y)`
- flashlight: `Device 0, Button 12 (D-Pad Up)`
- shift_weapon_positive: `Device 0, Button 15 (D-Pad Right)`
- shift_weapon_negative: `Device 0, Button 14 (D-Pad Right)`
- fire_grenade: `Device 0, Button 1 (PS Circle, XBox B, Nintendo A).`

Note: These are already set up for you if you downloaded the starter assets

Once you are happy with the input, close the project settings and save.

Now let's open up `Player.gd` and add joypad input.

First, we need to define a few new global variables. Add the following global variables to `Player.gd`:

```
# You may need to adjust depending on the sensitivity of your joypad
var JOYPAD_SENSITIVITY = 2
const JOYPAD_DEADZONE = 0.15
```

Let's go over what each of these do:

- `JOYPAD_SENSITIVITY`: This is how fast our joypad joysticks will move our camera.
- `JOYPAD_DEADZONE`: The dead zone for the joypad. You may need to adjust depending on your joypad.

Note: Many joypads jitter around a certain point. To counter this, we ignore any movement in a with a radius of JOYPAD_DEADZONE. If we did not ignore said movement, the camera will jitter.

Also, we are defining `JOYPAD_SENSITIVITY` as a variable instead of a constant because we'll later be changing it.

Now we are ready to start handling joypad input!

In `process_input` add the following code, just before `input_movement_vector = input_movement_vector.normalized()`:

```
# Add joypad input, if there is a joypad
if Input.get_connected_joypads().size() > 0:

    var joypad_vec = Vector2(0, 0)

    if OS.get_name() == "Windows":
        joypad_vec = Vector2(Input.get_joy_axis(0, 0), -Input.get_joy_axis(0, 1))
    elif OS.get_name() == "X11":
        joypad_vec = Vector2(Input.get_joy_axis(0, 1), Input.get_joy_axis(0, 2))
    elif OS.get_name() == "OSX":
        joypad_vec = Vector2(Input.get_joy_axis(0, 1), Input.get_joy_axis(0, 2))

    if joypad_vec.length() < JOYPAD_DEADZONE:
        joypad_vec = Vector2(0, 0)
    else:
        joypad_vec = joypad_vec.normalized() * ((joypad_vec.length() - JOYPAD_
↪DEADZONE) / (1 - JOYPAD_DEADZONE))

    input_movement_vector += joypad_vec
```

Let's go over what we're doing.

First we check to see if there is a connected joypad.

If there is a joypad connected, we then get its left stick axes for right/left and up/down. Because a wired Xbox 360 controller has different joystick axis mapping based on OS, we use different axes based on the OS.

Warning: This tutorial assumes you are using a XBox 360 wired controller. Also, I do not (currently) have access to a Mac computer, so the joystick axes may need changing. If they do, please open a GitHub issue on the Godot documentation repository!

Next we check to see if the joypad vector length is within the `JOYPAD_DEADZONE` radius. If it is, we set `joypad_vec` to an empty Vector2. If it is not, we use a scaled Radial Dead zone for precise dead zone calculating.

Note: You can find a great article explaining all about how to handle joypad/controller dead zones here: http://www.third-helix.com/2013/04/12/doing-thumbstick-dead-zones-right.html

We're using a translated version of the scaled radial dead zone code provided in that article. The article is a great read, and I highly suggest giving it a look!

Finally, we add `joypad_vec` to `input_movement_vector`.

Tip: Remember how we normalize `input_movement_vector`? This is why! If we did not normalize `input_movement_vector` players could move faster if they are pushing in the same direction with both their keyboard and their joypad!

Make a new function called `process_view_input` and add the following:

```
func process_view_input(delta):

    if Input.get_mouse_mode() != Input.MOUSE_MODE_CAPTURED:
        return

    # NOTE: Until some bugs relating to captured mouses are fixed, we cannot put the␣
↪mouse view
    # rotation code here. Once the bug(s) are fixed, code for mouse view rotation␣
↪code will go here!

    # -----------------------------------
    # Joypad rotation

    var joypad_vec = Vector2()
    if Input.get_connected_joypads().size() > 0:

        if OS.get_name() == "Windows":
            joypad_vec = Vector2(Input.get_joy_axis(0, 2), Input.get_joy_axis(0, 3))
        elif OS.get_name() == "X11":
            joypad_vec = Vector2(Input.get_joy_axis(0, 3), Input.get_joy_axis(0, 4))
        elif OS.get_name() == "OSX":
            joypad_vec = Vector2(Input.get_joy_axis(0, 3), Input.get_joy_axis(0, 4))

        if joypad_vec.length() < JOYPAD_DEADZONE:
            joypad_vec = Vector2(0, 0)
        else:
            joypad_vec = joypad_vec.normalized() * ((joypad_vec.length() - JOYPAD_
↪DEADZONE) / (1 - JOYPAD_DEADZONE))

        rotation_helper.rotate_x(deg2rad(joypad_vec.y * JOYPAD_SENSITIVITY))

        rotate_y(deg2rad(joypad_vec.x * JOYPAD_SENSITIVITY * -1))

        var camera_rot = rotation_helper.rotation_degrees
        camera_rot.x = clamp(camera_rot.x, -70, 70)
        rotation_helper.rotation_degrees = camera_rot
    # -----------------------------------
```

Let's go over what's happening:

First we check the mouse mode. If the mouse mode is not `MOUSE_MODE_CAPTURED`, we want to return, which will skip the code below.

Next we define a new *Vector2* called `joypad_vec`. This will hold the right joystick position. Based on the OS, we set its values so it is mapped to the proper axes for the right joystick.

Warning: As stated above, I do not (currently) has access to a Mac computer, so the joystick axes may need changing. If they do, please open a GitHub issue on the Godot documentation repository!

We then account for the joypad's dead zone, exactly like in `process_input`.

Then we rotate `rotation_helper` and our KinematicBody using `joypad_vec`.

Notice how the code that handles rotating ourselves and `rotation_helper` is exactly the same as the code in `_input`. All we've done is change the values to use `joypad_vec` and `JOYPAD_SENSITIVITY`.

Note: Due to few mouse related bugs on Windows, we cannot put mouse rotation in `process_view` as well. Once these bugs are fixed, this will likely be updated to place the mouse rotation here as well.

Finally, we clamp the camera's rotation so we cannot look upside down.

The last thing you need to do is add `process_view_input` to `_physics_process`.

Once `process_view_input` is added to `_physics_process`, you should be able to play using a joypad!

Note: I decided not to use the joypad triggers for firing because we'd then have to do some more axis managing, and because I prefer to use a shoulder button to fire.

If you want to use the triggers for firing, you will need to change how firing works in `process_input`. You need to get the proper axis value for the trigger, and check if it's over a certain value, say `0.8` for example. If it is, you add the same code as when the `fire` action was pressed.

Adding mouse scroll wheel input

Let's add one input related feature before we start working on the pick ups and target. Let's add the ability to change weapons using the scroll wheel on the mouse.

Open up `Player.gd` and add the following global variables:

```
var mouse_scroll_value = 0
const MOUSE_SENSITIVITY_SCROLL_WHEEL = 0.08
```

Let's go over what each of these new variables will be doing:

- `mouse_scroll_value`: The value of the mouse scroll wheel.
- `MOUSE_SENSITIVITY_SCROLL_WHEEL`: How much a single scroll action increases mouse_scroll_value

---

Now let's add the following to `_input`:

```
if event is InputEventMouseButton and Input.get_mouse_mode() == Input.MOUSE_MODE_
↪CAPTURED:
    if event.button_index == BUTTON_WHEEL_UP or event.button_index == BUTTON_WHEEL_
↪DOWN:
        if event.button_index == BUTTON_WHEEL_UP:
            mouse_scroll_value += MOUSE_SENSITIVITY_SCROLL_WHEEL
        elif event.button_index == BUTTON_WHEEL_DOWN:
            mouse_scroll_value -= MOUSE_SENSITIVITY_SCROLL_WHEEL

        mouse_scroll_value = clamp(mouse_scroll_value, 0, WEAPON_NUMBER_TO_NAME.
↪size()-1)
```

(continues on next page)

(continued from previous page)

```
        if changing_weapon == false:
            if reloading_weapon == false:
                var round_mouse_scroll_value = int(round(mouse_scroll_value))
                if WEAPON_NUMBER_TO_NAME[round_mouse_scroll_value] != current_weapon_
↪name:
                    changing_weapon_name = WEAPON_NUMBER_TO_NAME[round_mouse_scroll_
↪value]
                    changing_weapon = true
                    mouse_scroll_value = round_mouse_scroll_value
```

Let's go over what's happening here:

First we check if the event is a `InputEventMouseButton` event and that our mouse mode is `MOUSE_MODE_CAPTURED`. Then we check to see if the button index is either a `BUTTON_WHEEL_UP` or `BUTTON_WHEEL_DOWN` index.

If the event's index is indeed a button wheel index, we then check to see if it is a `BUTTON_WHEEL_UP` or `BUTTON_WHEEL_DOWN` index. Based on whether it is up or down we add/remove `MOUSE_SENSITIVITY_SCROLL_WHEEL` to/from `mouse_scroll_value`.

Next we clamp mouse scroll value to assure it is inside the range of our weapons.

We then check to see if we are changing weapons or reloading. If we are doing neither, we round `mouse_scroll_value` and cast it to a `int`.

Note: We are casting `mouse_scroll_value` to a `int` so we can use it as a key in our dictionary. If we left it as a float, we would get an error when we try to run the project.

Next we check to see if the weapon name at `round_mouse_scroll_value` is not equal to the current weapon name using `weapon_number_to_name`. If the weapon is different than our current weapon, we assign `changing_weapon_name`, set `changing_weapon` to true so we will change weapons in `process_changing_weapon`, and set `mouse_scroll_value` to `round_mouse_scroll_value`.

Tip: The reason we are setting `mouse_scroll_value` to the rounded scroll value is because we do not want the player to keep their mouse scroll wheel just in between values, giving them the ability to switch almost extremely fast. By assigning `mouse_scroll_value` to `round_mouse_scroll_value`, we assure that each weapon takes exactly the same amount of scrolling to change.

One more thing we need to change is in `process_input`. In the code for changing weapons, add the following right after the line `changing_weapon = true`:

```
mouse_scroll_value = weapon_change_number
```

Now our scroll value we be changed with the keyboard input. If we did not change this, our scroll value will be out of sync. If the scroll wheel is out of sync, scrolling forwards or backwards would not transition to the next/last weapon, but rather the next/last weapon the scroll wheel changed to.

Now you can change weapons using the scroll wheel! Go give it a whirl!

Adding the health pick ups

Now that our player has health and ammo, we ideally need a way to replenish those resources.

Open up `Health_Pickup.tscn`.

Expand `Holder` if it's not already expanded. Notice how we have two Spatial nodes, one called `Health_Kit` and another called `Health_Kit_Small`.

This is because we're actually going to be making two sizes of health pick ups, one small and one large/normal. `Health_Kit` and `Health_Kit_Small` only have a single *MeshInstance* as their children.

Next expand `Health_Pickup_Trigger`. This is an *Area* node we're going to use to check if the player has walked close enough to pick up the health kit. If you expand it you'll find two collision shapes, one for each size. We will be using a different collision shape size based on the size of the health pick up, so the smaller health pick up has a trigger collision shape closer to it's size.

The last thing to note is how we have a *AnimationPlayer* node so the health kit spins around slowly and bobs up and down.

Select `Health_Pickup` and add a new script called `Health_Pickup.gd`. Add the following:

```
extends Spatial

export (int, "full size", "small") var kit_size = 0 setget kit_size_change

# 0 = full size pickup, 1 = small pickup
const HEALTH_AMOUNTS = [70, 30]

const RESPAWN_TIME = 20
var respawn_timer = 0

var is_ready = false

func _ready():

    $Holder/Health_Pickup_Trigger.connect("body_entered", self, "trigger_body_entered
↪")

    is_ready = true

    kit_size_change_values(0, false)
    kit_size_change_values(1, false)
    kit_size_change_values(kit_size, true)


func _physics_process(delta):
    if respawn_timer > 0:
        respawn_timer -= delta

        if respawn_timer <= 0:
            kit_size_change_values(kit_size, true)


func kit_size_change(value):
    if is_ready:
        kit_size_change_values(kit_size, false)
        kit_size = value
        kit_size_change_values(kit_size, true)
```

(continues on next page)

(continued from previous page)

```
    else:
        kit_size = value


func kit_size_change_values(size, enable):
    if size == 0:
        $Holder/Health_Pickup_Trigger/Shape_Kit.disabled = !enable
        $Holder/Health_Kit.visible = enable
    elif size == 1:
        $Holder/Health_Pickup_Trigger/Shape_Kit_Small.disabled = !enable
        $Holder/Health_Kit_Small.visible = enable


func trigger_body_entered(body):
    if body.has_method("add_health"):
        body.add_health(HEALTH_AMOUNTS[kit_size])
        respawn_timer = RESPAWN_TIME
        kit_size_change_values(kit_size, false)
```

Let's go over what this script is doing, starting with its global variables:

- `kit_size`: The size of the health pick up. Notice how we're using a `setget` function to tell if it's changed.
- `HEALTH_AMMOUNTS`: The amount of health each pick up in each size contains.
- `RESPAWN_TIME`: The amount of time, in seconds, it takes for the health pick up to respawn
- `respawn_timer`: A variable used to track how long the health pick up has been waiting to respawn.
- `is_ready`: A variable to track whether the `_ready` function has been called or not.

We're using `is_ready` because `setget` functions are called before `_ready`, we need to ignore the first kit_size_change call, because we cannot access child nodes until `_ready` is called. If we did not ignore the first `setget` call, we would get several errors in the debugger.

Also, notice how we're using a exported variable. This is so we can change the size of the health pick up in the editor, for each pick up. This makes it where we do not have to make two scenes for the two sizes, since we can easily change sizes in the editor using the exported variable.

Tip: See *GDScript* and scroll down to the Exports section for a list of of export hints you can use.

Let's look at `_ready`:

First we connect the `body_entered` signal from our `Health_Pickup_Trigger` to the `trigger_body_entered` function. This makes is where any body that enters the *Area* triggers the `trigger_body_entered` function.

Next we set `is_ready` to `true` so we can use our `setget` function.

Then we hide all of the possible kits and their collision shapes using `kit_size_change_values`. The first argument is the size of the kit, while the second argument is whether to enable or disable the collision shape and mesh at that size.

Then we make only the kit size we selected visible, calling `kit_size_change_values` and passing in `kit_size` and `true`, so the size at `kit_size` is enabled.

Next let's look at `kit_size_changed`.

The first thing we do is check to see if `is_ready` is `true`.

If `is_ready` is `true`, we then make whatever kit is currently assigned to `kit_size` disabled using `kit_size_change_values`, passing in `kit_size` and `false`.

Then we assign `kit_size` to the new value passed in, `value`. Then we call `kit_size_change_values` passing in `kit_size` again, but this time with the second argument as `true` so we enable it. Because we changed `kit_size` to the passed in value, this will make whatever kit size we passed in visible.

If `is_ready` is not `true`, we simply assign `kit_size` to the passed in `value`.

Now let's look at `kit_size_change_values`.

The first thing we do is check to see which size we're using. Based on which size we're wanting to enable/disable, we want to get different nodes.

We get the collision shape for the node corresponding to `size` and disable it based on the `enabled` passed in argument/variable.

Note: Why are we using `!enable` instead of `enable`? This is so when we say we want to enable the node, we can pass in `true`, but since *CollisionShape* uses disabled instead of enabled, we need to flip it. By flipping it, we can enable the collision shape and make the mesh visible when `true` is passed in.

We then get the correct *Spatial* node holding the mesh and set its visibility to `enable`.

This function may be a little confusing, try to think of it like this: We're enabling/disabling the proper nodes for `size` using `enabled`. This is so we cannot pick up health for a size that is not visible, and so only the mesh for the proper size will be visible.

Finally, let's look at `trigger_body_entered`.

The first thing we do is see whether or not the body that just entered has a method/function called `add_health`. If it does, we then call `add_health` and pass in the health provided by the current kit size.

Then we set `respawn_timer` to `RESPAWN_TIME` so we have to wait before we can get health again. Finally, call `kit_size_change_values`, passing in `kit_size` and `false` so the kit at `kit_size` is invisible until we've waited long enough to respawn.

The last thing we need to do before we can use this health pick up is add a few things to our player.

Open up `Player.gd` and add the following global variable:

```
const MAX_HEALTH = 150
```

- `MAX_HEALTH`: The maximum amount of health a player can have.

Now we need to add the `add_health` function to our player. Add the following to `Player.gd`:

```
func add_health(additional_health):
    health += additional_health
    health = clamp(health, 0, MAX_HEALTH)
```

Let's quickly go over what this does.

We first add `additional_health` to our current health. We then clamp the health so that it cannot exceed a value higher than `MAX_HEALTH`, nor a value lower than `0`.

With that done, now we can collect health! Go place a few `Health_Pickup` scenes around and give it a try. You can change the size of the health pick up in the editor when a `Health_Pickup` instanced scene is selected, from a convenient drop down.

Adding the ammo pick ups

While adding health is good and all, we can't reap the rewards from it since nothing can (currently) damage us. Let's add some ammo pick ups next!

Open up `Ammo_Pickup.tscn`. Notice how it's structured exactly the same as `Health_Pickup.tscn`, but with the meshes and trigger collision shapes changed slightly to adjust for the difference in mesh sizes.

Select `Ammo_Pickup` and add a new script called `Ammo_Pickup.gd`. Add the following:

```
extends Spatial

export (int, "full size", "small") var kit_size = 0 setget kit_size_change

# 0 = full size pickup, 1 = small pickup
const AMMO_AMOUNTS = [4, 1]

const RESPAWN_TIME = 20
var respawn_timer = 0

var is_ready = false

func _ready():

    $Holder/Ammo_Pickup_Trigger.connect("body_entered", self, "trigger_body_entered")

    is_ready = true

    kit_size_change_values(0, false)
    kit_size_change_values(1, false)

    kit_size_change_values(kit_size, true)


func _physics_process(delta):
    if respawn_timer > 0:
        respawn_timer -= delta

        if respawn_timer <= 0:
            kit_size_change_values(kit_size, true)


func kit_size_change(value):
    if is_ready:
        kit_size_change_values(kit_size, false)
        kit_size = value
```

(continues on next page)

(continued from previous page)

```
        kit_size_change_values(kit_size, true)
    else:
        kit_size = value


func kit_size_change_values(size, enable):
    if size == 0:
        $Holder/Ammo_Pickup_Trigger/Shape_Kit.disabled = !enable
        $Holder/Ammo_Kit.visible = enable
    elif size == 1:
        $Holder/Ammo_Pickup_Trigger/Shape_Kit_Small.disabled = !enable
        $Holder/Ammo_Kit_Small.visible = enable


func trigger_body_entered(body):
    if body.has_method("add_ammo"):
        body.add_ammo(AMMO_AMOUNTS[kit_size])
        respawn_timer = RESPAWN_TIME
        kit_size_change_values(kit_size, false)
```

You may have noticed this code looks almost exactly the same as the health pick up. That's because it largely is the same! Only a few things have been changed, and that's what we're going to go over.

First, notice how we have `AMMO_AMOUNTS` instead of `HEALTH_AMMOUNTS`. `AMMO_AMOUNTS` will be how many ammo clips/magazines we add to the current weapon. (Unlike `HEALTH_AMMOUNTS` which was how many health points, we instead add an entire clip for the current weapon, instead of the raw ammo amount)

The only other thing to notice is in `trigger_body_entered` we're checking and calling a function called `add_ammo`, not `add_health`.

Other than those two small changes, everything else is exactly the same as the health pickup!

All we need to do make the ammo pick ups work is add a new function to our player. Open `Player.gd` and add the following function:

```
func add_ammo(additional_ammo):
    if (current_weapon_name != "UNARMED"):
        if (weapons[current_weapon_name].CAN_REFILL == true):
            weapons[current_weapon_name].spare_ammo += weapons[current_weapon_name].
↪AMMO_IN_MAG * additional_ammo
```

Let's go over what this function does.

The first thing we check is to see whether we're using `UNARMED` or not. Because `UNARMED` does not have a node/script, we want to make sure we're not using `UNARMED` before trying to get the node/script attached to `current_weapon_name`.

Next we check to see if the current weapon can be refilled. If the current weapon can, we add a full clip/magazine worth of ammo to the weapon by multiplying the current weapon's `AMMO_IN_MAG` variable times however much ammo clips we're adding (`additional_ammo`).

With that done, you should now be able to get additional ammo! Go place some ammo pick ups in one/both/all of the scenes and give it a try!

Note: Notice how we're not limiting the amount of ammo you can carry. To limit the amount of ammo each weapon can carry, you need to add a additional variable to each weapon's script, and then clamp the weapon's `spare_ammo` variable after adding ammo in `add_ammo`.

Adding breakable targets

Before we end this part, let's add some targets.

Open up `Target.tscn` and take a look at the scenes in the scene tree.

First, notice how we're not using a *RigidBody* node, but rather a *StaticBody* node instead. The reason behind this is our non-broken targets will not be moving anywhere, using a *RigidBody* would be more hassle then its worth, since all it has to do is stay still.

Tip: We also save a tiny bit of performance using a *StaticBody* over a *RigidBody*

The other thing to note is we have a node called `Broken_Target_Holder`. This node is going to hold a spawned/instanced scene called `Broken_Target.tscn`. Open up `Broken_Target.tscn`.

Notice how the target is broken up into five pieces, each a *RigidBody* node. We're going to spawn/instance this scene when the target takes too much damage and needs to be destroyed. Then we're going to hide the non-broken target, so it looks like the target shattered rather than a shattered target was spawned/instanced.

While you still have `Broken_Target.tscn` open, attach `RigidBody_hit_test.gd` to all of the *RigidBody* nodes. This will make it where we can shoot at the broken pieces and they will react to the bullets.

Alright, now switch back to `Target.tscn`, select the `Target` *StaticBody* node and created a new script called `Target.gd`.

Add the following code to `Target.gd`:

```
extends StaticBody

const TARGET_HEALTH = 40
var current_health = 40

var broken_target_holder

# The collision shape for the target.
# NOTE: this is for the whole target, not the pieces of the target
var target_collision_shape

const TARGET_RESPAWN_TIME = 14
var target_respawn_timer = 0

export (PackedScene) var destroyed_target

func _ready():
    broken_target_holder = get_parent().get_node("Broken_Target_Holder")
    target_collision_shape = $Collision_Shape


func _physics_process(delta):
    if target_respawn_timer > 0:
        target_respawn_timer -= delta
```

(continues on next page)

(continued from previous page)

```
        if target_respawn_timer <= 0:

            for child in broken_target_holder.get_children():
                child.queue_free()

            target_collision_shape.disabled = false
            visible = true
            current_health = TARGET_HEALTH


func bullet_hit(damage, bullet_hit_pos):
    current_health -= damage

    if current_health <= 0:
        var clone = destroyed_target.instance()
        broken_target_holder.add_child(clone)

        for rigid in clone.get_children():
            if rigid is RigidBody:
                var center_in_rigid_space = broken_target_holder.global_transform.
↪origin - rigid.global_transform.origin
                var direction = (rigid.transform.origin - center_in_rigid_space).
↪normalized()
                # Apply the impulse with some additional force (I find 12 works␣
↪nicely)
                rigid.apply_impulse(center_in_rigid_space, direction * 12 * damage)

        target_respawn_timer = TARGET_RESPAWN_TIME

        target_collision_shape.disabled = true
        visible = false
```

Let's go over what this script does, starting with the global variables:

- `TARGET_HEALTH`: The amount of damage needed to break a fully healed target.
- `current_health`: The amount of health this target currently has.
- `broken_target_holder`: A variable to hold the `Broken_Target_Holder` node so we can use it easily.
- `target_collision_shape`: A variable to hold the *CollisionShape* for the non-broken target.
- `TARGET_RESPAWN_TIME`: The length of time, in seconds, it takes for a target to respawn.
- `target_respawn_timer`: A variable to track how long a target has been broken.
- `destroyed_target`: A *PackedScene* to hold the broken target scene.

Notice how we're using an exported variable (a *PackedScene*) to get the broken target scene instead of using `preload`. By using an exported variable, we can chose the scene from the editor, and when/if we need to use a different scene, it's as easy as selecting a different scene in the editor, we don't need to go to the code to change the scene we're using.

Let's look at `_ready`.

The first thing we do is get the broken target holder and assign it to `broken_target_holder`. Notice how we're using `get_parent().get_node()` here, instead of `$`. If you want to use `$`, then you'd need to change `get_parent().get_node()` to `$"../Broken_Target_Holder"`.

Note: At the time of when this was written, I did not realize you can use `$"../NodeName"` to get the parent nodes using `$`, which is why `get_parent().get_node()` is used instead.

Next we get the collision shape and assign it to `target_collision_shape`. The reason we need to collision shape is because even when the mesh is invisible, the collision shape will still exist in the physics world. This makes it where the player can interact with a non-broken target even though it's invisible, which is not what we want. To get around this, we will disable/enable the collision shape as we make the mesh visible/invisible.

Next let's look at `_physics_process`.

We're only going to be using `_physics_process` for respawning, and so the first thing we do is check to see if `target_respawn_timer` is more than `0`.

If it is, we then remove `delta` from it.

Then we check to see if `target_respawn_timer` is `0` or less. The reason behind this is since we just removed `delta` from `target_respawn_timer`, if it's `0` or less then we've just got here, effectively allowing us to do whatever we need to do when the timer is finished.

In this case, we want to respawn our target.

The first thing we do is remove all children in the broken target holder. We do this by iterating over all of the children in `broken_target_holder` and free them.

Next we enable our collision shape by setting its `disabled` boolean to `false`.

Then we make ourselves, and all of our children nodes, visible.

Finally, we reset `current_health` to `TARGET_HEALTH`.

Finally, let's look at `bullet_hit`.

The first the we do is remove however much damage the bullet does from our health.

Next we check to see if we're at `0` health or lower. If we are, then we've just died and need to spawn a broken target.

We first instance a new destroyed target scene, and assign it to a new variable, `clone`.

Next we add `clone` as a child of our broken target holder.

For an added bonus, we want to make all of the target pieces explode outwards. Do to this, we iterate over all of the children in `clone`.

For each child, we first check to see if it's a *RigidBody* node. If it is, we then calculate the center position of the target relative to the child node. Then we figure out which direction we are relative to the center. Using those calculated variables, we push the child from the calculated center, in the direction away from the center, using the damage of the bullet as the force.

Note: We multiply the damage by `12` so it has a more dramatic effect. You can change this to a higher or lower value depending on how explosive you want your targets to shatter.

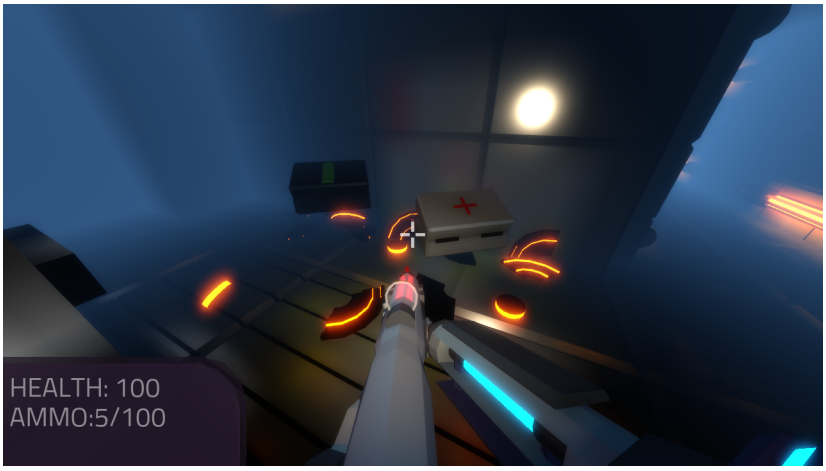
Next we set our respawn timer for our non-broken target. We set it to `TARGET_RESPAWN_TIME`, so it takes `TARGET_RESPAWN_TIME` many seconds to respawn.

Then we disable the non-broken target's collision shape, and set our visibility to `false`.

Warning: Make sure to set the exported `destroyed_target` value for `Target.tscn` in the editor! Otherwise the targets will not be destroyed and you will get an error!

With that done, go place some `Target.tscn` instances around in one/both/all of the levels. You should find they explode into five pieces after they've taken enough damage. After a little while, they'll respawn into a whole target again.

Final notes



Now you can use a joypad, change weapons with the mouse's scroll wheel, replenish your health and ammo, and break targets with your weapons.

In the next part, *Part 5*, we're going to add grenades to our player, give our player the ability to grab and throw objects, and add turrets!

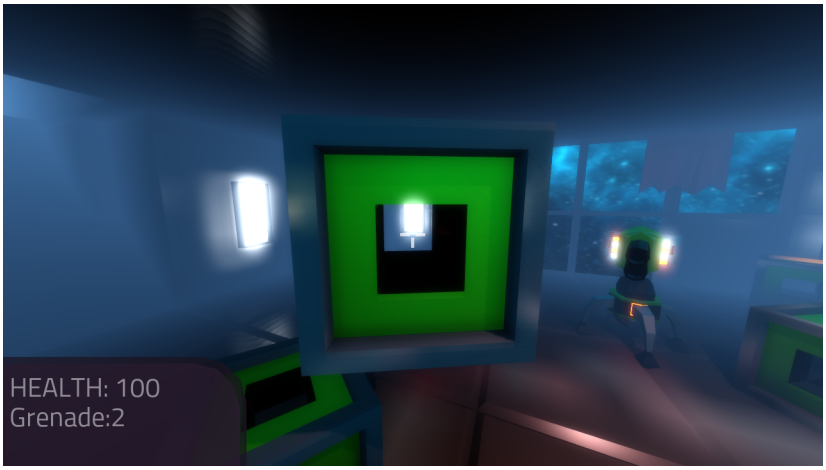
Warning: If you ever get lost, be sure to read over the code again!

You can download the finished project for this part here: `Godot_FPS_Part_4.zip`

7.16.5 Part 5

Part Overview

In this part we're going to add grenades to our player, give our player the ability to grab and throw objects, and add turrets!



Note: You are assumed to have finished *Part 4* before moving on to this part of the tutorial.

The finished project from *Part 4* will be the starting project for part 5

Let's get started!

Adding grenades

First, let's give our player some grenades to play with. Open up `Grenade.tscn`.

There's a few things to note here, the first and foremost being that our grenades are going to use *RigidBody* nodes. We're going to use *RigidBody* nodes for our grenades so they bounce around the world in a somewhat realistic manner.

The second thing to note is `Blast_Area`. This is a *Area* node that will represent the blast radius of our grenade.

Finally, the last thing to note is `Explosion`. This is the *Particles* node that will emit an explosion effect when the grenades explodes. One thing to note here is that we have `One shot` enabled. This is so we emit all of our particles at once. We're also emitting in world coordinates instead of local coordinates, so we have `Local Coords` unchecked as well.

Note: If you want you can see how the particles are set up by looking through it's `Process Material` and `Draw Passes`.

Let's write the code needed for our grenade. Select `Grenade` and make a new script called `Grenade.gd`. Add the following:

```
extends RigidBody

const GRENADE_DAMAGE = 60

const GRENADE_TIME = 2
var grenade_timer = 0

const EXPLOSION_WAIT_TIME = 0.48
var explosion_wait_timer = 0

var rigid_shape
var grenade_mesh
```

(continues on next page)

(continued from previous page)

```
var blast_area
var explosion_particles

func _ready():
    rigid_shape = $Collision_Shape
    grenade_mesh = $Grenade
    blast_area = $Blast_Area
    explosion_particles = $Explosion

    explosion_particles.emitting = false
    explosion_particles.one_shot = true

func _process(delta):

    if grenade_timer < GRENADE_TIME:
        grenade_timer += delta
        return
    else:
        if explosion_wait_timer <= 0:
            explosion_particles.emitting = true

            grenade_mesh.visible = false
            rigid_shape.disabled = true

            mode = RigidBody.MODE_STATIC

            var bodies = blast_area.get_overlapping_bodies()
            for body in bodies:
                if body.has_method("bullet_hit"):
                    body.bullet_hit(GRENADE_DAMAGE, global_transform.origin)

            # This would be the perfect place to play a sound!


        if explosion_wait_timer < EXPLOSION_WAIT_TIME:
            explosion_wait_timer += delta

            if explosion_wait_timer >= EXPLOSION_WAIT_TIME:
                queue_free()
```

Let's go over what's happening, starting with the global variables:

- `GRENADE_DAMAGE`: The amount of damage the grenade causes when it explodes.
- `GRENADE_TIME`: The amount of time the grenade takes (in seconds) to explode once it's created/thrown.
- `grenade_timer`: A variable for tracking how long the grenade has been created/thrown.
- `EXPLOSION_WAIT_TIME`: The amount of time needed (in seconds) to wait before we destroy the grenade scene after the explosion
- `explosion_wait_timer`: A variable for tracking how much time has passed since the grenade exploded.
- `rigid_shape`: The *CollisionShape* for the grenade's *RigidBody*.
- `grenade_mesh`: The *MeshInstance* for the grenade.
- `blast_area`: The blast *Area* used to damage things when the grenade explodes.
- `explosion_particles`: The *Particles* that play when the grenade explodes.

Notice how EXPLOSION_WAIT_TIME is a rather strange number (0.48). This is because we want EXPLOSION_WAIT_TIME to be the length of time the particles are emitting, so when the particles are done we destroy/free the grenade. We calculate EXPLOSION_WAIT_TIME by taking the particle's life time and dividing it by the particle's speed scale. This gets us the exact time the explosion particles will last.

Now let's turn our attention to _ready.

First we get all of the nodes we'll need and assign them to the proper global variables.

We need to get the *CollisionShape* and *MeshInstance* because similarly to the target in *Part 4*, we will be hiding the grenade's mesh and disabling the collision shape when the grenade explodes.

The reason we need to get the blast *Area* is so we can damage everything inside it when it explodes. We'll be using code similar to the knife weapon in our player. We need the *Particles* so we can emit them when we explode.

After we get all of the nodes and assign them to their global variables, we then make sure the explosion particles are not emitting, and that they are set to emit in one shot.

Now let's look at _process.

First we check to see if the grenade_timer is less than GRENADE_TIMER. If it is, we add delta and return. This is so we have to wait GRENADE_TIME seconds, allowing our *RigidBody* to move around.

If grenade_timer is at GRENADE_TIMER or higher, we then need to check if we waited long enough and need to explode. We do this by checking to see if explosion_wait_timer is equal to 0 or less. Since we will be adding delta to explosion_wait_timer right after, whatever code under the check will only be called once, right when we've waited long enough and need to explode.

If we've waited long enough to explode, we first tell the explosion_particles to emit. Then we make grenade_mesh invisible, and disable rigid_shape, effectively hiding our grenade.

We then set the *RigidBody*'s mode to MODE_STATIC so the grenade does not move.

Then we get all of the bodies in blast_area, check to see if they have the bullet_hit method/function, and if they do we call it and pass in GRENADE_DAMAGE and the grenade's position.

We then check to see if explosion_wait_timer is less than EXPLOSION_WAIT_TIME. If it is, we add delta to explosion_wait_time.

Next we check to see if explosion_wait_timer is more than or equal to EXPLOSTION_WAIT_TIME. Because we added delta, this will only be called once. If explosion_wait_timer is more or equal to EXPLOSION_WAIT_TIME, we've waited long enough to let the *Particles* play and can free/destroy ourselves.

Let's quickly get the sticky grenade set up too. Open up Sticky_Grenade.tscn.

Sticky_Grenade.tscn is almost identical to Grenade.tscn, with one small addition. We now have a second *Area*, called Sticky_Area. We'll be using Stick_Area to detect when we've collided with the environment and need to stick to something.

Select Sticky_Grenade and make a new script called Sticky_Grenade.gd. Add the following:

```
extends RigidBody

const GRENADE_DAMAGE = 40

const GRENADE_TIME = 3
var grenade_timer = 0
```

(continues on next page)

(continued from previous page)

```
const EXPLOSION_WAIT_TIME = 0.48
var explosion_wait_timer = 0

var attached = false
var attach_point = null

var rigid_shape
var grenade_mesh
var blast_area
var explosion_particles

var player_body

func _ready():
    rigid_shape = $Collision_Shape
    grenade_mesh = $Sticky_Grenade
    blast_area = $Blast_Area
    explosion_particles = $Explosion

    explosion_particles.emitting = false
    explosion_particles.one_shot = true

    $Sticky_Area.connect("body_entered", self, "collided_with_body")


func collided_with_body(body):

    if body == self:
        return

    if player_body != null:
        if body == player_body:
            return

    if attached == false:
        attached = true
        attach_point = Spatial.new()
        body.add_child(attach_point)
        attach_point.global_transform.origin = global_transform.origin

        rigid_shape.disabled = true

        mode = RigidBody.MODE_STATIC


func _process(delta):

    if attached == true:
        if attach_point != null:
            global_transform.origin = attach_point.global_transform.origin

    if grenade_timer < GRENADE_TIME:
        grenade_timer += delta
        return
    else:
        if explosion_wait_timer <= 0:
```

(continues on next page)

(continued from previous page)

```
            explosion_particles.emitting = true

            grenade_mesh.visible = false
            rigid_shape.disabled = true

            mode = RigidBody.MODE_STATIC

            var bodies = blast_area.get_overlapping_bodies()
            for body in bodies:
                if body.has_method("bullet_hit"):
                    body.bullet_hit(GRENADE_DAMAGE, global_transform.origin)

            # This would be the perfect place to play a sound!


        if explosion_wait_timer < EXPLOSION_WAIT_TIME:
            explosion_wait_timer += delta

            if explosion_wait_timer >= EXPLOSION_WAIT_TIME:
                if attach_point != null:
                    attach_point.queue_free()
                queue_free()
```

The code above is almost identical to the code for `Grenade.gd`, so let's go over what's changed.

First, we have a few more global variables:

- `attached`: A variable for tracking whether or not we've attached to a *PhysicsBody*.
- `attach_point`: A variable to hold a *Spatial* that will be at the position we collided at.
- `player_body`: The player's *KinematicBody*.

These additions are so we can stick to any *PhysicsBody* we happen to hit. We also now need the player's *KinematicBody* so we don't stick to the player that threw this grenade.

Now let's look at the small change in `_ready`. In `_ready` we've added a line of code so when any body enters `Stick_Area`, the `collided_with_body` function is called.

Next let's take a look at `collided_with_body`.

First we make sure we're not colliding with ourself. Because our *Area* does not know it's attached to the grenade's *RigidBody*, we need to make sure we're not going to stick to ourself. If we have collided with ourself, we ignore it by returning.

We then check to see if we have something assigned to `player_body`, and if the body we collided with is the player that threw this grenade. If the body we've collided with is indeed `player_body`, we ignore it by returning.

Next we check if we are attached already or not.

If we are not attached, we then set `attached` to true so we know we've attached to something.

We then make a new *Spatial* node, and make it a child of the body we collided with. We then set the *Spatial*'s position to our current position.

Note: Because we've added the *Spatial* as a child of the body we've collided with, it will follow along with said body. We can then use this *Spatial* to set our position, so we're always at the same position relative to the body we collided with.

We then disable `rigid_shape` so we're not constantly moving whatever body we've collided with. Finally, we set our mode to `MODE_STATIC` so the grenade does not move.

Finally, lets go over the few changes in `_process`.

Now we're checking to see if we are attached right at the top of `_process`.

If we are attached, we then make sure the attached point is not equal to `null`. If the attached point is not equal to `null`, we set our global position (using our global *Transform*'s origin) to the global position of the *Spatial* assigned to `attach_point` (using its global *Transform*'s origin).

The only other change is now before we free/destroy the grenade, we check to see if we have an attached point. If we do, we also call `queue_free` on it, so it's also freed/destroyed.

Adding grenades to the player

Now we need to add some code to `Player.gd` so we can use our grenades.

First, open up `Player.tscn` and expand the node tree until you get to `Rotation_Helper`. Notice how in `Rotation_Helper` we have a node called `Grenade_Toss_Pos`. This is where we will be spawning the grenades.

Also notice how it's slightly rotated on the `X` axis, so it's not pointing straight, but rather slightly up. By changing the rotation of `Grenade_Toss_Pos`, you can change the angle the grenades are tossed at.

Okay, now lets start making the grenades work with our player. Add the following global variables to `Player.gd`:

```
var grenade_amounts = {"Grenade":2, "Sticky Grenade":2}
var current_grenade = "Grenade"
var grenade_scene = preload("res://Grenade.tscn")
var sticky_grenade_scene = preload("res://Sticky_Grenade.tscn")
const GRENADE_THROW_FORCE = 50
```

- `grenade_amounts`: The amount of grenades we are currently carrying for each type of grenade.
- `current_grenade`: The name of the grenade type we're currently using.
- `grenade_scene`: The grenade scene we worked on earlier.
- `sticky_grenade_scene`: The sticky grenade scene we worked on earlier.
- `GRENADE_THROW_FORCE`: The force at which we throw the grenade at.

Most of these variables are similar to how we have out weapons set up.

Tip: While it's possible to make a more modular grenade system, I found it was not worth the additional complexity for just two grenades. If you were going to make a more complex FPS with more grenades, you'd likely want to make a system for grenades similar to how we have the weapons set up.

Now we need to add some code in `_process_input` Add the following to `_process_input`:

```
# ---------------------------------
# Changing and throwing grenades

if Input.is_action_just_pressed("change_grenade"):
    if current_grenade == "Grenade":
        current_grenade = "Sticky Grenade"
    elif current_grenade == "Sticky Grenade":
        current_grenade = "Grenade"

if Input.is_action_just_pressed("fire_grenade"):
    if grenade_amounts[current_grenade] > 0:
        grenade_amounts[current_grenade] -= 1

        var grenade_clone
        if (current_grenade == "Grenade"):
            grenade_clone = grenade_scene.instance()
        elif (current_grenade == "Sticky Grenade"):
            grenade_clone = sticky_grenade_scene.instance()
            # Sticky grenades will stick to the player if we do not pass ourselves
            grenade_clone.player_body = self

        get_tree().root.add_child(grenade_clone)
        grenade_clone.global_transform = $Rotation_Helper/Grenade_Toss_Pos.global_
↪transform
        grenade_clone.apply_impulse(Vector3(0,0,0), grenade_clone.global_transform.
↪basis.z * GRENADE_THROW_FORCE)
# ---------------------------------
```

Let's go over what's happening here.

First, we check to see if the `change_grenade` action has just been pressed. If it has, we then check to see which grenade we are currently using. Based on the name of the grenade we're currently using, we change `current_grenade` to the opposite grenade name.

Next we check to see if the `fire_grenade` action has just been pressed. If it has, we then check to see if we have more than `0` grenades for the current grenade we have selected.

If we have more than `0` grenades, we then remove one from the grenade amounts for the current grenade. Then, based on the grenade we're currently using we instance the proper grenade scene and assign it to `grenade_clone`.

Next we add `grenade_clone` as a child of the node at the root, and set its global *Transform* to `Grenade_Toss_Pos`'s global *Transform*. Finally, we apply an impulse to the grenade so that it is launched forward, relative to the `Z` directional vector of `grenade_clone`'s.

Now we can use both types of grenades, but there's a few things we should probably add before we move on to adding the other things.

We still need a way to see how many grenades we have left, and we should probably have a way to get more grenades when we pick up ammo.

First, let's change some of the code in `Player.gd` so we can see how many grenades we have left. Change `process_UI` to the following:

```
func process_UI(delta):
    if current_weapon_name == "UNARMED" or current_weapon_name == "KNIFE":
        # First line: Health, second line: Grenades
        UI_status_label.text = "HEALTH: " + str(health) + \
```

(continues on next page)

(continued from previous page)

```
        "\n" + current_grenade + ":" + str(grenade_amounts[current_grenade])
    else:
        var current_weapon = weapons[current_weapon_name]
        # First line: Health, second line: weapon and ammo, third line: grenades
        UI_status_label.text = "HEALTH: " + str(health) + \
        "\nAMMO:" + str(current_weapon.ammo_in_weapon) + "/" + str(current_weapon.
↪spare_ammo) + \
        "\n" + current_grenade + ":" + str(grenade_amounts[current_grenade])
```

Now we'll show how many grenades we have left in our UI.

While we're still in `Player.gd`, let's add a function to add grenades. Add the following function to `Player.gd`:

```
func add_grenade(additional_grenade):
    grenade_amounts[current_grenade] += additional_grenade
    grenade_amounts[current_grenade] = clamp(grenade_amounts[current_grenade], 0, 4)
```

Now we can add a grenade using `add_grenade`, and it will automatically be clamped to a maximum of `4` grenades.

Tip: You can change the `4` to a constant if you want. You'd need to make a new global constant, something like `MAX_GRENADES`, and then change the clamp from `clamp(grenade_amounts[current_grenade], 0, 4)` to `clamp(grenade_amounts[current_grenade], 0, MAX_GRENADES)`

If you do not want to limit how many grenades you can carry, remove the line that clamps the grenades altogether!

Now we have a function to add grenades, let's open up `AmmoPickup.gd` and use it!

Open up `AmmoPickup.gd` and go to the `trigger_body_entered` function. Change it to the following:

```
func trigger_body_entered(body):
    if body.has_method("add_ammo"):
        body.add_ammo(AMMO_AMOUNTS[kit_size])
        respawn_timer = RESPAWN_TIME
        kit_size_change_values(kit_size, false)

    if body.has_method("add_grenade"):
        body.add_grenade(GRENADE_AMOUNTS[kit_size])
        respawn_timer = RESPAWN_TIME
        kit_size_change_values(kit_size, false)
```

Now we're also checking to see if the body has the `add_grenade` function. If it does, we call it like we call `add_ammo`.

You may have noticed we're using a new constant we haven't defined yet, `GRENADE_AMOUNTS`. Let's add it! Add the following global variable to `AmmoPickup.gd` with the other global variables:

```
const GRENADE_AMOUNTS = [2, 0]
```

- `GRENADE_AMOUNTS`: The amount of grenades each pick up in each size contains.

Notice how the second element in `GRENADE_AMOUNTS` is `0`. This is so the small ammo pick up does not give our player any additional grenades.

Now you should be able to throw grenades now! Go give it a try!

Adding the ability to grab and throw RigidBody nodes to the player

Next let's give our player the ability to pick up and throw *RigidBody* nodes.

Open up `Player.gd` and add the following global variables:

```
var grabbed_object = null
const OBJECT_THROW_FORCE = 120
const OBJECT_GRAB_DISTANCE = 7
const OBJECT_GRAB_RAY_DISTANCE = 10
```

- `grabbed_object`: A variable to hold the grabbed *RigidBody* node.
- `OBJECT_THROW_FORCE`: The force we throw the grabbed object at.
- `OBJECT_GRAB_DISTANCE`: The distance away from the camera we hold the grabbed object at.
- `OBJECT_GRAB_RAY_DISTANCE`: The distance the *Raycast* goes. This is our grab distance.

With that done, all we need to do is add some code to `process_input`:

```
# ----------------------------------
# Grabbing and throwing objects

if Input.is_action_just_pressed("fire") and current_weapon_name == "UNARMED":
    if grabbed_object == null:
        var state = get_world().direct_space_state

        var center_position = get_viewport().size/2
        var ray_from = camera.project_ray_origin(center_position)
        var ray_to = ray_from + camera.project_ray_normal(center_position) * OBJECT_
↪GRAB_RAY_DISTANCE

        var ray_result = state.intersect_ray(ray_from, ray_to, [self, $Rotation_
↪Helper/Gun_Fire_Points/Knife_Point/Area])
        if ray_result:
            if ray_result["collider"] is RigidBody:
                grabbed_object = ray_result["collider"]
                grabbed_object.mode = RigidBody.MODE_STATIC

                grabbed_object.collision_layer = 0
                grabbed_object.collision_mask = 0

    else:
        grabbed_object.mode = RigidBody.MODE_RIGID

        grabbed_object.apply_impulse(Vector3(0,0,0), -camera.global_transform.basis.z.
↪normalized() * OBJECT_THROW_FORCE)

        grabbed_object.collision_layer = 1
        grabbed_object.collision_mask = 1

        grabbed_object = null

if grabbed_object != null:
    grabbed_object.global_transform.origin = camera.global_transform.origin + (-
↪camera.global_transform.basis.z.normalized() * OBJECT_GRAB_DISTANCE)
# ----------------------------------
```

Let's go over what's happening.

First we check to see if the action pressed is the `fire` action, and that we are using the `UNARMED` weapon. This is because we only want to be able to pick up and throw objects when we're not using any weapons. This is a design choice, but I feel it gives `UNARMED` a use.

Next we check to see whether or not `grabbed_object` is `null`.

If `grabbed_object` is `null`, we want to see if we can pick up a *RigidBody*.

We first get the direct space state from the current *World*. This is so we can cast a ray entirely from code, instead of having to use a *Raycast* node.

Note: see *Ray-casting* for more information on raycasting in Godot.

Then we get the center of the screen by dividing the current *Viewport* size in half. We then get the ray's origin point and end point using `project_ray_origin` and `project_ray_normal` from the camera. If you want to know more about how these functions work, see *Ray-casting*.

Next we send our ray into the space state and see if we get a result. We add ourselves and the knife's *Area* as two exceptions so we cannot carry ourselves or the knife's collision area.

Then we check to see if we got a result back. If we have, we then see if the collider the ray collided with is a *RigidBody*.

If the ray collided with a *RigidBody*, we set `grabbed_object` to the collider the ray collided with. We then set the mode on the *RigidBody* we collided with to `MODE_STATIC` so it's not moved.

Finally, we set its collision layer and collision mask to `0`. This will make it have no collision layer or mask, which will means it will not be able to collide with anything.

If `grabbed_object` is not `null`, then we need to throw the *RigidBody* we're holding.

We first set the *RigidBody* we holding mode to `MODE_RIGID`.

Note: This is making a rather large assumption that the all rigid bodies will be using `MODE_RIGID`. While that is the case for this tutorial series, that may not be the case in other projects.

If you have *RigidBody*'s with different modes, you may need to store the mode of the *RigidBody* you have picked up into a global variable so you can change it back to the mode it was in before you picked it up.

Then we apply an impulse to send it flying forward. We send it flying in the direction the camera is facing, at `OBJECT_THROW_FORCE` force.

We then set the grabbed *RigidBody*'s collision layer and mask to `1`, so it can collide with anything on layer `1` again.

Note: This is, once again, making a rather large assumption that all rigid bodies will be only on collision layer `1`, and all collision masks will be on layer `1`. If you are using this script in other projects, you may need to store the collision layer/mask of the *RigidBody* before you change them to `0`.

Finally, we set `grabbed_object` to `null` since we have successfully thrown the held object.

The last thing we do is check to see whether or not `grabbed_object` is equal to `null`, outside of the grabbing/throwing code.

Note: While technically not input related, it's easy enough to place the code moving the grabbed object here because it's only two lines, and then all of the grabbing/throwing code is in one place

If we are holding an object, we set its global position to the camera's position plus `OBJECT_GRAB_DISTANCE` in the direction the camera is facing.

Before we test this, we need to change something in `_physics_process`. While we're holding an object, we don't want to be able to change weapons or reload, so change `_physics_process` to the following:

```
func _physics_process(delta):
    process_input(delta)
    process_view_input(delta)
    process_movement(delta)

    if grabbed_object == null:
        process_changing_weapons(delta)
        process_reloading(delta)

    # Process the UI
    process_UI(delta)
```

Now we cannot change weapons or reload while holding an object.

Now you can grab and throw RigidBody nodes while in a `UNARMED` state! Go give it a try!

Adding a turret

Next, let's make a turret to shoot our player!

Open up `Turret.tscn`. Expand `Turret` if it's not already expanded.

Notice how our turret is broken up into several parts. We have a `Base`, `Head`, `Vision_Area`, and a `Smoke` *Particles*.

Open up `Base` and you'll find it's a *StaticBody* and a mesh. Open up `Head` and you'll find there's several meshes, a *StaticBody* and a *Raycast* node.

One thing to note with the `Head` is that the raycast will be where our bullets will fire from if we are using raycasting. We also have two meshes called `Flash` and `Flash_2`. These will be the muzzle flash that briefly shows when the turret fires.

`Vision_Area` is a *Area* we'll use as the turret's ability to see. When something enters `Vision_Area`, we'll assume the turret can see it.

`Smoke` is a *Particles* node that will play when the turret is destroyed and repairing.

Now that we've looked at how the scene is set up, lets start writting the code for the turret. Select `Turret` and create a new script called `Turret.gd`. Add the following to `Turret.gd`:

```
extends Spatial

export (bool) var use_raycast = false

const TURRET_DAMAGE_BULLET = 20
```

(continues on next page)

(continued from previous page)

```
const TURRET_DAMAGE_RAYCAST = 5

const FLASH_TIME = 0.1
var flash_timer = 0

const FIRE_TIME = 0.8
var fire_timer = 0

var node_turret_head = null
var node_raycast = null
var node_flash_one = null
var node_flash_two = null

var ammo_in_turret = 20
const AMMO_IN_FULL_TURRET = 20
const AMMO_RELOAD_TIME = 4
var ammo_reload_timer = 0

var current_target = null

var is_active = false

const PLAYER_HEIGHT = 3

var smoke_particles

var turret_health = 60
const MAX_TURRET_HEALTH = 60

const DESTROYED_TIME = 20
var destroyed_timer = 0

var bullet_scene = preload("Bullet_Scene.tscn")

func _ready():

    $Vision_Area.connect("body_entered", self, "body_entered_vision")
    $Vision_Area.connect("body_exited", self, "body_exited_vision")

    node_turret_head = $Head
    node_raycast = $Head/Ray_Cast
    node_flash_one = $Head/Flash
    node_flash_two = $Head/Flash_2

    node_raycast.add_exception(self)
    node_raycast.add_exception($Base/Static_Body)
    node_raycast.add_exception($Head/Static_Body)
    node_raycast.add_exception($Vision_Area)

    node_flash_one.visible = false
    node_flash_two.visible = false

    smoke_particles = $Smoke
    smoke_particles.emitting = false

    turret_health = MAX_TURRET_HEALTH
```

(continues on next page)

(continued from previous page)

```
func _physics_process(delta):

    if is_active == true:

        if flash_timer > 0:
            flash_timer -= delta

            if flash_timer <= 0:
                node_flash_one.visible = false
                node_flash_two.visible = false

        if current_target != null:

            node_turret_head.look_at(current_target.global_transform.origin +␣
↪Vector3(0, PLAYER_HEIGHT, 0), Vector3(0, 1, 0))

            if turret_health > 0:

                if ammo_in_turret > 0:
                    if fire_timer > 0:
                        fire_timer -= delta
                    else:
                        fire_bullet()
                else:
                    if ammo_reload_timer > 0:
                        ammo_reload_timer -= delta
                    else:
                        ammo_in_turret = AMMO_IN_FULL_TURRET

    if turret_health <= 0:
        if destroyed_timer > 0:
            destroyed_timer -= delta
        else:
            turret_health = MAX_TURRET_HEALTH
            smoke_particles.emitting = false


func fire_bullet():
    if use_raycast == false:

        var clone = bullet_scene.instance()
        var scene_root = get_tree().root.get_children()[0]
        scene_root.add_child(clone)

        clone.global_transform = $Head/Barrel_End.global_transform
        clone.scale = Vector3(8, 8, 8)
        clone.BULLET_DAMAGE = TURRET_DAMAGE_BULLET
        clone.BULLET_SPEED = 60

        ammo_in_turret -= 1

    else:
        node_raycast.look_at(current_target.global_transform.origin + PLAYER_HEIGHT,␣
↪Vector3(0,1,0))

        node_raycast.force_raycast_update()
```

(continues on next page)

(continued from previous page)

```
        if node_raycast.is_colliding():
            var body = node_raycast.get_collider()
            if body.has_method("bullet_hit"):
                body.bullet_hit(TURRET_DAMAGE_RAYCAST, node_raycast.get_collision_
↪point())

        ammo_in_turret -= 1

    node_flash_one.visible = true
    node_flash_two.visible = true

    flash_timer = FLASH_TIME
    fire_timer = FIRE_TIME

    if ammo_in_turret <= 0:
        ammo_reload_timer = AMMO_RELOAD_TIME


func body_entered_vision(body):
    if current_target == null:
        if body is KinematicBody:
            current_target = body
            is_active = true


func body_exited_vision(body):
    if current_target != null:
        if body == current_target:
            current_target = null
            is_active = false

            flash_timer = 0
            fire_timer = 0
            node_flash_one.visible = false
            node_flash_two.visible = false


func bullet_hit(damage, bullet_hit_pos):
    turret_health -= damage

    if turret_health <= 0:
        smoke_particles.emitting = true
        destroyed_timer = DESTROYED_TIME
```

This is quite a bit of code, so let's break it down function by function. Let's first look at the global variables:

- `use_raycast`: A exported boolean so we can change whether the turret uses objects or raycasting for bullets.
- `TURRET_DAMAGE_BULLET`: The amount of damage a single bullet scene does.
- `TURRET_DAMAGE_RAYCAST`: The amount of damage a single *Raycast* bullet does.
- `FLASH_TIME`: The amount of time (in seconds) the muzzle flash meshes are visible.
- `flash_timer`: A variable for tracking how long the muzzle flash meshes have been visible.
- `FIRE_TIME`: The amount of time (in seconds) needed to fire a bullet.
- `fire_timer`: A variable for tracking how much time has passed since the turret last fired.

- `node_turret_head`: A variable to hold the `Head` node.
- `node_raycast`: A variable to hold the *Raycast* node attached to the turret's head.
- `node_flash_one`: A variable to hold the first muzzle flash *MeshInstance*.
- `node_flash_two`: A variable to hold the second muzzle flash *MeshInstance*.
- `ammo_in_turret`: The amount of ammo currently in the turret.
- `AMMO_IN_FULL_TURRET`: The amount of ammo in a full turret.
- `AMMO_RELOAD_TIME`: The amount of time it takes the turret to reload.
- `ammo_reload_timer`: A variable for tracking how long the turret has been reloading.
- `current_target`: The turret's current target.
- `is_active`: A variable for tracking whether the turret is able to fire at the target.
- `PLAYER_HEIGHT`: The amount of height we're adding to the target so we're not shooting at its feet.
- `smoke_particles`: A variable to hold the smoke particles node.
- `turret_health`: The amount of health the turret currently has.
- `MAX_TURRET_HEALTH`: The amount of health a fully healed turret has.
- `DESTROYED_TIME`: The amount of time (in seconds) it takes for a destroyed turret to repair itself.
- `destroyed_timer`: A variable for tracking the amount of time a turret has been destroyed.
- `bullet_scene`: The bullet scene the turret fires (same scene as the player's pistol)

Phew, that's quite a few global variables!

Let's go through `_ready` next.

First we get the vision area and connect the `body_entered` and `body_exited` signals to `body_entered_vision` and `body_exited_vision` respectively.

We then get all of the nodes and assign them to their respective variables.

Next add some exceptions to the *Raycast* so the turret cannot hurt itself.

Then we make both flash meshes invisible to start, since we're not going to be firing during `_ready`.

We then get the smoke particles node and assign it to the `smoke_particles` node. We also set `emitting` to `false` to assure it's not emitting until the turret is broken.

Finally, we set the turret's health to `MAX_TURRET_HEALTH` so it starts at full health.

Now let's go through `_physics_process`.

First we check to see if the turret is active. If the turret is active we want to process the firing code.

Next we check to see if `flash_timer` is more than zero, meaning the flash meshes are visible, we want to remove delta from `flash_timer`. If `flash_timer` gets to zero or less after we've subtracted `delta`, we want to hide both of the flash meshes.

Next we check to see if we have a target or not. If we have a target, we make the turret head look at it, adding `PLAYER_HEIGHT` so we're not aiming at the player's feet.

We then check to see if the turret's health is more than zero. If it is, we then check to see if there is ammo in the turret.

If there is ammo in the turret, we then check to see if `fire_timer` is more than zero. If `fire_timer` is more than zero, we cannot fire and need to remove `delta` from `fire_timer`. If `fire_timer` is equal to or less than zero, we want to fire a bullet, so we call the `fire_bullet` function.

If there is not any ammo in the turret, we check to see if `ammo_reload_timer` is more than zero. If `ammo_reload_timer` is more than zero, we subtract `delta` from `ammo_reload_timer`. If `ammo_reload_timer` is equal to or less than zero, we set `ammo_in_turret` to `AMMO_IN_FULL_TURRET` because we've waited long enough to refill the turret.

Next we check to see if the turret's health is less than or equal to `0`, outside of whether we're active or not. If the turret's health is zero or less, we then check to see if `destroyed_timer` is more than zero. If destroyed timer is more than zero, we subtract `delta` from `destroyed_timer`.

If `destyored_timer` is less than or equal to zero, we set `turret_health` to `MAX_TURRET_HEALTH` and stop emitting smoke particles by setting `smoke_particles.emitting` to `false`.

Next let's go through `fire_bullet`.

First we check to see whether we're using a raycast or not.

The code for the using a raycast is almost entirely the same as the code in the rifle from *Part 2*, so I'm only going to go over it briefly.

We first make the raycast look at the target, assuring we'll hit the target. We then force the raycast to update so we get a frame perfect collision check. We then check if the raycast collided with anything. If the raycast has collided with something, we then check to see if the collided body has the `bullet_hit` function. If it does, we call it and pass in the damage a single raycast bullet does. We then remove `1` from `ammo_in_turret`.

If we are not using a raycast, we spawn a bullet object instead. This code is almost entirely the same as the code in the pistol from *Part 2*, so like with the raycast code, I'm only going to go over it briefly.

We first make a bullet clone and assign it to `clone`. We then add that as a child of the root node. We set it's global transform to the barrel end, scale it up since it's too small, and set it's damage and speed using the turret's constant global variables. We then remove `1` from `ammo_in_turret`.

Then, regardless of which bullet method we used, we make both of the muzzle flash meshes visible. We set `flash_timer` and `fire_timer` to to `FLASH_TIME` and `FIRE_TIME` respectively. We then check to see if we used the last bullet in the turret. If we have used the last bullet, we set `ammo_reload_timer` to `AMMO_RELOAD_TIME`.

Let's look at `body_entered_vision` next, and thankfully it's rather short.

We first check to see if we currently have a target by checking to see if `current_target` is equal to `null`. If we do not have a target, we then check to see if the body that just entered the vision *Area* is a *KinematicBody*

..note:: We're assuming the turret only should fire at *KinematicBody* nodes, since that's what our player(s) are using.

If the body that just the vision *Area* is a *KinematicBody*, we set `current_target` to the body, and set `is_active` to `true`.

Now let's look at `body_exited_vision`.

First we check to see if we have a target. If we have a target, we then check to see if the body that has just left our vision area is our target.

If the body that just left the area is the current target, we set `current_target` to `null`, set `is_active` to `false`, and reset all of the variables related to firing the turret, since we no longer have a target to fire at.

Finally, let's look at `bullet_hit`.

We first remove however much damage we have received from the turret's health.

Then we check to see if we've been destroyed. If we have, we start the smoke particles emitting and set `destroyed_timer` to `DESTROYED_TIME` so we have to wait to repair the turret.

Phew, with all of that done and coded we only have one last thing to do before our turrets are ready for use. Open up `Turret.tscn` if it's not already open and select one of the *StaticBody* nodes from either `Body` or `Head`. Create a new script called `TurretBodies.gd` and attach it to whichever *StaticBody* you have selected.

Add the following code to `TurretBodies.gd`:

```
extends StaticBody

export (NodePath) var path_to_turret_root

func _ready():
    pass

func bullet_hit(damage, bullet_hit_pos):
    if path_to_turret_root != null:
        get_node(path_to_turret_root).bullet_hit(damage, bullet_hit_pos)
```

All this code does is call `bullet_hit` on whatever node `path_to_turret_root` leads to. Go back to the editor and assign the *NodePath* to the `Turret` node.

Now select the other *StaticBody* node (either in `Body` or `Head`) and assign `TurretBodies.gd` to it. Once the script is attached, assign the *NodePath* to the `Turret` node.

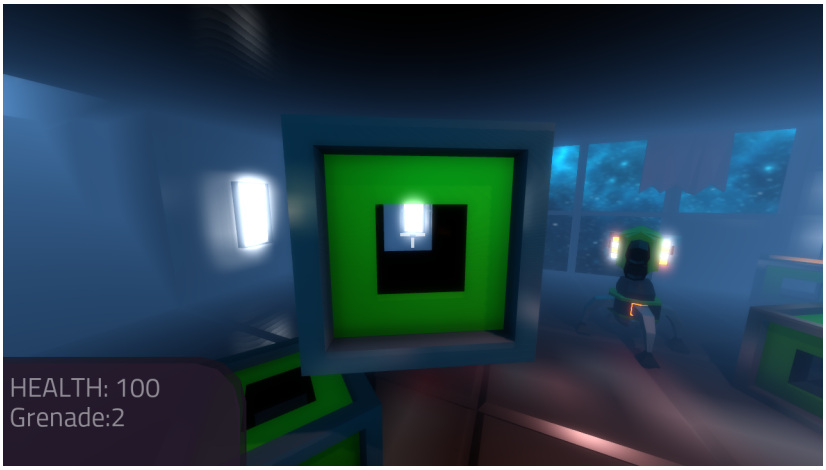
The last thing we need to do is add a way for the player to be hurt. Since all of our bullets use the `bullet_hit` function, we need to add that to our player.

Open `Player.gd` and add the following:

```
func bullet_hit(damage, bullet_hit_pos):
    health -= damage
```

With all that done, you should have fully operational turrets! Go place a few in one/both/all of the scenes and give them a try!

Final notes



Now you the player can pick up *RigidBody* nodes and throw grenades. We now also have turrets to fire at our player.

In *Part 6*, we're going to add a main menu and pause menu, add a respawn system for the player, and change/move the sound system so we can use it from any script.

Warning: If you ever get lost, be sure to read over the code again!

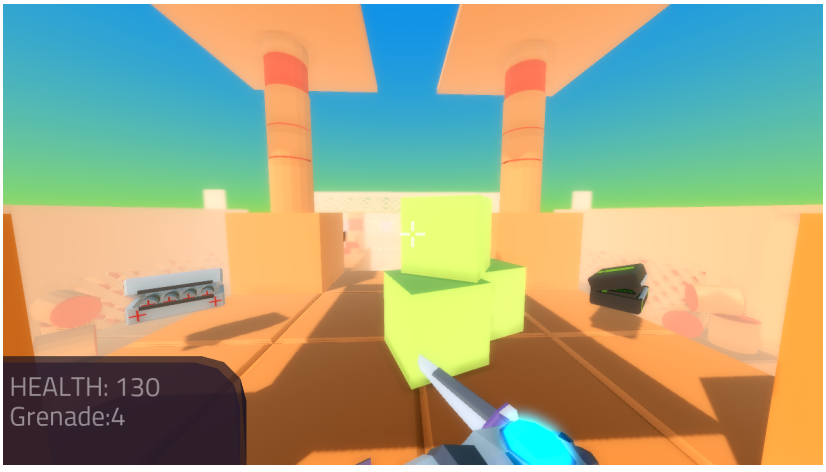
You can download the finished project for this part here: `Godot_FPS_Part_5.zip`

7.16.6 Part 6

Part Overview

In this part we're going to add a main menu and pause menu, add a respawn system for the player, and change/move the sound system so we can use it from any script.

This is the last part of the FPS tutorial, by the end of this you will have a solid base to build amazing FPS games with Godot!



Note: You are assumed to have finished *Part 5* before moving on to this part of the tutorial.

The finished project from *Part 4* will be the starting project for part 6

Let's get started!

Adding the main menu

First, open up `Main_Menu.tscn` and take a look at how the scene is set up.

The main menu is broken up into three different panels, each representing a different 'screen' of our main menu.

Note: The `Background_Animation` node is just so the background of the menu is a bit more interesting than a solid color. It's a camera looking around the skybox, nothing fancy.

Feel free to expand all of the nodes and see how their set up. Remember to keep only `Start_Menu` visible when you're done, as that's the screen we want to show first when we enter the main menu.

Select `Main_Menu` (the root node) and create a new script called `Main_Menu.gd`. Add the following:

```
extends Control

var start_menu
var level_select_menu
var options_menu

export (String, FILE) var testing_area_scene
export (String, FILE) var space_level_scene
export (String, FILE) var ruins_level_scene

func _ready():
    start_menu = $Start_Menu
    level_select_menu = $Level_Select_Menu
    options_menu = $Options_Menu

    $Start_Menu/Button_Start.connect("pressed", self, "start_menu_button_pressed", [
↪"start"])
    $Start_Menu/Button_Open_Godot.connect("pressed", self, "start_menu_button_pressed
↪", ["open_godot"])
    $Start_Menu/Button_Options.connect("pressed", self, "start_menu_button_pressed", [
↪"options"])
    $Start_Menu/Button_Quit.connect("pressed", self, "start_menu_button_pressed", [
↪"quit"])

    $Level_Select_Menu/Button_Back.connect("pressed", self, "level_select_menu_button_
↪pressed", ["back"])
    $Level_Select_Menu/Button_Level_Testing_Area.connect("pressed", self, "level_
↪select_menu_button_pressed", ["testing_scene"])
    $Level_Select_Menu/Button_Level_Space.connect("pressed", self, "level_select_menu_
↪button_pressed", ["space_level"])
    $Level_Select_Menu/Button_Level_Ruins.connect("pressed", self, "level_select_menu_
↪button_pressed", ["ruins_level"])

    $Options_Menu/Button_Back.connect("pressed", self, "options_menu_button_pressed",␣
↪["back"])
    $Options_Menu/Button_Fullscreen.connect("pressed", self, "options_menu_button_
↪pressed", ["fullscreen"])
```

(continues on next page)

(continued from previous page)

```
    $Options_Menu/Check_Button_VSync.connect("pressed", self, "options_menu_button_
↪pressed", ["vsync"])
    $Options_Menu/Check_Button_Debug.connect("pressed", self, "options_menu_button_
↪pressed", ["debug"])

    Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

    var globals = get_node("/root/Globals")
    $Options_Menu/HSlider_Mouse_Sensitivity.value = globals.mouse_sensitivity
    $Options_Menu/HSlider_Joypad_Sensitivity.value = globals.joypad_sensitivity


func start_menu_button_pressed(button_name):
    if button_name == "start":
        level_select_menu.visible = true
        start_menu.visible = false
    elif button_name == "open_godot":
        OS.shell_open("https://godotengine.org/")
    elif button_name == "options":
        options_menu.visible = true
        start_menu.visible = false
    elif button_name == "quit":
        get_tree().quit()


func level_select_menu_button_pressed(button_name):
    if button_name == "back":
        start_menu.visible = true
        level_select_menu.visible = false
    elif button_name == "testing_scene":
        set_mouse_and_joypad_sensitivity()
        get_node("/root/Globals").load_new_scene(testing_area_scene)
    elif button_name == "space_level":
        set_mouse_and_joypad_sensitivity()
        get_node("/root/Globals").load_new_scene(space_level_scene)
    elif button_name == "ruins_level":
        set_mouse_and_joypad_sensitivity()
        get_node("/root/Globals").load_new_scene(ruins_level_scene)


func options_menu_button_pressed(button_name):
    if button_name == "back":
        start_menu.visible = true
        options_menu.visible = false
    elif button_name == "fullscreen":
        OS.window_fullscreen = !OS.window_fullscreen
    elif button_name == "vsync":
        OS.vsync_enabled = $Options_Menu/Check_Button_VSync.pressed
    elif button_name == "debug":
        pass


func set_mouse_and_joypad_sensitivity():
    var globals = get_node("/root/Globals")
    globals.mouse_sensitivity = $Options_Menu/HSlider_Mouse_Sensitivity.value
    globals.joypad_sensitivity = $Options_Menu/HSlider_Joypad_Sensitivity.value
```

Most of the code here relates to making UIs, which is outside of the purpose of this tutorial series. **We're only going to look at the UI related code briefly.**

Tip: See *Design a title screen* and the tutorials following for better ways to make GUIs and UIs!

Let's look at the global variables first.

- `start_menu`: A variable to hold the `Start_Menu` *Panel*.
- `level_select_menu`: A variable to hold the `Level_Select_Menu` *Panel*.
- `options_menu`: A variable to hold the `Options_Menu` *Panel*.
- `testing_area_scene`: The path to the `Testing_Area.tscn` file, so we can change to it from this scene.
- `space_level_scene`: The path to the `Space_Level.tscn` file, so we can change to it from this scene.
- `ruins_level_scene`: The path to the `Ruins_Level.tscn` file, so we can change to it from this scene.

Warning: You'll have to set the paths to the correct files in the editor before testing this script! Otherwise it will not work!

Now let's go over `_ready`

First we get all of the *Panel* nodes and assign them to the proper variables.

Next we connect all of the buttons `pressed` signals to their respective `[panel_name_here]_button_pressed` functions.

We then set the mouse mode to `MOUSE_MODE_VISIBLE` to ensure whenever we return to this scene our mouse will be visible.

Then we get a singleton, called `Globals`. We then set the values for the *HSlider* nodes so their values line up with the mouse and joypad sensitivity in the singleton.

Note: We have not made the `Globals` singleton yet, so don't worry! We're going to make it soon!

In `start_menu_pressed`, we check to see which button is pressed.

Based on the button pressed, we either change the currently visible panel, quit the application, or open the Godot engine website.

In `level_select_menu_button_pressed`, we check to see which button is pressed.

If the `back` button has been pressed, we change the currently visible panels so we return to the main menu.

If one of the scene changing buttons are pressed, we fist call `set_mouse_and_joypad_sensitivity` so our singleton has the values from the *HSlider* nodes. Then we tell the singleton to change nodes using it's `load_new_scene` function, passing in the file path of the scene we're wanting to change to.

Note: Don't worry about the singleton, we'll get there soon!

In `options_menu_button_pressed`, we check to see which button is pressed.

If the `back` button has been pressed, we change the currently visible panels so we return to the main menu.

If the `fullscreen` button is pressed we toggle the *OS*'s full screen mode by setting it to the flipped version of it's current value.

If the `vsync` button is pressed we set the *OS*'s Vsync based on the state of the Vsync check button.

Finally, lets take a look at `set_mouse_and_joypad_sensitivity`.

First we get the `Globals` singleton and assign it to a local variable.

We then set the `mouse_sensitivity` and `joypad_sensitvity` variables to the values in their respective *HSlider* node counterparts.

Making the `Globals` singleton

Now, for this all to work we need to create the `Globals` singleton. Make a new script in the `Script` tab and call it `Globals.gd`.

Add the following to `Globals.gd`.

```
extends Node

var mouse_sensitivity = 0.08
var joypad_sensitivity = 2

func _ready():
    pass

func load_new_scene(new_scene_path):
    get_tree().change_scene(new_scene_path)
```

As you can see, it's quite small and simple. As this part progresses we will keeping adding complexities to `Global.gd`, but for now all it is doing is holding two variables for us, and abstracting how we change scenes.

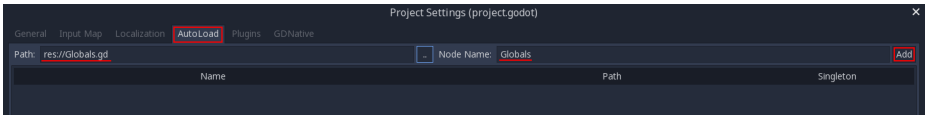
- `mouse_sensitivity`: The current sensitivity for our mouse, so we can load it in `Player.gd`.
- `joypad_sensitivity`: The current sensitivity for our joypad, so we can load it in `Player.gd`.

Right now all we're using `Globals.gd` for is a way to carry variables across scenes. Because the sensitivity for our mouse and joypad are stored in `Globals.gd`, any changes we make in one scene (like `Main_Menu`) effect the sensitivity for our player.

All we're doing in `load_new_scene` is calling *SceneTree*'s `change_scene` function, passing in the scene path given in `load_new_scene`.

That's all of the code needed for `Globals.gd` right now! Before we can test the main menu, we first need to set `Globals.gd` as an autoload script.

Open up the project settings and click the `AutoLoad` tab.



Then select the path to `Globals.gd` in the `Path` field by clicking the button beside it. Make sure the name in the `Node Name` field is `Globals`. If you have everything like the picture above, then press `Add`!

This will make `Globals.gd` a singleton/autoload script, which will allow us to access it from anywhere in any scene.

Tip: For more information on singleton/autoload scripts, see *Singletons (AutoLoad)*.

Now that `Globals.gd` is a singleton/autoload script, you can test the main menu!

You may also want to change the main scene from `Testing_Area.tscn` to `Main_Menu.tscn` so when we export the game we start at the main menu. You can do this through the project settings, under the `General` tab. Then in the `Application` category, click the `Run` subcategory and you can change the main scene by changing the value in `Main Scene`.

Warning: You'll have to set the paths to the correct files in `Main_Menu` in the editor before testing the main menu! Otherwise you will not be able to change scenes from the level select menu/screen.

Adding the debug menu

Now let's add a simple debugging scene so we can track things like FPS in game. Open up `Debug_Display.tscn`.

You can see it's a *Panel* positioned in the top right corner of the screen. It has three *Labels*, one for displaying the FPS the game is running at, one for showing what OS the game is running on, and a label for showing the Godot version the game is running with.

Let's add the code needed to fill these *Labels*. Select `Debug_Display` and create a new script called `Debug_Display.gd`. Add the following:

```
extends Control

func _ready():
    $OS_Label.text = "OS:" + OS.get_name()
    $Engine_Label.text = "Godot version:" + Engine.get_version_info()["string"]

func _process(delta):
    $FPS_Label.text = "FPS:" + str(Engine.get_frames_per_second())
```

Let's go over what this script does.

In `_ready` we set the `OS_Label`'s text to the name provided in *OS* using the `get_name` function. This will return the name of the OS (or Operating System) that Godot was compiled for. For example, when you are running Windows it will return `Windows`, while when you are running Linux it will return `X11`.

Then we set the `Engine_Label`'s text to the version info provided by `Engine.get_version_info`. `Engine.get_version_info` returns a dictionary full of useful information about the version Godot is currently running with. We only care for the string version for the purposes of this display, so we get the string and assign that as the `text` in `Engine_Label`. See *Engine* for more information on the values `get_version_info` returns.

In `_process` we set the text of the `FPS_Label` to `Engine.get_frames_per_second`, but because `get_frames_per_second` returns a integer, we have to cast it to a string using `str` before we can add it to our label.

Now let's jump back to `Main_Menu.gd` and change the following in `options_menu_button_pressed`:

```
elif button_name == "debug":
    pass
```

to this instead:

```
elif button_name == "debug":
    get_node("/root/Globals").set_debug_display($Options_Menu/Check_Button_Debug.
↪pressed)
```

This will call a new function in our singleton called `set_debug_display`, so let's add that next!

Open up `Globals.gd` and add the following global variables:

```
# -------------------------------------
# All of the GUI/UI related variables

var canvas_layer = null

const DEBUG_DISPLAY_SCENE = preload("res://Debug_Display.tscn")
var debug_display = null

# -------------------------------------
```

- `canvas_layer`: A canvas layer so our GUI/UI is always drawn on top.
- `DEBUG_DISPLAY`: The debug display scene we worked on earlier.
- `debug_display`: A variable to hold the debug display when there is one.

Now that we have our global variables defined, we need to add a few lines to ready so we have a canvas layer to use in `canvas_layer`. Change `_ready` to the following:

```
func _ready():
    canvas_layer = CanvasLayer.new()
    add_child(canvas_layer)
```

Now in `_ready` we're creating a new canvas layer and adding it as a child of the autoload script.

The reason we're adding a *CanvasLayer* is so all of our GUI and UI nodes we instance/spawn in `Globals.gd` are always drawn on top of everything else.

When adding nodes to a singleton/autoload, you have to be careful not to lose reference to any of the child nodes. This is because nodes will not be freed/destroyed when you change scene, meaning you can run into memory problems if you are instancing/spawning lots of nodes and are not freeing them.

Now we need to add `set_debug_display` to `Globals.gd`:

```
func set_debug_display(display_on):
    if display_on == false:
        if debug_display != null:
            debug_display.queue_free()
            debug_display = null
    else:
        if debug_display == null:
            debug_display = DEBUG_DISPLAY_SCENE.instance()
            canvas_layer.add_child(debug_display)
```

Let's go over what's happening.

First we check to see if we're trying to turn on the debug display, or turn it off.

If we are turning off the display, we then check to see if `debug_display` is not equal to `null`. If `debug_display` is not equal to `null`, then we most have a debug display currently active. If we have a debug display active, we free it using `queue_free` and then assign `debug_display` to `null`.

If we are turning on the display, we then check to make sure we do not already have a debug display active. We do this by making sure `debug_display` is equal to `null`. If `debug_display` is `null`, we instance a new `DEBUG_DISPLAY_SCENE`, and add it as a child of `canvas_layer`.

With that done, we can now toggle the debug display on and off by switching the *CheckButton* in the `Options_Menu` panel. Go give it a try!

Notice how the debug display stays even when you change scenes from the `Main_Menu.tscn` to another scene (like `Testing_Area.tscn`). This is the beauty of instancing/spawning nodes in a singleton/autoload and adding them as children to the singleton/autoload. Any of the nodes added as children of the singleton/autoload will stay for as long as the game is running, without any additional work on our part!

Adding a pause menu

Let's add a pause menu so we can return to the main menu when we press the `ui_cancel` action.

Open up `Pause_Popup.tscn`.

Notice how the root node in `Pause_Popup` is a *WindowDialog*. *WindowDialog* inherits from *Popup*, which means *WindowDialog* can act like a popup.

Select `Pause_Popup` and scroll down all the way till you get to the `Pause` menu in the inspector. Notice how the pause mode is set to `process` instead of `inherit` like it is normally set by default. This makes it where it will continue to process even when the game is paused, which we need in order to interact with the UI elements.

Now that we've looked at how `Pause_Popup.tscn` is set up, lets write the code to make it work. Normally we'd attach a script to the root node of the scene, `Pause_Popup` in this case, but since we'll need to receive a couple of signals in `Globals.gd`, we'll write all of the code for the pop up there.

Open up `Globals.gd` and add the following global variables:

```
const MAIN_MENU_PATH = "res://Main_Menu.tscn"
const POPUP_SCENE = preload("res://Pause_Popup.tscn")
var popup = null
```

- `MAIN_MENU_PATH`: The path to the main menu scene.
- `POPUP_SCENE`: The pop up scene we looked at earlier.
- `popup`: A variable to hold the pop up scene.

Now we need to add `_process` to `Globals.gd` so we can respond when the `ui_cancel` action is pressed. Add the following to `_process`:

```
func _process(delta):
    if Input.is_action_just_pressed("ui_cancel"):
        if popup == null:
            popup = POPUP_SCENE.instance()

            popup.get_node("Button_quit").connect("pressed", self, "popup_quit")
```

(continues on next page)

(continued from previous page)

```
            popup.connect("popup_hide", self, "popup_closed")
            popup.get_node("Button_resume").connect("pressed", self, "popup_closed")

            canvas_layer.add_child(popup)
            popup.popup_centered()

            Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

            get_tree().paused = true
```

Let's go over what's happening here.

First we check to see if the `ui_cancel` action is pressed. Then we check to make sure we do not already have a `popup` open by checking to see if `popup` is equal to `null`.

If we do not have a pop up open, we instance `POPUP_SCENE` and assign it to `popup`.

We then get the quit button and assign it's `pressed` signal to `popup_quit`, which we will be adding shortly.

Next we assign both the `popup_hide` signal from the *WindowDialog* and the `pressed` signal from the resume button to `popup_closed`, which we will be adding shortly.

Then we add `popup` as a child of `canvas_layer` so it's drawn on top. We then tell `popup` to pop up at the center of the screen using `popup_centered`.

Next we make sure the mouse mode is `MOUSE_MODE_VISIBLE` to we can interact with the pop up. If we did not do this, we would not be able to interact with the pop up in any scene where the mouse mode is `MOUSE_MODE_CAPTURED`.

Finally, get pause the entire *SceneTree*.

Note: For more information on pausing in Godot, see *Pausing games*

Now we need to add the functions we've connected the signals to. Let's add `popup_closed` first.

Add the following to `Globals.gd`:

```
func popup_closed():
    get_tree().paused = false

    if popup != null:
        popup.queue_free()
        popup = null
```

`popup_closed` will resume the game and destroy the pop up if there is one.

`popup_quit` is similar, but we're also making sure the mouse is visible and changing scenes to the title screen.

Add the following to `Globals.gd`:

```
func popup_quit():
    get_tree().paused = false

    Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)

```

(continues on next page)

(continued from previous page)

```
    if popup != null:
        popup.queue_free()
        popup = null

    load_new_scene(MAIN_MENU_PATH)
```

`popup_quit` will resume the game, set the mouse mode to `MOUSE_MODE_VISIBLE` to ensure the mouse is visible in the main menu, destroy the pop up if there is one, and change scenes to the main menu.

Before we're ready to test the pop up, we should change one thing in `Player.gd`.

Open up `Player.gd` and in `process_input`, change the code for capturing/freeing the cursor to the following:

```
if Input.get_mouse_mode() == Input.MOUSE_MODE_VISIBLE:
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)
```

Now instead of capturing/freeing the mouse, we check to see if the current mouse mode is `MOUSE_MODE_VISIBLE`. If it is, we set it back to `MOUSE_MODE_CAPTURED`.

Because the pop up makes the mouse mode `MOUSE_MODE_VISIBLE` whenever you pause, we no longer have to worry about freeing the cursor in `Player.gd`.

Now the pause menu pop up is finished. You can now pause at any point in the game and return to the main menu!

Starting the respawn system

Since our player can lose all their health, it would be ideal if our players died and respawned too, so let's add that!

First, open up `Player.tscn` and expand `HUD`. Notice how there's a *ColorRect* called `Death_Screen`. When the player dies, we're going to make `Death_Screen` visible, and show them how long they have to wait before they're able to respawn.

Open up `Player.gd` and add the following global variables:

```
const RESPAWN_TIME = 4
var dead_time = 0
var is_dead = false

var globals
```

- `RESPAWN_TIME`: The amount of time (in seconds) it takes to respawn.
- `dead_time`: A variable to track how long the player has been dead.
- `is_dead`: A variable to track whether or not the player is currently dead.
- `globals`: A variable to hold the `Globals.gd` singleton.

We now need to add a couple lines to `_ready`, so we can use `Globals.gd`. Add the following to `_ready`:

```
globals = get_node("/root/Globals")
global_transform.origin = globals.get_respawn_position()
```

Now we're getting the `Globals.gd` singleton and assigning it to `globals`. We also set our global position using the origin from our global *Transform* to the position returned by `globals.get_respawn_position`.

Note: Don't worry, we'll add `get_respawn_position` further below!

Next we need to make a few changes to `physics_process`. Change `physics_processing` to the following:

```
func _physics_process(delta):

    if !is_dead:
        process_input(delta)
        process_view_input(delta)
        process_movement(delta)

    if (grabbed_object == null):
        process_changing_weapons(delta)
        process_reloading(delta)

    process_UI(delta)
    process_respawn(delta)
```

Now we're not processing input or movement input when we're dead. We're also now calling `process_respawn`, but we haven't written `process_respawn` yet, so let's change that.

Let's add `process_respawn`. Add the following to `Player.gd`:

```
func process_respawn(delta):

    # If we just died
    if health <= 0 and !is_dead:
        $Body_CollisionShape.disabled = true
        $Feet_CollisionShape.disabled = true

        changing_weapon = true
        changing_weapon_name = "UNARMED"

        $HUD/Death_Screen.visible = true

        $HUD/Panel.visible = false
        $HUD/Crosshair.visible = false

        dead_time = RESPAWN_TIME
        is_dead = true

        if grabbed_object != null:
            grabbed_object.mode = RigidBody.MODE_RIGID
            grabbed_object.apply_impulse(Vector3(0,0,0), -camera.global_transform.
↪basis.z.normalized() * OBJECT_THROW_FORCE / 2)

            grabbed_object.collision_layer = 1
            grabbed_object.collision_mask = 1

            grabbed_object = null
```

(continues on next page)

(continued from previous page)

```
    if is_dead:
        dead_time -= delta

        var dead_time_pretty = str(dead_time).left(3)
        $HUD/Death_Screen/Label.text = "You died\n" + dead_time_pretty + " seconds␣
↪till respawn"

        if dead_time <= 0:
            global_transform.origin = globals.get_respawn_position()

            $Body_CollisionShape.disabled = false
            $Feet_CollisionShape.disabled = false

            $HUD/Death_Screen.visible = false

            $HUD/Panel.visible = true
            $HUD/Crosshair.visible = true

            for weapon in weapons:
                var weapon_node = weapons[weapon]
                if weapon_node != null:
                    weapon_node.reset_weapon()

            health = 100
            grenade_amounts = {"Grenade":2, "Sticky Grenade":2}
            current_grenade = "Grenade"

            is_dead = false
```

Let's go through what this function is doing.

First we check to see if we just died by checking to see if `health` is equal or less than `0` and `is_dead` is `false`.

If we just died, we disable our collision shapes for the player. We do this to make sure we're not blocking anything with our dead body.

We next set `changing_weapon` to `true` and set `changing_weapon_name` to `UNARMED`. This is so if we are using a weapon, we put it away when we die.

We then make the `Death_Screen` *ColorRect* visible so we get a nice grey overlay over everything. We then make the rest of the UI, the `Panel` and `Crosshair` nodes, invisible.

Next we set `dead_time` to `RESPAWN_TIME` so we can start counting down how long we've been dead. We also set `is_dead` to `true` so we know we've died.

If we are holding an object when we died, we need to throw it. We first check to see if we are holding an object or not. If we are, we then throw it, using the same code as the throwing code we added in *Part 5*.

Then we check to see if we are dead. If we are, we then remove `delta` from `dead_time`.

We then make a new variable called `dead_time_pretty`, where we convert `dead_time` to a string, using only the first three characters starting from the left. This gives us a nice looking string showing how much time we have left to wait before we respawn.

We then change the *Label* in `Death_Screen` to show how much time we have left.

Next we check to see if we've waited long enough and can respawn. We do this by checking to see if `dead_time` is `0` or less.

If we have waited long enough to respawn, we set the player's position to a new respawn position provided by `get_respawn_position`.

We then enable both of our collision shapes so the player can collide with the environment.

Next we make the `Death_Screen` invisible and make the rest of the UI, the `Panel` and `Crosshair` nodes, visible again.

We then go through each weapon and call it's `reset_weapon` function. We'll add `reset_weapon` soon.

Then we reset `health` to `100`, `grenade_amounts` to it's default values, and change `current_grenade` to `Grenade`.

Finally, we set `is_dead` to `false`.

Before we leave `Player.gd`, we need to add one quick thing to `_input`. Add the following at the beginning of `_input`:

```
if is_dead:
    return
```

Now when we're dead we cannot look around with the mouse.

Finishing the respawn system

First let's open `Weapon_Pistol.gd` and add the `reset_weapon` function. Add the following:

```
func reset_weapon():
    ammo_in_weapon = 10
    spare_ammo = 20
```

Now when we call `reset_weapon`, the ammo in our weapon and the ammo in the spares will be reset to their default values.

Now let's add `reset_weapon` in `Weapon_Rifle.gd`:

```
func reset_weapon():
    ammo_in_weapon = 50
    spare_ammo = 100
```

And add the following to `Weapon_Knife.gd`:

```
func reset_weapon():
    ammo_in_weapon = 1
    spare_ammo = 1
```

Now our weapons will reset when we die.

Now we need to add a few things to `Globals.gd`. First, add the following global variable:

```
var respawn_points = null
```

- `respawn_points`: A variable to hold all of the respawn points in a level

Because we're getting a random spawn point each time, we need to randomize the number generator. Add the following to `_ready`:

```
randomize()
```

`randomize` will get us a new random seed so we get a (relatively) random string of numbers when we using any of the random functions.

Now let's add `get_respawn_position` to `Globals.gd`:

```
func get_respawn_position():
    if respawn_points == null:
        return Vector3(0, 0, 0)
    else:
        var respawn_point = rand_range(0, respawn_points.size()-1)
        return respawn_points[respawn_point].global_transform.origin
```

Let's go over what this function does.

First we check to see if we have any `respawn_points` by checking to see if `respawn_points` is `null` or not.

If `respawn_points` is `null`, we return a position of empty *Vector 3* with the position `(0, 0, 0)`.

If `respawn_points` is not `null`, we then get a random number between `0` and the number of elements we have in `respawn_points`, minus `1` since most programming languages (including `GDScript`) start counting from `0` when you are accessing elements in a list.

We then return the position of the *Spatial* node at `respawn_point` position in `respawn_points`.

Before we're done with `Globals.gd`. We need to add the following to `load_new_scene`:

```
respawn_points = null
```

We set `respawn_points` to `null` so when/if we get to a level with no respawn points, we do not respawn at the respawn points in the level prior.

Now all we need is a way to set the respawn points. Open up `Ruins_Level.tscn` and select `Spawn_Points`. Add a new script called `Respawn_Point_Setter.gd` and attach it to `Spawn_Points`. Add the following to `Respawn_Point_Setter.gd`:

```
extends Spatial

func _ready():
    var globals = get_node("/root/Globals")
    globals.respawn_points = get_children()
```

Now when a node with `Respawn_Point_Setter.gd` has it's `_ready` function called, all of the children nodes of the node with `Respawn_Point_Setter.gd`, `Spawn_Points` in the case of `Ruins_Level.tscn`, will be added to `respawn_points` in `Globals.gd`.

Warning: Any node with `Respawn_Point_Setter.gd` has to be above the player in the *SceneTree* so the respawn points are set before the player needs them in the player's `_ready` function.

Now when you die you'll respawn after waiting `4` seconds!

Note: No spawn points are already set up for any of the levels besides `Ruins_Level.tscn`! Adding spawn points to `Space_Level.tscn` is left as an exercise for the reader.

Writing a sound system we can use anywhere

Finally, lets make a sound system so we can play sounds from anywhere, without having to use the player.

First, open up `SimpleAudioPlayer.gd` and change it to the following:

```
extends Spatial

var audio_node = null
var should_loop = false
var globals = null

func _ready():
    audio_node = $Audio_Stream_Player
    audio_node.connect("finished", self, "sound_finished")
    audio_node.stop()

    globals = get_node("/root/Globals")


func play_sound(audio_stream, position=null):
    if audio_stream == null:
        print ("No audio stream passed, cannot play sound")
        globals.created_audio.remove(globals.created_audio.find(self))
        queue_free()
        return

    audio_node.stream = audio_stream

    # If you are using a AudioPlayer3D, then uncomment these lines to set the␣
↪position.
    # if position != null:
    #       audio_node.global_transform.origin = position

    audio_node.play(0.0)


func sound_finished():
    if should_loop:
        audio_node.play(0.0)
    else:
        globals.created_audio.remove(globals.created_audio.find(self))
        audio_node.stop()
        queue_free()
```

There's several changes from the old version, first and foremost being we're no longer storing the sound files in `SimpleAudioPlayer.gd` anymore. This is much better for performance since we're no longer loading each audio clip when we create a sound, but instead we're forcing a audio stream to be passed in to `play_sound`.

Another change is we have a new global variable called `should_loop`. Instead of just destroying the audio player every time it's finished, we instead want check to see if we are set to loop or not. This allows us to have audio like looping background music without having to spawn a new audio player with the music when the old one is finished.

Finally, instead of being instanced/spawned in `Player.gd`, we're instead going to be spawned in `Globals.gd` so we can create sounds from any scene. We now need to store the `Globals.gd` singleton so when we destroy the audio player, we also remove it from a list in `Globals.gd`.

Let's go over the changes.

For the global variables we removed all of the `audio_[insert name here]` variables since we will instead have these passed in to. We also added two new global variables, `should_loop` and `globals`. We'll use `should_loop` to tell whether we want to loop when the sound has finished, and `globals` will hold the `Globals.gd` singleton.

The only change in `_ready` is now we're getting the `Globals.gd` singleton and assigning it to `globals`

In `play_sound` we now expect a audio stream, named `audio_stream`, to be passed in, instead of `sound_name`. Instead of checking the sound name and setting the stream for the audio player, we instead check to make sure an audio stream was passed in. If a audio stream is not passed in, we print an error message, remove the audio player from a list in the `Globals.gd` singleton called `created_audio`, and then free the audio player.

Finally, in `sound_finished` we first check to see if we are supposed to loop or not using `should_loop`. If we are supposed to loop, we play the sound again from the start of the audio, at position `0.0`. If we are not supposed to loop, we remove the audio player from a list in the `Globals.gd` singleton called `created_audio`, and then free the audio player.

Now that we've finished our changes to `SimpleAudioPlayer.gd`, we now need to turn our attention to `Globals.gd`. First, add the following global variables:

```
# -------------------------------------
# All of the audio files.

# You will need to provide your own sound files.
var audio_clips = {
    "pistol_shot":null, #preload("res://path_to_your_audio_here!")
    "rifle_shot":null, #preload("res://path_to_your_audio_here!")
    "gun_cock":null, #preload("res://path_to_your_audio_here!")
}

const SIMPLE_AUDIO_PLAYER_SCENE = preload("res://Simple_Audio_Player.tscn")
var created_audio = []
# -------------------------------------
```

Lets go over these global variables.

- `audio_clips`: A dictionary holding all of the audio clips we can play.
- `SIMPLE_AUDIO_PLAYER_SCENE`: The simple audio player scene.
- `created_audio`: A list to hold all of the simple audio players we create

---

Note: If you want to add additional audio, you need to add it to `audio_clips`. No audio files are provided in this tutorial, so you will have to provide your own.

One site I'd recommend is **GameSounds.xyz**. I'm using the Gamemaster audio gun sound pack included in the Sonniss' GDC Game Audio bundle for 2017. The tracks I've used (with some minor editing) are as follows:

- gun_revolver_pistol_shot_04,
- gun_semi_auto_rifle_cock_02,
- gun_submachine_auto_shot_00_automatic_preview_01

Now we need to add a new function called `play_sound` to `Globals.gd`:

```
func play_sound(sound_name, loop_sound=false, sound_position=null):
    if audio_clips.has(sound_name):
        var new_audio = SIMPLE_AUDIO_PLAYER_SCENE.instance()
        new_audio.should_loop = loop_sound

        add_child(new_audio)
        created_audio.append(new_audio)

        new_audio.play_sound(audio_clips[sound_name], sound_position)

    else:
        print ("ERROR: cannot play sound that does not exist in audio_clips!")
```

Let's go over what this script does.

First we check to see if we have a audio clip with the name `sound_name` in `audio_clips`. If we do not, we print an error message.

If we do have a audio clip with the name `sound_name`, we then instance/spawn a new `SIMPLE_AUDIO_PLAYER_SCENE` and assign it to `new_audio`.

We then set `should_loop`, and add `new_audio` as a child of `Globals.gd`.

Note: Remember, we have to be careful adding nodes to a singleton, since these nodes will not be destroyed when changing scenes.

We then call `play_sound`, passing in the audio clip associated with `sound_name`, and the sound position.

Before we leave `Globals.gd`, we need to add a few lines of code to `load_new_scene` so when we change scenes, we destroy all of the audio.

Add the following to `load_new_scene`:

```
for sound in created_audio:
    if (sound != null):
        sound.queue_free()
created_audio.clear()
```

Now before we change scenes we go through each simple audio player in `created_sounds` and free/destroy them. Once we've gone through all of the sounds in `created_audio`, we clear `created_audio` so it no longer holds any references to any of the previously created simple audio players.

Let's change `create_sound` in `Player.gd` to use this new system. First, remove `simple_audio_player` from `Player.gd`'s global variables, since we will no longer be directly instancing/spawning sounds from `Player.gd`.

Now, change `create_sound` to the following:

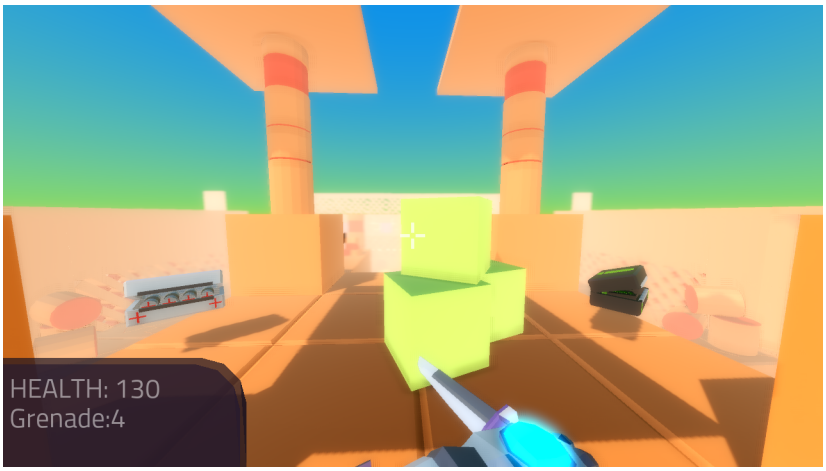
```
func create_sound(sound_name, position=null):
    globals.play_sound(sound_name, false, position)
```

Now whenever `create_sound` is called, we simply call `play_sound` in `Globals.gd`, passing in all of the arguments we've revived.

Now all of the sounds in our FPS can be played from anywhere. All we have to do is get the `Globals.gd` singleton, and call `play_sound`, passing in the name of the sound we want to play, whether we want it to loop or not, and the position to play the sound from.

For example, if you want to play an explosion sound when the grenades explode you'd need to add a new sound to `audio_clips` in `Globals.gd`, get the `Globals.gd` singleton, and then you just need to add something like `globals.play_sound("explosion", false, global_transform.origin)` in the grenades `_process` function, right after the grenade damages all of the bodies within it's blast radius.

Final notes



Now you have a fully working single player FPS!

At this point you have a good base to build more complicated FPS games.

> **Warning:** If you ever get lost, be sure to read over the code again!
>
> You can download the finished project for the entire tutorial here: `Godot_FPS_Part_6.zip`

Note: The finished project source files contain the same exact code, just written in a different order. This is because the finished project source files are what the tutorial is based on.

The finished project code was written in the order that features were created, not necessarily in a order that is ideal for learning.

Other than that, the source is exactly the same, just with helpful comments explaining what each part does.

Tip: The finished project source is hosted on Github as well: https://github.com/TwistedTwigleg/Godot_FPS_Tutorial

Please note that the code in Github may or may not be in sync with the tutorial on the documentation.

The code in the documentation is likely better managed and/or more up to date. If you are unsure on which to use, use the project(s) provided in the documentation as they are maintained by the Godot community.

You can download all of the `.blend` files used in this tutorial here: `Godot_FPS_BlenderFiles.zip`

All assets provided in the started assets (unless otherwise noted) were **originally created by TwistedTwigleg, with changes/additions by the Godot community.** All original assets provided for this tutorial are released under the `MIT` license.

Feel free to use these assets however you want! All original assets belong to the Godot community, with the other assets belonging to those listed below:

The skybox is created by **StumpyStrust** and can be found at OpenGameArt.org. https://opengameart.org/content/space-skyboxes-0 . The skybox is licensed under the `CC0` license.

The font used is **Titillium-Regular**, and is licensed under the `SIL Open Font License, Version 1.1`.

The skybox was convert to a 360 equirectangular image using this tool: https://www.360toolkit.co/convert-cubemap-to-spherical-equirectangular.html

While no sounds are provided, you can find many game ready sounds at https://gamesounds.xyz/

Warning: OpenGameArt.org, 360toolkit.co, the creator(s) of Titillium-Regular, StumpyStrust, and GameSounds.xyz are in no way involved in this tutorial.

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 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:

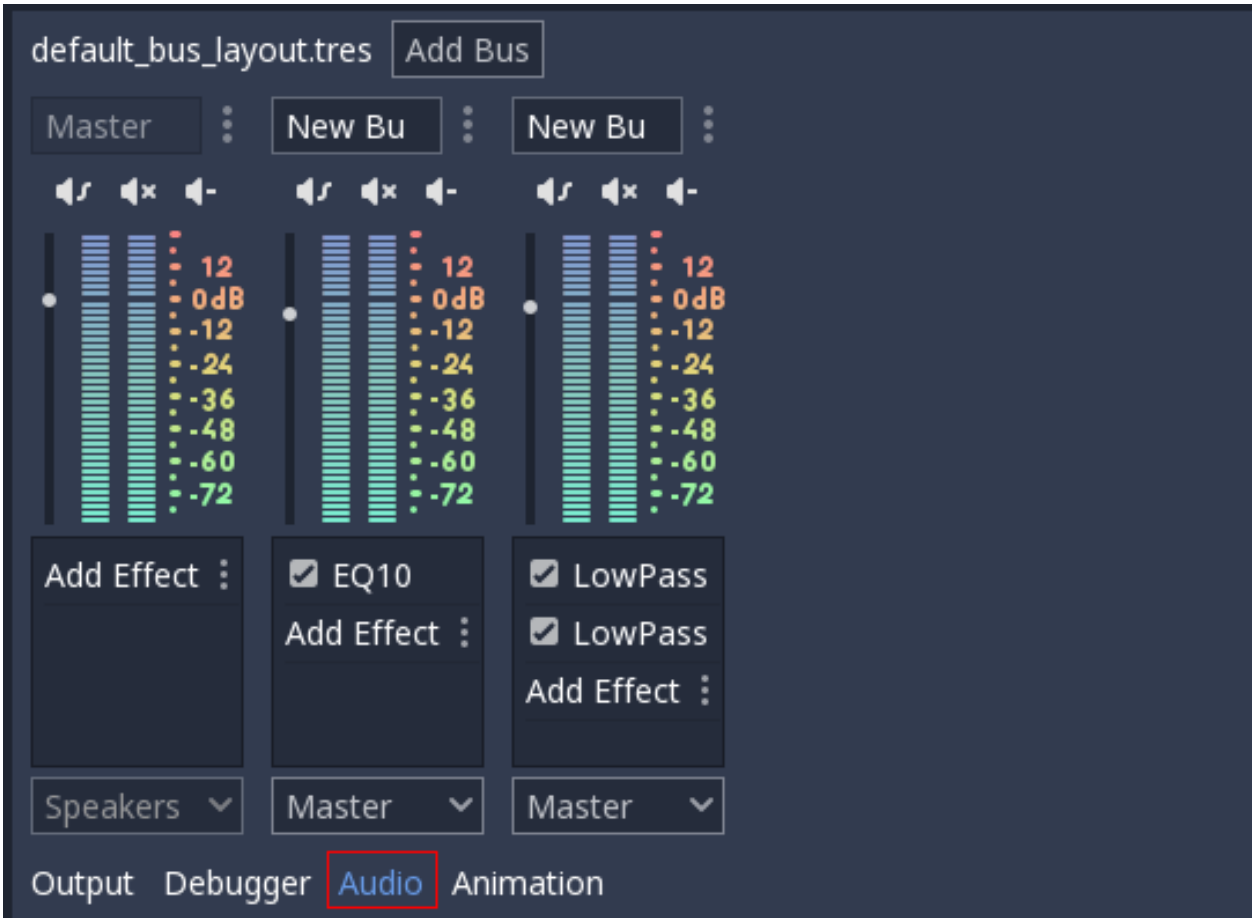
- The decibel (dB) scale is a relative scale. It represents the ratio of sound power by using 10 times the base 10 logarithm of the ratio ($10 \times \log_{10}(P/P_0)$).
- For every 3dB, sound doubles or halves. 6dB represents a factor 4, 9dB a factor 8, 10dB a factor 10, 20dB a factor 100, etc.
- Since 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:



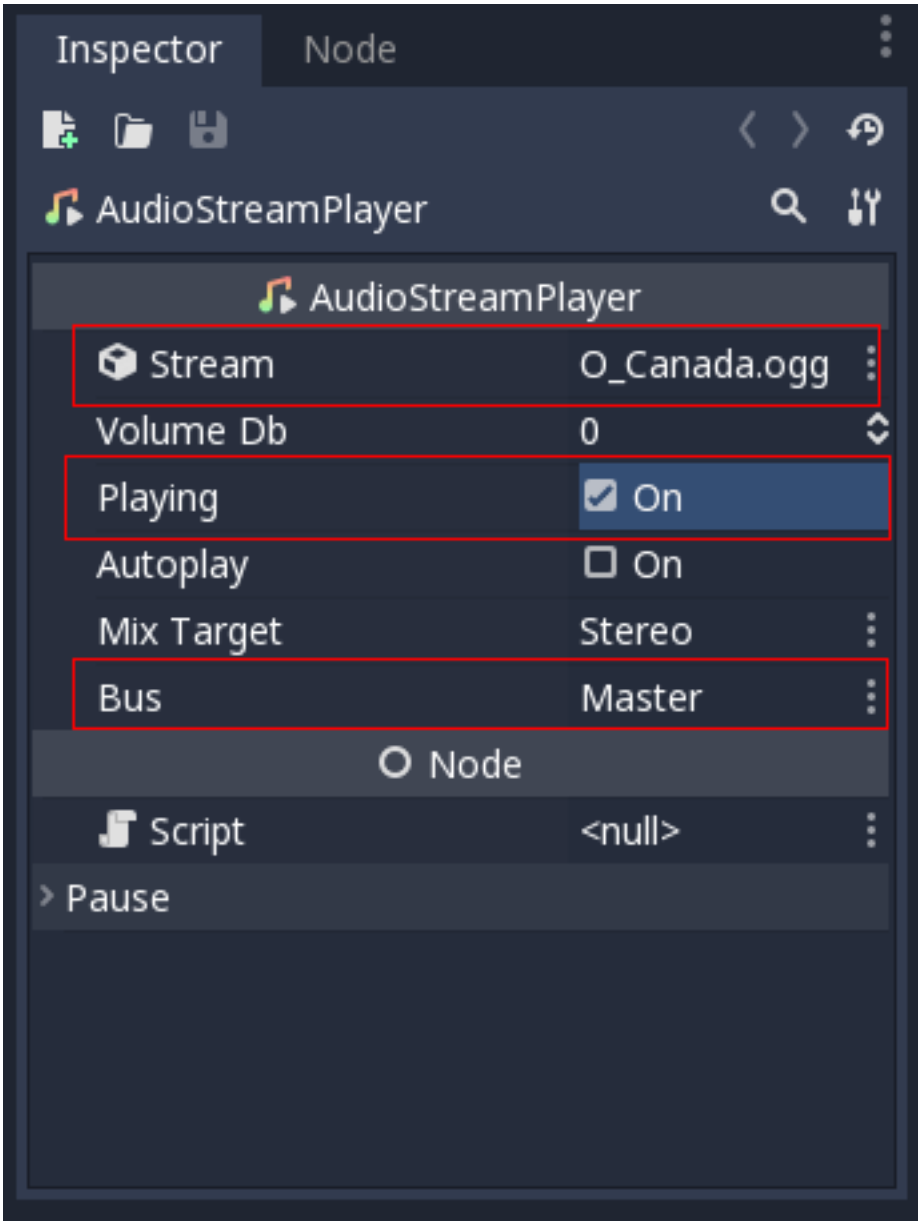
An *Audio Bus* bus (often called "Audio Channels" too) is a device where audio is channeled. Audio data passes through it and can be *modified* and *re-routed*. A VU-Meter (the bars that go up and down when sound is played) can measure the loudness of the sound in Decibel scale.

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:

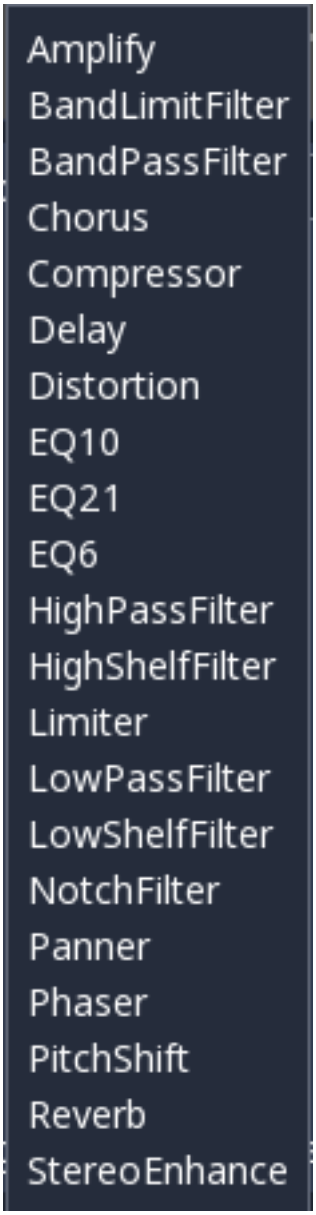


Finally toggle the "playing" property to on and sound will flow.

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.



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 may 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 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 efficiently.

EQ6, EQ10, EQ21

Godot provides three model of equalizers with different band counts. Equalizers are useful on the Master Bus to completely master a mix and give it character. They are also 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 too 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.

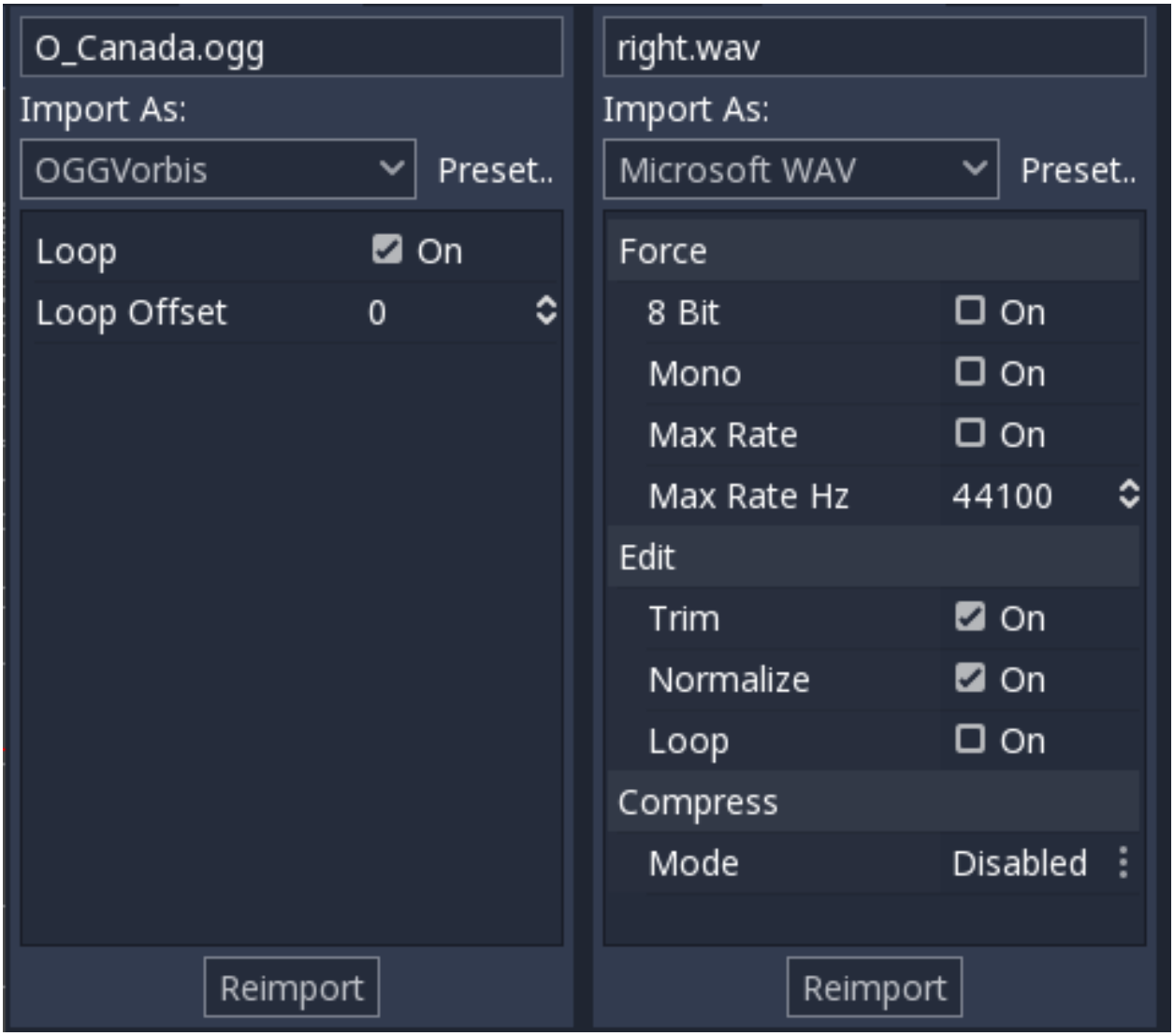
8.2.2 AudioStream

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 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.

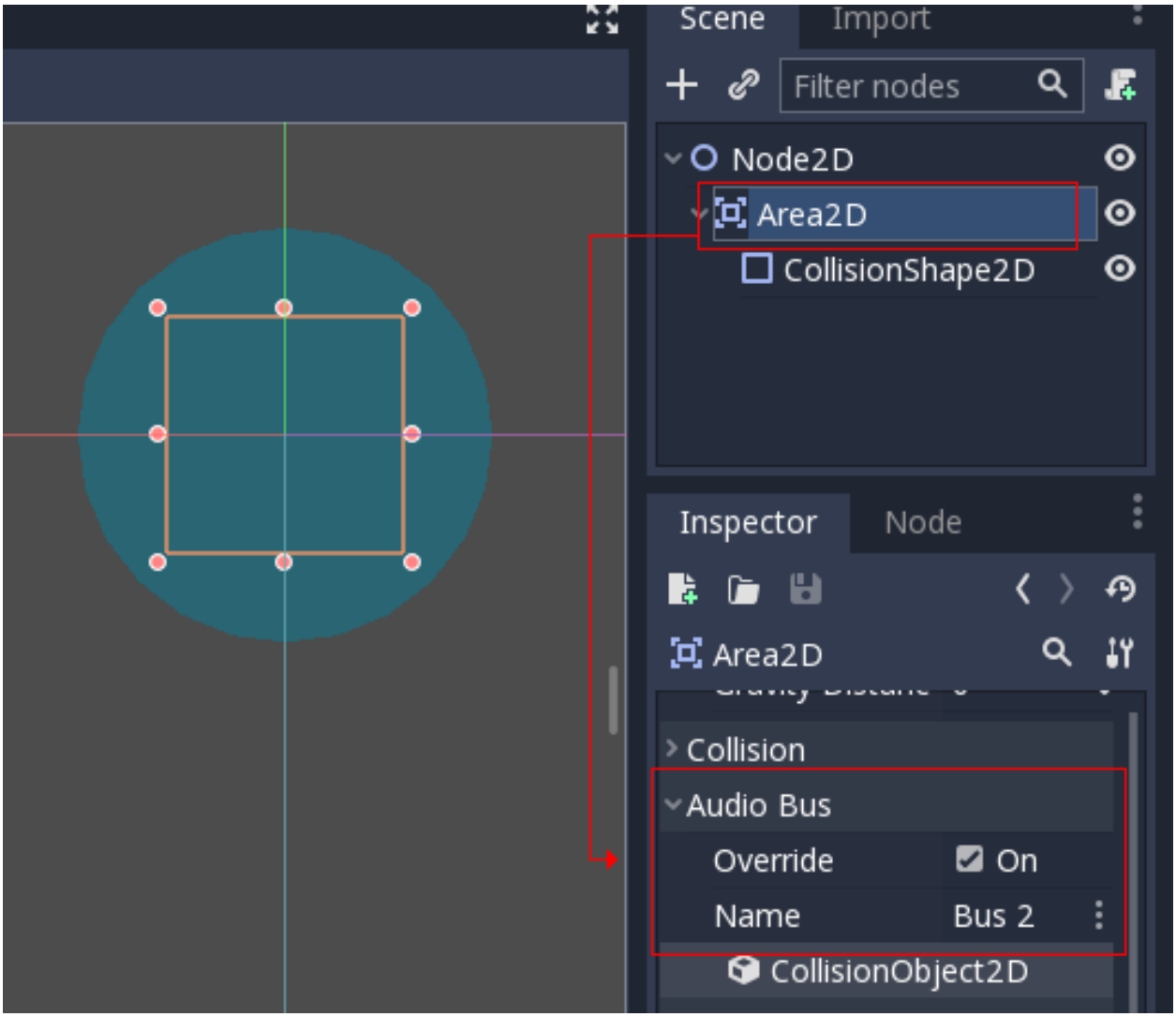
8.2.3 AudioStreamPlayer

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.

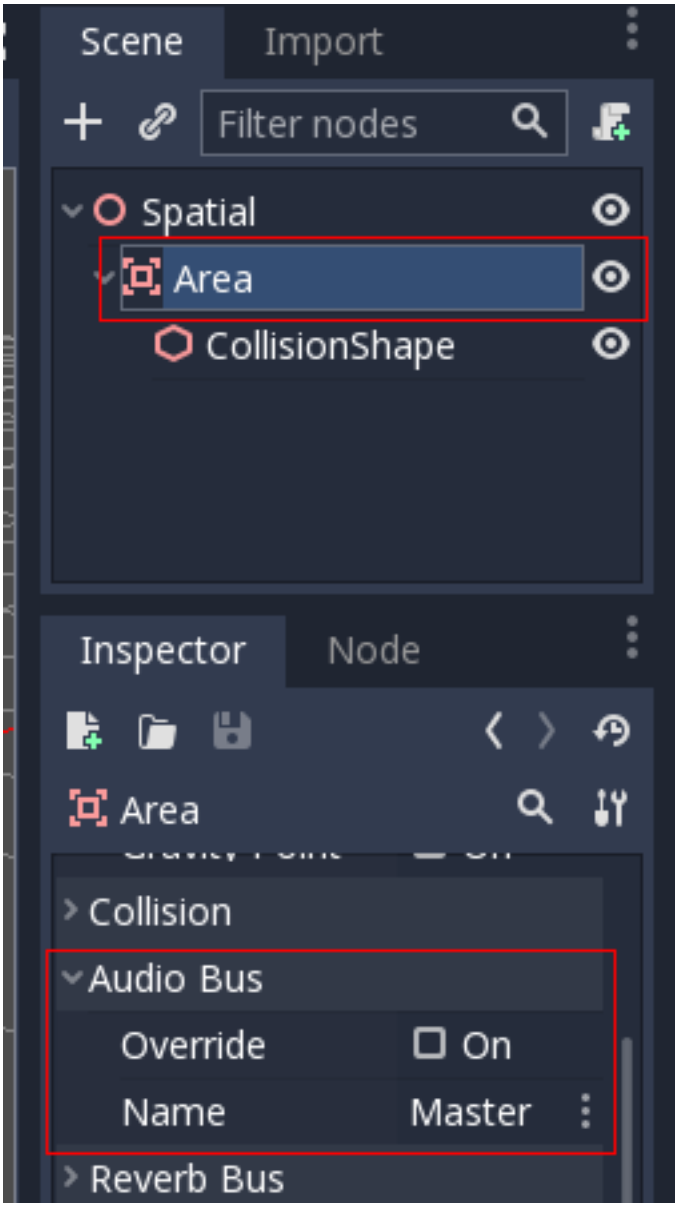
While it's possible to send these effects to specific audio buses, one of the best strategies is to use an Area2D to divert sound to a specific bus. This allows to create buses with different reverb or sound qualities and make the emitter will automatically send to them when entering the Area2D shapes.



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.

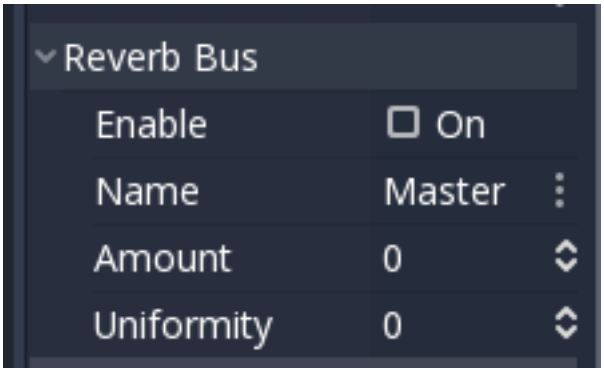
Similar to AudioStreamPlayer2D, an Area can divert the sound to an audio bus.



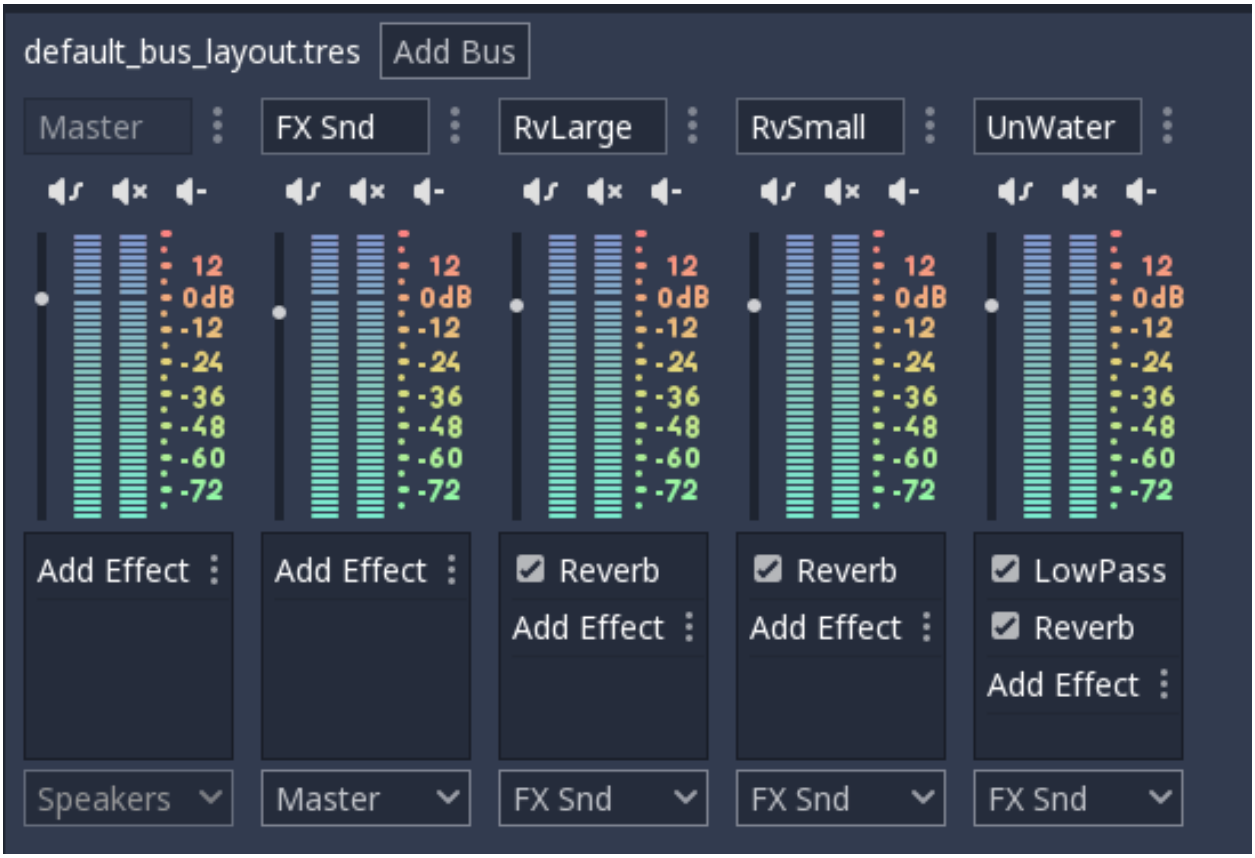
Unlike for 2D, the 3D version of AudioStreamPlayer has a few more advanced options:

Reverb Buses

Godot allows 3D Audio Streams that enter a specific *Area* to send dry and wet audio to separate buses. This is 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:



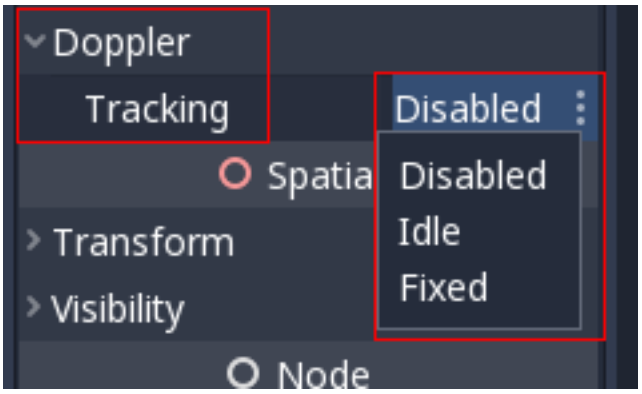
At the same time, a special bus layout is created where each area receives the reverb info from each area. Of course, an actual Reverb effect must be created in that bus for anything to happen:



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!

CHAPTER 9

Physics

9.1 Physics introduction

In game development you often need to know when two objects in the game intersect or come into contact. This is known as **collision detection**. When a collision is detected, you typically want something to happen. This is known as **collision response**.

Godot offers a number of collision objects in 2D and 3D to provide both collision detection and response. Trying to decide which one to use for your project can be confusing. You can avoid problems and simplify development if you understand how each works and what their pros and cons are.

In this guide you will learn:

- Godot's four collision object types
- How each collision object works
- When and why to choose one type over another

Note: This document's examples will use 2D objects. Every 2D physics object and collision shape has a direct equivalent in 3D and in most cases they work in much the same way.

9.1.1 Collision Objects

Godot offers four kinds of physics bodies, extending *CollisionObject2D*:

- ***Area2D*** `Area2D` nodes provide **detection** and **influence**. They can detect when objects overlap and can emit signals when bodies enter or exit. An `Area2D` can also be used to override physics properties such as gravity or damping in a defined area.

The other three bodies extend from *PhysicsBody2D*:

- ***StaticBody2D*** A static body is one that is not moved by the physics engine. It participates in collision detection, but does not move in response to the collision. They are most often used for objects that are part of the environment or that do not need to have any dynamic behavior.
- ***RigidBody2D*** This is the node that implements simulated 2D physics. You do not control a `RigidBody2D` directly, but instead you apply forces to it (gravity, impulses, etc.) and the physics engine calculates the resulting movement. *Read more about using rigid bodies.*
- ***KinematicBody2D*** A body that provides collision detection, but no physics. All movement and collision response must be implemented in code.

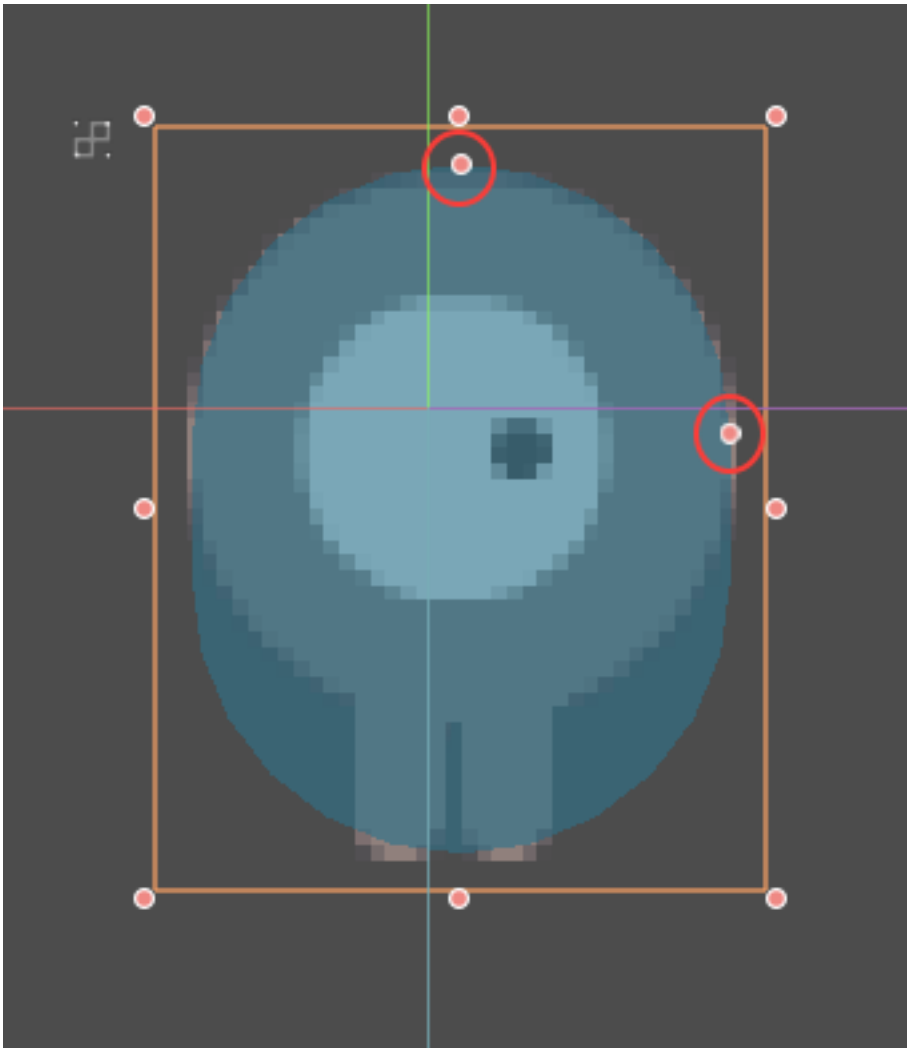
Collision Shapes

A physics body can hold any number of *Shape2D* objects as children. These shapes are used to define the object's collision bounds and to detect contact with other objects.

Note: In order to detect collisions, at least one `Shape2D` must be assigned to the object.

The most common way to assign a shape is by adding a *CollisionShape2D* or *CollisionPolygon2D* as a child of the object. These nodes allow you to draw the shape directly in the editor workspace.

Important: Be careful to never scale your collision shapes in the editor. The "Scale" property in the Inspector should remain `(1, 1)`. When changing sizing the collision shape, you should always use the size handles, **not** the `Node2D` scale handles. Scaling a shape can result in unexpected collision behavior.



Physics 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, the value of a body's state variables such as `position` or `linear velocity` may not be accurate for the current frame.

In order to avoid this inaccuracy, any code that needs to access a body's properties should be run in the *Node._physics_process()* callback, which is called before each physics step at a constant frame rate (60 times per second by default).

Collision Layers and Masks

One of the most powerful but frequently misunderstood collision features is the collision layer system. This system allows you to build up complex interactions between a variety of objects. The key concepts are **layers** and **masks**. Each `CollisionObject2D` has 20 different physics layers it can interact with.

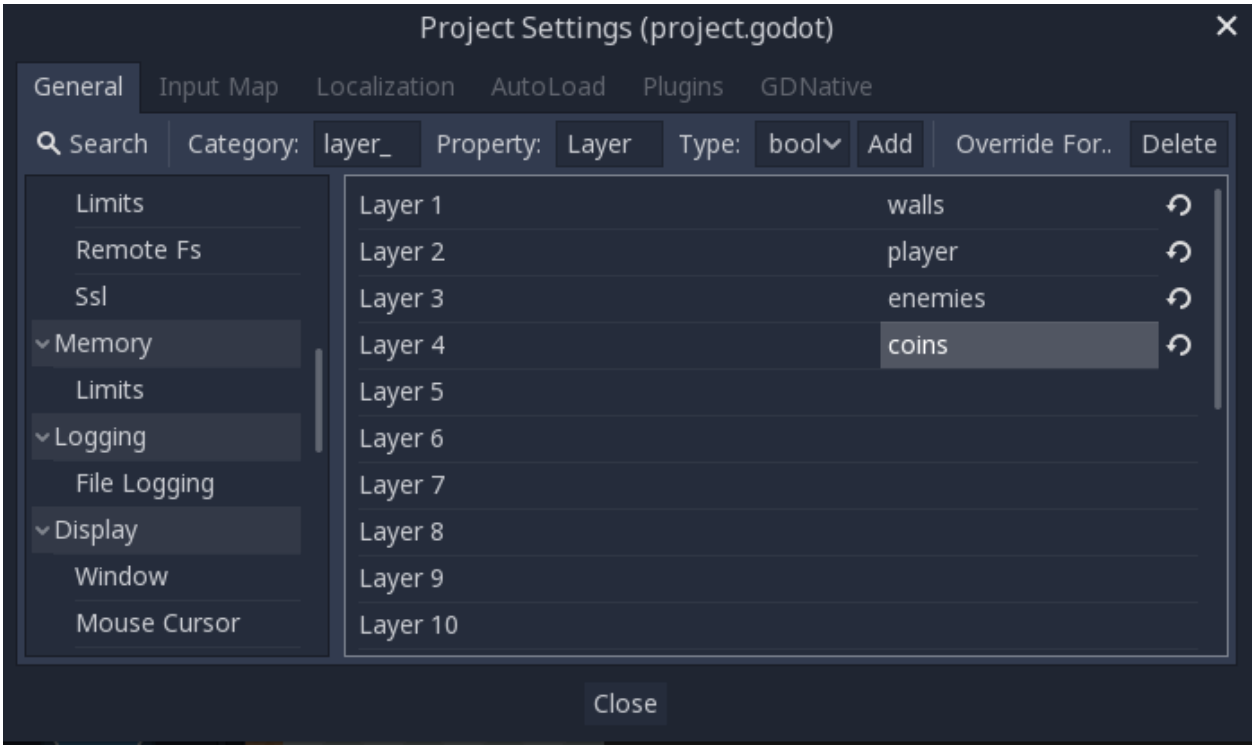
Let's look at each of the properties in turn:

- **collision_layer** This describes the layers that the object appears **in**. By default, all bodies are on layer `1`.

- **collision_mask** This describes what layers the body will **scan** for collisions. If an object isn't in one of the mask layers, the body will ignore it. By default, all bodies scan layer `1`.

These properties can be configured via code, or by editing them in the Inspector.

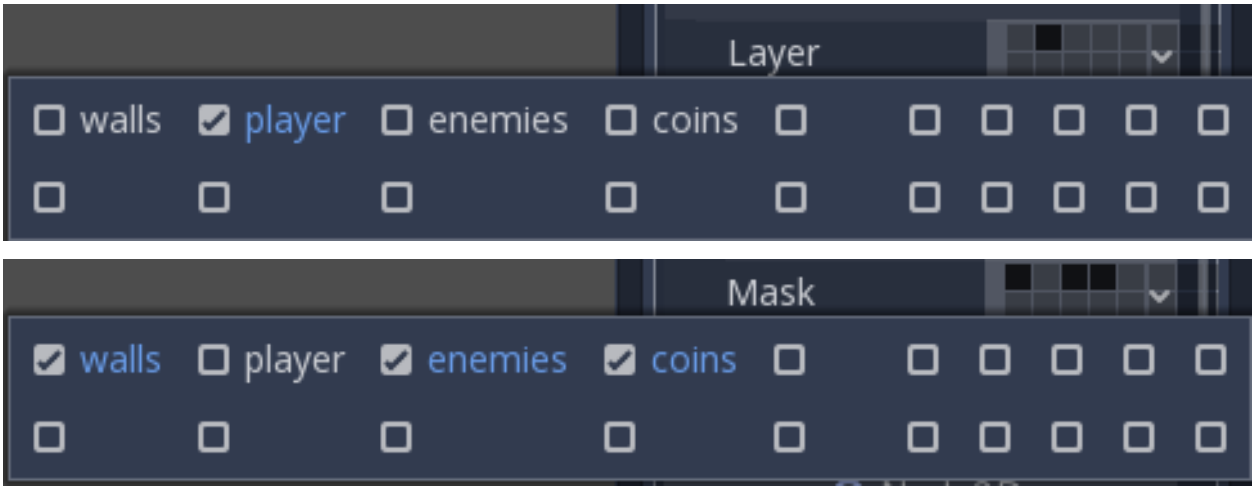
Keeping track of what you're using each layer for can be difficult, so you may find it useful to assign names to the layers you're using. Names can be assigned in Project Settings -> Layer Names.



Example:

You have four node types in your game: Walls, Player, Enemy, and Coin. Both Player and Enemy should collide with Walls. The Player node should detect collisions with both Enemy and Coin, but Enemy and Coin should ignore each other.

Start by naming layers 1-4 "walls", "player", "enemies", and "coins" and place each node type in its respective layer using the "Layer" property. Then set each node's "Mask" property by selecting the layers it should interact with. For example, the Player's settings would look like this:



9.1.2 Area2D

Area nodes provide **detection** and **influence**. They can detect when objects overlap and emit signals when bodies enter or exit. Areas can also be used to override physics properties such as gravity or damping in a defined area.

There are three main uses for *Area2D*:

- Overriding physics parameters such as gravity in a given region.
- Detecting when other bodies enter or exit a region or what bodies are currently in a region.
- Checking other areas for overlap.

By default, areas also receive mouse and touchscreen input.

9.1.3 StaticBody2D

A static body is one that is not moved by the physics engine. It participates in collision detection, but does not move in response to the collision. However, it can impart motion or rotation to a colliding body **as if** it were moving, using its `constant_linear_velocity` and `constant_angular_velocity` properties.

`StaticBody2D` nodes are most often used for objects that are part of the environment or that do not need to have any dynamic behavior.

Example uses for `StaticBody2D`:

- Platforms (including moving platforms)
- Conveyor belts
- Walls and other obstacles

9.1.4 RigidBody2D

This is the node that implements simulated 2D physics. You do not control a *RigidBody2D* directly. Instead you apply forces to it and the physics engine calculates the resulting movement, including collisions with other bodies, and collision responses such as bouncing, rotating, etc.

You can modify a rigid body's behavior via properties such as "Mass", "Friction", or "Bounce", which can be set in the Inspector.

The body's behavior is also affected by the world's properties, as set in *Project Settings* -> *Physics*, or by entering an *Area2D* that is overriding the global physics properties.

When a rigid body is at rest and hasn't moved for a time, it goes to sleep. A sleeping body acts like a static body, and its forces are not calculated by the physics engine. The body will wake up when forces are applied, either by a collision or via code.

Rigid body modes

A rigid body can be set to one of four modes:

- **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.

Using RigidBody2D

One of the benefits of using a rigid body is that a lot of behavior can be had "for free" without writing any code. For example, if you were making an "Angry Birds"-style game with falling blocks, you would only need to create RigidBody2Ds and adjust their properties. Stacking, falling, and bouncing would automatically be calculated by the physics engine.

However, if you do wish to have some control over the body, you should take care - altering the `position`, `linear_velocity`, or other physics properties of a rigid body can result in unexpected behavior. If you need to alter any of the physics-related properties, you should use the *_integrate_forces()* callback instead of `_physics_process()`. In this callback, you have access to the body's *Physics2DDirectBodyState*, which allows for safely changing properties and synchronizing them with the physics engine.

For example, here is the code for an "Asteroids" style spaceship:

GDScript

C#

```
extends RigidBody2D

var thrust = Vector2(0, 250)
var torque = 20000

func _integrate_forces(state):
    if Input.is_action_pressed("ui_up"):
        set_applied_force(thrust.rotated(rotation))
    else:
        set_applied_force(Vector2())
    var rotation_dir = 0
    if Input.is_action_pressed("ui_right"):
        rotation_dir += 1
    if Input.is_action_pressed("ui_left"):
        rotation_dir -= 1
    set_applied_torque(rotation_dir * torque)
```

```
class Spaceship : RigidBody2D
{
    private Vector2 thrust = new Vector2(0, 250);
    private float torque = 20000;

    public override void _IntegrateForces(Physics2DDirectBodyState state)
    {
        if (Input.IsActionPressed("ui_up"))
            SetAppliedForce(thrust.Rotated(Rotation));
        else
            SetAppliedForce(new Vector2());

        var rotationDir = 0;
        if (Input.IsActionPressed("ui_right"))
            rotationDir += 1;
        if (Input.IsActionPressed("ui_left"))
            rotationDir -= 1;
        SetAppliedTorque(rotationDir * torque);
    }
}
```

Note that we are not setting the `linear_velocity` or `angular_velocity` properties directly, but rather applying forces (`thrust` and `torque`) to the body and letting the physics engine calculate the resulting movement.

Note: When a rigid body goes to sleep the `_integrate_forces()` function will not be called. To override this behavior you will need to keep the body awake by creating a collision, applying a force to it, or by disabling the *can_sleep* property. Be aware that this can have a negative effect on performance.

Contact reporting

By default, rigid bodies do not keep track of contacts, because this can require a huge amount of memory if many bodies are in the scene. To enable contact reporting, set the *contacts_reported* property to a non-zero value. The contacts can then be obtained via *Physics2DDirectBodyState.get_contact_count()* and related functions.

Contact monitoring via signals can be enabled via the *contact_monitor* property. See *RigidBody2D* for the list of available signals.

9.1.5 KinematicBody2D

KinematicBody2D bodies detect collisions with other bodies, but are not affected by physics properties like gravity or friction. Instead, they must be controlled by the user via code. The physics engine will not move a kinematic body.

When moving a kinematic body, you should not set its `position` directly. Instead, you use the `move_and_collide()` or `move_and_slide()` methods. These methods move the body along a given vector, and it will instantly stop if a collision is detected with another body. After the body has collided, any collision response must be coded manually.

Kinematic collision response

After a collision, you may want the body to bounce, to slide along a wall, or to alter the properties of the object it hit. The way you handle collision response depends on which method you used to move the KinematicBody2D.

move_and_collide

When using `move_and_collide()`, the function returns a *KinematicCollision2D* object, which contains information about the collision and the colliding body. You can use this information to determine the response.

For example, if you want to find the point in space where the collision occurred:

GDScript

C#

```
extends KinematicBody2D

var velocity = Vector2(250, 250)

func _physics_process(delta):
    var collision_info = move_and_collide(velocity * delta)
    if collision_info:
        var collision_point = collision_info.position
```

```
class Body : KinematicBody2D
{
    private Vector2 velocity = new Vector2(250, 250);
```

(continues on next page)

(continued from previous page)

```
    public override void _PhysicsProcess(float delta)
    {
        var collisionInfo = MoveAndCollide(velocity * delta);
        if (collisionInfo != null)
        {
            var collisionPoint = collisionInfo.GetPosition();
        }
    }
}
```

Or to bounce off of the colliding object:

GDScript

C#

```
extends KinematicBody2D

var velocity = Vector2(250, 250)

func _physics_process(delta):
    var collision_info = move_and_collide(velocity * delta)
    if collision_info:
        velocity = velocity.bounce(collision_info.normal)
```

```
class Body : KinematicBody2D
{
    private Vector2 velocity = new Vector2(250, 250);

    public override void _PhysicsProcess(float delta)
    {
        var collisionInfo = MoveAndCollide(velocity * delta);
        if (collisionInfo != null)
            velocity = velocity.Bounce(collisionInfo.Normal);
    }
}
```

move_and_slide

Sliding is a common collision response; imagine a player moving along walls in a top-down game or running up and down slopes in a platformer. While it's possible to code this response yourself after using `move_and_collide()`, `move_and_slide()` provides a convenient way to implement sliding movement without writing much code.

> **Warning:** `move_and_slide()` automatically includes the timestep in its calculation, so you should **not** multiply the velocity vector by `delta`.

For example, use the following code to make a character that can walk along the ground (including slopes) and jump when standing on the ground:

GDScript

C#

```
extends KinematicBody2D

var run_speed = 350
var jump_speed = -1000
var gravity = 2500

var velocity = Vector2()

func get_input():
    velocity.x = 0
    var right = Input.is_action_pressed('ui_right')
    var left = Input.is_action_pressed('ui_left')
    var jump = Input.is_action_just_pressed('ui_select')

    if is_on_floor() and jump:
            velocity.y = jump_speed
    if right:
            velocity.x += run_speed
    if left:
            velocity.x -= run_speed

func _physics_process(delta):
    velocity.y += gravity * delta
    get_input()
    velocity = move_and_slide(velocity, Vector2(0, -1))
```

```
class Body : KinematicBody2D
{
    private float runSpeed = 350;
    private float jumpSpeed = -1000;
    private float gravity = 2500;

    private Vector2 velocity = new Vector2();

    private void getInput()
    {
        velocity.x = 0;

        var right = Input.IsActionPressed("ui_right");
        var left = Input.IsActionPressed("ui_left");
        var jump = Input.IsActionPressed("ui_select");

        if (IsOnFloor() && jump)
            velocity.y = jumpSpeed;
        if (right)
            velocity.x += runSpeed;
        if (left)
            velocity.x -= runSpeed;
    }

    public override void _PhysicsProcess(float delta)
    {
        velocity.y += gravity * delta;
    }
}
```

See *Kinematic Character (2D)* for more details on using `move_and_slide()`, including a demo project with detailed code.

9.2 Rigid Body

9.2.1 What is a rigid body?

A rigid body is one that is directly controlled by the physics engine in order to simulate the behavior of physical objects. In order to define the shape of the body, it must have one or more *Shape* objects assigned. Note that setting the position of these shapes will affect the body's center of mass.

9.2.2 How to control rigid body

A rigid body's behavior can be altered by setting its properties such as friction, mass, bounce, etc. These properties can be set in the Inspector or via code. See *RigidBody* for the full list of properties and their effects.

There are several ways to control a rigid body's movement, depending on your desired application.

If you only need to place a rigid body once, for example to set its initial location, you can use the methods provided by the *Spatial* node, such as `set_global_transform()` or `look_at()`. However, these functions can not be called every frame or the physics engine will not be able to correctly simulate the body's state. As an example, consider a rigid body that you want to rotate so that it points towards another object. A common mistake when implementing this kind of behavior is to use `look_at()` every frame, which breaks the physics simulation. Below, we'll demonstrate how to implement this correctly.

The fact that you can't use `set_global_transform()` or `look_at()` methods doesn't mean that you can't have full control of a rigid body. Instead, you can control it by using the `_integrate_forces()` callback. In this function, you can add *forces*, apply *impulses*, or set the *velocity* in order to achieve any movement you desire.

9.2.3 Look at function

As described above, using the Spatial node's `look_at()` function can't be used each frame to follow a target. Here is a custom `look_at()` function that will work reliably with rigid bodies:

GDScript

C#

```
extends RigidBody

func look_follow(state, current_transform, target_position):
    var up_dir = Vector3(0, 1, 0)
    var cur_dir = current_transform.basis.xform(Vector3(0, 0, 1))
    var target_dir = (target_position - current_transform.origin).normalized()
    var rotation_angle = acos(cur_dir.x) - acos(target_dir.x)

    state.set_angular_velocity(up_dir * (rotation_angle / state.get_step()))

func _integrate_forces(state):
    var target_position = $my_target_spatial_node.get_global_transform().origin
    look_follow(state, get_global_transform(), target_position)
```

```
class Body : RigidBody
{
    private void lookFollow(PhysicsDirectBodyState state, Transform currentTransform,␣
↪Vector3 targetPosition)
    {
```

(continues on next page)

(continued from previous page)

```
        var upDir = new Vector3(0, 1, 0);
        var curDir = currentTransform.basis.Xform(new Vector3(0, 0, 1));
        var targetDir = (targetPosition - currentTransform.origin).Normalized();
        var rotationAngle = Mathf.Acos(curDir.x) - Mathf.Acos(targetDir.x);

        state.SetAngularVelocity(upDir * (rotationAngle / state.GetStep()));
    }

    public override void _IntegrateForces(PhysicsDirectBodyState state)
    {
        var targetPosition = (GetNode("my_target_spatial_node") as Spatial).
↪GetGlobalTransform().origin;
        lookFollow(state, GetGlobalTransform(), targetPosition);
    }
}
```

This function uses the rigid body's `set_angular_velocity()` method to rotate the body. It first calculates the difference between the current and desired angle and then adds the velocity needed to rotate by that amount in one frame's time.

Note: This script will not work with rigid bodies in *character mode* because in this mode the body's rotation is locked. In this case, you would have to rotate the attached mesh node instead using the standard Spatial methods.

9.3 Using KinematicBody2D

9.3.1 Introduction

Godot offers a number of collision objects to provide both collision detection and response. Trying to decide which one to use for your project can be confusing. You can avoid problems and simplify development if you understand how each of them works and what their pros and cons are. In this tutorial, we'll look at the *KinematicBody2D* node and show some examples of how it can be used.

Note: This document assumes you're familiar with Godot's various physics bodies. Please read *Physics introduction* first.

9.3.2 What is a kinematic body?

`KinematicBody2D` is for implementing bodies that are to be controlled via code. They detect collisions with other bodies when moving, but are not affected by engine physics properties like gravity or friction. While this means that you have to write some code to create their behavior, it also means you have more precise control over how they move and react.

Tip: A *KinematicBody2D* can be affected by gravity and other forces, but you must calculate the movement in code. The physics engine will not move a *KinematicBody2D*.

9.3.3 Movement and Collision

When moving a `KinematicBody2D`, you should not set its `position` property directly. Instead, you use the `move_and_collide()` or `move_and_slide()` methods. These methods move the body along a given vector and will instantly stop if a collision is detected with another body. After a KinematicBody2D has collided, any *collision response* must be coded manually.

Warning: Kinematic body movement should only be done in the `_physics_process()` callback.

The two movement methods serve different purposes, and later in this tutorial you'll see examples of how they work.

move_and_collide

This method takes one parameter: a *Vector2* indicating the body's relative movement. Typically, this is your velocity vector multiplied by the frame timestep (`delta`). If the engine detects a collision anywhere along this vector, the body will immediately stop moving. If this happens, the method will return a *KinematicCollision2D* object.

`KinematicCollision2D` is an object containing data about the collision and the colliding object. Using this data you can calculate your collision response.

move_and_slide

The `move_and_slide()` method is intended to simplify the collision response in the common case where you want one body to slide along the other. This is especially useful in platformers or top-down games, for example.

Tip: `move_and_slide()` automatically calculates frame-based movement using `delta`. Do *not* multiply your velocity vector by `delta` before passing it to `move_and_slide()`.

In addition to the velocity vector, `move_and_slide()` takes a number of other parameters allowing you to customize the slide behavior:

- `floor_normal` - *default value:* `Vector2( 0, 0 )`

  This parameter allows you to define what surfaces the engine should consider to be the floor. Setting this lets you use the `is_on_floor()`, `is_on_wall()`, and `is_on_ceiling()` methods to detect what type of surface the body is in contact with. The default value means that all surfaces are considered walls.

- `slope_stop_min_velocity` - *default value:* `5`

  This is the minimum velocity when standing on a slope. This prevents a body from sliding down a slope when standing still.

- `max_bounces` - *default value:* `4`

  This is the maximum number of collisions before the body stops moving. Setting this too low may prevent movement entirely.

- `floor_max_angle` - *default value:* `0.785398` (in radians, equivalent to `45` degrees)

  This is the maximum angle before a surface is no longer considered a "floor".

9.3.4 Which movement method to use?

A common question from new Godot users is: "How do you decide which movement function to use?" Often the response is to use `move_and_slide()` because it's "simpler", but this is not necessarily the case. One way to think of it is that `move_and_slide()` is a special case, and `move_and_collide()` is more general. For example, the following two code snippets result in the same collision response:

GDScript

C#

```
# using move_and_collide
var collision = move_and_collide(velocity * delta)
if collision:
    velocity = velocity.slide(collision.normal)

# using move_and_slide
velocity = move_and_slide(velocity)
```

```
// using MoveAndCollide
var collision = MoveAndCollide(velocity * delta);
if (collision != null)
{
    velocity = velocity.Slide(collision.Normal);
}
// using MoveAndSlide
velocity = MoveAndSlide(velocity);
```

Anything you do with `move_and_slide()` can also be done with `move_and_collide()`, but it might take a little more code. However, as we'll see in the examples below, there are cases where `move_and_slide()` doesn't provide the response you want.

9.3.5 Examples

To see these examples in action, download the sample project: `using_kinematic2d.zip`.

Movement and walls

If you've downloaded the sample project, this example is in the "BasicMovement.tscn" scene.

For this example, Add a `KinematicBody2D` with two children: a `Sprite` and a `CollisionShape2D`. Use the Godot "icon.png" as the Sprite's texture (drag it from the Filesystem dock to the *Texture* property of the `Sprite`). In the `CollisionShape2D`'s *Shape* property, select "New RectangleShape2D" and size the rectangle to fit over the sprite image.

Note: See *2D Movement Overview* for examples of implementing 2D movement schemes.

Attach a script to the KinematicBody2D and add the following code:

GDScript

C#

```
extends KinematicBody2D

var speed = 250
var velocity = Vector2()

func get_input():
    # Detect up/down/left/right keystate and only move when pressed
    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
    velocity = velocity.normalized() * speed

func _physics_process(delta):
    get_input()
    move_and_collide(velocity * delta)
```

```
using Godot;
using System;

public class KBExample : KinematicBody2D
{
    public int Speed = 250;
    private Vector2 _velocity = new Vector2();

    public void getInput()
    {
        // Detect up/down/left/right keystate and only move when pressed
        _velocity = new Vector2();
        if (Input.IsActionPressed("ui_right"))
        {
            _velocity.x += 1;
        }
        if (Input.IsActionPressed("ui_left"))
        {
            _velocity.x -= 1;
        }
        if (Input.IsActionPressed("ui_down"))
        {
            _velocity.y += 1;
        }
        if (Input.IsActionPressed("ui_up"))
        {
            _velocity.y -= 1;
        }
    }

    public override void _PhysicsProcess(float delta)
    {
        getInput();
        MoveAndCollide(velocity * delta);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

Run this scene and you'll see that `move_and_collide()` works as expected, moving the body along the velocity vector. Now let's see what happens when you add some obstacles. Add a *StaticBody2D* with a rectangular collision shape. For visibility, you can use a sprite, a Polygon2D, or turn on "Visible Collision Shapes" from the "Debug" menu.

Run the scene again and try moving into the obstacle. You'll see that the `KinematicBody2D` can't penetrate the obstacle. However, try moving into the obstacle at an angle and you'll find that the obstacle acts like glue - it feels like the body gets stuck.

This happens because there is no *collision response*. `move_and_collide()` stops the body's movement when a collision occurs. We need to code whatever response we want from the collision.

Try changing the function to `move_and_slide(velocity)` and running again. Note that we removed `delta` from the velocity calculation.

`move_and_slide()` provides a default collision response of sliding the body along the collision object. This is useful for a great many game types, and may be all you need to get the behavior you want.

Bouncing/reflecting

What if you don't want a sliding collision response? For this example ("BounceandCollide.tscn" in the sample project), we have a character shooting bullets and we want the bullets to bounce off the walls.

This example uses three scenes. The main scene contains the Player and Walls. The Bullet and Wall are separate scenes so that they can be instanced.

The Player is controlled by the *w* and *s* keys for forward and back. Aiming uses the mouse pointer. Here is the code for the Player, using `move_and_slide()`:

GDScript

C#

```
extends KinematicBody2D

var Bullet = preload("res://Bullet.tscn")
var speed = 200
var velocity = Vector2()

func get_input():
    # add these actions in Project Settings -> Input Map
    velocity = Vector2()
    if Input.is_action_pressed('backward'):
        velocity = Vector2(-speed/3, 0).rotated(rotation)
    if Input.is_action_pressed('forward'):
        velocity = Vector2(speed, 0).rotated(rotation)
    if Input.is_action_just_pressed('mouse_click'):
        shoot()

func shoot():
    # "Muzzle" is a Position2D placed at the barrel of the gun
    var b = Bullet.instance()
    b.start($Muzzle.global_position, rotation)
    get_parent().add_child(b)

func _physics_process(delta):
```

(continues on next page)

(continued from previous page)

```
    get_input()
    var dir = get_global_mouse_position() - global_position
    # Don't move if too close to the mouse pointer
    if dir.length() > 5:
        rotation = dir.angle()
        velocity = move_and_slide(velocity)
```

```
using Godot;
using System;

public class KBExample : KinematicBody2D
{
    private PackedScene _bullet = (PackedScene)GD.Load("res://Bullet.tscn");
    public int Speed = 200;
    private Vector2 _velocity = new Vector2();

    public void getInput()
    {
        // add these actions in Project Settings -> Input Map
        _velocity = new Vector2();
        if (Input.IsActionPressed("backward"))
        {
            _velocity = new Vector2(-speed/3, 0).Rotated(Rotation);
        }
        if (Input.IsActionPressed("forward"))
        {
            _velocity = new Vector2(speed, 0).Rotated(Rotation);
        }
        if (Input.IsActionPressed("mouse_click"))
        {
            Shoot();
        }
    }

    public void Shoot()
    {
        // "Muzzle" is a Position2D placed at the barrel of the gun
        var b = (Bullet)_bullet.Instance();
        b.Start(((Node2D)GetNode("Muzzle")).GlobalPosition, Rotation);
        GetParent().AddChild(b);
    }

    public override void _PhysicsProcess(float delta)
    {
        getInput();
        var dir = GetGlobalMousePosition() - GlobalPosition;
        // Don't move if too close to the mouse pointer
        if (dir.Length() > 5)
        {
            Rotation = dir.Angle();
            _velocity = MoveAndSlide(_velocity);
        }
    }
}
```

And the code for the Bullet:

GDScript

C#

```
extends KinematicBody2D

var speed = 750
var velocity = Vector2()

func start(pos, dir):
    rotation = dir
    position = pos
    velocity = Vector2(speed, 0).rotated(rotation)

func _physics_process(delta):
    var collision = move_and_collide(velocity * delta)
    if collision:
        velocity = velocity.bounce(collision.normal)
        if collision.collider.has_method("hit"):
            collision.collider.hit()

func _on_VisibilityNotifier2D_screen_exited():
    queue_free()
```

```
using Godot;
using System;

public class Bullet : KinematicBody2D
{
    public int Speed = 750;
    private Vector2 _velocity = new Vector2();

    public void Start(Vector2 pos, float dir)
    {
        Rotation = dir;
        Position = pos;
        _velocity = new Vector2(speed, 0).Rotated(Rotation);
    }

    public override void _PhysicsProcess(float delta)
    {
        var collsion = MoveAndCollide(_velocity * delta);
        if (collsion != null)
        {
            _velocity = _velocity.Bounce(collsion.Normal);
            if (collsion.Collider.HasMethod("Hit"))
            {
                collsion.Collider.Hit();
            }
        }
    }

    public void OnVisibilityNotifier2DScreenExited()
    {
        QueueFree();
    }
}
```

The action happens in `_physics_process()`. After using `move_and_collide()` if a collision occurs, a

`KinematicCollision2D` object is returned (otherwise, the return is `Nil`).

If there is a returned collision, we use the `normal` of the collision to reflect the bullet's `velocity` with the `Vector2.bounce()` method.

If the colliding object (`collider`) has a `hit` method, we also call it. In the example project, we've added a flashing color effect to the Wall to demonstrate this.

Platformer movement

Let's try one more popular example: the 2D platformer. `move_and_slide()` is ideal for quickly getting a functional character controller up and running. If you've downloaded the sample project, you can find this in "Platformer.tscn".

For this example, we'll assume you have a level made of `StaticBody2D` objects. They can be any shape and size. In the sample project, we're using *Polygon2D* to create the platform shapes.

Here's the code for the player body:

GDScript

C#

```
extends KinematicBody2D

export (int) var run_speed = 100
export (int) var jump_speed = -400
export (int) var gravity = 1200

var velocity = Vector2()
var jumping = false

func get_input():
    velocity.x = 0
    var right = Input.is_action_pressed('ui_right')
    var left = Input.is_action_pressed('ui_left')
    var jump = Input.is_action_just_pressed('ui_select')

    if jump and is_on_floor():
        jumping = true
        velocity.y = jump_speed
    if right:
        velocity.x += run_speed
    if left:
        velocity.x -= run_speed

func _physics_process(delta):
    get_input()
    velocity.y += gravity * delta
    if jumping and is_on_floor():
        jumping = false
    velocity = move_and_slide(velocity, Vector2(0, -1))
```

```
using Godot;
using System;

public class KBExample : KinematicBody2D
```

(continues on next page)

(continued from previous page)

```
{
    [Export] public int RunSpeed = 100;
    [Export] public int JumpSpeed = -400;
    [Export] public int Gravity = 1200;

    Vector2 velocity = new Vector2();
    bool jumping = false;

    public void getInput()
    {
        velocity.x = 0;
        bool right = Input.IsActionPressed("ui_right");
        bool left = Input.IsActionPressed("ui_left");
        bool jump = Input.IsActionPressed("ui_select");

        if (jump && IsOnFloor())
        {
            jumping = true;
            velocity.y = JumpSpeed;
        }
        if (right)
        {
            velocity.x += RunSpeed;
        }
        if (left)
        {
            velocity.x -= RunSpeed;
        }
    }

    public override void _PhysicsProcess(float delta)
    {
        getInput();
        velocity.y += Gravity * delta;
        if (jumping && IsOnFloor())
        {
            jumping = false;
        }
        velocity = MoveAndSlide(velocity, new Vector2(0, -1));
    }
}
```

When using `move_and_slide()` the function returns a vector representing the movement that remained after the slide collision occurred. Setting that value back to the character's `velocity` allows us to smoothly move up and down slopes. Try removing `velocity =` and see what happens if you don't do this.

Also note that we've added `Vector2(0, -1)` as the floor normal. This is a vector pointing straight upward. This means that if the character collides with an object that has this normal, it will be considered a floor.

Using the floor normal allows us to make jumping work, using `is_on_floor()`. This function will only return `true` after a `move_and_slide()` collision where the colliding body's normal is within 45 degrees of the given floor vector (this can be adjusted by setting `floor_max_angle`).

This also allows you to implement other features like wall jumps using `is_on_wall()`, for example.

9.4 Ray-casting

9.4.1 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.

9.4.2 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 accessing *CanvasItem.get_world_2d().space*. For 3D, it's *Spatial.get_world().space*.

The resulting space *RID* can be used in *PhysicsServer* and *Physics2DServer* respectively for 3D and 2D.

9.4.3 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.

Use the following code in 2D:

GDscript

C#

```
func _physics_process(delta):
    var space_rid = get_world_2d().space
    var space_state = Physics2DServer.space_get_direct_state(space_rid)
```

```
public override void _PhysicsProcess(float delta)
{
    var spaceRid = GetWorld2d().Space;
    var spaceState = Physics2DServer.SpaceGetDirectState(spaceRid);
}
```

Or more directly:

GDScript

C#

```
func _physics_process(delta):
    var space_state = get_world_2d().direct_space_state
```

```
public override void _PhysicsProcess(float delta)
{
    var spaceState = GetWorld2d().DirectSpaceState;
}
```

And in 3D:

GDScript

C#

```
func _physics_process(delta):
    var space_state = get_world().direct_space_state
```

```
public override void _PhysicsProcess(float delta)
{
    var spaceState = GetWorld().DirectSpaceState;
}
```

9.4.4 Raycast query

For performing a 2D raycast query, the method *Physics2DDirectSpaceState.intersect_ray()* may be used. For example:

GDScript

C#

```
func _physics_process(delta):
    var space_state = get_world_2d().direct_space_state
    # use global coordinates, not local to node
    var result = space_state.intersect_ray(Vector2(0, 0), Vector2(50, 100))
```

```
public override void _PhysicsProcess(float delta)
{
    var spaceState = GetWorld2d().DirectSpaceState;
    // use global coordinates, not local to node
    var result = spaceState.IntersectRay(new Vector2(), new Vector2(50, 100));
}
```

The 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:

GDScript

C#

```
if result:
    print("Hit at point: ", result.position)
```

```
if (result.Count > 0)
    GD.Print("Hit at point: ", result["position"]);
```

The `result` dictionary when a collision occurs contains the following data:

```
{
   position: Vector2 # point in world space for collision
```

(continues on next page)

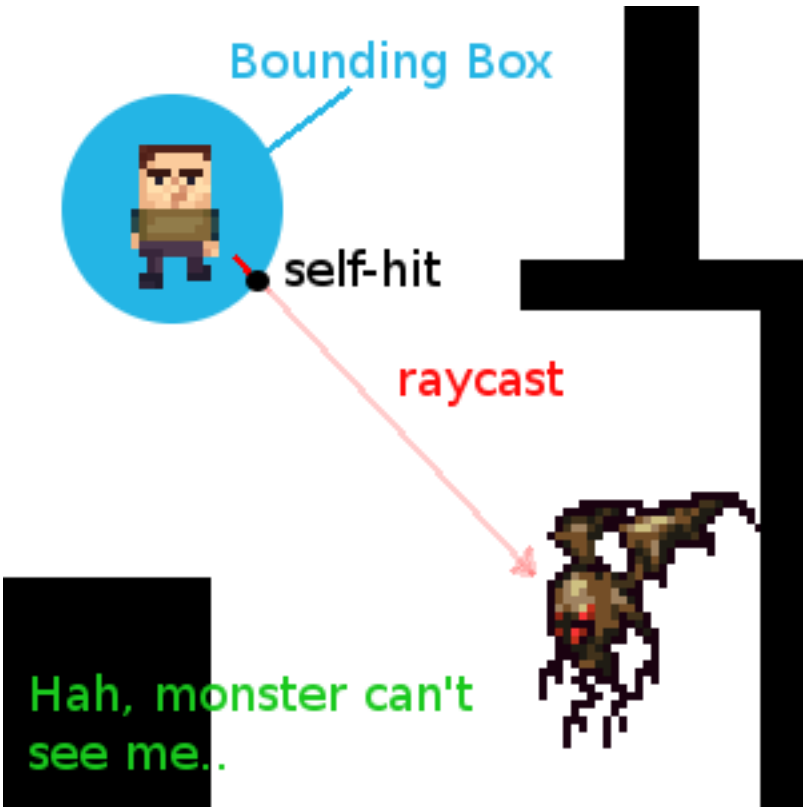
(continued from previous page)

```
    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
}
```

The data is similar in 3D space, using Vector3 coordinates.

9.4.5 Collision exceptions

A common use case for ray casting is to enable a character to gather data about the world around it. One problem with this is that the same character has a collider, so the ray will only detect its parent's collider, as shown 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 collision object node:

GDScript

C#

```
extends KinematicBody2D

func _physics_process(delta):
    var space_state = get_world_2d().direct_space_state
    var result = space_state.intersect_ray(global_position, enemy_position, [self])
```

```
class Body : KinematicBody2D
{
    public override void _PhysicsProcess(float delta)
    {
        var spaceState = GetWorld2d().DirectSpaceState;
        var result = spaceState.IntersectRay(globalPosition, enemyPosition, new␣
↪object[] { this });
    }
}
```

The exceptions array can contain objects or RIDs.

9.4.6 Collision Mask

While the exceptions method works fine for excluding the parent body, it becomes very inconvenient if you need a large and/or dynamic list of exceptions. In this case, it is much more efficient to use the collision layer/mask system.

The optional fourth argument for `intersect_ray()` is a collision mask. For example, to use same mask as the parent body, use the `collision_mask` member variable:

GDScript

C#

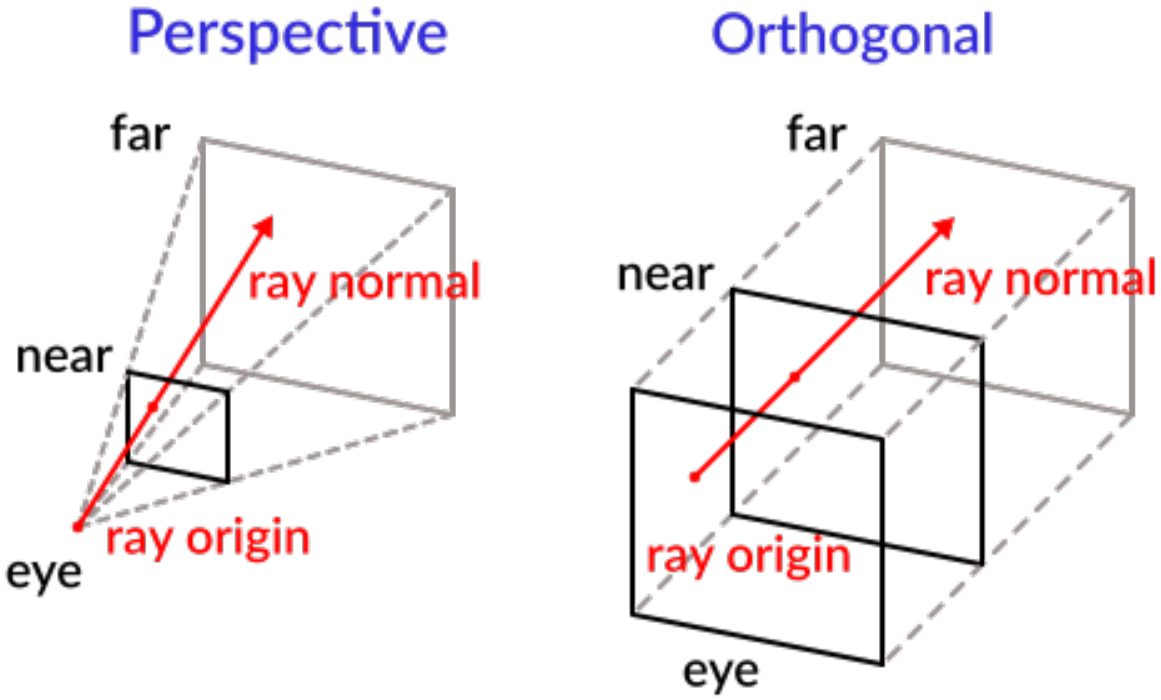
```
extends KinematicBody2D

func _physics_process(delta):
    var space_state = get_world().direct_space_state
    var result = space_state.intersect_ray(global_position, enemy_position,
                             [self], collision_mask)
```

```
class Body : KinematicBody2D
{
    public override void _PhysicsProcess(float delta)
    {
        var spaceState = GetWorld2d().DirectSpaceState;
        var result = spaceState.IntersectRay(globalPosition, enemyPosition,
                         new object[] { this }, CollisionMask);
    }
}
```

9.4.7 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, you need a *Camera* node. A `Camera` can be in two projection modes: perspective and orthogonal. Because of this, both the ray origin and direction must be obtained. This is because `origin` changes in orthogonal mode, while `normal` changes in perspective mode:



To obtain it using a camera, the following code can be used:

GDScript

C#

```
const ray_length = 1000

func _input(event):
    if event is InputEventMouseButton and event.pressed and event.button_index == 1:
          var camera = $Camera
          var from = camera.project_ray_origin(event.position)
          var to = from + camera.project_ray_normal(event.position) * ray_length
```

```
private const float rayLength = 1000;

public override void _Input(InputEvent @event)
{
    if (@event is InputEventMouseButton eventMouseButton && eventMouseButton.Pressed &
↪& eventMouseButton.ButtonIndex == 1)
    {
        var camera = (Camera)GetNode("Camera");
        var from = camera.ProjectRayOrigin(eventMouseButton.Position);
        var to = from + camera.ProjectRayNormal(eventMouseButton.Position) *␣
↪rayLength;
    }
}
```

Remember that during `_input()`, the space may be locked, so in practice this query should be run in `_physics_process()`.

9.5 Kinematic Character (2D)

9.5.1 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 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).

9.5.2 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.

GDScript

C#

```
extends KinematicBody2D

func _physics_process(delta):
    pass
```

```
using Godot;
using System;

public class PhysicsScript : KinematicBody2D
{

    public override void _PhysicsProcess(float delta)
    {
```

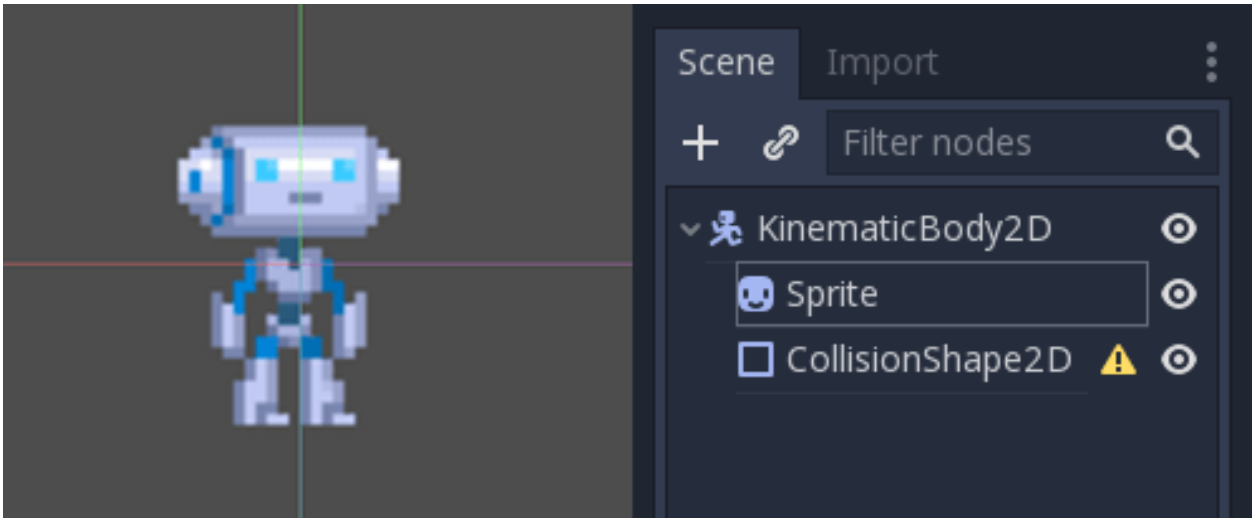
(continues on next page)

(continued from previous page)

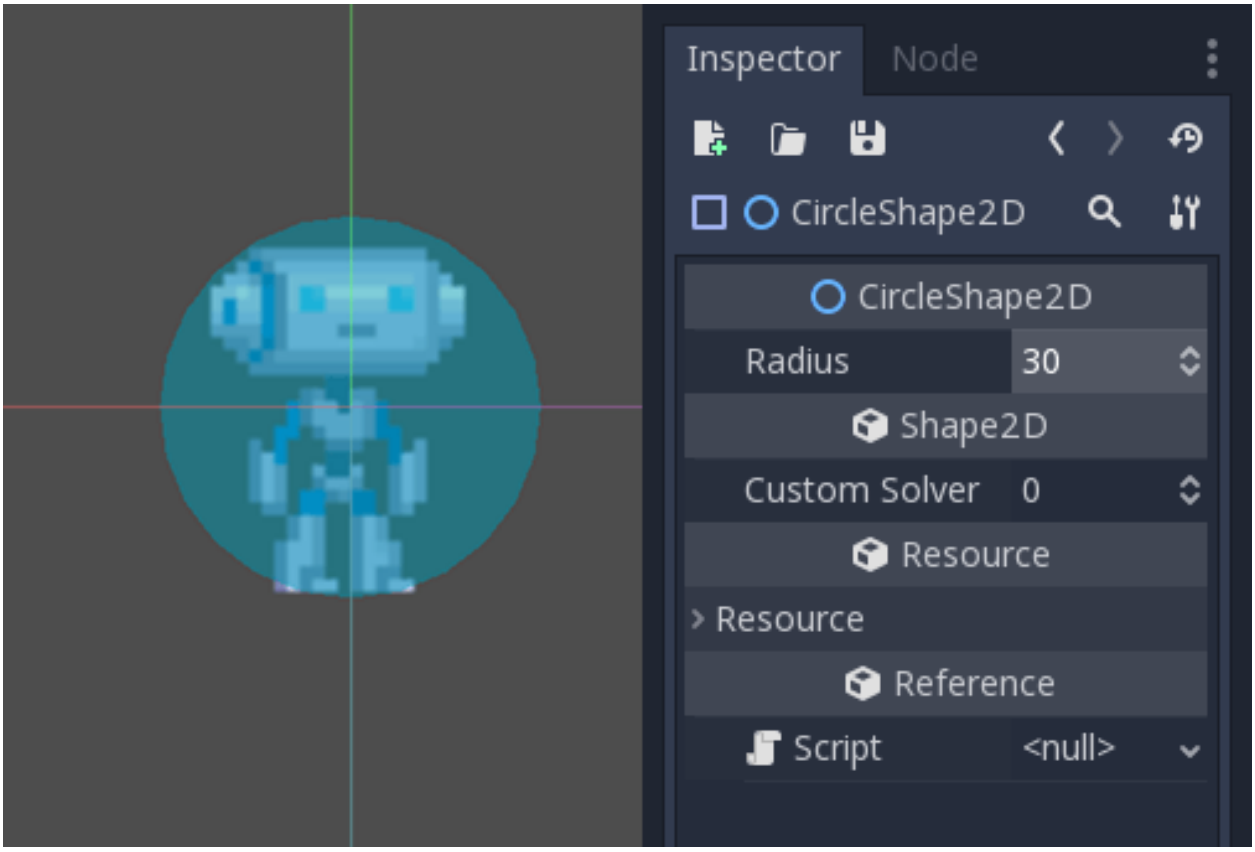
```
        }
}
```

9.5.3 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:



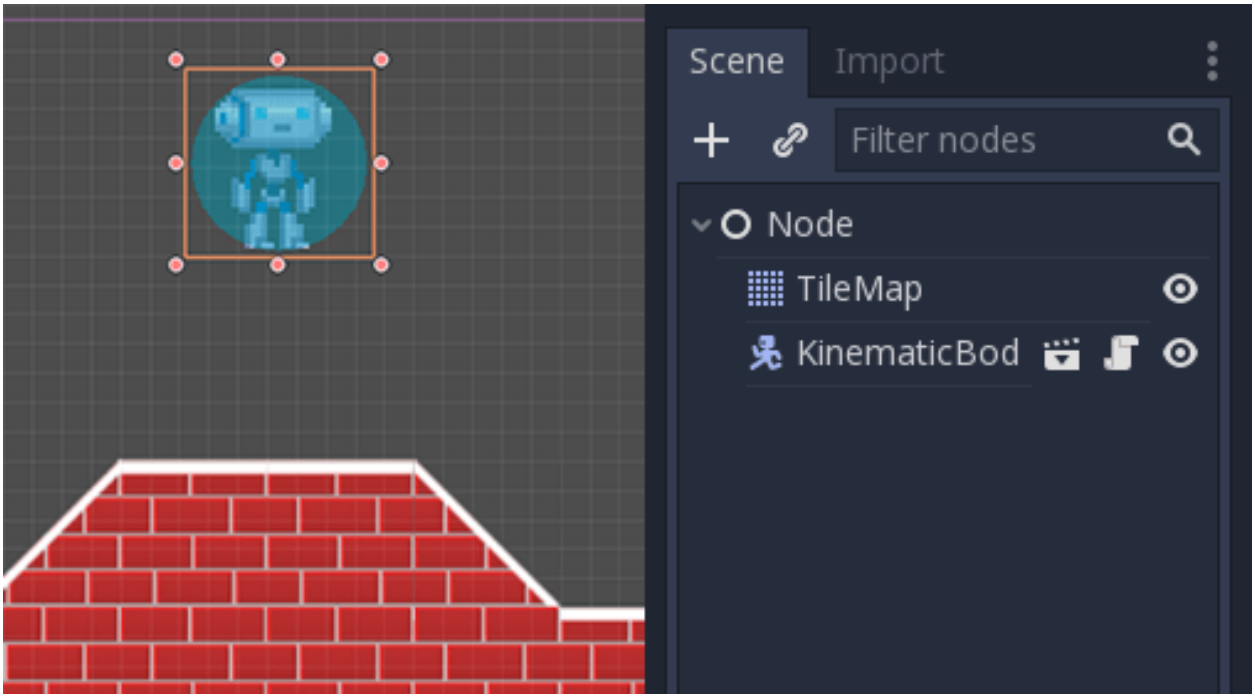
You'll notice that there's a warning icon next to our CollisionShape2D node, that's because we haven't defined a shape for it. Create a new CircleShape2D in the shape property of CollisionShape2D. Click on <CircleShape2D> to go to the options for it, and set the radius to 30:



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.



9.5.4 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 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:

GDScript

C#

```
extends KinematicBody2D

func _physics_process(delta):
    move_and_collide(Vector2(0, 1)) # Move down 1 pixel per physics frame
```

```
using Godot;
using System;

public class PhysicsScript : KinematicBody2D
{

    public override void _PhysicsProcess(float delta)
    {
        // Move down 1 pixel per physics frame
        MoveAndCollide(new Vector2 (0,1));
    }
}
```

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:

GDScript

C#

```
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)
```

```
using Godot;
using System;

public class PhysicsScript : KinematicBody2D
{
    const float gravity = 200.0f;
    Vector2 velocity;

    public override void _PhysicsProcess(float delta)
    {
        velocity.y += delta * gravity;

        var motion = velocity * delta;
        MoveAndCollide(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:

GDScript

C#

```
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

    # We don't need to multiply velocity by delta because MoveAndSlide already takes␣
↪delta time into account.
```

(continues on next page)

(continued from previous page)

```
    # 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(velocity, Vector2(0, -1))
```

```
using Godot;
using System;

public class PhysicsScript : KinematicBody2D
{
    const float gravity = 200.0f;
    const int walk_speed = 200;

    Vector2 velocity;

    public override void _PhysicsProcess(float delta)
    {
        velocity.y += delta * gravity;

        if (Input.IsActionPressed("ui_left"))
        {
            velocity.x = -walk_speed;
        }
        else if (Input.IsActionPressed("ui_right"))
        {
            velocity.x = walk_speed;
        }
        else
        {
            velocity.x = 0;
        }

        // We don't need to multiply velocity by delta because MoveAndSlide already␣
↪takes delta time into account.

        // The second parameter of MoveAndSlide 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.
        MoveAndSlide(velocity, new 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

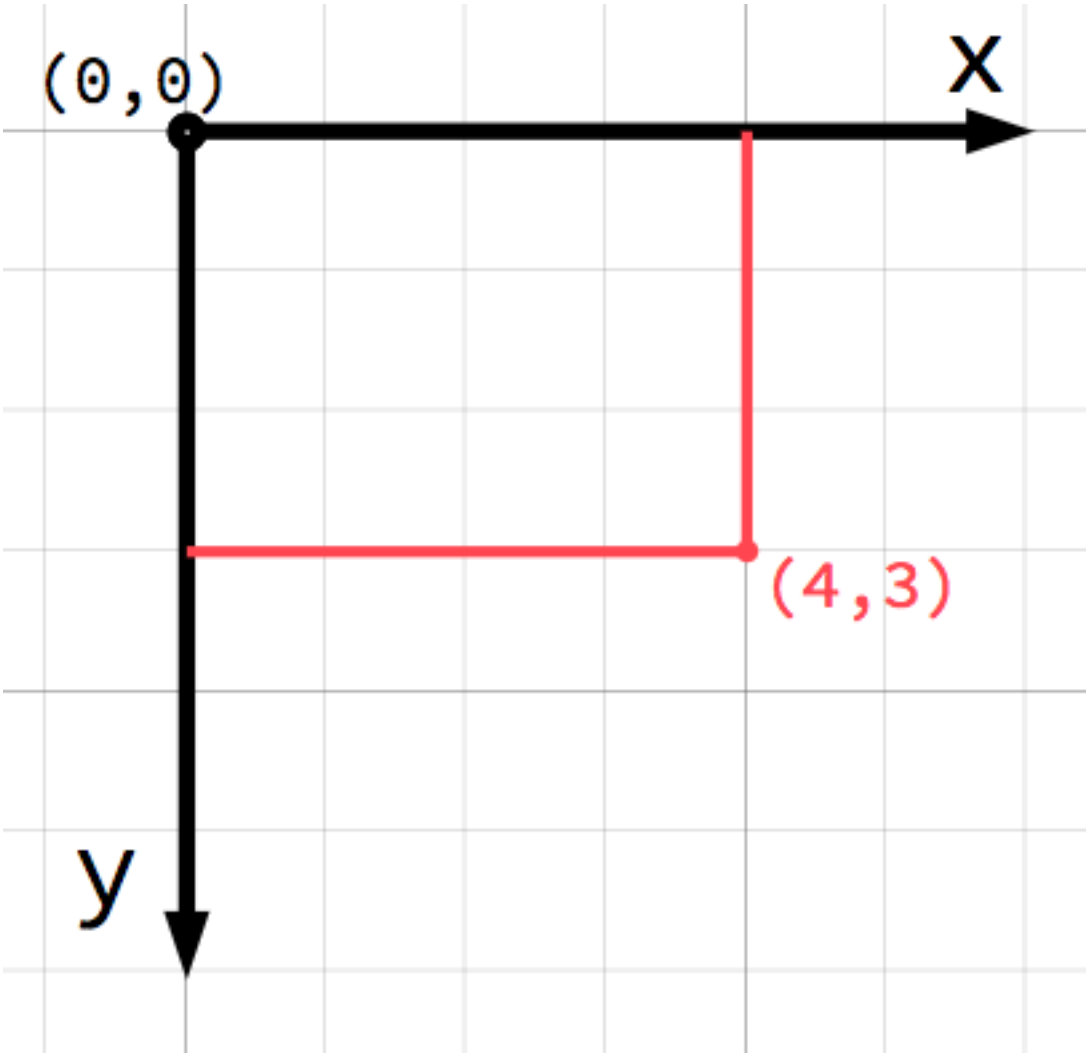
10.1.1 Introduction

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

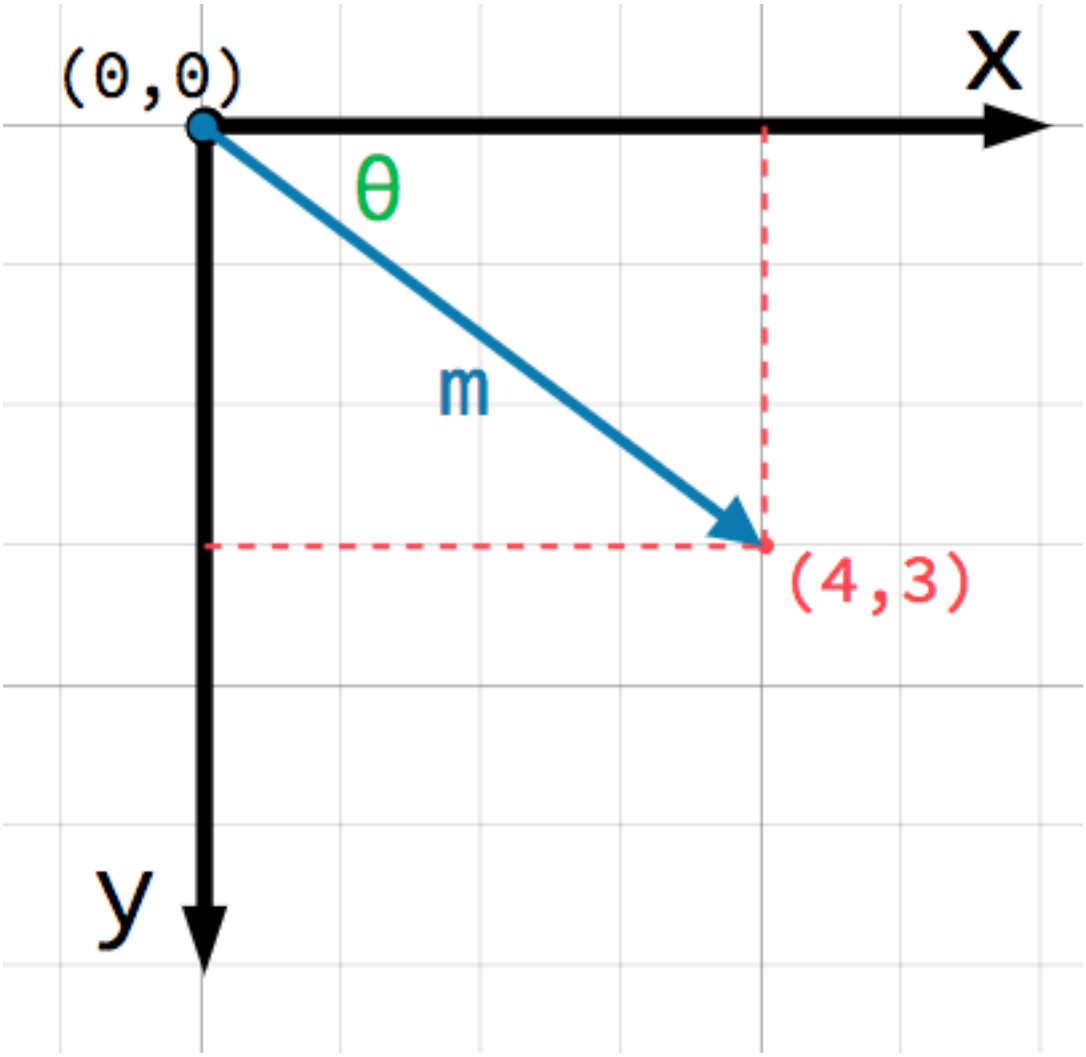
10.1.2 Coordinate systems (2D)

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)`.



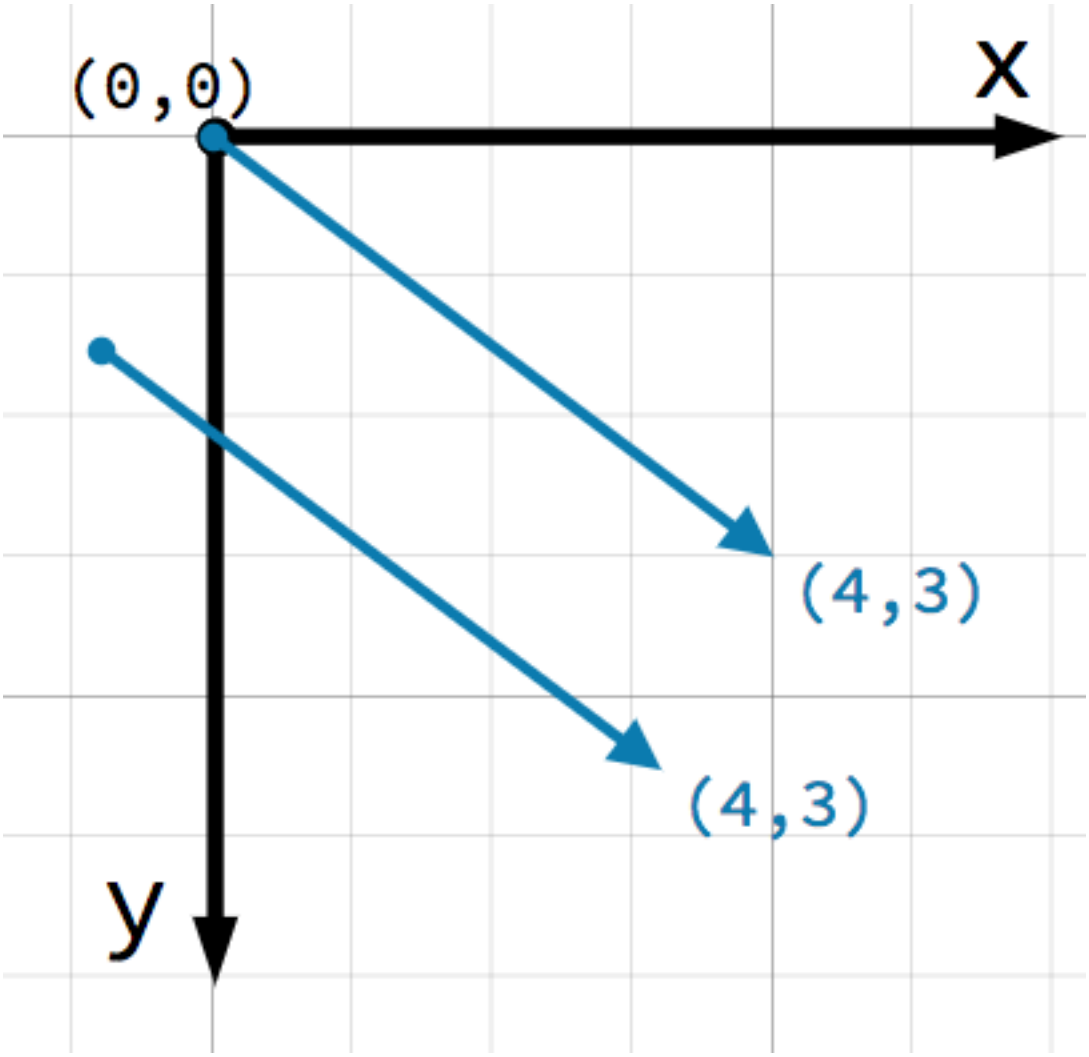
Note: If you're new to computer graphics, it might seem odd that the positive `y` axis points **downwards** instead of upwards, as you probably learned in math class. However, this is common in most computer graphics applications.

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:

GDScript

C#

```
$Node2D.position = Vector2(400, 300)
```

```
var node2D = (Node2D) GetNode("Node2D");
node2D.Position = new 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.

GDScript

C#

```
# 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
```

```
// create a vector with coordinates (2, 5)
var a = new Vector2(2, 5);
// create a vector and assign x and y manually
var b = new Vector2();
b.x = 3;
b.y = 1;
```

- Adding vectors

When adding or subtracting two vectors, the corresponding components are added:

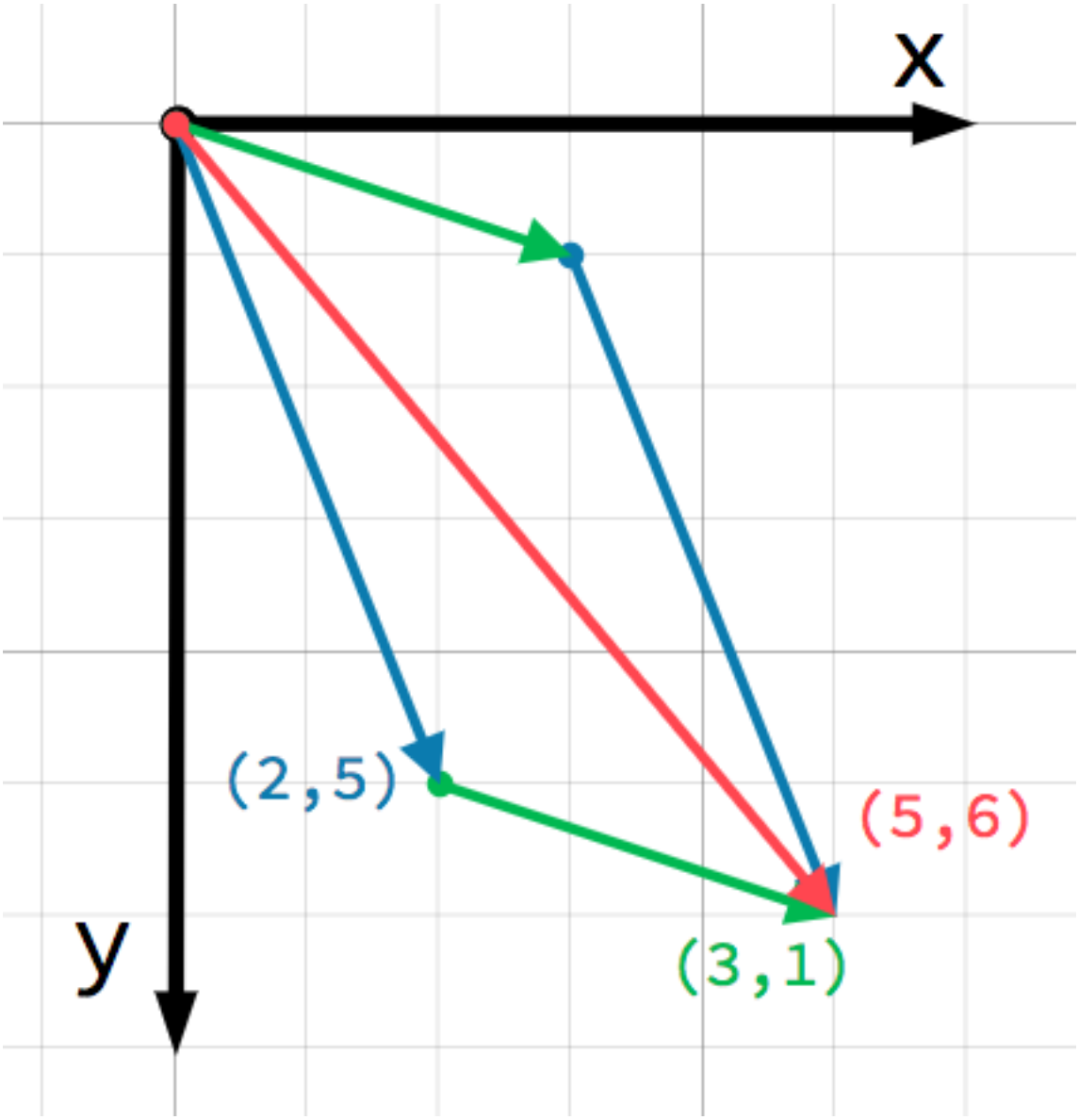
GDScript

C#

```
var c = a + b  # (2, 5) + (3, 1) = (5, 6)
```

```
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**.

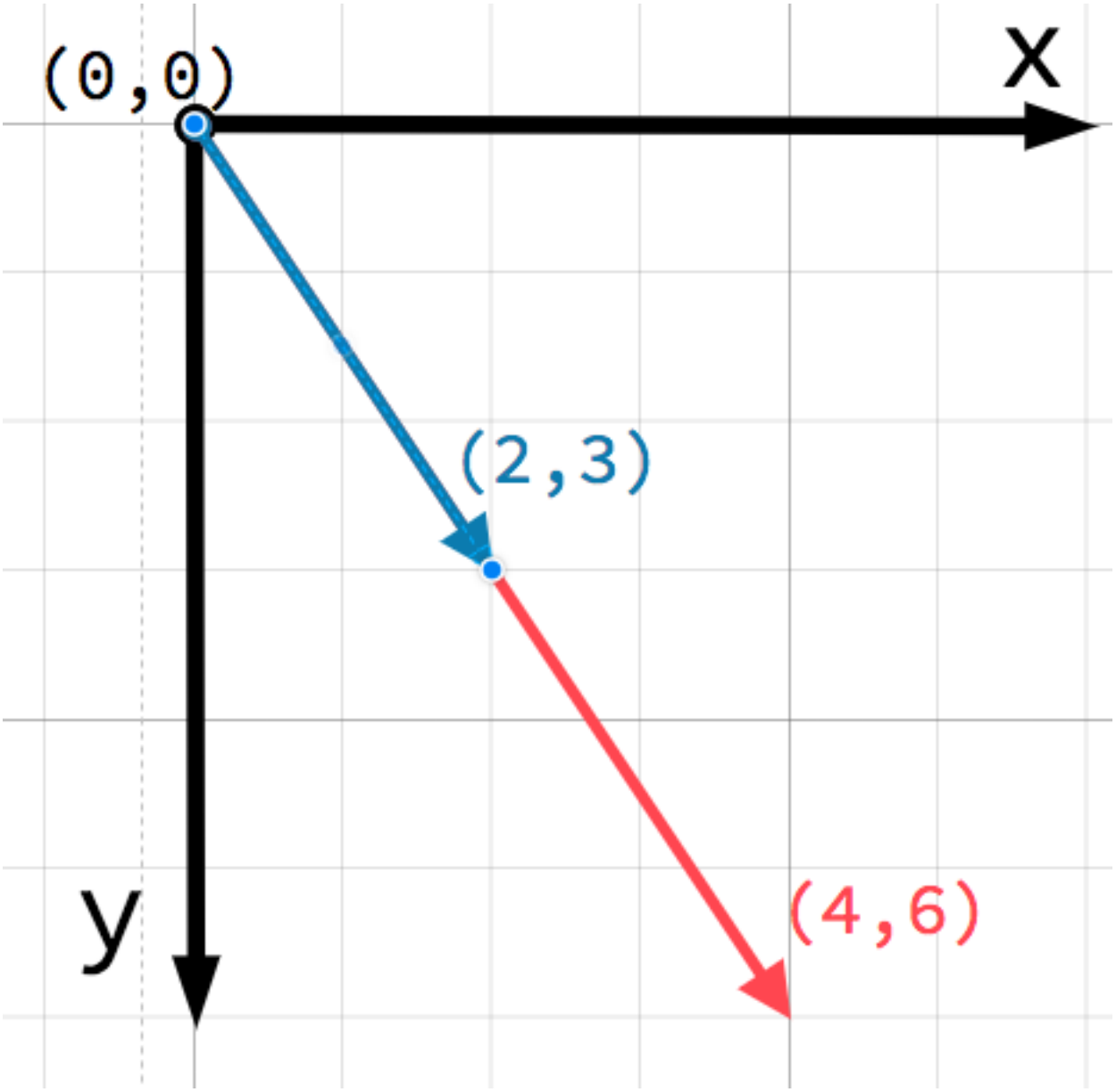
A vector can be multiplied by a **scalar**:

GDScript

C#

```
var c = a * 2  # (2, 5) * 2 = (4, 10)
var d = b / 3  # (3, 6) / 3 = (1, 2)
```

```
var c = a * 2;  // (2, 5) * 2 = (4, 10)
var d = b / 3;  // (3, 6) / 3 = (1, 2)
```



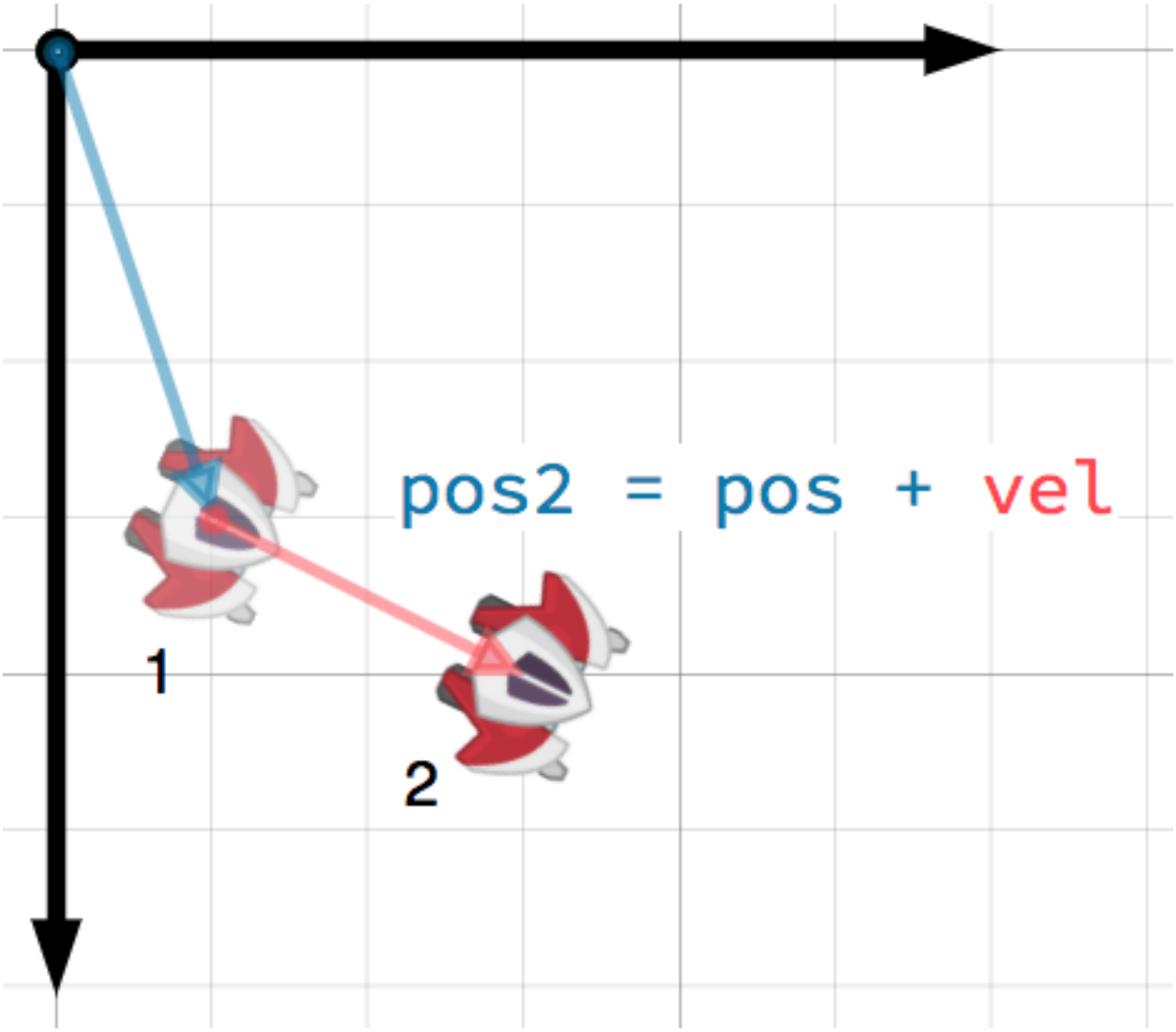
Note: Multiplying a vector by a scalar does not change its direction, only its magnitude. This is how you **scale** a vector.

10.1.4 Practical applications

Let's look at two common uses for vector addition and subtraction.

- Movement

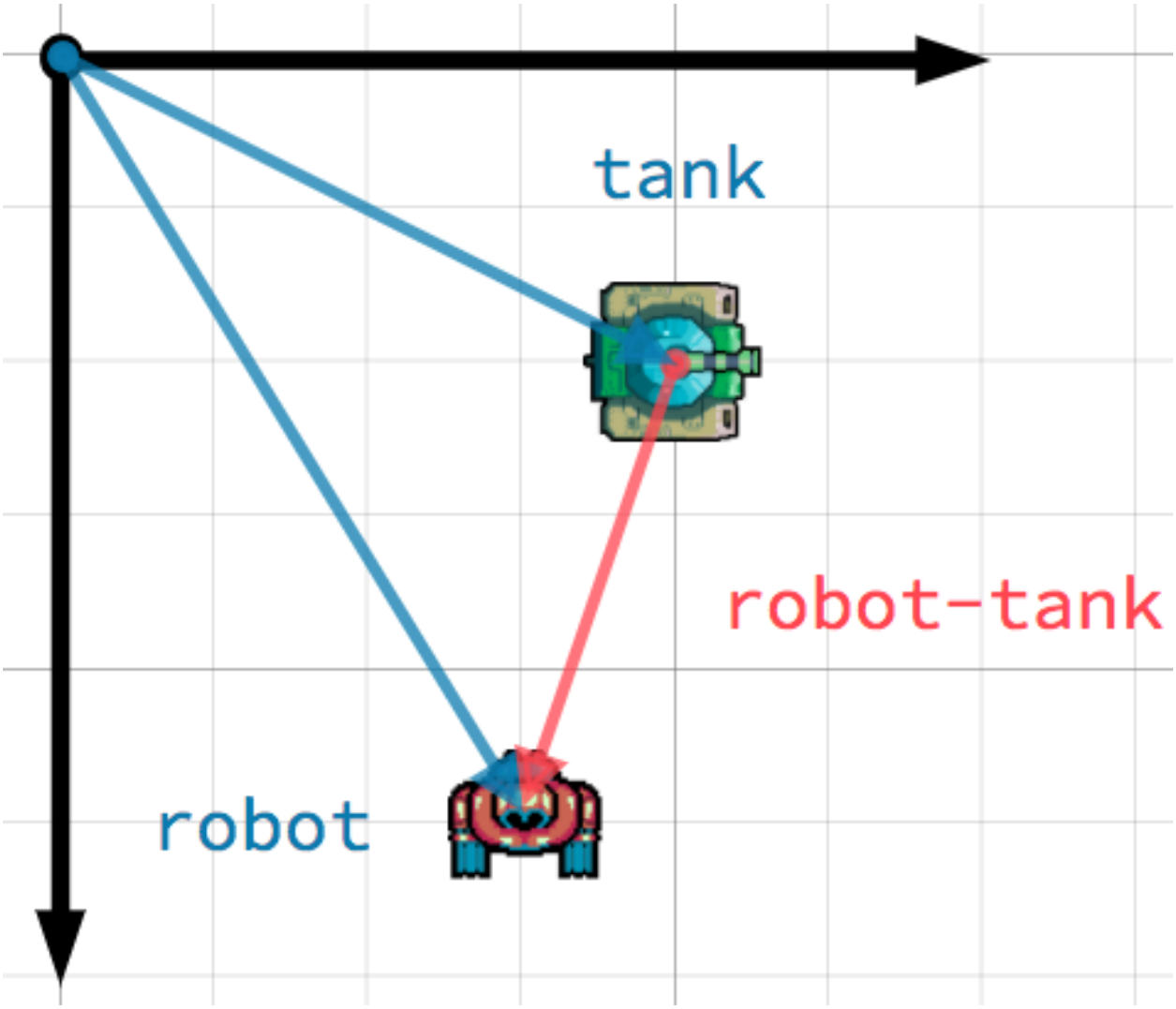
A vector can represent **any** quantity with a magnitude and direction. Typical examples are: position, velocity, acceleration, and force. In this image, the spaceship at step 1 has a position vector of `(1,3)` and a velocity vector of `(2,1)`. The velocity vector represents how far the ship moves each 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:

GDScript

C#

```
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
```

```
var a = new Vector2(2, 4);
var m = Mathf.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:

GDScript

C#

```
a = a.normalized()
```

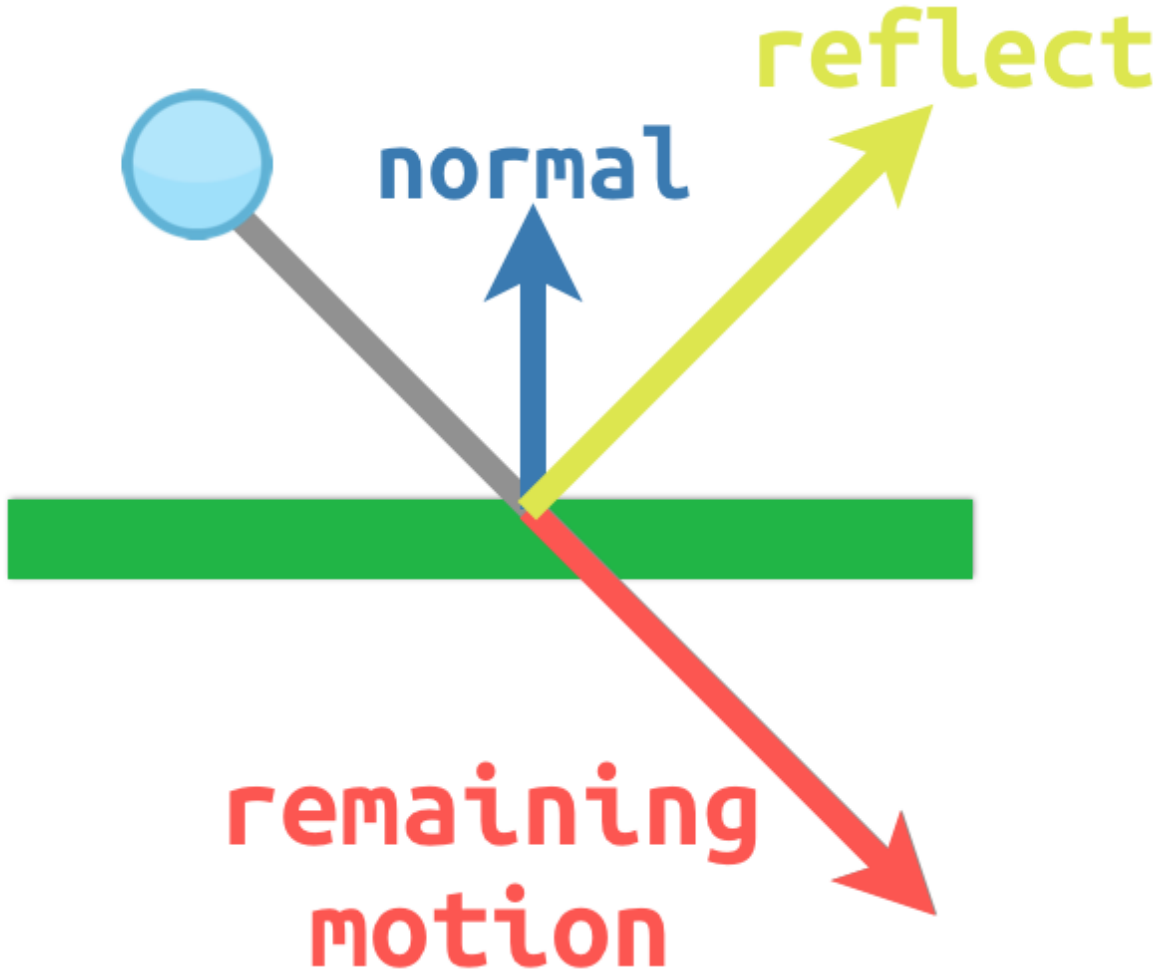
```
a = a.Normalized();
```

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*:

GDScript

C#

```
# object "collision" contains information about the collision
var collision = move_and_collide(velocity * delta)
if collision:
    var reflect = collision.remainder.bounce(collision.normal)
    velocity = velocity.bounce(collision.normal)
    move_and_collide(reflect)
```

```
// KinematicCollision2D contains information about the collision
KinematicCollision2D collision = MoveAndCollide(_velocity * delta);
if (collision != null)
{
    var reflect = collision.Remainder.Bounce(collision.Normal);
    _velocity = _velocity.Bounce(collision.Normal);
```

(continues on next page)

(continued from previous page)

```
    MoveAndCollide(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:

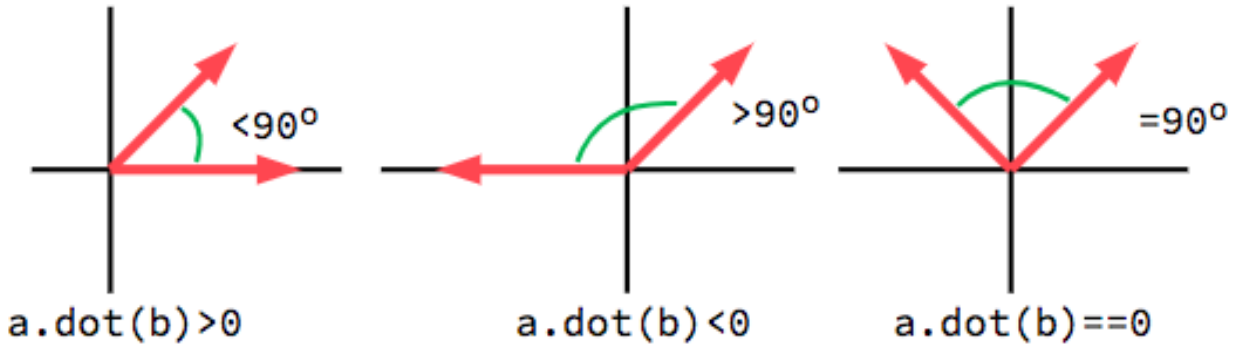
GDScript

C#

```
var c = a.dot(b)
var d = b.dot(a)  # these are equivalent
```

```
float c = a.Dot(b);
float d = b.Dot(a);  // these are equivalent
```

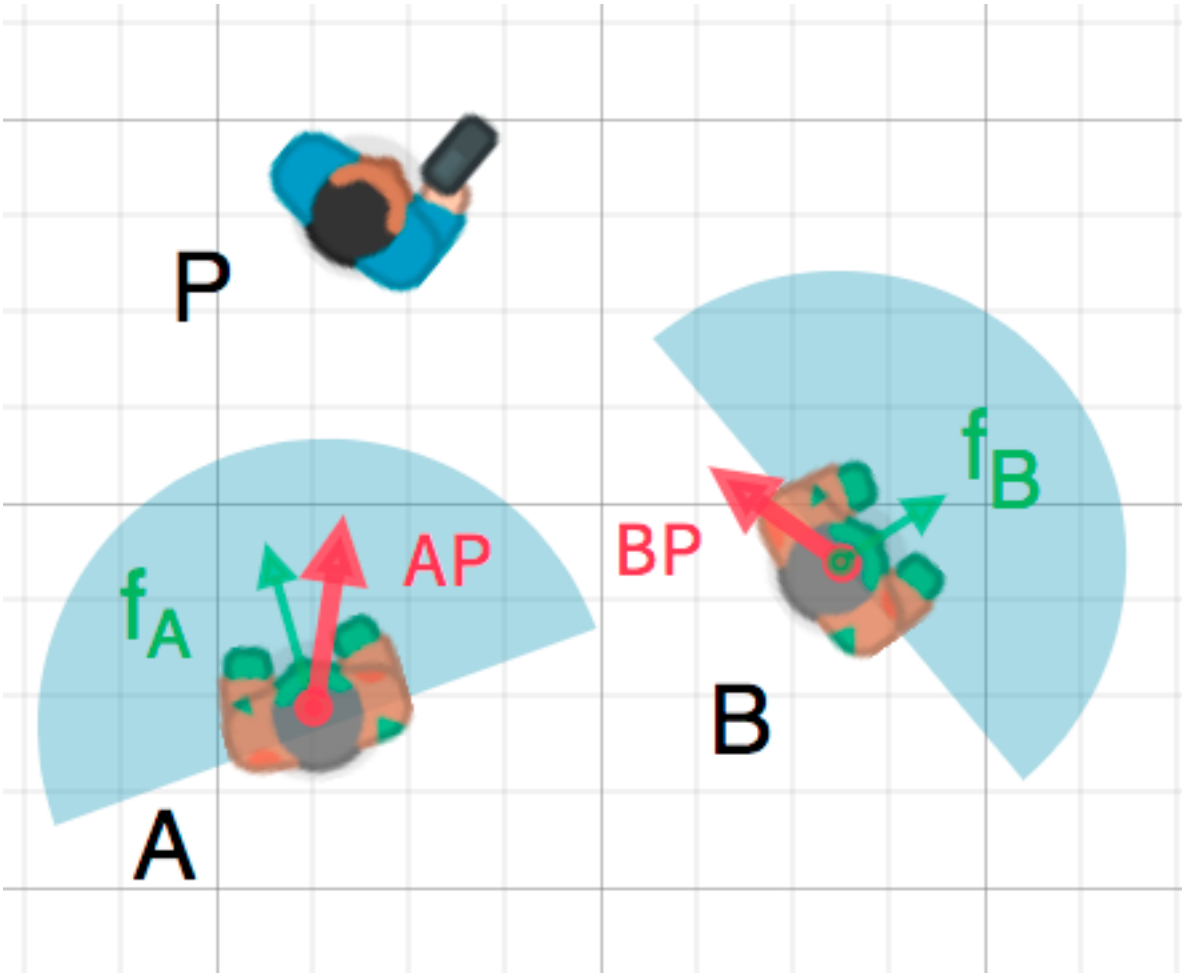
The dot product is most useful when used with unit vectors, making the first formula reduce to just $\cos\theta$. This means we can use the dot product to tell us something about the angle between two vectors:



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 `fA` and `fB` 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 code it would look like this:

GDScript

C#

```
var AP = (P - A).normalized()
if AP.dot(fA) > 0:
    print("A sees P!")
```

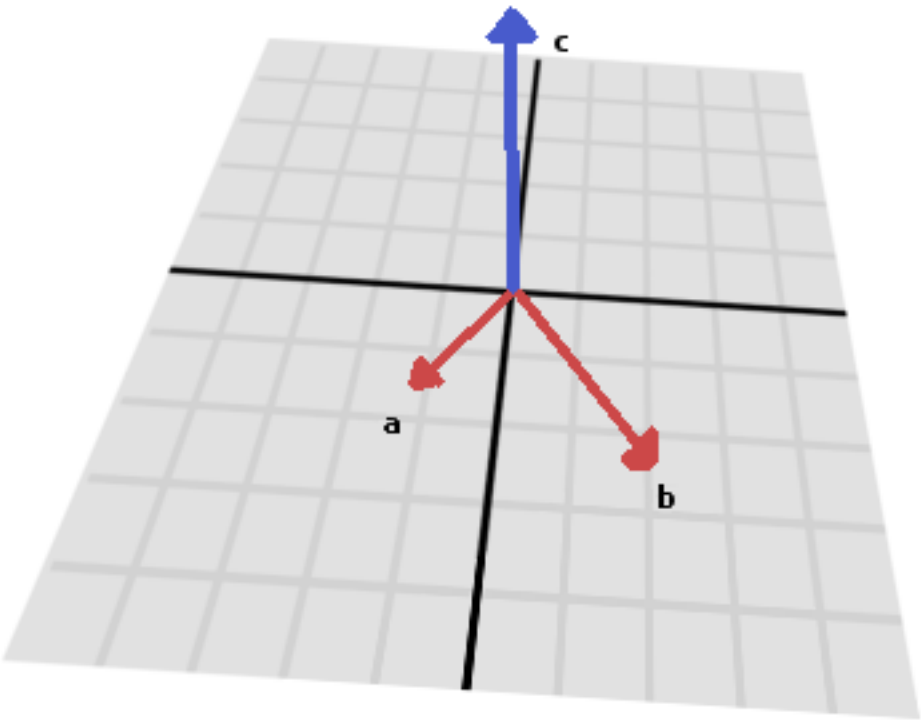
```
var AP = (P - A).Normalized();
if (AP.Dot(fA) > 0)
{
    GD.Print("A sees P!");
}
```

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:

GDScript

C#

```
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)
```

```
var c = new 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);
```

With Godot, you can use the built-in method:

GDScript

C#

```
var c = a.cross(b)
```

```
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, `AB x AC` produces a vector perpendicular to both: the surface normal.

Here is a function to calculate a triangle's normal:

GDScript

C#

```
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
```

```
Vector3 GetTriangleNormal(Vector3 a, Vector3 b, Vector3 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

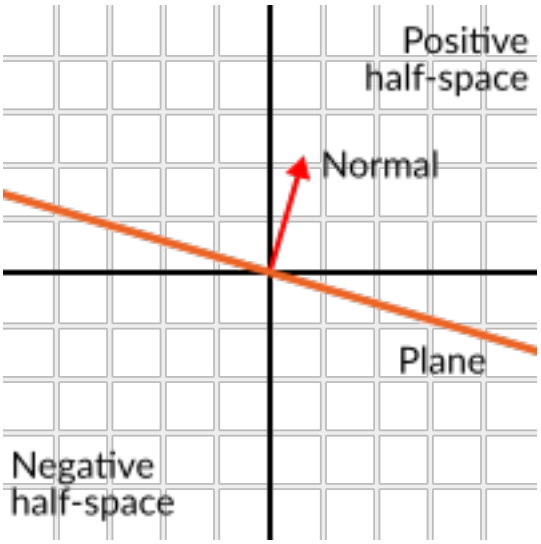
For more information on using vector math in Godot, see the following articles:

- *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 (0,0) the Origin!).

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**:

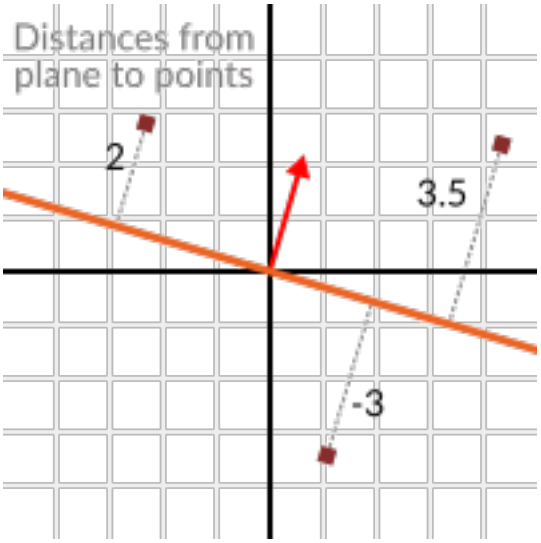
GDScript

C#

```
var distance = normal.dot(point)
```

```
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:



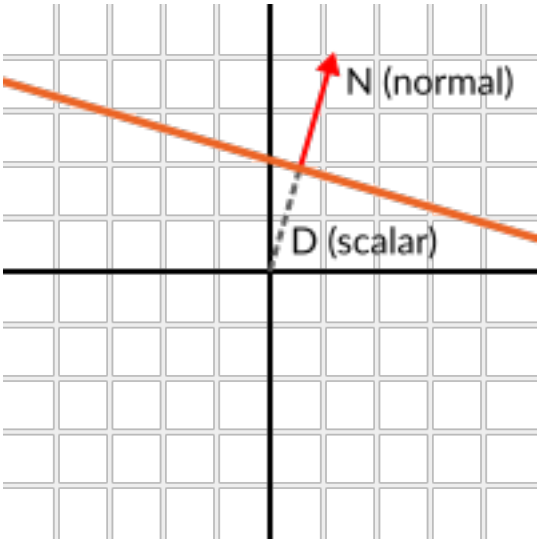
This allows us to tell which side of the plane a point is.

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:

GDScript

C#

```
var point_in_plane = N*D
```

```
var pointInPlane = 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:

GDScript

C#

```
var distance = N.dot(point) - D
```

```
var distance = N.Dot(point) - D;
```

The same thing, using a built-in function:

GDScript

C#

```
var distance = plane.distance_to(point)
```

```
var distance = plane.DistanceTo(point);
```

This will, again, return either a positive or negative distance.

Flipping the polarity of the plane can be done by negating both N and D. This will result in a plane in the same position, but with inverted negative and positive half spaces:

GDScript

C#

```
N = -N
D = -D
```

```
N = -N;
D = -D;
```

Of course, Godot also implements this operator in *Plane*, so doing:

GDScript

C#

```
var inverted_plane = -plane
```

```
var invertedPlane = -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.

GDScript

C#

```
var N = normal
var D = normal.dot(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:

GDScript

C#

```
# 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
```

```
// calculate vector from a to b
var dvec = (pointB - pointA).Normalized();
// rotate 90 degrees
var normal = new Vector2(dvec.y, -dvec.x);
// or alternatively
// var normal = new 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:

GDScript

C#

```
var N = normal
var D = normal.dot(point_a)
# this works the same
# var D = normal.dot(point_b)
```

```
var N = normal;
var D = normal.Dot(pointA);
// this works the same
// var D = normal.Dot(pointB);
```

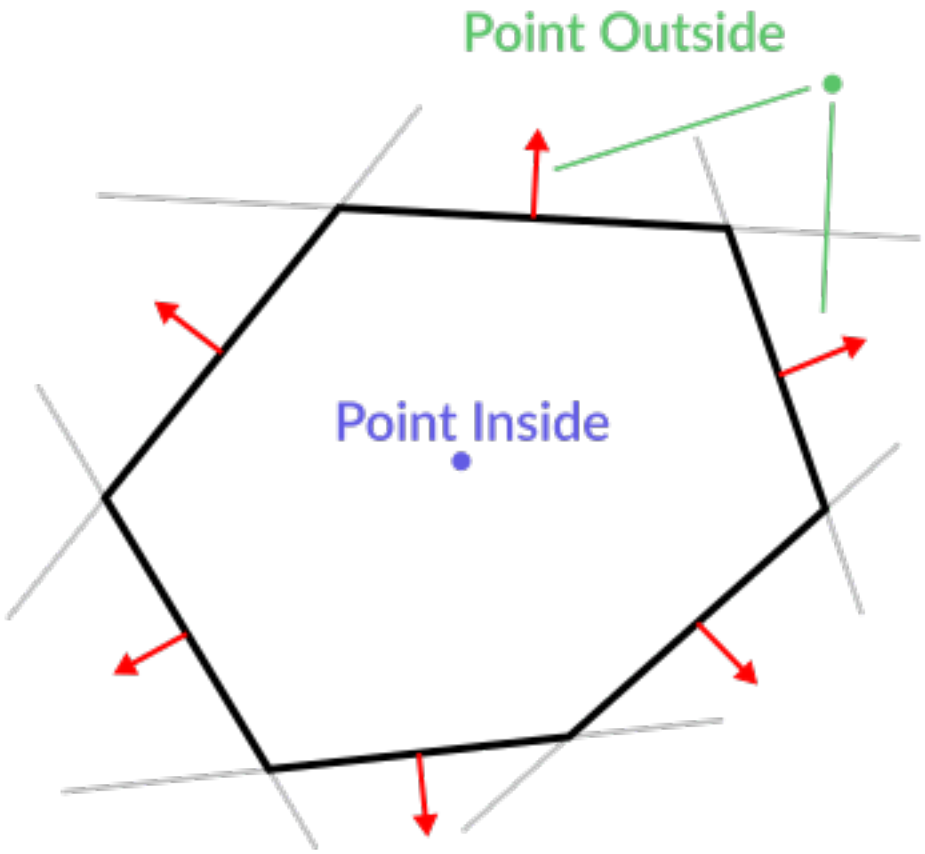
Doing the same in 3D is a little more complex and will be explained further down.

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:

GDScript

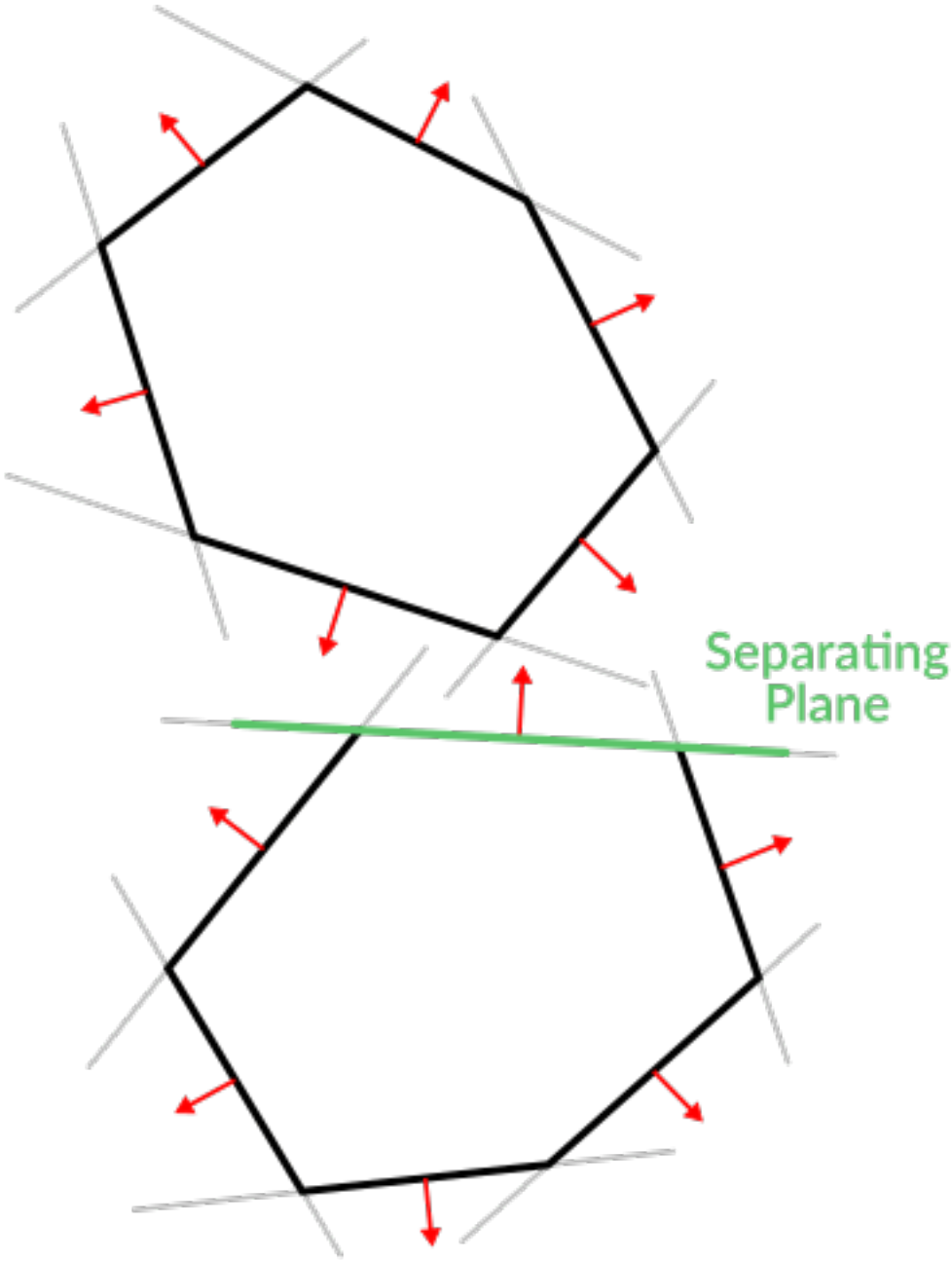
C#

```
var inside = true
for p in planes:
    # check if distance to plane is positive
    if (p.distance_to(point) > 0):
        inside = false
        break # with one that fails, it's enough
```

```
var inside = true;
foreach (var p in planes)
{
    // check if distance to plane is positive
    if (p.DistanceTo(point) > 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.

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:

GDScript

C#

```
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):
```

(continues on next page)

(continued from previous page)

```
            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!")
```

```
var overlapping = true;

foreach (Plane plane in planesOfA)
{
    var allOut = true;
    foreach (Vector3 point in pointsOfB)
    {
        if (plane.DistanceTo(point) < 0)
        {
            allOut = false;
            break;
        }
    }

    if (allOut)
    {
        // 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
    foreach (Plane plane in planesOfB)
    {
        var allOut = true;
```

(continues on next page)

(continued from previous page)

```
        foreach (Vector3 point in pointsOfA)
        {
            if (plane.DistanceTo(point) < 0)
            {
                allOut = false;
                break;
            }
        }

        if (allOut)
        {
            overlapping = false;
            break;
        }
    }
}

if (overlapping)
{
    GD.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

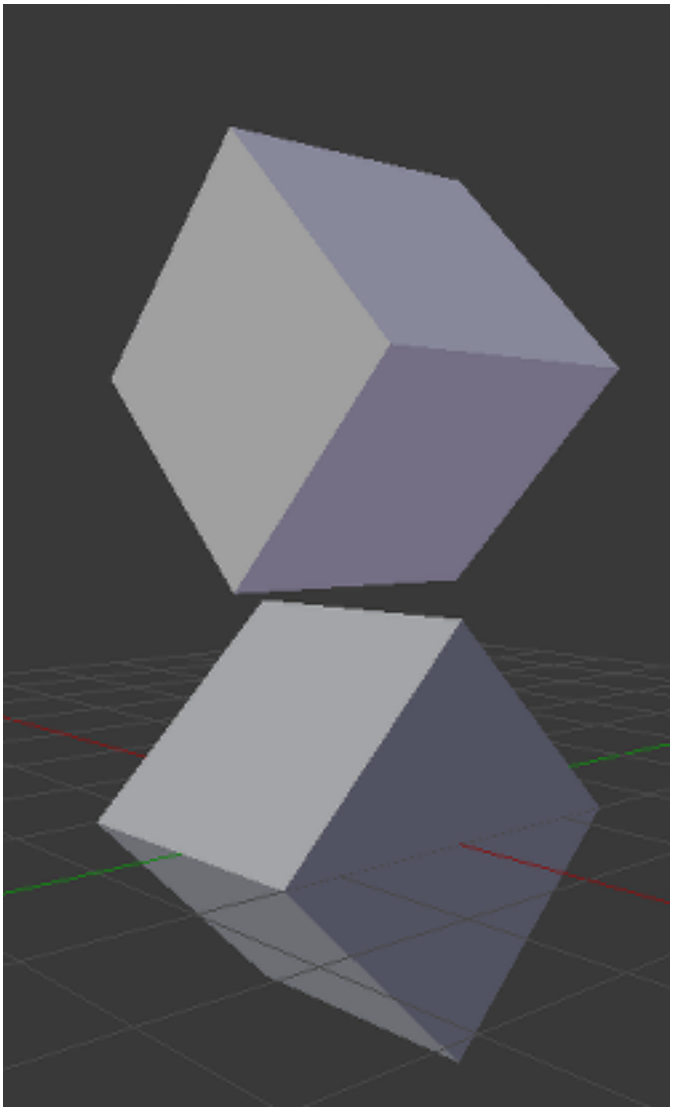
This is another bonus bit, a reward for being patient and keeping up with this long tutorial. Here is another piece of wisdom. This might not be something with a direct use case (Godot already does collision detection pretty well) but it's used by almost all physics engines and collision detection libraries :)

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.

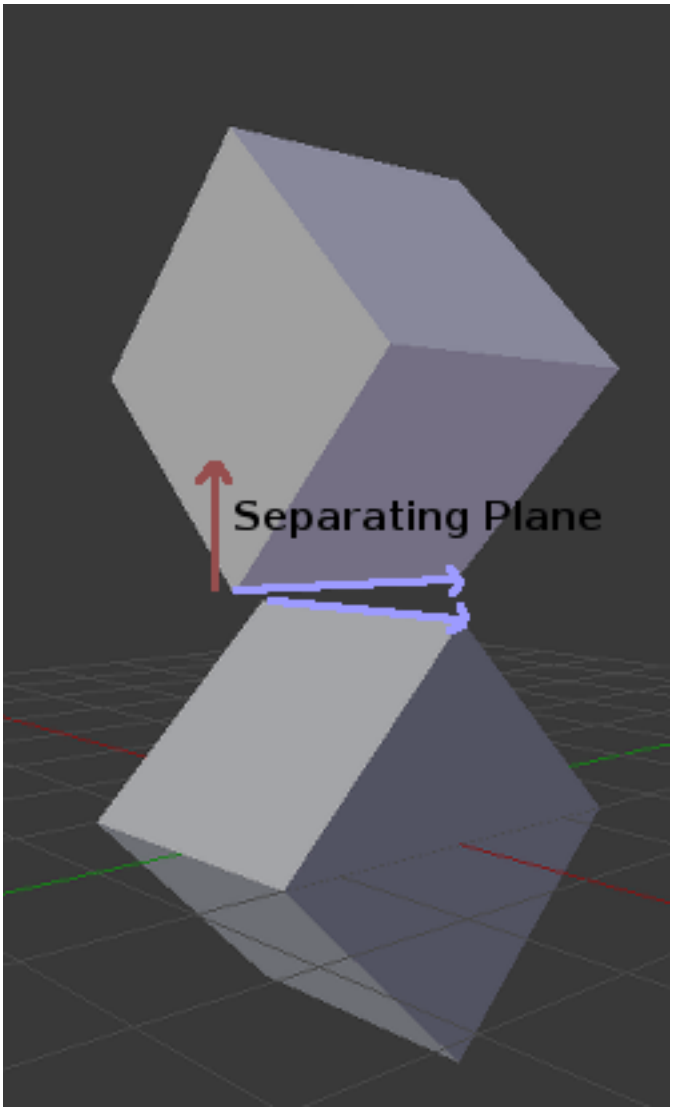
Well, this works in 3D too, if two 3D polyhedral shapes are colliding, you won't be able to find a separating plane. If a separating plane is found, then the shapes are definitely not colliding.

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:

GDScript

C#

```
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
```

(continues on next page)

(continued from previous page)

```
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)
                max_A = max(max_A, d)
                min_A = min(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)
                max_B = max(max_B, d)
                min_B = min(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!")
```

```
var overlapping = true;

```

(continues on next page)

(continued from previous page)

```
foreach (Plane plane in planesOfA)
{
    var allOut = true;
    foreach (Vector3 point in pointsOfB)
    {
        if (plane.DistanceTo(point) < 0)
        {
            allOut = false;
            break;
        }
    }

    if (allOut)
    {
        // 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
    foreach (Plane plane in planesOfB)
    {
        var allOut = true;
        foreach (Vector3 point in pointsOfA)
        {
            if (plane.DistanceTo(point) < 0)
            {
                allOut = false;
                break;
            }
        }

        if (allOut)
        {
            overlapping = false;
            break;
        }
    }
}

if (overlapping)
{
    foreach (Vector3 edgeA in edgesOfA)
    {
        foreach (Vector3 edgeB in edgesOfB)
        {
            var normal = edgeA.Cross(edgeB);
            if (normal.Length() == 0)
            {
                continue;
            }
```

(continues on next page)

(continued from previous page)

```
            var maxA = float.MinValue; // tiny number
            var minA = float.MaxValue; // 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.

            foreach (Vector3 point in pointsOfA)
            {
                var distance = normal.Dot(point);
                maxA = Mathf.Max(maxA, distance);
                minA = Mathf.Min(minA, distance);
            }

            var maxB = float.MinValue; // tiny number
            var minB = float.MaxValue; // huge number

            foreach (Vector3 point in pointsOfB)
            {
                var distance = normal.Dot(point);
                maxB = Mathf.Max(maxB, distance);
                minB = Mathf.Min(minB, distance);
            }

            if (minA > maxB || minB > maxA)
            {
                // not overlapping!
                overlapping = false;
                break;
            }
        }

        if (!overlapping)
        {
            break;
        }

    }
}

if (overlapping)
{
    GD.Print("Polygons Collided!");
}
```

10.3 Rotations

In the context of 3D transforms, rotations are the nontrivial and complicated part. While it is possible to describe 3D rotations using geometrical drawings and derive the rotation matrices, which is what most people would be more familiar with, it offers a limited picture, and fails to give any insight on related practical things such quaternions and SLERP. For this reason, this document takes a different approach, a general (although a little abstract at first) approach which was popularized by its extensive use in local gauge theories in physics. That being said, the aim of this text is to provide a minimal background to understand 2D and 3D rotations in a general way and shed light on practical things

such as gimbal lock, quaternions and SLERP using an accessible language for programmers, and not completeness or mathematical rigor.

10.3.1 A crash course in Lie groups and algebras for programmers

Lie groups is a branch of mathematics which deals with rotations in a systematic way. While it is a extensive subject and most programmers haven't even heard of it, it is relevant in game development, because it provides a coherent and unified view of 3D rotations.

So let's start with small steps. Suppose that you want to make a tiny rotation around some axis. For, concreteness let's say around z-axis by an infinitesimal angle $\delta\theta$. For small angles, as illustrated in the figure, the rotation operator can be written as $R_z(\delta\theta) = I + \delta\theta \boldsymbol{e}_z\times$ (neglecting higher order terms $\mathcal{O}(\delta\theta^2)$) where I is identity operator and $\boldsymbol{e}_z$ is a unit vector along the z-axis, such that a vector $\boldsymbol{v}$ becomes

$$\boldsymbol{v}' = R_z(\delta\theta)\boldsymbol{v} = \boldsymbol{v} + \delta\theta \boldsymbol{e}_z \times \boldsymbol{v}.$$

(If this isn't clear, you can verify this by looking at the figure: $\boldsymbol{v}$ is rotated into $\boldsymbol{v}'$ in the plane (so the rotation axis $\boldsymbol{e}_z$ is pointing out of screen). The overlap of two vectors is $\boldsymbol{v}\cdot\boldsymbol{v}' = v^2\cos\delta\theta \approx v^2 + \mathcal{O}(\delta\theta^2)$ since the rotation is just a tiny amount, so the part of $\boldsymbol{v}'$ parallel to $\boldsymbol{v}$ is indeed $\boldsymbol{v}$. Here, $v = |\boldsymbol{v}|$. What about the perpendicular part, which must be $\boldsymbol{v}' - \boldsymbol{v}$? Using the right-hand rule, the direction is $\boldsymbol{e}_z \times \boldsymbol{v}/v$, and to the first order in $\delta\theta$, we can approximate the length of the difference vector by the arc length (blue arc in the figure) which is $\delta\theta v$, so the perpendicular component $\delta\theta\boldsymbol{e}_z \times \boldsymbol{v}$ also checks out.)

Now, a practical way of representing operators is by using matrices (note that an operator is not a matrix, and there are different mathematical objects other than matrices which be used to represent operators, such as quaternions, a point which we will come back to later when we're discussing representations . In terms of real 3×3 matrices, the identity operator I simply corresponds to a 3×3 identity matrix, and the cross product $\boldsymbol{e}_z\times$ can be represented as

$$J_z \cong \boldsymbol{e}_z\times \equiv \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

(If you're curious how you can find the matrix representation of an operator A in some set of real basis $\{\boldsymbol{e}_i\}$: the matrix element on i th row j th column is given by $\boldsymbol{e}_i\cdot(A\boldsymbol{e}_j)$; the basis we use here is $\{\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z\}$) It is no accident that J_z rotates a vector in the xy plane around the z-axis by $\pi/2$; for an arbitrary axis $\boldsymbol{n}$, the operator $J_n \cong \boldsymbol{n}\times$ is a rotation by $\pi/2$ around that axis for vectors in the plane perpendicular to $\boldsymbol{n}$.

In terms of J_z, we can express the infinitesimal rotation operator around the z-axis as

$$R_z(\delta\theta) = I + \delta\theta J_z.$$

So, how about finite rotations? We simply can apply this infinitesimal rotation operator N times to obtain a finite rotation $\theta \equiv N\delta\theta$:

$$R_z(\theta) = \lim_{N\to\infty} R_z^N(\delta\theta) = \left(I + \frac{\theta}{N}J_z\right)^N = e^{\theta J_z}.$$

(If you're confused about seeing a matrix as an exponent: the meaning of an operator A in exponential map is given by its series expansion as $e^A = 1 + A + A^2/2! + \dots$). This is arguably the most important relation in this write up, and lies at the heart of Lie groups, whose significance will be clarified in a moment. But first, let's take step back and observe the significance of this result: using a simple picture of an infinitesimal rotation, we derived a general expression for arbitrary rotations around the z-axis. In fact, this gets even better. If we repeated the same analysis for rotations around x- and y-axes, we would have obtained similar results $e^{\theta J_x}$ and $e^{\theta J_y}$ respectively, where

$$J_x \cong \boldsymbol{e}_x\times \equiv \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad J_y \cong \boldsymbol{e}_y\times \equiv \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Or, if we did a rotation around an arbitrary axis $\boldsymbol{n}$, the result would have been

$$R_n(\theta) = \lim_{N\to\infty}\left(I + \frac{\theta}{N}\boldsymbol{n}\times\right)^N = \lim_{N\to\infty}\left(I + \frac{\theta}{N}\boldsymbol{n}\cdot\boldsymbol{J}\right)^N = e^{\theta\boldsymbol{n}\cdot\boldsymbol{J}},$$

where $\boldsymbol{J} = (J_x, J_y, J_z)$. Note how the rotation axis $\boldsymbol{n}$ and rotation angle θ plays a central role in the final expression. Axis-angle is *the* parametrization for *all* Lie groups, not just 3D rotations. (We will come back to this point later when we're discussing Euler angle parametrization, which is an unnatural and defective parametrization of rotations in 3D.)

If you ever tried to derive the rotation matrix corresponding to a θ rotation around $\boldsymbol{n}$, you can appreciate the simplicity and elegance of how we obtained this result. To be fair, we still need to figure out how to actually use an operator sitting on top of an exponent (by summing up the Taylor series), of course, but that's merely a "programmatic" labor and you just need to follow the finite multiplication table of J_i operators. But this is much straightforward than trying to draw geometric diagrams and angles and figure out the rotation matrix. For the particular case of the algebra corresponding to rotations in 3D Euclidean space that we've been talking about so far, the exponent can simply be written as

$$R_n(\theta) = e^{\theta\boldsymbol{n}\cdot\boldsymbol{J}} = I + 2\cos\frac{\theta}{2}\sin\frac{\theta}{2}(\boldsymbol{n}\cdot\boldsymbol{J}) + 2\sin^2\frac{\theta}{2}(\boldsymbol{n}\cdot\boldsymbol{J})^2$$

which is known as Rodrigues' rotation formula. Note that we only ended up with terms up to second order in $\boldsymbol{n}\cdot\boldsymbol{J}$ when we summed the series expansion; the reason is higher powers can be reduced to 0th, 1st or 2nd order terms due to the relation $(\boldsymbol{n}\cdot\boldsymbol{J})^3 = -\boldsymbol{n}\cdot\boldsymbol{J}$, which makes summing up the series straightforward.

The thing that sits on top of e, which is a linear combination J_i operators (where $i = x, y, z$), forms an algebra; in fact, it forms a vector space whose basis "vectors" are J_i. Furthermore, the algebra is closed under the Lie bracket (which is essentially a commutator: $[a, b] = ab - ba$, and is something like a cross-product in this vector space). In the particular case of 3D rotations, this "multiplication table" is $[J_x, J_y] = J_z$ and its cyclic permutations $x \to y, y \to z, z \to x$.

Rotations form what is called a group: simply put, it means that if you combine two rotations, you get another rotation. And you can observe it here too: when you put an element of the Lie algebra (which are simply linear combinations of J_i) on top of e, you get what is called a Lie group, and the Lie algebra is said to *generate* the Lie group. For example, the operator $J_z \equiv \boldsymbol{e}_z\times$ is said to *generate* the rotations around the z-axis. The group of rotations in the 3D Euclidean space is called SO(3).

The order of rotations in 2D don't matter: you can first rotate by π and rotate by $\pi/2$, or do it in reverse order, and either way, the result is a rotation by $3\pi/2$ in the plane. But the order of rotations in 3D do matter, in general, when different rotation axes are involved (see this picture for an example) (rotations around the same axes do commute, of course). When the ordering of group elements don't matter, that group is said to be Abelian, and non-Abelian otherwise. SO(2) is an Abelian group, and SO(3) is a non-Abelian group.

Lie groups and algebras are *not* matrices. You can *represent* both by using object which emulate their "multiplication" rules: this can be real or complex matrices of varying dimensions, or something like quaternions. A single Lie group/algebra have infinitely many different representations in vector spaces in different dimensions (see these for example for SO(3)). Above, we use the 3D real representation of SO(3), which happens to be the fundamental representation, and accidentally coincides with the adjoint representation.

Some mathematical remarks (feel free to skip)

There are many different Lie groups, corresponding to different symmetries, and they all have different names. For example, the group which contains all rotations in n-dimensional Euclidean space is called SO(n), and has $n(n-1)/2$ linearly independent generators (yes, Lie groups can handle rotations is higher dimensions as-is, and even in non-Euclidean ones). This is called the *rank* of the Lie algebra. You can think of the generators as independent axes of rotations. For 2D, we can only rotate in the xy plane meaning we have only 1 generator. For 3D, you can rotate around 3 different planes/axes.

Lie groups have deep connections with symmetries, and have played central role in theoretical physics since around mid 20th century.

For example, if something is symmetric under 3D rotation, that something (in physics, it is typically Lagrangian, which leads to conservation laws through Noether's theorem) remains invariant under SO(3) transformations (we will cover transformations below).

In the context of Lie groups and group theory in general, some common words have specific meanings and a part of the math jargon: representation, generator, group, algebra, parametrization, operator are such words. You don't need to know their precise definitions to understand this write up; just be aware that they are special terms and may not mean what you think they mean. All of these terms a described in this write up in a colloquial language.

10.3.2 Representation of rotations

Representation of rotations is an independent concept from parametrization of rotations. These two concepts are commonly conflated, which leads to the current state of confusion among many programmers. People tend to associate Euler angles parametrization with matrices (or sometimes even vectors!), and axis-angle parametrization with quaternions.

In game engines, rotation operators are represented using either matrices, or quaternions. *As will be clear in what follows, you can use a matrix or a quaternion to represent a rotation parameterized using Euler angles, and same goes for axis-angle parametrization.* Unfortunately, even graphics programming books and documentations of expensive game engines often make a mistake here, and this causes programmers to start comparing Euler angles (a parametrization) to quaternions (a representation) and even discussing their trade-offs, which is "not even wrong".

Representation here refers to a technical term in group theory. So will many other things that will be mentioned in what follows. To gain a basic understand of these concepts, let's first go through simpler and better understood example of rotations in 2D first.

Representation of rotations in 2D

Since there is only one possible axis of rotation in a two dimensional plane, there is no Euler angle parametrization for them (or if you like, there is only one Euler-angle). Rather, axis-angle parametrization is used, with the axis being fixed to z-axis, which leaves only the angle of rotation φ as a free parameter.

A point in the 2D Euclidean space can be represented by a pair of 2D real numbers as $\boldsymbol{v} = (x, y)$ (called vector representation), or they can alternatively be represented by a complex number as $v = x + \imath y$ where $\imath \equiv \sqrt{-1}$ is the unit imaginary number. In the vector representation, we can rotate the point through a rotation matrix (an element of the Lie group SO(2), which can be represented by 2×2 orthogonal matrices with determinant +1) as follows:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

So for example, when $\theta = \pi/2$, we get $R(\pi/2)\boldsymbol{v} = (-y, x)$.

In the complex representation, a rotation is represented by a unit complex number $e^{\imath\theta} = \cos\theta + \imath\sin\theta$, where we used Euler's formula, is an element of the Lie group U(1), which can be represented by complex numbers of unit norm. Again, for $\theta = \pi/2$, you recover $e^{\imath\pi/2}(x + \imath y) = \imath(x + \imath y) = (-y) + \imath x$.

Rotations in the complex number representation look simpler, but it's only an illusion: the complications of performing a matrix multiplication is absorbed by the introduction of something that lives outside of the realm of real numbers, which follows a rather "odd" algebra: $\imath^2 = -1$. The way complex numbers mimic 2D rotations can be made clearer if we rewrite the rotation matrix in terms of

$$I_2 \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad J_z \equiv \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

as $R(\theta) = I_2\cos\theta + J_z\sin\theta$, which can then be compared to $1x + \imath y$ directly. Now we can see the equivalence (the technical term is isomorphism in this context) of the representations clearer through their multiplication table: $I_2 I_2 = I_2, I_2 J_z = J_z, J_z I_2 = J_z, J_z J_z = -I_2$ which behaves the same way as $1 \times 1 = 1, 1 \times \imath = \imath, \imath \times 1 =$

$\imath, \imath \times \imath = -1$. Also note that both $\imath$ and J_z represent a $\pi/2$ rotation. And as it should be, $\imath$ and J_z behave the same under multiplication.

Furthermore, by Taylor series expansion, it is straightforward to show that $R(\theta) = e^{J_z \theta}$.

We have then the following table:

What	Matrix representation of SO(2)	Complex representation of U(1)
Vector	(x, y)	$x + \imath y$
Generator	$J_z \cong \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$J_z \cong \imath$
Rotation operator	$e^{J_z \theta} \equiv I_2 \cos\theta + J_z \sin\theta \equiv \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$	$e^{J_z \theta} \equiv e^{\imath\theta} = 1 \cos\theta + \imath \sin\theta$

Clearly, introduction of the unit imaginary number is enough to capture the behavior of 2D rotation matrices. As a footnote remark, while the number $e^{\imath\theta}$ can only represent a rotation matrix, it can't of course represent an arbitrary 2×2 matrix (meaning no scaling, no shearing, etc): after all, U(1) isn't isomorphic to $\mathrm{GL}(2, \mathbb{R})$, the group of all (invertible) real 2×2 matrices.

The equivalence of these two seemingly different way of representing vectors and rotations in 2D lies in the isomorphism between the Lie groups SO(2) and U(1).

Furthermore, in this representation, it is clear that you can do a "smooth" rotation by slowly changing θ, which is the 2D analogue of SLERP (could as well be called Circular Linear Interpolation!). Note that if you linearly interpolate the *elements* of two rotation matrices (that is, linearly interpolating between R_{ij} and R'_{ij}), you'll get a weird trajectory with jerky motion; to do SLERP with a matrix, you need to extract the angles from each matrix (which can only happen for matrices whose entries have to form given by $R(\theta)$ above; that is, elements of SO(2)), interpolate between the angles linearly, and construct the intermediate matrix using that angle.

The take-aways of this short visit to the more understandable 2D land are:

1. There can be different (but "equivalent", of course) *representations* of rotations: like matrices and complex numbers.
2. Despite the fact that you can use complex numbers to represent vectors and rotations in 2D, the *concept* of rotations in 2D doesn't require an understanding/knowledge of complex numbers or Euler's formula.
3. The introduction of the imaginary $\imath$ is not black magic: it's just something that mimics 2×2 matrix J_z: 1 and $\imath$ behave the same way as I_2 and J_z under multiplication (see the group multiplication table given above).

These are often sources of confusion in 3D: introducing a third dimension means we have new rotation axes (rotations around X and Y axes) giving rise to alternative parametrizations (such as Euler angles), and new generators J_x and J_y, which will be the generalization of J_z above.

Some mathematical remarks (again, feel free to skip)

The fact that SO(3) has 3 generators is an accident: SO(n) has $n(n-1)/2$ generators. Furthermore, the next step (after quaternions, which is another accident) of Cayley-Dickson construction does *not* correspond to a Lie algebra, but rather a non-associative algebra called octonions. Rather, in arbitrary dimensions, the "complex" representation can be written using the generators of Spin(n), which is a double cover of SO(n). Also, throughout this page, when we say representation of SO(2), U(1) or any other group, we are talking about the *fundamental* irreducible representation, corresponding to a Young diagram with a single box.

Representation of rotations in 3D

Let's first review how 3D rotations work using familiar vectors and matrices.

In 2D, we considered vectors lying in the xy plane, and the only axis we could can rotate them was the z-axis. In 3D, we can perform a rotation around any axis. And this doesn't just mean around x, y, z axes, the rotation can also be around an axis which is a linear combination of those, where $\boldsymbol{n}$ is the unit vector (meaning $\boldsymbol{n} \cdot \boldsymbol{n} = 1$) aligned with the axis we want to perform the rotation.

Just like the 2D rotation matrix, the 3D rotation matrix can also be derived with some effort by drawing lots of arrows and angles and some linear algebra, but this would be opaque and won't give us much insight to what's going on. A less straightforward, but more rewarding way of deriving this matrix is to understand the rotation group SO(3).

SO(3) is the group of rotations in Euclidean 3D space (for which the signature is $(+1, +1, +1)$), which preserve the magnitude and handedness of the vectors it acts on. The most typical way to represent its elements is to use 3×3 real orthogonal matrices with determinant $+1$. This $\mathrm{Mat}(3, \mathbb{R})$ representation is called the fundamental representation of SO(3).

To recap what we discussed earlier, SO(3) has 3 generators, J_x, J_y, J_z and we found that they can be represented using these 3×3 real matrices:

$$J_x \cong \boldsymbol{e}_x \times \equiv \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad J_y \cong \boldsymbol{e}_y \times \equiv \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad J_z \cong \boldsymbol{e}_z \times \equiv \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

These matrices have the same "multiplication table" as J_i (they're isomorphic), so for all practical purposes, you can replace the operators with their matrix representations.

We also found that an element of SO(3), that is, a rotation operator is

$$R_n(\varphi) = e^{\varphi \boldsymbol{n} \cdot \boldsymbol{J}} = I + 2 \cos \frac{\varphi}{2} \sin \frac{\varphi}{2} (\boldsymbol{n} \cdot \boldsymbol{J}) + 2 \sin^2 \frac{\varphi}{2} (\boldsymbol{n} \cdot \boldsymbol{J})^2$$

If you want, you can plug-in the matrix representations for J_i and derive the complicated 3×3 rotation matrix which is

$$R_n(\varphi) = \begin{pmatrix} \cos\varphi + n_x^2 (1 - \cos\varphi) & n_x n_y (1 - \cos\varphi) - n_z \sin\varphi & n_x n_z (1 - \cos\varphi) + n_y \sin\varphi \\ n_y n_x (1 - \cos\varphi) + n_z \sin\varphi & \cos\varphi + n_y^2 (1 - \cos\varphi) & n_y n_z (1 - \cos\varphi) - n_x \sin\varphi \\ n_z n_x (1 - \cos\varphi) - n_y \sin\varphi & n_z n_y (1 - \cos\varphi) + n_x \sin\varphi & \cos\varphi + n_z^2 (1 - \cos\varphi). \end{pmatrix}$$

(Hint: you can use the relation $(\boldsymbol{n} \cdot \boldsymbol{J})^2 = \boldsymbol{n} \otimes \boldsymbol{n} - I$ to quickly evaluate the last term in the Rodrigues' formula, where $\otimes$ is the Kronecker product which is also called outer product for vectors. Using the half-angle formulae to rewrite $\sin\varphi = 2 \cos\frac{\varphi}{2} \sin\frac{\varphi}{2}$ and $1 - \cos\varphi = 2 \sin^2\frac{\varphi}{2}$ in Rodrigues' formula, you can use cosine and sine terms as a visual aid when comparing to the matrix form.)

However, we don't *have to* use matrices to represent SO(3) generators J_i. Remember how we used $\imath$, the imaginary unit to emulate J_z rather than using a 2×2 matrix? As it turns out we can do something similar here.

Hamilton is mostly commonly known for the omnipresent Hamiltonian in physics. One of his less known contributions is essentially an alternative way of representing 3D cross product, which eventually gave in to popularity of usual vector cross products. He essentially realized that there are three different non-commuting rotations in 3D, and gave a name to the generator for each. He identified the operators $\{\boldsymbol{e}_x \times, \boldsymbol{e}_y \times, \boldsymbol{e}_z \times\}$ as the elements of an algebra, naming them as $\{i, j, k\}$.

This may sound trivial at this point, because we're equipped with all the machinery of Lie groups and Lie algebras: apparently, quaternion units $\{i, j, k\}$ are just another representation of the SO(3) generators, which satisfy the Lie bracket. Well, no so fast. While the Lie *algebra* $\mathfrak{so}(3)$, whose elements are the linear combination of J_i s are isomorphic to unit quaternions, but quaternions are $1w + xi + yj + zk$ in general, so there's also an identity part, which isn't a vector that is a part of any Lie algebra. Quaternions look more like the *group* SO(3) (when they're normalized, because SO(3) preserves vector norms). But it actually isn't isomorphic to SO(3). It turns out that unit quaternions are isomorphic to the group SU(2) (which is isomorphic to Spin(3)), which in turn is a double cover of SO(3).

SU(2) is essentially the group of unitary rotations with determinant +1 (called Special Unitary groups) which preserve the norm of complex vectors it acts on, generated by Pauli spin matrices σ_i, and i, j, k correspond to $\sigma_x/\imath\, \sigma_y/\imath, \sigma_z/\imath$.

To exemplify, $R = e^{\varphi \boldsymbol{n}\cdot\boldsymbol{J}} \in \mathrm{SO}(3)$ rotates a real vector by $R\boldsymbol{v}$ and the corresponding rotation $U = e^{-\imath\varphi \boldsymbol{n}\cdot\boldsymbol{\sigma}/2} \in \mathrm{SU}(2)$ rotates the same vector through $U(\boldsymbol{v}\cdot\boldsymbol{\sigma})U^\dagger$. Note that $U \to -U$ achieves the same SO(3) rotation, SU(2) it's said to be a double cover of SO(3) (this is mapping gives the adjoint representation of SU(2) by the way). Here $-\imath\boldsymbol{\sigma} = -\imath(\sigma_x, \sigma_y, \sigma_z) \cong (i, j, k)$.

SU(2) and SO(3) look the same locally (their tangent spaces dictated by their Lie algebras are isomorphic), but they're different globally. While this sounds like just a technicality, this has topological implications, but we won't get into that much. The take away from this discussion is that unit quaternions *can* be used emulate SO(3) rotations.

But taking a step back, why do we bother *emulating* SO(3) at all? For computational purposes, we already have something that works: the matrix representation. Why do we need to both with a weird group that isn't even exactly the same as SO(3)?

The answer is the cost of computation, and this is two fold. First, you see, a rotation operator has only 3 degrees of freedom: two for the unit vector which is the rotation axis, and one for the rotation angle around that axis. A 3×3 matrix, on the other hand has 9 elements. It's an overkill. For example, whenever you multiply two rotations, you need to multiply two 3×3 matrices, summing and multiplying every single element. In terms of CPU cycles, this is wasted effort and we can be more optimal. Second part is precision errors. The errors are worse in matrix representation, because originally, we have only 3 degrees of freedom, which means we can have precision errors in axis and angle (only 3 errors) but it's still an element of SO(3), whereas with matrices, we can have errors in any one of the 9 elements in the matrix and so we can even have a matrix that isn't even an element of SO(3). These errors can quickly build up quickly especially if you're for example modifying the orientation of an object every frame by doing a smooth interpolation between an initial and a target orientation (discussed further in SLERP section).

Sure, we know that elements of SO(3) can be represented by using orthogonal matrices with determinant +1 (hence the name Special Orthogonal) such that $RR^T = I$; in plain language, this means the columns of R form an orthonormal set of vectors, so we can eliminate the errors if we perform Gram-Schmidt orthonormalization once in a while, and force it back into SO(3), such that it's an actual rotation matrix (albeit still noisy in axis and angle). But this is expensive and still quite bad in terms of errors.

So, what is the alternative then? Shall we go back to the Rodrigues' formula and hardcode the behavior of J_i into our program? A nicer alternative is, we use SU(2) (which we know covers SO(3), twice in fact!), because the equivalent of the Rodrigues' formula is much simpler:

$$U = e^{-\imath\varphi \boldsymbol{n}\cdot\boldsymbol{\sigma}/2} = I\cos\frac{\varphi}{2} - \imath(\boldsymbol{n}\cdot\boldsymbol{\sigma})\sin\frac{\varphi}{2}$$

owing to the nice relation $(\boldsymbol{n}\cdot\boldsymbol{\sigma})^2 = I$ (something like this happens only at the third power with SO(3) generators, remember? and it doesn't give identity), or if you prefer quaternion version which is more commonly used to represent the isomorphic group Spin(3), this is

$$U = e^{\varphi \boldsymbol{n}\cdot(i,j,k)/2} = I\cos\frac{\varphi}{2} + \boldsymbol{n}\cdot(i, j, k)\sin\frac{\varphi}{2}.$$

In game engines, rather than storing the axis-angle $(\boldsymbol{n}(\phi, \theta), \varphi)$ pair where ϕ, θ are the azimuthal and polar angles parametrizing the unit vector $\boldsymbol{n}$, people typically store $\boldsymbol{q} = (q_0, q_x, q_y, q_z) \equiv (\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}n_x, \sin\frac{\varphi}{2}n_y, \sin\frac{\varphi}{2}n_z)$ such that $U = \boldsymbol{q}\cdot(1, i, j, k)$, and enforce the condition $|\boldsymbol{q}| = 1$ once in a while by renormalization (note that while you can see many people referring to $\boldsymbol{q}$ as a quaternion, it's not; U is the actual quaternion here and $\boldsymbol{q}$ is just an artificial vector containing the coefficients in the expansion of the exponential map). Of course, if they store (φ, ϕ, θ), there is no need for a renormalization because such parametrization guarantees the normalization, but this choice would come at the cost of calculating a bunch of sines and cosines whenever you use them, so this is a middle ground in terms of errors and speed.

So, the take aways of this section are:

1. Matrices can represent SO(3) just fine, but are a little too general and extravagant in terms of CPU cycles and behave bad in the presence of floating point errors, and errors can even lead to a matrix that doesn't correspond to a rotation matrix at all!

2. For all practical purposes, we can use an element of SU(2) to represent an SO(3) rotation. It's a double cover of SO(3), so we wouldn't be losing anything in doing so. The main reason for choosing one over another is the SO(3) Rodrigues' formula is a little nasty to work with whereas SU(2) expansion is neat, clean and simple to work with.

3. Using matrices, you can practically do everything you do with quaternions, vice versa. The real differences, as highlighted, are in computation trade-offs, not mathematics.

3. The relationship between quaternions and 3D rotation matrices is the roughly the same as the relation between the complex number $e^{\imath\theta}$ and a 2D rotation matrix. Just as the complex number $\imath \cong J_z$ rotates by $\pi/2$ (which is, as we saw, what a *generator* does), i, j, k (which are $\cong J_x, J_y, J_z$) rotate by $\pi/2$ around x, y, z axes; they don't commute with each other because in 3D, the order of rotations is important. Owing to this isomorphism between their generators, an SO(3) rotation $e^{\varphi \boldsymbol{n}\cdot\boldsymbol{J}}$ corresponds to the SU(2) rotation $e^{\varphi \boldsymbol{n}\cdot(i,j,k)/2}$. This is a helpful picture to gain an intuition on quaternions. While the SO(3) is familiar to many people, the "Rodrigues' formula" for the SU(2) one is much preferable graphics programming due to it's simplicity, and hence you see quaternions in game engines.

4. This doesn't mean quaternions are a generalization of complex numbers in our construction when considering rotations in a strict sense; they're rather the 3D generalization of the 2D rotation generator in some loose sense (loose, because SO(3) and SU(2) are different groups). There *is* a construction which generalizes real numbers to complex numbers to quaternions, called Cayley-Dickson construction, but it's next steps give off topic objects octonions and sedenions (setting exceptional Lie groups aside for octonions).

5. Note that we haven't said a word about Euler angles. In both matrix and quaternion *representations*, we sticked to the axis-angle *parametrization*. We will discuss different parametrizations in what follows.

What	Matrix representation of SO(3)	Quaternion representation of SU(2)
Vector	(x, y, z)	$\sqrt{r}(\cos\frac{\theta}{2}, e^{\imath\phi}\sin\frac{\theta}{2})$
Generator	Matrices for $J_x, J_y, J_z \cong \boldsymbol{e}_x\times, \boldsymbol{e}_y\times, \boldsymbol{e}_z\times$	i, j, k
Rotation operator	$e^{\varphi \boldsymbol{n}\cdot\boldsymbol{J}}$, can expand with Rodrigues' formula	$e^{\varphi \boldsymbol{n}\cdot(i,j,k)/2}$ can expand with SU(2) "Rodrigues' formula"

Above, r, θ, ϕ are spherical coordinates corresponding to x, y, z.

A note about the SU(2) vector

We earlier mentioned that rotation of a real vector in SU(2) is done via $(R\boldsymbol{v})\cdot\boldsymbol{\sigma} = U(\boldsymbol{v}\cdot\boldsymbol{\sigma})U^\dagger$, and you may be wondering what the complex vector listed above $|\psi\rangle = (\cos\frac{\theta}{2}, e^{\imath\phi}\sin\frac{\theta}{2})$ has to do with that. The answer is, there vector $|\psi\rangle$ is the one a single U acts on, and in terms of it, we can rewrite $U(\boldsymbol{v}\cdot\boldsymbol{\sigma})U^\dagger$ as $rU(|\psi\rangle\otimes\langle\psi|)U^\dagger$ up to some trivial identity term, where $\langle\psi| = |\psi\rangle^\dagger$ (this is the way complex vectors are typically denoted in quantum mechanics and is called bra-ket notation: bra is like the vector and ket is the conjugate transpose, and people typically omit the $\otimes$ in between because that's already implied when you multiply a ket with a bra, and likewise there is no $\cdot$ when you multiply a bra with ket since that's also implied).

So, notational concerns aside, long story short, the vector listed above is in a generalized sense "half" of what we are rotating:

$$\frac{\boldsymbol{v}\cdot\boldsymbol{\sigma} + I|\boldsymbol{v}|}{2} = r|\psi\rangle\langle\psi| = r\left(\cos\frac{\theta}{2}, e^{\imath\phi}\sin\frac{\theta}{2}\right)\otimes\left(\cos\frac{\theta}{2}, e^{\imath\phi}\sin\frac{\theta}{2}\right)^\dagger$$

and each "half" goes to one of the U s on each side of the modified/generalized relation

$$r = |\psi'\rangle\langle\psi'| = \frac{(R\boldsymbol{v})\cdot\boldsymbol{\sigma} + I|\boldsymbol{v}|}{2} = rU|\psi\rangle\langle\psi|U^\dagger = U\left(\frac{\boldsymbol{v}\cdot\boldsymbol{\sigma} + I|\boldsymbol{v}|}{2}\right)U^\dagger$$

so everything is compatible, and $|\psi'\rangle = U|\psi(\boldsymbol{v})\rangle = |\psi(R\boldsymbol{v})\rangle$ is satisfied. This parametrization of an SU(2) vector is typically done in spherical coordinates θ, ϕ (for $r = 1$, because state vectors are normalized in quantum mechanics), and the sphere is called Bloch sphere).

10.3.3 Parametrization of rotations

From general arguments of linear independence, it is clear that a general rotation in 3D requires 3 independent parameters, which is known as Euler's rotation theorem. It is tempting to think rotations as three-dimensional vectors, but they are rather linear operators that transform vectors.

There are different ways of parametrizing rotations in the 3D Euclidean space using 3 parameters, and we will go through the two commonly used in game programming.

Axis-angle parametrization: (ϕ, θ, φ)

As we found out, this is the *de facto* parametrization of rotations in 3D (or in fact, in any dimensions), which is also the direct generalization of rotations in 2D, is this: choose an axis in 3D space (a unit vector to specify a direction, which has two independent parameters, because its length is conventionally fixed to 1) and is typically parametrized using polar and azimuthal angles as $\boldsymbol{n}(\phi, \theta) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ and specify the angle of rotation around that axis φ (which is the third parameter) whose sense of rotation is fixed by the right-hand rule (for a right-hand coordinate system). For (embedded) 2D rotations in the xy-plane, the rotation axis is taken to be the z-axis, and we only need to specify the rotation angle. In 3D, the rotation axis can be pointing toward any direction (since the axis is a unit vector, you can think of it as a point on the unit sphere, if you like). This way of parametrizing rotations is called axis-angle parametrization, and is the easiest to picture intuitively.

Another advantage of the axis angle parametrization is, it is easy to interpolate between two different rotations (let's call their parameters $\boldsymbol{n}_1, \varphi_1$ and $\boldsymbol{n}_2, \varphi_2$), which is useful when you're changing the orientation of an object, starting from a given initial orientation and going toward a final one. A nice way of doing this is described in a later section where we describe SLERP. It results in a smooth motion, rather than a "jerky" motion which may start fast, go slow in the middle and go fast again etc. It turns out that if a mass is transported this way, it experiences the least amount of torque (compared to other possible trajectories). This way of linearly interpolating rotations in the axis-angle parametrization is called Spherical Linear Interpolation or SLERP, and is almost ubiquitously used in games.

Euler angles (and Tait-Bryan angles): $(\varphi_1, \varphi_2, \varphi_3)$

Another way of parametrizing 3D rotations is by considering a sequence of at least 3 rotations around certain, fixed axes. This could be, for example "rotate around the Z-axis by φ_1, then rotate around the Y-axis by φ_2, and finally rotate around the x-axis by φ_3". Of course, these axes don't have to be Z, then Y, then X. As long as two sequential rotations are not around the same axis (in which case they would combine into a single rotation, hence reducing the number of actually independent parameters by 1), they can be anything. They don't even need to be aligned with any "named" axis. Another thing important think to watch out is, if you imagine that there is a local coordinate system "painted" on the object, as you go through the 3-step rotation sequence, that local frame rotates with the object itself, while the global frame of course remains as-is. The rotation angles specified with respect to a global, fixed reference frame are sometimes called Tait-Bryan angles. So when we're specifying a general rotation in terms of 3 rotations around certain "fixed" axes, you need to specify whether those axes refer to the global (called extrinsic, usually written with an additional prime, X' rather than X, after each rotation) or the local (called intrinsic) frame as well. Typically, extrinsic is used as it is relatively easier to picture, and axes are chosen to coincide with X,Y or Z. The 3 rotation angles used in such representation of rotations is called Euler angles.

Ancient mechanical devices called gimbal (which are long obsolete in this age) used to calculate realize 3D rotations in engineering applications in vehicles increased the popularity of Euler angles. Furthermore, since the 3×3 rotation matrix around X,Y or Z axis is easier to write down, it is commonly said that Euler angles are easier to understand when compared to axis-angle representation (which is commonly, although not necessarily, implemented through

quaternions). While this may be true if you're creating a linear algebra library from scratch, or filling in the elements of the transformation matrix on your own, this is not the case when writing games with game engines, such as Godot. The popularity of Euler angles endured despite the fact that they, in fact, can not represent a smooth transition between two different rotations by a smooth variation of the three angles. The reason behind this lies in the fact that Euler angles describe a chart on 3-torus, which is not a covering map of SO(3), three dimensional rotations (if this sounds too abstract, see the subsection about 3-torus and sphere below). As the mapping from Euler angles to SO(3) is ill-defined at certain points, during the smooth interpolation between two rotations, we may bump into such points, and as a result the rank may drop to 2 or even 1 (which mechanically corresponds to a situation in which two or three gimbals become aligned as they go around), which is known as the gimbal-lock problem. Thus, while it's OK to use Euler angles to represent a fixed rotation, they're not suitable for slowly changing the orientation of objects. You can still do that if you put some additional effort to avoid gimbal-lock, of course, but even if by some luck you don't bump into such bad points, a linear interpolation between two sets of Euler angles is going to result in a "jerky" motion.

All in all, despite being an ill-defined parametrization for rotations, Euler angles enjoyed a popularity due to gimbals.

While Euler angles may not be too useful when calculating the rotation operator (be it a matrix or a quaternion) in body dynamics, people still find them useful to think about and describe the *orientation* of 3D objects starting from the initial default *"unrotated"* state (it's difficult to calculate the 3 Euler angles for driving an object from a non-trivial initial orientation to a non-trivial final orientation —it can't be calculated intuitively in general, it is a task for computers). For this reason, Godot's property editor uses Euler angles (in extrinsic YXZ convention; if the node has a parent node, the YXZ axes refer to the local axes of the parent) for the rotation property of a Transform –this is pretty much the only place Euler angles are used in Godot.

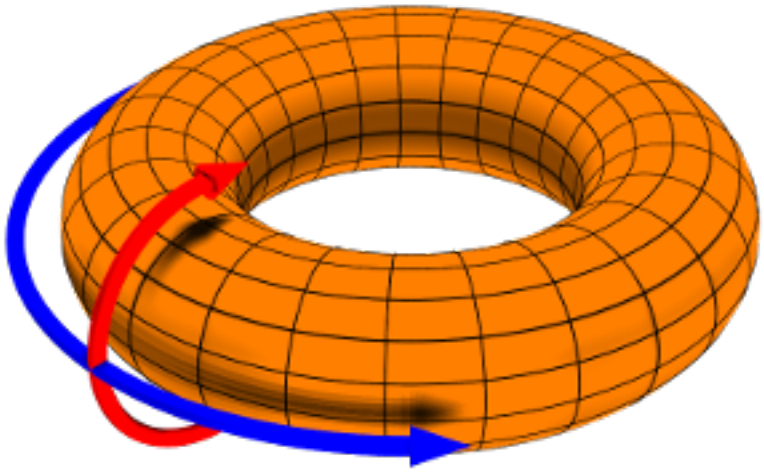
So the answer to the question "should I use Euler angles or axis-angle parametrization in my game" is: unless you're trying to *statically* orient an object in a particular way starting from an *unrotated, default orientation* (for which you can still use axis-angle parametrization in your script, if you prefer), you shouldn't be using Euler angles. Major reasons are:

1. Jerky interpolations. You can't interpolate two orientations smoothly (torque-free) in a way that is reference independent (which doesn't depend on how you choose the 3 fixed rotation axes).
2. Gimbal lock. Rotation dynamics (which includes interpolations) can get worse than jerky, you can get stuck in a weird coordinate singularity (the kind which doesn't exist in axis-angle parametrization) and never reach your target.

A note about surface of 3-torus and sphere, and Gimbal lock

What is a 3-torus, to begin with? The surface of a donut is a 2-torus, referring to the fact that the surface itself is two-dimensional (although curved), which is easy to visualize. However, it's difficult to visualize a torus in higher dimensions. But as it turns out, there is another way of thinking about torus, which generalizes to higher dimensions.

Imagine an ant walking on the surface of a donut illustrated below (image borrowed from here).



If it keeps walking along either the red or the blue lines, it will eventually come back to where it started. At any time, we can use two angles to describe where the ant is: one angle (θ which goes between 0 and 2π) describing it's position along the red line, and another one (ϕ, again between 0 and 2π) for the blue line. And note that we have periodicity: (θ, ϕ) and $(\theta + 2\pi N, \phi + 2\pi N)$ describe exactly the same point on the donut for an integer N. We have two angles, of course, because we're talking about a two-dimensional surface. Now we're ready to talk about n-dimensional surfaces.

If you have a set of n "coordinates" (or angles) which are periodic, just like θ and ϕ were (which is the case for the three Euler angles), then people say those coordinates describe a point on the surface of an n-torus. (In the case that the period of a "coordinate" is different than 2π, that coordinate can be scaled to make it so, such that it corresponds to an n-torus).

Now, back to our original issue. The axis of the axis-angle parametrization (which is *the* natural way of parametrizing rotations, and is a one-to-one covering map of SO(3); in fact, this is true for all Lie groups, not just rotations in 3D) spans a sphere (every point in a ball of radius π corresponds to a rotation in SO(3) where the rotation amount is mapped to radius and rotation axis points is the direction from the origin; with the additional caveat that if you flip the sign of the axis and angle simultaneously, it maps to the same rotation), which is described using spherical coordinates, whereas Euler angles span the surface of a 3-torus (such that every point on the 3-torus corresponds to a rotation), which is described by the three Euler angles. The problem here is, a sphere is topologically different from a torus. In simple terms, this means that it's impossible to deform a sphere to a torus "smoothly": at some point, you have to punch a hole in it to make a donut from a ball (and you can never "punch a hole" or change the topology when you stick with a parametrization: if you use Euler angles, you're forever stuck walking the surface of a 3-donut, and if you use axis-angle, you're forever stuck flying inside the sphere).

Why is this a problem at all? Because it means that a smooth walk (flight?) inside the sphere doesn't always correspond to a smooth walk on the surface of the 3-torus, vice versa: a 3-torus and sphere are globally different, which is a fact you're bound to discover if you walk the parameter space by a smoothly rotating a body using these parametrizations. Axis-angle parametrization has singularities at the poles (where the azimuthal angle is ill-defined) but that's fine because that's exactly how SO(3) is, after all, axis-angle is how Lie groups are parametrized. Euler angle parametrization also has singularities corresponding to points where two gimbals are aligned, but those singularities are different from SO(3)'s poles.

Suppose that you're at a nice spot, a rotation that can be parametrized by both axis-angle and Euler angles uniquely. Sometimes, it can just happen that if you take a wrong step, you'll fall into a "hole" (meaning a singularity at which infinitely many parameters correspond to the same rotation, like at the poles, all choices for the azimuthal angle give the same rotation, or the similar situation with gimbal lock) in one parametrization, while you can continue a smooth journey in the other parametrization. When you fall a "hole" using Euler angles, it's called gimbal lock, and since there may not be a corresponding "hole" when you take the similar step in the sphere, this tells us that Euler angles is

a defective parametrization of SO(3).

The root of the problem isn't just the fact that Euler angle parametrization has singularties, just as axis-angle does which is fine on its own, but that those singularities don't match with SO(3)'s singularities.

This is the mathematical description of the gimbal-lock problem.

Here's an example of gimbal lock in Euler angle parametrization. Suppose that one of the rotations is $\pi/2$, let's say the middle one. By inserting an identity operator $X(-\pi/2)X(\pi/2)$ to the right side and rearranging terms, we can show that

$$R = Y(\varphi_y)X(\pi/2)Z(\varphi_z) = Y(\varphi_y)[X(\pi/2)Z(\varphi_z)X(-\pi/2)]X(\pi/2) = Y(\varphi_y)Y(-\varphi_z)X(\pi/2) = Y(\varphi_y - \varphi_z)X(\pi/2),$$

(see the section about active transformation below about how a rotation matrix itself transforms, which also explains why and how a Z rotation turns into a Y rotation when surrounded by $\pi/2$ and $-\pi/2$ X rotations) which means we lost a degree of freedom: $\varphi_y - \varphi_z$ effectively became a single parameter determining the Y rotation and we completely lost the first Z rotation. You can follow similar steps to show that when *any* of the YXZ Euler angles become $\pm\pi/2$, you get a gimbal lock.

This happens for $\pm\pi/2$ simply because in the YXZ convention, neighboring axes are related to each other by a $\pm\pi/2$ rotation around the axis given by the other neighbor. For XZX convention, the gimbal lock would happen at $\varphi_z = \pm\pi$ for example.

Summary: representation versus parametrization

We can sum it up in a table:

Parametrization/Representation	Matrix representation of SO(3)	Quaternion representation of SU(2)
Axis-angle	$e^{\varphi \boldsymbol{v}\cdot\boldsymbol{J}}$	$e^{\varphi \boldsymbol{v}\cdot(i,j,k)/2}$
Euler-angles	$e^{\varphi_3 J_y} e^{\varphi_2 J_x} e^{\varphi_1 J_z}$	$e^{\varphi_3 j/2} e^{\varphi_2 i/2} e^{\varphi_1 k/2}$

where $\boldsymbol{J}$ denotes the matrix representation of the $\boldsymbol{J}$ operators (too lazy to introduce a new symbol for that). YXZ Euler angles is chosen for concreteness (as it is the default convention in Godot), and can be replaced by any three axes (as long as two neighboring axes aren't the same).

In all cases listed in the above table, Rodrigues' formula (or it's analogue for quaternions) given above can be used for practical purposes.

In the context of 3D rotations, one representation isn't superior or inferior to another. Whatever representation you're using, you are representing exactly the same thing. A difference appears only when you implement it on a computer: different representations have trade offs when it comes to precision errors, CPU cycles and memory usage (for example, accessing to rotation axis-angle with quaternions is trivial but requires some algebra in matrix representation whereas the converse is true when accessing the basis vectors, SLERP, composition of rotations and orthonormalization is faster with quaternions but a conversion to matrix representation, which isn't free, is always required because that's the representation OpenGL uses and rotating a 3D vector is faster in matrix representation since they are written in the same basis), and they may have different characteristics when doing finite precision arithmetic with floating point numbers. In principle, you can do everything you do with quaternions using matrices, vice versa. The performance could be bad in one representation, but the point is, there is nothing in their mathematics that prevent you from doing that.

Parametrizations, on the other hand, are vastly different. Axis-angle is the "one true" parametrization of rotations. Euler angles, despite being a defective parametrization of rotations, could be more intuitive for simple (involving only 1 or 2 angles) and *static* situations like orienting a body/vehicle in the editor, but should be avoided for rotational *dynamics* which would eventually lead to a gimbal lock.

10.3.4 Interpolating rotations

In games, it's common problem to interpolate between two different orientation in a "nice way" which doesn't depend on arbitrary things like how the reference frame, and which doesn't result in a "jerky" motion (that is, a torque-free motion) such that the angular speed remains constant during the interpolation. These are the properties that we seek when we say "nice".

Formally, we'd like to interpolate between an initial rotation $R_1 = e^{\lambda\varphi_1 \boldsymbol{n}_1\cdot\boldsymbol{J}}$ and a final one $R_2 = e^{\varphi_2\boldsymbol{n}\cdot\boldsymbol{J}}$ a function of time, and what we're seeking is something that transforms one into another smoothly, $R(\lambda)$, where λ is the normalized time which is 0 at the beginning and 1 at the end. Clearly, we must have $R(\lambda = 0) = R_1$ and $R(\lambda = 1) = R_2$. Since we also know that rotations form a group, we can relate $R(\lambda)$ to R_1 and R_2 using another rotation, such that we can for example write

$$R(\lambda) = e^{\lambda\varphi\boldsymbol{n}\cdot\boldsymbol{J}}R_1.$$

This form makes sense because for $\lambda = 0$, the interpolation hasn't started and $R(\lambda)$ automatically becomes R_1. But why not pick a different form for the exponent as a function of λ which evaluates to 0 when $\lambda = 0$? That's simply because we don't want to have a jerky motion, meaning $\boldsymbol{\omega}\cdot\boldsymbol{J} = R^T(\lambda)\dot{R}(\lambda)$, where $\boldsymbol{\omega}$ is the angular velocity vector, has to be a constant, which can only happen if the time derivative of the exponent is linear in time (in which case we obtain $\boldsymbol{\omega} = \varphi\boldsymbol{n}$). Or equivalently, you can simply observe that a rotation around a fixed axis (fixed because otherwise if you tilt the rotation axis in time, you'll again get a "jerky motion" due to Euler force) with constant angular speed is $e^{\boldsymbol{\omega}t\cdot\boldsymbol{J}}$ where the exponent is linear in time.

But how do we choose $\boldsymbol{n}$ and φ? Well, we simply enforce that $R(\lambda)$ has to becomes R_2 at the end, when $\lambda = 1$. Although this looks like a difficult problem, it's actually not. We first make a rearrangement:

$$R(\lambda) = e^{\lambda\varphi\boldsymbol{n}\cdot\boldsymbol{J}}R_1 = (e^{\varphi\varphi\boldsymbol{n}\cdot\boldsymbol{J}})^\lambda R_1.$$

From this expression, it becomes evident that the solution is $e^{\varphi\boldsymbol{n}\cdot\boldsymbol{J}} = R_2R_1^T$.

We can also do the same thing in SU(2) and obtain:

$$U(\lambda) = (U_2U_1^\dagger)^\lambda U_1,$$

or isomorphically, in terms of quaternions:

$$q(\lambda) = (q_2q_1^{-1})^\lambda q_1.$$

For any Lie group, including SO(3) and SU(2), this "nice" interpolation is called SLERP.

Note that at no step we made any reference to axes of some fixed reference frame (as it is in the case of Euler angle parametrization, which are defined with respect to certain "special" 3 axes), everything we did is independent of the reference frame. Also note that this scheme can work for any Lie group, and is independent of the representation used (matrix, quaternion, ...).

After some tedious algebra (which involves using $\text{tr}(\sigma_i\sigma_j) = 2\delta_{ij}$), this result can be simplified to

$$q(\lambda) = \frac{\sin([1-\lambda]\Omega)q_1 + \sin(\Omega\lambda)q_2}{\sin\Omega}$$

when $q_1 \neq \pm q_2$, where we introduced $\cos\Omega \equiv \cos\frac{\varphi_1}{2}\cos\frac{\varphi_2}{2} + \sin\frac{\varphi_1}{2}\sin\frac{\varphi_2}{2}\boldsymbol{n}_1\cdot\boldsymbol{n}_2$ (or equivalently, in terms of an artificial vector which contains the coefficients of the quaternions, as we discussed above, $\boldsymbol{q}_1\cdot\boldsymbol{q}_2$). This result has a simple geometric interpretation when a (unit) quaternion is taken to be a point on the 3-sphere and when one introduces an inner product of the 4D Euclidean space, but we'll skip that because it's a bit off topic in our treatment. We'll however note that this is where the name *Spherical* Linear intERpolation comes from, which refers to the 4D sphere.

10.3.5 Reference frames: global, parent-local, object-local

A reference frame essentially how and where how place the origin and axes of your coordinate system. In Godot, there are three different reference frames. The first is the global reference frame. It exist as the "anchor" frame, where all other frames can be defined with respect to. The other types of reference frames appear when you have objects which have children objects. Here's an example which illustrates these frames.

Consider a ship in the ocean. You can imagine, however that there's a coordinate system painted on the ground somewhere in the ship. This coordinate system moves and rotates with the ship, so it's called ship's local frame. Furthermore, we can attach a reference frame to passengers on the ship (for example, let's say, for each passenger, their left is their x-axis and their forward is their z axis, and their origin is where they stand).

The global coordinate system corresponds to a coordinate system world painted somewhere on the ground in the world (let's say we live in a flat world for simplicity). Then every object, including the ship and the passengers have their own reference frames, they're called object-local frames. But notice that for passengers, the most natural frame would be where they are located (and how they're oriented) relative to the ship. After all, when someone asks "Where's Joe?", you'd reply with something like "I saw him in the front deck" rather than trying to look up GPS coordinates (global frame) or saying something like "7 meters to my five o'clock" (passenger/object-local). The ship is referred to as the parent-local frame.

In Godot, Basis matrices refer to the parent-local frame. If the object is top level, then it's Basis is written in the global frame.

10.3.6 Active and passive transformations

While operators (such as $e^{\varphi \boldsymbol{n}\cdot\boldsymbol{J}}$) and vectors $\boldsymbol{v}$ are "out there" and are independent of the reference frame, their coordinates aren't. For example, a vector can be aligned with the x-axis in some frame whereas in can be aligned with y-axis in another, if you choose to draw your coordinate system differently. But the vector doesn't know about your arbitrary choice of how and where you draw your coordinate system. When we have multiple reference frames, it's important how the coordinates would transform between them.

For example, if you know that the coordinates of a vector $\boldsymbol{v}$ are given as v_x, v_y, v_z in some reference frame, you can obtain the coordinates in a different frame which is rotated which respect to the first one around some $\boldsymbol{n}$ axis by $-\varphi$ as

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix} = R_n(\varphi) \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix},$$

where primed coordinates correspond to coordinates in the rotated *frame*. You can also transform the matrix representations of operators. For example, $M_{ij} = \boldsymbol{e}_i \cdot M\boldsymbol{e}_j$ but what is the rotation matrix if we used the basis vectors of a different frame $\{\boldsymbol{e}'_i\}$? To obtain the matrix representation of M in the new frame, you can do

$$M' = R_n(\varphi) M R_n^T(\varphi).$$

These are called passive transformations.

Now let's consider actually moving those objects (we consider only one reference frame and it's fixed). We rotate a vector around some $\boldsymbol{n}$ axis by φ

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix} = R_n(\varphi) \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix},$$

where primed coordinates are the coordinates of the rotated *vector*, in the same reference frame. Similarly, for matrices

$$M' = R_n(\varphi) M R_n^T(\varphi).$$

These are called active transformations.

Note how things fit nicely, for example, when you consider how a rotated vector $R_0 \boldsymbol{v}_0$ transforms:

$$R_n(\varphi)[R_0 \boldsymbol{v}_0] = [R_n(\varphi)(R_0)R_n(\varphi)^T][R_n(\varphi)\boldsymbol{v}_0]$$

The left hand side is rotation of the vector $(R_0 \boldsymbol{v}_0)$ and the right hand side is the rotation of the vector v_0 and the matrix R_0 that acts on it, which of course agree.

The rotation operator itself rotates in a expected way (you can use Rodrigues' formula along with the equation above, if you prefer):

$$R_n(\varphi)(e^{\varphi_0 \boldsymbol{n}_0 \cdot \boldsymbol{J}})R_n(\varphi)^T = e^{\varphi_0 [R_n(\varphi)\boldsymbol{n}_0] \cdot \boldsymbol{J}}$$

For example, if we have a rotation around the z-axis by $\varphi_0 = \varphi_z$ and we would like to rotate it around the x-axis by $\varphi = \pi/2$, we'd have

$$R_x(\pi/2)(e^{\varphi_z \boldsymbol{n}_z \cdot \boldsymbol{J}})R_x(\pi/2)^T = e^{\varphi_z [R_x(\pi/2)\boldsymbol{e}_z] \cdot \boldsymbol{J}} = e^{\varphi_z [-\boldsymbol{e}_y] \cdot \boldsymbol{J}},$$

that is, the original rotation axis $\boldsymbol{n}_0 = \boldsymbol{e}_z$ gets rotated around the x-axis by $\varphi = \pi/2$ and becomes a rotation around the y-axis by $-\varphi_z$.

10.4 Matrices and transforms

10.4.1 Introduction

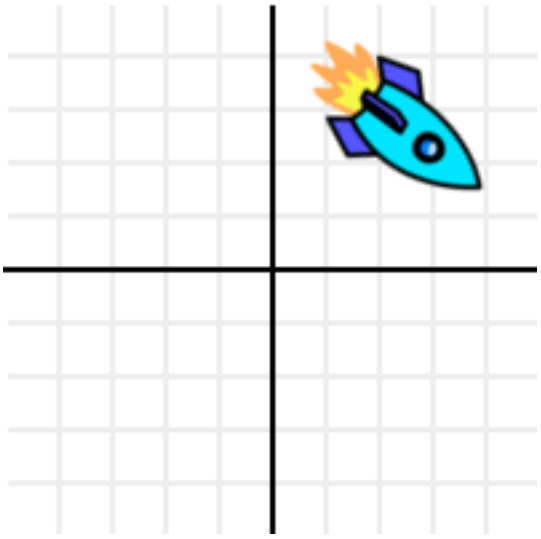
Before reading this tutorial, it is advised to read the previous one about *Vector math* as this one is a direct continuation.

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.

10.4.2 Oriented coordinate system (OCS)

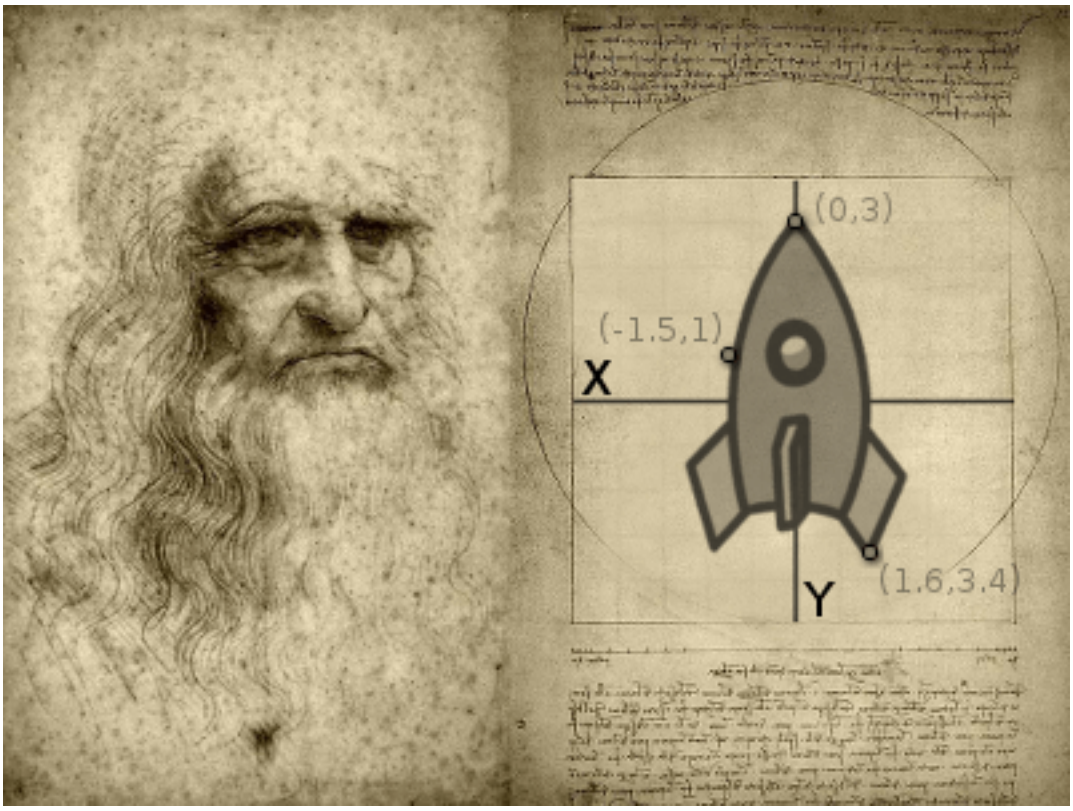
Imagine we have a spaceship somewhere in space. In Godot this is easy, just move the ship somewhere and rotate it:



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 even that useful when working in 3D).

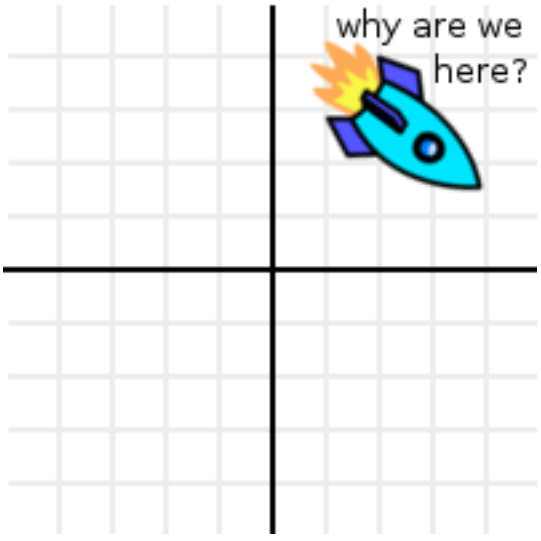
We should realize that at some point, someone *designed* this spaceship. Be it for 2D in a drawing such as Paint.net, Gimp, Photoshop, etc. or in 3D through a 3D DCC tool such as Blender, Max, Maya, etc.

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).

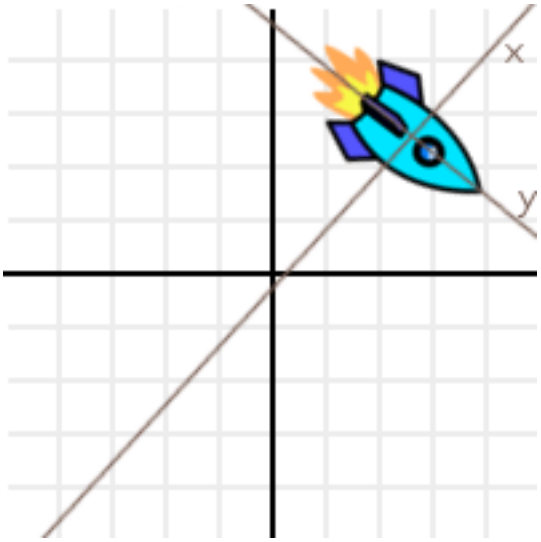
So, let's recall again that the ship was somewhere in space:



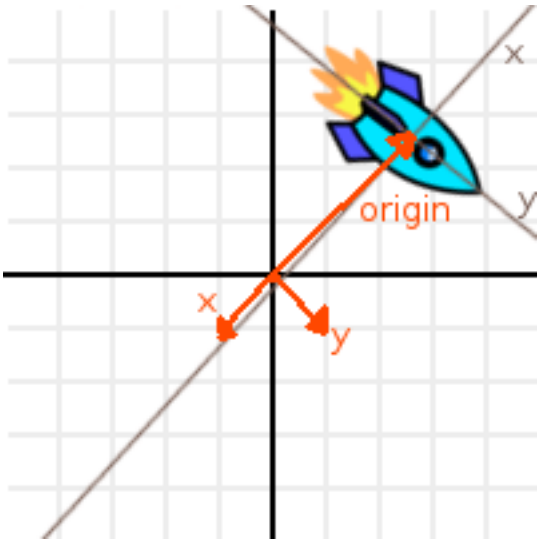
How did it get there? What moved it and rotated it from the place it was designed to its current position? The answer

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.

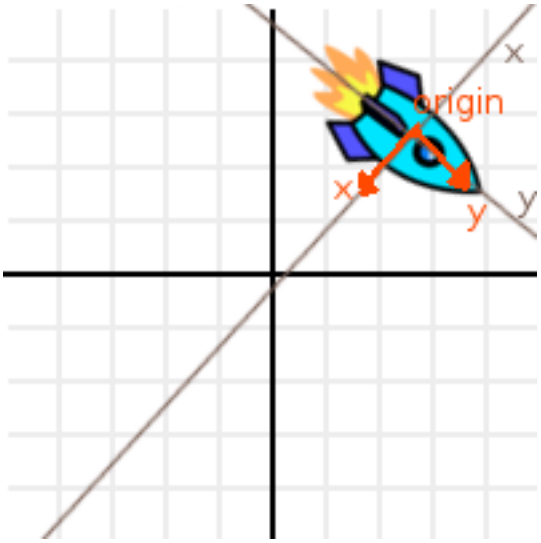
But transform is too generic of a term to describe this process. To solve this puzzle, we will superimpose the ship's original design position at their current position:



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.

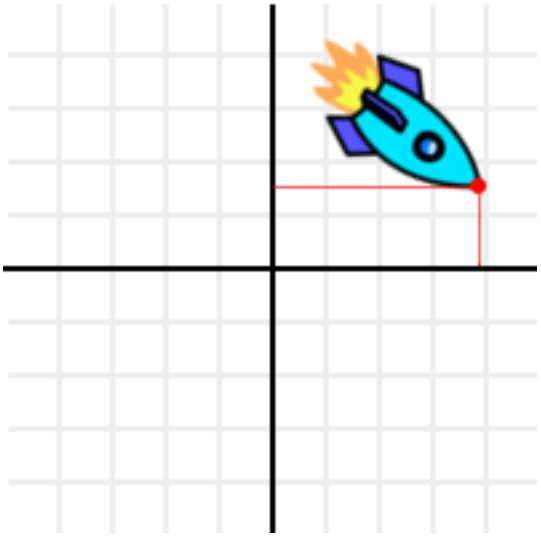


Let's call the 3 vectors "X", "Y" and "Origin", and let's also superimpose them over the ship so it makes more sense:



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):

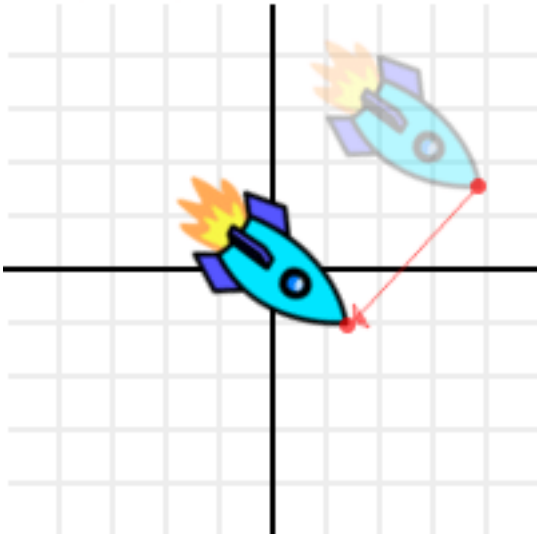
GDScript

C#

```
var new_pos = pos - origin
```

```
var newPosition = 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:

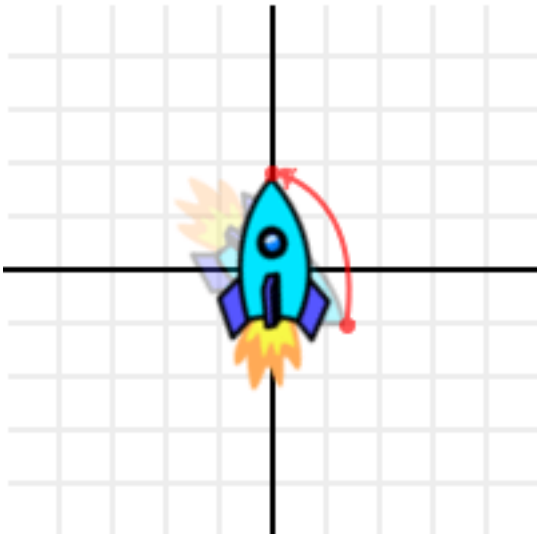
GDScript

C#

```
var final_pos = Vector2(x.dot(new_pos), y.dot(new_pos))
```

```
var finalPosition = new Vector2(x.Dot(newPosition), y.Dot(newPosition));
```

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 an Y are the **Basis**, and *Origin* is the offset.

10.4.3 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*.

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.

10.4.4 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.

10.4.5 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.

GDScript

C#

```
var m = Transform2D()
var x = m[0] # 'X'
var y = m[1] # 'Y'
var o = m[2] # 'Origin'
```

```
var m = new Transform2D();
Vector2 x = m[0]; // 'X'
Vector2 y = m[1]; // 'Y'
Vector2 o = m[2]; // 'Origin'
```

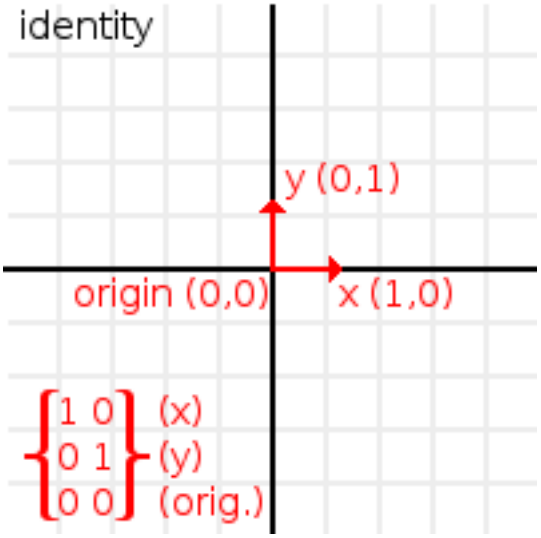
Most operations will be explained with this datatype (Transform2D), but the same logic applies to 3D.

10.4.6 Identity

An important transform is the "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.

GDScript

C#

```
# The Transform2D constructor will default to Identity
var m = Transform2D()
print(m)
# prints: ((1, 0), (0, 1), (0, 0))
```

```
// Due to technical limitations on structs in C# the default
// constructor will contain zero values for all fields.
var defaultTransform = new Transform2D();
GD.Print(defaultTransform);
// prints: ((0, 0), (0, 0), (0, 0))

// Instead we can use the Identity property.
var identityTransform = Transform2D.Identity;
GD.Print(identityTransform);
// prints: ((1, 0), (0, 1), (0, 0))
```

10.4.7 Operations

10.4.8 Rotation

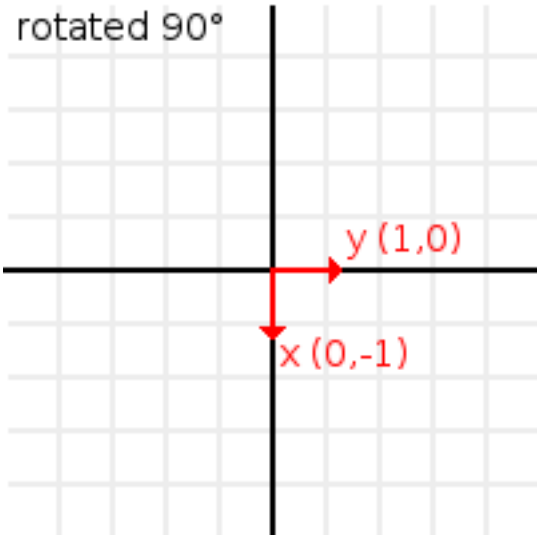
Rotating Transform2D is done by using the "rotated" function:

GDScript

C#

```
var m = Transform2D()
m = m.rotated(PI/2) # rotate 90°
```

```
var m = Transform2D.Identity;
m = m.Rotated(Mathf.PI / 2); // rotate 90°
```



10.4.9 Translation

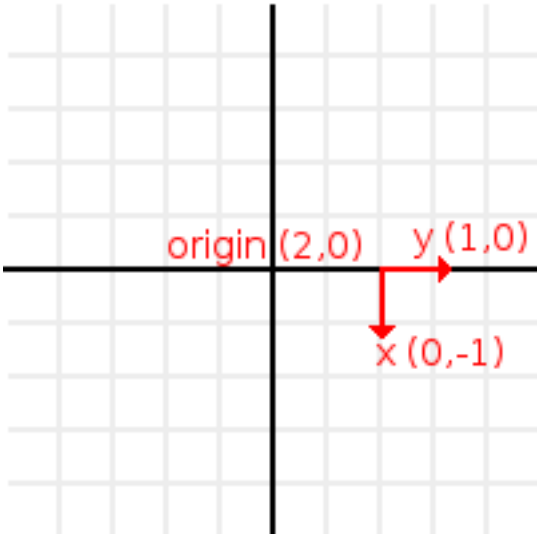
There are two ways to translate a Transform2D, the first one is moving the origin:

GDScript

C#

```
# Move 2 units to the right
var m = Transform2D()
m = m.rotated(PI/2) # rotate 90°
m[2] += Vector2(2,0)
```

```
// Move 2 units to the right
var m = Transform2D.Identity;
m = m.Rotated(Mathf.PI / 2); // rotate 90°
m[2] += new Vector2(2, 0);
```



This will always work in global coordinates.

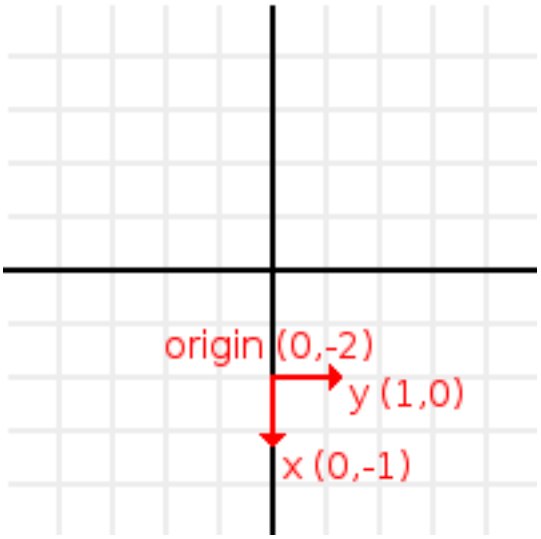
If instead, translation is desired in *local* coordinates of the matrix (towards where the *basis* is oriented), there is the *Transform2D.translated()* method:

GDScript

C#

```
# 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) )
```

```
// Move 2 units towards where the basis is oriented
var m = Transform2D.Identity;
m = m.Rotated(Mathf.PI / 2); // rotate 90°
m = m.Translated(new Vector2(2, 0));
```



You could also transform the global coordinates to local coordinates manually:

GDScript

C#

```
var local_pos = m.xform_inv(point)
```

```
var localPosition = m.XformInv(point);
```

But even better, there are helper functions for this as you can read in the next sections.

10.4.10 Local to global coordinates and vice versa

There are helper methods for converting between local and global coordinates.

There are *Node2D.to_local()* and *Node2D.to_global()* for 2D as well as *Spatial.to_local()* and *Spatial.to_global()* for 3D.

10.4.11 Scale

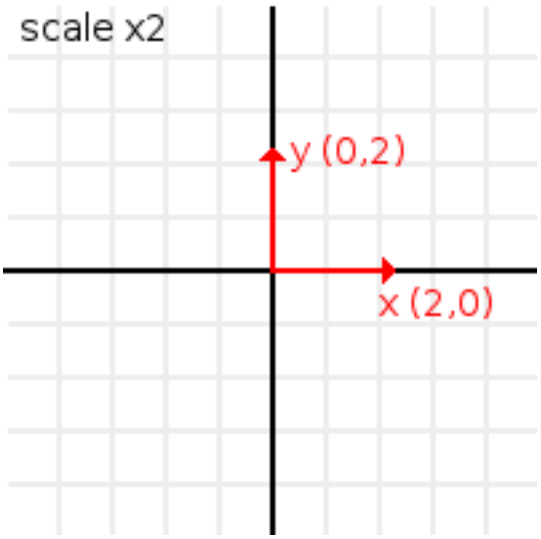
A matrix can be scaled too. Scaling will multiply the basis vectors by a vector (X vector by x component of the scale, Y vector by y component of the scale). It will leave the origin alone:

GDScript

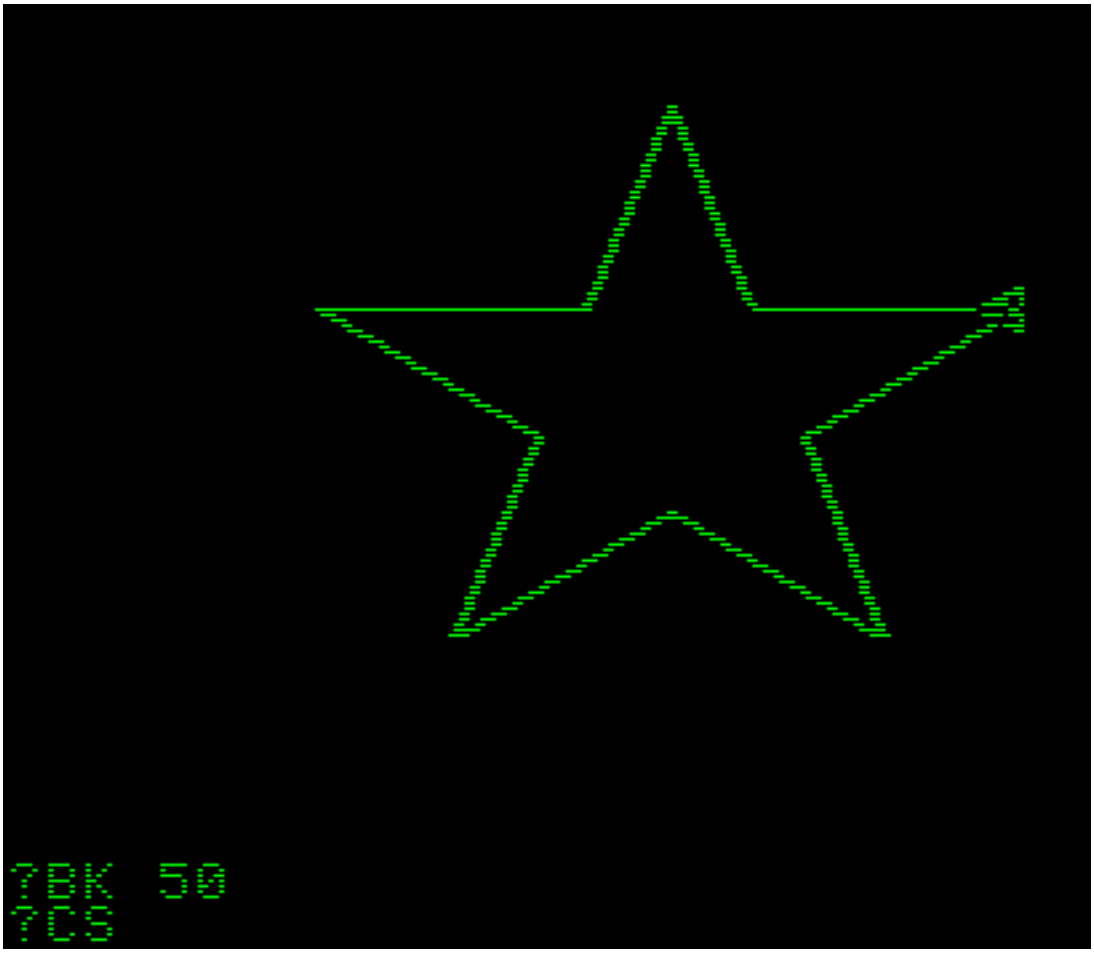
C#

```
# Make the basis twice its size.
var m = Transform2D()
m = m.scaled( Vector2(2,2) )
```

```
// Make the basis twice its size.
var m = Transform2D.Identity;
m = m.Scaled(new 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:



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).

10.4.12 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.

GDScript

C#

```
var new_pos = m.xform(pos)
```

```
var newPosition = m.Xform(position);
```

And only for basis (no translation):

GDScript

C#

```
var new_pos = m.basis_xform(pos)
```

```
var newPosition = m.BasisXform(position);
```

10.4.13 Inverse transform

To do the opposite operation (what we did up there with the rocket), the "xform_inv" method is used:

GDScript

C#

```
var new_pos = m.xform_inv(pos)
```

```
var newPosition = m.XformInv(position);
```

Only for Basis:

GDScript

C#

```
var new_pos = m.basis_xform_inv(pos)
```

```
var newPosition = m.BasisXformInv(position);
```

10.4.14 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:

GDScript

C#

```
# Does nothing, pos is unchanged
pos = Transform2D().xform(pos)
```

```
// Does nothing, position is unchanged
position = Transform2D.Identity.Xform(position);
```

10.4.15 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:

GDScript

C#

```
var mi = m.affine_inverse()
pos = m.xform(pos)
pos = mi.xform(pos)
# pos is unchanged
```

```
var mi = m.AffineInverse();
position = m.Xform(position);
position = mi.Xform(position);
// position is unchanged
```

If the matrix is orthonormal, then:

GDScript

C#

```
# if m is orthonormal, then
pos = mi.xform(pos)
# is the same is
pos = m.xform_inv(pos)
```

```
// if m is orthonormal, then
position = mi.Xform(position);
// is the same is
position = m.XformInv(position);
```

10.4.16 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:

GDScript

C#

```
var m = more_transforms * some_transforms
```

```
var m = moreTransforms * someTransforms;
```

To make it a little clearer, this:

GDScript

C#

```
pos = transform1.xform(pos)
pos = transform2.xform(pos)
```

```
position = transform1.Xform(position);
position = transform2.Xform(position);
```

Is the same as:

GDScript

C#

```
# note the inverse order
pos = (transform2 * transform1).xform(pos)
```

```
// note the inverse order
position = (transform2 * transform1).Xform(position);
```

However, this is not the same:

GDScript

C#

```
# yields a different results
pos = (transform1 * transform2).xform(pos)
```

```
// yields a different results
position = (transform1 * transform2).Xform(position);
```

Because in matrix math, A * B is not the same as B * A.

10.4.17 Multiplication by inverse

Multiplying a matrix by its inverse, results in identity:

GDScript

C#

```
# No matter what A is, B will be identity
var B = A.affine_inverse() * A
```

```
// No matter what A is, B will be identity
var B = A.AffineInverse() * A;
```

10.4.18 Multiplication by identity

Multiplying a matrix by identity, will result in the unchanged matrix:

GDScript

C#

```
# B will be equal to A
B = A * Transform2D()
```

```
// B will be equal to A
var B = A * Transform2D.Identity;
```

10.4.19 Matrix tips

When using a transform hierarchy, remember that matrix multiplication is reversed! To obtain the global transform for a hierarchy, do:

GDScript

C#

```
var global_xform = parent_matrix * child_matrix
```

```
var globalTransform = parentMatrix * childMatrix;
```

For 3 levels:

GDScript

C#

```
var global_xform = gradparent_matrix * parent_matrix * child_matrix
```

```
var globalTransform = grandparentMatrix * parentMatrix * childMatrix;
```

To make a matrix relative to the parent, use the affine inverse (or regular inverse for orthonormal matrices).

GDScript

C#

```
# transform B from a global matrix to one local to A
var B_local_to_A = A.affine_inverse() * B
```

```
// transform B from a global matrix to one local to A
var bLocalToA = A.AffineInverse() * B;
```

Revert it just like the example above:

GDScript

C#

```
# transform back local B to global B
B = A * B_local_to_A
```

```
// transform back local B to global B
B = A * bLocalToA;
```

OK, hopefully this should be enough! Let's complete the tutorial by moving to 3D matrices.

10.4.20 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.

10.4.21 Basis

Godot has a special type for a 3x3 matrix, named *Basis*. It can be used to represent a 3D rotation and scale. Sub vectors can be accessed as:

GDScript

C#

```
var m = Basis()
var x = m[0] # Vector3
var y = m[1] # Vector3
var z = m[2] # Vector3
```

```
var m = new Basis();
Vector3 x = m[0];
Vector3 y = m[1];
Vector3 z = m[2];
```

Or, alternatively as:

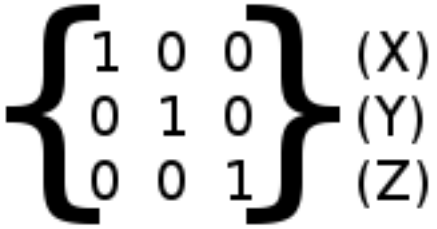
GDScript

C#

```
var m = Basis()
var x = m.x # Vector3
var y = m.y # Vector3
var z = m.z # Vector3
```

```
var m = new Basis();
Vector3 x = m.x;
Vector3 y = m.y;
Vector3 z = m.z;
```

The Identity Basis has the following values:



And can be accessed like this:

GDScript

C#

```
# The Basis constructor will default to Identity
var m = Basis()
print(m)
# prints: ((1, 0, 0), (0, 1, 0), (0, 0, 1))
```

```
// Due to technical limitations on structs in C# the default
// constructor will contain zero values for all fields.
var defaultBasis = new Basis();
GD.Print(defaultBasis);
// prints: ((0, 0, 0), (0, 0, 0), (0, 0, 0))

// Instead we can use the Identity property.
var identityBasis = Basis.Identity;
```

(continues on next page)

(continued from previous page)

```
GD.Print(identityBasis);;
// prints: ((1, 0, 0), (0, 1, 0), (0, 0, 1))
```

10.4.22 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).

GDScript

C#

```
#rotate in Y axis
var m3 = Basis()
m3 = m3.rotated( Vector3(0,1,0), PI/2 )
```

```
// rotate in Y axis
var m3 = Basis.Identity;
m3 = m3.Rotated(new Vector3(0, 1, 0), Mathf.PI / 2);
```

10.4.23 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:

GDScript

C#

```
var t = Transform()
pos = t.xform(pos) # transform 3D position
pos = t.basis.xform(pos) # (only rotate)
pos = t.origin + pos # (only translate)
```

```
var t = new Transform(Basis.Identity, Vector3.Zero);
position = t.Xform(position); // transform 3D position
position = t.basis.Xform(position); // (only rotate)
position = t.origin + position; // (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

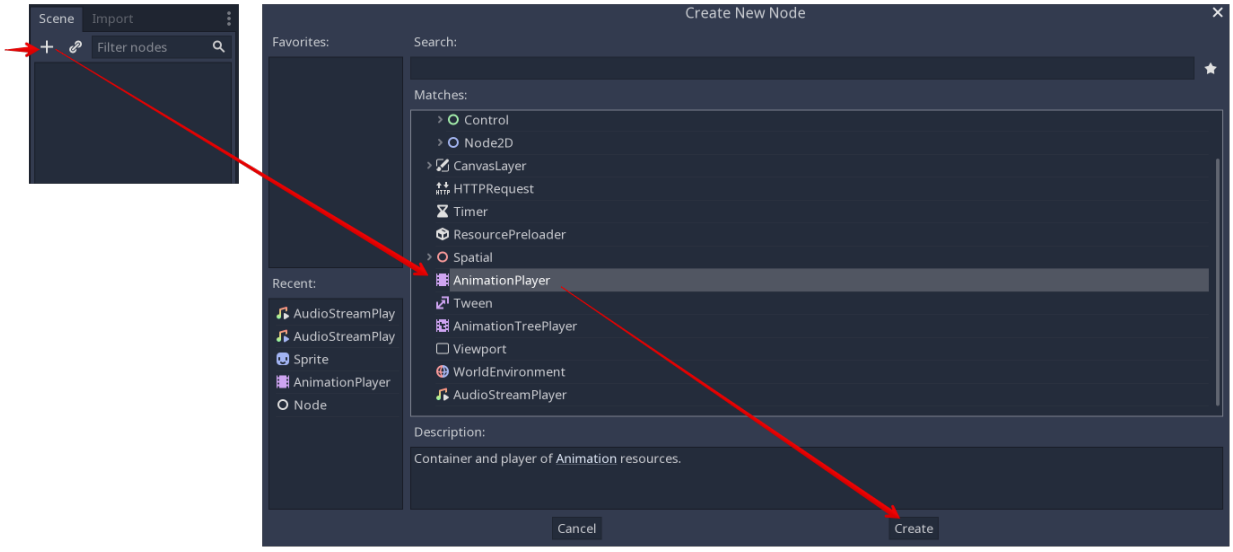


Fig. 1: The AnimationPlayer node

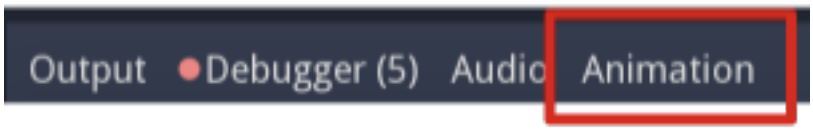


Fig. 2: The animation panel position

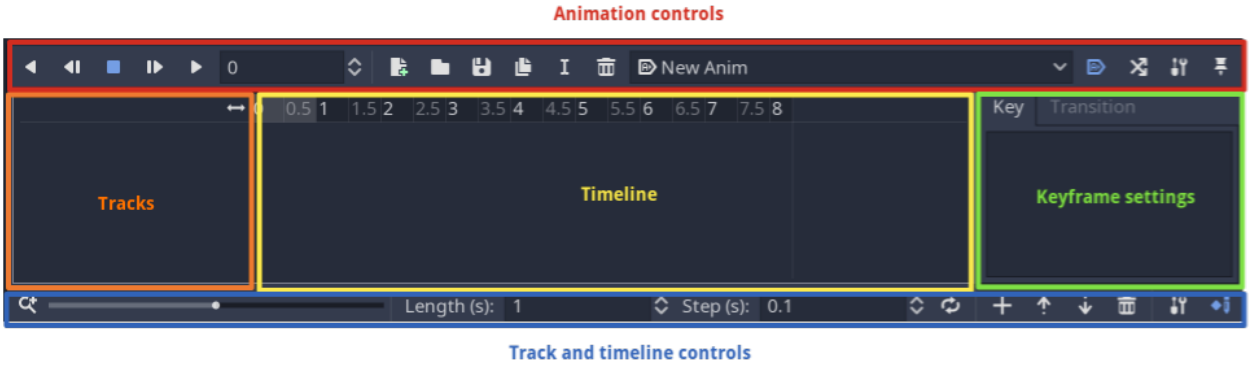


Fig. 3: The animation panel

- 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.

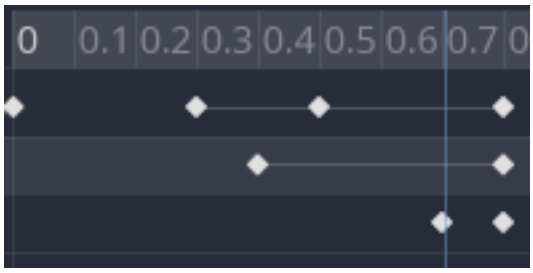


Fig. 4: Keyframes in Godot

The engine interpolates values between keyframes, resulting in a gradual change in values over time.

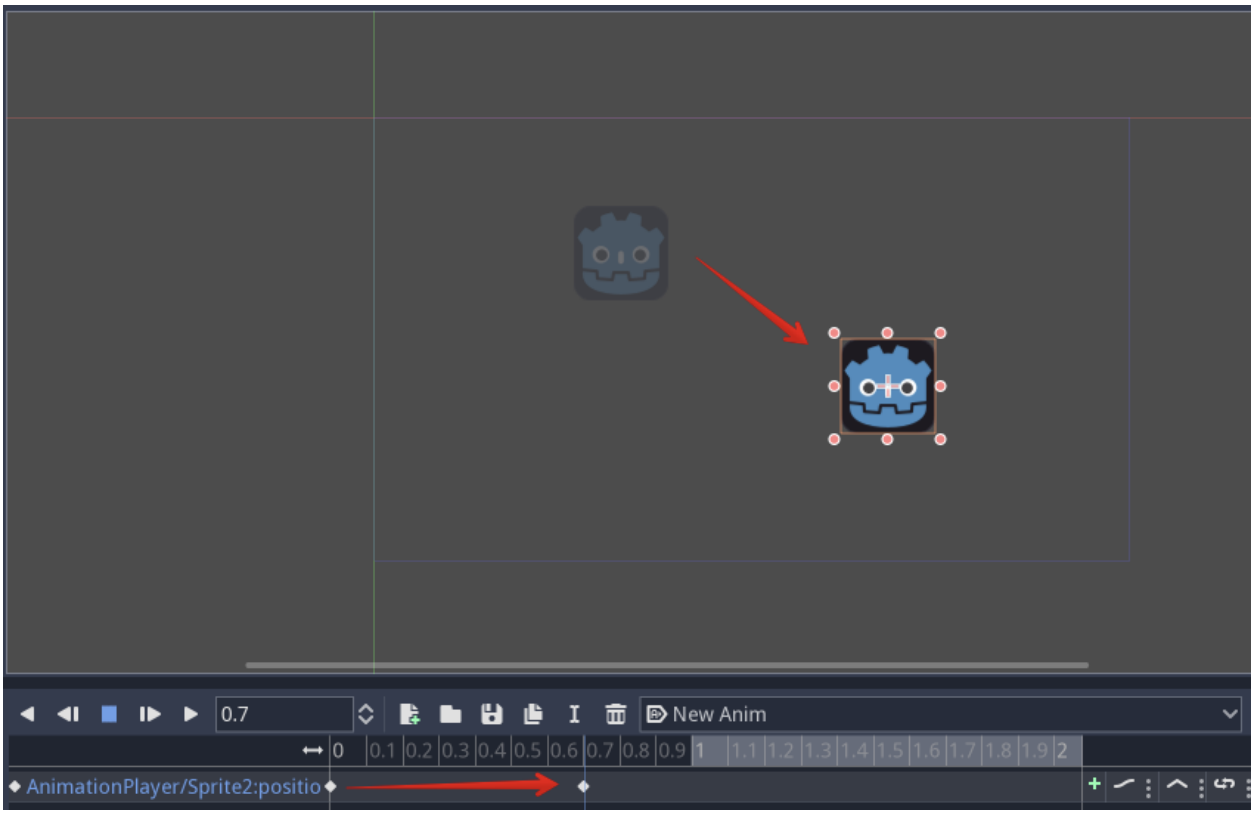


Fig. 5: Two keyframes are all it takes to obtain a smooth motion

The timeline lets you insert keyframes and change their timing. It also defines how long the animation is.

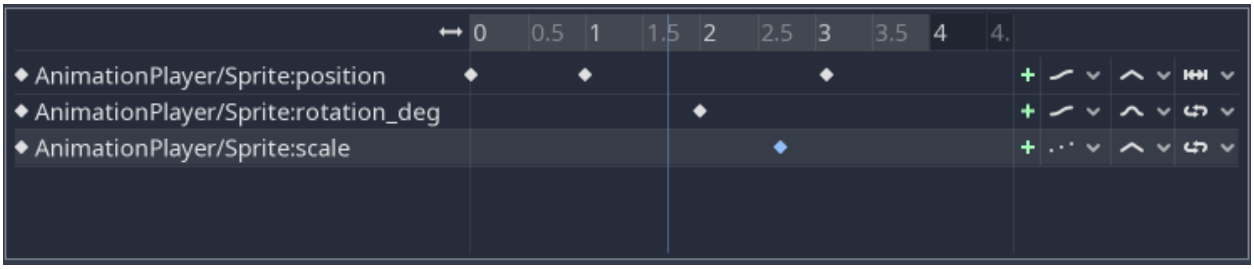


Fig. 6: The timeline in the animation panel

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.

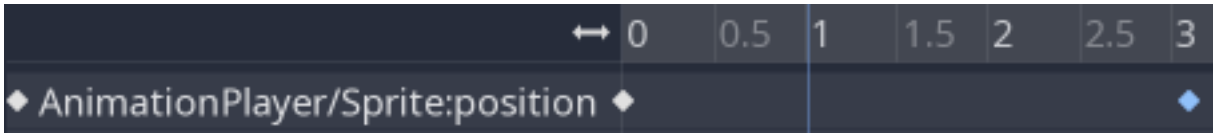


Fig. 7: Example of Normal animation tracks

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.

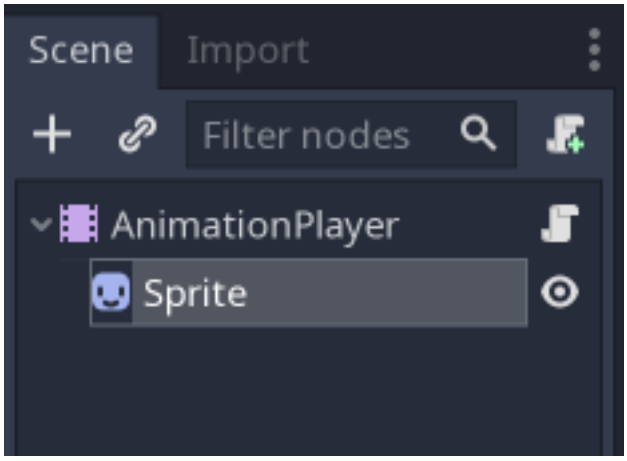


Fig. 8: Our scene setup

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.

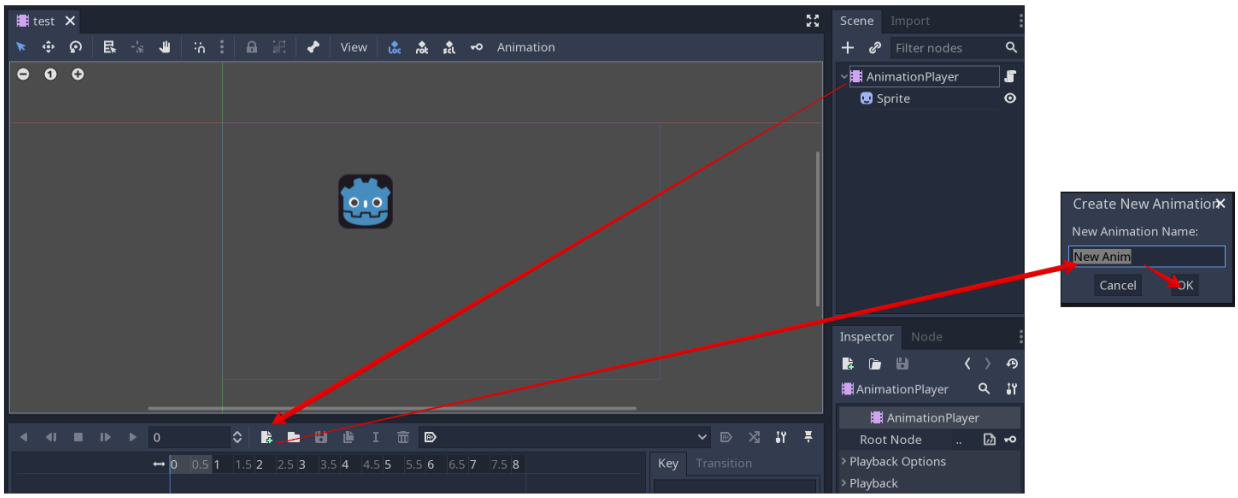


Fig. 9: Add a new animation

Adding a track

To add a new track for our sprite, select it and take a look in the toolbar:

Fig. 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 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: 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.

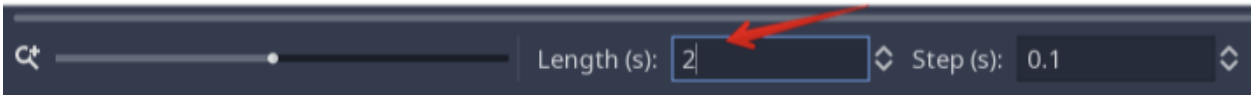


Fig. 12: Animation length

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. 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. 14: Animation loop

Fig. 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. 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.

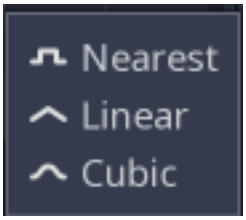


Fig. 17: Track interpolation

Fig. 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.

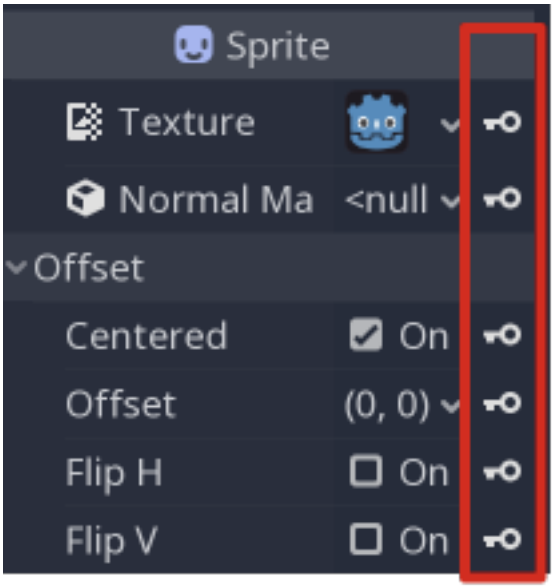


Fig. 19: Keyframes for other properties

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.

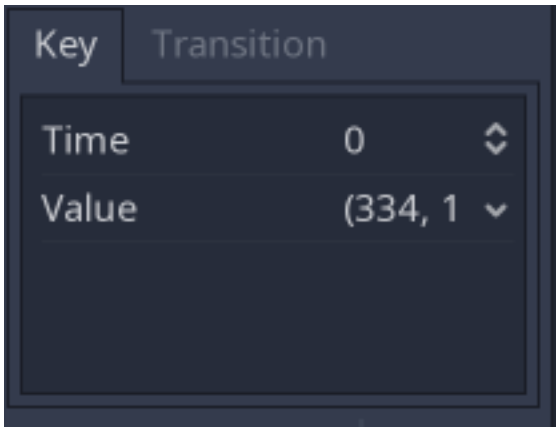


Fig. 20: Keyframe editor editing a key

Fig. 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.

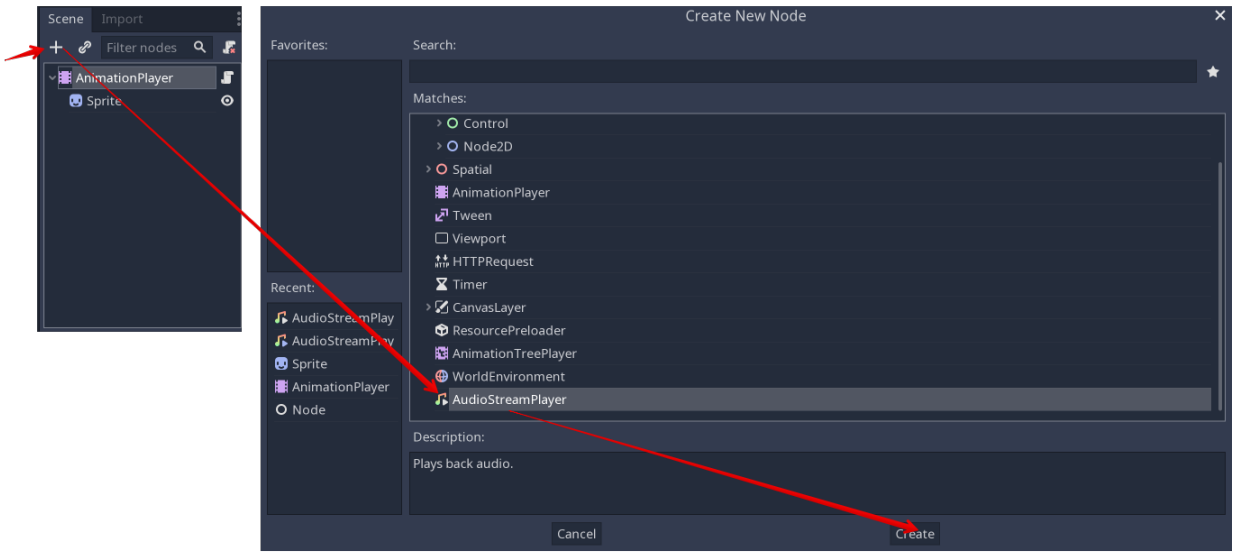


Fig. 22: Add AudioStreamPlayer

Click on "Add track" ( ) on the animation panel's track controls.

Select "Add Call Func Track" from the list of possible track types.

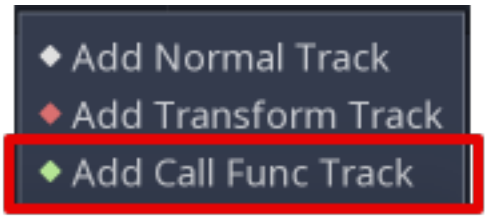


Fig. 23: Add Call Func Track

Select the *AudioStreamPlayer* node in the selection window. Godot adds the track with the reference to the node.

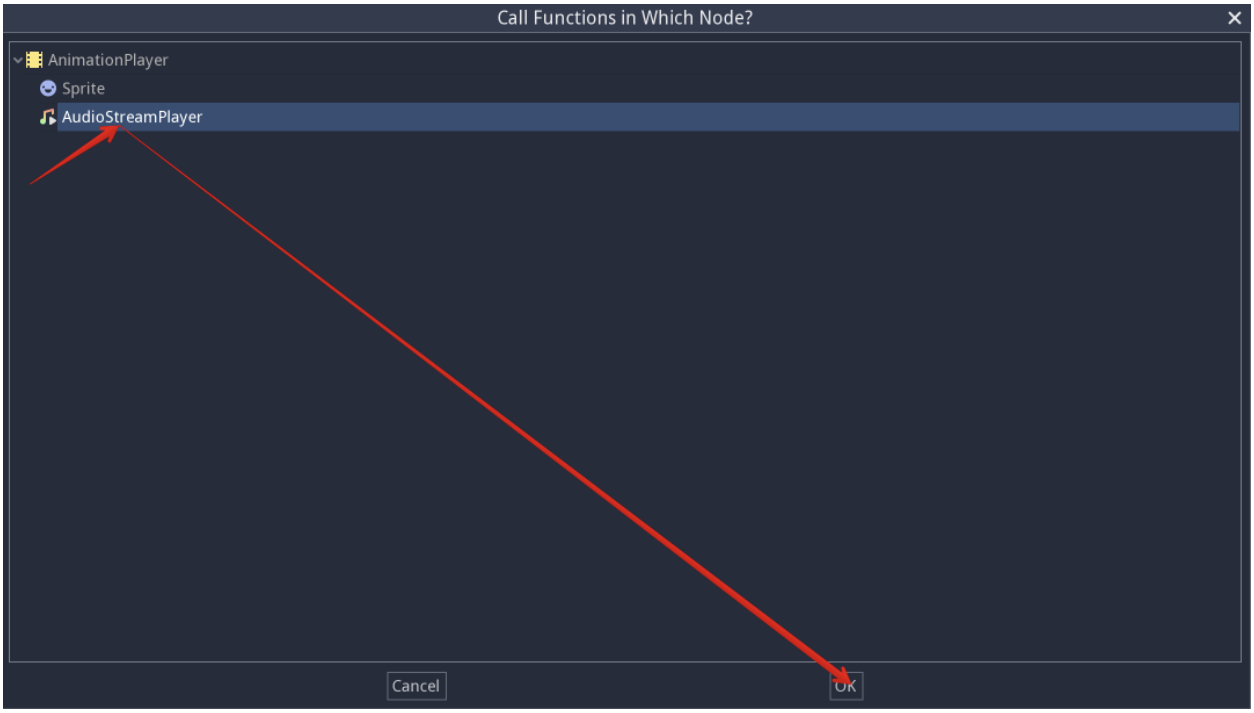


Fig. 24: Select AudioStreamPlayer

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.

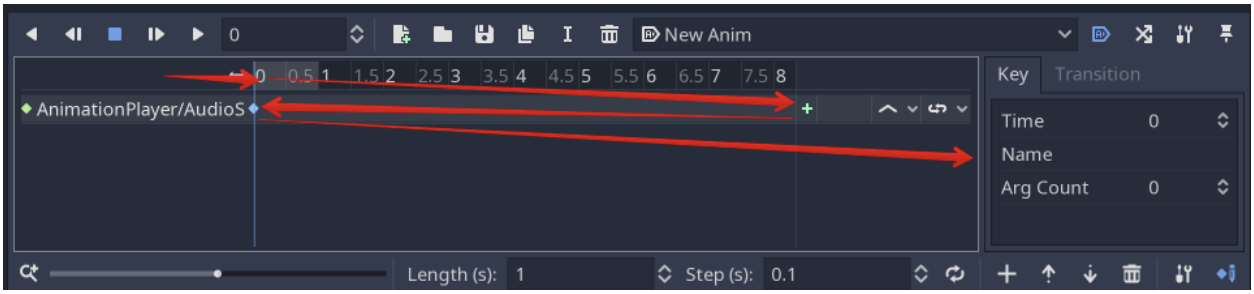


Fig. 25: Add a keyframe to the call func track

Enter “play” as the function name.

When Godot reaches the keyframe, Godot calls the *AnimationPlayer* node’s “play” function and the stream plays.

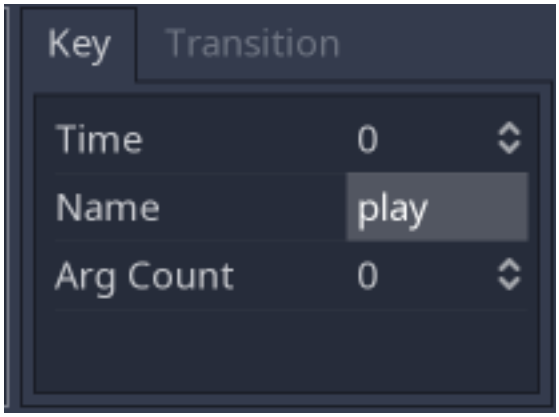


Fig. 26: Keyframe settings of a call func track

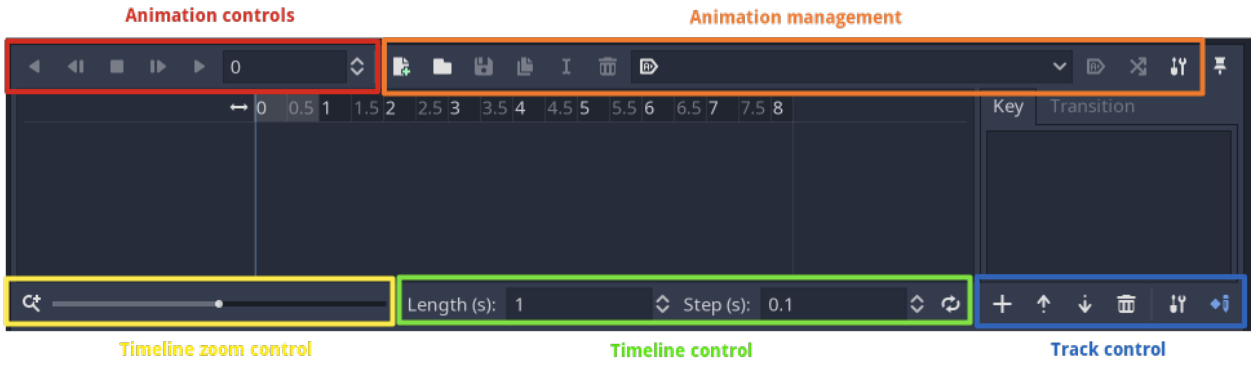


Fig. 27: The animation panel reference

11.1.8 References

Animation panel reference

The animation panel has the following parts (from left to right):

Animation controls

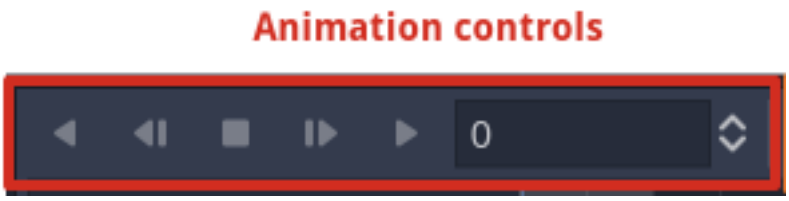


Fig. 28: Animation controls

- 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. 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

Fig. 30: Timeline zoom level contro

Timeline control

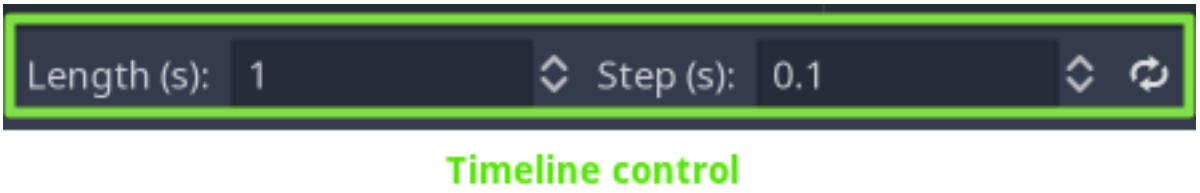


Fig. 31: Timeline control

- Length of animation
- Steps of animation
- Toggle loop animation

Track control

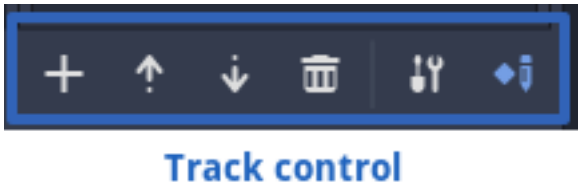


Fig. 32: Track control

- 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 has also become 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, 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!

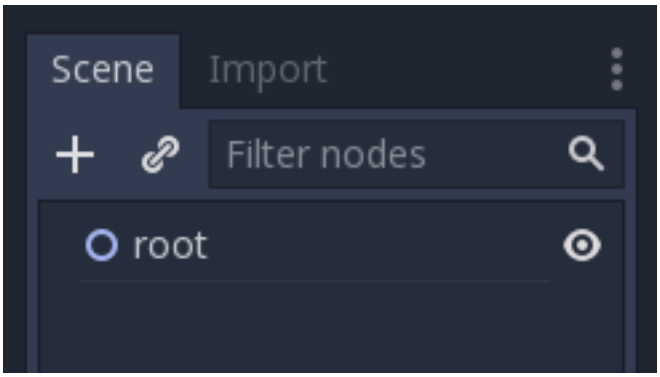
11.2.3 Making of GBot!

For this tutorial, we will use as demo content the pieces of the GBot character, created by Andreas Esau.

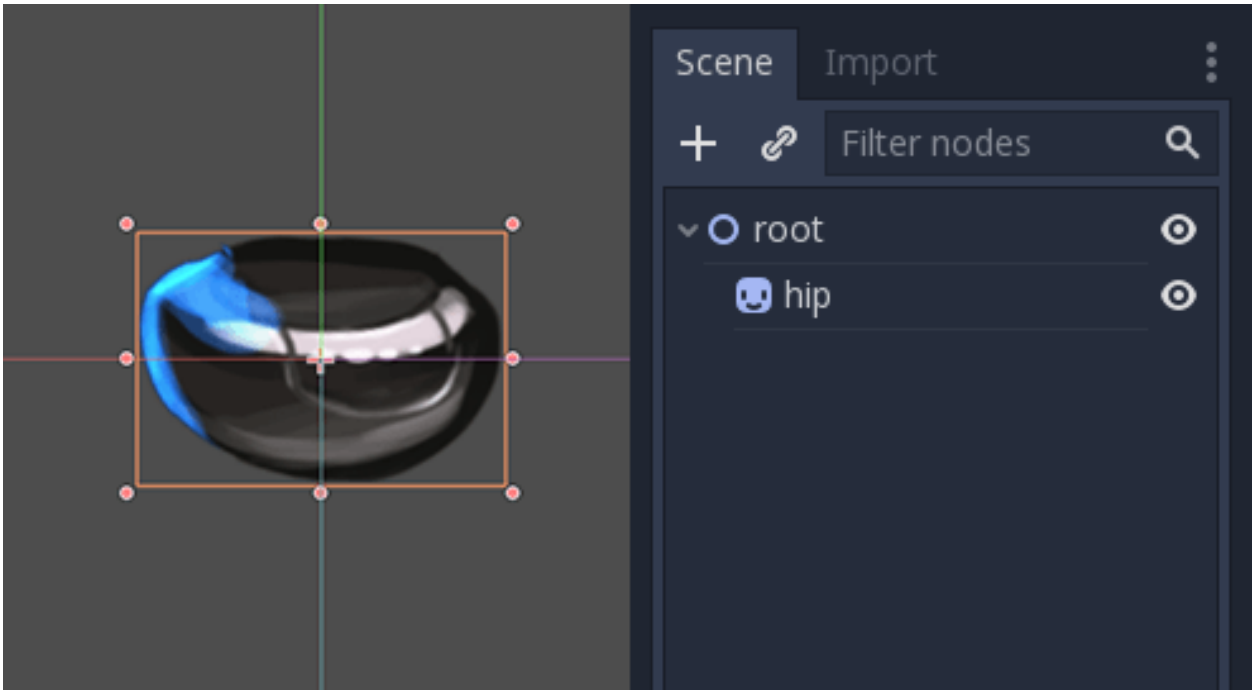
Get your assets: `gbot_resources.zip`.

11.2.4 Setting up the rig

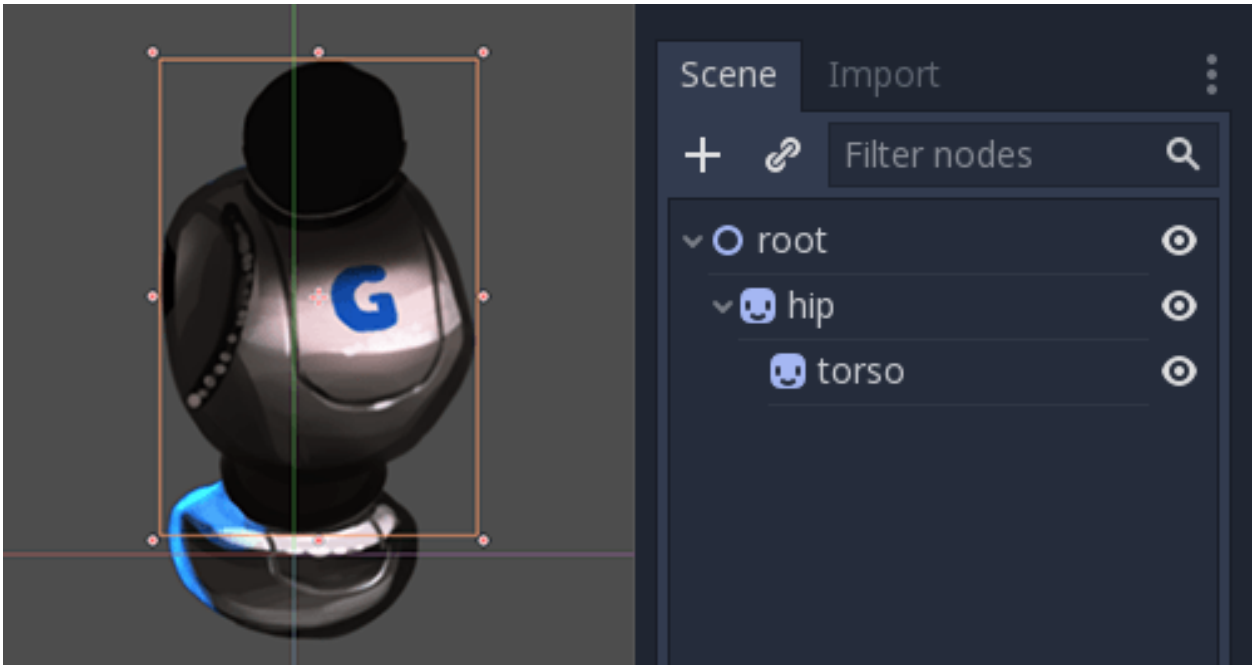
Create an empty Node2D as root of the scene, we will work under it:



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:



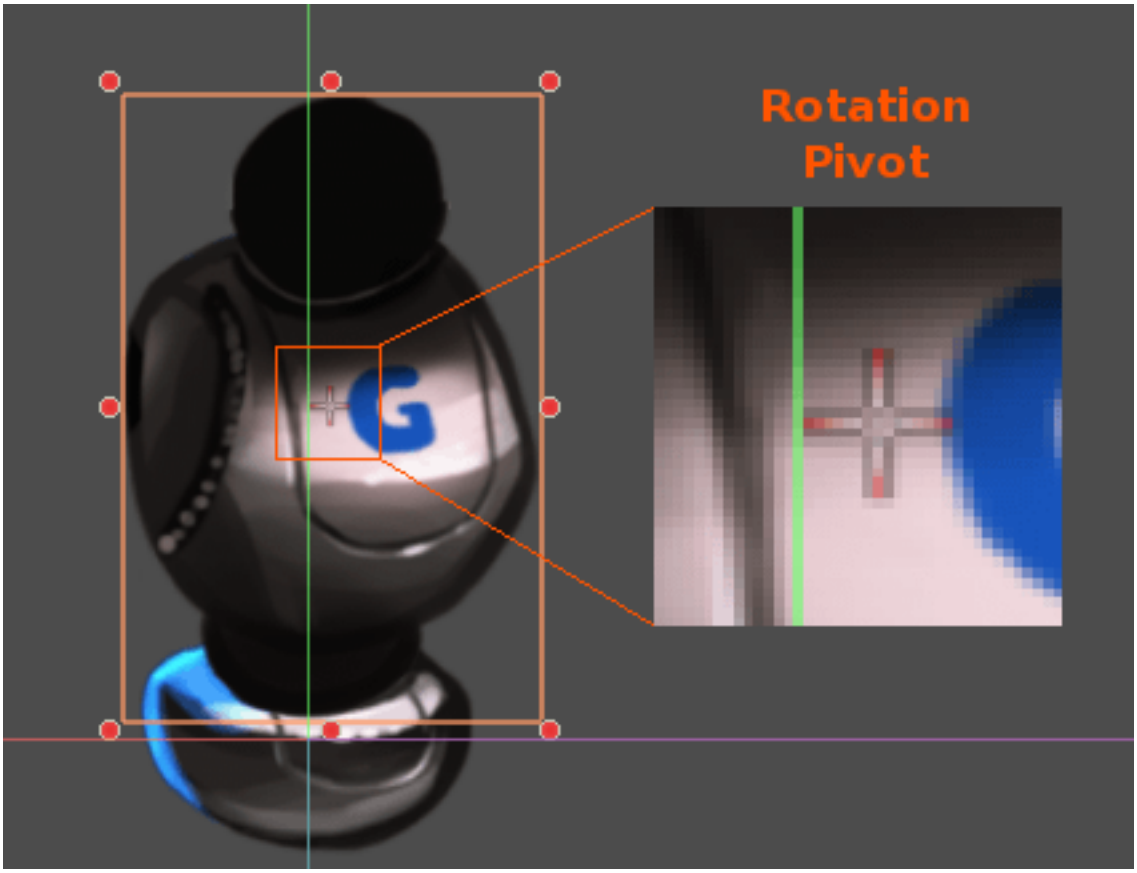
Next will be the torso. The torso needs to be a child of the hip, so create a child sprite and load the torso, later accommodate it properly:



This looks good. Let's see if our hierarchy works as a skeleton by rotating the torso. We can do this be pressing `E` to enter rotate mode, and dragging with the left mouse button. To exit rotate mode hit `ESC`.

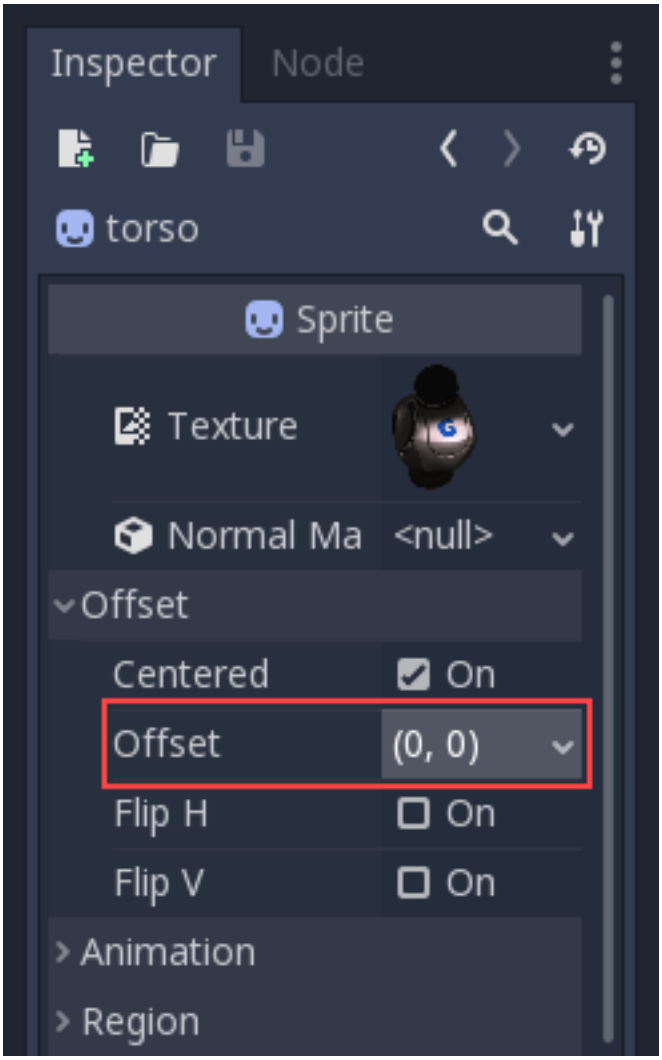
Ouch, that doesn't look good! The rotation pivot is wrong, this means it needs to be adjusted.

This small little cross in the middle of the *Sprite* is the rotation pivot:



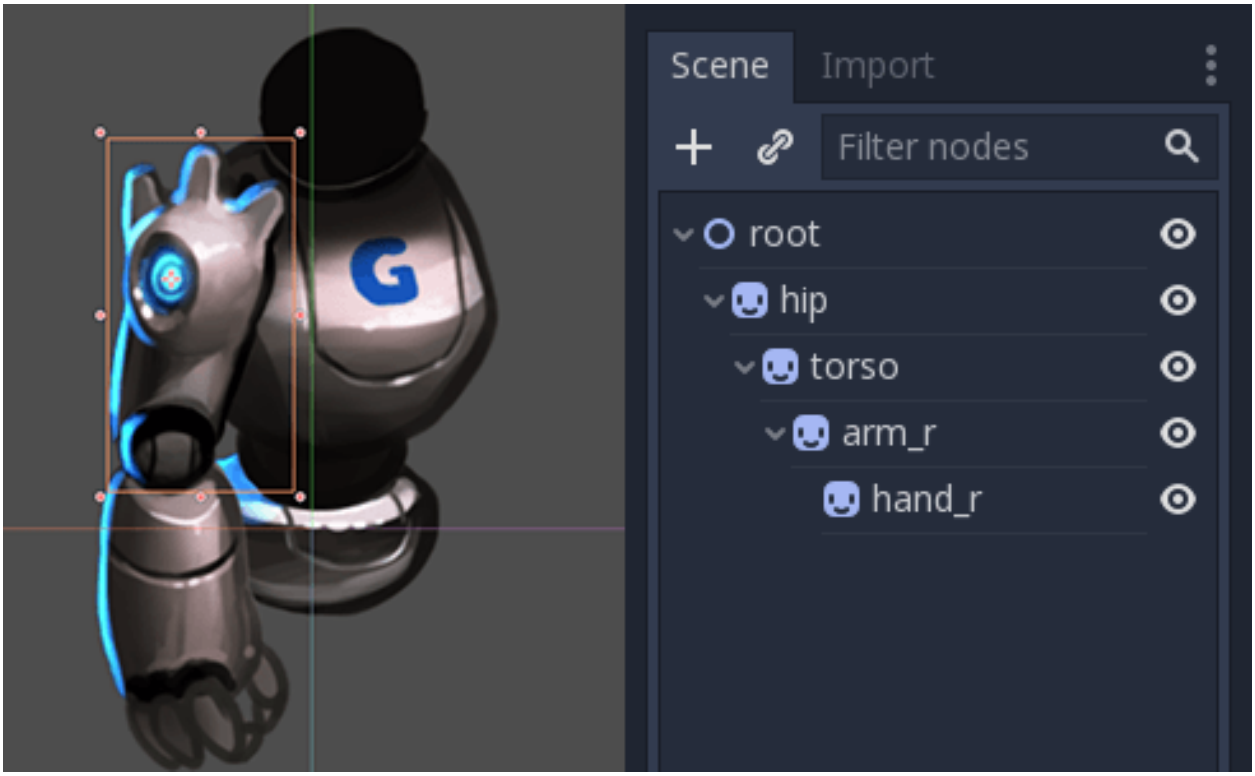
11.2.5 Adjusting the pivot

The pivot can be adjusted by changing the *offset* property in the Sprite:

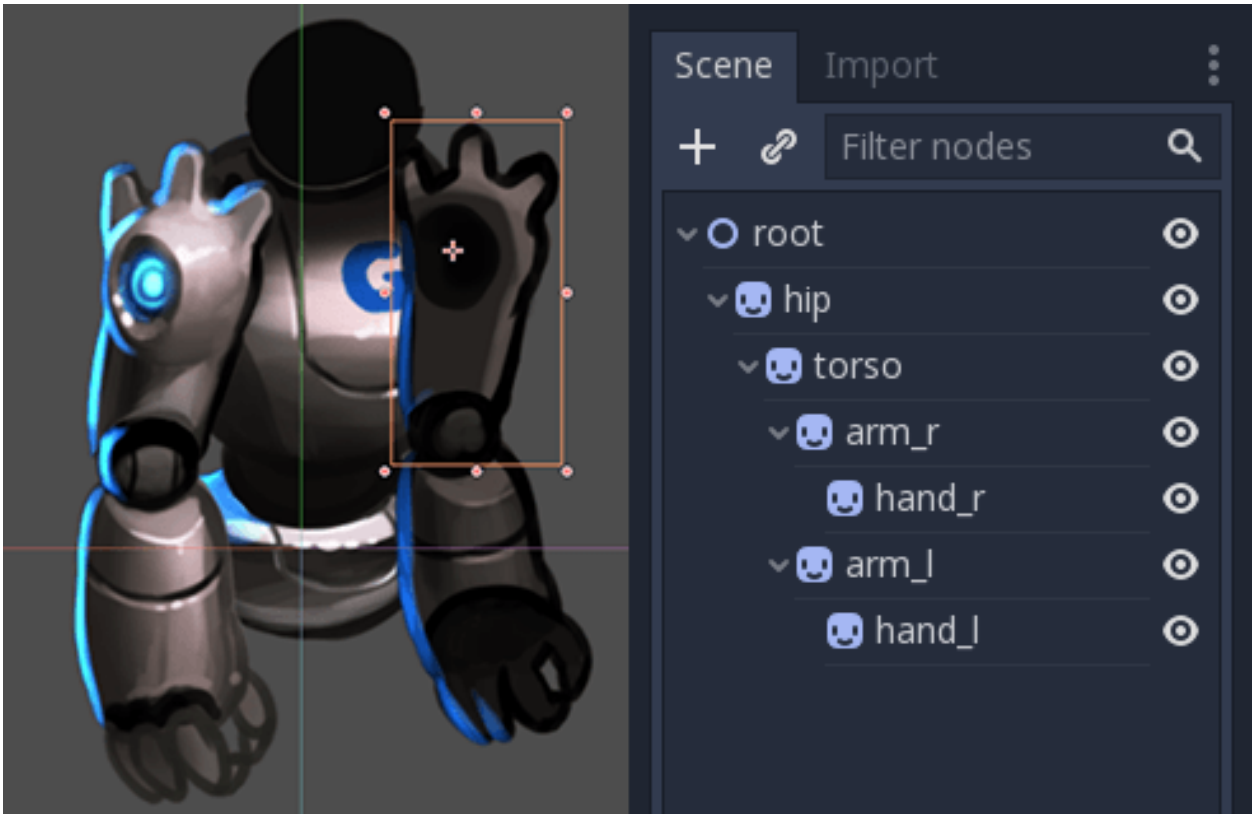


However, there is a way to do it more *visually*. While hovering over the desired pivot point, simply press the "v" key to move the pivot there for the selected Sprite. Alternately, there is a tool in the tool bar that has a similar function.

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:

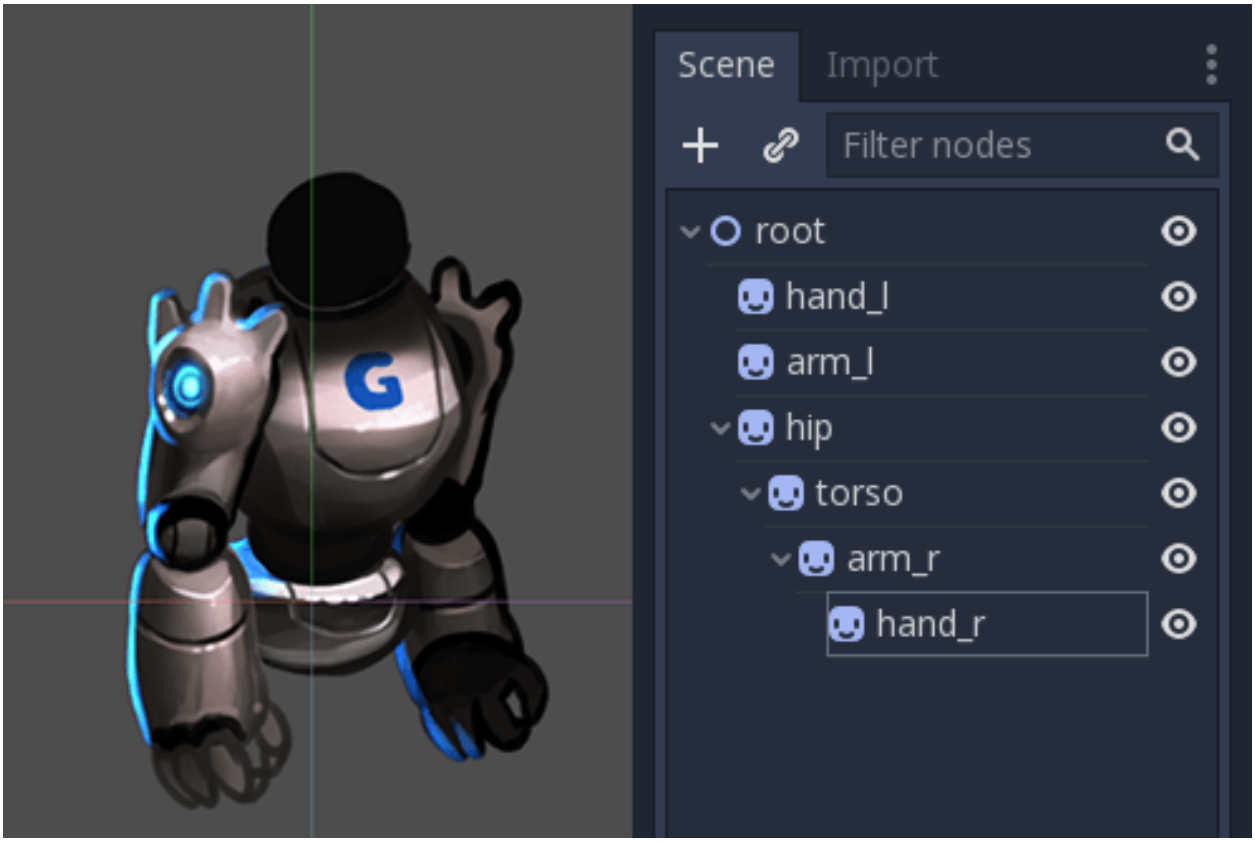


This seems easy, so continue with the left arm. The rest should be simple! Or maybe not:



Right. In 2D, parent nodes appear below children nodes. Well, this sucks. 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.

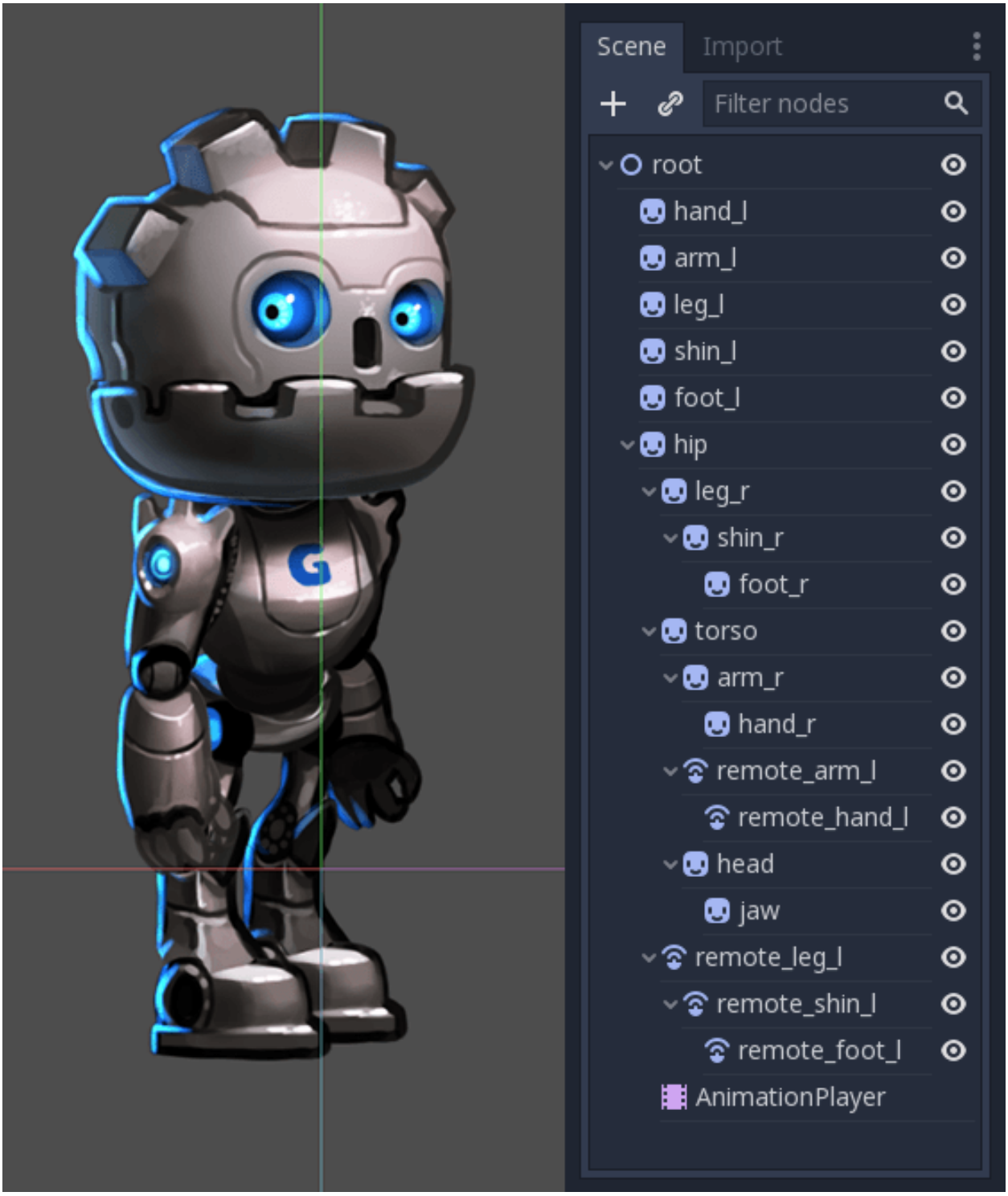
This enables to have a visibility order independent from the hierarchy.

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.

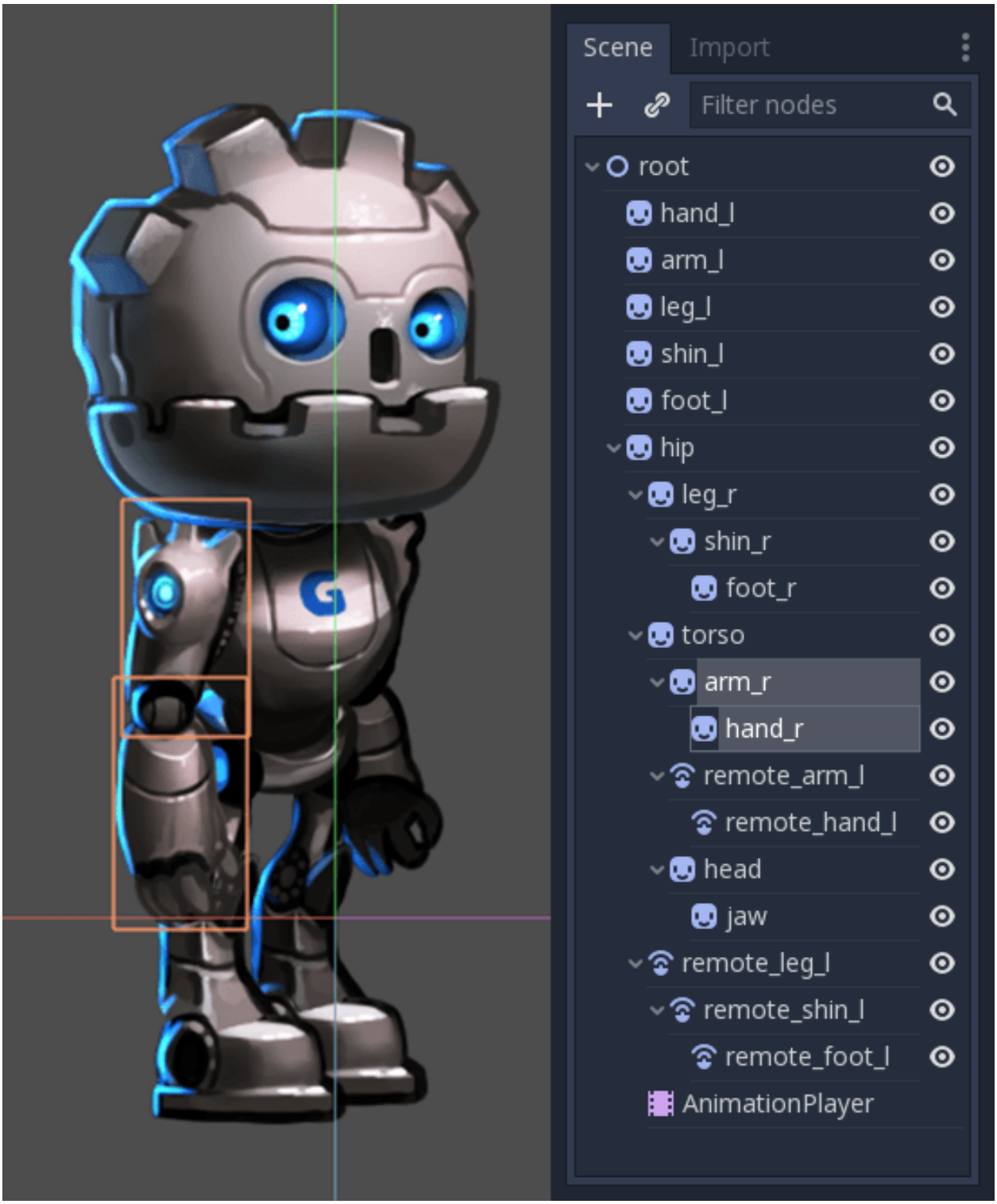
- Inverse Kinematics is often desired for extremities.

To solve these problems, Godot supports a simple method of skeletons.

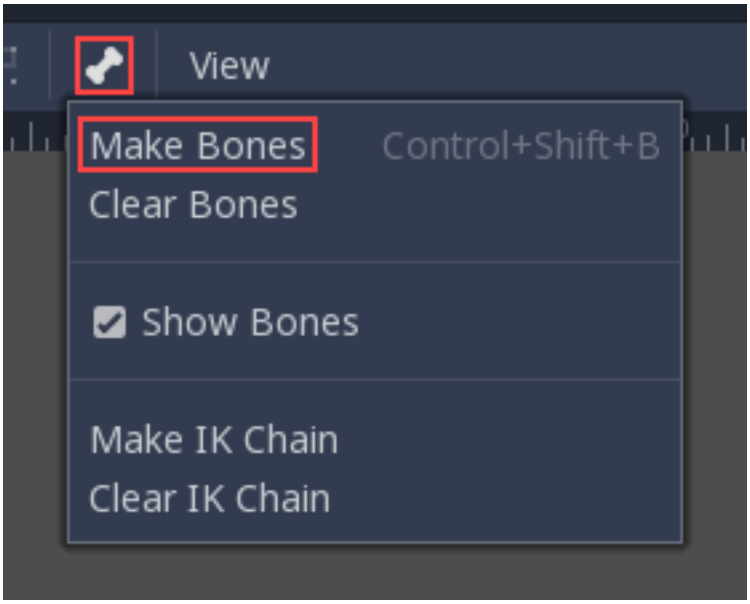
11.2.8 Skeletons

Godot doesn't actually support *true* Skeketons, 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.

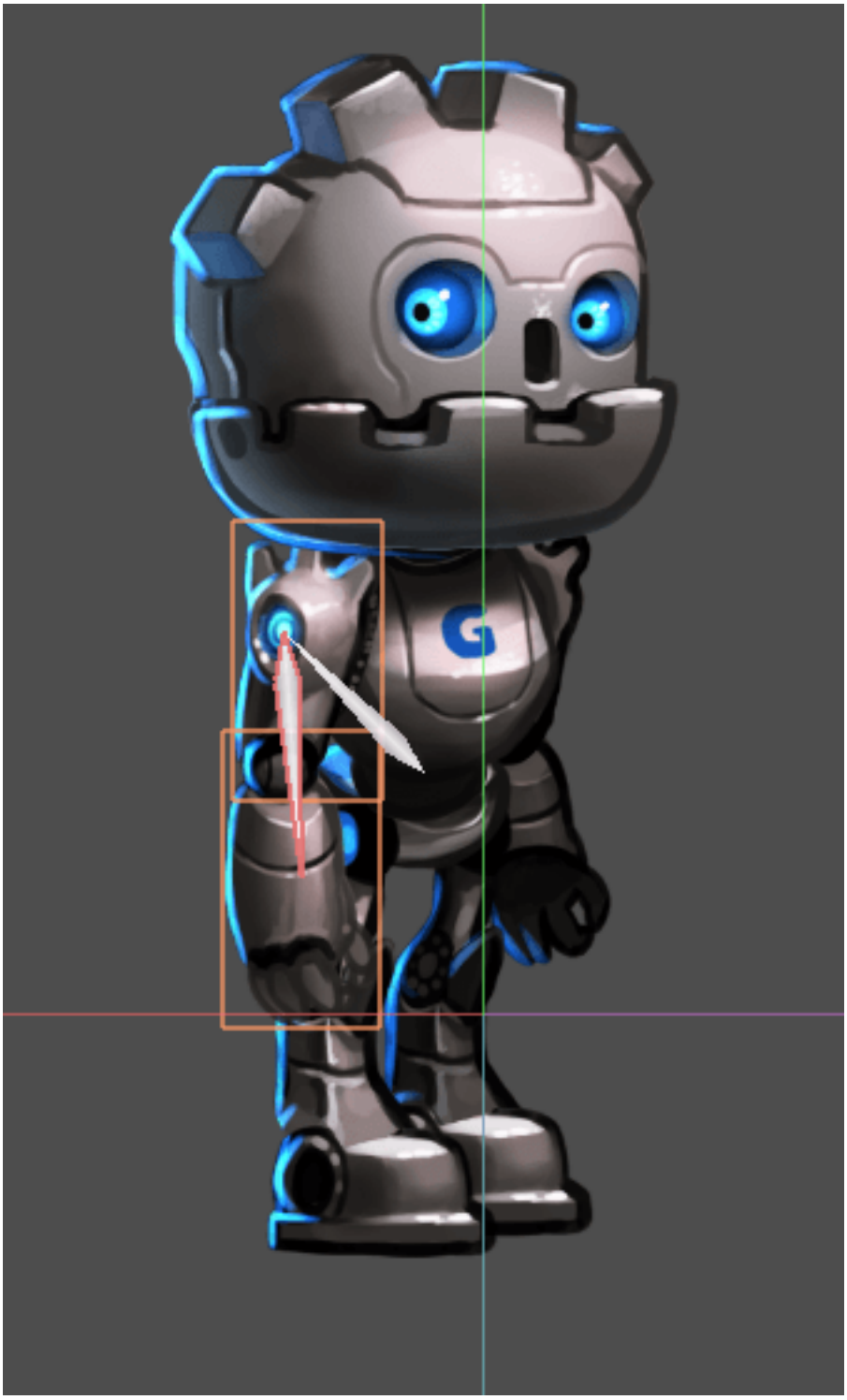
As an example, let's turn the right arm into a skeleton. To create skeletons, a chain of nodes must be selected from top to bottom:



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:

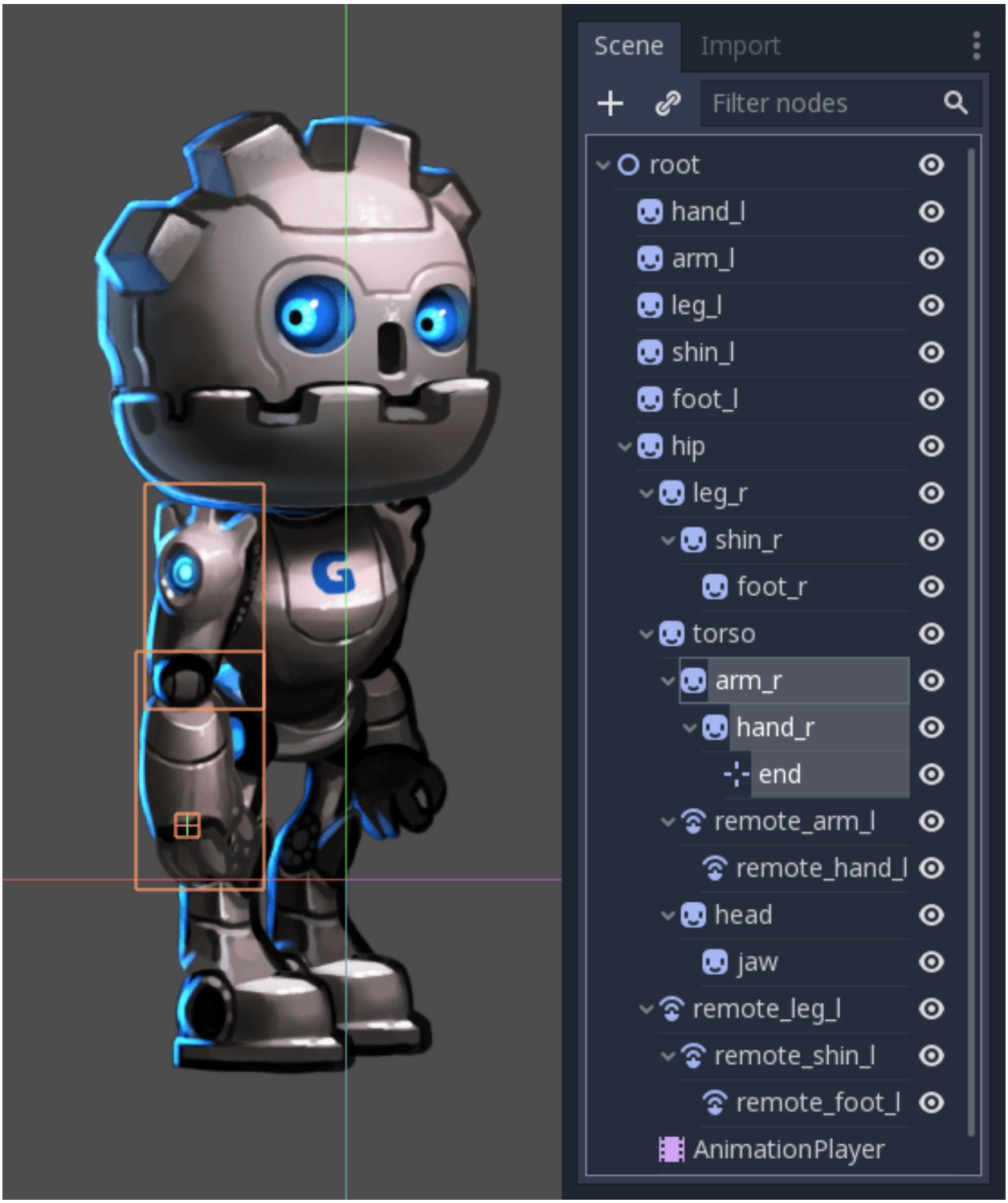
- Why does the hand lack a bone?
- Why does the arm connect to the body?

This might seem strange at first, but will make sense later on. In traditional skeleton systems, bones have a position, an

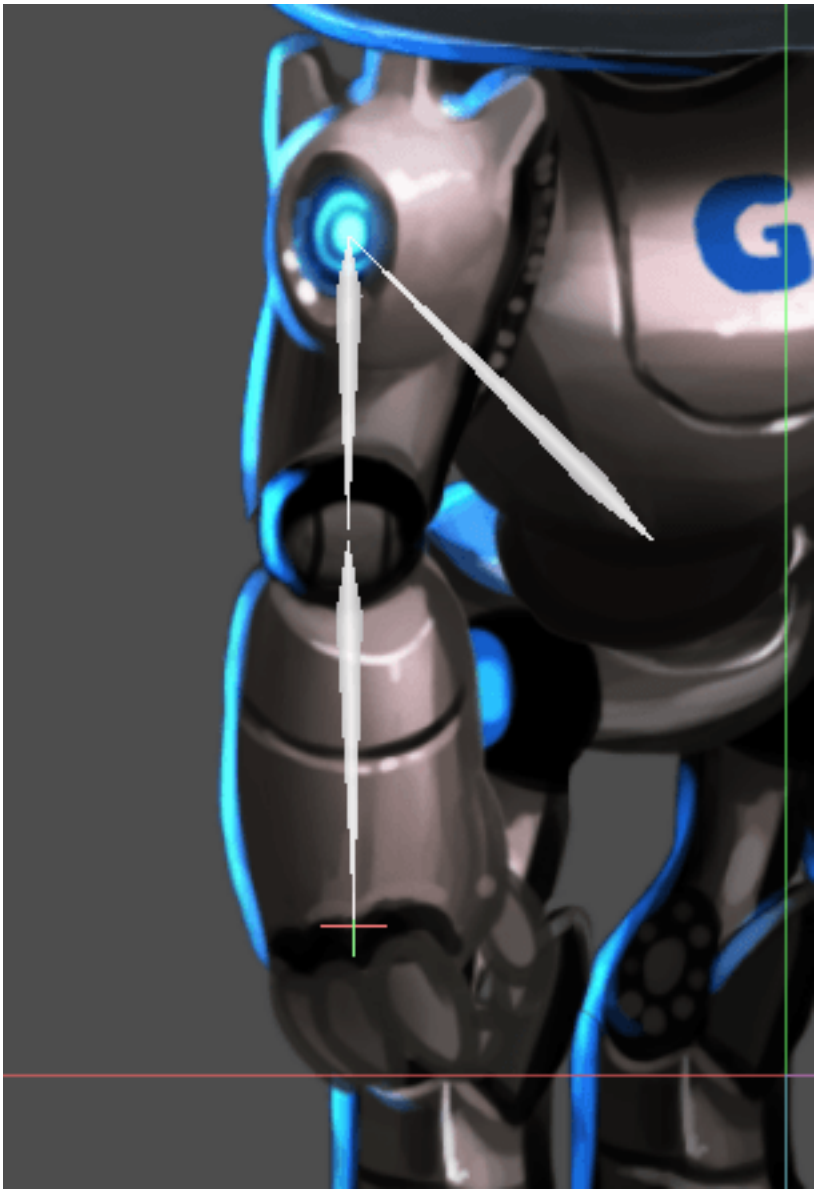
orientation and a length. In Godot, bones are mostly helpers so they connect the current node with the parent. Because of this, **toggling a node as a bone will just connect it to the parent**.

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:

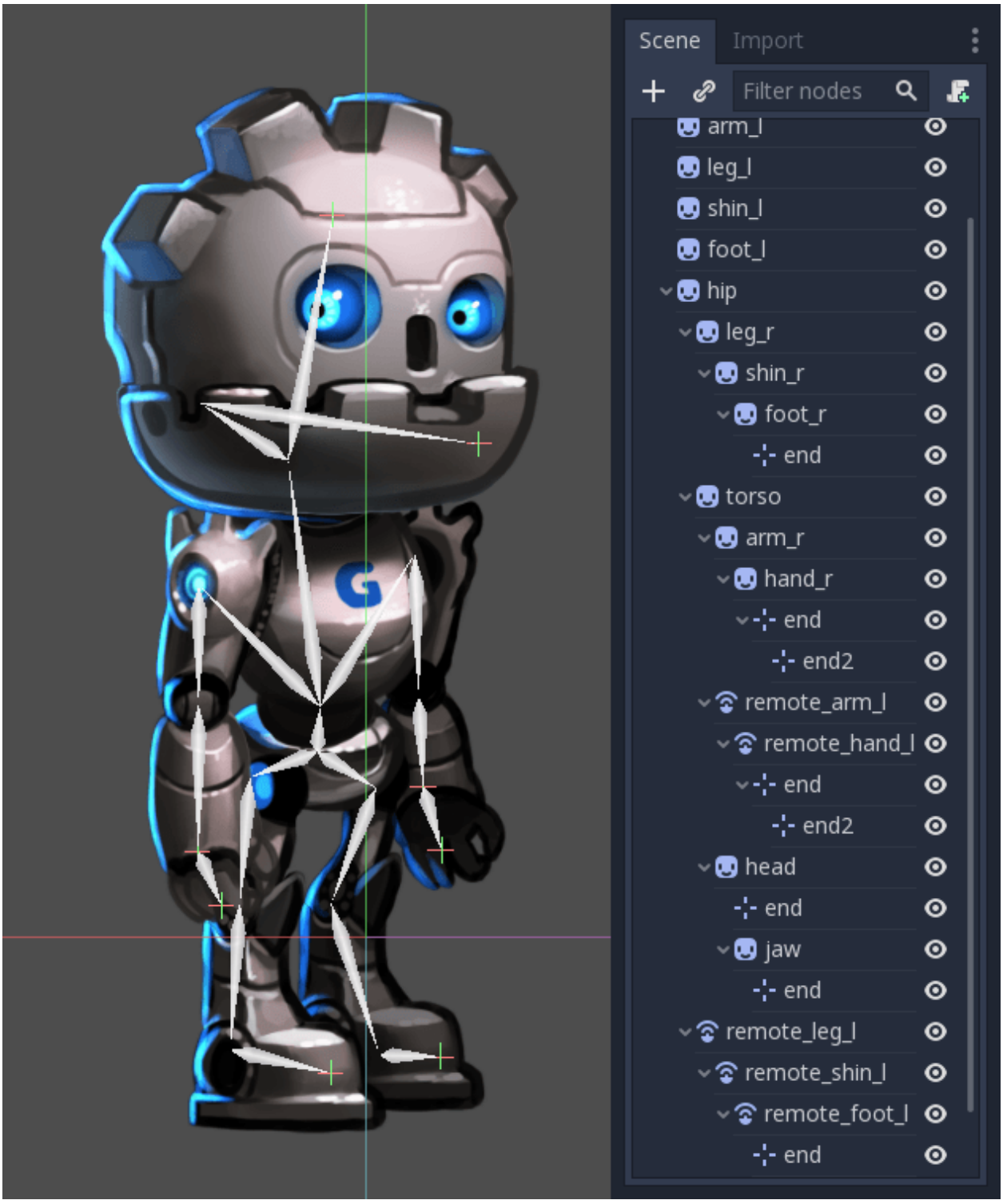


The result resembles a skeleton a lot more, and now the arm and forearm can be selected and animated.

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.

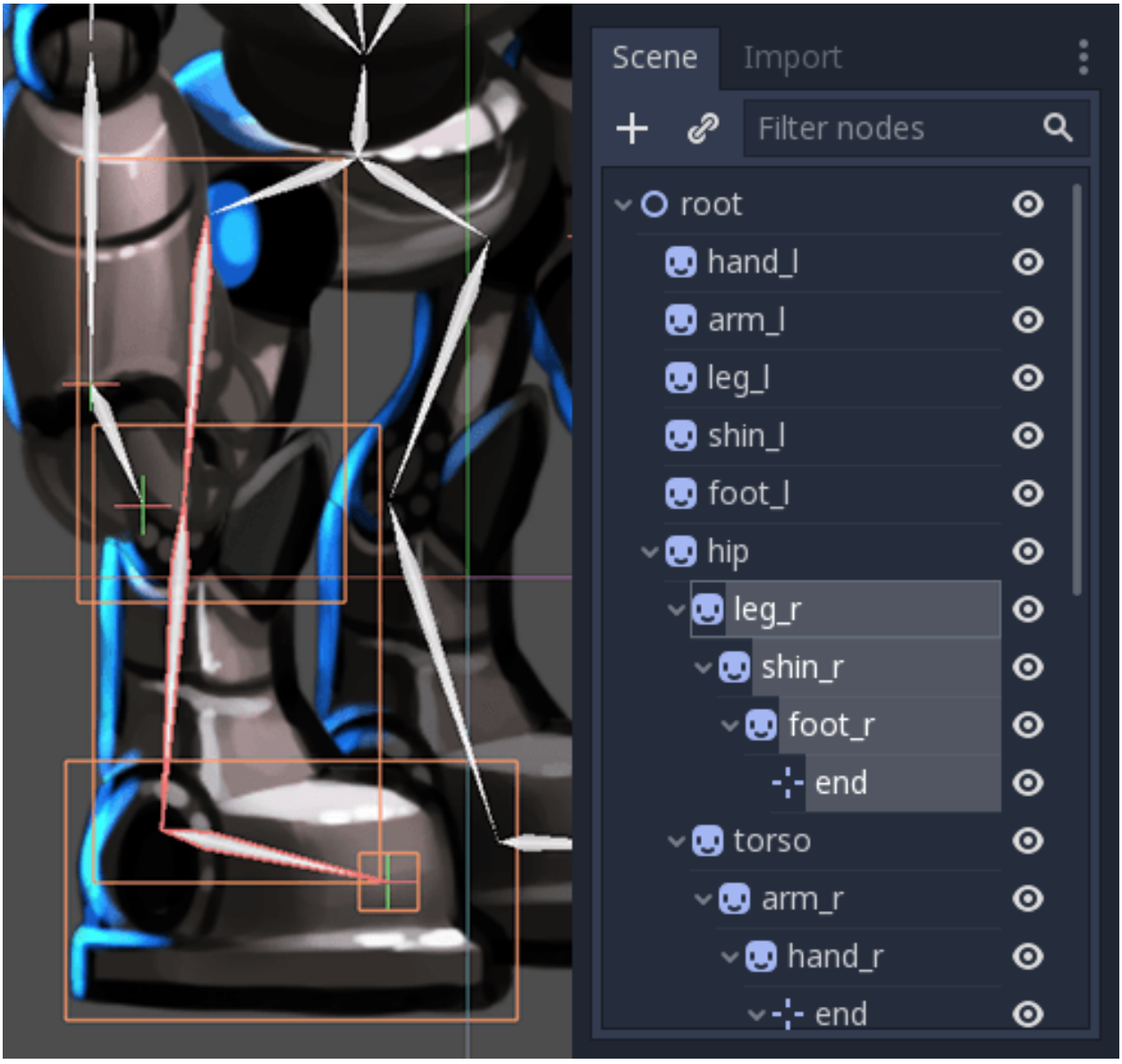
11.2.9 IK chains

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.

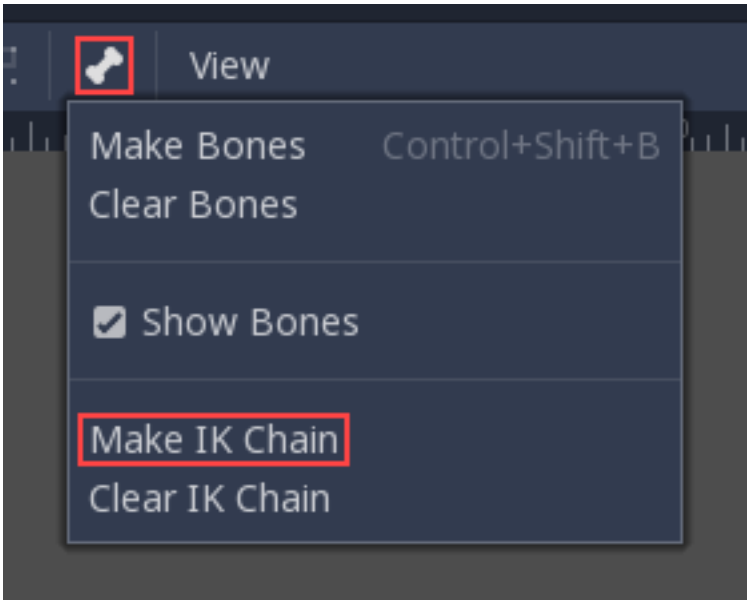
What if we could move the foot and let the rest of the leg self-adjust?

This type of posing is called IK (Inverse Kinematic).

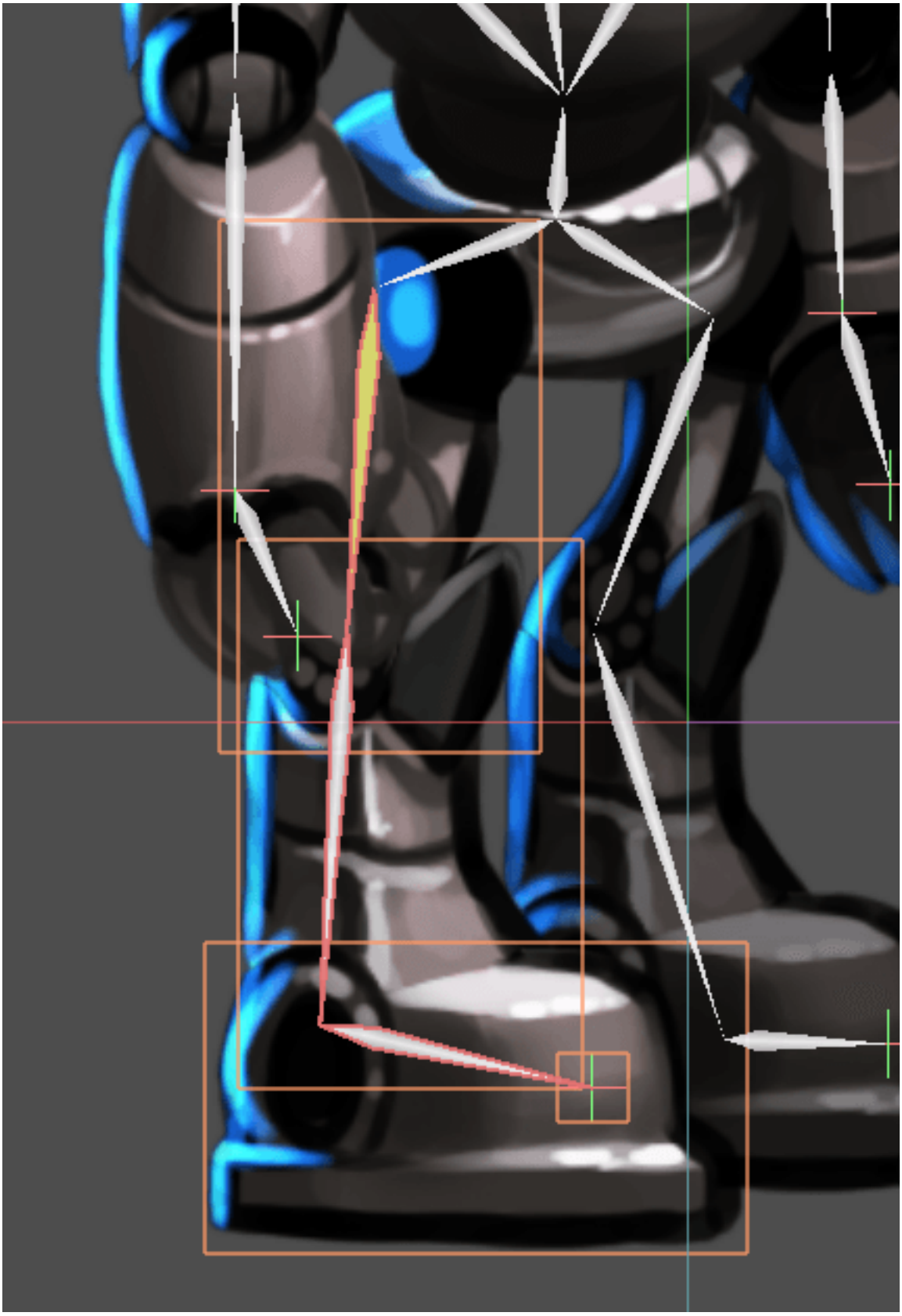
To create an IK chain, simply select a chain of bones from endpoint to the base for the chain. For example, to create an IK chain for the right leg, select the following:



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

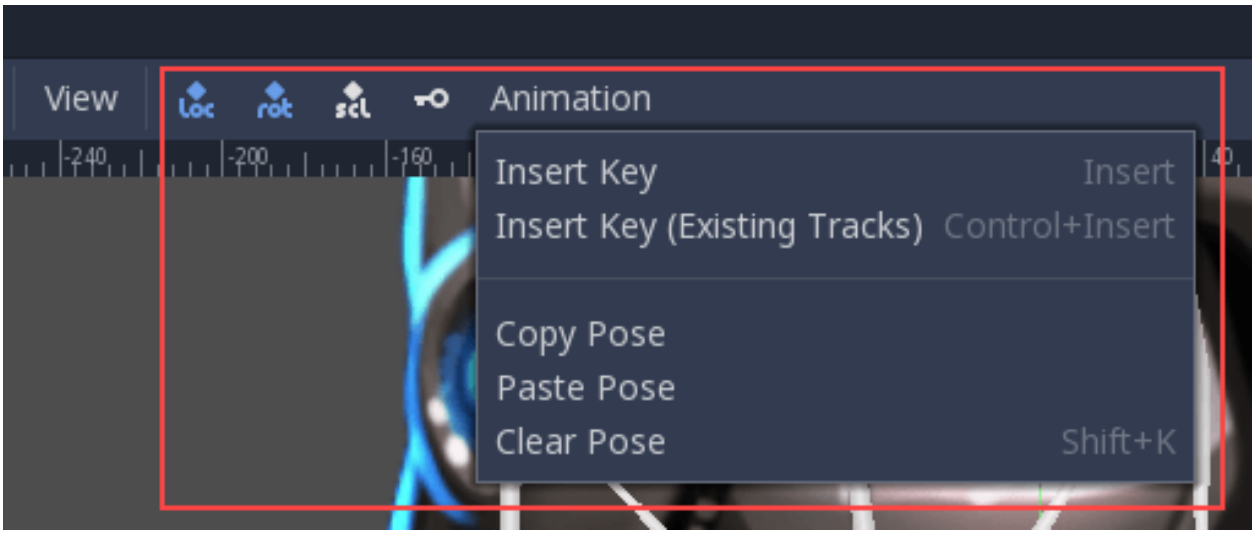
chain and try to grab it and move it. Result will be pleasant, satisfaction warranted!

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:



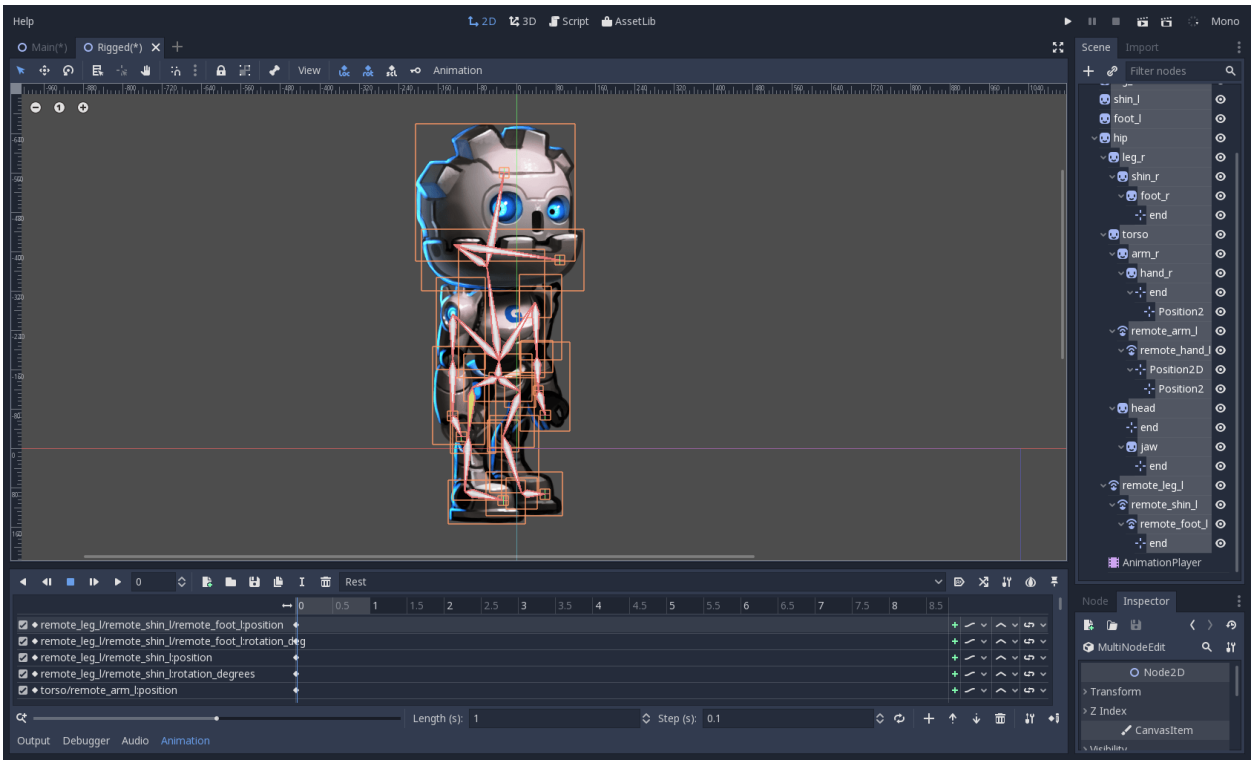
The key button will insert location/rotation/scale keyframes to the selected objects or bones. This depends on the mask enabled. Green items will insert keys while red ones will not, so modify the key insertion mask to your preference.

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:

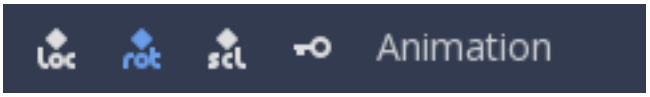
1. Make sure the rig is in "rest" (not doing any specific pose).
2. Create a new animation, rename it to "rest".
3. Select all nodes (box selection should work fine).
4. Select "loc" and "rot" on the top menu.
5. Push the key button. Keys will be inserted for everything, creating a default pose.



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:



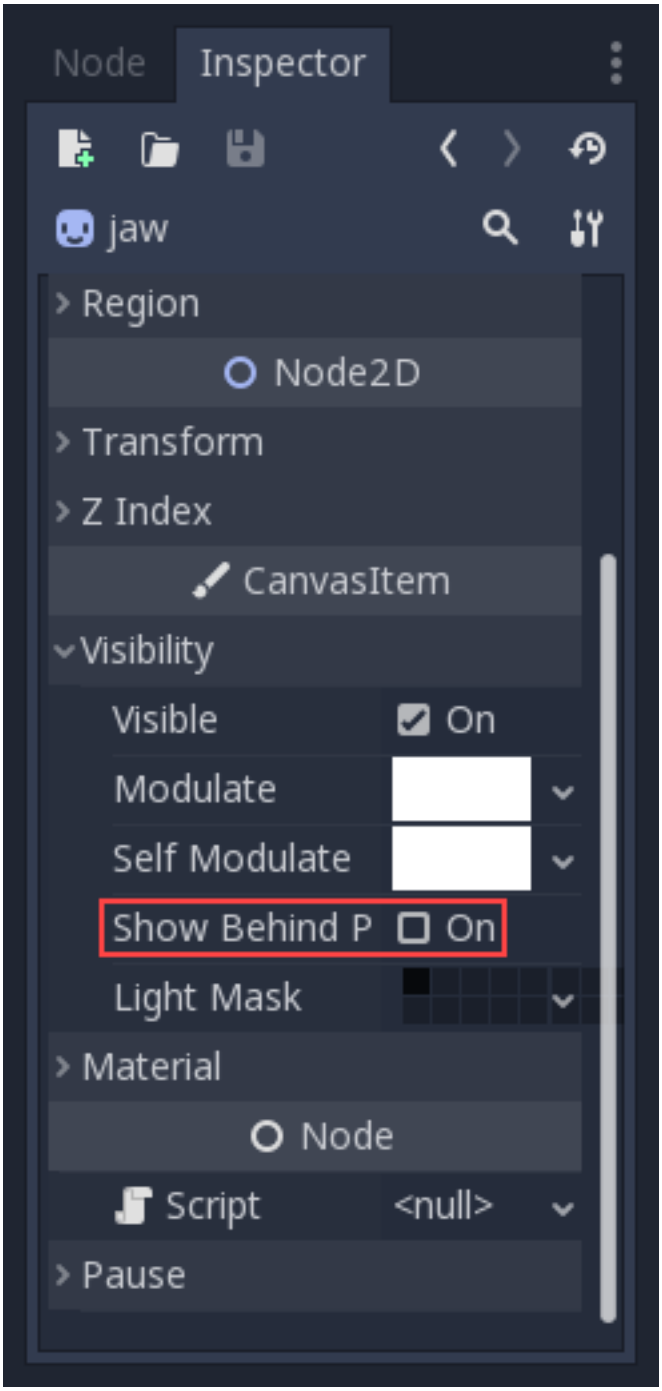
This will avoid the creation of extra animation tracks for the position that will remain unused.

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.

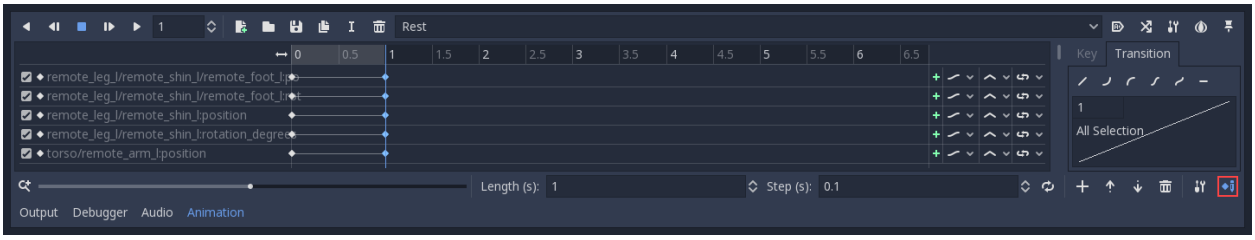
11.2.14 Moving sprites above and behind others.

RemoteTransform2D works in most cases, but sometimes it is 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 complex animations and inserting many 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.


Rest
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
Key
Transition
remote_leg_l/remote_shin_l/remote_foot_l:p
remote_leg_l/remote_shin_l/remote_foot_l:r
remote_leg_l/remote_shin_l:position
remote_leg_l/remote_shin_l:rotation_degree
torso/remote_arm_l:position
1
All Selection
Length (s):
1
Step (s):
0.1
Output
Debugger
Audio
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.

Here is a quick example, closing your game if the escape key is hit:

GDScript

C#

```
func _unhandled_input(event):
    if event is InputEventKey:
        if event.pressed and event.scancode == KEY_ESCAPE:
            get_tree().quit()
```

```
public override void _UnhandledInput(InputEvent @event)
{
    if (@event is InputEventKey eventKey)
        if (eventKey.Pressed && eventKey.Scancode == (int)KeyList.Escape)
            GetTree().Quit();
}
```

However, it is cleaner and more flexible to use the provided *InputMap* feature, which allows you to define input actions and assign them different keys. This way, you can define multiple keys for the same action (e.g. they keyboard escape key and the start button on a gamepad). You can then more easily change this mapping in the project settings without updating your code, and even build a key mapping feature on top of it to allow your game to change the key mapping at runtime!

You can setup your InputMap under **Project > Project Settings > Input Map** and then use those actions like this:

GDScript

C#

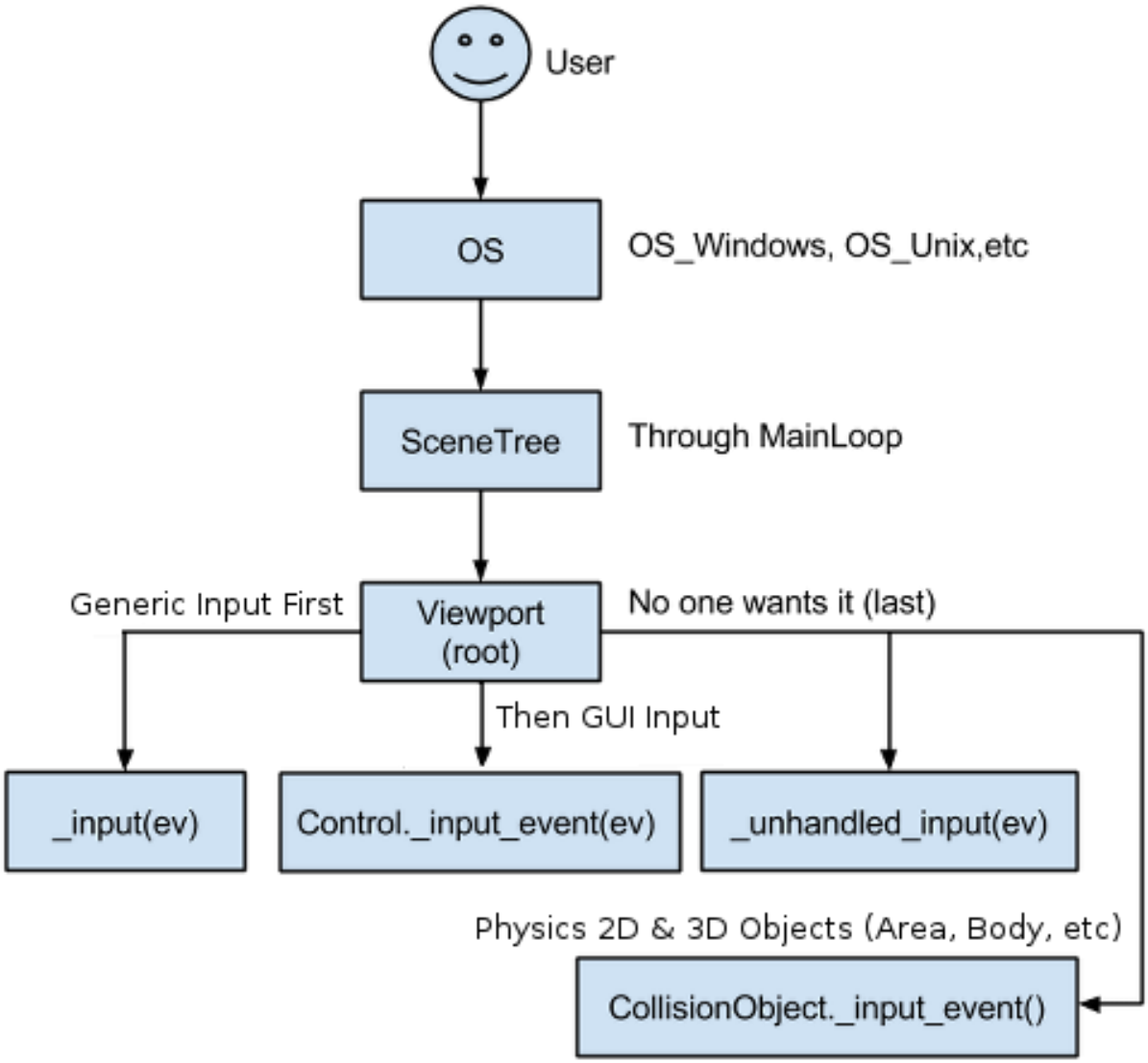
```
func _process(delta):
    if Input.is_action_pressed("ui_right"):
        # Move right
```

```
public override void _Process(float delta)
{
    if (Input.IsActionPressed("ui_right"))
    {
        // Move right
    }
}
```

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_process_unhandled_input()*). 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.

There are several specialised types of InputEvent, described in the table below:

Event	Type Index	Description
InputEvent	NONE	Empty Input Event.
InputEventKey	KEY	Contains a scancode and unicode value, as well as modifiers.
InputEventMouseButton	MOUSE_BUTTON	Contains click information, such as button, modifiers, etc.
InputEventMouseMotion	MOUSE_MOTION	Contains motion information, such as relative, absolute positions and speed.
InputEventJoypadMotion	JOYSTICK_MOTION	Contains Joystick/Joypad analog axis information.
InputEventJoypadButton	JOYSTICK_BUTTON	Contains Joystick/Joypad button information.
InputEventScreenTouch	SCREEN_TOUCH	Contains multi-touch press/release information. (only available on mobile devices)
InputEventScreenDrag	SCREEN_DRAG	Contains multi-touch drag information. (only available on mobile devices)
InputEventAction	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.

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:

GDScript

C#

```
var ev = InputEventAction.new()
# set as move_left, pressed
ev.set_as_action("move_left", true)
# feedback
Input.parse_input_event(ev)
```

```
var ev = new InputEventAction();
// set as move_left, pressed
ev.SetAction("move_left");
ev.SetPressed(true);
// feedback
Input.ParseInputEvent(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:

GDScript

C#

```
func _input(event):
    # Mouse in viewport coordinates
    if event is InputEventMouseButton:
        print("Mouse Click/Unclick at: ", event.position)
    elif event is InputEventMouseMotion:
        print("Mouse Motion at: ", event.position)

    # Print the size of the viewport
    print("Viewport Resolution is: ", get_viewport_rect().size)
```

```
public override void _Input(InputEvent @event)
{
    // Mouse in viewport coordinates
    if (@event is InputEventMouseButton eventMouseButton)
        GD.Print("Mouse Click/Unclick at: ", eventMouseButton.Position);
    else if (@event is InputEventMouseMotion eventMouseMotion)
        GD.Print("Mouse Motion at: ", eventMouseMotion.Position);

    // Print the size of the viewport
    GD.Print("Viewport Resolution is: ", GetViewPortRect().Size);
}
```

Alternatively it's possible to ask the viewport for the mouse position:

GDScript

C#

```
get_viewport().get_mouse_position()
```

```
GetViewport().GetMousePosition();
```

12.3 Customizing mouse cursor

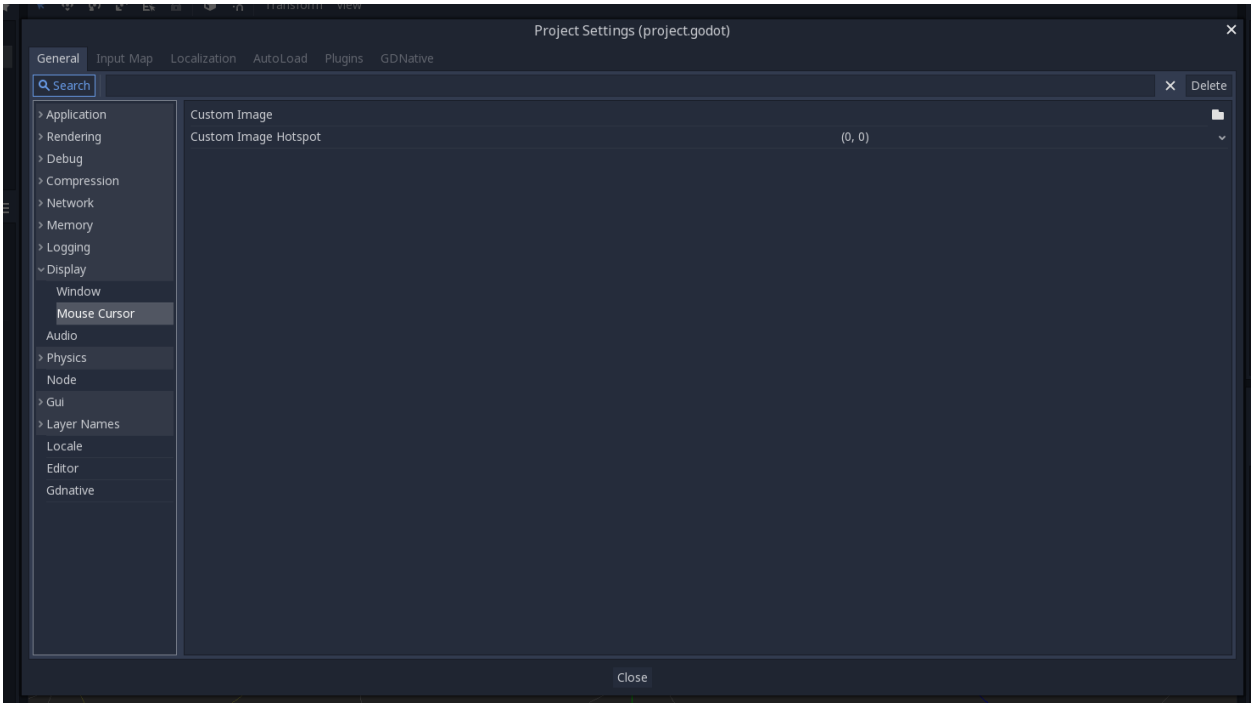
You might want to change the appearance of the mouse cursor in your game in order to suit the overall design. There are two ways to customize the mouse cursor:

1. Using project settings
2. Using a script

Using project settings is a simpler but more limited way to customize the mouse cursor. The second way is more customizable but involves scripting.

12.3.1 Using project settings

Open project settings, go to Display>Mouse Cursor. You will see Custom Image and Custom Image Hotspot.



Custom Image is the desired image that you would like to set as the mouse cursor. Custom Hotspot is the point in the image that you would like to use as the cursor's detection point.

Note: The custom image **must** be less than 256x256.

12.3.2 Using a script

Create a Node and attach the following script.

GDScript

C#

```
extends Node

# Load the custom images for the mouse cursor
var arrow = load("res://arrow.png")
var beam = load("res://beam.png")

func _ready():
    # Changes only the arrow shape of the cursor
    # This is similar to changing it in the project settings
    Input.set_custom_mouse_cursor(arrow)

    # Changes a specific shape of the cursor (here the IBeam shape)
    Input.set_custom_mouse_cursor(beam, Input.CURSOR_IBEAM)
```

```
public override void _Ready()
{
    // Load the custom images for the mouse cursor
    var arrow = ResourceLoader.Load("res://arrow.png");
```

(continues on next page)

(continued from previous page)

```
    var beam = ResourceLoader.Load("res://beam.png");

    // Changes only the arrow shape of the cursor
    // This is similar to changing it in the project settings
    Input.SetCustomMouseCursor(arrow);

    // Changes a specific shape of the cursor (here the IBeam shape)
    Input.SetCustomMouseCursor(beam, Input.CursorShape.Ibeam);
}
```

Note: Check *Input.set_custom_mouse_cursor()*.

12.3.3 Demo project

Find out more by studying this demo project: https://github.com/guilhermefelipecgs/custom_hardware_cursor

12.3.4 Cursor list

As documented in the *Input* class (see the **CursorShape** enum), there are multiple mouse cursors you can define. Which ones you want to use depends on your use case.

CHAPTER 13

I/O

13.1 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.

13.1.1 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.

13.1.2 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.

13.1.3 Example

This example demostrates 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)
```

(continues on next page)

(continued from previous page)

```
    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 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?
```

(continues on next page)

(continued from previous page)

```
    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)
```

13.1.4 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.

13.1.5 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
                                                 # of the queue, pausing the load of␣
↪any other resource

# 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.

13.2 Data paths

13.2.1 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`.

13.2.2 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`.

13.2.3 Userdata path (persistent data)

While the project is running, it is a 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 `~/.local/share/godot/app_userdata/Name` (check the project name under the settings) in macOS and Linux, and `APPDATA/Name` for Windows.

13.3 Saving games

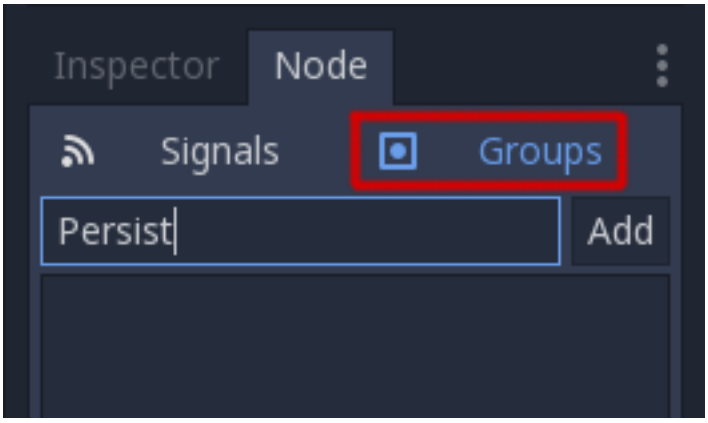
13.3.1 Introduction

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.

13.3.2 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:

GDScript

C#

```
var save_nodes = get_tree().get_nodes_in_group("Persist")
for i in save_nodes:
    # Now we can call our save function on each node.
```

```
var saveNodes = GetTree().GetNodesInGroup("Persist");
foreach (Node saveNode in saveNodes)
{
    // Now we can call our save function on each node.
}
```

13.3.3 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:

GDScript

C#

```
func save():
    var save_dict = {
        "filename" : get_filename(),
        "parent" : get_parent().get_path(),
        "pos_x" : position.x, # Vector2 is not supported by JSON
        "pos_y" : position.y,
        "attack" : attack,
        "defense" : defense,
        "current_health" : current_health,
        "max_health" : max_health,
        "damage" : damage,
        "regen" : regen,
        "experience" : experience,
        "tnl" : tnl,
        "level" : level,
        "attack_growth" : attack_growth,
        "defense_growth" : defense_growth,
        "health_growth" : health_growth,
        "is_alive" : is_alive,
        "last_attack" : last_attack
    }
    return save_dict
```

```
public Dictionary<object, object> Save()
{
    return new Dictionary<object, object>()
    {
        { "Filename", GetFilename() },
        { "Parent", GetParent().GetPath() },
        { "PosX", Position.x }, // Vector2 is not supported by JSON
        { "PosY", Position.y },
```

(continues on next page)

(continued from previous page)

```
        { "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 },
        { "LastAttack", LastAttack }
    };
}
```

This gives us a dictionary with the style `{ "variable_name":that_variables_value }` which will be useful when loading.

13.3.4 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.

GDScript

C#

```
# 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 save_game = File.new()
    save_game.open("user://savegame.save", File.WRITE)
    var save_nodes = get_tree().get_nodes_in_group("Persist")
    for i in save_nodes:
        var node_data = i.call("save");
        save_game.store_line(to_json(node_data))
    save_game.close()
```

```
// 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
public void SaveGame()
{
    var saveGame = new File();
    saveGame.Open("user://savegame.save", (int)File.ModeFlags.Write);

    var saveNodes = GetTree().GetNodesInGroup("Persist");
    foreach (Node saveNode in saveNodes)
```

(continues on next page)

(continued from previous page)

```
    {
        var nodeData = saveNode.Call("Save");
        saveGame.StoreLine(JSON.Print(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:

GDScript

C#

```
# Note: This can be called from anywhere inside the tree. This function is path␣
↪independent.
func load_game():
    var save_game = File.new()
    if not save_game.file_exists("user://save_game.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 save_nodes = get_tree().get_nodes_in_group("Persist")
    for i in save_nodes:
        i.queue_free()

    # Load the file line by line and process that dictionary to restore the object it␣
↪represents
    save_game.open("user://savegame.save", File.READ)
    while not save_game.eof_reached():
        var current_line = parse_json(save_game.get_line())
        # First we need to create the object and add it to the tree and set its␣
↪position.
        var new_object = load(current_line["filename"]).instance()
        get_node(current_line["parent"]).add_child(new_object)
        new_object.position = Vector2(current_line["pos_x"], current_line["pos_y"])
        # Now we set the remaining variables.
        for i in current_line.keys():
            if i == "filename" or i == "parent" or i == "pos_x" or i == "pos_y":
                continue
            new_object.set(i, current_line[i])
    save_game.close()
```

```
// Note: This can be called from anywhere inside the tree.  This function is path␣
↪independent.
public void LoadGame()
{
    var saveGame = new File();
    if (!saveGame.FileExists("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.
```

(continues on next page)

(continued from previous page)

```
    // For our example, we will accomplish this by deleting savable objects.
    var saveNodes = GetTree().GetNodesInGroup("Persist");
    foreach (Node saveNode in saveNodes)
        saveNode.QueueFree();

    // Load the file line by line and process that dictionary to restore the object␣
↪it represents
    saveGame.Open("user://savegame.save", (int)File.ModeFlags.Read);

    while (!saveGame.EofReached())
    {
        var currentLine = (Dictionary<object, object>)JSON.Parse(saveGame.GetLine()).
↪Result;
        if (currentLine == null)
            continue;

        // First we need to create the object and add it to the tree and set its␣
↪position.
        var newObjectScene = (PackedScene)ResourceLoader.Load(currentLine["Filename"].
↪ToString());
        var newObject = (Node)newObjectScene.Instance();
        GetNode(currentLine["Parent"].ToString()).AddChild(newObject);
        newObject.Set("Position", new Vector2((float)currentLine["PosX"],␣
↪(float)currentLine["PosY"]));

        // Now we set the remaining variables.
        foreach (KeyValuePair<object, object> entry in currentLine)
        {
            string key = entry.Key.ToString();
            if (key == "Filename" || key == "Parent" || key == "PosX" || key == "PosY
↪")
                continue;
            newObject.Set(key, entry.Value);
        }
    }

    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.

13.3.5 Some notes

We may have glossed over a step, but setting the game state to one fit to start loading data can be 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.

13.4 Encrypting save games

13.4.1 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.

13.4.2 How?

The class *File* can open a file at a location and read/write data (integers, strings and variants). It also supports encryption. To create an encrypted file, a passphrase must be provided, like this:

GDScript

C#

```
var f = File.new()
var err = f.open_encrypted_with_pass("user://savedata.bin", File.WRITE, "mypass")
f.store_var(game_state)
f.close()
```

```
var f = new File();
var err = f.OpenEncryptedWithPass("user://savedata.bin", (int)File.ModeFlags.Write,
↪"mypass");
f.StoreVar(gameState);
f.Close();
```

This will make the file unreadable to users, but will still not prevent them sharing save files. To solve this, use the device unique id or some unique user identifier, for example:

GDScript

C#

```
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()
```

```
var f = new File();
var err = f.OpenEncryptedWithPass("user://savedata.bin", (int)File.ModeFlags.Write,␣
↪OS.GetUniqueId());
```

(continues on next page)

(continued from previous page)

```
f.StoreVar(gameState);
f.Close();
```

Note that `OS.get_unique_id()` only works on iOS and Android.

This is all! Thanks for your cooperation, citizen.

CHAPTER 14

Internationalization

14.1 Internationalizing games

14.1.1 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 common that games going into a more massive market require localization.

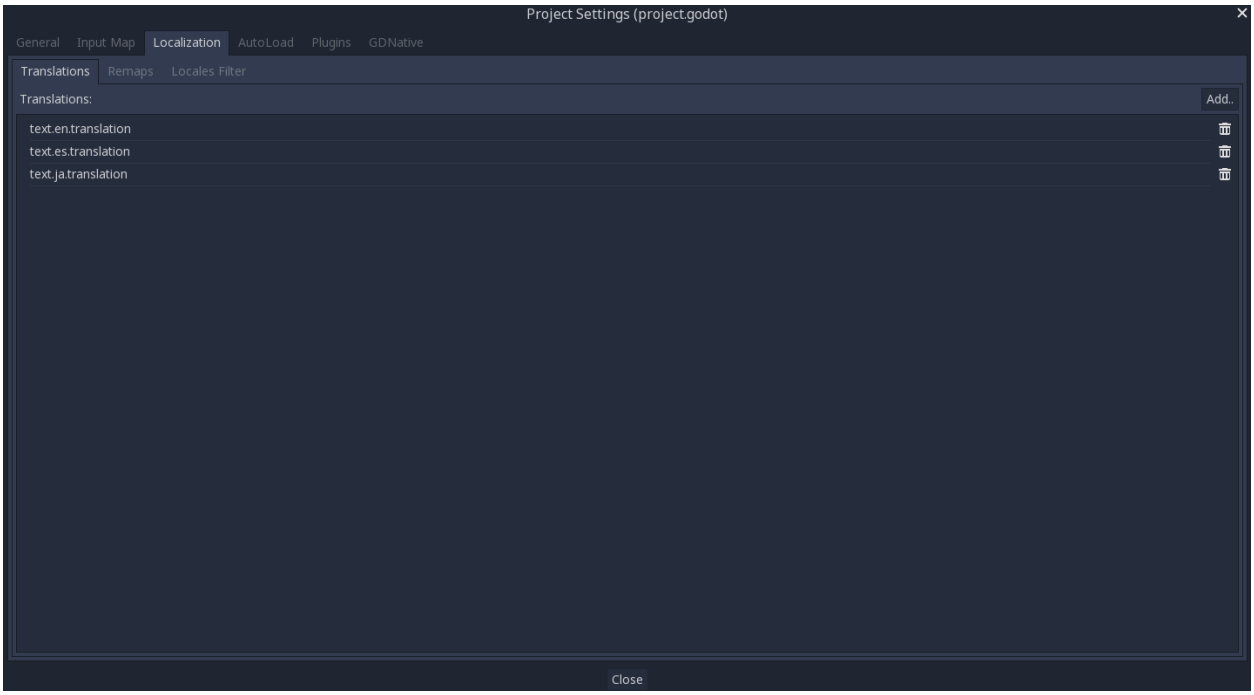
Godot offers many tools to make this process more straightforward, so this tutorial is more like a collection of tips and tricks.

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.

Note: We would be using the offical demo as an example, you can download it in the AssetLib: https://godotengine.org/asset-library/asset/134 or find it in the demo_projects/gui/translation

14.1.2 Configuring the imported translation

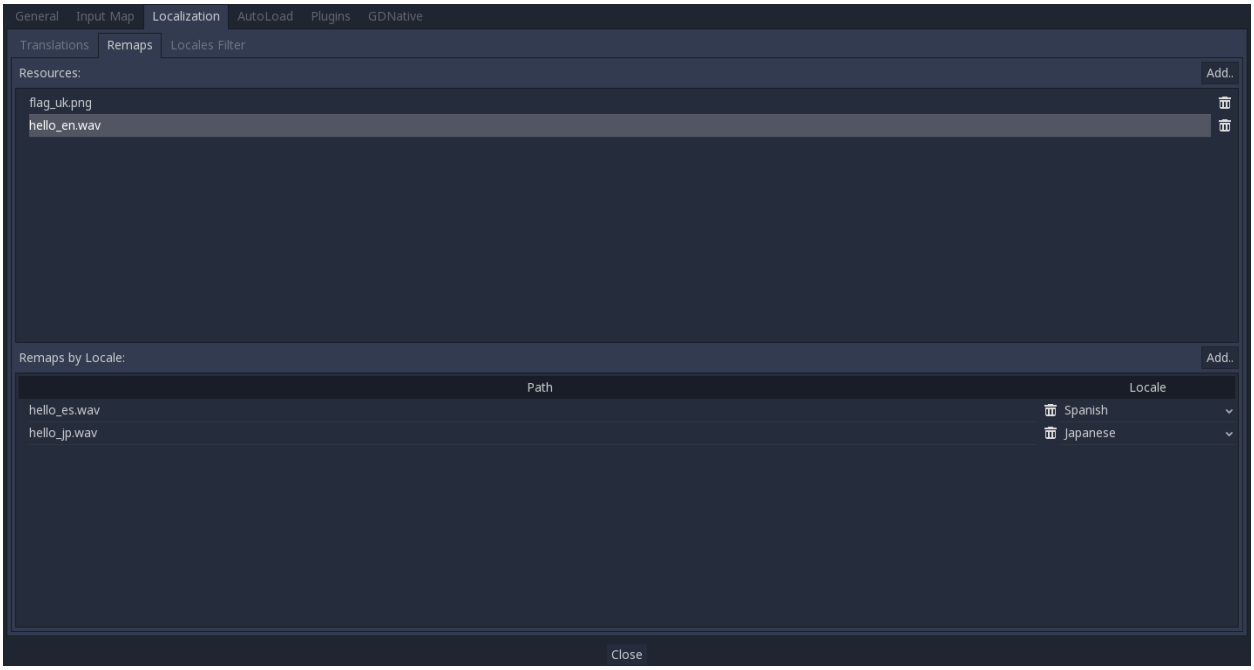
The translations can get updated and re-imported when they change, but they still have to be added to the project. This is done in Scene > Project Settings > Localization:



This dialog allows to add or remove translations project-wide.

14.1.3 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.

14.1.4 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.

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)))
```

14.1.5 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 useful, as well as the multiple options in *Label* for text wrapping.

14.1.6 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.

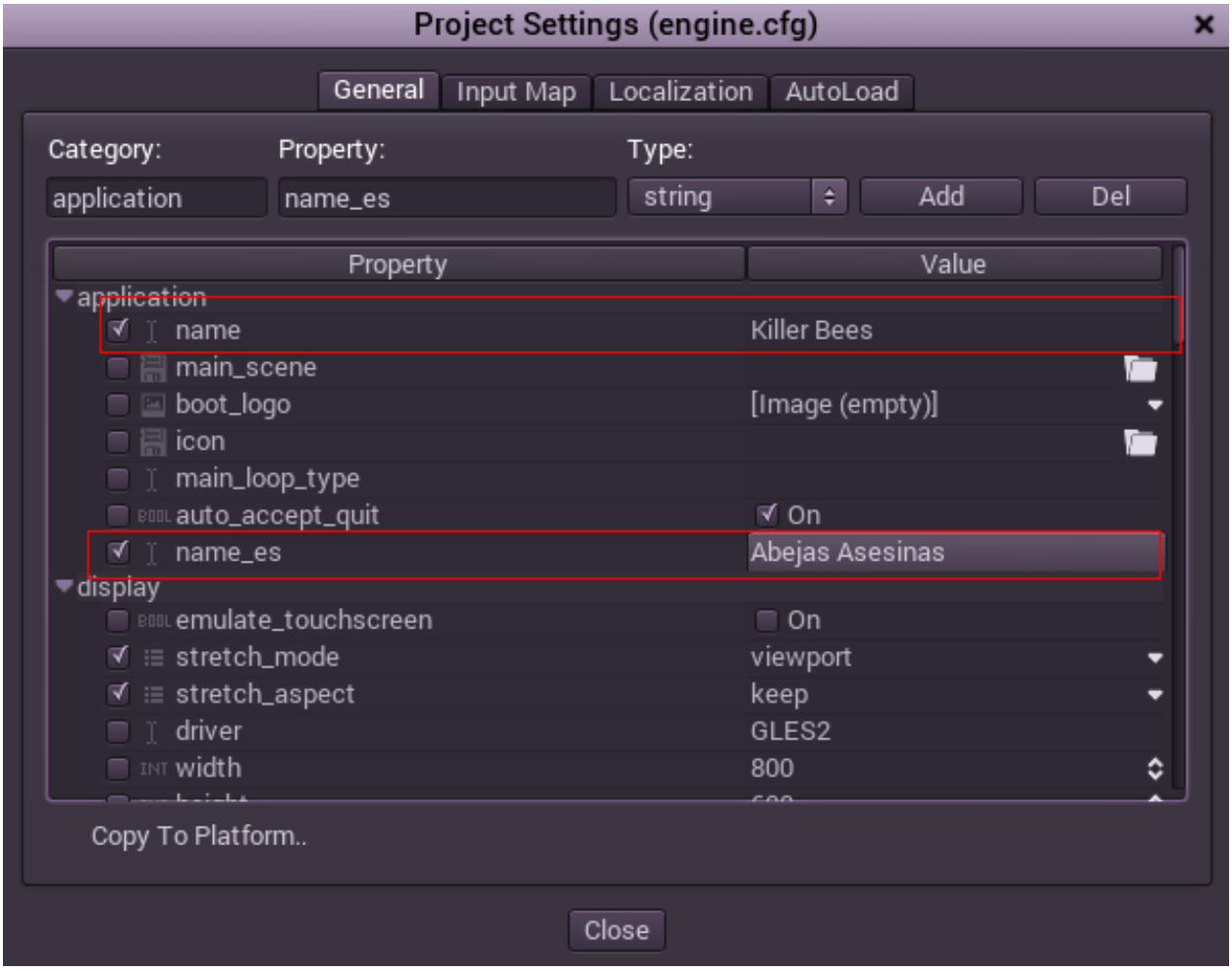
14.1.7 Command line

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 -l fr
```

14.1.8 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*.

14.2 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 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 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 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)
fy_DE	Western Frisian (Germany)
fy_NL	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 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 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	Panjabi (India)
pap	Papiamento
pap_AN	Papiamento (Netherlands Antilles)
pap_AW	Papiamento (Aruba)

Continued on next page

Table 1 – continued from previous page

Locale	Language and Variant
pap_CW	Papiamento (Curaçao)
pa_PK	Panjabi (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 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 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)

CHAPTER 15

GUI

15.1 GUI skinning

15.1.1 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.

15.1.2 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 (especially 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.

15.1.3 Theme options

Each kind of option in a theme can be:

- **An 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:

GDScript

C#

```
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)
```

```
var t = new Theme();
t.SetColor("fontColor", "Label", new Color(1.0f, 1.0f, 1.0f));

var l = new Label();
l.SetTheme(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()*:

GDScript

C#

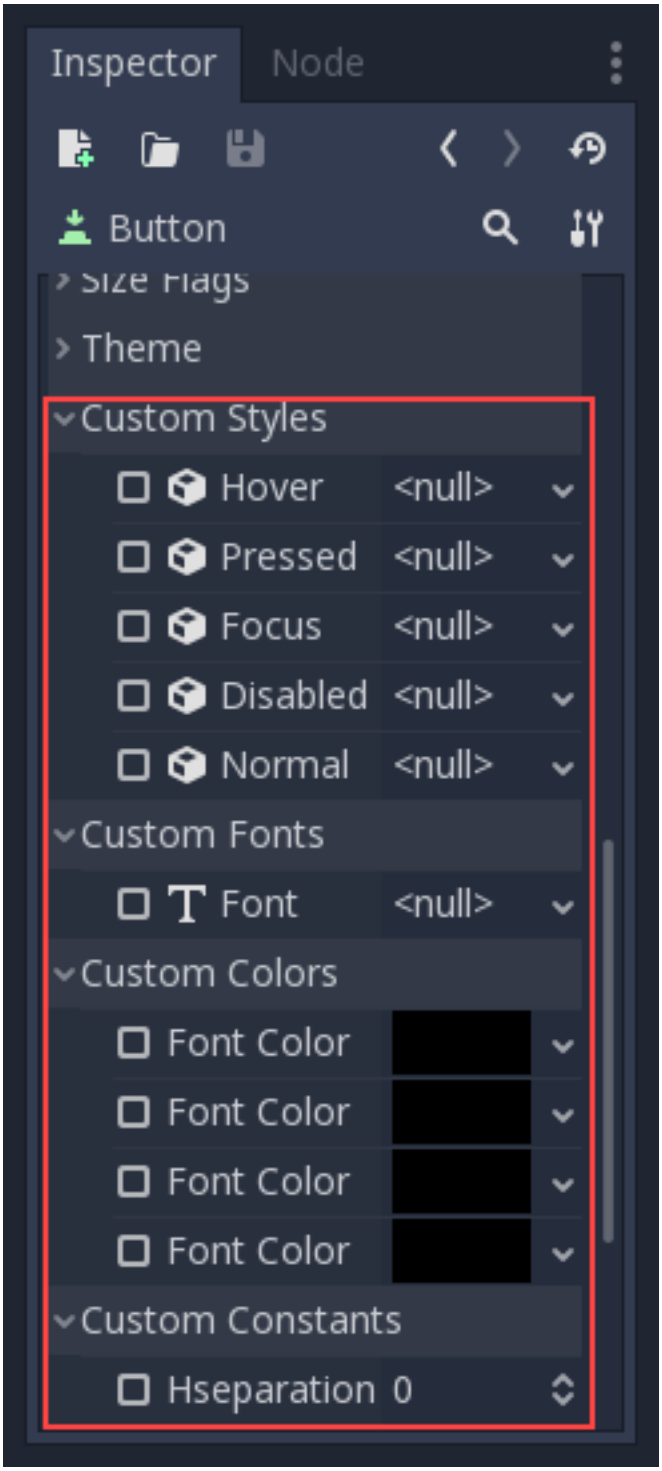
```
var l = Label.new()
l.add_color_override("font_color", Color(1.0, 1.0, 1.0))
```

```
var l = new Label();
l.AddColorOverride("fontColor", new Color(1.0f, 1.0f, 1.0f));
```

In the inline help of Godot (in the script tab) you can check which theme options are overrideable, or check the *Control* class reference.

15.1.4 Customizing a control

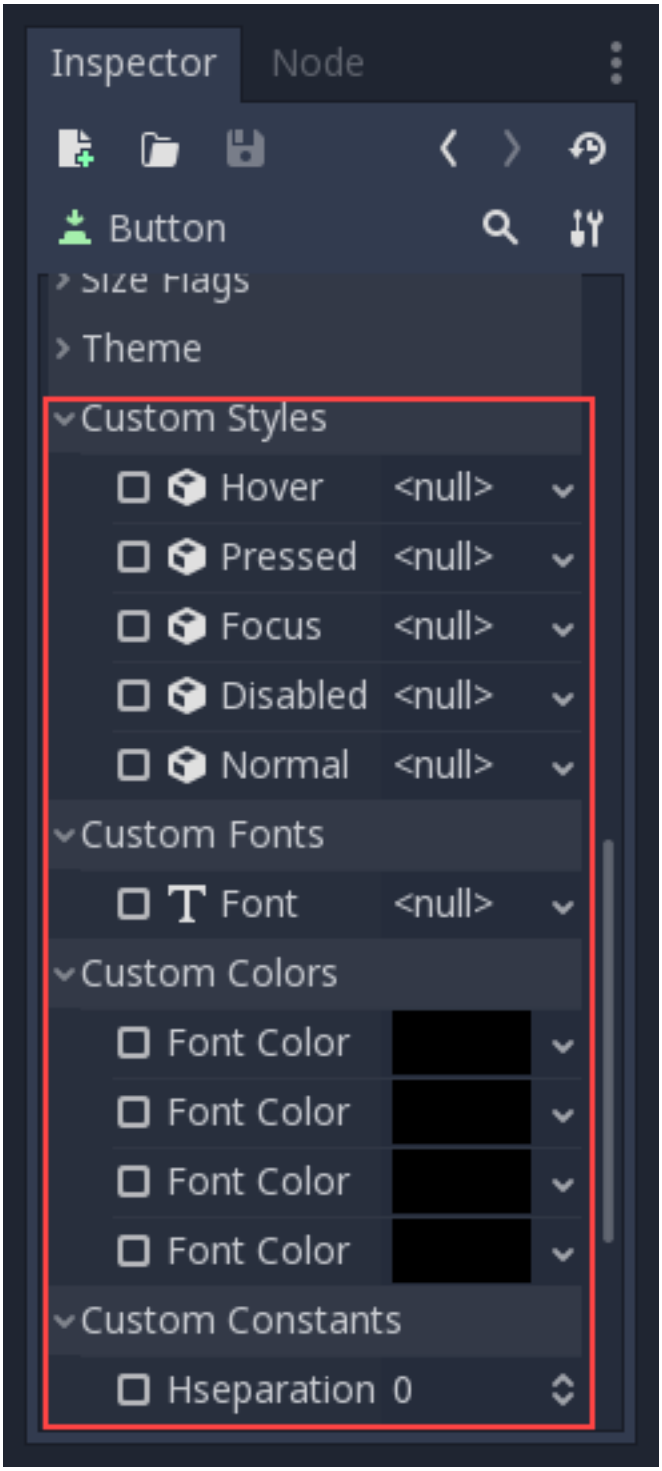
If only a few controls need to be skinned, it is often not necessary to create a new theme. Controls offer their theme options as special kinds of properties. If checked, overriding will take place:



As can be see 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.

15.1.5 Creating a theme

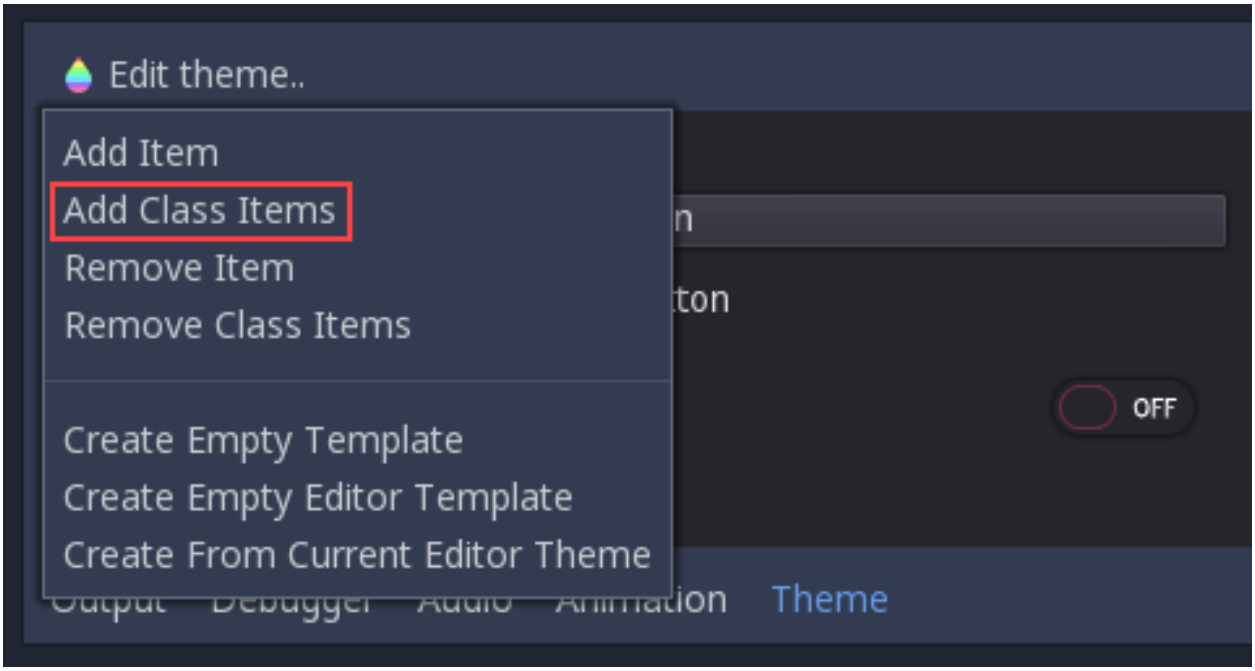
The simplest way to create a theme is to edit a theme resource. Create a Theme from the resource menu, the editor will appear immediately. Following to this, save it (to, as example, mytheme.thm):



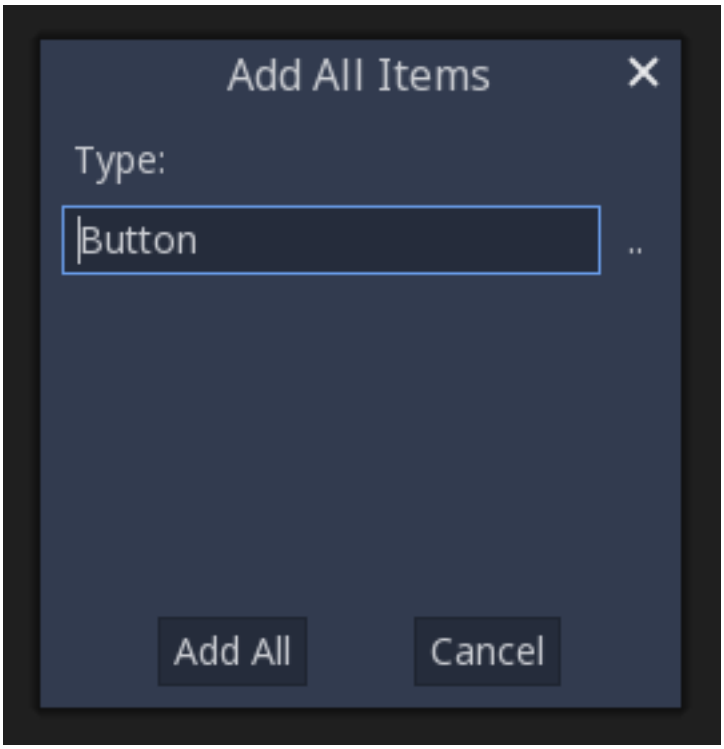
This will create an empty theme that can later be loaded and assigned to controls.

15.1.6 Example: theming a button

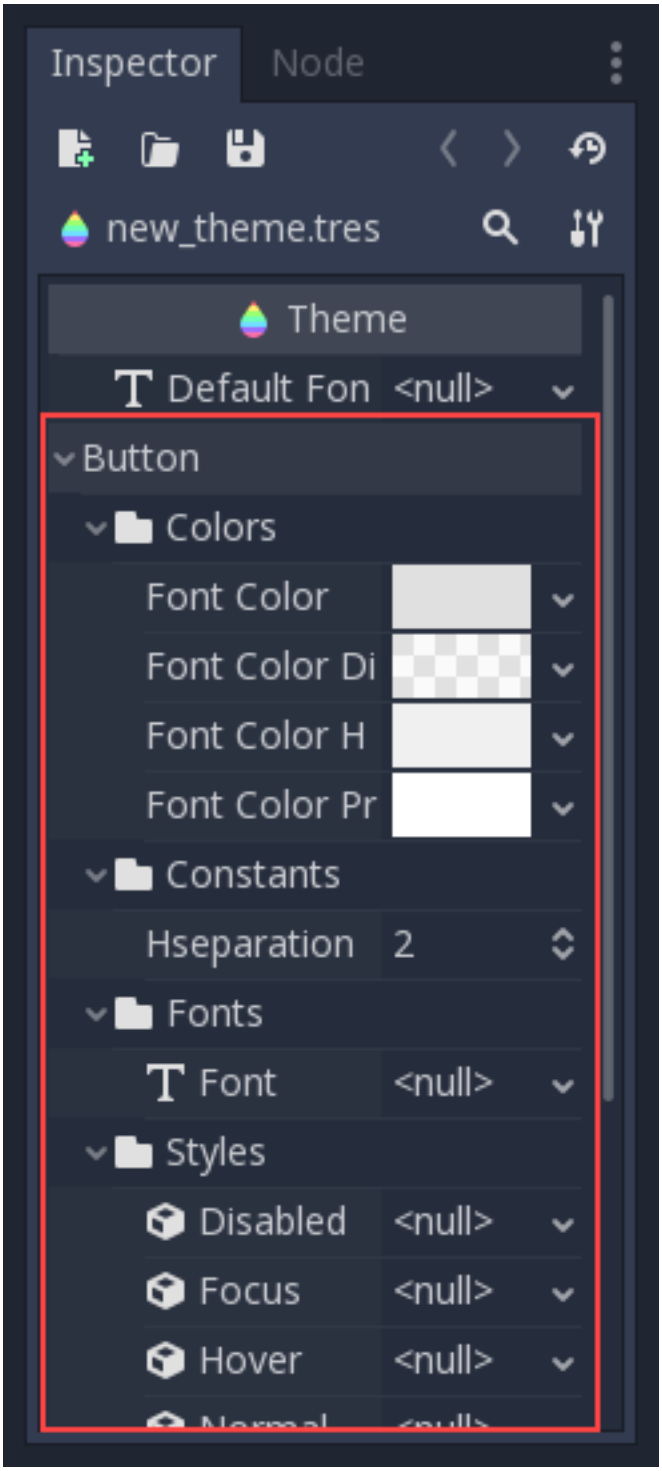
Take some assets (`skin_assets.zip`), go to the "theme" menu and select "Add Class Item":



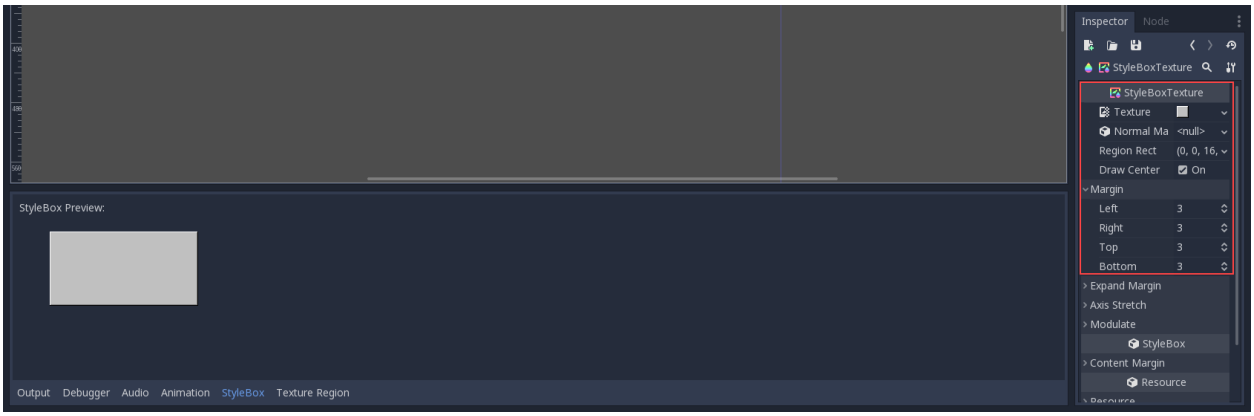
A menu will appear prompting the type of control to create. Select "Button":



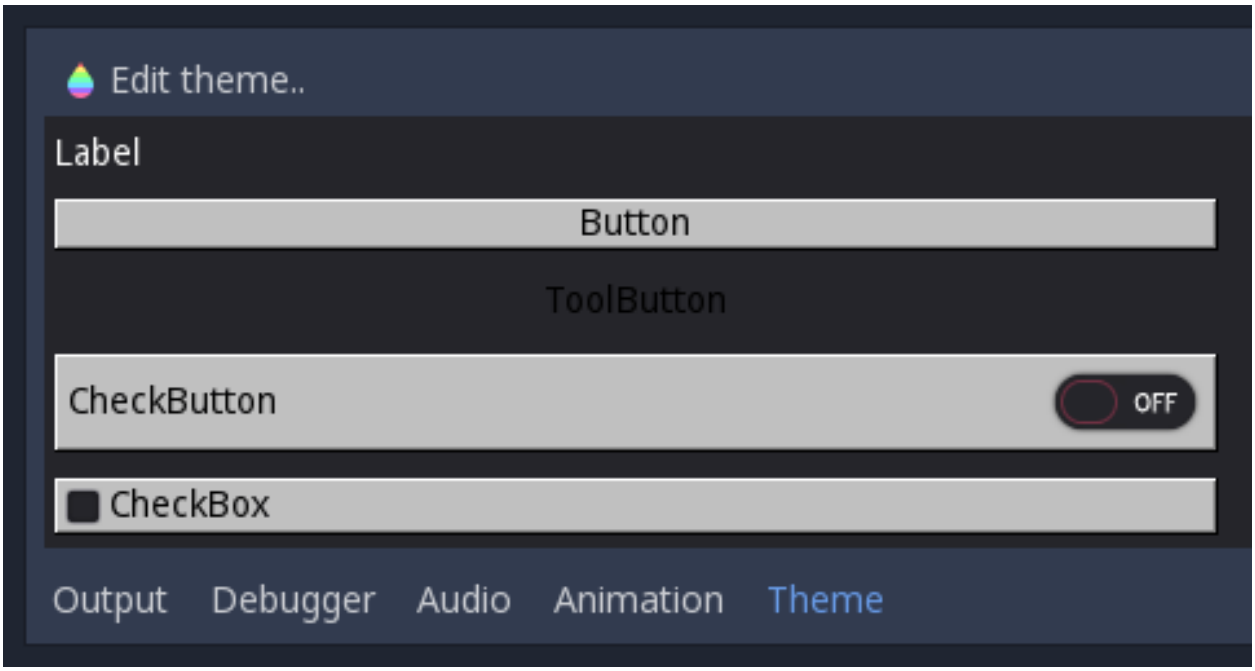
Immediately, all button theme options will appear in the property editor, where they can be edited:



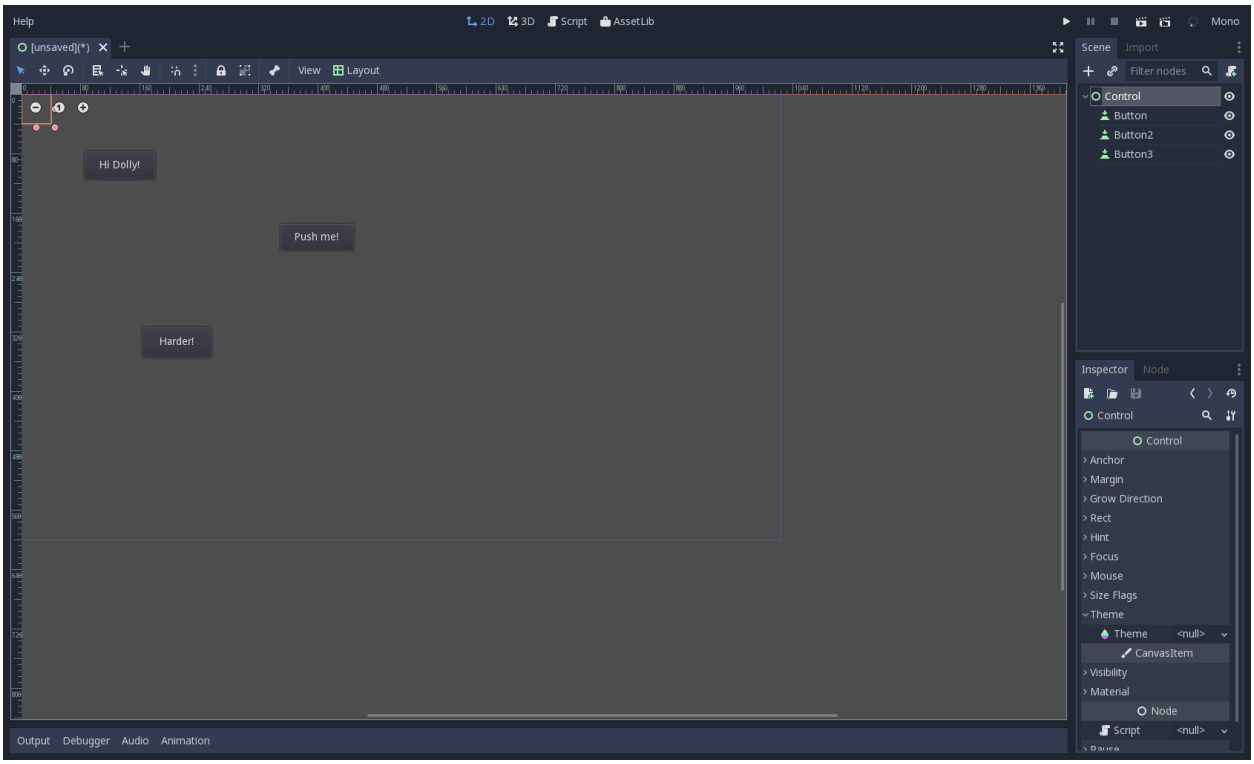
Select the “normal” stylebox and create a new “StyleBoxTexture”, then edit it. A texture stylebox basically contains a texture and the size of the margins that will not stretch when the texture is stretched. This is called “3x3” stretching:



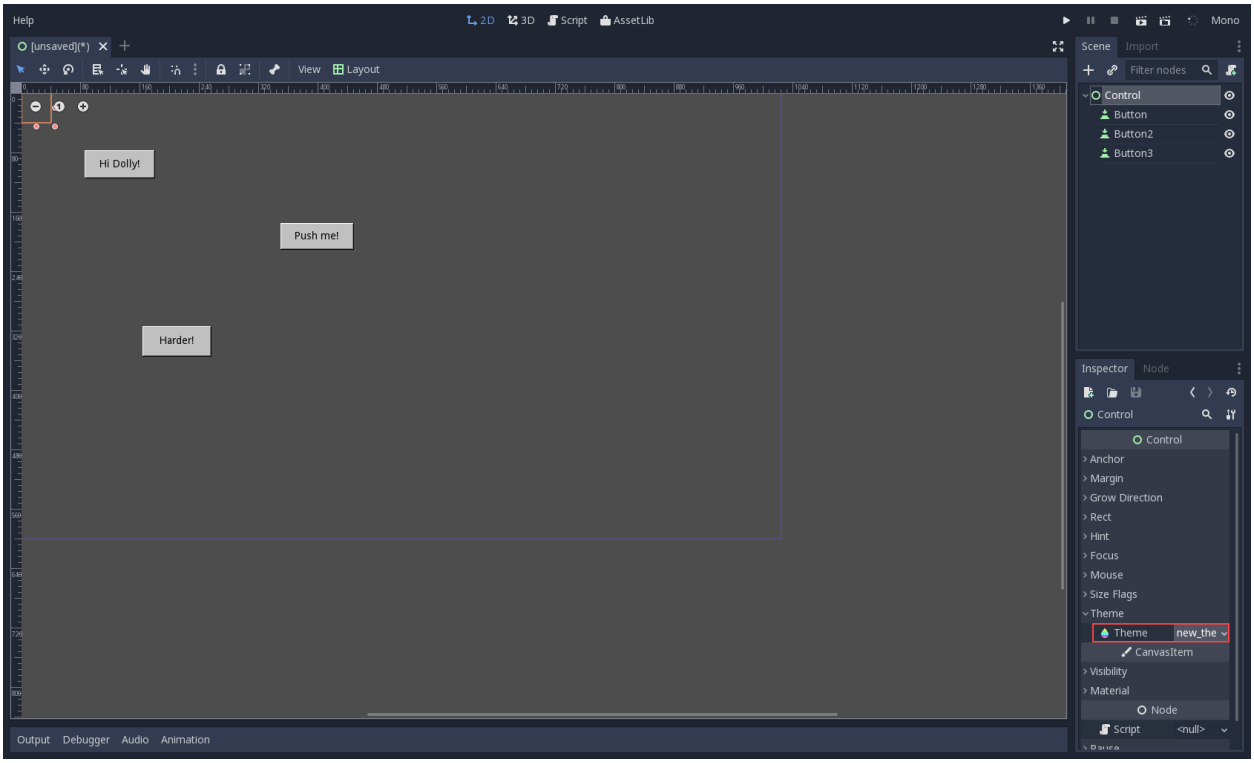
Repeat the steps and add the other assets. There is no hover or disabled image in the example files, so use the same stylebox as in normal. Set the supplied font as the button font and change the font color to black. Soon, your button will look different and retro:



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!

15.2 Custom GUI controls

15.2.1 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.

15.2.2 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 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

GDScript

C#

```
func _draw():
    if has_focus():
        draw_selected()
    else:
        draw_normal()
```

```
public override void _Draw()
{
    if(HasFocus())
    {
        DrawSelected()
    }
    else
    {
        DrawNormal();
    }
}
```

15.2.3 Sizing

As mentioned before, size is 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:

GDScript

C#

```
func get_minimum_size():
    return Vector2(30, 30)
```

```
public override Vector2 _GetMinimumSize()
{
    return new Vector2(20, 20);
}
```

Or alternatively, set it via function:

GDScript

C#

```
func _ready():
    set_custom_minimum_size(Vector2(30, 30))
```

```
public override void _Ready()
{
    SetCustomMinimumSize(new Vector2(20, 20));
}
```

15.2.4 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.

GDScript

C#

```
extends Control

func _gui_input(event):
   if event is InputEventMouseButton and event.button_index == BUTTON_LEFT and event.
↪pressed:
       print("Left mouse button was pressed!")
```

```
public override void _GuiInput(InputEvent @event)
{
    var mouseButtonEvent = @event as InputEventMouseButton;
    if (mouseButtonEvent != null)
    {
        if (mouseButtonEvent.ButtonIndex == (int)ButtonList.Left && mouseButtonEvent.
↪Pressed)
        {
            GD.Print("Left mouse button was pressed!");
        }
    }
}

// or alternatively when using C# 7 or greater we can use pattern matching
public override void _GuiInput(InputEvent @event)
{
    if (@event is InputEventMouseButton mbe && mbe.ButtonIndex == (int)ButtonList.
↪Left && mbe.Pressed)
        {
            GD.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:

GDScript

C#

```
func _notification(what):
    match what:
        NOTIFICATION_MOUSE_ENTER:
            pass # mouse entered the area of this control
        NOTIFICATION_MOUSE_EXIT:
            pass # mouse exited the area of this control
        NOTIFICATION_FOCUS_ENTER:
            pass # control gained focus
        NOTIFICATION_FOCUS_EXIT:
            pass # control lost focus
        NOTIFICATION_THEME_CHANGED:
            pass # theme used to draw the control changed
            # update and redraw is recommended if using a theme
        NOTIFICATION_VISIBILITY_CHANGED:
```

(continues on next page)

(continued from previous page)

```
            pass # control became visible/invisible
            # check new status with is_visible()
        NOTIFICATION_RESIZED:
            pass # control changed size, check new size
            # with get_size()
        NOTIFICATION_MODAL_CLOSED):
            pass # for modal popups, notification
            # that the popup was closed
```

```
public override void _Notification(int what)
{
    switch(what)
    {
        case NotificationMouseEnter:
            // mouse entered the area of this control
            break;

        case NotificationMouseExit:
            // mouse exited the area of this control
            break;

        case NotificationFocusEnter:
            // control gained focus
            break;

        case NotificationFocusExit:
            // control lost focus
            break;

        case NotificationThemeChanged:
            // theme used to draw the control changed
            // update and redraw is recommended if using a theme
            break;

        case NotificationVisibilityChanged:
            // control became visible/invisible
            // check new status with is_visible()
            break;

        case NotificationResized:
            // control changed size, check new size with get_size()
            break;

        case NotificationModalClose:
            // for modal popups, notification that the popup was closed
            break;
    }
}
```

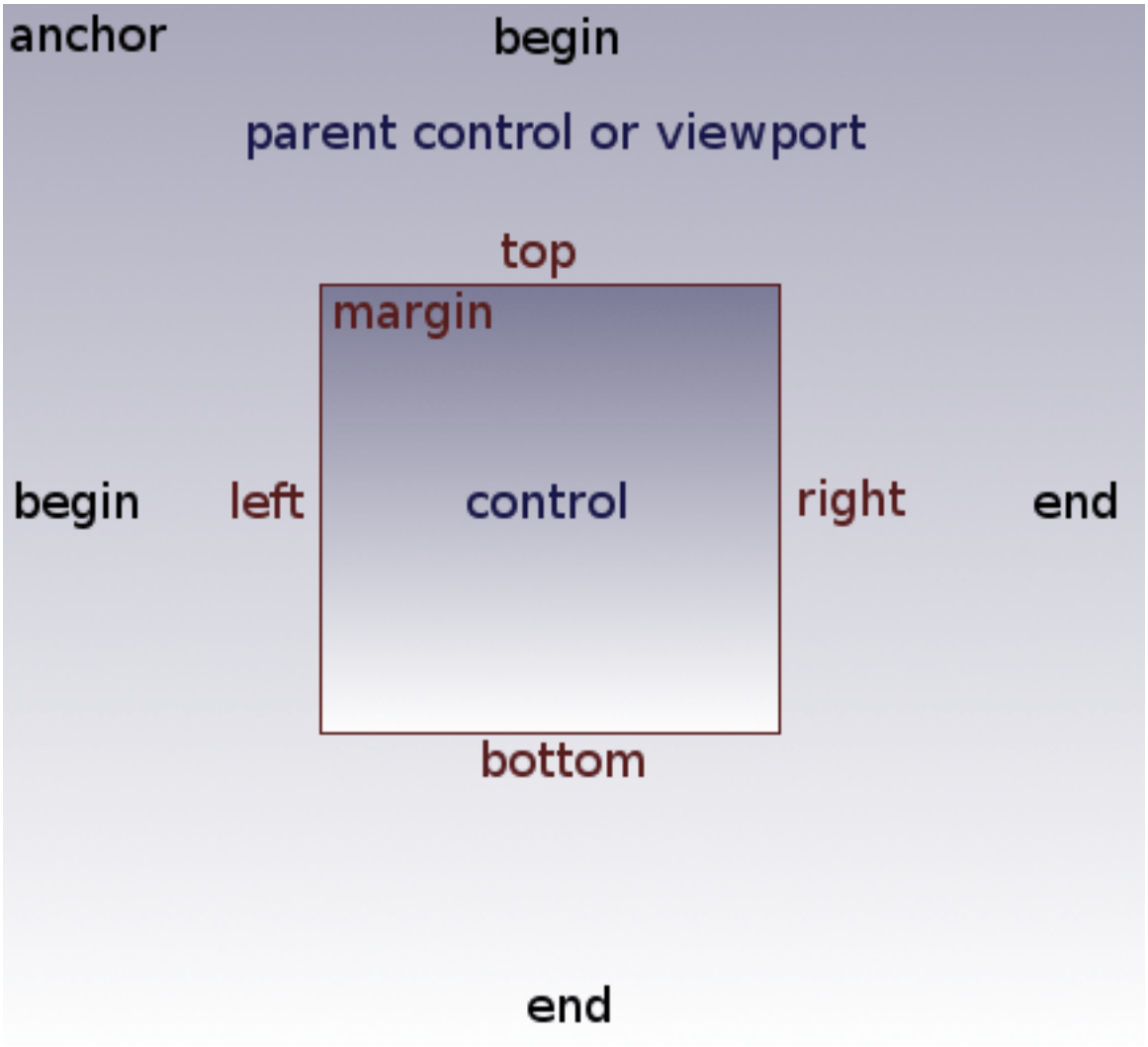
15.3 Size and anchors

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.

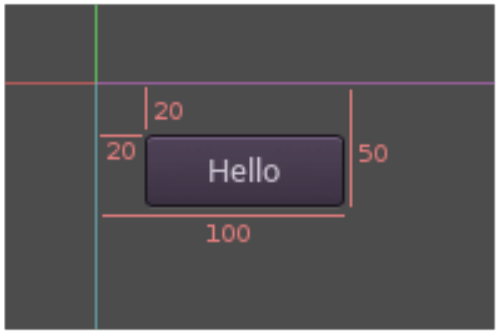
Only TVs nowadays have a standard resolution and aspect ratio. Everything else, from computer monitors to tablets,

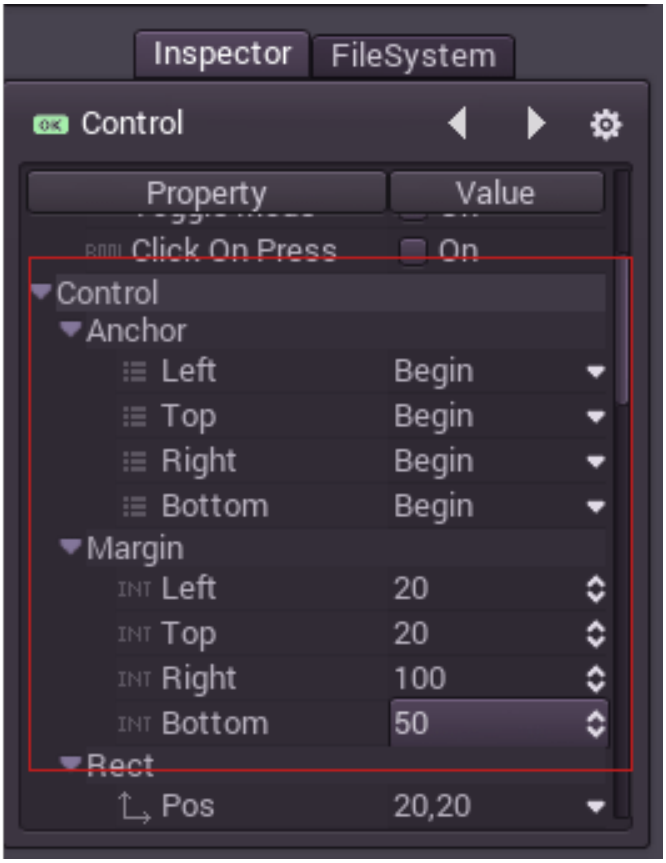
portable consoles and mobile phones have different resolutions and aspect ratios.

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.

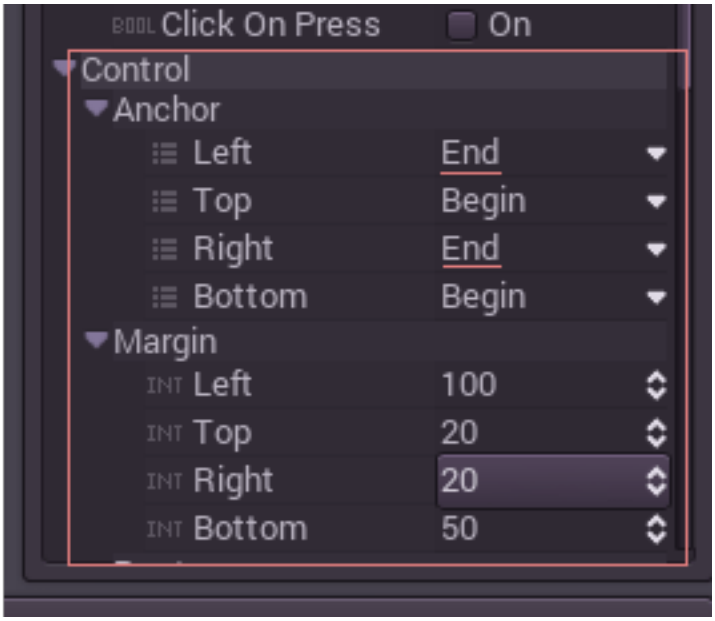


This is done by editing the *margin* properties of controls. Each control has four margins: left, right, bottom and top. By default all of them represent a distance in pixels relative to the top-left corner of the parent control or (in case there is no parent control) the viewport.

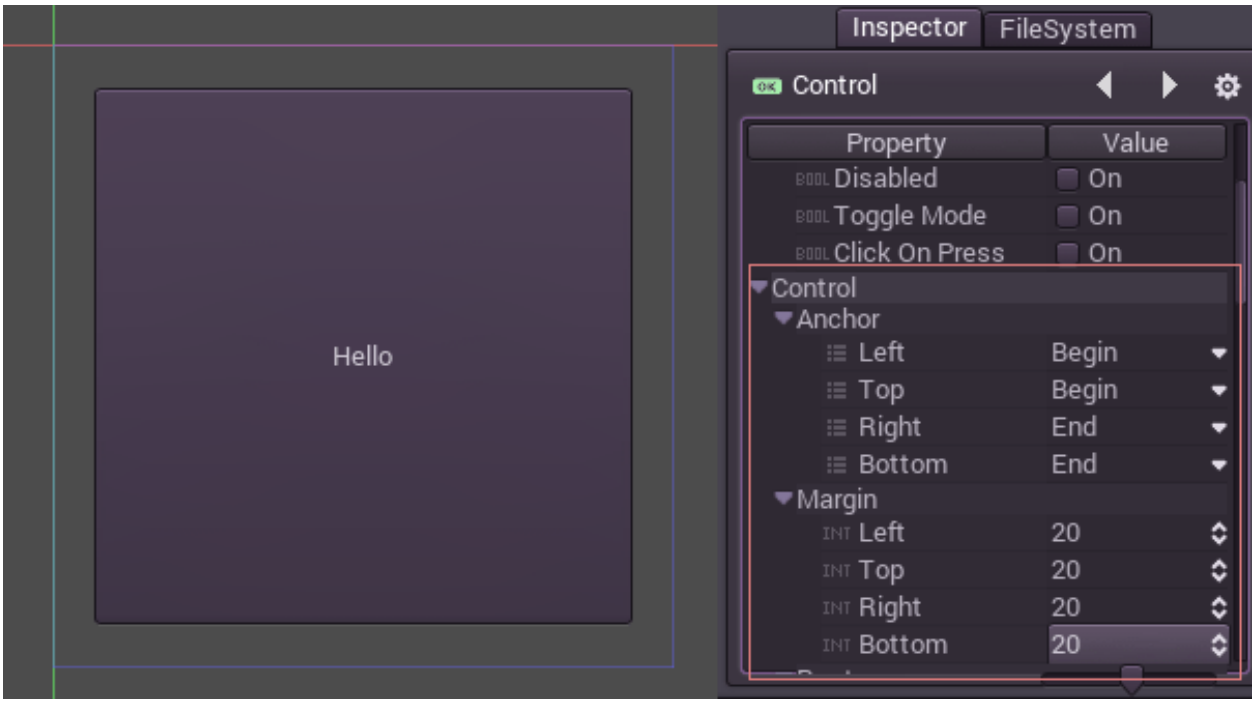




When horizontal (left,right) and/or vertical (top,bottom) anchors are changed to END, the margin values become relative to the bottom-right corner of the parent control or viewport.



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:



15.4 BBCode in RichTextLabel

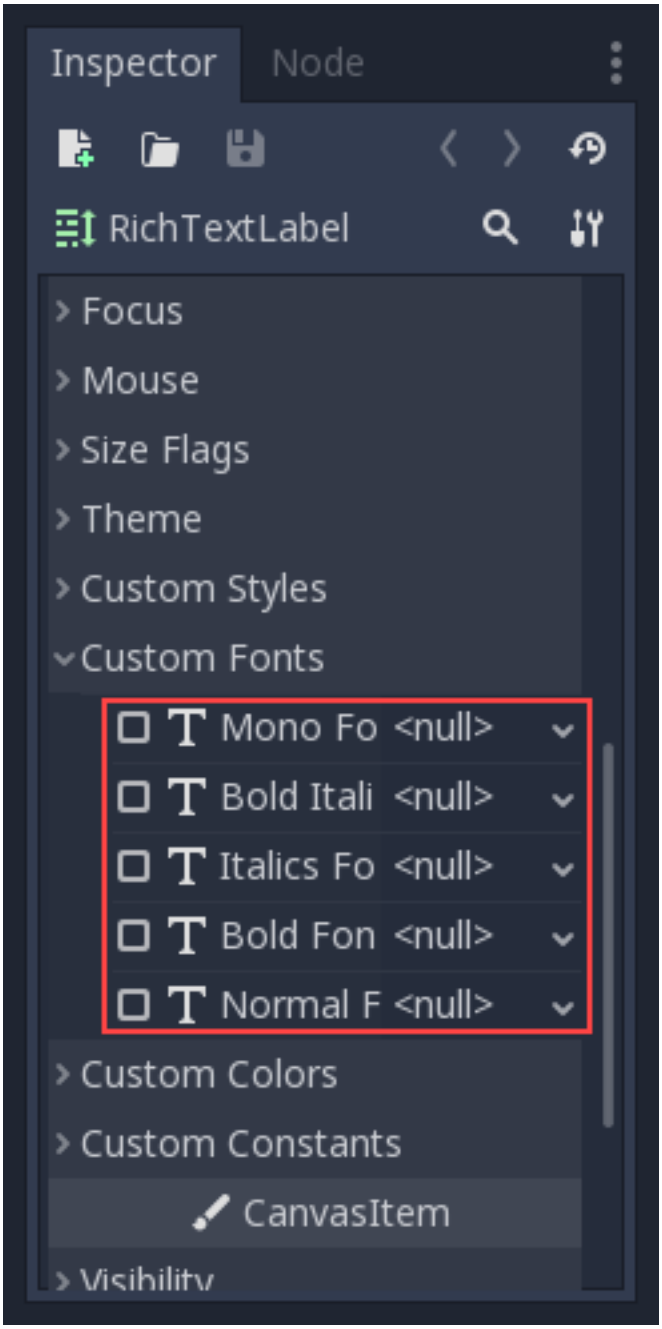
15.4.1 Introduction

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*.

15.4.2 Setting up

For RichTextLabel to work properly, it must be set up. This means loading the intended fonts in the relevant properties:


Inspector
Node
RichTextLabel
Focus
Mouse
Size Flags
Theme
Custom Styles
Custom Fonts
Mono Fo <null>
Bold Itali <null>
Italics Fo <null>
Bold Fon <null>
Normal F <null>
Custom Colors
Custom Constants
CanvasItem
Visibility

15.4.3 Reference

Command	Tag	Description
bold	`[b]{text}[/b]`	Makes {text} bold.
italics	`[i]{text}[/i]`	Makes {text} italics.
underline	`[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 16

Viewports

16.1 Viewports

16.1.1 Introduction

Godot has a small but useful feature called viewports. Viewports are, as the 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*.

16.1.2 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.

16.1.3 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!

16.1.4 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()
```

16.1.5 Scale & stretching

Viewports have a "rect" property. X and Y are often not used (only the root viewport 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) # enable stretch for custom size
```

The root viewport uses this for the stretch options in the project settings.

16.1.6 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 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.

16.1.7 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:

```
# Retrieve the captured Image using get_data()
var img = get_viewport().get_texture().get_data()
# Also remember to flip the texture (because it's flipped)
img.flip_y()
# Convert Image to ImageTexture
var tex = ImageTexture.new()
tex.create_from_image(img)
# Set Sprite Texture
$sprite.texture = tex
```

But if you use this in _ready() or from the first frame of the viewport's initialization you will get an empty texture cause there is nothing to get as texture. You can deal with it using (for example):

```
# Let two frames pass to make sure the screen can be captured
yield(get_tree(), "idle_frame")
yield(get_tree(), "idle_frame")
# You can get the image after this
```

If the returned image is empty, capture still didn't happen, wait a little more, as this API is asynchronous.

16.1.8 Sub-viewport

If the viewport is a child of a *ViewportContainer*, it will become active and display anything it has inside. The layout is something like this:

- ViewportContainer
  - Viewport

The viewport will cover the area of its parent control completely, if stretch is set to true in Viewport Container. But you will have to setup the Viewport Size to get the the appropriate part of the Viewport. And Viewport Container can not be smaller than the size of the Viewport.

16.1.9 Render target

To set as a render target, 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 method remains the same. This can be requested via code using (for example):

```
#This gets us the render_target texture
var rtt = viewport.get_texture()
sprite.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.

TODO: Review the doc, change outdated and add more images.

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

16.2 Multiple resolutions

16.2.1 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.

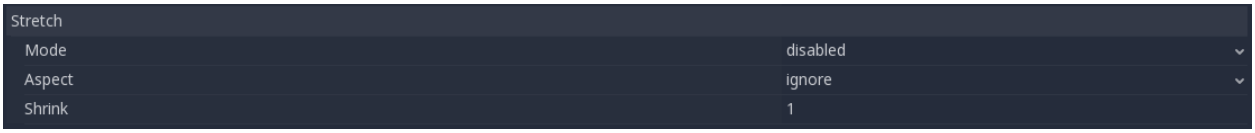
16.2.2 Resizing

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.

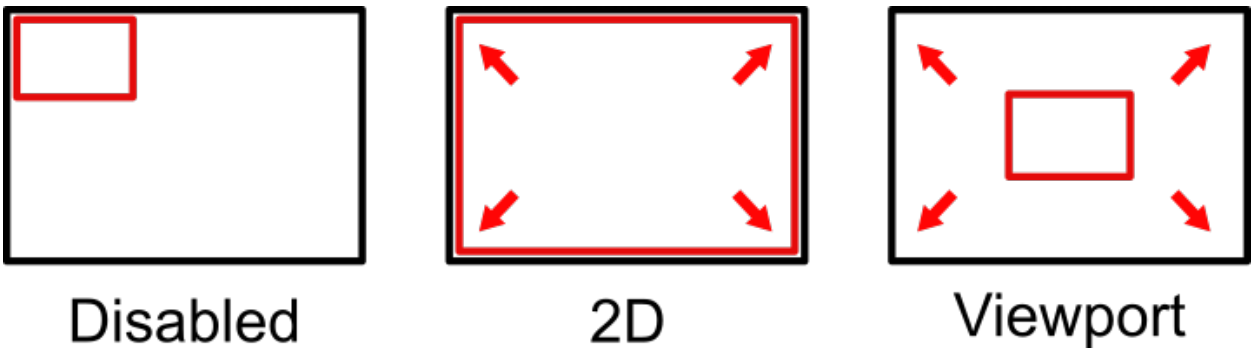
16.2.3 Stretch settings

Stretch settings are located in the project settings, it's just a bunch of configuration variables that provide several options:



16.2.4 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 for the sake of rendering to a lower resolution for improving performance.



16.2.5 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).
- **Expand**: Keep aspect ratio when stretching the screen, but keep neither width nor height. Depending on the screen aspect ratio, the viewport will either report more horizontal resolution (if the screen is wider than the original resolution) or more vertical resolution (if the screen is taller than the original resolution).

CHAPTER 17

Shading

17.1 Shading language

17.1.1 Introduction

Godot uses a shading language similar to GLSL ES 3.0. Most datatypes and functions are supported, and the few remaining ones will likely be added over time.

Unlike the shader language in Godot 2.x, this implementation is much closer to the original.

17.1.2 Shader Types

Instead of supplying a general purpose configuration, Godot Shading Language must specify what a 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.

17.1.3 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;
```

17.1.4 Data types

Most GLSL ES 3.0 datatypes are supported:

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 between scalars and vectors of the same size but different type is not allowed. 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 m[row][column] indexing syntax to access each scalar, or m[idx] for access a vector by row index. For example, for accessing y position of object in mat4 you must use m[3][1] syntax.

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);
```

Construction of matrix types requires pass vectors of same dimension as matrix. You could also build a diagonal matrix using matx(float) syntax. So the mat4(1.0) is an identity matrix.

```
mat2 m2 = mat2(vec2(1.0, 0.0), vec2(0.0, 1.0));
mat3 m3 = mat3(vec3(1.0, 0.0, 0.0), vec3(0.0, 1.0, 0.0), vec3(0.0, 0.0, 1.0));
mat4 identity = mat4(1.0);
```

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 buggy so to stay out of trouble make simple shaders without specifying precision unless you *really* need to.

17.1.5 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	**||**

17.1.6 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!

17.1.7 Discarding

Fragment and light functions can use the **discard** keyword. If used, the fragment is discarded and nothing is written.

17.1.8 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;
}
```

17.1.9 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() {
    DIFFUSE_LIGHT = 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);
uniform vec4 other_color : hint_color = vec4(1.0);
```

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);
uniform vec4 some_color : hint_color = vec4(1.0);
```

17.1.10 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
vec_type **radians** (vec_type)	Convert degrees to radians
vec_type **degrees** (vec_type)	Convert radians to degrees
vec_type **sin** (vec_type)	Sine
vec_type **cos** (vec_type)	Cosine
vec_type **tan** (vec_type)	Tangent
vec_type **asin** (vec_type)	Arc-Sine
vec_type **acos** (vec_type)	Arc-Cosine
vec_type **atan** (vec_type)	Arc-Tangent
vec_type **atan** (vec_type x, vec_type y)	Arc-Tangent to convert vector to angle
vec_type **sinh** (vec_type)	Hyperbolic-Sine
vec_type **cosh** (vec_type)	Hyperbolic-Cosine
vec_type **tanh** (vec_type)	Hyperbolic-Tangent
vec_type **asinh** (vec_type)	Inverse-Hyperbolic-Sine
vec_type **acosh** (vec_type)	Inverse-Hyperbolic-Cosine
vec_type **atanh** (vec_type)	Inverse-Hyperbolic-Tangent
vec_type **pow** (vec_type, vec_type)	Power
vec_type **exp** (vec_type)	Base-e Exponential
vec_type **exp2** (vec_type)	Base-2 Exponential
vec_type **log** (vec_type)	Natural Logarithm
vec_type **log2** (vec_type)	Base-2 Logarithm
vec_type **sqrt** (vec_type)	Square Root
vec_type **inversesqrt** (vec_type)	Inverse Square Root
vec_type **abs** (vec_type)	Absolute
vec_int_type **abs** (vec_int_type)	Absolute
vec_type **sign** (vec_type)	Sign
vec_int_type **sign** (vec_int_type)	Sign
vec_type **floor** (vec_type)	Floor
vec_type **round** (vec_type)	Round
vec_type **roundEven** (vec_type)	Round nearest even
vec_type **trunc** (vec_type)	Truncation
vec_type **ceil** (vec_type)	Ceiling
vec_type **fract** (vec_type)	Fractional
vec_type **mod** (vec_type, vec_type)	Remainder
vec_type **mod** (vec_type, float)	Remainder
vec_type **modf** (vec_type x, out vec_type i)	Fractional of x, with i has integer part
vec_scalar_type **min** (vec_scalar_type a, vec_scalar_type b)	Minimum
vec_scalar_type **max** (vec_scalar_type a, vec_scalar_type b)	Maximum
vec_scalar_type **clamp** (vec_scalar_type value, vec_scalar_type min, vec_scalar_type max)	Clamp to Min-Max
vec_type **mix** (vec_type a, vec_type b, float c)	Linear Interpolate (Scalar Coef.)
vec_type **mix** (vec_type a, vec_type b, vec_type c)	Linear Interpolate (Vector Coef.)
vec_type **mix** (vec_type a, vec_type b, bool c)	Linear Interpolate (Bool Selection)
vec_type **mix** (vec_type a, vec_type b, vec_bool_type c)	Linear Interpolate (Bool-Vector Select
vec_type **step** (vec_type a, vec_type b)	‘ b[i] < a[i] ? 0.0 : 1.0 ‘
vec_type **step** (float a, vec_type b)	‘ b[i] < a ? 0.0 : 1.0 ‘
vec_type **smoothstep** (vec_type a, vec_type b, vec_type c)	Hermite Interpolate
vec_type **smoothstep** (float a, float b, vec_type c)	Hermite Interpolate
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 conversio

Continued on n

Table 1 – continued from previous page

Function	Description
vec_type **intBitsToFloat** (vec_int_type)	Int->Float bit copying, no conversion
vec_type **uintBitsToFloat** (vec_uint_type)	UInt->Float bit copying, no conversio
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 I, vec3 N)	Reflect
vec3 **refract** (vec3 I, vec3 N, float eta)	Refract
vec_type **faceforward** (vec_type N, vec_type I, vec_type Nref)	If dot(Nref, I) < 0 return N, otherwise
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, vec_scalar_type)	Bool vector cmp on < int/uint/float ve
vec_bool_type **greaterThan** (vec_scalar_type, vec_scalar_type)	Bool vector cmp on > int/uint/float ve
vec_bool_type **lessThanEqual** (vec_scalar_type, vec_scalar_type)	Bool vector cmp on <= int/uint/float v
vec_bool_type **greaterThanEqual** (vec_scalar_type, vec_scalar_type)	Bool vector cmp on >= int/uint/float v
vec_bool_type **equal** (vec_scalar_type, vec_scalar_type)	Bool vector cmp on == int/uint/float v
vec_bool_type **notEqual** (vec_scalar_type, vec_scalar_type)	Bool vector cmp on != int/uint/float ve
bool **any** (vec_bool_type)	Any component is true
bool **all** (vec_bool_type)	All components are true
bool **not** (vec_bool_type)	No components are true
ivec2 **textureSize** (sampler2D_type s, int lod)	Get the size of a texture
ivec2 **textureSize** (samplerCube s, int lod)	Get the size of a cubemap
vec4_type **texture** (sampler2D_type s, vec2 uv [, float bias])	Perform a 2D texture read
vec4_type **texture** (samplerCube s, vec3 uv [, float bias])	Perform a Cube texture read
vec4_type **textureProj** (sampler2D_type s, vec3 uv [, float bias])	Perform a texture read with projection
vec4_type **textureProj** (sampler2D_type s, vec4 uv [, float bias])	Perform a texture read with projection
vec4_type **textureLod** (sampler2D_type s, vec2 uv, float lod)	Perform a 2D texture read at custom m
vec4_type **textureLod** (samplerCube s, vec3 uv, float lod)	Perform a Cube texture read at custom
vec4_type **textureProjLod** (sampler2D_type s, vec3 uv, float lod)	Perform a texture read with projection
vec4_type **textureProjLod** (sampler2D_type s, vec4 uv, float lod)	Perform a texture read with projection
vec4_type **texelFetch** (sampler2D_type s, ivec2 uv, int lod)	Fetch a single texel using integer coor
vec_type **dFdx** (vec_type)	Derivative in x using local differencin
vec_type **dFdy** (vec_type)	Derivative in y using local differencin
vec_type **fwidth** (vec_type)	Sum of absolute derivative in x and y

17.1.11 Shader Types In-Depth

Spatial

Accepted render modes and built-ins for **shader_type spatial;**.

Render Modes

Render Mode	Description
blend_mix	Mix blend mode (alpha is transparency), default.
blend_add	Additive blend mode.
blend_sub	Substractive blend mode.
blend_mul	Multiplicative blend mode.
depth_draw_opaque	Only draw depth for opaque geometry (not transparent).
depth_draw_always	Always draw depth (opaque and transparent).
depth_draw_never	Never draw depth.
depth_draw_alpha_prepass	Do opaque depth pre-pass for transparent geometry.
depth_test_disable	Disable depth testing.
cull_front	Cull front-faces.
cull_back	Cull back-faces (default).
cull_disabled	Culling disabled (double sided).
unshaded	Result is just albedo. No lighting/shading happens in material.
diffuse_lambert	Lambert shading for diffuse (default).
diffuse_lambert_wrap	Lambert wrapping (roughness dependent) for diffuse.
diffuse_oren_nayar	Oren Nayar for diffuse.
diffuse_burley	Burley (Disney PBS) for diffuse.
diffuse_toon	Toon shading for diffuse.
specular_schlick_ggx	Schlick-GGX for specular (default).
specular_blinn	Blinn for specular (compatibility).
specular_phong	Phong for specular (compatibility).
specular_toon	Toon for specular.
specular_disabled	Disable specular.
skip_vertex_transform	VERTEX/NORMAL/etc need to be transformed manually in VS.
world_vertex_coords	VERTEX/NORMAL/etc are modified in world coordinates instead of local.
vertex_lighting	Use vertex-based lighting.

Vertex Built-Ins

Built-In	Description
out mat4 **WORLD_MATRIX**	Model space to world space transform.
in mat4 **INV_CAMERA_MATRIX**	World space to view space transform.
out mat4 **PROJECTION_MATRIX**	View space to clip space transform.
in mat4 **CAMERA_MATRIX**	View space to world space transform.
out mat4 **MODELVIEW_MATRIX**	Model space to view space transform (use if possible).
out 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).
out vec3 **VERTEX**	Vertex in local coords (see doc below).
out vec3 **NORMAL**	Normal in local coords.
out vec3 **TANGENT**	Tangent in local coords.
out vec3 **BINORMAL**	Binormal in local coords.
out vec2 **UV**	UV main channel.
out vec2 **UV2**	UV secondary channel.
out vec4 **COLOR**	Color from vertices.
out float **POINT_SIZE**	Point size for point rendering.
in int **INSTANCE_ID**	Instance ID for instancing.
in vec4 **INSTANCE_CUSTOM**	Instance custom data (for particles, mostly).
out float **ROUGHNESS**	Roughness for vertex lighting.

Values marked as "in" are read-only. Values marked as "out" are for optional writing. Samplers are not subjects of writing and they are not marked.

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 **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 **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.
out vec3 **NORMAL**	Normal that comes from vertex function, in view space.
out vec3 **TANGENT**	Tangent that comes from vertex function.
out 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 vec2 **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).
sampler2D **SCREEN_TEXTURE**	Built-in Texture for reading from the screen. Mipmaps contain increasingly blurred copie
sampler2D **DEPTH_TEXTURE**	Built-in Texture for reading depth from the screen. Must convert to linear using INV_PR
out vec2 **SCREEN_UV**	Screen UV coordinate for current pixel.
in vec2 **POINT_COORD**	Point Coord for drawing points with POINT_SIZE.
out float **ALPHA_SCISSOR**	If written to, values below a certain amount of alpha are discarded.

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 **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).
out vec2 **VERTEX**	Vertex in image space.
out vec2 **UV**	UV.
out vec4 **COLOR**	Color from vertex primitive.
out float **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 canvas_item;
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.
out vec4 **COLOR**	Color from vertex function.
sampler2D **TEXTURE**	Default 2D texture.
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**	Size of individual pixels. Equal to inverse of resolution.
in vec2 **POINT_COORD**	Coordinate for drawing points.
in float **TIME**	Global time in seconds.
in bool **AT_LIGHT_PASS**	True if this is a light pass (for multi-pass light rendering).
sampler2D **SCREEN_TEXTURE**	Screen texture, mipmaps contain gaussian blurred versions.

Light Built-Ins

Built-In	Description
in vec2 **POSITION**	Fragment coordinate, pixel adjusted.
in vec3 **NORMAL**	Input Normal. Although this value is passed in, **normal calculation still happens outside of this function**.
in vec2 **UV**	UV from vertex function, equivalent to the UV in the fragment function.
in vec4 **COLOR**	Input Color. This is the output of the fragment function with final modulation applied.
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**	Global time in seconds.
out vec2 **LIGHT_VEC**	Vector from light to fragment, can be modified to alter shadow computation.
out float **LIGHT_HEIGHT**	Height of Light.
out vec4 **LIGHT_COLOR**	Color of Light.
out vec2 **LIGHT_UV**	UV for Light texture.
out vec4 **SHADOW_COLOR**	Shadow Color of Light. **(not yet implemented)**
out vec4 **LIGHT**	Value from the Light texture. **(shader is ignored if this is not used).** Can be modified.

Particles

Accepted render modes and built-ins for **shader_type particles;**.

Render Modes

Render Mode	Description
keep_data	Do not clear previous data on restart.
disable_force	Disable force.
disable_velocity	Disable velocity.

Vertex Built-Ins

Built-In	Description
out vec4 **COLOR**	Particle color, can be written to.
out vec3 **VELOCITY**	Particle velocity, can be modified.
out float **MASS**	Particle mass, use for attractors (default 1).
out bool **ACTIVE**	Particle is active, can be set to false.
in bool **RESTART**	Set to true when particle must restart (lifetime cycled).
out vec4 **CUSTOM**	Custom particle data.
out mat4 **TRANSFORM**	Particle transform.
in float **TIME**	Global time in seconds.
in float **LIFETIME**	Particle lifetime.
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.

17.2 Shader materials

17.2.1 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.

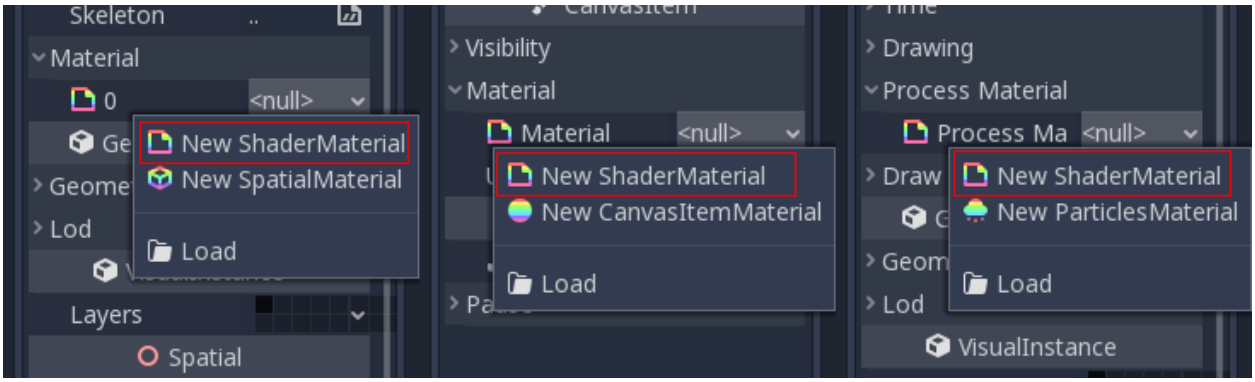
Shader materials allow writing a custom shader directly, for maximum flexibility. Examples of this are:

- 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!

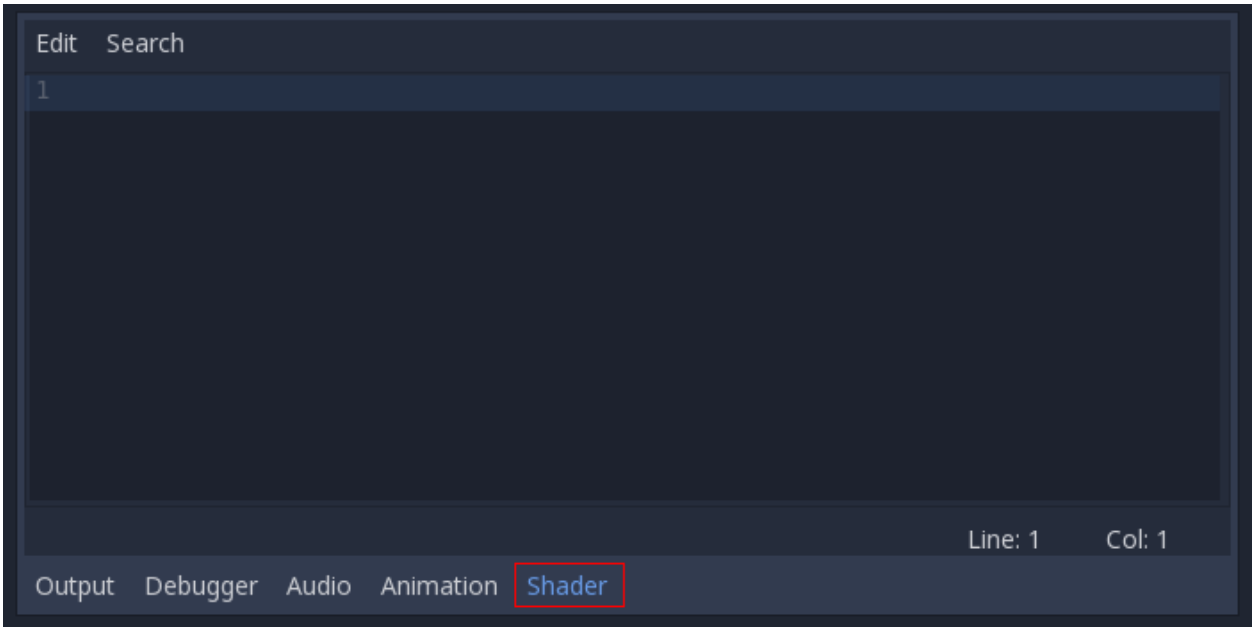
Traditionally, most engines will ask you to learn GLSL, HLSL or CG, which are pretty complex for the skillset of most artists. Godot uses a simplified version of a shader language that will detect errors as you type, so you can see your edited shaders in real-time. Additionally, it is possible to edit shaders using a visual, node-based graph editor.

17.2.2 Creating a ShaderMaterial

Create a new ShaderMaterial in some object of your choice. Go to the "Material" property and create a ShaderMaterial.

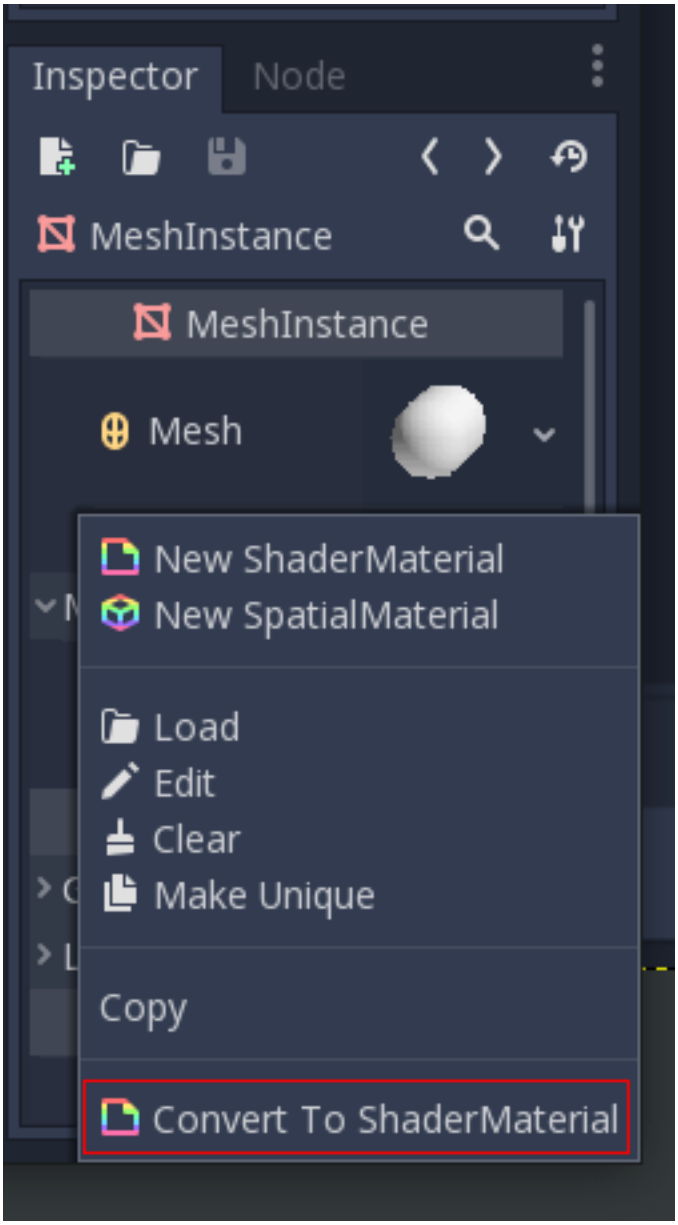


Edit the newly created shader, and the shader editor will open:



17.2.3 Converting to ShaderMaterial

It is possible to convert from SpatialMaterial, CanvasItemMaterial and ParticlesMaterial to ShaderMaterial. To do this go to the material properties and enable the convert option.



17.3 Screen-reading shaders

17.3.1 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!

17.3.2 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:

```
void fragment() {}
        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 separatable gaussian blur to it's mipmaps.

This allows for not only reading from the screen, but reading from it with different amounts of blur at no cost.

17.3.3 SCREEN_TEXTURE example

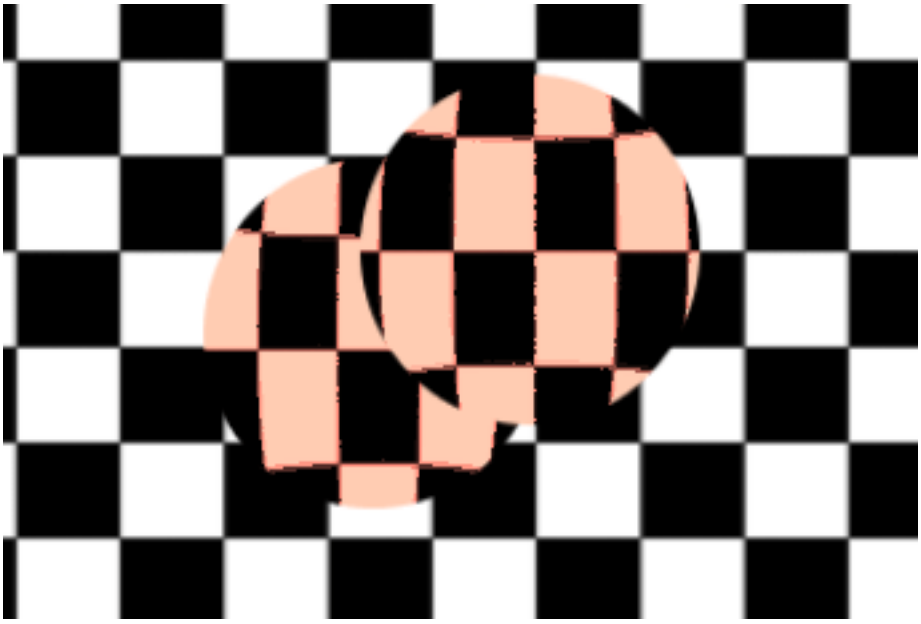
SCREEN_TEXTURE can be used for a lot of things. There is a special demo for *Screen Space Shaders*, that you can download to see and learn. One example is a simple shader to adjust brightness, contrast and saturation:

```
shader_type canvas_item;

uniform float brightness = 1.0;
uniform float contrast = 1.0;
uniform float saturation = 1.0;

void fragment() {
        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;
}
```

17.3.4 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 inefficient.

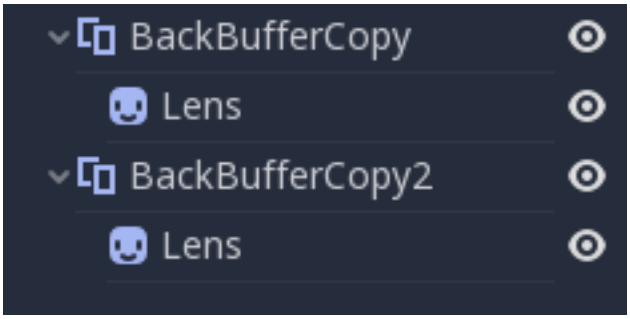
As a result, if shaders that use SCREEN_TEXTURE overlap, the second one will not use the result of the first one, resulting in unexpected visuals:



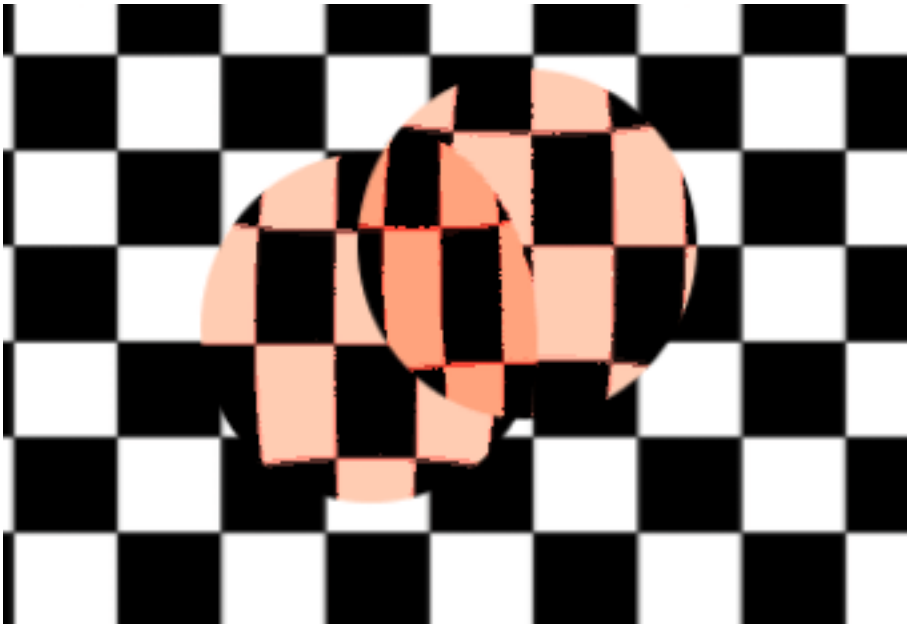
In the above image, the second sphere (top right) is using the same source for SCREEN_TEXTURE as the first one below, so the first one "disappears", or is not visible.

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:



17.3.5 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!

17.3.6 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:

```
void fragment() {
        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 18

Networking

18.1 High level multiplayer

18.1.1 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.

18.1.2 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.

18.1.3 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_network_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.)

18.1.4 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".

18.1.5 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.

18.1.6 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.

18.1.7 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))
```

(continues on next page)

(continued from previous page)

```
        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 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!
```

18.1.8 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).

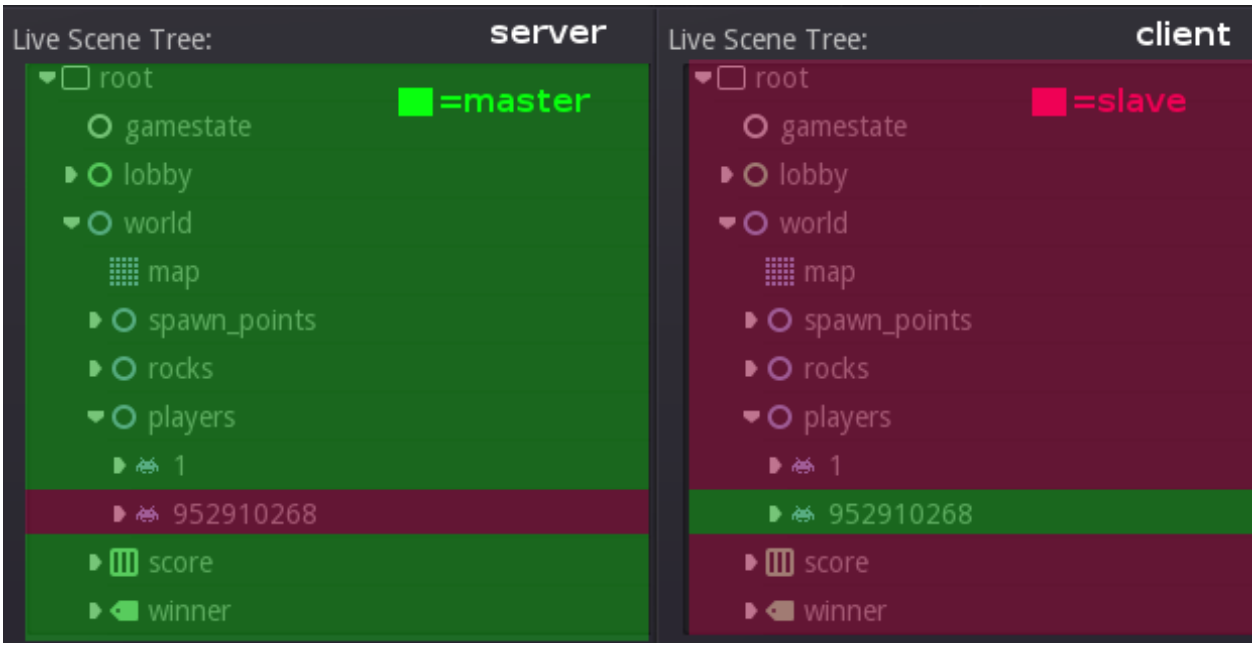
Checking that a specific node instance on a peer is the network master for this node for all connected peers is done by calling *Node.is_network_master()*. This will return true when executed on the server and false on all client peers.

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
```

18.2 Making HTTP requests

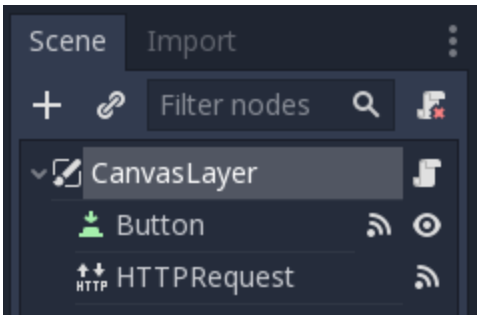
The *HTTPRequest* node is the easiest way to make HTTP requests in Godot. It is backed by the more low-level *HTTPClient*, for which a tutorial is available *here*.

For the sake of this example, we will create a simple UI with a button, that when pressed will start the HTTP request to the specified URL.

18.2.1 Preparing scene

Create a new empty scene, add a CanvasLayer as the root node and add an script to it. Then add two child nodes to it: a Button and an HTTPRequest node. You will need to connect the following signals to the CanvasLayer script:

- Button.pressed: When the button is pressed, we will start the request.
- HTTPRequest.request_completed: When the request is completed, we will get the requested data as an argument.



18.2.2 Scripting

Below is all the code we need to make it work. The URL points to an online API mocker; it returns a pre-defined JSON string, which we will then parse to get access to the data.

```
extends CanvasLayer

func _ready():
    pass

func _on_Button_pressed():
    $HTTPRequest.request("http://www.mocky.io/v2/5185415ba171ea3a00704eed")

func _on_HTTPRequest_request_completed( result, response_code, headers, body ):
    var json = JSON.parse(body.get_string_from_utf8())
    print(json.result)
```

With this, you should see `(hello:world)` printed on the console; hello being a key, and world being a value, both of them strings.

For more information on parsing JSON, see the class references for *JSON* and *JSONParseResult*.

Note that you may want to check whether the `result` equals `RESULT_SUCCESS` and whether a JSON parsing error occurred, see the JSON class reference and *HTTPRequest* for more.

Of course, you can also set custom HTTP headers. These are given as a string array, with each string containing a header in the format `"header: value"`. For example, to set a custom user agent (the HTTP `user-agent` header) you could use the following:

```
$HTTPRequest.request("http://www.mocky.io/v2/5185415ba171ea3a00704eed", ["user-agent:␣
↪YourCustomUserAgent"])
```

Please note that for SSL/TLS encryption and thus HTTPS URLs to work you may need to take some steps as described *here*.

Also, when calling APIs using authorization, be aware that someone might analyse and decompile your released application and thus may gain access to any embedded authorization information like tokens, usernames or passwords. That means it is usually not a good idea to embed things such as database access credentials inside your game. Avoid providing information useful to an attacker whenever possible.

18.2.3 Sending data to server

Until now we have limited ourselves to requesting data from a server. But what if you need to send data to the server? Here is a common way of doing it:

```
func _make_post_request(url, data_to_send, use_ssl):
    # Convert data to json string:
    var query = JSON.print(data_to_send)
    # Add 'Content-Type' header:
    var headers = ["Content-Type: application/json"]
    $HTTPRequest.request(url, headers, use_ssl, HTTPClient.METHOD_POST, query)
```

18.3 HTTP client class

HTTPClient provides low-level access to HTTP communication. For a more higher-level interface, you may want to take a look at *HTTPRequest* first, which has a tutorial available *here*.

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)
```

(continues on next page)

(continued from previous page)

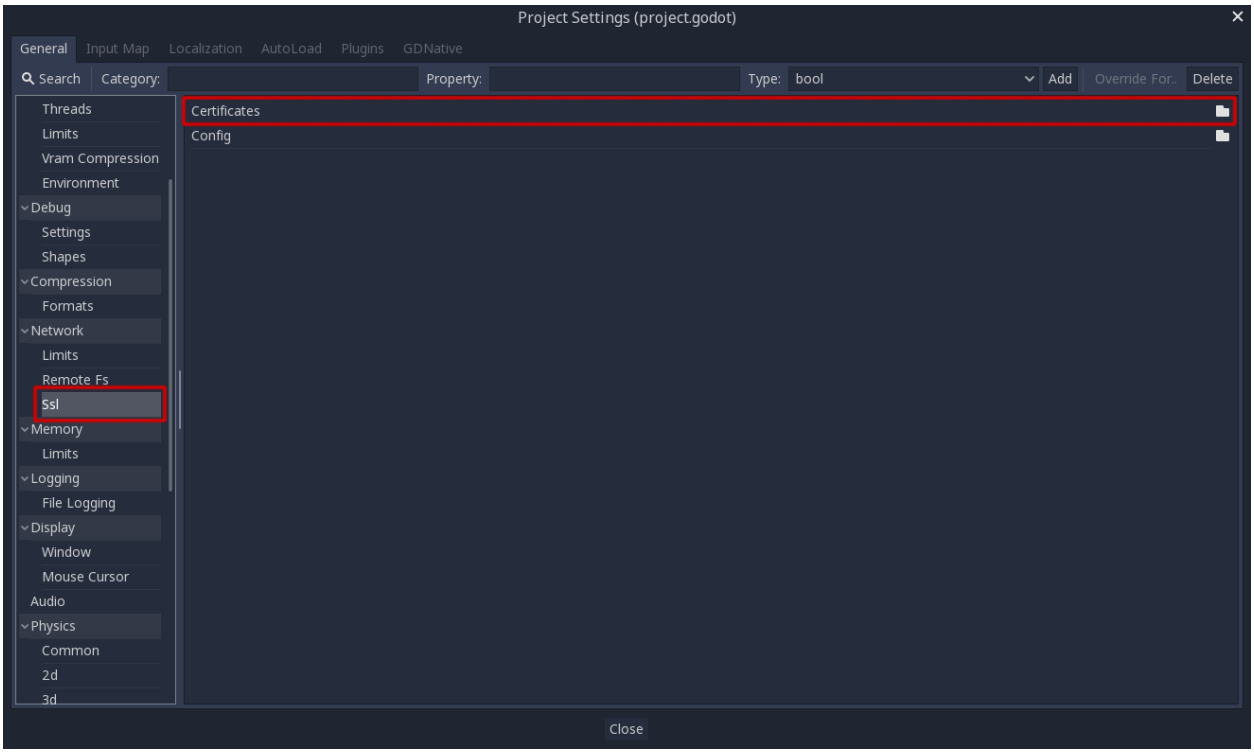
```
    assert(http.get_status() == HTTPClient.STATUS_BODY or http.get_status() ==␣
↪HTTPClient.STATUS_CONNECTED) # Make sure request finished well.

    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()
```

18.4 SSL certificates

18.4.1 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 https://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:

18.4.2 Approach 1: self signed cert

The first approach is the simplest: 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.

18.4.3 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 pick any of the more specific certificates there if you are connecting to a specific one.

CHAPTER 19

Asset Library

19.1 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!

19.1.1 Frequently asked questions

Can paid assets be uploaded to the asset library?

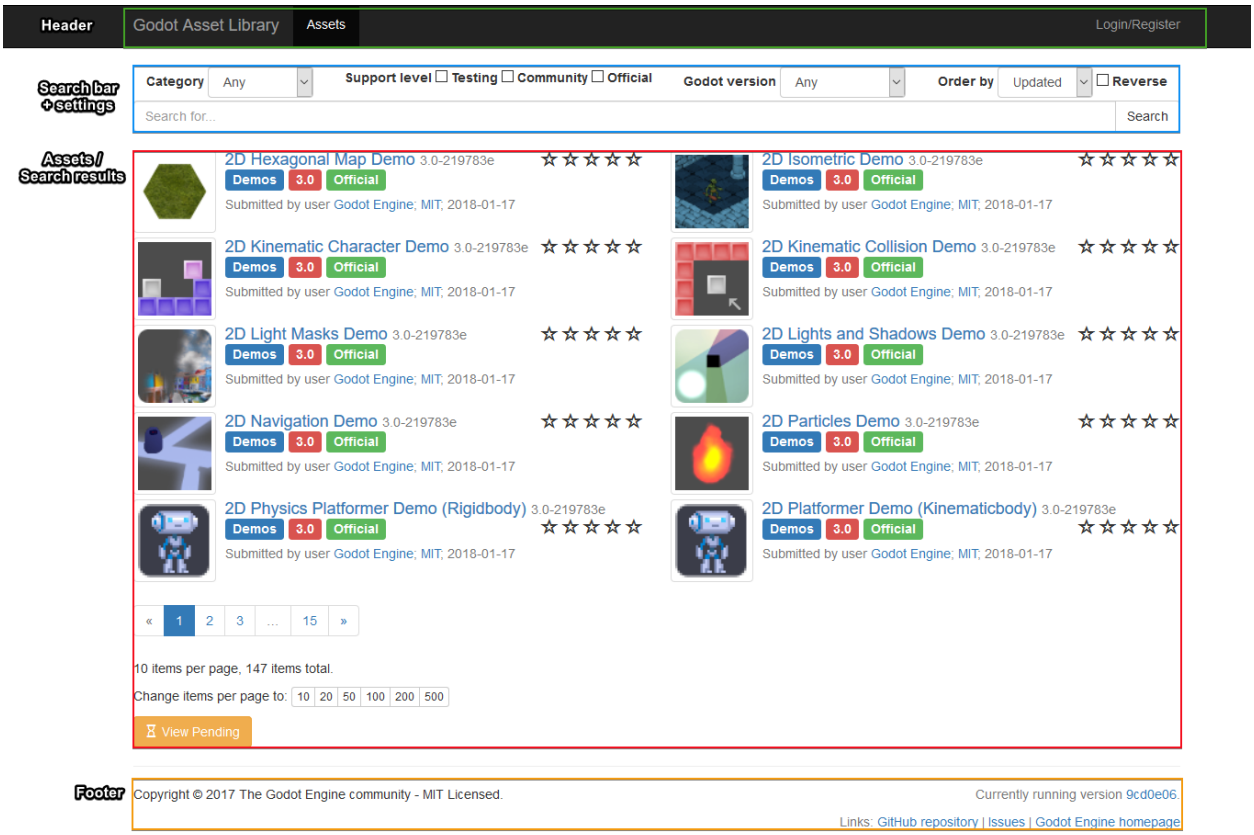
Not to the official one, though in the future there might be other asset libraries which allow it. That said, you are allowed to monetize and sell Godot assets outside the Asset Library.

19.2 Using the AssetLib

19.2.1 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:



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**.

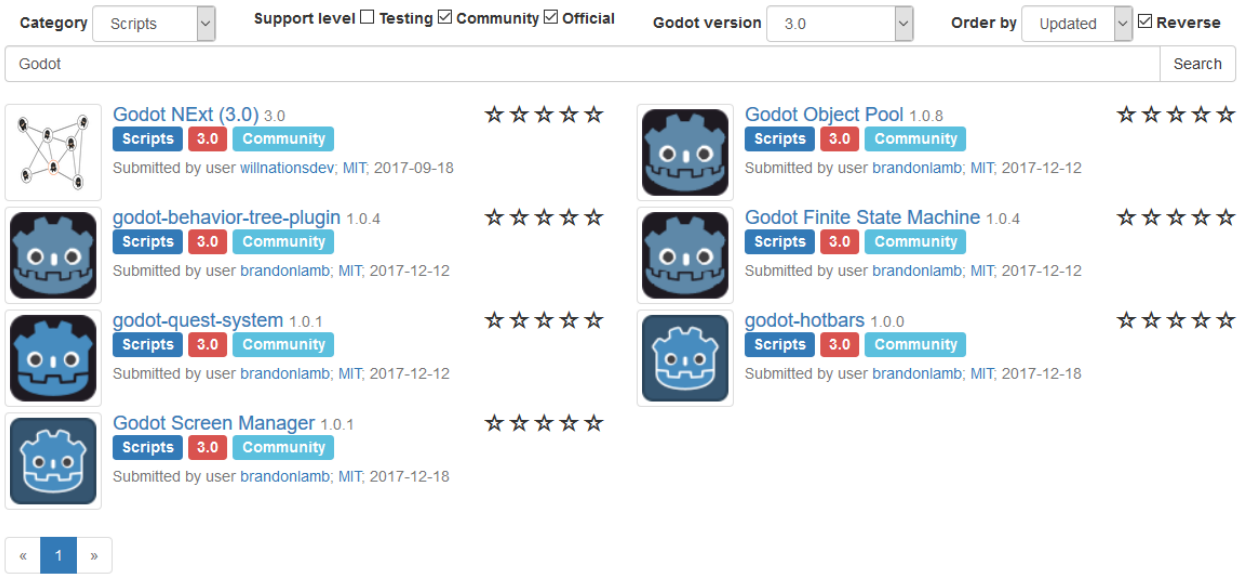
While most other filter settings should be fairly self-explanatory, it's worth going over what "support level" means in the Asset Library. Currently there are three support levels, and each asset can belong to only one.

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.

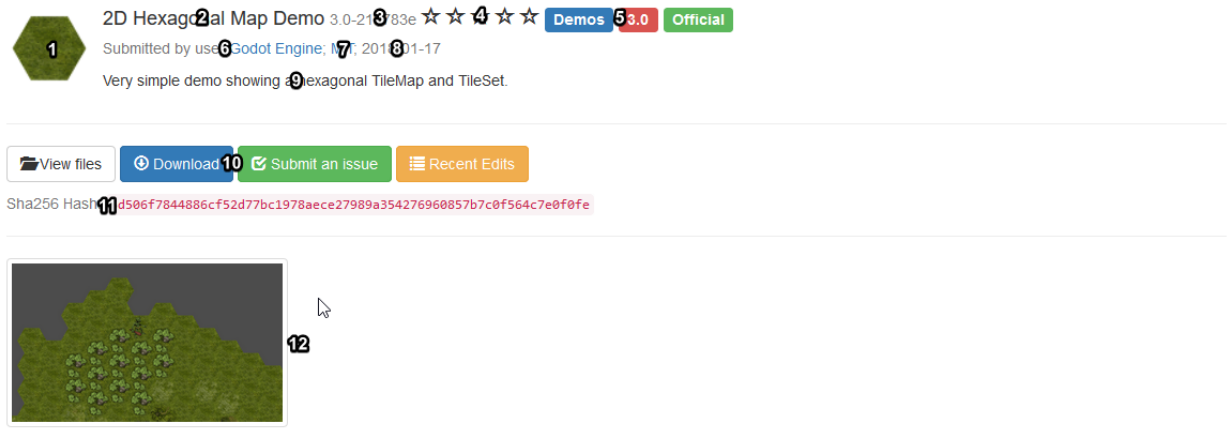
You can mix and match any of the search filters and criteria, and upon clicking the Search button, receive the list of all assets in the Library that match them.



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.



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.

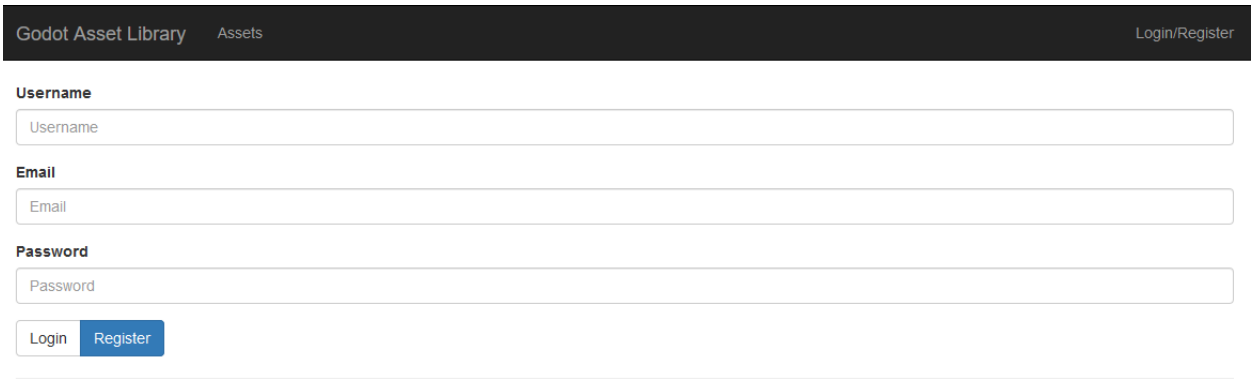
8. The date of the asset's latest edit/update.
9. A textual description of the asset.
10. Links related to the asset (download link, file list, issue tracker).
11. A SHA-256 hash of the asset, for download validation purposes.
12. Images and videos showcasing the asset.

Registering and logging in

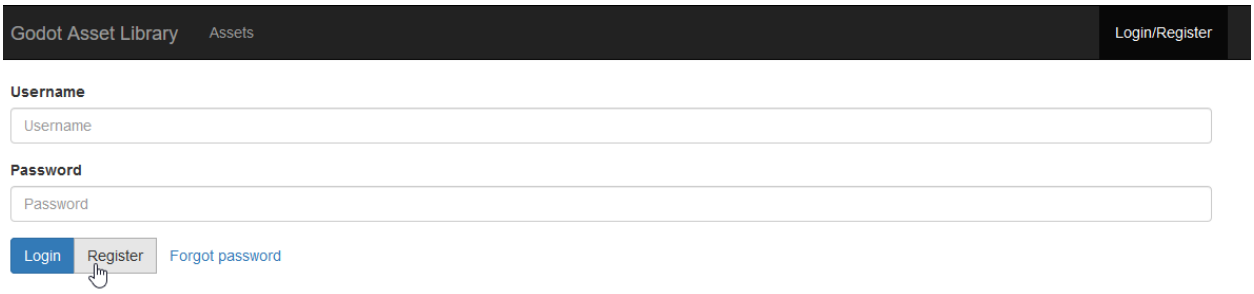
In order to upload assets to the AssetLib, you need to be logged in, and to do that, you need a registered user account. In the future, this may also give you access to other features, such as commenting on or rating the existing assets. You do *not* need to be logged in to browse and download the assets.

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.



Then, you can use your username and password to log in.



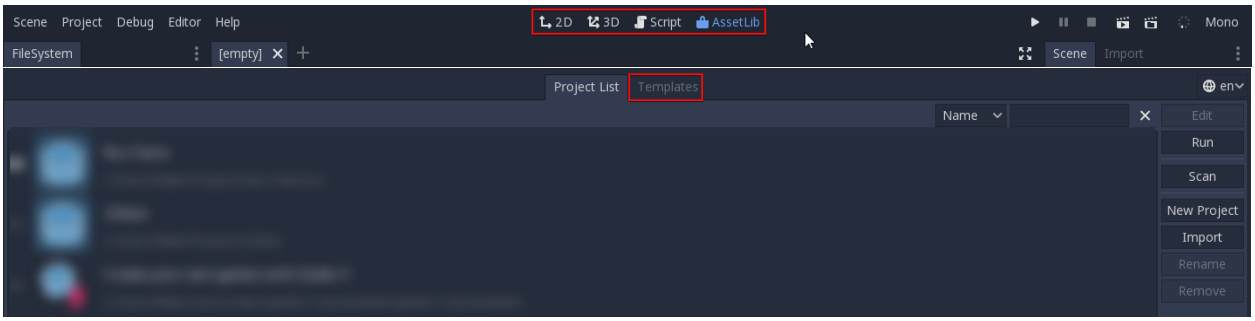
This will change the look of the AssetLib header. Now you get access to a handful new functions:

- 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.

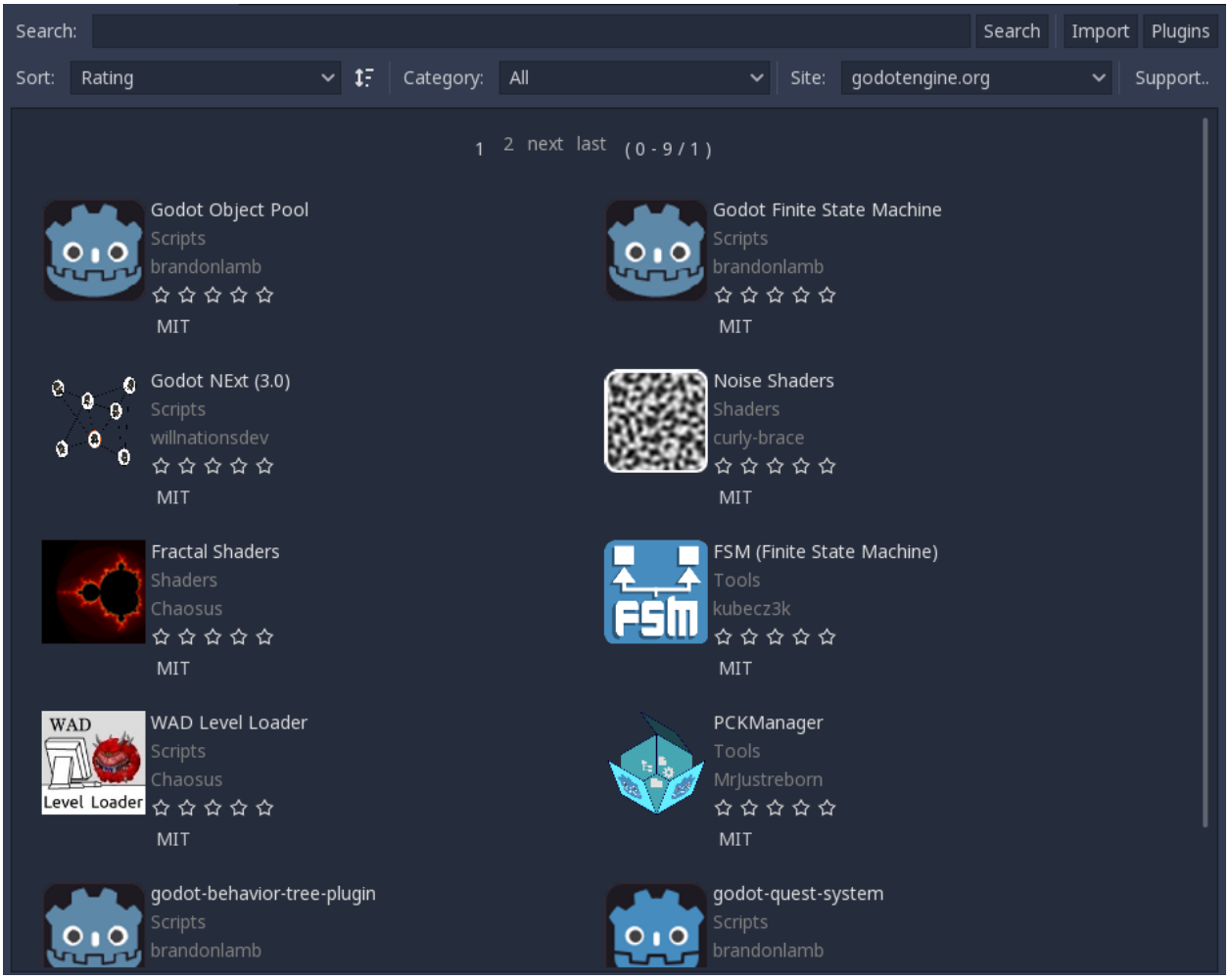
You can learn how to submit assets to the Library, and what the asset submission guidelines are, in the next part of this tutorial, :*Submitting to the Asset Library*.

19.2.2 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:

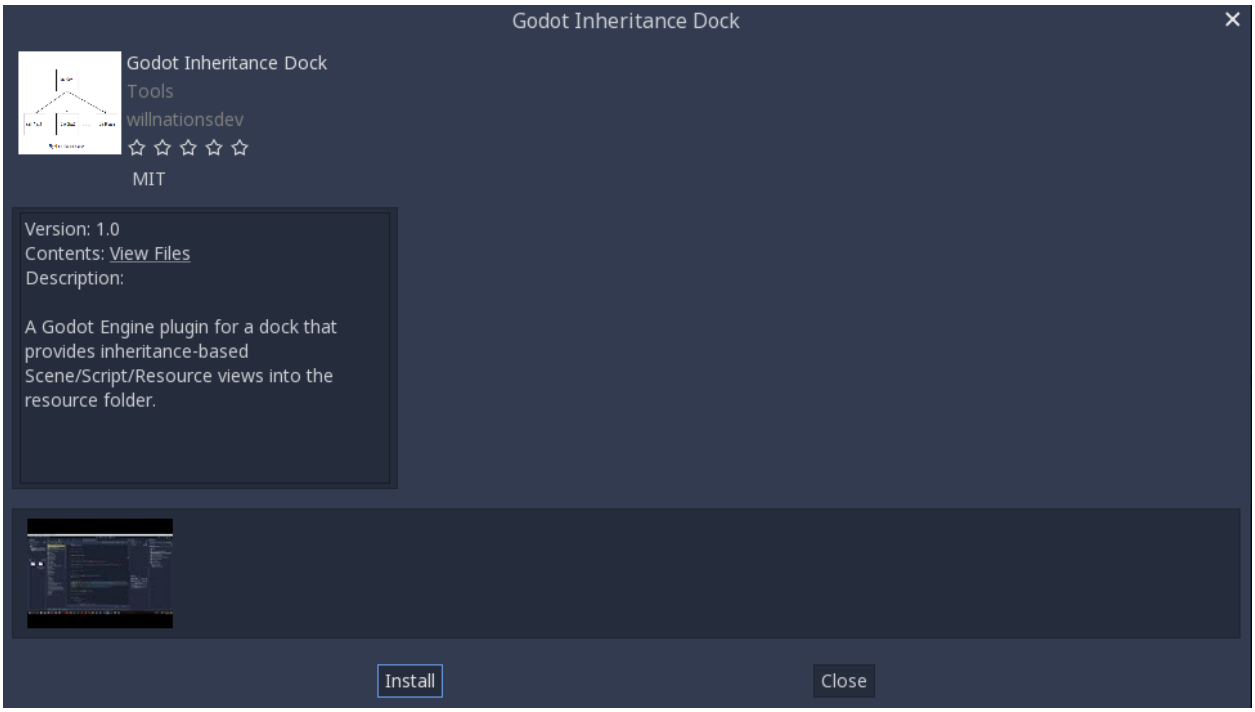


Similarly to the web version of the AssetLib, here you can search for assets by category, name, and sort them by factors such as name or edit date.

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).

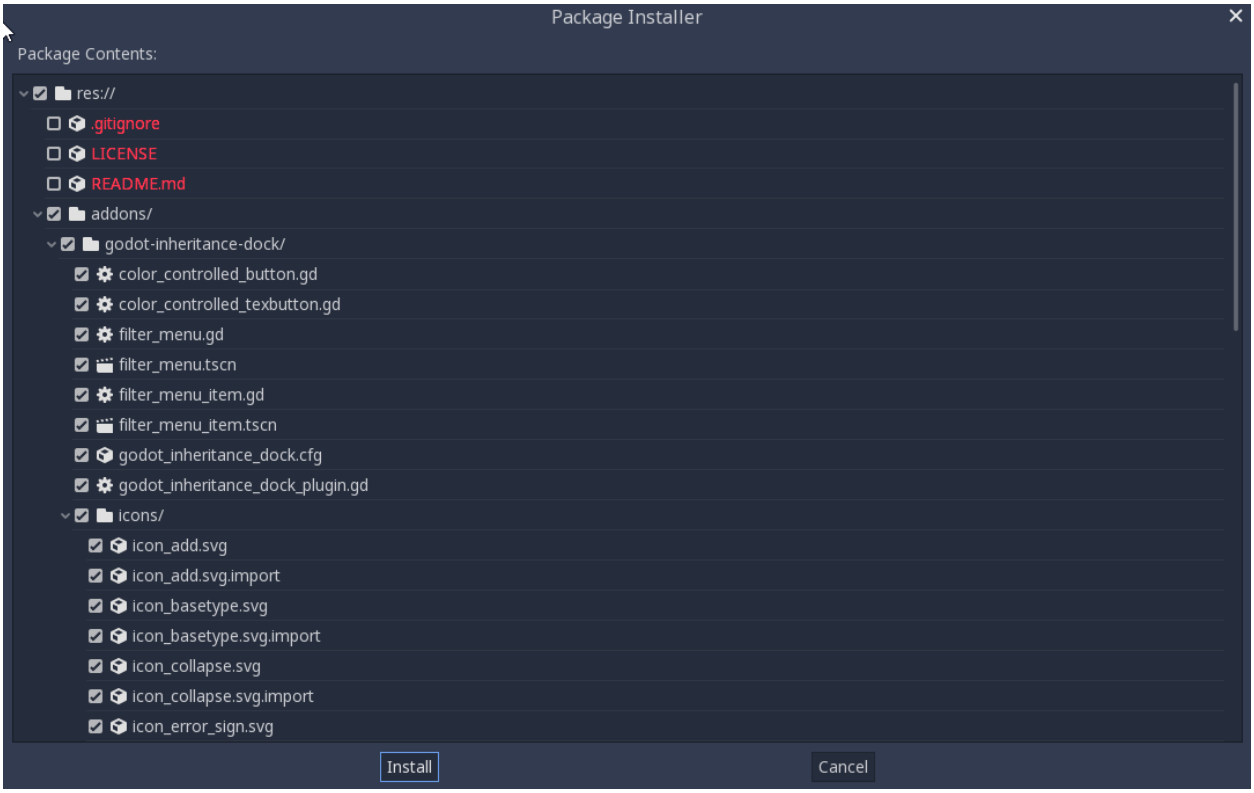
In the future, you will be able to choose a different AssetLib provider to fetch assets from (using the Site dropdown menu), however currently only the official Godot website version of the AssetLib is supported, as well as the version that may be running on your local machine's web server (the localhost option).

When you click on an asset, you will see more information about it.

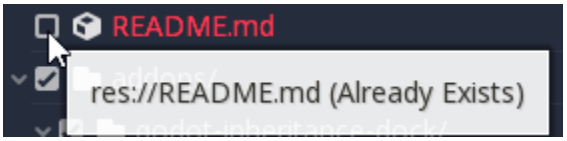


If you click on the Install button, Godot will fetch an archive of the asset, and will track download progress of it at the bottom of the editor window. If the download fails, you can retry it using the Retry button.

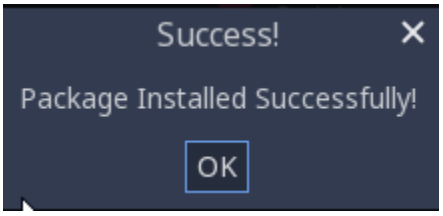
When it finishes, you can proceed to install it using the Install button. This will bring up the Package Installer window.



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.

19.3 Submitting to the Asset Library

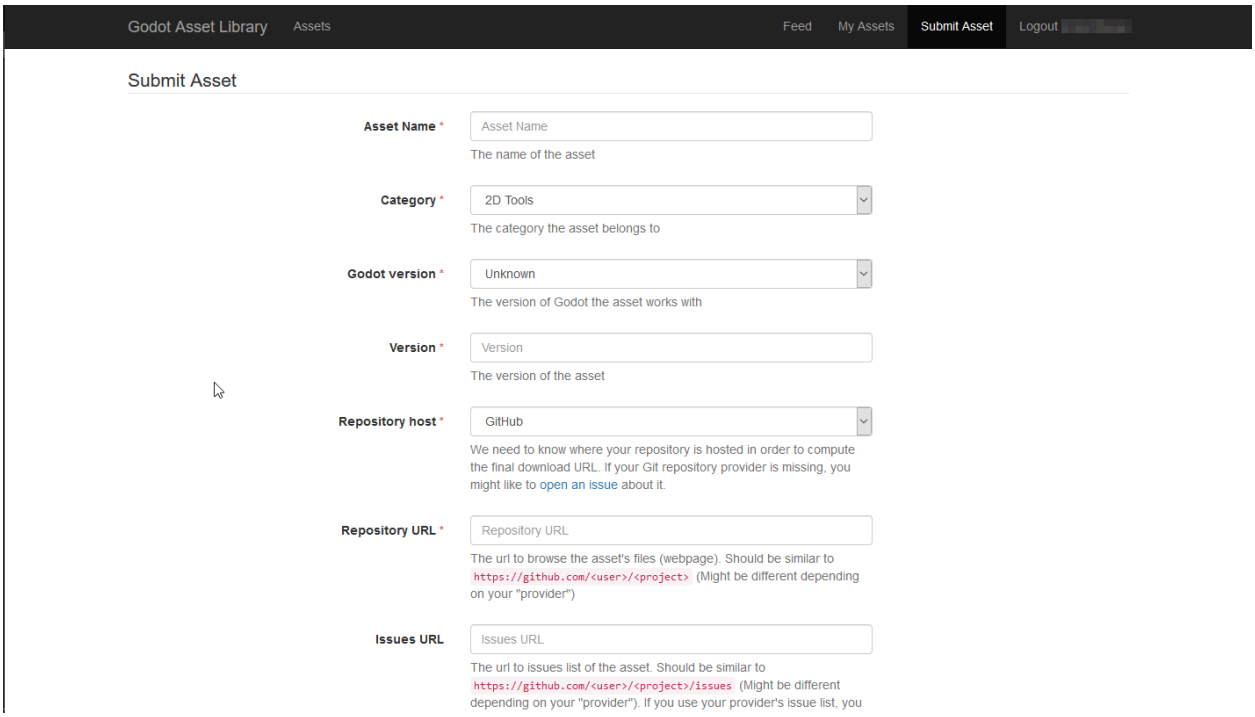
19.3.1 Introduction

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.

19.3.2 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:



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.cmo/<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 liceses, 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.

19.3.3 Submission guidelines

[TODO]

CHAPTER 20

VR

20.1 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.

20.1.1 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 straightforward 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:

GDScript

C#

```
var arvr_interface = ARVRServer.find_interface("Native mobile")
if arvr_interface and arvr_interface.initialize():
    get_viewport().arvr = true
```

```
var arvrInterface = ARVRServer.FindInterface("Native mobile");
if (arvrInterface != null && arvrInterface.Initialize())
{
    GetViewport().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 disconcerting to the gamer.

20.1.2 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
- *ARVRController* - 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.

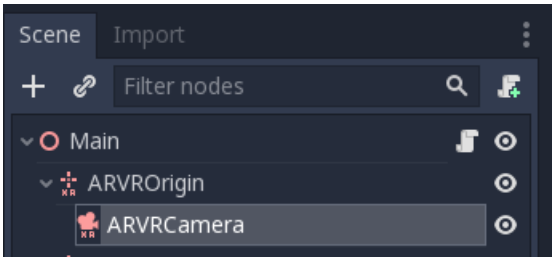
ARVROrigin is an important node, you must have one and only one of these somewhere in your scene. This node maps the center of your real world tracking space to a location in your virtual world. Everything else is positionally tracked in relation to this point. Where this point lies exactly differs from 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.

20.1.3 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 then 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 should target your games to run at a stable 90fps 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 foviated 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 21

Plugins

21.1 Editor plugins

21.1.1 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.

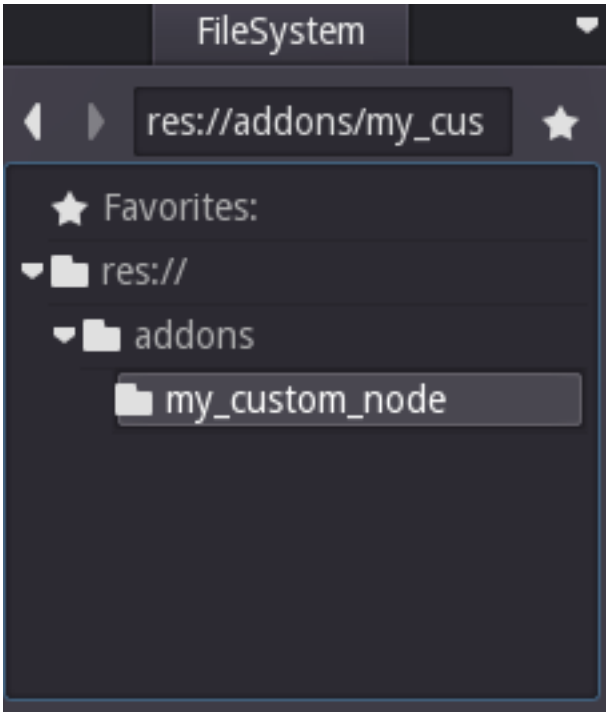
This tutorial will guide you through the creation of two simple plugins so you can understand how they work and be able to develop your own. The first will be a custom node that you can add to any scene in the project and the other will be a custom dock added to the editor.

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
```

(continues on next page)

(continued from previous page)

```
    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:

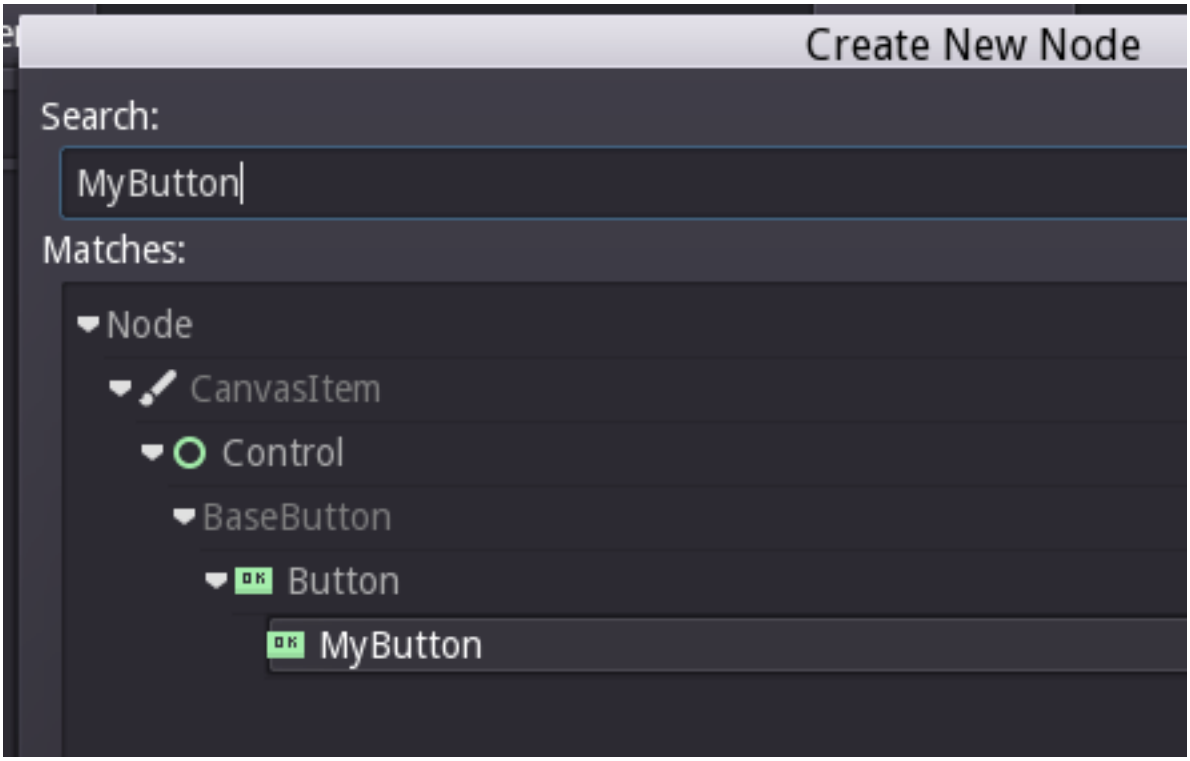
```
tool
extends Button

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:

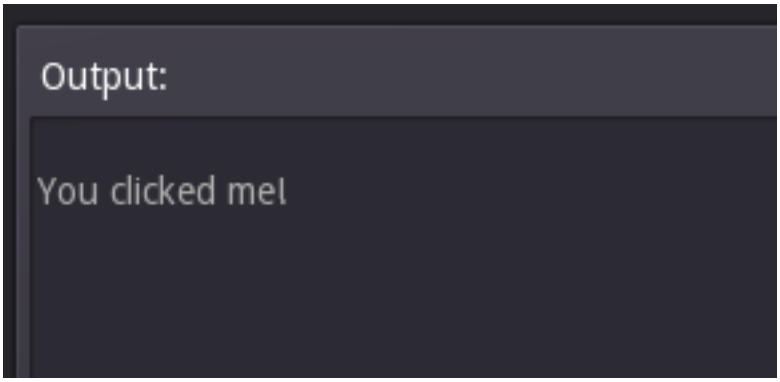
Now we need to add it as a custom type so it shows on the Create New Node dialog. For that, change the `custom_node.gd` script to the following:

```
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"
```

(continues on next page)

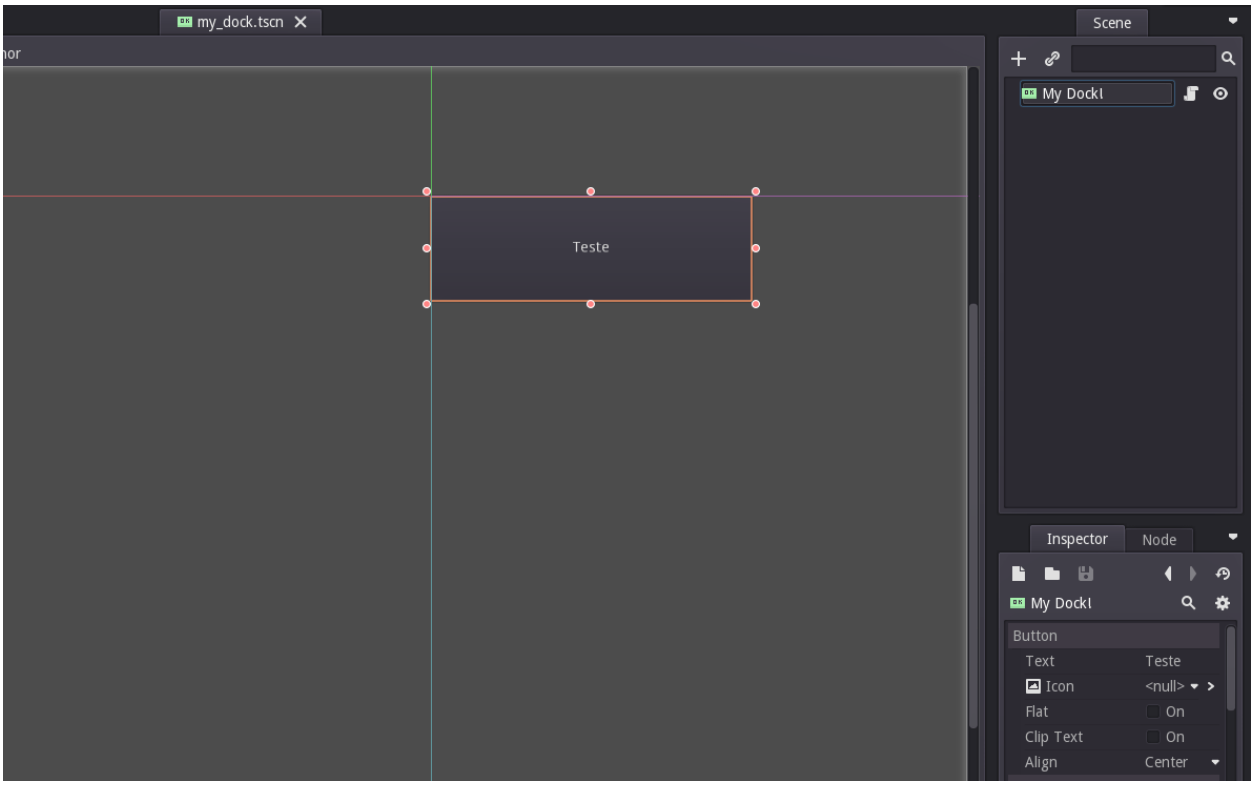
(continued from previous page)

```
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 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 straightforward, you 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 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:
```

(continues on next page)

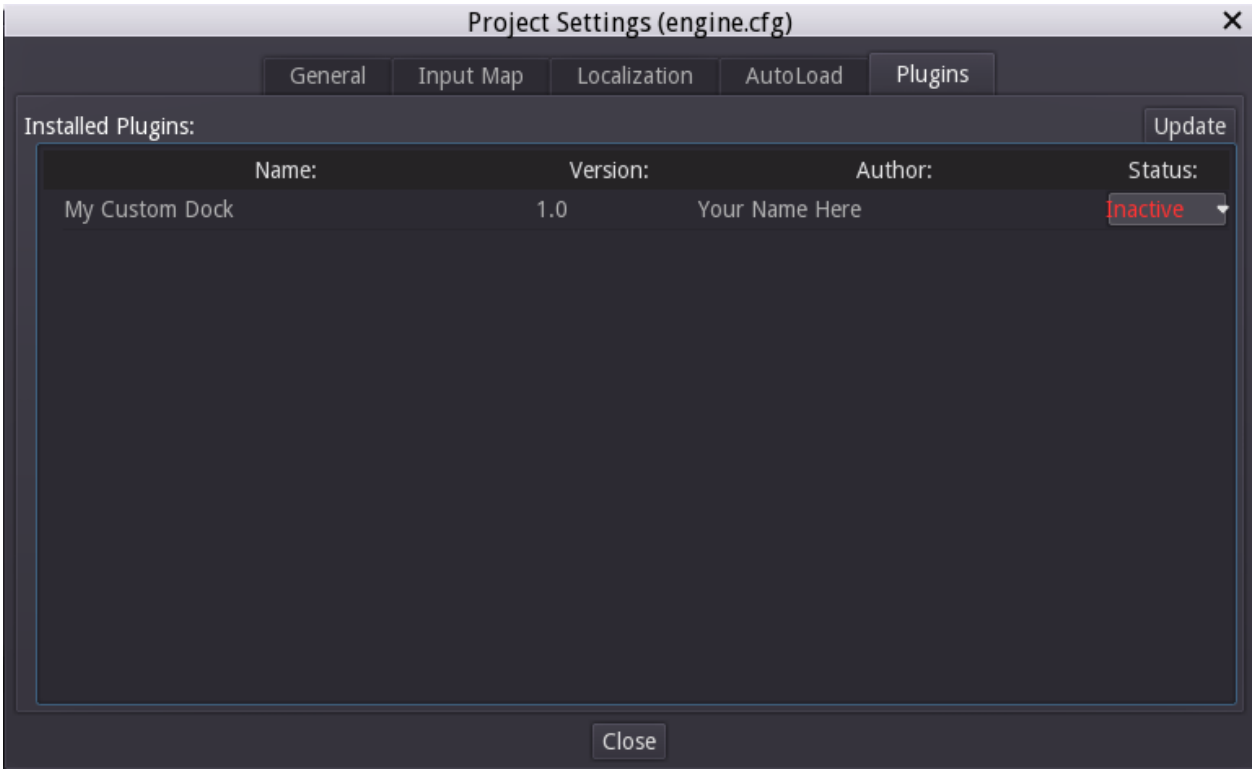
(continued from previous page)

```
    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 need to click on the status to select *Active*. The dock should be immediately visible, even before you close the settings window. You should have a custom dock:

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.

21.1.2 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"
```

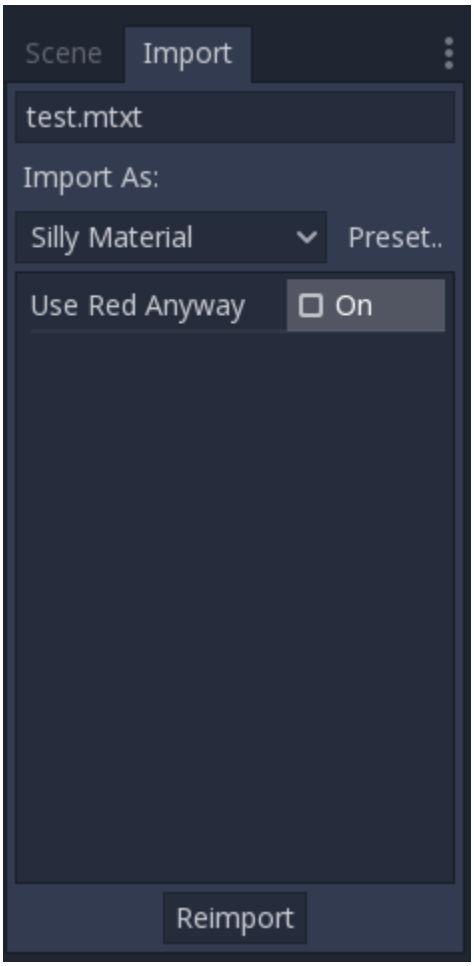
The imported resource has a specific type, so the editor can know which property slot it belongs to. This allows drag and drop from the FileSystem dock to a property in the Inspector.

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 plugins 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 its 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 *PropertyHint* values to use as hint.
`hint_string`	String	The hint text of the property. The same as you'd add in the `export` statement in GDScript.
`usage`	Enum value	One of the *PropertyUsageFlags* 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 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:

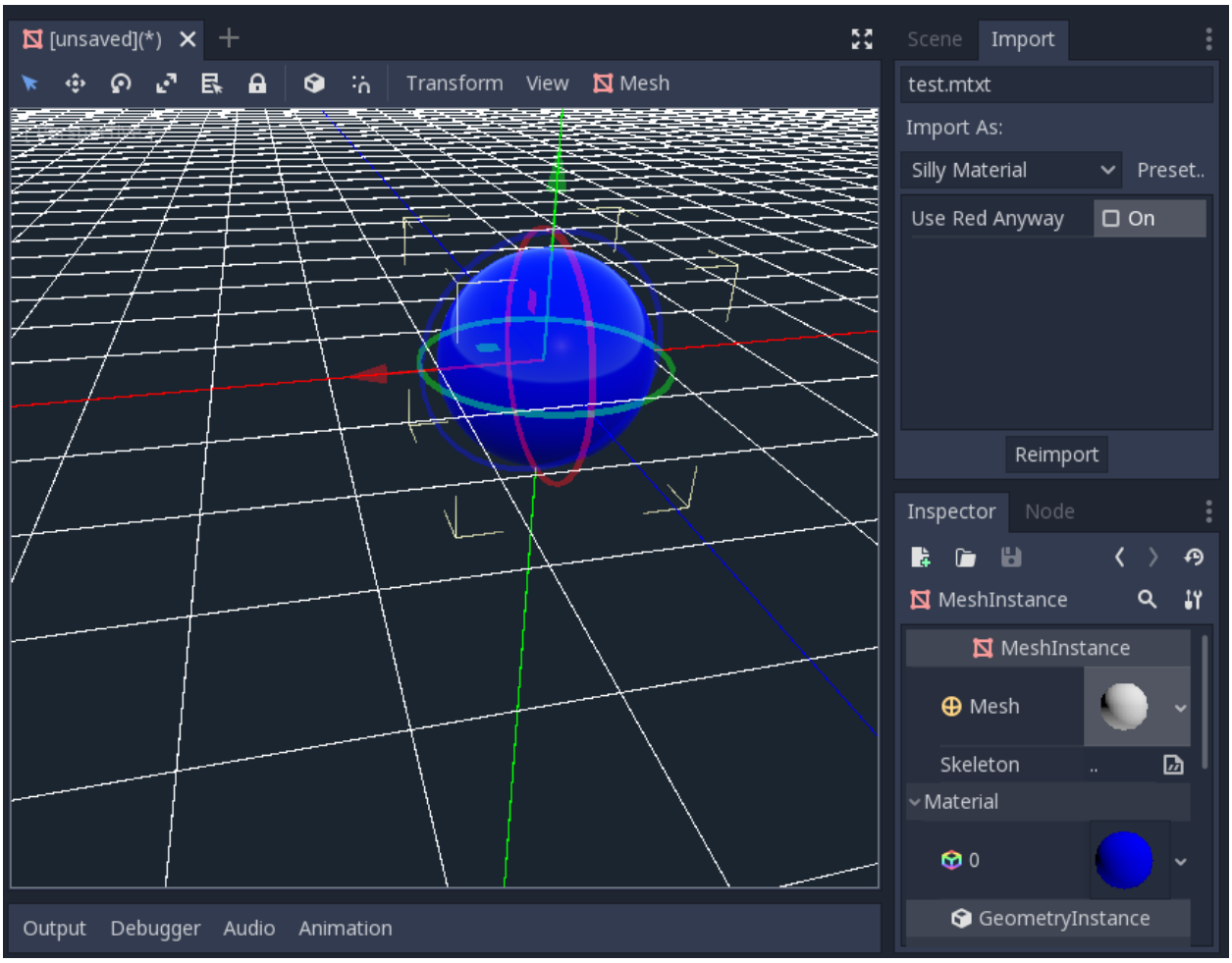
```
var next_pass = SpatialMaterial.new()
next_pass.albedo_color = color.inverted()
var next_pass_path = "%s.next_pass.%s" % [save_path, get_save_extension()]

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 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 it's 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.

21.2 GDNative

21.2.1 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 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 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;
```

(continues on next page)

(continued from previous page)

```
            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);
    api->godot_string_parse_utf8(&data, 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 -
↪arch x86_64
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.gdnlib 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

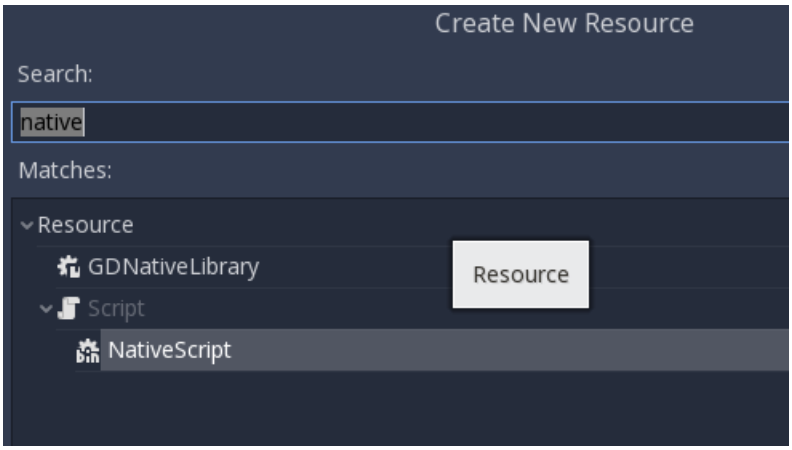
Now that we should have a working GDNative library it is time to fire up Godot and use it. Open up the sample project if you haven't left it open after creating the project all the way at the beginning of this tutorial.

Creating our GDNS file

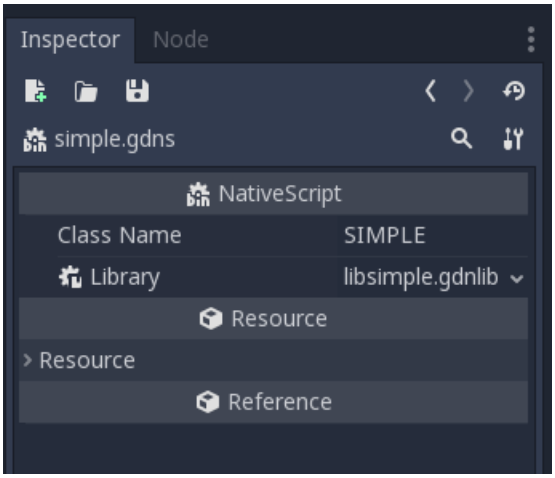
With our GDNLIB file we've told Godot how to load our library, now we need to tell it about our "Simple" object class. This we do by creating a GDNS resource file.

Start by clicking the create resource button in the Inspector:

And select NativeScript:

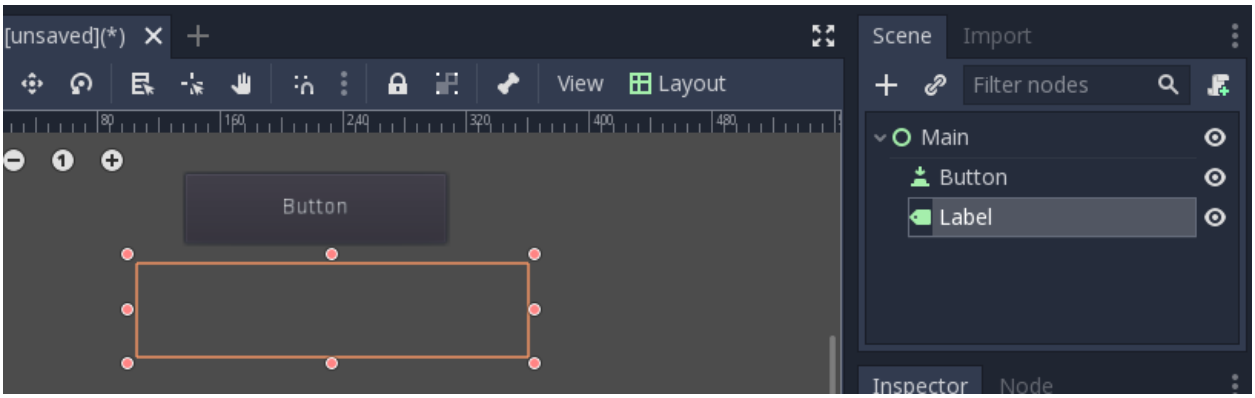


Press Create, now the inspector will show a few fields we need to enter. In Class Name we enter "SIMPLE" which is the object class name we used in our C source when calling godot_nativescript_register_class. We also need to select our GDNLIB file by clicking on Library and selecting Load:



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:

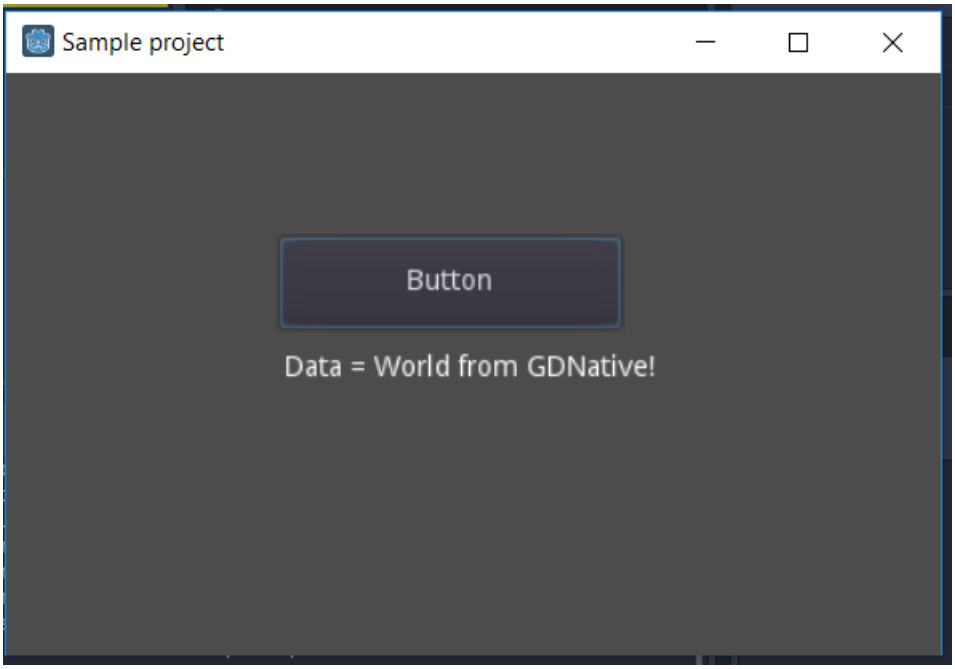
Next link up the pressed signal on the button to your script:

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:


Sample project
Button
Data = World from GDNative!

CHAPTER 22

Platform-specific

22.1 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.

22.1.1 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.

22.1.2 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.

22.1.3 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

22.1.4 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.

22.1.5 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

22.2 Services for iOS

At the moment, there are two iOS APIs partially implemented, GameCenter and Storekit. Both use the same model of asynchronous calls explained below.

22.2.1 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 two 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
```

(continues on next page)

(continued from previous page)

```
    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 two 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.

22.2.2 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 two 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 ],
}
```

22.2.3 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);`
- `Erroraward_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 two 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 two 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",
}
```

22.2.4 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"):
```

(continues on next page)

(continued from previous page)

```
        GameCenter = Globals.get_singleton("GameCenter")
        # connect your timer here to the "check_events" function
```

22.3 Console Support in Godot

22.3.1 Official Support

Godot does not officially support consoles (save for XBox One via UWP) currently.

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 a 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 consoles. Quite the contrary.

22.3.2 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.

CHAPTER 23

Miscellaneous

23.1 Handling quit requests

23.1.1 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).

23.1.2 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):

GDScript

C#

```
func _notification(what):
    if what == MainLoop.NOTIFICATION_WM_QUIT_REQUEST:
        get_tree().quit() # default behavior
```

```
public override void _Notification(int what)
{
    if (what == MainLoop.NotificationWmQuitRequest)
        GetTree().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:

GDScript

C#

```
get_tree().set_auto_accept_quit(false)
```

```
GetTree().SetAutoAcceptQuit(false);
```

23.2 Pausing games

23.2.1 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.

23.2.2 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:

GDScript

C#

```
get_tree().paused = true
```

```
GetTree().Paused = true;
```

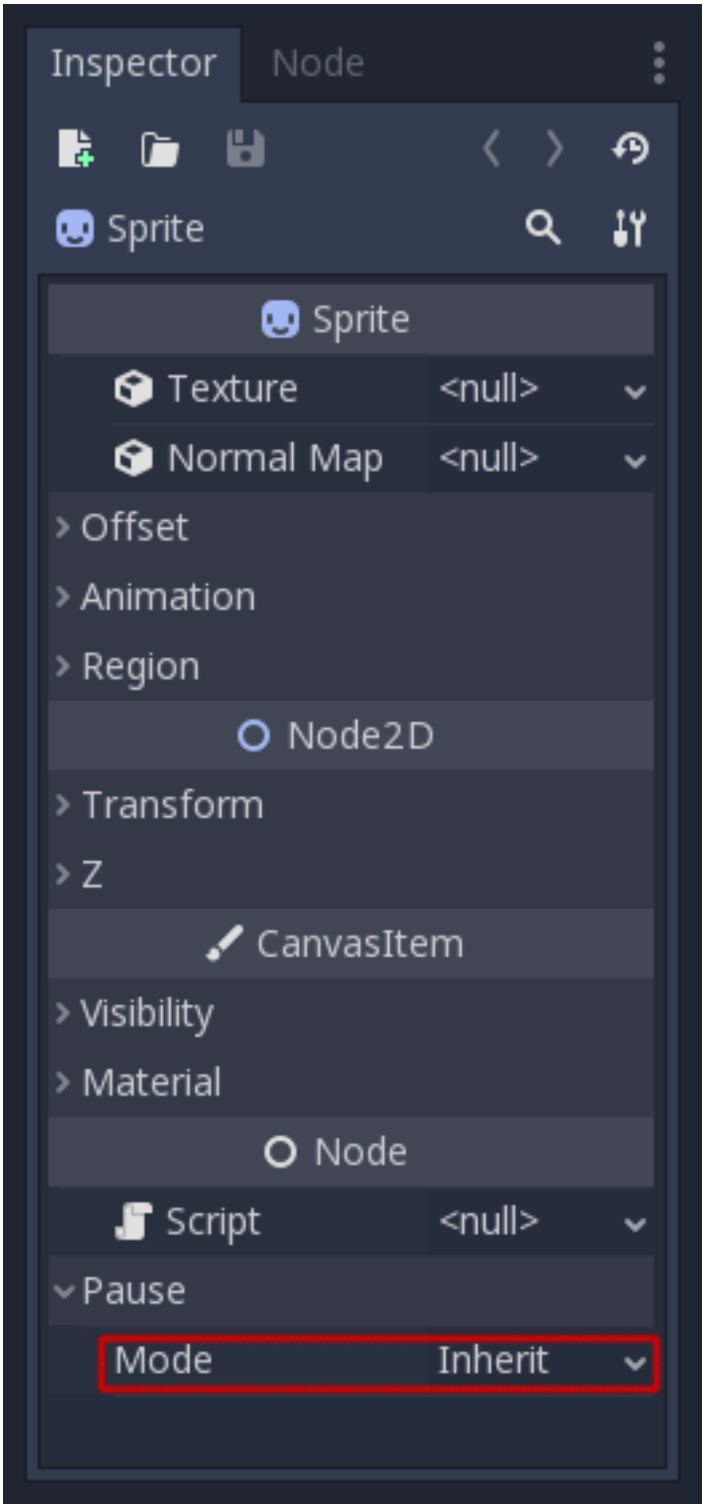
Doing so will have the following behavior:

- 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!).

23.2.3 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:



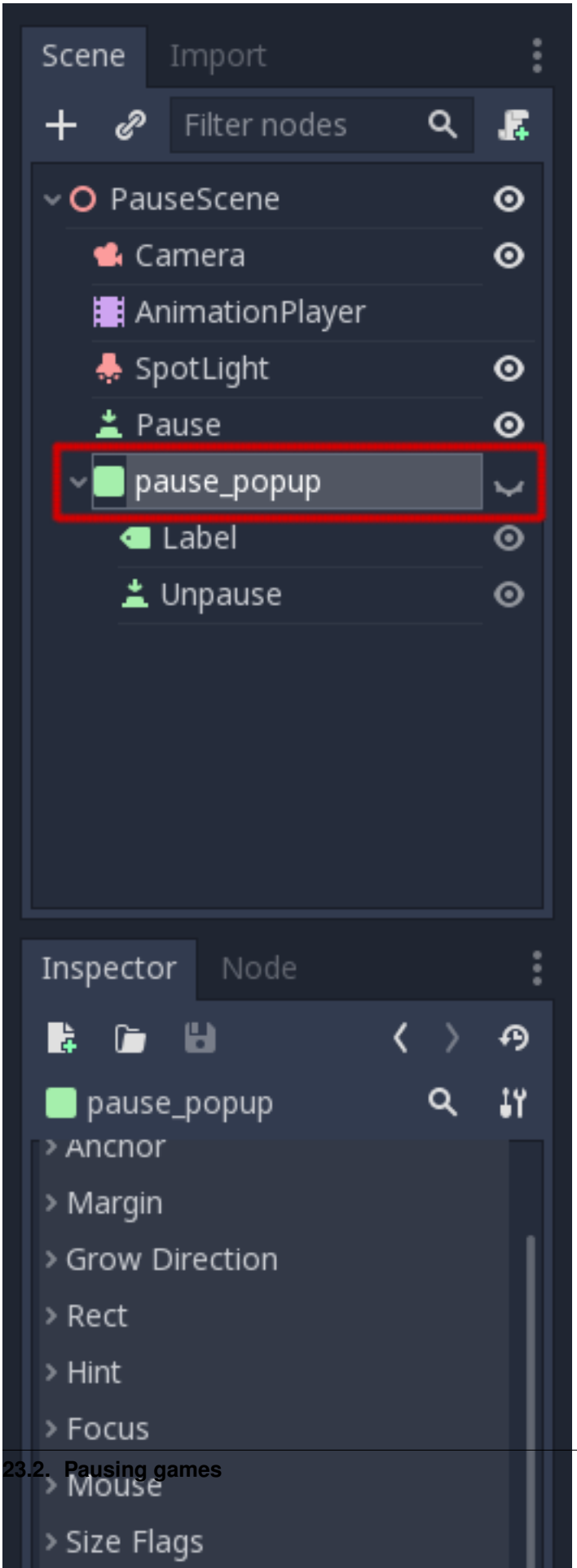
By default all nodes have this property in the "Inherit" state. This means, that they will only process (or not) depending on what this same property is set on the parent node. If the parent is set to "Inherit" , then the grandparent will be checked and so on. Ultimately, if a state can't be found in 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.

23.2.4 Example

An example of this is creating a popup or panel with controls inside, and set its pause mode to "Process" then hide it:


Scene
Import
Filter nodes
PauseScene
Camera
AnimationPlayer
SpotLight
Pause
pause_popup
Label
Unpause
Inspector
Node
pause_popup
Anchor
Margin
Grow Direction
Rect
Hint
Focus
Mouse
Size Flags

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.

GDScript

C#

```
func _on_pause_button_pressed():
    get_tree().paused = true
    $pause_popup.show()
```

```
public void _on_pause_button_pressed()
{
    GetTree().Paused = true;
    ((Control)GetNode("pause_popup")).Show();
}
```

To remove the pause, do the opposite when the pause screen is closed:

GDScript

C#

```
func _on_pause_popup_close_pressed():
    $pause_popup.hide()
    get_tree().paused = false
```

```
public void _on_pause_popup_close_pressed()
{
    ((Control)GetNode("pause_popup")).Hide();
    GetTree().Paused = false;
}
```

And that should be all!

23.3 Binary serialization API

23.3.1 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.

23.3.2 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 24

Debug

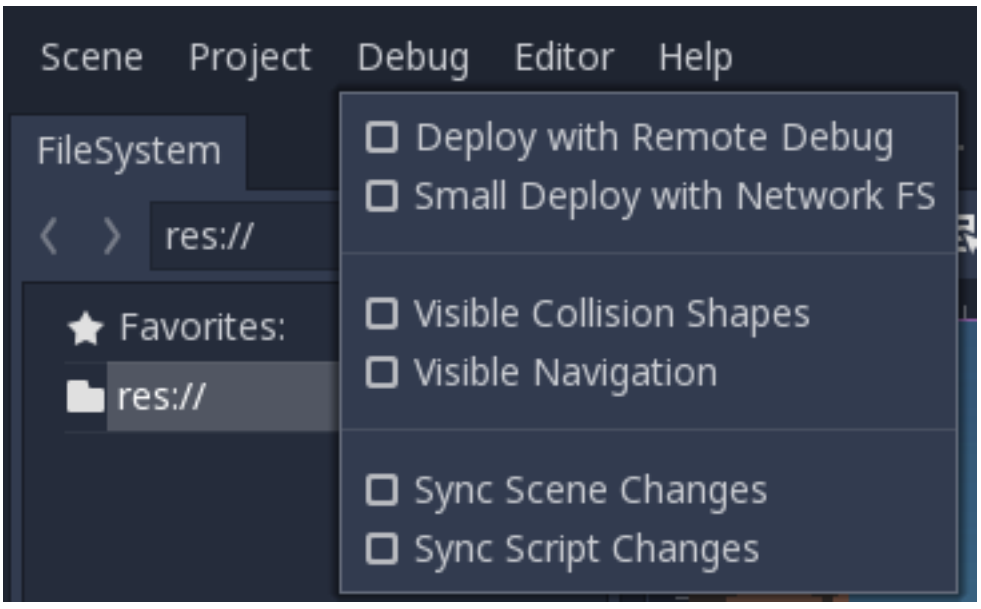
24.1 Overview of debugging tools

24.1.1 Introduction

When developing your game, you would want to test your game and debug when problems occurred. Godot provides several debugging options and tools which aid your debugging process.

24.1.2 Debug options

There are a few options that you can enable when running your game in the editor which can help you in debugging your game. These options are located in `DEBUG` in the main menus.



Here are the descriptions of the options:

Deploy with Remote Debug

When exporting and deploying, the resulting executable will attempt to connect to the IP of your computer, in order to be debugged.

Small Deploy with Network FS

Export or deploy will produce minimal executable. The filesystem will be provided from the project by the editor over the network. On Android, deploy will use the USB cable for faster performance. This option speeds up testing for games with a large footprint.

Visible Collision Shapes

Collision shapes and raycast nodes(for 2D and 3D) will be visible on the running game.

Visible Navigation

Navigation meshes and polygons will be visible on the running game.

Sync Scene Changes

Any changes made to the scene in the editor will be replicated in the running game. When used remotely on a device, this is more efficient with network filesystem.

Sync Script Changes

Any script that is saved will be reloaded on the running game. When used remotely on a device, this is more efficient with network filesystem.

24.1.3 Debugging tools

The `Debugger` is the second option in the bottom panel. Click on it and a new panel occurs.

The `Debugger` provides certain tools under different tabs.

Here are some brief descriptions of the tools:

Debugger

Monitor the game running process.

Errors

Print out errors when running the game.

Profiler

Profiling the performance of the any function call in the running game.

Monitors

Monitors the performance of the running game, such as the fps and physics collisions.

Video Mem

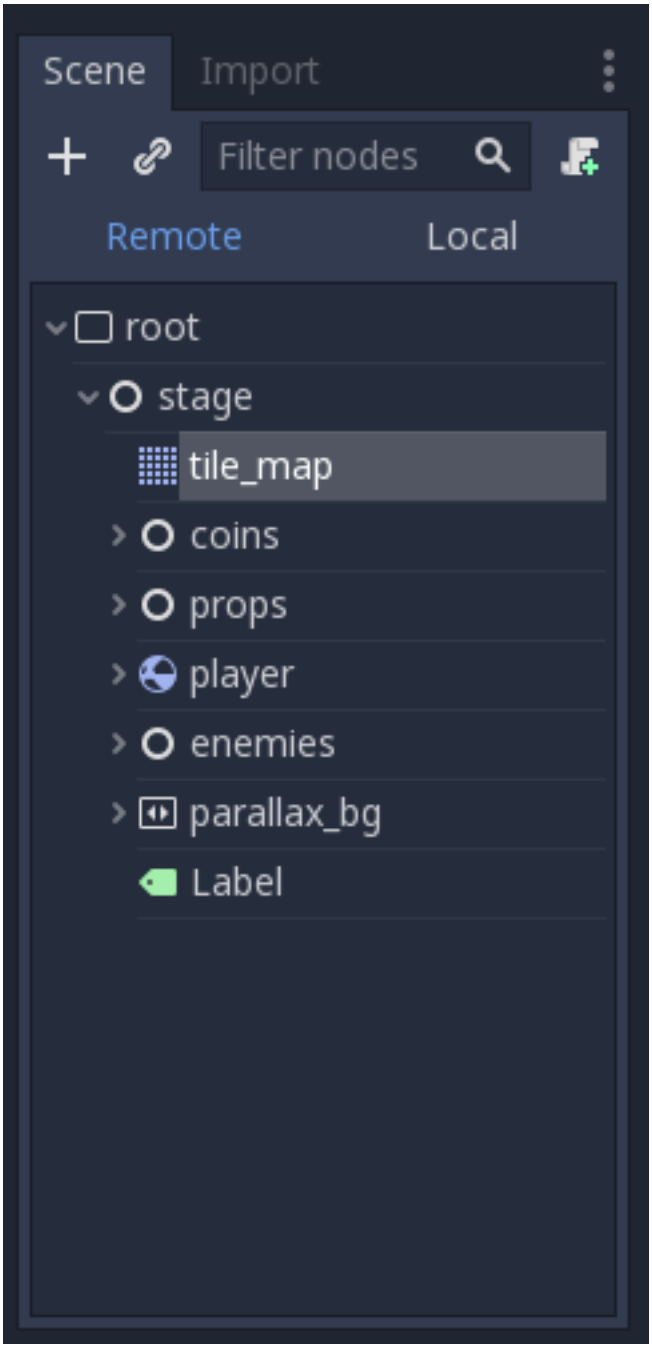
Listing the video memory usage of the running game.

Misc

Miscellaneous options for debug.

24.1.4 Remote in Scene dock

When running your game, a bar will occur at the top of the `Scene` dock. You can switch to `Remote` and inspect or change the nodes' parameters in the running game.



Note: Some editor settings related to debugging can be find in the `Editor Settings`, under Network>Debug and Debugger sections.

CHAPTER 25

Compiling

25.1 Getting the source

25.1.1 Downloading the Godot source code

Before *getting into the SCons build system* and compiling Godot, you need to actually download the Godot source code.

The source code is available on GitHub and while you can manually download it via the website, in general you want to do it via the `git` version control system.

If you don't know much about `git` yet, there are a great number of tutorials available on various websites.

In general, you need to install `git` and/or one of the various GUI clients.

Afterwards, to get the latest development version of the Godot source code (the unstable `master` branch), you can use `git clone`.

If you are using the `git` command line client, this is done by entering the following in a terminal:

```
git clone https://github.com/godotengine/godot.git
```

For any stable release, visit the release page and click on the link for the release you want. You can then download and extract the source from the download link on the page.

There are also generally branches besides `master` for each major version.

After downloading the Godot source code, you can *continue to compiling Godot*.

25.2 Introduction to the buildsystem

25.2.1 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.

25.2.2 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.

25.2.3 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
```

(continues on next page)

(continued from previous page)

```
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.

25.2.4 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.

25.2.5 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
```

25.2.6 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.

25.2.7 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.

25.2.8 Other build options

There are several other build options that you can use to configure the way Godot should be built (compiler, debug options, etc.) as well as the features to include/disable.

Check the output of `scons --help` for details about each option for the version you are willing to compile.

25.2.9 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
```

(continues on next page)

(continued from previous page)

```
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.

25.3 Compiling for Windows

25.3.1 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.

25.3.2 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.

25.3.3 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).

25.3.4 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***.

25.3.5 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`.

25.3.6 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
```

25.3.7 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 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.

25.3.8 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	`pacman -S scons mingw-w64-gcc`
Debian / Ubuntu	`apt-get install scons mingw-w64`
Fedora	`dnf install scons mingw32-gcc-c++ mingw64-gcc-c++`
macOS	`brew install scons mingw-w64`
Mageia	`urpmi scons mingw32-gcc-c++ mingw64-gcc-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>
```

25.3.9 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
```

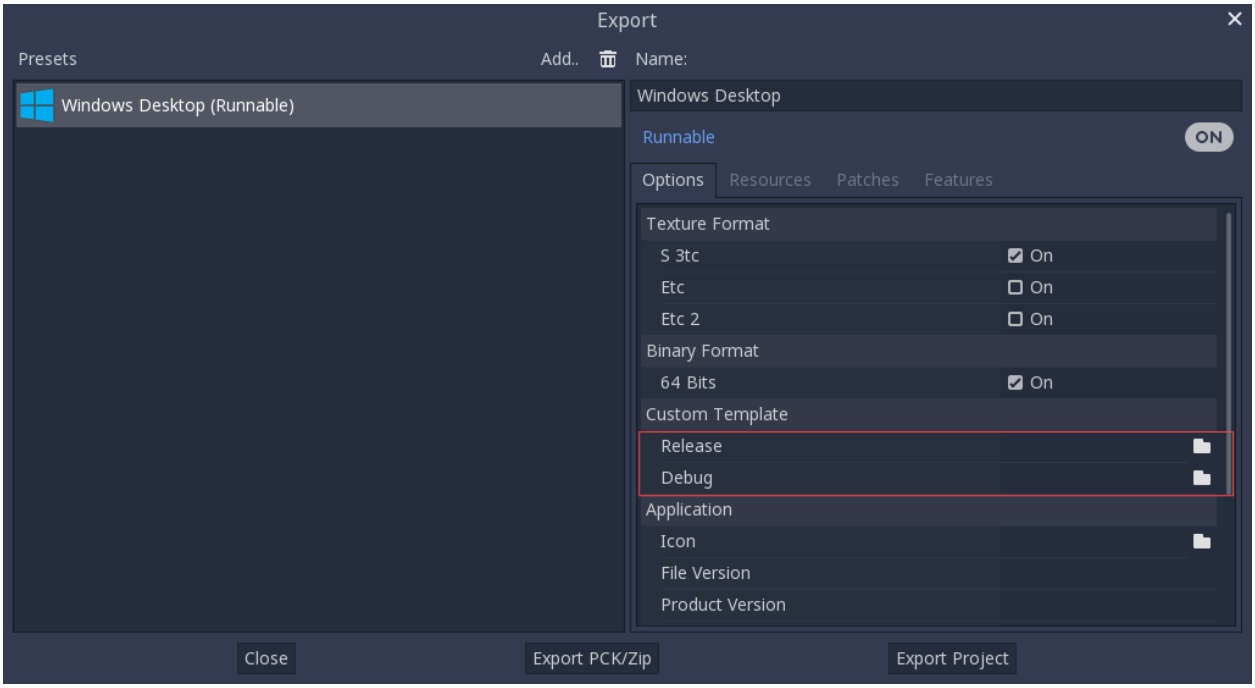
If you plan on replacing the standard templates, copy these to:

```
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.

25.4 Compiling for X11 (Linux, *BSD)

25.4.1 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)
- *Optional* - libudev (build with `udev=yes`)
- *Optional* - yasm (for WebM SIMD optimizations)

Distro-specific oneliners

Arch	`pacman -S scons libxcursor libxinerama libxi libxrandr mesa glu alsa-lib 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`
Fedora	`sudo dnf install scons pkgconfig libX11-devel libXcursor-devel libXrandr-devel libXinerama-devel \ libXi-devel mesa-libGL-devel alsa-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 task-c++-devel 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 alsa-devel libpulse-devel freetype-devel openssl-devel libudev-devel libGLU1`
Solus	`sudo eopkg install -c system.devel scons`

25.4.2 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.

25.4.3 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.

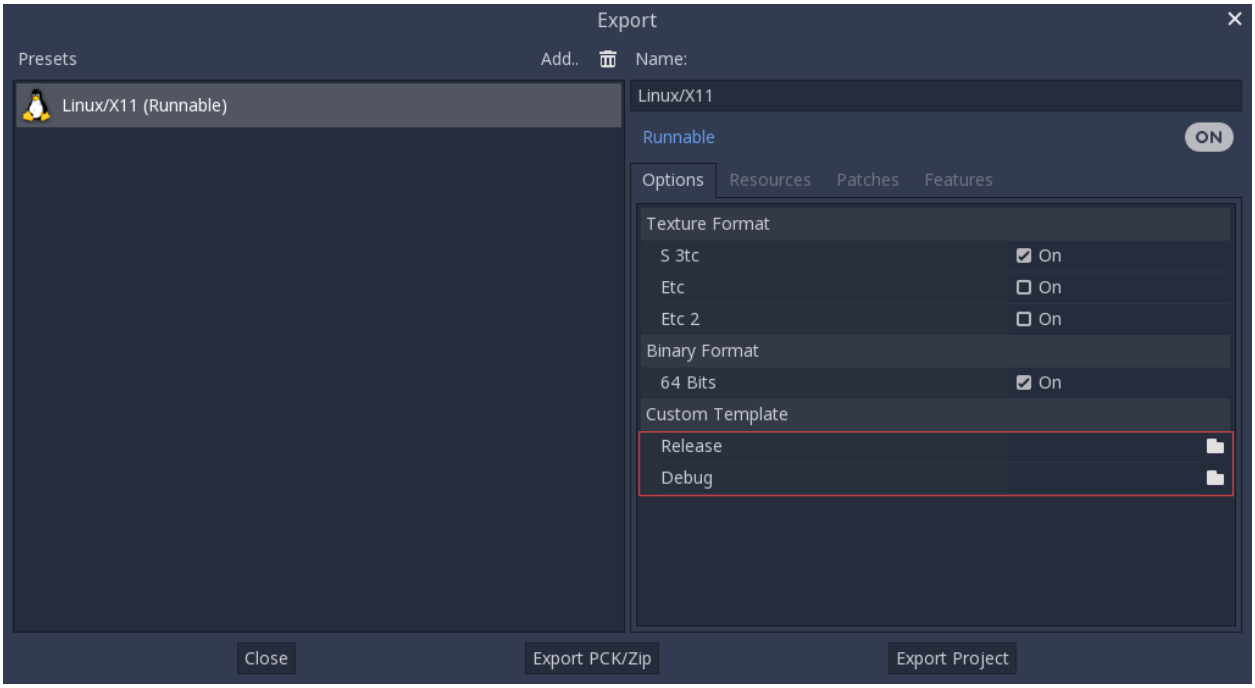
To create standard export templates, the resulting files must be copied to:

```
/home/[username]/.local/share/godot/templates/[gd-version]/
```

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.

25.5 Compiling for OSX

25.5.1 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)

25.5.2 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
```

25.5.3 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
```

25.5.4 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
```

25.6 Compiling for Android

25.6.1 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.

25.6.2 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

25.6.3 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 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")
```

25.6.4 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.

25.6.5 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.

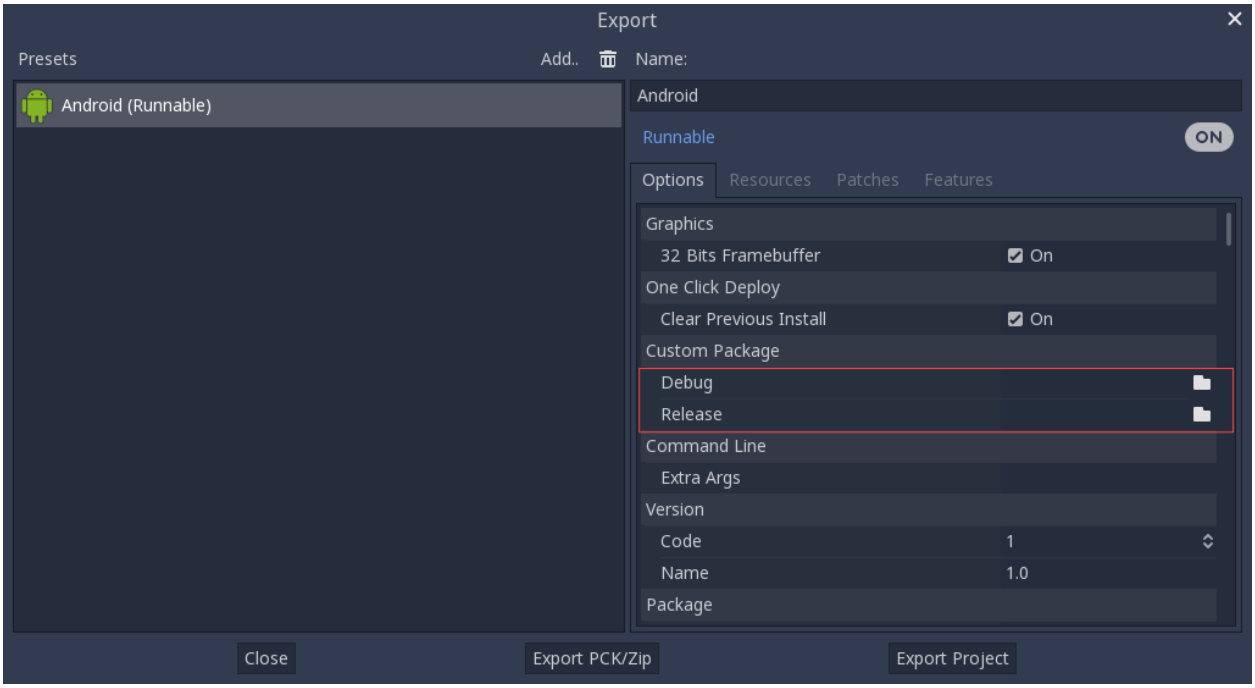
When exporting your game, Godot opens the APK, changes a few things inside and adds your files.

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]/.local/share/godot/templates/[gd-version]/`
- 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.

25.6.6 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.

25.7 Compiling for iOS

25.7.1 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.

25.7.2 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
```

25.7.3 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.

25.8 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.

25.8.1 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.

25.8.2 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.

25.8.3 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`.

25.8.4 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`.

25.9 Compiling for Universal Windows Platform

25.9.1 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.

25.9.2 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
```

25.9.3 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 `[versionstring]\templates` folder in Godot settings path, where *versionstring* is the version of Godot you have compiled the export templates for - e.g. *3.0.alpha* for the alpha version of Godot 3. If you don't want to replace the templates, you can set the "Custom Package" property in the export window.

25.9.4 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 Unversal)" 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 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.

25.10 Compiling for the Web

25.10.1 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
- Python 2.7+ or Python 3.5+
- SCons build system

25.10.2 Building export templates

Before starting, confirm that the Emscripten configuration file exists and specifies all settings correctly. This file is available as `~/.emscripten` on UNIX-like systems and `%USERPROFILE%\.emscripten` on Windows. It's usually written by the Emscripten SDK, e.g. when invoking `emsdk activate latest`, or by your package manager. It's also created when starting Emscripten's `emcc` program if the file doesn't exist.

Open a terminal and navigate to the root directory of the engine source code. Then 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
```

By default, the *JavaScript singleton* will be built into the engine. Since `eval()` calls can be a security concern, the `javascript_eval` option can be used to build without the singleton:

```
scons platform=javascript tools=no target=release javascript_eval=no
scons platform=javascript tools=no target=release_debug javascript_eval=no
```

The engine will now be compiled to WebAssembly by Emscripten. Once finished, 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
```

25.10.3 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 is used. 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 `/`.

25.11 Compiling with Mono

25.11.1 Requirements

- Mono (the required version is listed here: *Introduction*)
- MSBuild
- pkg-config

25.11.2 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`.

25.11.3 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.

25.11.4 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.

As always, `godot` refers to the compiled Godot binary, so if it isn't in your PATH, you need to give the full path to the executable, e.g. if it is located in the `bin` subfolder, it becomes `bin/godot`.

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.

25.11.5 Examples

Example (Windows)

```
# Build temporary binary
scons p=windows tools=yes module_mono_enabled=yes mono_glue=no
# Generate glue sources
bin\godot.windows.tools.64.mono --generate-mono-glue modules/mono/glue

### Build binaries normally
# Editor
scons p=windows target=release_debug tools=yes module_mono_enabled=yes
# Export templates
scons p=windows target=debug tools=no module_mono_enabled=yes
scons p=windows target=release tools=no module_mono_enabled=yes
```

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.

25.12 Packaging Godot

Godot has features to make it easier to distribute and to package it for application repositories.

25.12.1 Default behaviour

By 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.

25.12.2 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.

25.12.3 Self contained mode

The self contained mode can be used to package Godot for distribution systems where it doesn't live at a fixed location. If the editor finds a `._sc_` file in the directory the executable is located in, Godot will continue in "self contained mode". On Windows, the file name to use is `_sc_` (without the preceding dot).

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 `project.godot`, 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 26

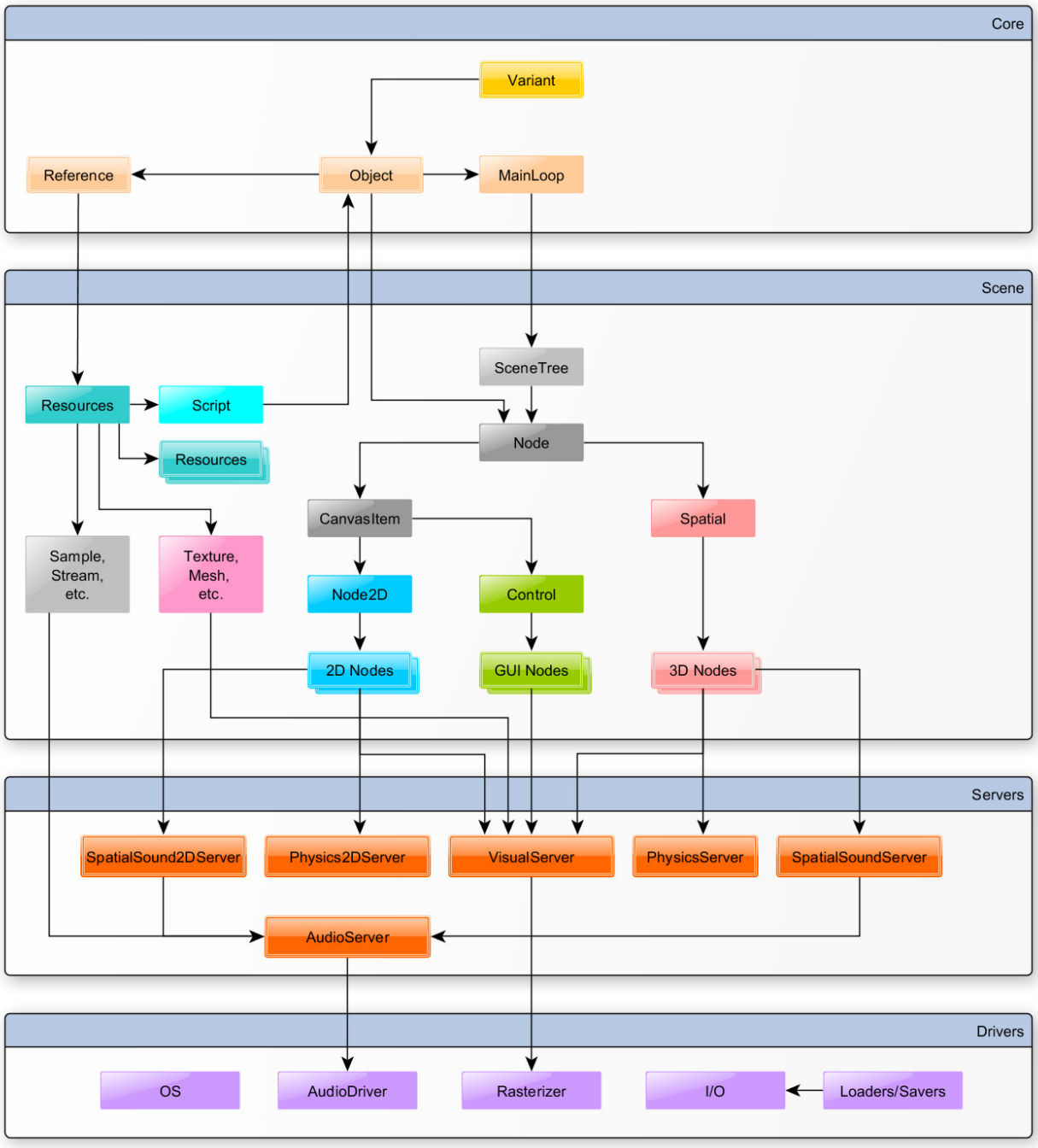
Engine development

26.1 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.

26.1.1 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.



26.1.2 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
```

(continues on next page)

(continued from previous page)

```
> run -e
```

Or:

```
$ gdb godot
> run -e -path ~/myproject
```

26.2 Configuring an IDE

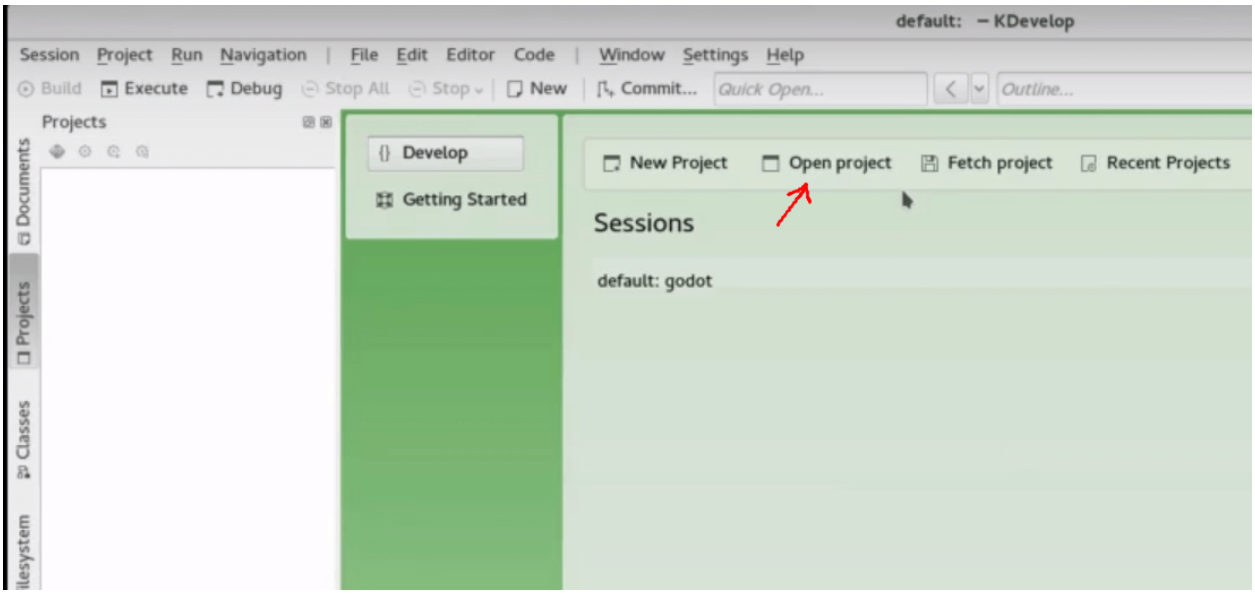
We assume that you already cloned and *compiled* Godot.

26.2.1 Kdevelop

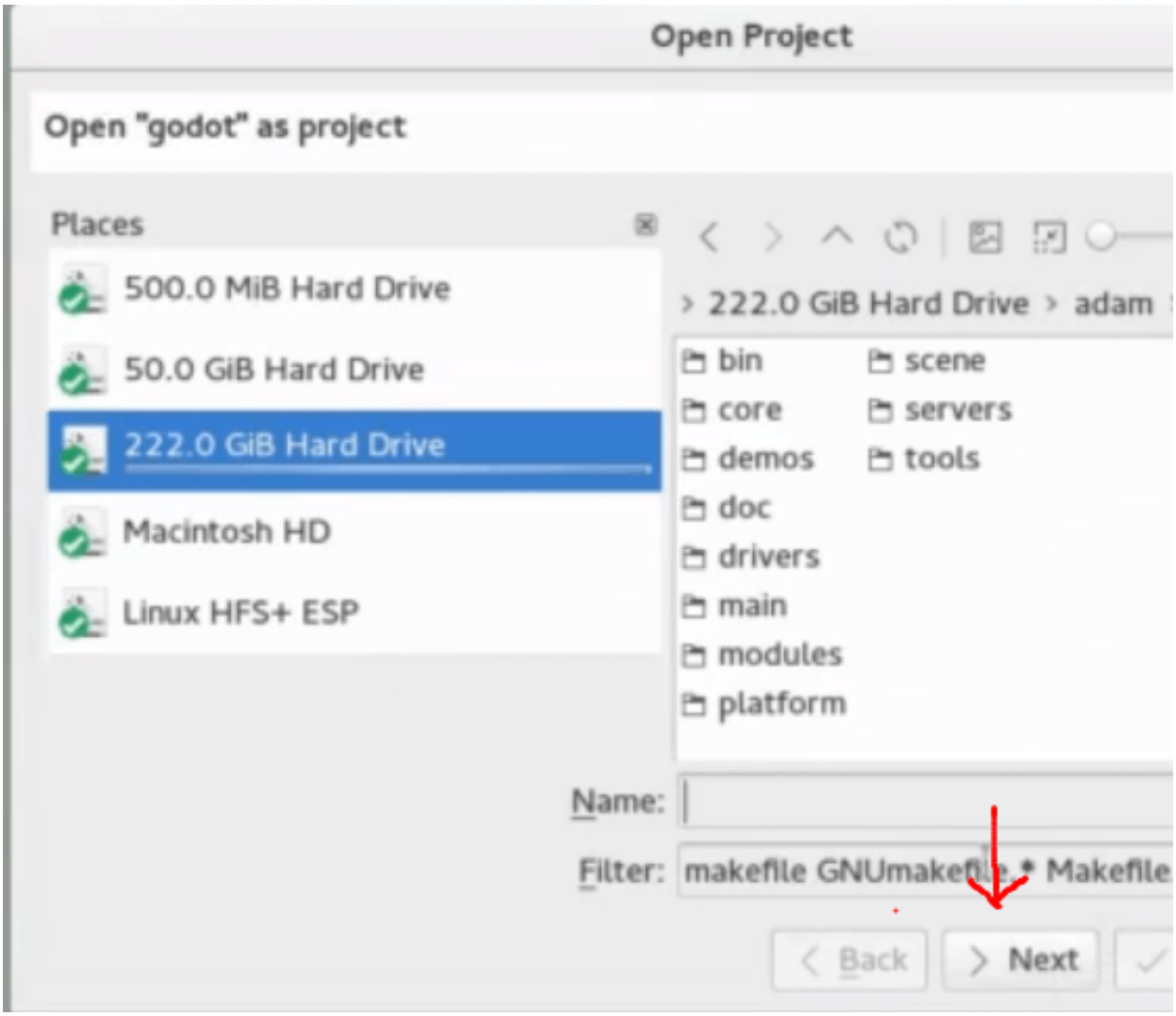
It is a free, open source IDE (Integrated Development Environment) for Linux, Solaris, FreeBSD, Mac OS X and other Unix flavors.

You can find a video tutorial here. Or you may follow this text version tutorial.

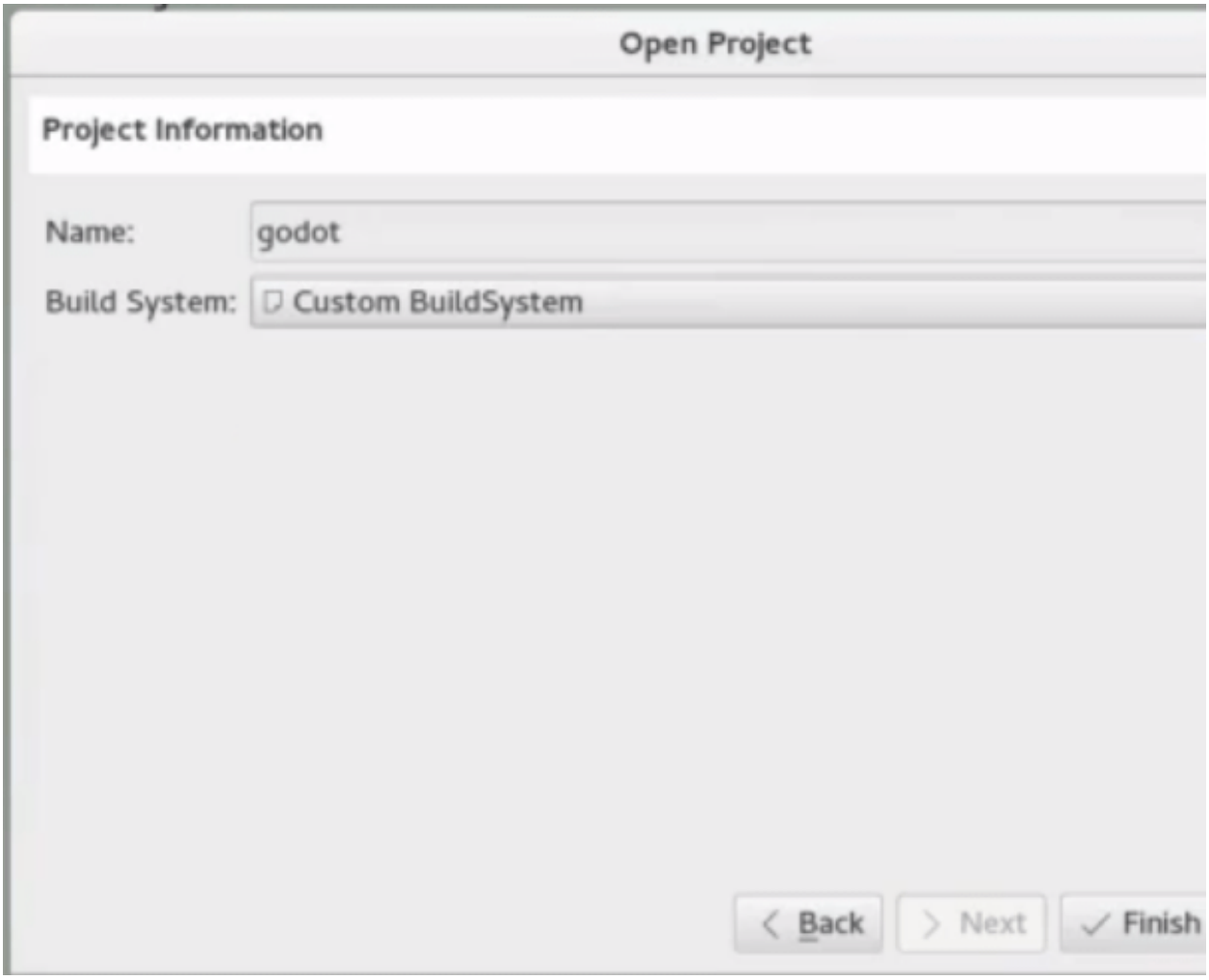
Start by opening Kdevelop and choosing "open project".



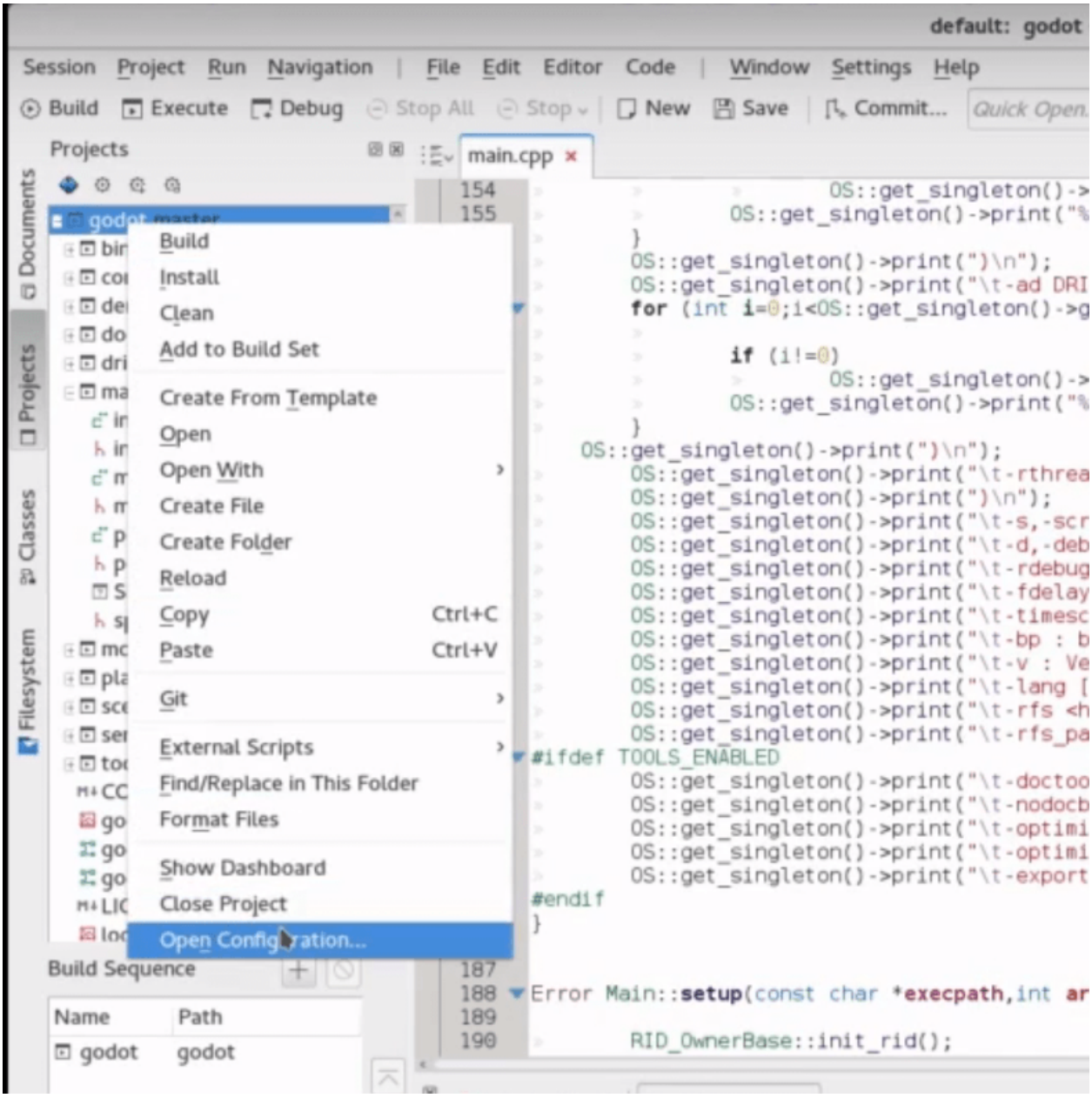
Choose the directory where you cloned Godot.



For the build system, choose “custom build system”.

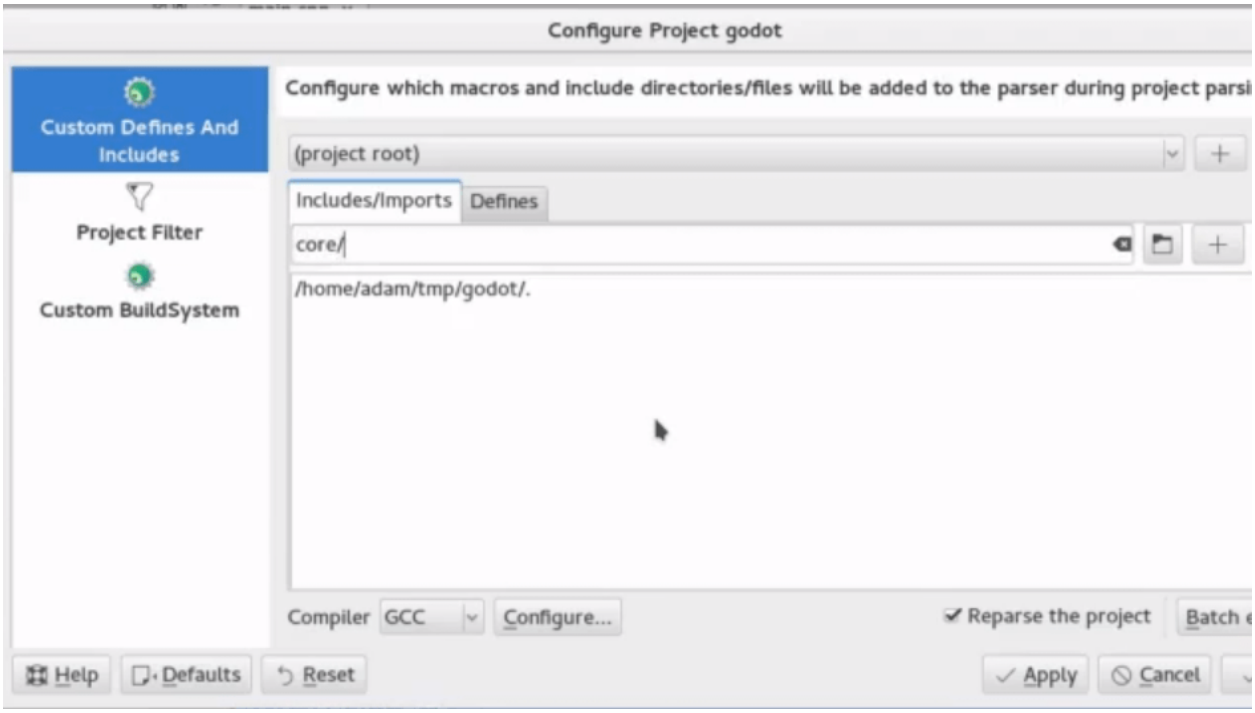


Now that the project has been imported, open the project configuration.

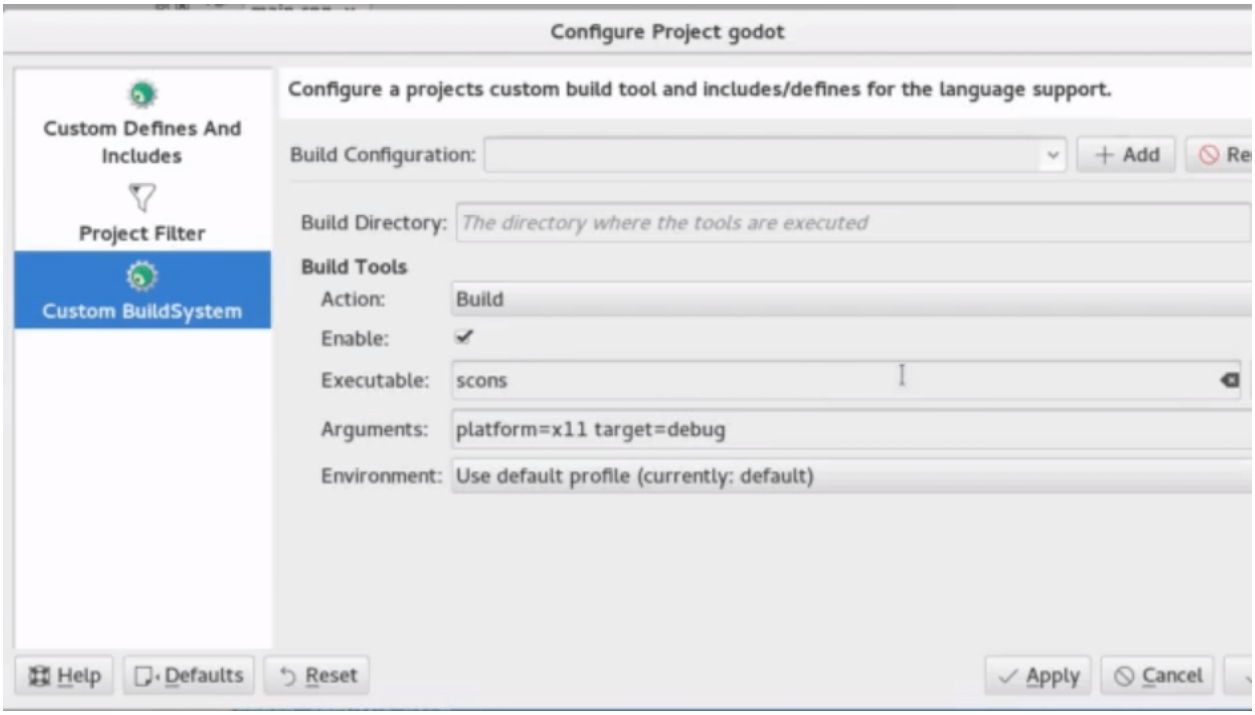


Add the following includes/imports:

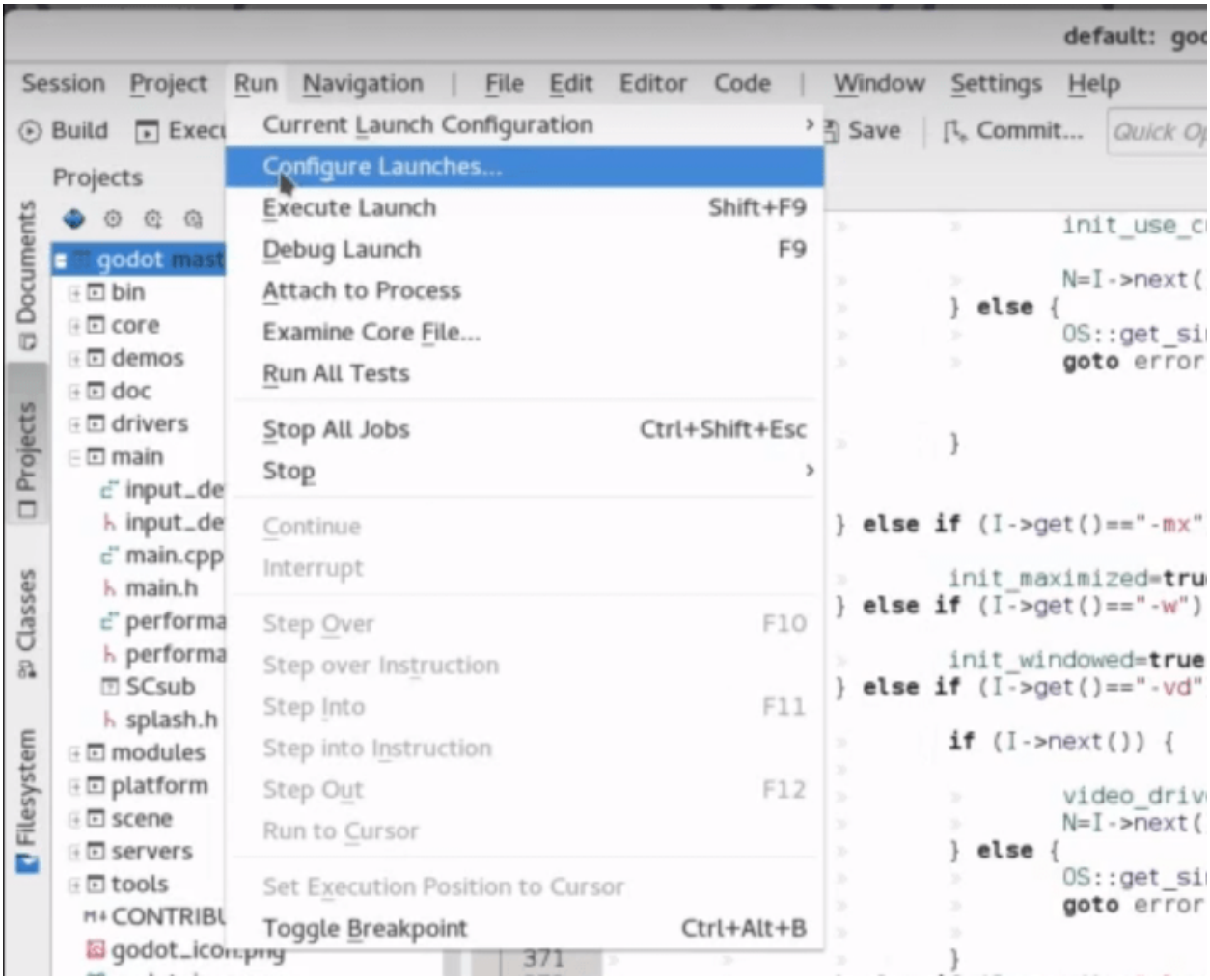
```
.  // a dot to indicate the root of the Godot project
core/
core/os/
core/math/
tools/
drivers/
platform/x11/  // make that platform/osx/ if you're using OS X
```



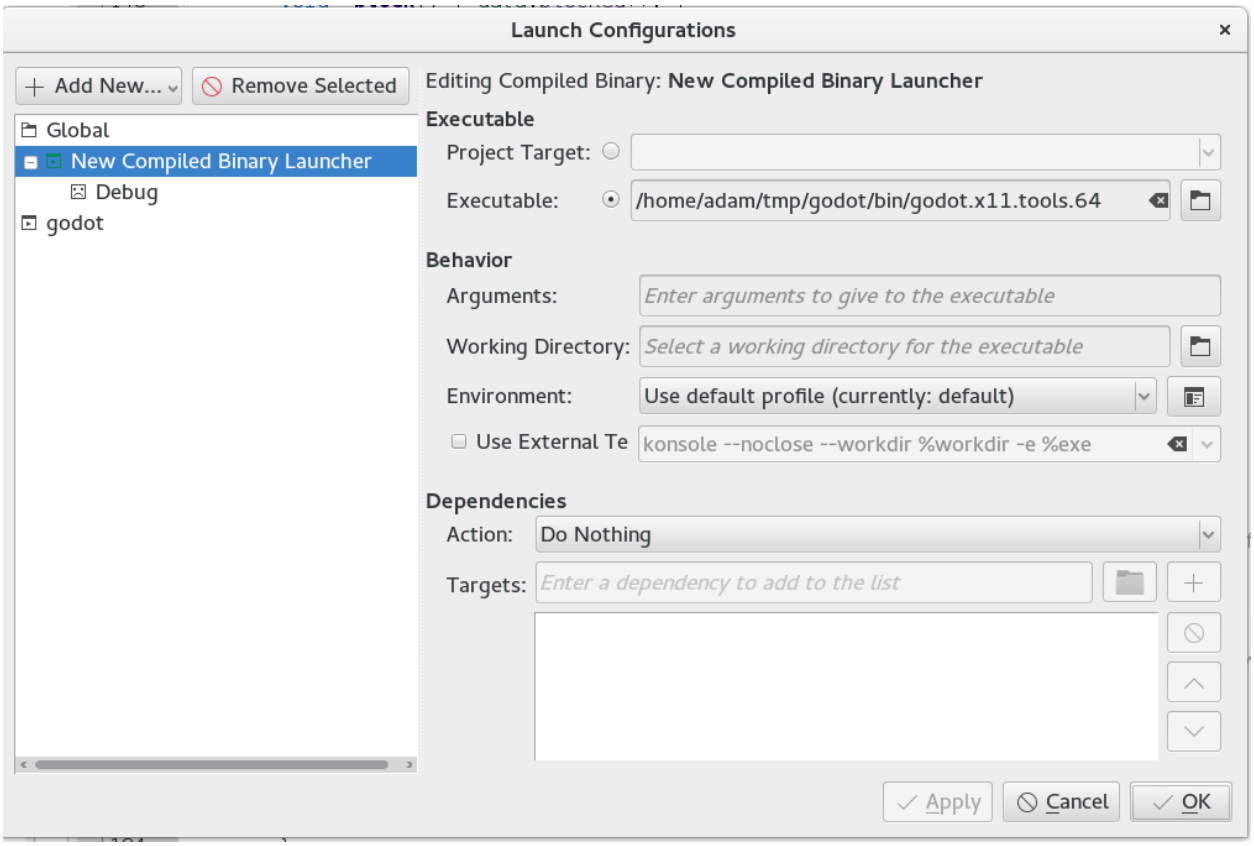
Apply the changes then switch to the "Custom Buildsystem" tab. Leave the build directory blank. Enable build tools and add `scons` as the executable and add `platform=x11 target=debug` (`platform=osx` if you're on OS X).



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).

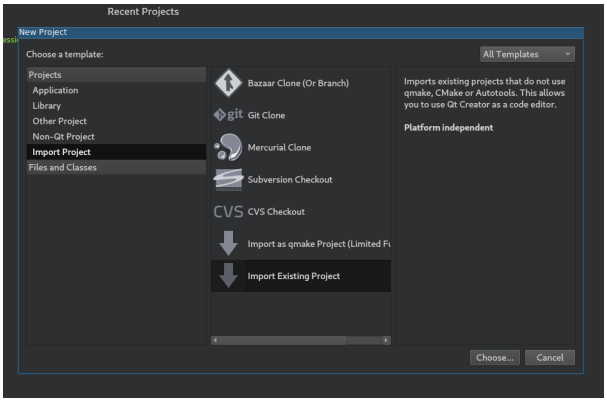


That's it! Now you should be good to go :)

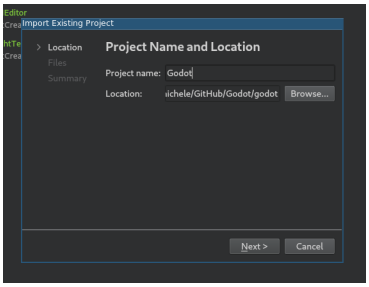
26.2.2 Qt Creator

Importing the project

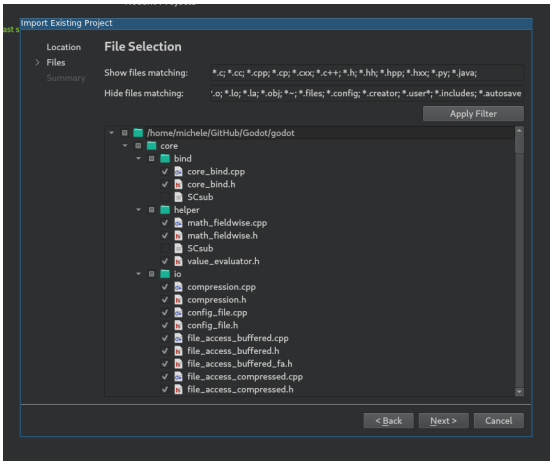
- Choose *New Project -> Import Project -> Import Existing Project*.



- 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".

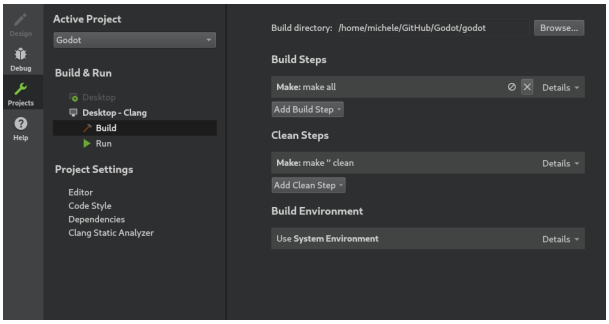


- Click *Finish*.
- Add a line containing `.` to *project_name.includes* to get working code completion.

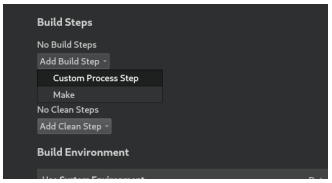
Build and run

Build configuration:

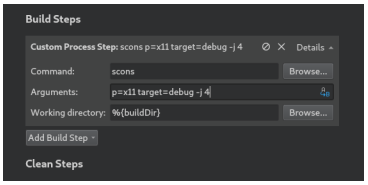
- Click on *Projects* and open the *Build* tab.
- Delete the pre-defined `make` build step.



- Click *Add Build Step -> Custom Process Step*.

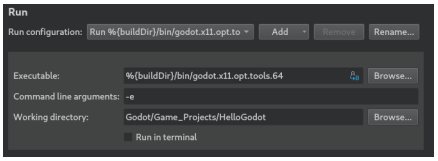


- Type `scons` in the *Command* field.
- Fill the *Arguments* field with your compilation options. (e.g.: `p=x11 target=debug -j 4`)



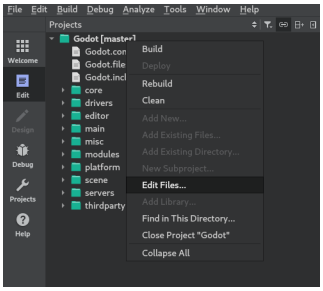
Run configuration:

- Open the *Run* tab.
- Point the *Executable* to your compiled Godot binary (e.g: `%{buildDir}/bin/godot.x11.opt.tools.64`)
- 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.

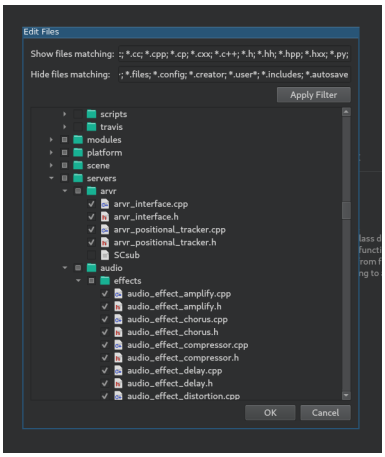


Updating Sources after pulling latest commits

As a developer you usually want to frequently pull the latest commits from the upstream git repository or a specific fork etc. However this brings a little problem with it: as the development continues, source files (and folders) are added or removed. These changes needs to be reflected in your project files for Qt Creator too, so you continue to have a nice experience coding in it. A simple way to check these things, is to right click at your root folder in the "Projects View" and click on "Edit files..."



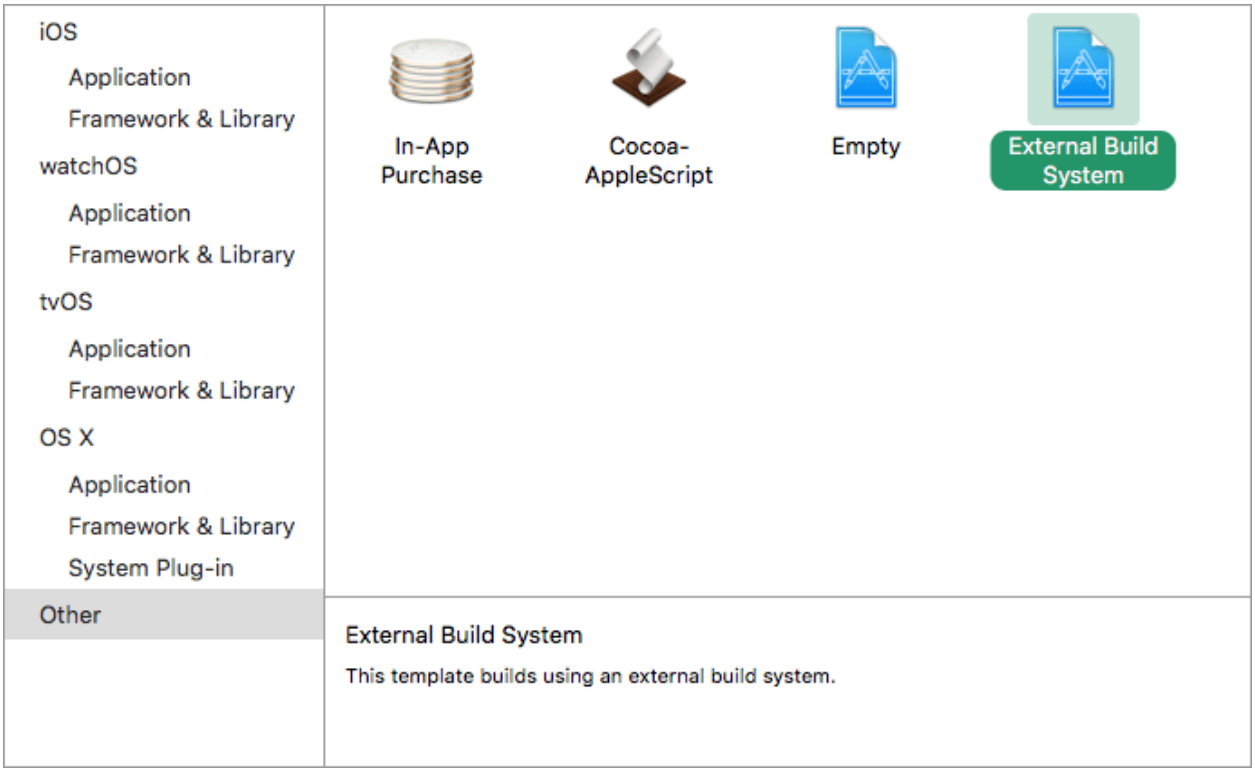
Now a new dialog should appear that is similar in functionality to the one in the third step of the "Importing the project" section. Here now you can check whether you want to add/remove specific files and/or folders. You can chose by clicking with your mouse or just simply by clicking the "Apply Filter" button. A simple click on "Ok" and you're ready to continue your work.



26.2.3 Xcode

Project Setup

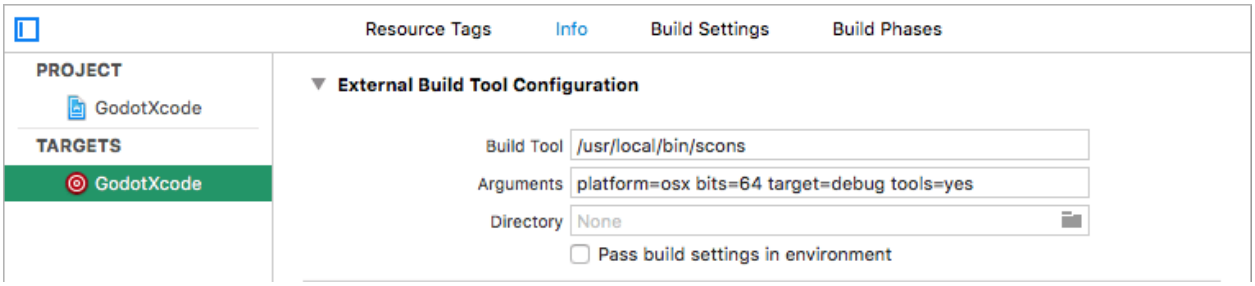
- Create an Xcode external build project anywhere



- Set the *Build tool* to the path to scons

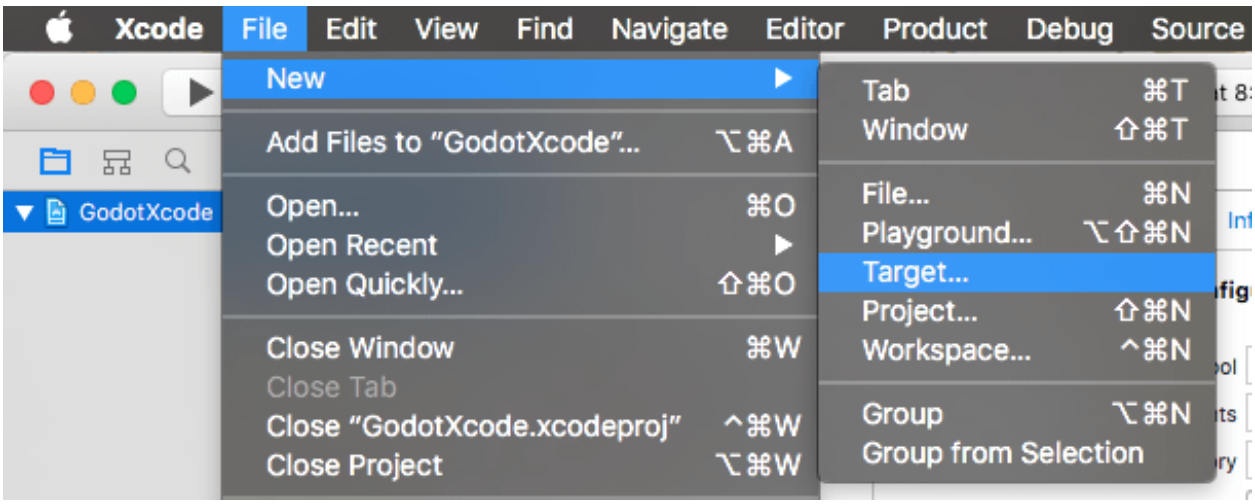
Modify Build Target's Xcode Info Tab:

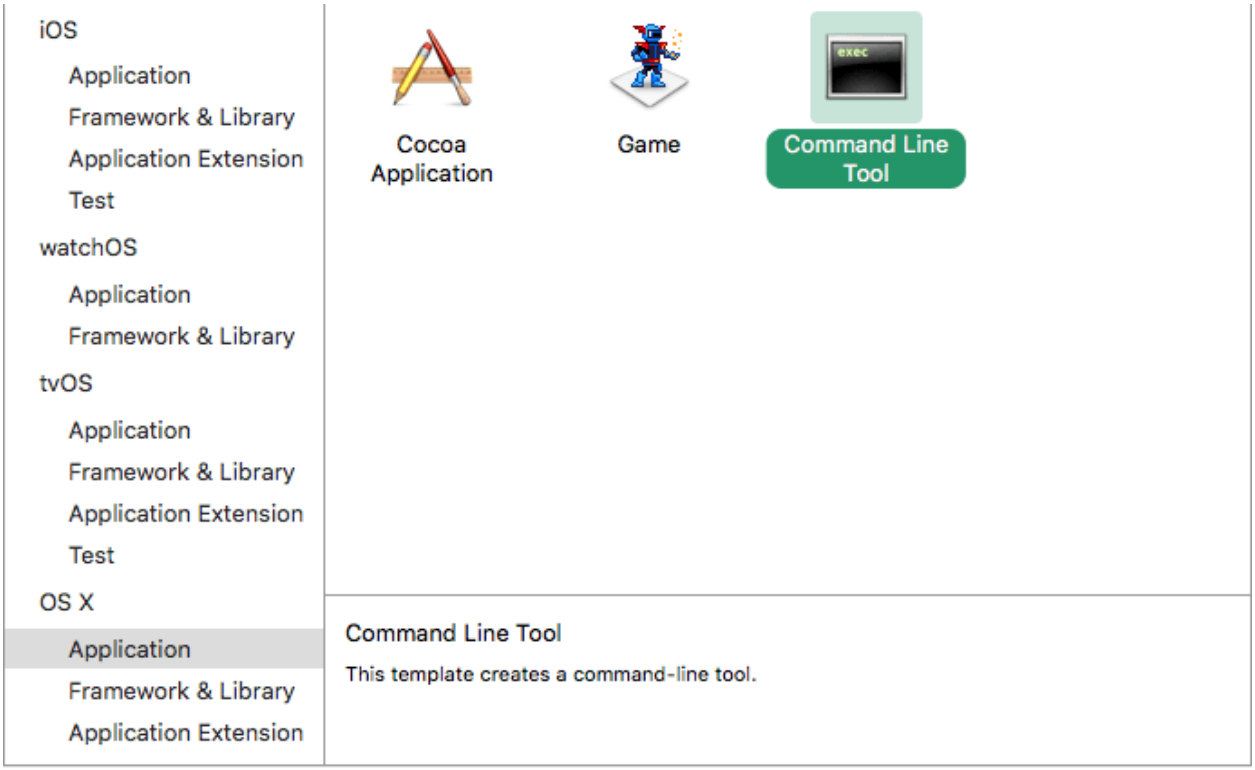
- Set *Arguments* to something like: platform=osx tools=yes bits=64 target=debug
- Set *Directory* to the path to Godot's source folder. Keep it blank if project is already there.
- You may uncheck *Pass build settings in environment*



Add a Command Line Target:

- Go to Xcode File > New > Target. . . and add a new Xcode command line target



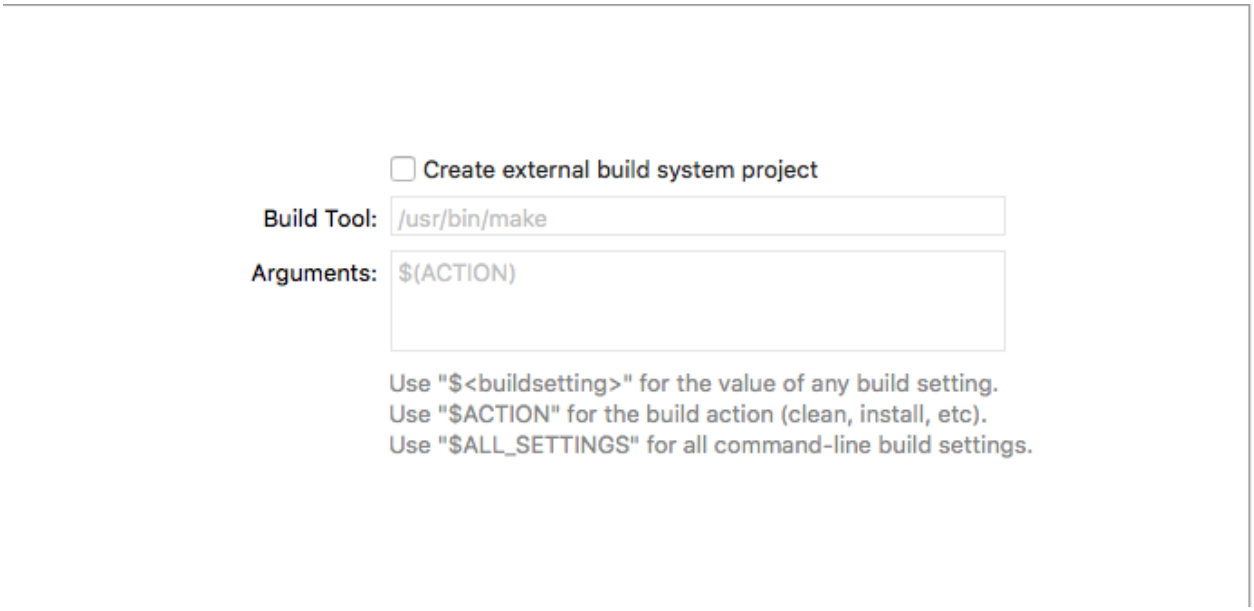


- Name it something so you know not to compile with this target

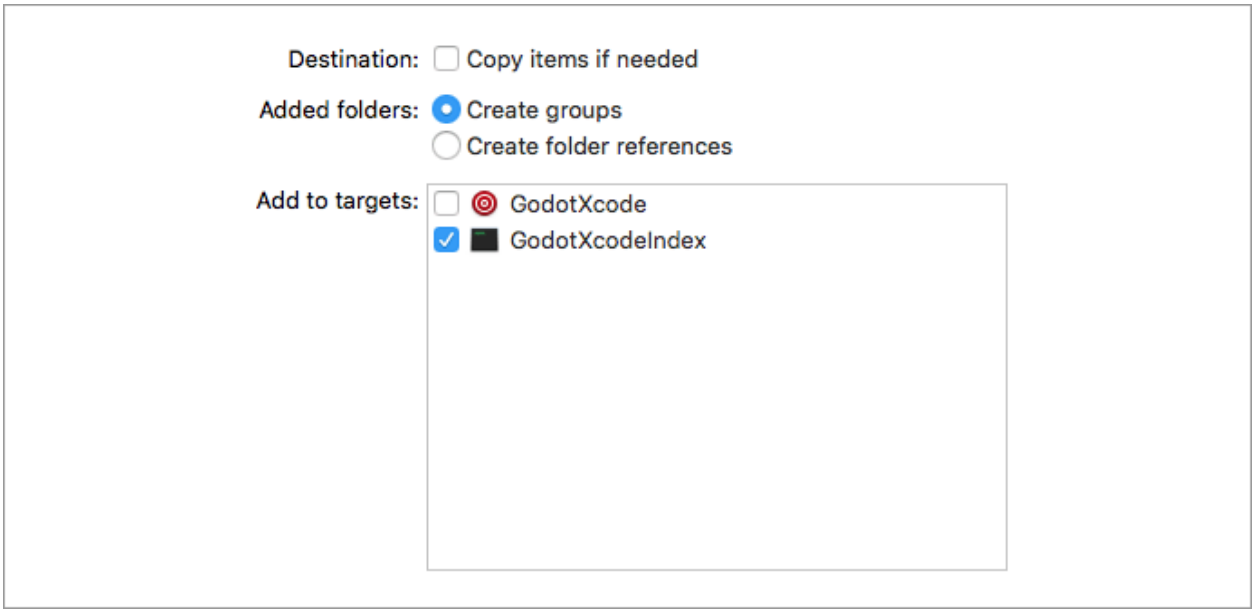
- e.g. `GodotXcodeIndex`
- Goto the newly created target's *Build Settings* tab and search for *Header Search Paths*
- Set *Header Search Paths* to an absolute path to Godot's source folder
- Make it recursive by adding two *'s to the end of the path
- e.g. `/Users/me/repos/godot-source/\**`

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

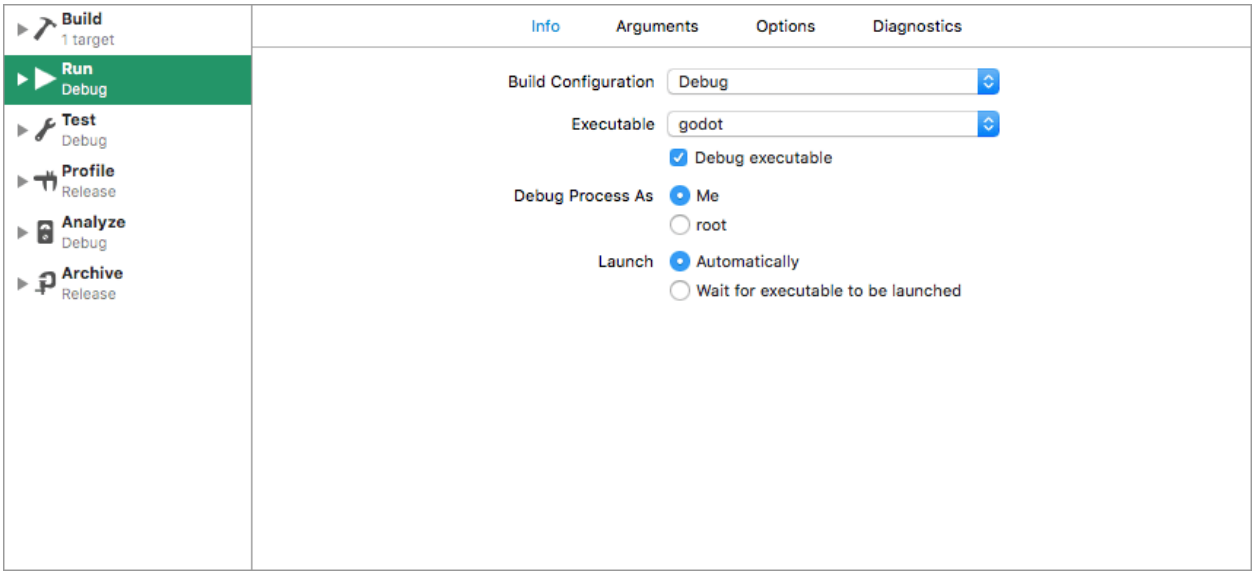
Edit Build Scheme of External Build Target:

- 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!

26.3 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.

26.3.1 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

26.3.2 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 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 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, 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.

26.3.3 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. Use it like this:

```
DVector<int>
```

DVector is 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

26.3.4 Containers

Godot provides also a set of common containers:

- Vector
- List
- Set
- Map

They are 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

26.3.5 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

26.3.6 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 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

26.3.7 Math types

There are several linear math types available in the core/math directory.

References:

- core/math

26.3.8 NodePath

This is a special datatype used for storing paths in a scene tree and referencing them fast.

References:

- core/node_path.h

26.3.9 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

26.4 Variant class

26.4.1 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

26.4.2 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

26.5 Object class

26.5.1 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

26.5.2 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

26.5.3 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);
```

26.5.4 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.

26.5.5 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.

26.5.6 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 small binary size is ideal, such as HTML5 or consoles (with low memory footprint).

26.5.7 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"))
```

26.5.8 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

26.5.9 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

26.5.10 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

26.5.11 Resource saving

Saving a resource can be done with the resource saver API:

```
ResourceSaver::save("res://someresource.res", instance)
```

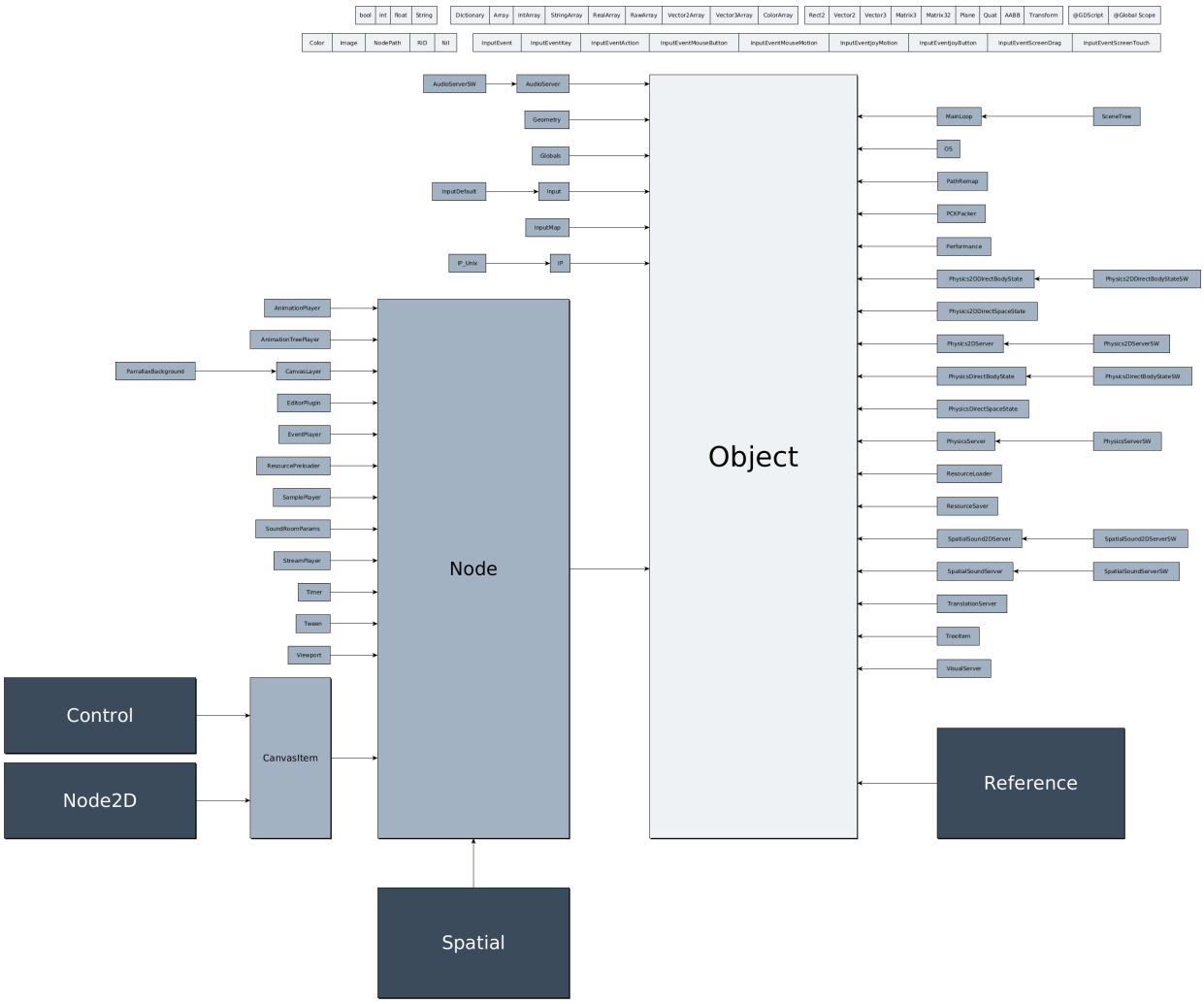
Instance will be saved. Sub resources that have a path to a file will be saved as a reference to that resource. Sub resources without a path will be bundled with the saved resource and assigned sub-IDs, like "res://someresource.res::1". This also helps to cache them when loaded.

References:

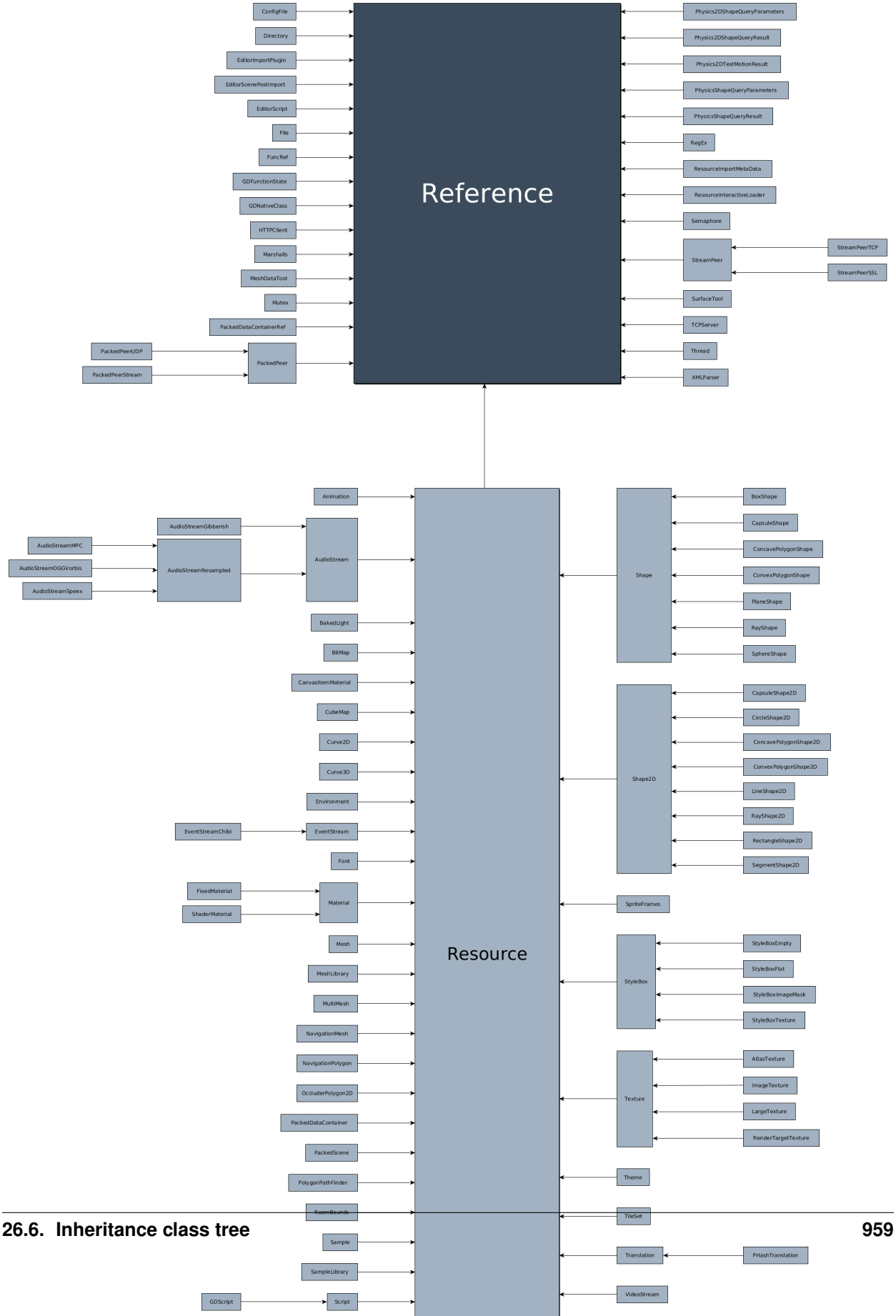
- core/io/resource_saver.h

26.6 Inheritance class tree

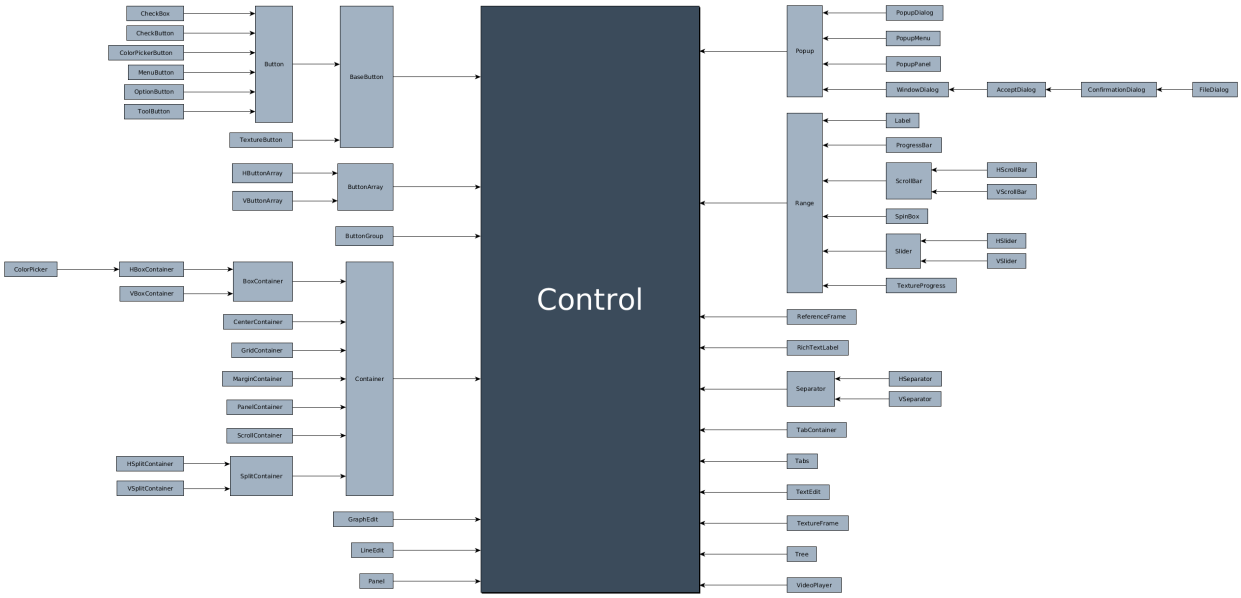
26.6.1 Object



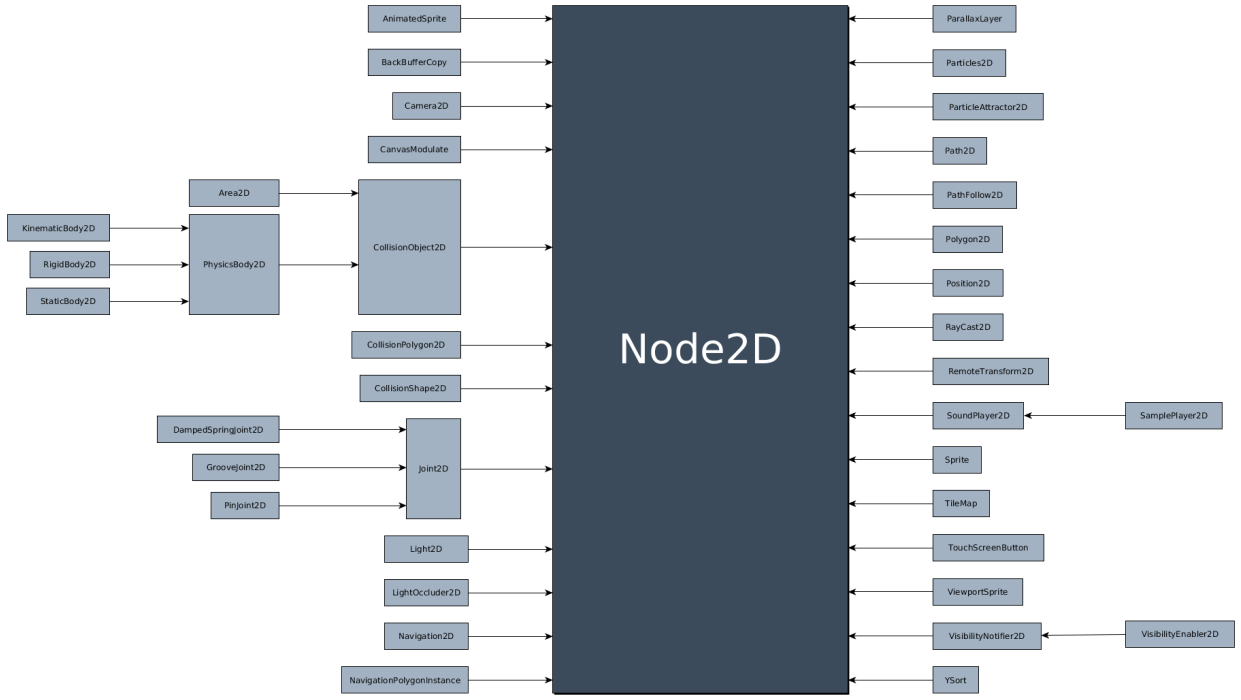
26.6.2 Reference



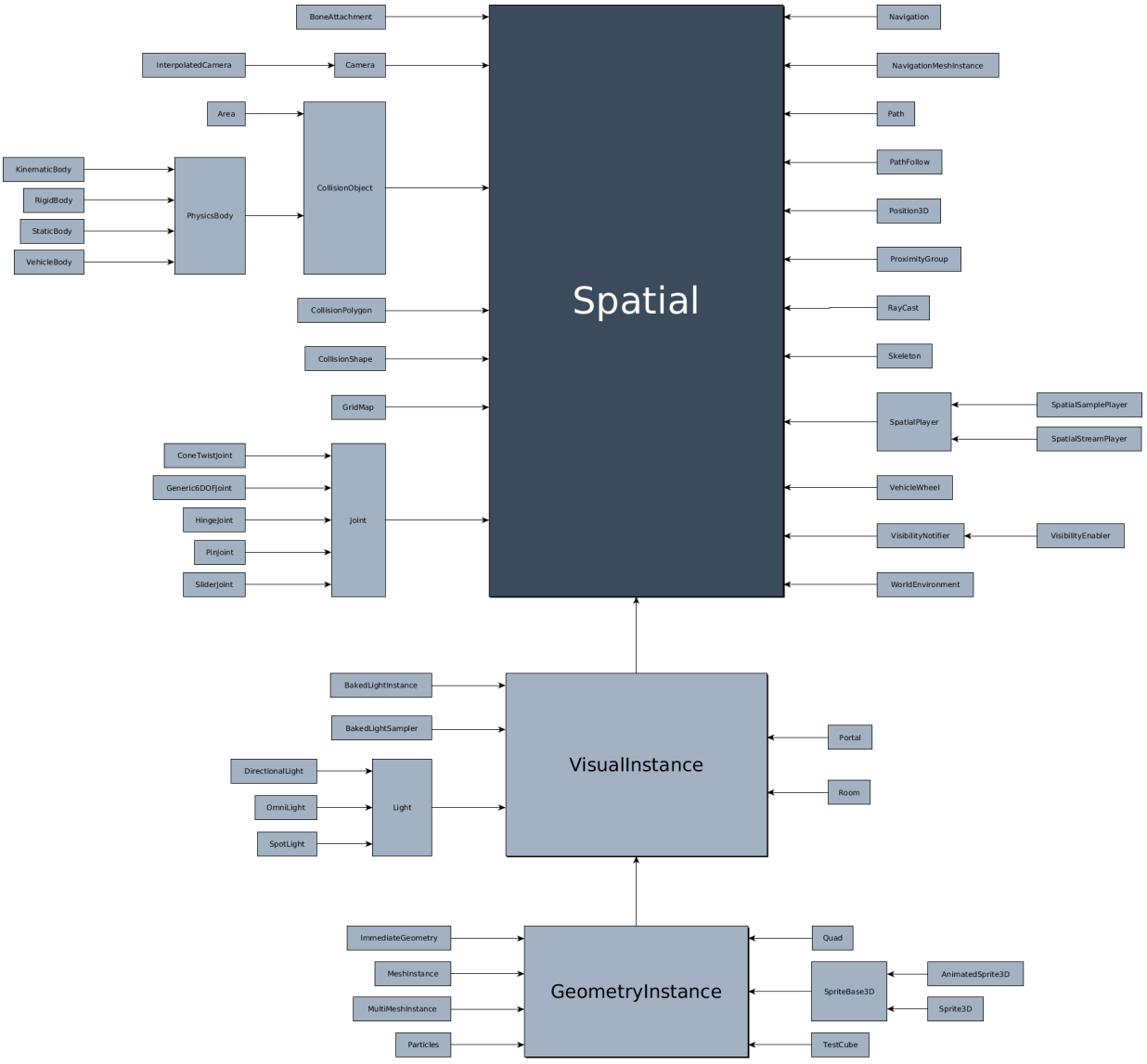
26.6.3 Control



26.6.4 Node2D



26.6.5 Spatial



Source files: `class_tree.zip`.

26.7 Custom modules in C++

26.7.1 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.

26.7.2 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++.

26.7.3 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();
};
```

(continues on next page)

(continued from previous page)

```
#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() {
```

(continues on next page)

(continued from previous page)

```
        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") # 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 contruct 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
```

(continues on next page)

(continued from previous page)

```
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.

26.7.4 Using the module

You can now use your newly created module from any script:

```
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`.

26.7.5 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-of 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 lirary as a new dependency.
```

(continues on next page)

(continued from previous page)

```
# To do so, SCons wants the name of the lib with it 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 you 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 platform=x11 bin/libsummator.x11.tools.
↪64.so
```

26.7.6 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.

26.7.7 Summing up

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 (hopefully positive) 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.

26.8 Custom Resource Format Loaders

26.8.1 Introduction

ResourceFormatLoader is a factory interface for loading file assets. Resources are primary containers. When load is called on the same file path again, the previous loaded Resource will be referenced. Naturally, loaded resources must be stateless.

This guide assumes the reader knows how to create C++ modules and godot data types. If not, refer to this guide *Custom modules in C++*.

References:

- *ResourceLoader*
- core/io/resource_loader.cpp

26.8.2 What for?

- Adding new support for many file formats
- Audio formats
- Video formats
- Machine learning models

26.8.3 What not?

- Raster images

ImageFormatLoader should be used to load images.

References:

- core/io/image_loader.h

26.8.4 Creating a ResourceFormatLoader

Each file format consist of a data container and a `ResourceFormatLoader`.

ResourceFormatLoaders are usually simple classes which return all the necessary metadata for supporting new extensions in Godot. The class must the return the format name and the extension string.

In addition, ResourceFormatLoaders must convert file paths into resources with the `load` function. To load a resource, `load` must read and handle data serialization.

```
#ifndef MY_JSON_LOADER_H
#define MY_JSON_LOADER_H

#include "io/resource_loader.h"

class ResourceFormatLoaderMyJson : public ResourceFormatLoader {
public:
        virtual RES load(const String &p_path, const String &p_original_path, Error␣
↪*r_error = NULL);
        virtual void get_recognized_extensions(List<String> *p_extensions) const;
        virtual bool handles_type(const String &p_type) const;
        virtual String get_resource_type(const String &p_path) const;

        ResourceFormatLoaderMyJson();
        virtual ~ResourceFormatLoaderMyJson() {}
};
#endif // MY_JSON_LOADER_H
```

```
#include "my_json_loader.h"
#include "my_json.h"

ResourceFormatLoaderMyJson::ResourceFormatLoaderMyJson() {
}

RES ResourceFormatLoaderMyJson::load(const String &p_path, const String &p_original_
↪path, Error *r_error) {
        MyJson *my = memnew(MyJson);
        if (r_error)
                *r_error = OK;
        Error err = my->set_file(p_path);
        return Ref<MyJson>(my);
}

void ResourceFormatLoaderMyJson::get_recognized_extensions(List<String> *p_
↪extensions) const {
        p_extensions->push_back("mjson");
}

String ResourceFormatLoaderMyJson::get_resource_type(const String &p_path) const {

        if (p_path.get_extension().to_lower() == "mjson")
                return "MyJson";
        return "";
}

bool ResourceFormatLoaderMyJson::handles_type(const String &p_type) const {
        return (p_type == "MyJson");
}
```

26.8.5 Creating Custom Data Types

Godot may not have a proper substitute within its *Core types* or managed resources. Godot needs a new registered data type to understand additional binary formats such as machine learning models.

Here is an example of how to create a custom datatype

```
#ifndef MY_JSON_H
#define MY_JSON_H

#include "core/variant.h"
#include "reference.h"
#include "variant_parser.h"
#include "io/json.h"
#include "dictionary.h"

class MyJson : public Resource{
        GDCLASS(MyJson, Resource);

protected:
        static void _bind_methods() {
                ClassDB::bind_method(D_METHOD("toString"), &MyJson::toString);
        }

private:
        Dictionary dict;
public:
        Error set_file(const String &p_path){
                Error error_file;
                FileAccess *file = FileAccess::open(p_path, FileAccess::READ, &error_
↪file);

                String buf = String("");
                while(!file->eof_reached()){
                        buf += file->get_line();
                }
                String err_string;
                int err_line;
                JSON cmd;
                Variant ret;
                Error err = cmd.parse( buf, ret, err_string, err_line);
                dict = Dictionary(ret);
                file -> close();
                return OK;
        }

        String toString() const {
                return String(*this);
        }

        operator String() const {
                JSON a;
                return a.print(dict);
        }

    MyJson() {};
    ~MyJson() {};
};
#endif
```

Considerations

Some libraries may not define certain common routines such as i/o handling. Therefore, Godot call translations are required.

For example, here is the code for translating `FileAccess` calls into `std::istream`.

```
#include <istream>
#include <streambuf>

class GodotFileInStreamBuf : public std::streambuf{
public:
        GodotFileInStreamBuf(FileAccess * fa) {
                _file = fa;
        }
        int underflow() {
                if (_file->eof_reached()) {
                        return EOF;
                } else {
                        size_t pos = _file->get_position();
                        uint8_t ret = _file->get_8();
                        _file->seek(pos); //required since get_8() advances the read
↪head
                        return ret;
                }
        }
        int uflow() {
                return _file->eof_reached() ? EOF : _file -> get_8();
        }
private:
        FileAccess * _file;
};
```

References:

- istream
- streambuf
- core/io/fileaccess.h

26.8.6 Registering the New File Format

Godot registers `ResourcesFormatLoader` with a `ResourceLoader` handler. The handler selects the proper loader automatically when `load` is called.

```
/* register_types.cpp */
#include "register_types.h"
#include "class_db.h"

#include "my_json_loader.h"
#include "my_json.h"

static ResourceFormatLoaderMyJson *my_json_loader = NULL;
void register_my_json_types() {
        my_json_loader = memnew(ResourceFormatLoaderMyJson);
        ResourceLoader::add_resource_format_loader(my_json_loader);
        ClassDB::register_class<MyJson>();
}

void unregister_my_json_types() {
```

(continues on next page)

(continued from previous page)

```
        memdelete(my_json_loader);
}
```

References:

- core/io/resource_loader.cpp

26.8.7 Loading it on GDScript

```
{
  "savefilename" : "demo.mjson",
  "demo": [
    "welcome",
    "to",
    "godot",
    "resource",
    "loaders"
  ]
}
```

```
extends Node

func _ready():
        var myjson = load("res://demo.mjson")
        print( myjson.toString())
```

26.9 Custom AudioStreams

26.9.1 Introduction

AudioStream is the base class of all audio emitting objects. AudioStreamPlayer binds onto an AudioStream to emit PCM data into an AudioServer which manages audio drivers.

All audio resources require two audio based classes: AudioStream and AudioStreamPlayback. As a data container, AudioStream contains the resource and exposes itself to GDScript. AudioStream references its own internal custom AudioStreamPlayback which translates AudioStream into PCM data.

This guide assumes the reader knows how to create C++ modules. If not, refer to this guide *Custom modules in C++*.

References:

- servers/audio/audio_stream.h
- scene/audio/audioplayer.cpp

26.9.2 What for?

- Binding external libraries (like Wwise, FMOD, etc).

- Adding custom audio queues
- Adding support for more audio formats

26.9.3 Create an AudioStream

An AudioStream consists of three components: data container, stream name, and an AudioStreamPlayback friend class generator. Audio data can be loaded in a number of ways such as with an internal counter for a tone generator, internal/external buffer, or a file reference.

Some AudioStreams need to be stateless such as objects loaded from ResourceLoader. ResourceLoader loads once and references the same object regardless how many times `load` is called on a specific resource. Therefore, playback state must be self contained in AudioStreamPlayback.

```
/* audiostream_mytone.h */

#include "reference.h"
#include "resource.h"
#include "servers/audio/audio_stream.h

class AudioStreamMyTone : public AudioStream {
        GDCLASS(AudioStreamMyTone, AudioStream)
private:
        friend class AudioStreamPlaybackMyTone;
        uint64_t pos;
        int mix_rate;
        bool stereo;
        int hz;
public:
        void reset();
        void set_position(uint64_t pos);
        virtual Ref<AudioStreamPlayback> instance_playback();
        virtual String get_stream_name() const;
        void gen_tone(int16_t *, int frames);
        virtual float get_length() const { return 0; } //if supported, otherwise␣
↪return 0
        AudioStreamMyTone();

protected:
        static void _bind_methods();
};
```

```
/* audiostream_mytone.cpp */
AudioStreamMyTone::AudioStreamMyTone()
        : mix_rate(44100), stereo(false), hz(639) {
}

Ref<AudioStreamPlayback> AudioStreamMyTone::instance_playback(){
        Ref<AudioStreamPlaybackMyTone> talking_tree;
        talking_tree.instance();
        talking_tree->base = Ref<AudioStreamMyTone>(this);
        return talking_tree;
}

String AudioStreamMyTone::get_stream_name() const {
        return "MyTone";
}
```

(continues on next page)

(continued from previous page)

```
void AudioStreamMyTone::reset() {
        set_position(0);
}
void AudioStreamMyTone::set_position(uint64_t p) {
        pos = p;
}
void AudioStreamMyTone::gen_tone(int16_t * pcm_buf, int size){
        for( int i = 0; i < size; i++){
                pcm_buf[i] = 32767.0 * sin(2.0*Math_PI*double(pos+i)/(double(mix_
↪rate)/double(hz)));
        }
        pos += size;
}
void AudioStreamMyTone::_bind_methods(){
        ClassDB::bind_method(D_METHOD("reset"), &AudioStreamMyTone::reset);
        ClassDB::bind_method(D_METHOD("get_stream_name"), &AudioStreamMyTone::get_
↪stream_name);

}
```

References:

- servers/audio/audio_stream.h

26.9.4 Create an AudioStreamPlayback

AudioStreamPlayer uses `mix` callback to obtain PCM data. The callback must match sample rate and fill the buffer.

Since AudioStreamPlayback is controlled by the audio thread, i/o and dynamic memory allocation are forbidden.

```
/*  audiostreamplayer_mytone.h */
#include "reference.h"
#include "resource.h"
#include "servers/audio/audio_stream.h"


class AudioStreamPlaybackMyTone : public AudioStreamPlayback {
        GDCLASS(AudioStreamPlaybackMyTone, AudioStreamPlayback)
        friend class AudioStreamMyTone;
private:
        enum{
                PCM_BUFFER_SIZE = 4096
        };
        enum {
                MIX_FRAC_BITS = 13,
                MIX_FRAC_LEN = (1 << MIX_FRAC_BITS),
                MIX_FRAC_MASK = MIX_FRAC_LEN - 1,
        };
        void * pcm_buffer;
        Ref<AudioStreamMyTone> base;
        bool active;
public:
        virtual void start(float p_from_pos = 0.0);
        virtual void stop();
```

(continues on next page)

(continued from previous page)

```
        virtual bool is_playing() const;
        virtual int get_loop_count() const; //times it looped
        virtual float get_playback_position() const;
        virtual void seek(float p_time);
        virtual void mix(AudioFrame *p_buffer, float p_rate_scale, int p_frames);
        virtual float get_length() const; //if supported, otherwise return 0
        AudioStreamPlaybackMyTone();
        ~AudioStreamPlaybackMyTone();

};
```

```
/* audiostreamplayer_mytone.cpp */
#include "audiostreamplayer_mytone.h"
#include "math/math_funcs.h"
#include "print_string.h"

AudioStreamPlaybackMyTone::AudioStreamPlaybackMyTone()
        : active(false){
        AudioServer::get_singleton()->lock();
        pcm_buffer = AudioServer::get_singleton()->audio_data_alloc(PCM_BUFFER_SIZE);
        zeromem(pcm_buffer, PCM_BUFFER_SIZE);
        AudioServer::get_singleton()->unlock();
}
AudioStreamPlaybackMyTone::~AudioStreamPlaybackMyTone() {
        if(pcm_buffer) {
                AudioServer::get_singleton()->audio_data_free(pcm_buffer);
                pcm_buffer = NULL;
        }
}
void AudioStreamPlaybackMyTone::stop(){
        active = false;
        base->reset();
}

void AudioStreamPlaybackMyTone::start(float p_from_pos){
        seek(p_from_pos);
        active = true;
}
void AudioStreamPlaybackMyTone::seek(float p_time){
        float max = get_length();
        if (p_time < 0) {
                        p_time = 0;
        }
        base->set_position(uint64_t(p_time * base->mix_rate) << MIX_FRAC_BITS);
}
void AudioStreamPlaybackMyTone::mix(AudioFrame *p_buffer, float p_rate, int p_frames){
        ERR_FAIL_COND(!active);
        if (!active) {
                        return;
        }
        zeromem(pcm_buffer, PCM_BUFFER_SIZE);
        int16_t * buf = (int16_t * )pcm_buffer;
        base->gen_tone(buf, p_frames);

        for(int i = 0;  i < p_frames; i++){
                float sample =  float(buf[i])/32767.0;
```

(continues on next page)

(continued from previous page)

```
                p_buffer[i] = AudioFrame(sample, sample);
        }
}
int AudioStreamPlaybackMyTone::get_loop_count() const {
        return 0;
}
float AudioStreamPlaybackMyTone::get_playback_position() const {
        return 0.0;
}
float AudioStreamPlaybackMyTone::get_length() const {
        return 0.0;
}
bool AudioStreamPlaybackMyTone::is_playing() const {
        return active;
}
```

Resampling

Godot's AudioServer currently uses 44100 Hz sample rate. When other sample rates are needed such as 48000, either provide one or use AudioStreamPlaybackResampled. Godot provides cubic interpolation for audio resampling.

Instead of overloading `mix`, AudioStreamPlaybackResampled uses `_mix_internal` to query AudioFrames and `get_stream_sampling_rate` to query current mix rate.

```
#include "reference.h"
#include "resource.h"

#include "servers/audio/audio_stream.h"

class AudioStreamMyToneResampled;

class AudioStreamPlaybackResampledMyTone : public AudioStreamPlaybackResampled {
        GDCLASS(AudioStreamPlaybackResampledMyTone, AudioStreamPlaybackResampled)
        friend class AudioStreamMyToneResampled;
private:
        enum{
                PCM_BUFFER_SIZE = 4096
        };
        enum {
                MIX_FRAC_BITS = 13,
                MIX_FRAC_LEN = (1 << MIX_FRAC_BITS),
                MIX_FRAC_MASK = MIX_FRAC_LEN - 1,
        };
        void * pcm_buffer;
        Ref<AudioStreamMyToneResampled> base;
        bool active;
protected:
        virtual void _mix_internal(AudioFrame *p_buffer, int p_frames);
public:
        virtual void start(float p_from_pos = 0.0);
        virtual void stop();
        virtual bool is_playing() const;
        virtual int get_loop_count() const; //times it looped
        virtual float get_playback_position() const;
        virtual void seek(float p_time);
```

(continues on next page)

(continued from previous page)

```
        virtual float get_length() const; //if supported, otherwise return 0
        virtual float get_stream_sampling_rate();
        AudioStreamPlaybackResampledMyTone();
        ~AudioStreamPlaybackResampledMyTone();

};
```

```
#include "mytone_audiostream_resampled.h"
#include "math/math_funcs.h"
#include "print_string.h"

AudioStreamPlaybackResampledMyTone::AudioStreamPlaybackResampledMyTone()
        : active(false){
        AudioServer::get_singleton()->lock();
        pcm_buffer = AudioServer::get_singleton()->audio_data_alloc(PCM_BUFFER_SIZE);
        zeromem(pcm_buffer, PCM_BUFFER_SIZE);
        AudioServer::get_singleton()->unlock();
}
AudioStreamPlaybackResampledMyTone::~AudioStreamPlaybackResampledMyTone() {
        if (pcm_buffer) {
                AudioServer::get_singleton()->audio_data_free(pcm_buffer);
                pcm_buffer = NULL;
        }
}
void AudioStreamPlaybackResampledMyTone::stop(){
        active = false;
        base->reset();
}

void AudioStreamPlaybackResampledMyTone::start(float p_from_pos){
        seek(p_from_pos);
        active = true;
}
void AudioStreamPlaybackResampledMyTone::seek(float p_time){
        float max = get_length();
        if (p_time < 0) {
                        p_time = 0;
        }
        base->set_position(uint64_t(p_time * base->mix_rate) << MIX_FRAC_BITS);
}
void AudioStreamPlaybackResampledMyTone::_mix_internal(AudioFrame *p_buffer, int p_
↪frames){
        ERR_FAIL_COND(!active);
        if (!active) {
                return;
        }
        zeromem(pcm_buffer, PCM_BUFFER_SIZE);
        int16_t * buf = (int16_t * )pcm_buffer;
        base->gen_tone(buf, p_frames);

        for(int i = 0;  i < p_frames; i++){
                float sample =  float(buf[i])/32767.0;
                        p_buffer[i] = AudioFrame(sample, sample);
        }
}
float AudioStreamPlaybackResampledMyTone::get_stream_sampling_rate(){
```

(continues on next page)

(continued from previous page)

```
        return float(base->mix_rate);
}
int AudioStreamPlaybackResampledMyTone::get_loop_count() const {
        return 0;
}
float AudioStreamPlaybackResampledMyTone::get_playback_position() const {
        return 0.0;
}
float AudioStreamPlaybackResampledMyTone::get_length() const {
        return 0.0;
}
bool AudioStreamPlaybackResampledMyTone::is_playing() const {
        return active;
}
```

References:

- core/math/audio_frame.h
- servers/audio/audio_stream.h
- scene/audio/audioplayer.cpp

26.10 Custom Godot Servers

26.10.1 Introduction

Godot implements multi threading as servers. Servers are daemons which manages data, processes, and pushes the result. Server implements the mediator pattern which interprets resource ID and process data for the engine and other modules. In addition, the server claims ownership for its RID allocations.

This guide assumes the reader knows how to create C++ modules and godot data types. If not, refer to this guide *Custom modules in C++*.

References:

- Why does Godot use Servers and RIDs?
- Sigleton Pattern
- Mediator Pattern

26.10.2 What for?

- Adding AI
- Adding a custom asynchronous threads
- Adding Input support
- Adding writing threads
- Adding custom VOIP protocol

- etc.

26.10.3 Creating a Godot Server

At minimum, a server must to have: a static instance, sleep timer, thread loop, initialize state, and cleanup.

```
#ifndef HILBERT_HOTEL_H
#define HILBERT_HOTEL_H

#include "object.h"
#include "list.h"
#include "rid.h"
#include "set.h"
#include "variant.h"
#include "os/thread.h"
#include "os/mutex.h"

class HilbertHotel : public Object {
        GDCLASS(HilbertHotel, Object);

        static HilbertHotel *singleton;
        static void thread_func(void *p_udata);

private:
        bool thread_exited;
        mutable bool exit_thread;
        Thread *thread;
        Mutex *mutex;

public:
        static HilbertHotel *get_singleton();
        Error init();
        void lock();
        void unlock();
        void finish();

protected:
        static void _bind_methods();

private:
        uint64_t counter;
        RID_Owner<InfiniteBus> bus_owner;
        //https://github.com/godotengine/godot/blob/master/core/rid.h#L196
        Set<RID> buses;
        void _emit_occupy_room(uint64_t room, RID rid);

public:
        RID create_bus();
        Variant get_bus_info(RID id);
        bool empty();
        bool delete_bus(RID id);
        void clear();
        void register_rooms();
        HilbertHotel();
};
#endif
```

```
#include "hilbert_hotel.h"
#include "variant.h"
#include "os/os.h"
#include "list.h"
#include "dictionary.h"
#include "prime_225.h"

oid HilbertHotel::thread_func(void *p_udata){
        HilbertHotel *ac = (HilbertHotel *) p_udata;

        uint64_t msdelay = 1000;
        while(!ac -> exit_thread){
                if(!ac -> empty()) {
                        ac->lock();
                        ac->register_rooms();
                        ac->unlock();
                }
                OS::get_singleton()->delay_usec(msdelay * 1000);
        }
}

Error HilbertHotel::init(){
        thread_exited = false;
        counter = 0;
        mutex = Mutex::create();
        thread = Thread::create(HilbertHotel::thread_func, this);
        return OK;
}
HilbertHotel *HilbertHotel::singleton = NULL;
HilbertHotel *HilbertHotel::get_singleton() { return singleton; }
void HilbertHotel::register_rooms() {
        for( Set<RID>::Element *e = buses.front(); e; e = e->next()) {
                auto bus = bus_owner.getornull(e->get());
                if(bus){
                        uint64_t room = bus->next_room();
                        _emit_occupy_room(room, bus->get_self());
                }
        }
}

void HilbertHotel::unlock() {
        if (!thread || !mutex)
                return;
        mutex->unlock();
}
void HilbertHotel::lock() {
        if (!thread || !mutex)
                return;
        mutex->lock();
}
void HilbertHotel::_emit_occupy_room(uint64_t room, RID rid) {
        _HilbertHotel::get_singleton()->_occupy_room(room, rid);
}
Variant HilbertHotel::get_bus_info(RID id){
        InfiniteBus * bus = bus_owner.getornull(id);
        if(bus){
                Dictionary d;
```

(continues on next page)

(continued from previous page)

```
                d["prime"] = bus->get_bus_num();
                d["current_room"] = bus->get_current_room();
                return d;
        }
        return Variant();
}
void HilbertHotel::finish() {
        if (!thread)
                return;

        exit_thread = true;
        Thread::wait_to_finish(thread);

        memdelete(thread);
        if (mutex)
                memdelete(mutex);
        thread = NULL;
}
RID HilbertHotel::create_bus() {
        lock();
        InfiniteBus *ptr = memnew(InfiniteBus(PRIME[counter++]));
        RID ret = bus_owner.make_rid(ptr);
        ptr->set_self(ret);
        buses.insert(ret);
        unlock();
        return ret;
}
//https://github.com/godotengine/godot/blob/master/core/rid.h#L187
bool HilbertHotel::delete_bus(RID id) {
        if (bus_owner.owns(id)) {
                lock();
                InfiniteBus *b = bus_owner.get(id);
                bus_owner.free(id);
                buses.erase(id);
                memdelete(b);
                unlock();
                return true;
        }
        return false;
}
void HilbertHotel::clear() {

        for( Set<RID>::Element *e = buses.front(); e; e = e->next()) {
                delete_bus(e->get());
        }
}
bool HilbertHotel::empty() {
        return buses.size() <= 0;
}
void HilbertHotel::_bind_methods() {
}

HilbertHotel::HilbertHotel() {
        singleton = this;
}
```

```
/* prime_225.h */

#include "int_types.h"
const uint64_t PRIME[225] = {2,3,5,7,11,13,17,19,23,
29,31,37,41,43,47,53,59,61,
67,71,73,79,83,89,97,101,103,
107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,
199,211,223,227,229,233,239,241,251,
257,263,269,271,277,281,283,293,307,
311,313,317,331,337,347,349,353,359,
367,373,379,383,389,397,401,409,419,
421,431,433,439,443,449,457,461,463,
467,479,487,491,499,503,509,521,523,
541,547,557,563,569,571,577,587,593,
599,601,607,613,617,619,631,641,643,
647,653,659,661,673,677,683,691,701,
709,719,727,733,739,743,751,757,761,
769,773,787,797,809,811,821,823,827,
829,839,853,857,859,863,877,881,883,
887,907,911,919,929,937,941,947,953,
967,971,977,983,991,997,1009,1013,1019,
1021,1031,1033,1039,1049,1051,1061,1063,1069,
1087,1091,1093,1097,1103,1109,1117,1123,1129,
1151,1153,1163,1171,1181,1187,1193,1201,1213,
1217,1223,1229,1231,1237,1249,1259,1277,1279,
1283,1289,1291,1297,1301,1303,1307,1319,1321,
1327,1361,1367,1373,1381,1399,1409,1423,1427};
```

26.10.4 Custom Managed Resource Data

Godot servers implement a mediator pattern. All data types inherit `RID_Data`. *RID_Owner<MyRID_Data>* owns the object when `make_rid` is called. Only during debug mode, RID_Owner maintains a list of RID. In practice, RID is similar to writing object oriented C code.

```
class InfiniteBus : public RID_Data {
        RID self;

private:
        uint64_t prime_num;
        uint64_t num;

public:
        uint64_t next_room() {
                return prime_num * num++;
        }
        uint64_t get_bus_num() const {
                return prime_num;
        }
        uint64_t get_current_room() const {
                return prime_num * num;
        }
        _FORCE_INLINE_ void set_self(const RID &p_self) { self = p_self; }
        _FORCE_INLINE_ RID get_self() const { return self; }
```

(continues on next page)

(continued from previous page)

```
        InfiniteBus(uint64_t prime) : prime_num(prime), num(1) {};
        ~InfiniteBus() {};
}
```

References:

- *RID*
- core/rid.h

26.10.5 Registering the class to GDScript

Server are allocated in `register_types.cpp`. The constructor sets the static instance and init creates the managed thread. `unregister_types.cpp` cleans up the server

Since Godot Server class creates an instance and binds it to a static singleton, binding the class might not reference the correct instance. Therefore, a dummy class must be created to reference the proper Godot Server.

In `register_godotserver_types()`, `Engine::get_singleton()->add_singleton` is used to register the dummy class to GDScript.

```
/* register_types.cpp */

#include "register_types.h"
#include "class_db.h"
#include "hilbert_hotel.h"
#include "engine.h"

static HilbertHotel *hilbert_hotel = NULL;
static _HilbertHotel *_hilbert_hotel = NULL;


void register_hilbert_hotel_types() {
                hilbert_hotel = memnew(HilbertHotel);
                hilbert_hotel->init();
                _hilbert_hotel = memnew(_HilbertHotel);
                ClassDB::register_class<_HilbertHotel>();
                Engine::get_singleton()->add_singleton(Engine::Singleton("HilbertHotel
↪", _HilbertHotel::get_singleton()));
}

void unregister_hilbert_hotel_types() {
                if(hilbert_hotel){
                                hilbert_hotel->finish();
                                memdelete(hilbert_hotel);
                }
                if(_hilbert_hotel) {
                                memdelete(_hilbert_hotel);
                }
}
```

```
/* register_types.h */
void register_hilbert_hotel_types();
void unregister_hilbert_hotel_types();
/* yes, the word in the middle must be the same as the module folder name */
```

- servers/register_server_types.cpp

Bind methods

The dummy class binds singleton methods to gdscript. In most cases, the dummy class methods wraps around.

```
Variant _HilbertHotel::get_bus_info(RID id) {
        return HilbertHotel::get_singleton()->get_bus_info(id);
}
```

Binding Signals

It is possible to emit signals to gdscript but calling the GDScript dummy object.

```
void HilbertHotel::_emit_occupy_room(uint64_t room, RID rid) {
        _HilbertHotel::get_singleton()->_occupy_room(room, rid);
}
```

```
class _HilbertHotel : public Object {
        GDCLASS(_HilbertHotel, Object);

        friend class HilbertHotel;
        static _HilbertHotel *singleton;

protected:
        static void _bind_methods();

private:
        void _occupy_room(int room_number, RID bus);

public:
        RID create_bus();
        void connect_singals();
        bool delete_bus(RID id);
        static _HilbertHotel *get_singleton();
        Variant get_bus_info(RID id);

        _HilbertHotel();
        ~_HilbertHotel();
};
#endif
```

```
_HilbertHotel *_HilbertHotel::singleton = NULL;
_HilbertHotel *_HilbertHotel::get_singleton() { return singleton; }

RID _HilbertHotel::create_bus() {
        return HilbertHotel::get_singleton()->create_bus();
}
bool _HilbertHotel::delete_bus(RID rid) {
        return HilbertHotel::get_singleton()->delete_bus(rid);
}
void _HilbertHotel::_occupy_room(int room_number, RID bus){
        emit_signal("occupy_room", room_number, bus);
}
Variant _HilbertHotel::get_bus_info(RID id) {
        return HilbertHotel::get_singleton()->get_bus_info(id);
```

(continues on next page)

(continued from previous page)

```
}
void _HilbertHotel::_bind_methods() {
        ClassDB::bind_method(D_METHOD("get_bus_info", "r_id"), &_HilbertHotel::get_
↪bus_info);
        ClassDB::bind_method(D_METHOD("create_bus"), &_HilbertHotel::create_bus);
        ClassDB::bind_method(D_METHOD("delete_bus"), &_HilbertHotel::delete_bus);
        ADD_SIGNAL(MethodInfo("occupy_room", PropertyInfo(Variant::INT, "room_number
↪"), PropertyInfo(Variant::_RID, "r_id")));
}
void _HilbertHotel::connect_singals() {
        HilbertHotel::get_singleton()->connect("occupy_room", _HilbertHotel::get_
↪singleton(), "_occupy_room");
}
_HilbertHotel::_HilbertHotel() {
        singleton = this;
}
_HilbertHotel::~_HilbertHotel() {
}
```

26.10.6 MessageQueue

In order to send commands into scenetree, MessageQueue is a thread safe buffer to queue set and call methods for other threads. To queue a command, obtain the target object RID and use either push_call, push_set, or push_notification to execute the desired behavior. Queue will be flushed whenever either `SceneTree::idle` or `SceneTree::iteration` are executed.

References:

- core/message_queue.cpp

26.10.7 Summing it up

Here is the GDScript sample code

```
extends Node

# class member variables go here, for example:
# var a = 2
# var b = "textvar"

func _ready():
        # Called when the node is added to the scene for the first time.
        # Initialization here
        print("start Debugging")
        HilbertHotel.connect("occupy_room", self, "_print_occupy_room")
        var rid = HilbertHotel.create_bus()
        OS.delay_msec(2000)
        HilbertHotel.create_bus()
        OS.delay_msec(2000)
        HilbertHotel.create_bus()
        OS.delay_msec(2000)
        print(HilbertHotel.get_bus_info(rid))
```

(continues on next page)

(continued from previous page)

```
        HilbertHotel.delete_bus(rid)
        print("ready done")
        pass



func _print_occupy_room(room_number, r_id):
        print("room_num: "  + str(room_number) + " rid: " + str(r_id))
        print(HilbertHotel.get_bus_info(r_id))
```

Notes

- Actual Hilbert Hotel is impossible
- Connecting signal example code is pretty hacky

26.11 Creating Android modules

26.11.1 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 thingst like:

- 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

On iOS, you can write a C++ module and take advantage of the C++/ObjC intercommunication.

On Android, interfacing with C++ through JNI (Java Native Interface) isn't as convenient.

26.11.2 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 on every platform. Only write the code once and you are set to go.

26.11.3 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 the target platform being built for 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 for Android (or other platform-specific) code:

```
def can_build(plat):
    return plat=="android" or plat=="iphone"

def configure(env):
    if env['platform'] == 'android':
        # android specific code
```

26.11.4 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 org.godotengine.godot;

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
```

(continues on next page)

(continued from previous page)

```
        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 project.godot
                    String key = GodotLib.getGlobal("plugin/api_key");
                    SDK.initializeHere();
                }
        });

    }

    // forwarded callbacks you can reimplement, as SDKs often need them

    protected void onMainActivityResult(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:

```
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")
```

26.11.5 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:

```
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 contains MySingelton.java")
        env.android_add_to_manifest("AndroidManifestChunk.xml")
```

26.11.6 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.

26.11.7 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, put it in the module directory and add it:

```
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 contains MySingelton.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
```

26.11.8 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:

```
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","")
```

26.11.9 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, 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 project.godot:

```
[android]

modules="org/godotengine/godot/MySingleton"
```

More than one singleton module can be enabled by separating with commas:

```
[android]

modules="org/godotengine/godot/MySingleton,corg/godotengine/godot/MyOtherSingleton"
```

Then 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"))
```

26.11.10 Troubleshooting

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.

26.11.11 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 27

Godot file formats

27.1 TSCN File Format

A `.tscn` File format is the "Text SCeNe" file format and represents a single scene-tree inside Godot. TSCN files have the advantage of being nearly human-readable and easy for version control systems to manage. During import the TSCN files are compiled into binary `.scn` files stored inside the .import folder. This reduces the data size and speed up loading.

The `.escn` file format is identical to the TSCN file format, but is used to indicate to Godot that the file has been exported from another program and should not be edited by the user from within Godot.

For those looking for a complete description, the parsing is handled in the file scene_format_text.cpp in the class `ResourceFormatLoaderText`

27.1.1 File Structure

There are five main sections inside the TSCN File:

0. File Descriptor
1. External resources
2. Internal resources
3. Nodes
4. Connections

The file descriptor looks like `[gd_scene load_steps=1 format=2]` And should be the first entry in the file. The load_steps parameter should (in theory) be the number of resources within the file, though in practice it's value seems not to matter.

These sections should appear in order, but it can be hard to distinguish them. The only difference between them is the first element in the heading for all of the items in the section. For example, the heading of all external resources should start with `[ext_resource .....]`

Entries inside the file

A heading looks like: `[<resource_type> key=value key=value key=value ...]` Where resource_type is one of:

- ext_resource
- sub_resource
- node
- connection

Underneath every heading comes zero or more `key = value` pairs. The values can be complex datatypes such as arrays, transformations, colors, and so on. For example, a spatial node looks like:

```
[node name="Cube" type="Spatial" parent="."]
transform=Transform( 1.0, 0.0, 0.0 ,0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 )
```

27.1.2 Resources

Resources are components that make up the nodes. For example, a MeshInstance node will have an accompanying ArrayMesh resource. The ArrayMesh resource may be either internal or external to the TSCN file.

References to the resources are handled by id numbers in the resources heading. External resources and internal resource are referred to with `ExtResource(id)` and `SubResource(id)`. Because there have different methods to refer to internal and external resource, you can have the same ID for both an internal and external resource.

For example, to refer to the resource `[ext_resource id=3 type="PackedScene" path=....]` you would use `ExtResource(3)`

External Resources

External resources are links to resources not contained within the TSCN file itself. An external resource consists of:

- A path
- A type
- An ID

Godot alway generates absolute paths relative to the resource directory and thus prefixed with `res://`, but paths relative to the TSCN file's location are also valid.

Some example external resources are:

```
[ext_resource path="res://characters/player.dae" type="PackedScene" id=1]
[ext_resource path="metal.tres" type="Material" id=2]
```

Internal Resources

A TSCN file can contain meshes, materials and other data, and these are contained in the internal resources section of the file. The heading for an internal resource looks similar to those of external resources, but does not have a path. Internal resources also have `key=value` pairs under each heading. For example, a capsule collision shape looks like:

```
[sub_resource  type="CapsuleShape" id=2]

radius = 0.5
height = 3.0
```

Some internal resource contain links to other internal resources (such as a mesh having a material). In this case, the referring resource must appear before the reference to it. Thus, in the internal resources section of the file, order does matter.

Unfortunately, documentation on the formats for these subresources is completely absent, and while some can be found through inspecting resources of saved files, but others can only be found by looking through Godot's source.

27.1.3 The Scene Tree

The scene tree is made up of ... nodes! The heading of each node consists of it's name, parent and (most of the time) a type. For example `[node type="Camera" name="PlayerCamera" parent="Player/Head"]`

Other valid keywords include:

- instance
- instance_placeholder
- owner
- index (if two nodes have the same name)
- groups

The first node in the file should not have the `parent=Path/To/Node` entry in it's heading, and it is the scene root. All scene files should have exactly one scene root. It it does not, Godot will fail to import the file. The parent path of other nodes should be absolute, but without the scene root's name. If it is a direct child of the scene root, it should be `"."`. Here is an example scene tree (but without any node content).

```
[node name="Player" type="Spatial"]             ; The scene root
[node name="Arm" parent="." type="Spatial"]     ; Parented to the scene root
[node name="Hand" parent="Arm" type="Spatial"]
[node name="Finger" parent="Arm/Hand" type="Spatial"]
```

Similar to the internal resource, the content for each node is currently undocumented. Fortunately it is easy to find out because you can simply save a file with that node in it. Some example nodes are:

```
[node  type="CollisionShape" name="SphereCollision" parent="SpherePhysics"]

shape = SubResource(8)
transform = Transform( 1.0 , 0.0 , -0.0 , 0.0 , -4.371138828673793e-08 , 1.0 , -0.0 ,␣
↪-1.0 , -4.371138828673793e-08 ,0.0 ,0.0 ,-0.0  )


[node  type="MeshInstance" name="Sphere" parent="SpherePhysics"]

mesh = SubResource(9)
transform = Transform( 1.0 , 0.0 , -0.0 , 0.0 , 1.0 , -0.0 , -0.0 , -0.0 , 1.0 ,0.0 ,
↪0.0 ,-0.0  )


[node  type="OmniLight" name="Lamp" parent="."]
```

(continues on next page)

(continued from previous page)

```
light_energy = 1.0
light_specular = 1.0
transform = Transform( -0.29086464643478394 , -0.7711008191108704 , 0.
↪5663931369781494 , -0.05518905818462372 , 0.6045246720314026 , 0.7946722507476807 ,␣
↪-0.9551711678504944 , 0.199883371591568 , -0.21839118003845215 ,4.076245307922363 ,
↪7.3235554695129395 ,-1.0054539442062378  )
omni_range = 30
shadow_enabled = true
light_negative = false
light_color = Color( 1.0, 1.0, 1.0, 1.0 )


[node  type="Camera" name="Camera" parent="."]

projection = 0
near = 0.10000000149011612
fov = 50
transform = Transform( 0.6859206557273865 , -0.32401350140571594 , 0.6515582203865051␣
↪, 0.0 , 0.8953956365585327 , 0.44527143239974976 , -0.7276763319969177 , -0.
↪3054208755493164 , 0.6141703724861145 ,14.430776596069336 ,10.093015670776367 ,13.
↪058500289916992  )
far = 100.0
```

CHAPTER 28

Community

28.1 Contributing

28.1.1 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) and its translations. 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 core developers so they 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 MIT license grants 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.

A nice place to start may be the issue tagged as junior jobs on GitHub.

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 release candidate (RC) 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 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 and its translations.** 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.

28.1.2 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 branches 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 documention, 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 (although it's tolerated for small fixes or 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:

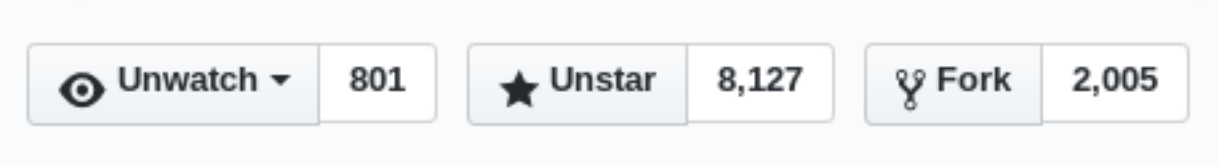
The first sections of Git's "Book" are a good introduction to the tool's philosophy and the various commands you need to master in your daily workflow. You can read them online on the Git SCM website.

The branches on the Git repository are organized as follows:

- The `master` branch is where the development of the next major version 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. `3.0` and `2.1`. They are used to backport bugfixes and enhancements from the `master` branch to the currently maintained stable release (e.g. 3.0.2 or 2.1.5). 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:

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 macOS, or install it through your package manager if you're using Linux.

Note: If you are on Windows, open Git Bash to type commands. macOS 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 to 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.

Therefore, the command that you should (almost) always use is:

```
$ 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 configuration) 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>]
```

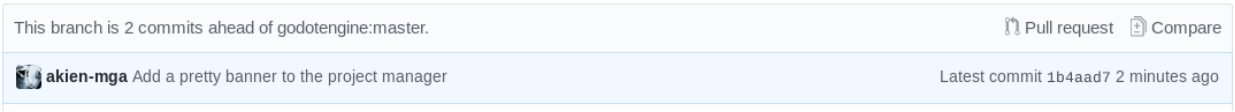
The part about the remote branch can be omitted if you want it to have the same name as the local branch, which is our case in this example, so we will do:

```
$ 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 your 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
```

28.1.3 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 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 come 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 my 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.

28.1.4 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.

28.1.5 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. Also have a look at README of the godot-docs GitHub repository and the docs front page on what steps to follow and how to contact the docs team.

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. Read more here: *Contribute to the Class Reference*.

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. Also have a look at the *Docs writing guidelines*.

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).

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

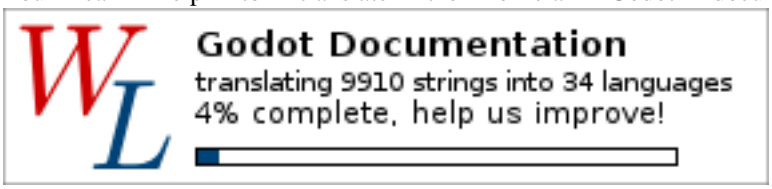
And this is a bad example:

- Insert Your Title Here

Only project, people and node class names should have capitalized first letter.

Translating existing pages

You can help to translate the official Godot documentation on our Hosted Weblate.



There also is the official Godot I18N repository. where you can see when the data was last synced.

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.

28.1.6 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`.

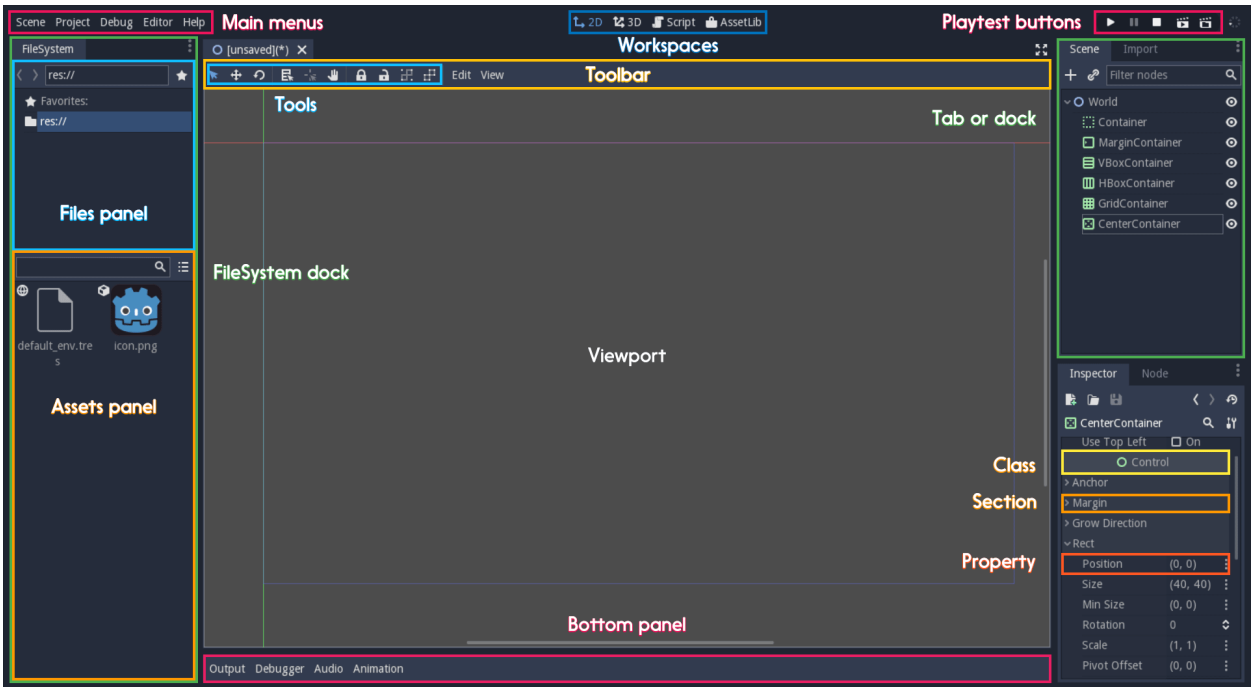


Fig. 1: Overview of the interface and common vocabulary

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`.

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.

28.1.7 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. Some 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, and all of us, to contribute.

Important: If you are planning to make larger changes or a more substantial contribution, it is usually a good idea to create an issue (or a comment in an existing one) to let others know so they don't start working on the same thing too.

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 two: `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 *Sprite*.
[method methodname]	Link to a method in this class	Call [method hide].	See *hide*.
[method Class.methodname]	Link to another class's method	Call [method Spatial.hide].	See *hide*.
[member membername]	Link to a member in this class	Get [member scale].	Get *scale*.
[member Class.membername]	Link to another class's member	Get [member Node2D.scale].	Get *scale*.
[signal signalname]	Link to a signal in this class	Emit [signal renamed].	Emit *renamed*.
[signal Class.signalname]	Link to another class's signal	Emit [signal Node.renamed].	Emit *renamed*.
[b] [/b]	Bold	Some [b]bold[/b] text.	Some **bold** text.
[i] [/i]	Italic	Some [i]italic[/i] text.	Some *italic* text.
[code] [/code]	Monospace	Some [code]monospace[/code] text.	Some `monospace` text.
[codeblock] [/codeblock]	Multiline preformatted block	*See below.*	*See below.*

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.

28.2 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.

28.2.1 Q & A

- Official Godot Questions & Answers

28.2.2 IRC on Freenode

- General: #godotengine
- Engine development: #godotengine-devel
- Documentation: #godotengine-doc
- GDNative: #godotengine-gdnative
- Webseite/PR: #godotengine-atelier
- IRC logs

28.2.3 Other chats

- Matrix (IRC compatible)
- Discord

28.2.4 Social networks

- GitHub
- Facebook group
- Twitter (also, #godotengine)
- Reddit
- Youtube
- Steam

28.2.5 Forum

- Forum (godotdevelopers.org)

28.3 Tutorials

This is a list of third-party tutorials created by the community that may be of interest.

28.3.1 Video tutorials

- GDQuest
- KidsCanCode
- Game from Scratch: Godot 3 Tutorial Series
- HeartBeast

28.4 Resources

This is a list of third-party resources created by the community that may be of interest.

28.4.1 General

- awesome-godot: A curated list of resources by Calinou
- Zeef Godot Engine: A curated directory of resources by Andre Schmitz

CHAPTER 29

Godot API

29.1 @C#

Category: Core

29.1.1 Brief Description

29.2 @GDScript

Category: Core

29.2.1 Brief Description

Built-in GDScript functions.

29.2.2 Member Functions

Color	*Color8* (*int* r8, *int* g8, *int* b8, *int* a8=255)
Color	*ColorN* (*String* name, *float* alpha=1.0)
float	*abs* (*float* s)
float	*acos* (*float* s)
float	*asin* (*float* s)
void	*assert* (*bool* condition)
float	*atan* (*float* s)
float	*atan2* (*float* x, *float* y)
Variant	*bytes2var* (*PoolByteArray* bytes)
Vector2	*cartesian2polar* (*float* x, *float* y)

Continued on next page

Table 1 – continued from previous page

float	*ceil* (*float* s)
String	*char* (*int* ascii)
float	*clamp* (*float* value, *float* min, *float* max)
Object	*convert* (*Variant* what, *int* type)
float	*cos* (*float* s)
float	*cosh* (*float* s)
float	*db2linear* (*float* db)
float	*decimals* (*float* step)
float	*dectime* (*float* value, *float* amount, *float* step)
float	*deg2rad* (*float* deg)
Object	*dict2inst* (*Dictionary* dict)
float	*ease* (*float* s, *float* curve)
float	*exp* (*float* s)
float	*floor* (*float* s)
float	*fmod* (*float* x, *float* y)
float	*fposmod* (*float* x, *float* y)
Object	*funcref* (*Object* instance, *String* funcname)
int	*hash* (*Variant* var)
Dictionary	*inst2dict* (*Object* inst)
Object	*instance_from_id* (*int* instance_id)
float	*inverse_lerp* (*float* from, *float* to, *float* weight)
bool	*is_inf* (*float* s)
bool	*is_instance_valid* (*Object* instance)
bool	*is_nan* (*float* s)
int	*len* (*Variant* var)
float	*lerp* (*Variant* from, *Variant* to, *float* weight)
float	*linear2db* (*float* nrg)
Object	*load* (*String* path)
float	*log* (*float* s)
float	*max* (*float* a, *float* b)
float	*min* (*float* a, *float* b)
int	*nearest_po2* (*int* value)
Variant	*parse_json* (*String* json)
Vector2	*polar2cartesian* (*float* r, *float* th)
float	*pow* (*float* x, *float* y)
Resource	*preload* (*String* path)
void	*print* ( ) vararg
void	*print_stack* ( )
void	*printerr* ( ) vararg
void	*printraw* ( ) vararg
void	*prints* ( ) vararg
void	*printt* ( ) vararg
float	*rad2deg* (*float* rad)
float	*rand_range* (*float* from, *float* to)
Array	*rand_seed* (*int* seed)
float	*randf* ( )
int	*randi* ( )
void	*randomize* ( )
Array	*range* ( ) vararg
float	*range_lerp* (*float* value, *float* istart, *float* istop, *float* ostart, *float* ostop)

Continued on next page

Table 1 – continued from previous page

float	*round* (*float* s)
void	*seed* (*int* seed)
float	*sign* (*float* s)
float	*sin* (*float* s)
float	*sinh* (*float* s)
float	*sqrt* (*float* s)
float	*stepify* (*float* s, *float* step)
String	*str* ( ) vararg
Variant	*str2var* (*String* string)
float	*tan* (*float* s)
float	*tanh* (*float* s)
String	*to_json* (*Variant* var)
bool	*type_exists* (*String* type)
int	*typeof* (*Variant* what)
String	*validate_json* (*String* json)
PoolByteArray	*var2bytes* (*Variant* var)
String	*var2str* (*Variant* var)
Object	*weakref* (*Object* obj)
float	*wrapf* (*float* value, *float* min, *float* max)
int	*wrapi* (*int* value, *int* min, *int* max)
GDScriptFunctionState	*yield* (*Object* object=null, *String* signal="")

29.2.3 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.

29.2.4 Description

This contains the list of built-in gdscript functions. Mostly math functions and other utilities. Everything else is expanded by objects.

29.2.5 Member Function Description

- *Color* **Color8** (*int* r8, *int* g8, *int* b8, *int* a8=255)

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=1.0)**

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())
```

(continues on next page)

(continued from previous page)

```
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)

The natural exponential function. It raises the mathematical 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_instance_valid** (*Object* instance)
- *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** (*Variant* from, *Variant* 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.

Note that JSON objects do not preserve key order like Godot dictionaries, thus you should not rely on keys being in a certain order if a dictionary is constructed from JSON. In contrast, JSON arrays retain the order of their elements:

```
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=null, *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.

29.3 @GlobalScope

Category: Core

29.3.1 Brief Description

Global scope constants and variables.

29.3.2 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

29.3.3 Numeric Constants

- **SPKEY = 16777216** — Scancodes with this bit applied are non printable.

29.3.4 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** — ¼ key
- **KEY_ONEHALF = 189** — ½ key
- **KEY_THREEQUARTERS = 190** — ¾ 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** — ÿ 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

29.3.5 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.

29.4 @NativeScript

Category: Core

29.4.1 Brief Description

29.5 @VisualScript

Category: Core

29.5.1 Brief Description

Built-in visual script functions.

29.5.2 Description

A list of built-in visual script functions, see *VisualScriptBuiltinFunc* and *VisualScript*.

29.6 AABB

Category: Built-In Types

29.6.1 Brief Description

Axis-Aligned Bounding Box.

29.6.2 Member Functions

AABB	*AABB* (*Vector3* position, *Vector3* size)
bool	*encloses* (*AABB* with)
AABB	*expand* (*Vector3* to_point)
float	*get_area* ( )
Vector3	*get_endpoint* (*int* idx)
Vector3	*get_longest_axis* ( )
int	*get_longest_axis_index* ( )
float	*get_longest_axis_size* ( )
Vector3	*get_shortest_axis* ( )
int	*get_shortest_axis_index* ( )
float	*get_shortest_axis_size* ( )
Vector3	*get_support* (*Vector3* dir)
AABB	*grow* (*float* by)
bool	*has_no_area* ( )
bool	*has_no_surface* ( )
bool	*has_point* (*Vector3* point)
AABB	*intersection* (*AABB* with)
bool	*intersects* (*AABB* with)
bool	*intersects_plane* (*Plane* plane)
bool	*intersects_segment* (*Vector3* from, *Vector3* to)
AABB	*merge* (*AABB* with)

29.6.3 Member Variables

- *Vector3* **end** - Ending corner.
- *Vector3* **position** - Beginning corner.
- *Vector3* **size** - Size from position to end.

29.6.4 Description

AABB consists of a position, a size, and several utility functions. It is typically used for fast overlap tests.

29.6.5 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.

29.7 AcceptDialog

Inherits: *WindowDialog* < *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *ConfirmationDialog*

Category: Core

29.7.1 Brief Description

Base dialog for user notification.

29.7.2 Member Functions

Button	*add_button* (*String* text, *bool* right=false, *String* action="")
Button	*add_cancel* (*String* name)
Label	*get_label* ( )
Button	*get_ok* ( )
void	*register_text_enter* (*Node* line_edit)

29.7.3 Signals

- **confirmed** ( )

Emitted when the dialog is accepted.

- **custom_action** (*String* action)

Emitted when a custom button is pressed. See *add_button*.

29.7.4 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.

29.7.5 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.

29.7.6 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.

29.8 AnimatedSprite

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.8.1 Brief Description

Sprite node that can use multiple textures for animation.

29.8.2 Member Functions

bool	*is_playing* ( ) const
void	*play* (*String* anim="")
void	*stop* ( )

29.8.3 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.

29.8.4 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.
- *float* **speed_scale**

29.8.5 Description

Animations are created using a *SpriteFrames* resource, which can be configured in the editor via the SpriteFrames panel.

29.8.6 Member Function Description

- *bool* **is_playing** ( ) const

Return true if an animation if 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).

29.9 AnimatedSprite3D

Inherits: *SpriteBase3D* < *GeometryInstance* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.9.1 Brief Description

2D sprite node in 3D world, that can use multiple 2D textures for animation.

29.9.2 Member Functions

bool	*is_playing* ( ) const
void	*play* (*String* anim="")
void	*stop* ( )

29.9.3 Signals

- **frame_changed** ( )

Emitted when *frame* changed.

29.9.4 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.

29.9.5 Description

Animations are created using a *SpriteFrames* resource, which can be configured in the editor via the SpriteFrames panel.

29.9.6 Member Function Description

- *bool* **is_playing** ( ) const

Return true if an animation if 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).

29.10 Animation

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.10.1 Brief Description

Contains data used to animate everything in the engine.

29.10.2 Member Functions

int	*add_track* (*TrackType* type, *int* at_position=-1)
void	*clear* ( )
void	*copy_track* (*int* track, *Animation* to_animation)
int	*find_track* (*NodePath* path) const
int	*get_track_count* ( ) const
PoolIntArray	*method_track_get_key_indices* (*int* idx, *float* time_sec, *float* delta) const
String	*method_track_get_name* (*int* idx, *int* key_idx) const
Array	*method_track_get_params* (*int* idx, *int* key_idx) const
void	*remove_track* (*int* idx)
int	*track_find_key* (*int* idx, *float* time, *bool* exact=false) const
bool	*track_get_interpolation_loop_wrap* (*int* idx) const
InterpolationType	*track_get_interpolation_type* (*int* idx) const
int	*track_get_key_count* (*int* idx) const
float	*track_get_key_time* (*int* idx, *int* key_idx) const
float	*track_get_key_transition* (*int* idx, *int* key_idx) const
Variant	*track_get_key_value* (*int* idx, *int* key_idx) const
NodePath	*track_get_path* (*int* idx) const
TrackType	*track_get_type* (*int* idx) const
void	*track_insert_key* (*int* idx, *float* time, *Variant* key, *float* transition=1)
bool	*track_is_enabled* (*int* idx) const
bool	*track_is_imported* (*int* idx) const
void	*track_move_down* (*int* idx)
void	*track_move_up* (*int* idx)
void	*track_remove_key* (*int* idx, *int* key_idx)
void	*track_remove_key_at_position* (*int* idx, *float* position)
void	*track_set_enabled* (*int* idx, *bool* enabled)
void	*track_set_imported* (*int* idx, *bool* imported)
void	*track_set_interpolation_loop_wrap* (*int* idx, *bool* interpolation)
void	*track_set_interpolation_type* (*int* idx, *InterpolationType* interpolation)
void	*track_set_key_transition* (*int* idx, *int* key_idx, *float* transition)
void	*track_set_key_value* (*int* idx, *int* key, *Variant* value)
void	*track_set_path* (*int* idx, *NodePath* path)
int	*transform_track_insert_key* (*int* idx, *float* time, *Vector3* location, *Quat* rotation, *Vector3* scale)
Array	*transform_track_interpolate* (*int* idx, *float* time_sec) const
PoolIntArray	*value_track_get_key_indices* (*int* idx, *float* time_sec, *float* delta) const
UpdateMode	*value_track_get_update_mode* (*int* idx) const
void	*value_track_set_update_mode* (*int* idx, *UpdateMode* mode)

29.10.3 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 uses for correct interpolation of animation cycles, and for hinting the player that it must restart the animation.
- *float* **step** - The animation step value.

29.10.4 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.

29.10.5 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 odes such as an *AnimationPlayer* or *AnimationTreePlayer* to be played back.

29.10.6 Member Function Description

- *int* **add_track** (*TrackType* 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`.

- *InterpolationType* **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*

- *TrackType* **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, *InterpolationType* 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.

- *UpdateMode* **value_track_get_update_mode** (*int* idx) const

Return the update mode of a value track.

- void **value_track_set_update_mode** (*int* idx, *UpdateMode* mode)

Set the update mode (UPDATE_*) of a value track.

29.11 AnimationPlayer

Inherits: *Node* < *Object*

Category: Core

29.11.1 Brief Description

Container and player of *Animation* resources.

29.11.2 Member Functions

Error	*add_animation* (*String* name, *Animation* animation)
void	*advance* (*float* delta)
String	*animation_get_next* (*String* anim_from) const
void	*animation_set_next* (*String* anim_from, *String* anim_to)
void	*clear_caches* ( )
void	*clear_queue* ( )
String	*find_animation* (*Animation* animation) const
Animation	*get_animation* (*String* name) const
PoolStringArray	*get_animation_list* ( ) const
float	*get_blend_time* (*String* anim_from, *String* anim_to) const
float	*get_playing_speed* ( ) const
bool	*has_animation* (*String* name) const
bool	*is_playing* ( ) const
void	*play* (*String* name="", *float* custom_blend=-1, *float* custom_speed=1.0, *bool* from_end=false)
void	*play_backwards* (*String* name="", *float* custom_blend=-1)
void	*queue* (*String* name)
void	*remove_animation* (*String* name)
void	*rename_animation* (*String* name, *String* newname)
void	*seek* (*float* seconds, *bool* update=false)
void	*set_blend_time* (*String* anim_from, *String* anim_to, *float* sec)
void	*stop* (*bool* reset=true)

29.11.3 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.

29.11.4 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: 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: `".."`.

29.11.5 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.

29.11.6 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.

29.11.7 Member Function Description

- *Error* **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.

- *float* **get_playing_speed** ( ) const

Get the actual playing speed of current animation or 0 if not playing. This speed is the `playback_speed` property multiplied by `custom_speed` argument specified when calling the `play` method.

- *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`.

29.12 AnimationTreePlayer

Inherits: *Node* < *Object*

Category: Core

29.12.1 Brief Description

Animation Player that uses a node graph for blending Animations.

29.12.2 Member Functions

void	*add_node* (*NodeType* type, *String* id)
void	*advance* (*float* delta)
Animation	*animation_node_get_animation* (*String* id) const
String	*animation_node_get_master_animation* (*String* id) const
float	*animation_node_get_position* (*String* id) const
void	*animation_node_set_animation* (*String* id, *Animation* animation)
void	*animation_node_set_filter_path* (*String* id, *NodePath* path, *bool* enable)
void	*animation_node_set_master_animation* (*String* id, *String* source)
bool	*are_nodes_connected* (*String* id, *String* dst_id, *int* dst_input_idx) const
float	*blend2_node_get_amount* (*String* id) const
void	*blend2_node_set_amount* (*String* id, *float* blend)
void	*blend2_node_set_filter_path* (*String* id, *NodePath* path, *bool* enable)
float	*blend3_node_get_amount* (*String* id) const
void	*blend3_node_set_amount* (*String* id, *float* blend)
Vector2	*blend4_node_get_amount* (*String* id) const
void	*blend4_node_set_amount* (*String* id, *Vector2* blend)
Error	*connect_nodes* (*String* id, *String* dst_id, *int* dst_input_idx)
void	*disconnect_nodes* (*String* id, *int* dst_input_idx)
PoolStringArray	*get_node_list* ( )
float	*mix_node_get_amount* (*String* id) const
void	*mix_node_set_amount* (*String* id, *float* ratio)
bool	*node_exists* (*String* node) const
int	*node_get_input_count* (*String* id) const
String	*node_get_input_source* (*String* id, *int* idx) const
Vector2	*node_get_position* (*String* id) const
NodeType	*node_get_type* (*String* id) const
Error	*node_rename* (*String* node, *String* new_name)
void	*node_set_position* (*String* id, *Vector2* screen_position)
float	*oneshot_node_get_autorestart_delay* (*String* id) const
float	*oneshot_node_get_autorestart_random_delay* (*String* id) const
float	*oneshot_node_get_fadein_time* (*String* id) const
float	*oneshot_node_get_fadeout_time* (*String* id) const
bool	*oneshot_node_has_autorestart* (*String* id) const
bool	*oneshot_node_is_active* (*String* id) const

Continued on next page

Table 3 – continued from previous page

void	*oneshot_node_set_autorestart* (*String* id, *bool* enable)
void	*oneshot_node_set_autorestart_delay* (*String* id, *float* delay_sec)
void	*oneshot_node_set_autorestart_random_delay* (*String* id, *float* rand_sec)
void	*oneshot_node_set_fadein_time* (*String* id, *float* time_sec)
void	*oneshot_node_set_fadeout_time* (*String* id, *float* time_sec)
void	*oneshot_node_set_filter_path* (*String* id, *NodePath* path, *bool* enable)
void	*oneshot_node_start* (*String* id)
void	*oneshot_node_stop* (*String* id)
void	*recompute_caches* ( )
void	*remove_node* (*String* id)
void	*reset* ( )
float	*timescale_node_get_scale* (*String* id) const
void	*timescale_node_set_scale* (*String* id, *float* scale)
void	*timeseek_node_seek* (*String* id, *float* seconds)
void	*transition_node_delete_input* (*String* id, *int* input_idx)
int	*transition_node_get_current* (*String* id) const
int	*transition_node_get_input_count* (*String* id) const
float	*transition_node_get_xfade_time* (*String* id) const
bool	*transition_node_has_input_auto_advance* (*String* id, *int* input_idx) const
void	*transition_node_set_current* (*String* id, *int* input_idx)
void	*transition_node_set_input_auto_advance* (*String* id, *int* input_idx, *bool* enable)
void	*transition_node_set_input_count* (*String* id, *int* count)
void	*transition_node_set_xfade_time* (*String* id, *float* time_sec)

29.12.3 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: ANIMATION_PROCESS_IDLE.

29.12.4 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.

29.12.5 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 *Animation*s from an *AnimationPlayer* node and mixes them depending on the graph.

29.12.6 Member Function Description

- void **add_node** (*NodeType* 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.

- *float* **animation_node_get_position** (*String* id) const
- 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.

- *Error* **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.

- *NodeType* **node_get_type** (*String* id) const

Get the node type, will return from NODE_* enum.

- *Error* **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 *Animation*s 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 *Animation*s 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`.

29.13 Area

Inherits: *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

29.13.1 Brief Description

General purpose area node for detection and 3D physics influence.

29.13.2 Member Functions

bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
Array	*get_overlapping_areas* ( ) const
Array	*get_overlapping_bodies* ( ) const
bool	*overlaps_area* (*Node* area) const
bool	*overlaps_body* (*Node* body) const
void	*set_collision_layer_bit* (*int* bit, *bool* value)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.13.3 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 *PhysicsBody* object exits.

- **body_shape_entered** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody* object enters, reporting which shapes overlapped.

- **body_shape_exited** (*int* body_id, *Object* body, *int* body_shape, *int* area_shape)

Emitted when a *PhysicsBody* object exits, reporting which shapes were overlapping.

29.13.4 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.

29.13.5 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.

29.13.6 Description

3D area that detects *CollisionObject* nodes overlapping, entering, or exiting. Can also alter or override local physics parameters (gravity, damping).

29.13.7 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 *PhysicsBody*s. 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.

29.14 Area2D

Inherits: *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.14.1 Brief Description

2D area for detection and 2D physics influence.

29.14.2 Member Functions

bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
Array	*get_overlapping_areas* ( ) const
Array	*get_overlapping_bodies* ( ) const
bool	*overlaps_area* (*Node* area) const
bool	*overlaps_body* (*Node* body) const
void	*set_collision_layer_bit* (*int* bit, *bool* value)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.14.3 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.

29.14.4 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.

29.14.5 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.

29.14.6 Description

2D area that detects *CollisionObject2D* nodes overlapping, entering, or exiting. Can also alter or override local physics parameters (gravity, damping).

29.14.7 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 `Area2D`s. 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 *PhysicsBody2D*s. 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.

29.15 Array

Category: Built-In Types

29.15.1 Brief Description

Generic array datatype.

29.15.2 Member Functions

Array	Array (*PoolColorArray* from)

Continued on next page

Table 4 – continued from previous page

Array	Array (*PoolVector3Array* from)
Array	Array (*PoolVector2Array* from)
Array	Array (*PoolStringArray* from)
Array	Array (*PoolRealArray* from)
Array	Array (*PoolIntArray* from)
Array	Array (*PoolByteArray* from)
void	*append* (var value)
var	*back* ( )
int	*bsearch* (var value, *bool* before=True)
int	*bsearch_custom* (var value, *Object* obj, *String* func, *bool* before=True)
void	*clear* ( )
int	*count* (var value)
Array	*duplicate* (*bool* deep=False)
bool	*empty* ( )
void	*erase* (var value)
int	*find* (var what, *int* from=0)
int	*find_last* (var value)
var	*front* ( )
bool	*has* (var value)
int	*hash* ( )
void	*insert* (*int* position, var value)
void	*invert* ( )
var	*pop_back* ( )
var	*pop_front* ( )
void	*push_back* (var value)
void	*push_front* (var value)
void	*remove* (*int* position)
void	*resize* (*int* size)
int	*rfind* (var what, *int* from=-1)
void	*shuffle* ( )
int	*size* ( )
void	*sort* ( )
void	*sort_custom* (*Object* obj, *String* func)

29.15.3 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.

29.15.4 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** (*bool* deep=False)

Returns a copy of the array.

If `deep` is `true`, a deep copy is be performed: all nested arrays and dictionaries are duplicated and will not be shared with the original array. If `false`, a shallow copy is made and references to the original nested arrays and dictionaries are kept, so that modifying a sub-array or dictionary in the copy will also impact those referenced in the source 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.

- void **shuffle** ( )

Shuffle the array such that the items will have a random order.

- *int* **size** ( )

Return the amount of elements in the array.

- void **sort** ( )

Sort the array using natural order.

- void **sort_custom** (*Object* obj, *String* func)

Sort the array using a custom method. 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")
```

29.16 ArrayMesh

Inherits: *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.16.1 Brief Description

29.16.2 Member Functions

void	*add_blend_shape* (*String* name)
void	*add_surface_from_arrays* (*PrimitiveType* primitive, *Array* arrays, *Array* blend_shapes=[ ], *int* compress_flags=97792)
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
Error	*lightmap_unwrap* (*Transform* arg0, *float* arg1)
void	*regen_normalmaps* ( )
int	*surface_get_array_index_len* (*int* surf_idx) const
int	*surface_get_array_len* (*int* surf_idx) const
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
Material	*surface_get_material* (*int* surf_idx) const
String	*surface_get_name* (*int* surf_idx) const
PrimitiveType	*surface_get_primitive_type* (*int* surf_idx) const
void	*surface_remove* (*int* surf_idx)
void	*surface_set_material* (*int* surf_idx, *Material* material)
void	*surface_set_name* (*int* surf_idx, *String* name)
void	*surface_update_region* (*int* surf_idx, *int* offset, *PoolByteArray* data)

29.16.3 Member Variables

- *BlendShapeMode* **blend_shape_mode**
- *AABB* **custom_aabb**

29.16.4 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).

29.16.5 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, colors, normals, tangents, and textures. All of those arrays must have the same number of elements as the vertex array. No index can be beyond the vertex array size. When this index array is present, it puts the function into "index mode," where the index selects the *i*'th vertex, normal, tangent, color, UV, etc. This means if you want to have different normals or colors along an edge, you have to duplicate the vertices.

For triangles, the index array is interpreted as triples, referring to the vertices of each triangle. For lines, the index array is in pairs indicating the start and end of each line. - **ARRAY_MAX = 9**

29.16.6 Member Function Description

- void **add_blend_shape** (*String* name)
- void **add_surface_from_arrays** (*PrimitiveType* primitive, *Array* arrays, *Array* blend_shapes=[ ], *int* compress_flags=97792)

Creates a new surface.

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. See *Mesh* for details. (As a note, when using indices, it is recommended to only use points, lines or triangles). *get_surface_count* will become the surf_idx for this new surface.

The `arrays` argument is an array of arrays. See ArrayType for the values used in this array. For example, `arrays[0]` is the array of vertices. That first vertex sub-array is always required; the others are optional. Adding an index array puts this function into "index mode" where the vertex and other arrays become the sources of data and the index array defines the vertex order. All sub-arrays must have the same length as the vertex array or be empty, except for `ARRAY_INDEX` if it is used.

Adding an index array puts this function into "index mode" where the vertex and other arrays become the sources of data, and the index array defines the order of the vertices.

Godot uses clockwise winding order for front faces of triangle primitive modes.

- 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.

- *Error* **lightmap_unwrap** (*Transform* arg0, *float* arg1)
- 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_from_arrays*).

- *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_from_arrays*).

- *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_from_arrays*).

- *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

- *PrimitiveType* **surface_get_primitive_type** (*int* surf_idx) const

Return the primitive type of the requested surface (see *add_surface_from_arrays*).

- 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)

29.17 ARVRAnchor

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.17.1 Brief Description

Anchor point in AR Space.

29.17.2 Member Functions

String	*get_anchor_name* ( ) const
bool	*get_is_active* ( ) const
Plane	*get_plane* ( ) const
Vector3	*get_size* ( ) const

29.17.3 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.

29.17.4 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.

29.17.5 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.

29.18 ARVRCamera

Inherits: *Camera* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.18.1 Brief Description

A camera node with a few overrules for AR/VR applied such as location tracking.

29.18.2 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.

29.19 ARVRController

Inherits: *Spatial* **<** *Node* **<** *Object*

Category: Core

29.19.1 Brief Description

A spatial node representing a spatially tracked controller.

29.19.2 Member Functions

String	*get_controller_name* ( ) const
TrackerHand	*get_hand* ( ) const
bool	*get_is_active* ( ) const
float	*get_joystick_axis* (*int* axis) const
int	*get_joystick_id* ( ) const
int	*is_button_pressed* (*int* button) const

29.19.3 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.

29.19.4 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.

29.19.5 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.

29.19.6 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.

- *TrackerHand* **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.

29.20 ARVRInterface

Inherits: *Reference* < *Object*

Inherited By: *ARVRInterfaceGDNative*, *MobileVRInterface*

Category: Core

29.20.1 Brief Description

Base class for ARVR interface implementation.

29.20.2 Member Functions

int	*get_capabilities* ( ) const
String	*get_name* ( ) const
Vector2	*get_render_targetsize* ( )
Tracking_status	*get_tracking_status* ( ) const
bool	*initialize* ( )
bool	*is_stereo* ( )
void	*uninitialize* ( )

29.20.3 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?

29.20.4 Enums

enum **Eyes**

- **EYE_MONO = 0** — Mono output, this is mostly used internally when retrieving positioning information for our camera node or when stereo scopic 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 then 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 support 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.

29.20.5 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.

29.20.6 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.

- *Tracking_status* **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 **uninitialize** **(** **)**

Turns the interface off.

29.21 ARVRInterfaceGDNative

Inherits: *ARVRInterface* **<** *Reference* **<** *Object*

Category: Core

29.21.1 Brief Description

GDNative wrapper for an ARVR interface

29.21.2 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.

29.22 ARVROrigin

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.22.1 Brief Description

Our origin point in AR/VR.

29.22.2 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.

29.22.3 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 ARVRAnchor 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 you're 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.

29.23 ARVRPositionalTracker

Inherits: *Object*

Category: Core

29.23.1 Brief Description

A tracked object

29.23.2 Member Functions

TrackerHand	*get_hand* ( ) const
int	*get_joy_id* ( ) const
String	*get_name* ( ) const
Basis	*get_orientation* ( ) const
Vector3	*get_position* ( ) const
bool	*get_tracks_orientation* ( ) const
bool	*get_tracks_position* ( ) const
Transform	*get_transform* (*bool* adjust_by_reference_frame) const
TrackerType	*get_type* ( ) const

29.23.3 Member Variables

- *float* **rumble** - The degree to which the tracker rumbles. Ranges from `0.0` to `1.0` with precision `.01`.

29.23.4 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.

29.23.5 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 ARVRController and ARVRAnchor 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.

29.23.6 Member Function Description

- *TrackerHand* **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.

- *TrackerType* **get_type** ( ) const

Returns the tracker's type.

29.24 ARVRServer

Inherits: *Object*

Category: Core

29.24.1 Brief Description

This is our AR/VR Server.

29.24.2 Member Functions

void	*center_on_hmd* (*RotationMode* rotation_mode, *bool* keep_height)
ARVRInterface	*find_interface* (*String* name) const
Transform	*get_hmd_transform* ( )
ARVRInterface	*get_interface* (*int* idx) const
int	*get_interface_count* ( ) const
Array	*get_interfaces* ( ) const
int	*get_last_commit_usec* ( )
int	*get_last_frame_usec* ( )
int	*get_last_process_usec* ( )
Transform	*get_reference_frame* ( ) const
ARVRPositionalTracker	*get_tracker* (*int* idx) const
int	*get_tracker_count* ( ) const

29.24.3 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 ARVRAnchors for an AR solution it is important to react to this signal and add the appropriate ARVRController or ARVRAnchor 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 ARVRAnchor 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).

29.24.4 Member Variables

- *ARVRInterface* **primary_interface**
- *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.

29.24.5 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.

29.24.6 Description

The AR/VR Server is the heart of our AR/VR solution and handles all the processing.

29.24.7 Member Function Description

- void **center_on_hmd** (*RotationMode* 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.

- *Transform* **get_hmd_transform** ( )

Returns the primary interface's transformation.

- *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.

- *int* **get_last_commit_usec** ( )
- *int* **get_last_frame_usec** ( )
- *int* **get_last_process_usec** ( )
- *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.

29.25 AStar

Inherits: *Reference* < *Object*

Category: Core

29.25.1 Brief Description

AStar class representation that uses vectors as edges.

29.25.2 Member Functions

float	*_compute_cost* (*int* from_id, *int* to_id) virtual
float	*_estimate_cost* (*int* from_id, *int* to_id) virtual
void	*add_point* (*int* id, *Vector3* position, *float* weight_scale=1.0)
bool	*are_points_connected* (*int* id, *int* to_id) const
void	*clear* ( )
void	*connect_points* (*int* id, *int* to_id, *bool* bidirectional=true)
void	*disconnect_points* (*int* id, *int* to_id)
int	*get_available_point_id* ( ) const
int	*get_closest_point* (*Vector3* to_position) const
Vector3	*get_closest_position_in_segment* (*Vector3* to_position) const
PoolIntArray	*get_id_path* (*int* from_id, *int* to_id)
PoolIntArray	*get_point_connections* (*int* id)
PoolVector3Array	*get_point_path* (*int* from_id, *int* to_id)
Vector3	*get_point_position* (*int* id) const
float	*get_point_weight_scale* (*int* id) const
Array	*get_points* ( )
bool	*has_point* (*int* id) const
void	*remove_point* (*int* id)
void	*set_point_position* (*int* id, *Vector3* position)
void	*set_point_weight_scale* (*int* id, *float* weight_scale)

29.25.3 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*.

29.25.4 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.

29.26 AtlasTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.26.1 Brief Description

Packs multiple small textures in a single, bigger one. Helps to optimize video memory costs and render calls.

29.26.2 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.

29.26.3 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.

29.27 AudioBusLayout

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.27.1 Brief Description

Stores information about the audiobusses.

29.27.2 Description

Stores position, muting, solo, bypass, effects, effect position, volume, and the connections between busses. See *AudioServer* for usage.

29.28 AudioEffect

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *AudioEffectCompressor*, *AudioEffectLimiter*, *AudioEffectAmplify*, *AudioEffectFilter*, *AudioEffectDistortion*, *AudioEffectDelay*, *AudioEffectStereoEnhance*, *AudioEffectReverb*, *AudioEffectPanner*, *AudioEffectEQ*, *AudioEffectPitchShift*, *AudioEffectChorus*, *AudioEffectPhaser*

Category: Core

29.28.1 Brief Description

Audio Effect For Audio.

29.28.2 Description

Base resource for audio bus. Applies an audio effect on the bus that the resource is applied on.

29.29 AudioEffectAmplify

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.29.1 Brief Description

Adds a Amplify audio effect to an Audio bus.

Increases or decreases the volume of the selected audio bus.

29.29.2 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`.

29.29.3 Description

Increases or decreases the volume being routed through the audio bus.

29.30 AudioEffectBandLimitFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.30.1 Brief Description

Adds a band limit filter to the Audio Bus.

29.30.2 Description

Limits the frequencies in a range around the cutoff_hz and allows frequencies outside of this range to pass.

29.31 AudioEffectBandPassFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.31.1 Brief Description

Adds a band pass filter to the Audio Bus.

29.31.2 Description

Attenuates the frequencies inside of a range around the cutoff_hz and cuts frequencies outside of this band.

29.32 AudioEffectChorus

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.32.1 Brief Description

Adds a chorus audio effect.

29.32.2 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.

29.32.3 Description

Adds a chorus audio effect. The effect applies a filter with voices to duplicate the audio source and manipulate it through the filter.

29.33 AudioEffectCompressor

Inherits: *AudioEffect* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.33.1 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.

29.33.2 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`.

29.33.3 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.

29.34 AudioEffectDelay

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.34.1 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.

29.34.2 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`.

29.34.3 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.

29.35 AudioEffectDistortion

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.35.1 Brief Description

Adds a Distortion audio effect to an Audio bus.

Modify the sound to make it dirty.

29.35.2 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`.

29.35.3 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.

29.35.4 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.

29.36 AudioEffectEQ

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Inherited By: *AudioEffectEQ6*, *AudioEffectEQ21*, *AudioEffectEQ10*

Category: Core

29.36.1 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.

29.36.2 Member Functions

int	*get_band_count* ( ) const
float	*get_band_gain_db* (*int* band_idx) const
void	*set_band_gain_db* (*int* band_idx, *float* volume_db)

29.36.3 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).

29.36.4 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.

29.37 AudioEffectEQ10

Inherits: *AudioEffectEQ* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.37.1 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.

29.37.2 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*.

29.38 AudioEffectEQ21

Inherits: *AudioEffectEQ* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.38.1 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.

29.38.2 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*.

29.39 AudioEffectEQ6

Inherits: *AudioEffectEQ* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.39.1 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.

29.39.2 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*.

29.40 AudioEffectFilter

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Inherited By: *AudioEffectHighPassFilter*, *AudioEffectBandLimitFilter*, *AudioEffectLowShelfFilter*, *AudioEffectHighShelfFilter*, *AudioEffectBandPassFilter*, *AudioEffectNotchFilter*, *AudioEffectLowPassFilter*

Category: Core

29.40.1 Brief Description

Adds a filter to the Audio Bus.

29.40.2 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.

29.40.3 Enums

enum **FilterDB**

- **FILTER_6DB = 0**
- **FILTER_12DB = 1**
- **FILTER_18DB = 2**
- **FILTER_24DB = 3**

29.40.4 Description

Allows frequencies other than the *cutoff_hz* to pass.

29.41 AudioEffectHighPassFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.41.1 Brief Description

Adds a high pass filter to the Audio Bus.

29.41.2 Description

Cuts frequencies lower than the cutoff_hz and allows higher frequencies to pass.

29.42 AudioEffectHighShelfFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.42.1 Brief Description

29.43 AudioEffectLimiter

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.43.1 Brief Description

Adds a soft clip Limiter audio effect to an Audio bus.

29.43.2 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`.

29.43.3 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.

29.44 AudioEffectLowPassFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.44.1 Brief Description

Adds a low pass filter to the Audio Bus.

29.44.2 Description

Cuts frequencies higher than the cutoff_hz and allows lower frequencies to pass.

29.45 AudioEffectLowShelfFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.45.1 Brief Description

29.46 AudioEffectNotchFilter

Inherits: *AudioEffectFilter* < *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.46.1 Brief Description

Adds a notch filter to the Audio Bus.

29.46.2 Description

Attenuates frequencies in a narrow band around the cutoff_hz and cuts frequencies outside of this range.

29.47 AudioEffectPanner

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.47.1 Brief Description

Adds a Panner audio effect to an Audio bus. Pans sound left or right.

29.47.2 Member Variables

- *float* **pan** - Pan position. Value can range from -1 (fully left) to 1 (fully right).

29.47.3 Description

Determines how much of an audio signal is sent to the left and right buses.

29.48 AudioEffectPhaser

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.48.1 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.

29.48.2 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.

29.48.3 Description

Combines phase-shifted signals with the original signal. The movement of the phase-shifted signals is controlled using a Low Frequency Oscillator.

29.49 AudioEffectPitchShift

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.49.1 Brief Description

Adds a Pitch shift audio effect to an Audio bus.

Raises or lowers the pitch of original sound.

29.49.2 Member Variables

- *float* **pitch_scale** - Pitch value. Can range from 0 (-1 octave) to 16 (+16 octaves).

29.49.3 Description

Allows modulation of pitch independently of tempo. All frequencies can be increased/decreased with minimal effect on transients.

29.50 AudioEffectReverb

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.50.1 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.

29.50.2 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`.

29.50.3 Description

Simulates rooms of different sizes. Its parameters can be adjusted to simulate the sound of a specific room.

29.51 AudioEffectStereoEnhance

Inherits: *AudioEffect* < *Resource* < *Reference* < *Object*

Category: Core

29.51.1 Brief Description

29.51.2 Member Variables

- *float* **pan_pullout**
- *float* **surround**
- *float* **time_pullout_ms**

29.52 AudioServer

Inherits: *Object*

Category: Core

29.52.1 Brief Description

Server interface for low level audio access.

29.52.2 Member Functions

void	*add_bus* (*int* at_position=-1)
void	*add_bus_effect* (*int* bus_idx, *AudioEffect* effect, *int* at_position=-1)
AudioBusLayout	*generate_bus_layout* ( ) const
int	*get_bus_count* ( ) const
AudioEffect	*get_bus_effect* (*int* bus_idx, *int* effect_idx)
int	*get_bus_effect_count* (*int* bus_idx)
int	*get_bus_index* (*String* bus_name) const
String	*get_bus_name* (*int* bus_idx) const
float	*get_bus_peak_volume_left_db* (*int* bus_idx, *int* channel) const
float	*get_bus_peak_volume_right_db* (*int* bus_idx, *int* channel) const
String	*get_bus_send* (*int* bus_idx) const
float	*get_bus_volume_db* (*int* bus_idx) const
String	*get_device* ( )
Array	*get_device_list* ( )
float	*get_mix_rate* ( ) const
SpeakerMode	*get_speaker_mode* ( ) const
bool	*is_bus_bypassing_effects* (*int* bus_idx) const
bool	*is_bus_effect_enabled* (*int* bus_idx, *int* effect_idx) const
bool	*is_bus_mute* (*int* bus_idx) const
bool	*is_bus_solo* (*int* bus_idx) const
void	*lock* ( )
void	*move_bus* (*int* index, *int* to_index)
void	*remove_bus* (*int* index)
void	*remove_bus_effect* (*int* bus_idx, *int* effect_idx)
void	*set_bus_bypass_effects* (*int* bus_idx, *bool* enable)
void	*set_bus_count* (*int* amount)

Continued on next page

Table 5 – continued from previous page

void	*set_bus_effect_enabled* (*int* bus_idx, *int* effect_idx, *bool* enabled)
void	*set_bus_layout* (*AudioBusLayout* bus_layout)
void	*set_bus_mute* (*int* bus_idx, *bool* enable)
void	*set_bus_name* (*int* bus_idx, *String* name)
void	*set_bus_send* (*int* bus_idx, *String* send)
void	*set_bus_solo* (*int* bus_idx, *bool* enable)
void	*set_bus_volume_db* (*int* bus_idx, *float* volume_db)
void	*set_device* (*String* arg0)
void	*swap_bus_effects* (*int* bus_idx, *int* effect_idx, *int* by_effect_idx)
void	*unlock* ( )

29.52.3 Signals

- **bus_layout_changed** ( )

Emitted when the *AudioBusLayout* changes.

29.52.4 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.

29.52.5 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.

29.52.6 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.

- *String* **get_device** ( )
- *Array* **get_device_list** ( )
- *float* **get_mix_rate** ( ) const

Returns the sample rate at the output of the audioserver.

- *SpeakerMode* **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 **set_device** (*String* arg0)
- 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 audiodriver's main loop. After locking it always unlock it.

29.53 AudioStream

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *AudioStreamSample*, *AudioStreamRandomPitch*, *AudioStreamOGGVorbis*

Category: Core

29.53.1 Brief Description

Base class for audio streams.

29.53.2 Member Functions

float	*get_length* ( ) const

29.53.3 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.

29.53.4 Member Function Description

- *float* **get_length** **( )** const

29.54 AudioStreamOGGVorbis

Inherits: *AudioStream* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.54.1 Brief Description

OGG Vorbis audio stream driver.

29.54.2 Member Variables

- *PoolByteArray* **data** - Contains the audio data in bytes.
- *bool* **loop**
- *float* **loop_offset**

29.54.3 Description

OGG Vorbis audio stream driver.

29.55 AudioStreamPlayback

Inherits: *Reference* **<** *Object*

Category: Core

29.55.1 Brief Description

Meta class for playing back audio.

29.55.2 Description

Can play, loop, pause a scroll through Audio. See *AudioStream* and *AudioStreamOGGVorbis* for usage.

29.56 AudioStreamPlayer

Inherits: *Node* < *Object*

Category: Core

29.56.1 Brief Description

Plays back audio.

29.56.2 Member Functions

float	*get_playback_position* ( )
void	*play* (*float* from_position=0.0)
void	*seek* (*float* to_position)
void	*stop* ( )

29.56.3 Signals

- **finished** ( )

Emitted when the audio stops playing.

29.56.4 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.
- *float* **pitch_scale** - Changes the pitch and the tempo of the audio.
- *bool* **playing** - If `true` audio is playing.
- *AudioStream* **stream** - The *AudioStream* object to be played.
- *float* **volume_db** - Volume of sound, in dB.

29.56.5 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.

29.56.6 Description

Plays background audio.

29.56.7 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.

29.57 AudioStreamPlayer2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.57.1 Brief Description

Plays audio in 2D.

29.57.2 Member Functions

float	*get_playback_position* ( )
void	*play* (*float* from_position=0.0)
void	*seek* (*float* to_position)
void	*stop* ( )

29.57.3 Signals

- **finished** ( )

Emitted when the audio stops playing.

29.57.4 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.
- *float* **pitch_scale** - Changes the pitch and the tempo of the audio.
- *bool* **playing** - If `true` audio is playing.
- *AudioStream* **stream** - The *AudioStream* object to be played.
- *float* **volume_db** - Base volume without dampening.

29.57.5 Description

Plays audio that dampens with distance from screen center.

29.57.6 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.

29.58 AudioStreamPlayer3D

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.58.1 Brief Description

Plays 3D sound in 3D space.

29.58.2 Member Functions

float	*get_playback_position* ( )
void	*play* (*float* from_position=0.0)
void	*seek* (*float* to_position)
void	*stop* ( )

29.58.3 Signals

- **finished** ( )

Fires when the audio stops playing.

29.58.4 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 dampened 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.
- *float* **pitch_scale** - Changes the pitch and the tempo of the audio.
- *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.

29.58.5 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.

29.58.6 Description

Plays a sound effect with directed sound effects, dampens with distance if needed, generates effect of hearable position in space.

29.58.7 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.

29.59 AudioStreamRandomPitch

Inherits: *AudioStream* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.59.1 Brief Description

Plays audio with random pitch tweaking.

29.59.2 Member Variables

- *AudioStream* **audio_stream** - The current *AudioStream*.
- *float* **random_pitch** - The intensity of random pitch variation.

29.59.3 Description

Randomly varies pitch on each start.

29.60 AudioStreamSample

Inherits: *AudioStream* < *Resource* < *Reference* < *Object*

Category: Core

29.60.1 Brief Description

Plays audio.

29.60.2 Member Variables

- *PoolByteArray* **data** - Contains the audio data in bytes.
- *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`.

29.60.3 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.

29.60.4 Description

Plays audio, can loop.

29.61 BackBufferCopy

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.61.1 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.

29.61.2 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`.

29.61.3 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.

29.61.4 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.

29.62 BakedLightmap

Inherits: *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.62.1 Brief Description

29.62.2 Member Functions

BakeError	*bake* (*Node* from_node=null, *bool* create_visual_debug=false)
void	*debug_bake* ( )

29.62.3 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**

29.62.4 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**

29.62.5 Member Function Description

- *BakeError* **bake** (*Node* from_node=null, *bool* create_visual_debug=false)
- void **debug_bake** ( )

29.63 BakedLightmapData

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.63.1 Brief Description

29.63.2 Member Functions

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

29.63.3 Member Variables

- *AABB* **bounds**
- *Transform* **cell_space_transform**
- *int* **cell_subdiv**
- *float* **energy**
- *PoolByteArray* **octree**

29.63.4 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

29.64 BaseButton

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *LinkButton*, *TextureButton*, *Button*

Category: Core

29.64.1 Brief Description

Base class for different kinds of buttons.

29.64.2 Member Functions

void	*_pressed* ( ) virtual
void	*_toggled* (*bool* button_pressed) virtual
DrawMode	*get_draw_mode* ( ) const
bool	*is_hovered* ( ) const

29.64.3 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.

29.64.4 Member Variables

- *ActionMode* **action_mode** - Determines when the button is considered clicked, one of the ACTION_MODE_* constants.
- *int* **button_mask** - Binary mask to choose which mouse buttons this button will respond to.

To allow both left-click and right-click, set this to 3, because it's BUTTON_MASK_LEFT | BUTTON_MASK_RIGHT.

- *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.

29.64.5 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.

29.64.6 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.

29.64.7 Member Function Description

- void **_pressed** **(** **)** virtual

Called when the button is pressed.

- void **_toggled** **(** *bool* button_pressed **)** virtual

Called when the button is toggled (only if toggle_mode is active).

- *DrawMode* **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 the mouse has entered the button and has not left it yet.

29.65 Basis

Category: Built-In Types

29.65.1 Brief Description

3x3 matrix datatype.

29.65.2 Member Functions

Basis	Basis (*Quat* from)
Basis	Basis (*Vector3* from)
Basis	Basis (*Vector3* axis, *float* phi)
Basis	Basis (*Vector3* x_axis, *Vector3* y_axis, *Vector3* z_axis)
float	*determinant* ( )
Vector3	*get_euler* ( )
int	*get_orthogonal_index* ( )
Vector3	*get_scale* ( )
Basis	*inverse* ( )
Basis	*orthonormalized* ( )
Basis	*rotated* (*Vector3* axis, *float* phi)
Basis	*scaled* (*Vector3* scale)
Basis	*slerp* (*Basis* b, *float* t)
float	*tdotx* (*Vector3* with)
float	*tdoty* (*Vector3* with)
float	*tdotz* (*Vector3* with)
Basis	*transposed* ( )
Vector3	*xform* (*Vector3* v)
Vector3	*xform_inv* (*Vector3* v)

29.65.3 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.

29.65.4 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 (due to scaling). 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).

29.65.5 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). The axis must be a normalized vector.

- *Basis* **scaled (** *Vector3* scale **)**

Introduce an additional scaling specified by the given 3D scaling factor.

- *Basis* **slerp (** *Basis* b, *float* t **)**

Assuming that the matrix is a proper rotation matrix, slerp performs a spherical-linear interpolation with another rotation matrix.

- *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.

29.66 BitMap

Inherits: *Resource* **<** *Reference* **<** *Object*

Category: Core

29.66.1 Brief Description

Boolean matrix.

29.66.2 Member Functions

void	*create* **(** *Vector2* size **)**
void	*create_from_image_alpha* **(** *Image* image, *float* threshold=0.1 **)**
bool	*get_bit* **(** *Vector2* position **)** const
Vector2	*get_size* **(** **)** const
int	*get_true_bit_count* **(** **)** const
void	*set_bit* **(** *Vector2* position, *bool* bit **)**
void	*set_bit_rect* **(** *Rect2* p_rect, *bool* bit **)**

29.66.3 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.

29.66.4 Member Function Description

- void **create (***Vector2* size **)**

Creates a bitmap with the specified size, filled with false.

- void **create_from_image_alpha (***Image* image, *float* threshold=0.1 **)**

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 equal to `threshold` or less, 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.

29.67 BitmapFont

Inherits: *Font* < *Resource* < *Reference* < *Object*

Category: Core

29.67.1 Brief Description

Renders text using `*.fnt` fonts.

29.67.2 Member Functions

void	*add_char* (*int* character, *int* texture, *Rect2* rect, *Vector2* align=Vector2(0, 0), *float* advance=-1)
void	*add_kerning_pair* (*int* char_a, *int* char_b, *int* kerning)
void	*add_texture* (*Texture* texture)
void	*clear* ( )
Error	*create_from_fnt* (*String* path)
Vector2	*get_char_size* (*int* char, *int* next=0) const
int	*get_kerning_pair* (*int* char_a, *int* char_b) const
Texture	*get_texture* (*int* idx) const
int	*get_texture_count* ( ) const

29.67.3 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.

29.67.4 Description

Renders text using `*.fnt` fonts containing texture atlases. Supports distance fields. For using vector font files like TTF directly, see *DynamicFont*.

29.67.5 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.

- *Error* **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.

29.68 Bone2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.68.1 Brief Description

29.68.2 Member Functions

void	*apply_rest* ( )
int	*get_index_in_skeleton* ( ) const
Transform2D	*get_skeleton_rest* ( ) const

29.68.3 Member Variables

- *float* **default_length**
- *Transform2D* **rest**

29.68.4 Member Function Description

- void **apply_rest** ( )
- *int* **get_index_in_skeleton** ( ) const
- *Transform2D* **get_skeleton_rest** ( ) const

29.69 BoneAttachment

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.69.1 Brief Description

A node that will attach to a bone.

29.69.2 Member Variables

- *String* **bone_name** - The name of the attached bone.

29.69.3 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.

29.70 bool

Category: Built-In Types

29.70.1 Brief Description

Boolean built-in type

29.70.2 Member Functions

bool	bool (*int* from)
bool	bool (*float* from)
bool	bool (*String* from)

29.70.3 Description

Boolean built-in type.

29.70.4 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

29.71 BoxContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *VBoxContainer*, *HBoxContainer*, *ColorPicker*

Category: Core

29.71.1 Brief Description

Base class for box containers.

29.71.2 Member Functions

void	*add_spacer* (*bool* begin)

29.71.3 Member Variables

- *AlignMode* **alignment** - The alignment of the container's children (must be one of ALIGN_BEGIN, ALIGN_CENTER, or ALIGN_END).

29.71.4 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.

29.71.5 Description

Arranges child controls vertically or horizontally, and rearranges the controls automatically when their minimum size changes.

29.71.6 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.

29.72 BoxShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.72.1 Brief Description

Box shape resource.

29.72.2 Member Variables

- *Vector3* **extents** - The shape's half extents.

29.72.3 Description

3D box shape that can be a child of a *PhysicsBody* or *Area*.

29.73 BulletPhysicsDirectBodyState

Inherits: *PhysicsDirectBodyState* < *Object*

Category: Core

29.73.1 Brief Description

29.74 BulletPhysicsServer

Inherits: *PhysicsServer* < *Object*

Category: Core

29.74.1 Brief Description

29.75 Button

Inherits: *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *OptionButton*, *ColorPickerButton*, *CheckButton*, *MenuButton*, *ToolButton*, *CheckBox*

Category: Core

29.75.1 Brief Description

Standard themed Button.

29.75.2 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.

29.75.3 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.

29.75.4 Description

Button is the standard themed button. It can contain text and an icon, and will display them according to the current *Theme*.

29.76 ButtonGroup

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.76.1 Brief Description

Group of Buttons.

29.76.2 Member Functions

BaseButton	*get_pressed_button* ( )

29.76.3 Description

Group of *Button*. All direct and indirect children buttons become radios. Only one allows being pressed.

BaseButton.toggle_mode should be `true`.

29.76.4 Member Function Description

- *BaseButton* **get_pressed_button** ( )

Return the pressed button.

29.77 Camera

Inherits: *Spatial* < *Node* < *Object*

Inherited By: *InterpolatedCamera*, *ARVRCamera*

Category: Core

29.77.1 Brief Description

Camera node, displays from a point of view.

29.77.2 Member Functions

void	*clear_current* (*bool* enable_next=true)
Transform	*get_camera_transform* ( ) const
bool	*is_position_behind* (*Vector3* world_point) const
void	*make_current* ( )
Vector3	*project_local_ray_normal* (*Vector2* screen_point) const
Vector3	*project_position* (*Vector2* screen_point) const
Vector3	*project_ray_normal* (*Vector2* screen_point) const
Vector3	*project_ray_origin* (*Vector2* screen_point) const
void	*set_orthogonal* (*float* size, *float* z_near, *float* z_far)
void	*set_perspective* (*float* fov, *float* z_near, *float* z_far)
Vector2	*unproject_position* (*Vector3* world_point) const

29.77.3 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** - 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. Can be either `KEEP_WIDTH` or `KEEP_HEIGHT`.
- *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 vertical (Y) offset of the Camera viewport.

29.77.4 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.

29.77.5 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.

29.77.6 Member Function Description

- void **clear_current** (*bool* enable_next=true)

If this is the current Camera, remove it from being current. If `enable_next` is true, request to make the next Camera current, if any.

- *Transform* **get_camera_transform** ( ) const

Gets 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. Note that a position which returns `false` may still be outside the Camera's field of view.

- void **make_current** ( )

Makes 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 the 3D point in worldspace that maps to the given 2D coordinate in the *Viewport* rectangle.

- *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)

Sets 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)

Sets 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 the 2D coordinate in the *Viewport* rectangle that maps to the given 3D point in worldspace.

29.78 Camera2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.78.1 Brief Description

Camera node for 2D scenes.

29.78.2 Member Functions

void	*align* ( )
void	*clear_current* ( )
void	*force_update_scroll* ( )
Vector2	*get_camera_position* ( ) const
Vector2	*get_camera_screen_center* ( ) const
void	*make_current* ( )
void	*reset_smoothing* ( )

29.78.3 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.

29.78.4 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.

29.78.5 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*.

29.78.6 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.

29.79 CanvasItem

Inherits: *Node* < *Object*

Inherited By: *Node2D*, *Control*

Category: Core

29.79.1 Brief Description

Base class of anything 2D.

29.79.2 Member Functions

void	*_draw* ( ) virtual
float	*draw_char* (*Font* font, *Vector2* position, *String* char, *String* next, *Color* modulate=Color(1, 1, 1, 1))
void	*draw_circle* (*Vector2* position, *float* radius, *Color* color)
void	*draw_colored_polygon* (*PoolVector2Array* points, *Color* color, *PoolVector2Array* uvs=PoolVector2Array( ), *Texture* t
void	*draw_line* (*Vector2* from, *Vector2* to, *Color* color, *float* width=1.0, *bool* antialiased=false)
void	*draw_mesh* (*Mesh* mesh, *Texture* texture, *Texture* normal_map=null)
void	*draw_multiline* (*PoolVector2Array* points, *Color* color, *float* width=1.0, *bool* antialiased=false)
void	*draw_multiline_colors* (*PoolVector2Array* points, *PoolColorArray* colors, *float* width=1.0, *bool* antialiased=false)
void	*draw_multimesh* (*Mesh* mesh, *Texture* texture, *Texture* normal_map=null)
void	*draw_polygon* (*PoolVector2Array* points, *PoolColorArray* colors, *PoolVector2Array* uvs=PoolVector2Array( ), *Textu*
void	*draw_polyline* (*PoolVector2Array* points, *Color* color, *float* width=1.0, *bool* antialiased=false)
void	*draw_polyline_colors* (*PoolVector2Array* points, *PoolColorArray* colors, *float* width=1.0, *bool* antialiased=false)
void	*draw_primitive* (*PoolVector2Array* points, *PoolColorArray* colors, *PoolVector2Array* uvs, *Texture* texture=null, *float*
void	*draw_rect* (*Rect2* rect, *Color* color, *bool* filled=true)
void	*draw_set_transform* (*Vector2* position, *float* rotation, *Vector2* scale)
void	*draw_set_transform_matrix* (*Transform2D* xform)
void	*draw_string* (*Font* font, *Vector2* position, *String* text, *Color* modulate=Color(1, 1, 1, 1), *int* clip_w=-1)
void	*draw_style_box* (*StyleBox* style_box, *Rect2* rect)
void	*draw_texture* (*Texture* texture, *Vector2* position, *Color* modulate=Color(1, 1, 1, 1), *Texture* normal_map=null)
void	*draw_texture_rect* (*Texture* texture, *Rect2* rect, *bool* tile, *Color* modulate=Color(1, 1, 1, 1), *bool* transpose=false, *Te*

Table 6 – continued from previous page

void	*draw_texture_rect_region* (*Texture* texture, *Rect2* rect, *Rect2* src_rect, *Color* modulate=Color(1, 1, 1, 1), *bool* transp
RID	*get_canvas* ( ) const
RID	*get_canvas_item* ( ) const
Transform2D	*get_canvas_transform* ( ) const
Vector2	*get_global_mouse_position* ( ) const
Transform2D	*get_global_transform* ( ) const
Transform2D	*get_global_transform_with_canvas* ( ) const
Vector2	*get_local_mouse_position* ( ) const
Transform2D	*get_transform* ( ) const
Rect2	*get_viewport_rect* ( ) const
Transform2D	*get_viewport_transform* ( ) const
World2D	*get_world_2d* ( ) const
void	hide ( )
bool	*is_local_transform_notification_enabled* ( ) const
bool	*is_set_as_toplevel* ( ) const
bool	*is_transform_notification_enabled* ( ) const
bool	*is_visible_in_tree* ( ) const
Vector2	*make_canvas_position_local* (*Vector2* screen_point) const
InputEvent	*make_input_local* (*InputEvent* event) const
void	*set_as_toplevel* (*bool* enable)
void	*set_notify_local_transform* (*bool* enable)
void	*set_notify_transform* (*bool* enable)
void	*show* ( )
void	*update* ( )

29.79.3 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.

29.79.4 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`.

29.79.5 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.

29.79.6 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.
- **BLEND_MODE_DISABLED = 5** — Disable blending mode. Colors including alpha are written as is. Only applicable for render targets with a transparent background. No lighting will be applied.

29.79.7 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 draw 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.

29.79.8 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_mesh** (*Mesh* mesh, *Texture* texture, *Texture* normal_map=null)
- 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_multimesh** (*Mesh* mesh, *Texture* texture, *Texture* normal_map=null)
- 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.

29.80 CanvasItemMaterial

Inherits: *Material* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.80.1 Brief Description

A material for *CanvasItem*s.

29.80.2 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.

29.80.3 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.

29.80.4 Description

`CanvasItemMaterial`s 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*.

29.81 CanvasLayer

Inherits: *Node* < *Object*

Inherited By: *ParallaxBackground*

Category: Core

29.81.1 Brief Description

Canvas drawing layer.

29.81.2 Member Functions

RID	*get_canvas* ( ) const

29.81.3 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.

29.81.4 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).

29.81.5 Member Function Description

- *RID* **get_canvas** ( ) const

Returns the RID of the canvas used by this layer.

29.82 CanvasModulate

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.82.1 Brief Description

Tint the entire canvas.

29.82.2 Member Variables

- *Color* **color** - The tint color to apply.

29.82.3 Description

`CanvasModulate` tints the canvas elements using its assigned `color`.

29.83 CapsuleMesh

Inherits: *PrimitiveMesh* < *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.83.1 Brief Description

Class representing a capsule-shaped *PrimitiveMesh*.

29.83.2 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.

29.83.3 Description

Class representing a capsule-shaped *PrimitiveMesh*.

29.84 CapsuleShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.84.1 Brief Description

Capsule shape for collisions.

29.84.2 Member Variables

- *float* **height** - The capsule's height.
- *float* **radius** - The capsule's radius.

29.84.3 Description

Capsule shape for collisions.

29.85 CapsuleShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.85.1 Brief Description

Capsule shape for 2D collisions.

29.85.2 Member Variables

- *float* **height** - The capsule's height.
- *float* **radius** - The capsule's radius.

29.85.3 Description

Capsule shape for 2D collisions.

29.86 CenterContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.86.1 Brief Description

Keeps children controls centered.

29.86.2 Member Variables

- *bool* **use_top_left** - If `true` centers children relative to the `CenterContainer`'s top left corner. Default value: `false`.

29.86.3 Description

CenterContainer Keeps children controls centered. This container keeps all children to their minimum size, in the center.

29.87 CheckBox

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.87.1 Brief Description

Binary choice user interface widget.

29.87.2 Description

A checkbox allows the user to make a binary choice (choosing only one of two possible options), for example Answer 'yes' or 'no'.

29.88 CheckButton

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.88.1 Brief Description

Checkable button.

29.88.2 Description

CheckButton is a toggle button displayed as a check field.

29.89 CircleShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.89.1 Brief Description

Circular shape for 2D collisions.

29.89.2 Member Variables

- *float* **radius** - The circle's radius.

29.89.3 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.

29.90 ClassDB

Inherits: *Object*

Category: Core

29.90.1 Brief Description

Class information repository.

29.90.2 Member Functions

bool	*can_instance* (*String* class) const
bool	*class_exists* (*String* class) const
String	*class_get_category* (*String* class) const
int	*class_get_integer_constant* (*String* class, *String* name) const
PoolStringArray	*class_get_integer_constant_list* (*String* class, *bool* no_inheritance=false) const
Array	*class_get_method_list* (*String* class, *bool* no_inheritance=false) const
Variant	*class_get_property* (*Object* object, *String* property) const
Array	*class_get_property_list* (*String* class, *bool* no_inheritance=false) const
Dictionary	*class_get_signal* (*String* class, *String* signal) const
Array	*class_get_signal_list* (*String* class, *bool* no_inheritance=false) const
bool	*class_has_integer_constant* (*String* class, *String* name) const
bool	*class_has_method* (*String* class, *String* method, *bool* no_inheritance=false) const
bool	*class_has_signal* (*String* class, *String* signal) const
Error	*class_set_property* (*Object* object, *String* property, *Variant* value) const
PoolStringArray	*get_class_list* ( ) const
PoolStringArray	*get_inheriters_from_class* (*String* class) const
String	*get_parent_class* (*String* class) const
Variant	*instance* (*String* class) const
bool	*is_class_enabled* (*String* class) const
bool	*is_parent_class* (*String* class, *String* inherits) const

29.90.3 Description

Provides access to metadata stored for every available class.

29.90.4 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.

- *Error* **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_inheriters_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.

29.91 CollisionObject

Inherits: *Spatial* < *Node* < *Object*

Inherited By: *PhysicsBody*, *Area*

Category: Core

29.91.1 Brief Description

Base node for collision objects.

29.91.2 Member Functions

void	*_input_event* (*Object* camera, *InputEvent* event, *Vector3* click_position, *Vector3* click_normal, *int* shape_idx) virtual
int	*create_shape_owner* (*Object* owner)
RID	*get_rid* ( ) const
Array	*get_shape_owners* ( )
bool	*is_shape_owner_disabled* (*int* owner_id) const
void	*remove_shape_owner* (*int* owner_id)
int	*shape_find_owner* (*int* shape_index) const
void	*shape_owner_add_shape* (*int* owner_id, *Shape* shape)
void	*shape_owner_clear_shapes* (*int* owner_id)
Object	*shape_owner_get_owner* (*int* owner_id) const
Shape	*shape_owner_get_shape* (*int* owner_id, *int* shape_id) const
int	*shape_owner_get_shape_count* (*int* owner_id) const
int	*shape_owner_get_shape_index* (*int* owner_id, *int* shape_id) const
Transform	*shape_owner_get_transform* (*int* owner_id) const
void	*shape_owner_remove_shape* (*int* owner_id, *int* shape_id)
void	*shape_owner_set_disabled* (*int* owner_id, *bool* disabled)
void	*shape_owner_set_transform* (*int* owner_id, *Transform* transform)

29.91.3 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.

29.91.4 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 *RayCast*s. Default value: `true`.

29.91.5 Description

CollisionObject is the base class for physics objects. It can hold any number of collision *Shape*s. 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.

29.91.6 Member Function Description

- void **_input_event** (*Object* camera, *InputEvent* event, *Vector3* click_position, *Vector3* click_normal, *int* shape_idx) virtual

Accepts unhandled *InputEvent*s. `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.

29.92 CollisionObject2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Inherited By: *Area2D*, *PhysicsBody2D*

Category: Core

29.92.1 Brief Description

Base node for 2D collision objects.

29.92.2 Member Functions

void	*_input_event* (*Object* viewport, *InputEvent* event, *int* shape_idx) virtual
int	*create_shape_owner* (*Object* owner)
RID	*get_rid* ( ) const
Array	*get_shape_owners* ( )
bool	*is_shape_owner_disabled* (*int* owner_id) const
bool	*is_shape_owner_one_way_collision_enabled* (*int* owner_id) const
void	*remove_shape_owner* (*int* owner_id)
int	*shape_find_owner* (*int* shape_index) const
void	*shape_owner_add_shape* (*int* owner_id, *Shape2D* shape)
void	*shape_owner_clear_shapes* (*int* owner_id)
Object	*shape_owner_get_owner* (*int* owner_id) const
Shape2D	*shape_owner_get_shape* (*int* owner_id, *int* shape_id) const
int	*shape_owner_get_shape_count* (*int* owner_id) const
int	*shape_owner_get_shape_index* (*int* owner_id, *int* shape_id) const
Transform2D	*shape_owner_get_transform* (*int* owner_id) const
void	*shape_owner_remove_shape* (*int* owner_id, *int* shape_id)
void	*shape_owner_set_disabled* (*int* owner_id, *bool* disabled)
void	*shape_owner_set_one_way_collision* (*int* owner_id, *bool* enable)
void	*shape_owner_set_transform* (*int* owner_id, *Transform2D* transform)

29.92.3 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.

29.92.4 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.

29.92.5 Description

CollisionObject2D is the base class for 2D physics objects. It can hold any number of 2D collision *Shape2D*s. 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.

29.92.6 Member Function Description

- void **_input_event** (*Object* viewport, *InputEvent* event, *int* shape_idx) virtual

Accepts unhandled *InputEvent*s. `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 `CollisionObject2D`s.

- 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 `CollisionObject2D`s.

- void **shape_owner_set_transform** (*int* owner_id, *Transform2D* transform)

Sets the *Transform2D* of the given shape owner.

29.93 CollisionPolygon

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.93.1 Brief Description

Editor-only class for defining a collision polygon in 3D space.

29.93.2 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.

29.93.3 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.

29.94 CollisionPolygon2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.94.1 Brief Description

Defines a 2D collision polygon.

29.94.2 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.

29.94.3 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.

29.94.4 Description

Provides a 2D collision polygon to a *CollisionObject2D* parent. Polygon can be drawn in the editor or specified by a list of vertices.

29.95 CollisionShape

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.95.1 Brief Description

Node that represents collision shape data in 3D space.

29.95.2 Member Functions

void	*make_convex_from_brothers* ( )
void	*resource_changed* (*Resource* resource)

29.95.3 Member Variables

- *bool* **disabled** - A disabled collision shape has no effect in the world.
- *Shape* **shape** - The actual shape owned by this collision shape.

29.95.4 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 create a solid object. **IMPORTANT**: this is an Editor-only helper to create shapes, use get_shape to get the actual shape.

29.95.5 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.

29.96 CollisionShape2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.96.1 Brief Description

Node that represents collision shape data in 2D space.

29.96.2 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.

29.96.3 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 create a solid object. **IMPORTANT**: this is an Editor-only helper to create shapes, use get_shape to get the actual shape.

29.97 Color

Category: Built-In Types

29.97.1 Brief Description

Color in RGBA format with some support for ARGB format.

29.97.2 Member Functions

Color	Color (*float* r, *float* g, *float* b, *float* a)
Color	Color (*float* r, *float* g, *float* b)
Color	Color (*int* from)
Color	Color (*String* from)
Color	*blend* (*Color* over)
Color	*contrasted* ( )
Color	*darkened* (*float* amount)
Color	*from_hsv* (*float* h, *float* s, *float* v, *float* a=1)
float	*gray* ( )
Color	*inverted* ( )
Color	*lightened* (*float* amount)
Color	*linear_interpolate* (*Color* b, *float* t)
int	*to_argb32* ( )
String	*to_html* (*bool* with_alpha=True)
int	*to_rgba32* ( )

29.97.3 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)

29.97.4 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*.

29.97.5 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("#ffb2d90a") # 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
```

- *Color* **from_hsv (** *float* h, *float* s, *float* v, *float* a=1 **)**

Constructs a color from an HSV profile. `h`, `s`, and `v` are values between 0 and 1.

```
var c = Color.from_hsv(0.58, 0.5, 0.79, 0.8) # equivalent to HSV(210, 50, 79, 0.8) or␣
↪Color8(100, 151, 201, 0.8)
```

- *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.

29.98 ColorPicker

Inherits: *BoxContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.98.1 Brief Description

Color picker control.

29.98.2 Member Functions

void	*add_preset* (*Color* color)

29.98.3 Signals

- **color_changed** (*Color* color)

Emitted when the color is changed.

29.98.4 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).

29.98.5 Description

This is a simple color picker *Control*. It's useful for selecting a color from an RGB/RGBA colorspace.

29.98.6 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.

29.99 ColorPickerButton

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.99.1 Brief Description

Button that pops out a *ColorPicker*.

29.99.2 Member Functions

ColorPicker	*get_picker* ( )
PopupPanel	*get_popup* ( ) const

29.99.3 Signals

- **color_changed** (*Color* color)

Emitted when the color changes.

- **popup_closed** ( )

29.99.4 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`.

29.99.5 Description

Encapsulates a *ColorPicker* making it accesible by pressing a button, pressing the button will toggle the *ColorPicker* visibility

29.99.6 Member Function Description

- *ColorPicker* **get_picker** ( )

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.

29.100 ColorRect

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.100.1 Brief Description

Colored rect for canvas.

29.100.2 Member Variables

- *Color* **color** - The color to fill the `ColorRect`.

```
$ColorRect.color = Color(1, 0, 0, 1) # Set ColorRect node's color to red
```

29.100.3 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.

29.101 ConcavePolygonShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.101.1 Brief Description

Concave polygon shape.

29.101.2 Member Functions

PoolVector3Array	*get_faces* ( ) const
void	*set_faces* (*PoolVector3Array* faces)

29.101.3 Description

Concave polygon shape resource, which can be set into a *PhysicsBody* or area. This shape is created by feeding a list of triangles.

29.101.4 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).

29.102 ConcavePolygonShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.102.1 Brief Description

Concave polygon 2D shape resource for physics.

29.102.2 Member Variables

- *PoolVector2Array* **segments** - The array of points that make up the `ConcavePolygonShape2D`'s line segments.

29.102.3 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.

29.103 ConeTwistJoint

Inherits: *Joint* < *Spatial* < *Node* < *Object*

Category: Core

29.103.1 Brief Description

A twist joint between two 3D bodies.

29.103.2 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: `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.

29.103.3 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: `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.

29.103.4 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.

29.104 ConfigFile

Inherits: *Reference* < *Object*

Category: Core

29.104.1 Brief Description

Helper class to handle INI-style files.

29.104.2 Member Functions

void	*erase_section* (*String* section)
PoolStringArray	*get_section_keys* (*String* section) const
PoolStringArray	*get_sections* ( ) const
Variant	*get_value* (*String* section, *String* key, *Variant* default=null) const
bool	*has_section* (*String* section) const
bool	*has_section_key* (*String* section, *String* key) const
Error	*load* (*String* path)
Error	*save* (*String* path)
void	*set_value* (*String* section, *String* key, *Variant* value)

29.104.3 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")
```

29.104.4 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.

- *Error* **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`.

- *Error* **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.

29.105 ConfirmationDialog

Inherits: *AcceptDialog* < *WindowDialog* < *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *EditorFileDialog*, *FileDialog*

Category: Core

29.105.1 Brief Description

Dialog for confirmation of actions.

29.105.2 Member Functions

Button	*get_cancel* ( )

29.105.3 Description

Dialog for confirmation of actions. This dialog inherits from *AcceptDialog*, but has by default an OK and Cancel button (in host OS order).

29.105.4 Member Function Description

- *Button* **get_cancel** ( )

Return the cancel button.

29.106 Container

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *PanelContainer*, *GridContainer*, *ScrollContainer*, *ViewportContainer*, *MarginContainer*, *CenterContainer*, *GraphNode*, *SplitContainer*, *BoxContainer*, *EditorProperty*

Category: Core

29.106.1 Brief Description

Base node for containers.

29.106.2 Member Functions

void	*fit_child_in_rect* (*Control* child, *Rect2* rect)
void	*queue_sort* ( )

29.106.3 Signals

- **sort_children** ( )

Emitted when sorting the children is needed.

29.106.4 Numeric Constants

- **NOTIFICATION_SORT_CHILDREN = 50** — Notification for when sorting the children, it must be obeyed immediately.

29.106.5 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.

29.106.6 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.

29.107 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

29.107.1 Brief Description

All User Interface nodes inherit from Control. Features anchors and margins to adapt its position and size to its parent.

29.107.2 Member Functions

Vector2	*_get_minimum_size* ( ) virtual
void	*_gui_input* (*InputEvent* event) virtual
void	*accept_event* ( )
void	*add_color_override* (*String* name, *Color* color)
void	*add_constant_override* (*String* name, *int* constant)
void	*add_font_override* (*String* name, *Font* font)
void	*add_icon_override* (*String* name, *Texture* texture)
void	*add_shader_override* (*String* name, *Shader* shader)
void	*add_stylebox_override* (*String* name, *StyleBox* stylebox)
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
CursorShape	*get_cursor_shape* (*Vector2* position=Vector2(0, 0)) const
Object	*get_drag_data* (*Vector2* position) virtual
Vector2	*get_end* ( ) const
Control	*get_focus_owner* ( ) const
Font	*get_font* (*String* name, *String* type="") const
Rect2	*get_global_rect* ( ) const
Texture	*get_icon* (*String* name, *String* type="") const
Vector2	*get_minimum_size* ( ) const
Vector2	*get_parent_area_size* ( ) const
Control	*get_parent_control* ( ) const
Rect2	*get_rect* ( ) const
float	*get_rotation* ( ) const
StyleBox	*get_stylebox* (*String* name, *String* type="") const
String	*get_tooltip* (*Vector2* at_position=Vector2(0, 0)) const
void	*grab_click_focus* ( )
void	*grab_focus* ( )
bool	*has_color* (*String* name, *String* type="") const
bool	*has_color_override* (*String* name) const
bool	*has_constant* (*String* name, *String* type="") const

Continued on next page

Table 7 – continued from previous page

bool	*has_constant_override* (*String* name) const
bool	*has_focus* ( ) const
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* ( )
void	*set_anchor* (*Margin* margin, *float* anchor, *bool* keep_margin=false, *bool* push_opposite_anchor=true)
void	*set_anchor_and_margin* (*Margin* margin, *float* anchor, *float* offset, *bool* push_opposite_anchor=false)
void	*set_anchors_and_margins_preset* (*LayoutPreset* preset, *LayoutPresetMode* resize_mode=0, *int* margin=0)
void	*set_anchors_preset* (*LayoutPreset* preset, *bool* keep_margin=false)
void	*set_begin* (*Vector2* position)
void	*set_drag_forwarding* (*Control* target)
void	*set_drag_preview* (*Control* control)
void	*set_end* (*Vector2* position)
void	*set_margins_preset* (*LayoutPreset* preset, *LayoutPresetMode* resize_mode=0, *int* margin=0)
void	*set_rotation* (*float* radians)
void	*show_modal* (*bool* exclusive=false)
void	*warp_mouse* (*Vector2* to_position)

29.107.3 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*.

29.107.4 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`.
- *float* **anchor_top** - Anchors the top edge of the node to the origin, the center or the end of its parent container. It changes how the top 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.

29.107.5 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*.
- **NOTIFICATION_SCROLL_BEGIN = 47**
- **NOTIFICATION_SCROLL_END = 48**

29.107.6 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**
- **GROW_DIRECTION_BOTH = 2**

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*.

29.107.7 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 *grab_focus*. `Control` nodes lose focus when another node grabs it, or if you hide the node in focus.

Set *mouse_filter* to MOUSE_FILTER_IGNORE 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.

29.107.8 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

Godot calls this method to test if `data` from a control's *get_drag_data* can be dropped at `position`. `position` is local to this control.

This method should only be used to test the data. Process the data in *drop_data*.

```
extends Control

func can_drop_data(position, data):
    # check position if it is relevant to you
    # otherwise just check data
    return typeof(data) == TYPE_DICTIONARY and data.has('expected')
```

- void **drop_data** (*Vector2* position, *Variant* data) virtual

Godot calls this method to pass you the `data` from a control's *get_drag_data* result. Godot first calls *can_drop_data* to test if `data` is allowed to drop at `position` where `position` is local to this control.

```
extends ColorRect

func can_drop_data(position, data):
    return typeof(data) == TYPE_DICTIONARY and data.has('color')

func drop_data(position, data):
    color = data['color']
```

- void **force_drag** (*Variant* data, *Control* preview)

Forces drag and bypasses *get_drag_data* and *set_drag_preview* by passing `data` and `preview`. Drag will start even if the mouse is neither over nor pressed on this control.

The methods *can_drop_data* and *drop_data* must be implemented on controls that want to receive drop data.

- *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
- *CursorShape* **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

Godot calls this method to get data that can be dragged and dropped onto controls that expect drop data. Return null if there is no data to drag. Controls that want to receive drop data should implement *can_drop_data* and *drop_data*. `position` is local to this control. Drag may be forced with *force_drag*.

A preview that will follow the mouse that should represent the data can be set with *set_drag_preview*. A good time to set the preview is in this method.

```
extends Control

func get_drag_data(position):
    var mydata = make_data()
```

(continues on next page)

(continued from previous page)

```
    set_drag_preview(make_preview(mydata))
    return mydata
```

- *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** (*Margin* margin, *float* anchor, *bool* keep_margin=false, *bool* push_opposite_anchor=true)
- void **set_anchor_and_margin** (*Margin* margin, *float* anchor, *float* offset, *bool* push_opposite_anchor=false)
- void **set_anchors_and_margins_preset** (*LayoutPreset* preset, *LayoutPresetMode* resize_mode=0, *int* margin=0)
- void **set_anchors_preset** (*LayoutPreset* 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)

Forwards the handling of this control's drag and drop to `target` control.

Forwarding can be implemented in the target control similar to the methods *get_drag_data*, *can_drop_data*, and *drop_data* but with two differences:

1. The function name must be suffixed with **_fw**
2. The function must take an extra argument that is the control doing the forwarding

```
# ThisControl.gd
extends Control
func _ready():
    set_drag_forwarding(target_control)

# TargetControl.gd
extends Control
func can_drop_data_fw(position, data, from_control):
    return true

func drop_data_fw(position, data, from_control):
    my_handle_data(data)

func get_drag_data_fw(position, from_control):
    set_drag_preview(my_preview)
    return my_data()
```

- void **set_drag_preview** (*Control* control)

Shows the given control at the mouse pointer. A good time to call this method is in *get_drag_data*.

- 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** (*LayoutPreset* preset, *LayoutPresetMode* 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)

29.108 ConvexPolygonShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.108.1 Brief Description

Convex polygon shape for 3D physics.

29.108.2 Member Variables

- *PoolVector3Array* **points** - The list of 3D points forming the convex polygon shape.

29.108.3 Description

Convex polygon shape resource, which can be added to a *PhysicsBody* or area.

29.109 ConvexPolygonShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.109.1 Brief Description

Convex Polygon Shape for 2D physics.

29.109.2 Member Functions

void	*set_point_cloud* (*PoolVector2Array* point_cloud)

29.109.3 Member Variables

- *PoolVector2Array* **points** - The polygon's list of vertices. Can be in either clockwise or counterclockwise order.

29.109.4 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.

29.109.5 Member Function Description

- void **set_point_cloud** **(** *PoolVector2Array* point_cloud **)**

Currently, this method does nothing.

29.110 CSGBox

Inherits: *CSGPrimitive* **<** *CSGShape* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.110.1 Brief Description

29.110.2 Member Variables

- *float* **depth**
- *float* **height**
- *Material* **material**
- *float* **width**

29.111 CSGCombiner

Inherits: *CSGShape* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.111.1 Brief Description

29.112 CSGCylinder

Inherits: *CSGPrimitive* < *CSGShape* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.112.1 Brief Description

29.112.2 Member Variables

- *bool* **cone**
- *float* **height**
- *Material* **material**
- *float* **radius**
- *int* **sides**
- *bool* **smooth_faces**

29.113 CSGMesh

Inherits: *CSGPrimitive* < *CSGShape* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.113.1 Brief Description

29.113.2 Member Variables

- *Mesh* **mesh**

29.114 CSGPolygon

Inherits: *CSGPrimitive* < *CSGShape* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.114.1 Brief Description

29.114.2 Member Variables

- *float* **depth**
- *Material* **material**
- *Mode* **mode**
- *float* **path_interval**
- *NodePath* **path_node**
- *PathRotation* **path_rotation**
- *PoolVector2Array* **polygon**
- *bool* **smooth_faces**
- *float* **spin_degrees**
- *int* **spin_sides**

29.114.3 Enums

enum **Mode**

- **MODE_DEPTH = 0**
- **MODE_SPIN = 1**
- **MODE_PATH = 2**

enum **PathRotation**

- **PATH_ROTATION_POLYGON = 0**
- **PATH_ROTATION_PATH = 1**
- **PATH_ROTATION_PATH_FOLLOW = 2**

29.115 CSGPrimitive

Inherits: *CSGShape* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Inherited By: *CSGSphere*, *CSGPolygon*, *CSGCylinder*, *CSGBox*, *CSGTorus*, *CSGMesh*

Category: Core

29.115.1 Brief Description

29.115.2 Member Variables

- *bool* **invert_faces**

29.116 CSGShape

Inherits: *VisualInstance* < *Spatial* < *Node* < *Object*

Inherited By: *CSGCombiner*, *CSGPrimitive*

Category: Core

29.116.1 Brief Description

29.116.2 Member Functions

bool	*is_root_shape* ( ) const

29.116.3 Member Variables

- *Operation* **operation**
- *float* **snap**
- *bool* **use_collision**

29.116.4 Enums

enum **Operation**

- **OPERATION_UNION = 0**
- **OPERATION_INTERSECTION = 1**
- **OPERATION_SUBTRACTION = 2**

29.116.5 Member Function Description

- *bool* **is_root_shape** ( ) const

29.117 CSGSphere

Inherits: *CSGPrimitive* < *CSGShape* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.117.1 Brief Description

29.117.2 Member Variables

- *Material* **material**
- *int* **radial_segments**
- *float* **radius**
- *int* **rings**
- *bool* **smooth_faces**

29.118 CSGTorus

Inherits: *CSGPrimitive* < *CSGShape* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.118.1 Brief Description

29.118.2 Member Variables

- *float* **inner_radius**
- *Material* **material**
- *float* **outer_radius**
- *int* **ring_sides**
- *int* **sides**
- *bool* **smooth_faces**

29.119 CSharpScript

Inherits: *Script* < *Resource* < *Reference* < *Object*

Category: Core

29.119.1 Brief Description

29.119.2 Member Functions

Object	*new* ( ) vararg

29.119.3 Member Function Description

- *Object* **new** ( ) vararg

29.120 CubeMap

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.120.1 Brief Description

A CubeMap is a 6 sided 3D texture.

29.120.2 Member Functions

int	*get_height* ( ) const
Image	*get_side* (*Side* side) const
int	*get_width* ( ) const
void	*set_side* (*Side* side, *Image* image)

29.120.3 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.

29.120.4 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`.

29.120.5 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.

29.120.6 Member Function Description

- *int* **get_height ( )** const

Returns the `CubeMap`'s height.

- *Image* **get_side (** *Side* 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 (** *Side* side, *Image* image **)**

Sets an *Image* for a side of the `CubeMap` using one of the `SIDE_*` constants or an integer 0-5.

29.121 CubeMesh

Inherits: *PrimitiveMesh* **<** *Mesh* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.121.1 Brief Description

Generate an axis-aligned cuboid *PrimitiveMesh*.

29.121.2 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.

29.121.3 Description

Generate an axis-aligned cuboid *PrimitiveMesh*.

29.122 Curve

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.122.1 Brief Description

A mathematic curve.

29.122.2 Member Functions

int	*add_point* (*Vector2* position, *float* left_tangent=0, *float* right_tangent=0, *TangentMode* left_mode=0, *TangentMode* right_mode=0)
void	*bake* ( )
void	*clean_dupes* ( )
void	*clear_points* ( )
TangentMode	*get_point_left_mode* (*int* index) const
float	*get_point_left_tangent* (*int* index) const
Vector2	*get_point_position* (*int* index) const
TangentMode	*get_point_right_mode* (*int* index) const
float	*get_point_right_tangent* (*int* index) const
float	*interpolate* (*float* offset) const
float	*interpolate_baked* (*float* offset)
void	*remove_point* (*int* index)
void	*set_point_left_mode* (*int* index, *TangentMode* mode)
void	*set_point_left_tangent* (*int* index, *float* tangent)
int	*set_point_offset* (*int* index, *float* offset)
void	*set_point_right_mode* (*int* index, *TangentMode* mode)
void	*set_point_right_tangent* (*int* index, *float* tangent)
void	*set_point_value* (*int* index, *float* y)

29.122.3 Signals

- **range_changed** ( )

Emitted when *max_value* or *min_value* is changed.

29.122.4 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`.

29.122.5 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.

29.122.6 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.

29.122.7 Member Function Description

- *int* **add_point** (*Vector2* position, *float* left_tangent=0, *float* right_tangent=0, *TangentMode* left_mode=0, *TangentMode* 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_dupes** ( )

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.

- *TangentMode* **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`.

- *TangentMode* **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, *TangentMode* 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, *TangentMode* 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`.

29.123 Curve2D

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.123.1 Brief Description

Describes a Bezier curve in 2D space.

29.123.2 Member Functions

void	*add_point* (*Vector2* position, *Vector2* in=Vector2(0, 0), *Vector2* out=Vector2(0, 0), *int* at_position=-1)
void	*clear_points* ( )
float	*get_baked_length* ( ) const
PoolVector2Array	*get_baked_points* ( ) const
float	*get_closest_offset* (*Vector2* to_point) const
Vector2	*get_closest_point* (*Vector2* to_point) const
int	*get_point_count* ( ) const
Vector2	*get_point_in* (*int* idx) const
Vector2	*get_point_out* (*int* idx) const
Vector2	*get_point_position* (*int* idx) const
Vector2	*interpolate* (*int* idx, *float* t) const
Vector2	*interpolate_baked* (*float* offset, *bool* cubic=false) const
Vector2	*interpolatef* (*float* fofs) const
void	*remove_point* (*int* idx)
void	*set_point_in* (*int* idx, *Vector2* position)
void	*set_point_out* (*int* idx, *Vector2* position)
void	*set_point_position* (*int* idx, *Vector2* position)
PoolVector2Array	*tessellate* (*int* max_stages=5, *float* tolerance_degrees=4) const

29.123.3 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.

29.123.4 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.

29.123.5 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*.

- *float* **get_closest_offset** (*Vector2* to_point) const

Returns the closest offset to `to_point`. This offset is meant to be used in *interpolate_baked*.

`to_point` must be in this curve's local space.

- *Vector2* **get_closest_point** (*Vector2* to_point) const

Returns the closest point (in curve's local space) to `to_point`.

`to_point` must be in this curve's local space.

- *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.

29.124 Curve3D

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.124.1 Brief Description

Describes a Bezier curve in 3D space.

29.124.2 Member Functions

void	*add_point* (*Vector3* position, *Vector3* in=Vector3(0, 0, 0), *Vector3* out=Vector3(0, 0, 0), *int* at_position=-1)
void	*clear_points* ( )
float	*get_baked_length* ( ) const
PoolVector3Array	*get_baked_points* ( ) const
PoolRealArray	*get_baked_tilts* ( ) const
PoolVector3Array	*get_baked_up_vectors* ( ) const
float	*get_closest_offset* (*Vector3* to_point) const
Vector3	*get_closest_point* (*Vector3* to_point) const
int	*get_point_count* ( ) const
Vector3	*get_point_in* (*int* idx) const
Vector3	*get_point_out* (*int* idx) const
Vector3	*get_point_position* (*int* idx) const
float	*get_point_tilt* (*int* idx) const
Vector3	*interpolate* (*int* idx, *float* t) const
Vector3	*interpolate_baked* (*float* offset, *bool* cubic=false) const
Vector3	*interpolate_baked_up_vector* (*float* offset, *bool* apply_tilt=false) const
Vector3	*interpolatef* (*float* fofs) const
void	*remove_point* (*int* idx)
void	*set_point_in* (*int* idx, *Vector3* position)
void	*set_point_out* (*int* idx, *Vector3* position)
void	*set_point_position* (*int* idx, *Vector3* position)
void	*set_point_tilt* (*int* idx, *float* tilt)
PoolVector3Array	*tessellate* (*int* max_stages=5, *float* tolerance_degrees=4) const

29.124.3 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.
- *bool* **up_vector_enabled** - If `true`, the curve will bake up vectors used for orientation. See *OrientedPathFollow*. Changing it forces the cache to be recomputed.

29.124.4 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.

29.124.5 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.

- *PoolVector3Array* **get_baked_up_vectors** ( ) const

Returns the cache of up vectors as a *PoolVector3Array*.

If *up_vector_enabled* is `false`, the cache will be empty.

- *float* **get_closest_offset** (*Vector3* to_point) const

Returns the closest offset to `to_point`. This offset is meant to be used in one of the interpolate_baked* methods.

`to_point` must be in this curve's local space.

- *Vector3* **get_closest_point** (*Vector3* to_point) const

Returns the closest point (in curve's local space) to `to_point`.

`to_point` must be in this curve's local space.

- *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 >= 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, 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* **interpolate_baked_up_vector** (*float* offset, *bool* apply_tilt=false) const

Returns an up vector 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 up vectors where the `offset` lies between, then interpolates the values. If `apply_tilt` is `true`, an interpolated tilt is applied to the interpolated up vector.

If the curve has no up vectors, the function sends an error to the console, and returns (0, 1, 0).

- *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* or *OrientedPathFollow*, this tilt is an offset over the natural tilt the *PathFollow* or *OrientedPathFollow* 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.

29.125 CurveTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.125.1 Brief Description

A texture that shows a curve.

29.125.2 Member Variables

- *Curve* **curve** - The `curve` rendered onto the texture.
- *int* **width** - The width of the texture.

29.125.3 Description

Renders a given *Curve* provided to it. Simplifies the task of drawing curves and/or saving them as image files.

29.126 CylinderMesh

Inherits: *PrimitiveMesh* < *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.126.1 Brief Description

Class representing a cylindrical *PrimitiveMesh*.

29.126.2 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.

29.126.3 Description

Class representing a cylindrical *PrimitiveMesh*.

29.127 DampedSpringJoint2D

Inherits: *Joint2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.127.1 Brief Description

Damped spring constraint for 2D physics.

29.127.2 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`

29.127.3 Description

Damped spring constraint for 2D physics. This resembles a spring joint that always wants to go back to a given length.

29.128 Dictionary

Category: Built-In Types

29.128.1 Brief Description

Dictionary type.

29.128.2 Member Functions

void	*clear* ( )
Dictionary	*duplicate* (*bool* deep=False)
bool	*empty* ( )
void	*erase* (var key)
bool	*has* (var key)
bool	*has_all* (*Array* keys)
int	*hash* ( )
Array	*keys* ( )
int	*size* ( )
Array	*values* ( )

29.128.3 Description

Dictionary type. Associative container which contains values referenced by unique keys. Dictionaries are always passed by reference.

29.128.4 Member Function Description

- void **clear** ( )

Clear the dictionary, removing all key/value pairs.

- *Dictionary* **duplicate** (*bool* deep=False)

Creates a copy of the dictionary, and returns it.

- *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`.

29.129 DirectionalLight

Inherits: *Light* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.129.1 Brief Description

Directional Light, such as the Sun or the Moon.

29.129.2 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**

29.129.3 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**

29.129.4 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 do determine light direction.

29.130 Directory

Inherits: *Reference* < *Object*

Category: Core

29.130.1 Brief Description

Type used to handle the filesystem.

29.130.2 Member Functions

Error	*change_dir* (*String* todir)
Error	*copy* (*String* from, *String* to)
bool	*current_is_dir* ( ) const
bool	*dir_exists* (*String* path)
bool	*file_exists* (*String* path)
String	*get_current_dir* ( )
int	*get_current_drive* ( )
String	*get_drive* (*int* idx)
int	*get_drive_count* ( )
String	*get_next* ( )
int	*get_space_left* ( )
Error	*list_dir_begin* (*bool* skip_navigational=false, *bool* skip_hidden=false)
void	*list_dir_end* ( )
Error	*make_dir* (*String* path)
Error	*make_dir_recursive* (*String* path)
Error	*open* (*String* path)
Error	*remove* (*String* path)
Error	*rename* (*String* from, *String* to)

29.130.3 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()
```

(continues on next page)

(continued from previous page)

```
        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.")
```

29.130.4 Member Function Description

- *Error* **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_*).

- *Error* **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 existed, 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_navigational` 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.

- *Error* **list_dir_begin (** *bool* skip_navigational=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_navigational`, 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).

- *Error* **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_*).

- *Error* **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_*).

- *Error* **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_*).

- *Error* **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).

- *Error* **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).

29.131 DynamicFont

Inherits: *Font* < *Resource* < *Reference* < *Object*

Category: Core

29.131.1 Brief Description

DynamicFont renders vector font files at runtime.

29.131.2 Member Functions

void	*add_fallback* (*DynamicFontData* data)
DynamicFontData	*get_fallback* (*int* idx) const
int	*get_fallback_count* ( ) const
void	*remove_fallback* (*int* idx)
void	*set_fallback* (*int* idx, *DynamicFontData* data)

29.131.3 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.
- *Color* **outline_color**
- *int* **outline_size**
- *int* **size** - The font size.
- *bool* **use_filter** - If `true` filtering is used.
- *bool* **use_mipmaps** - If `true` mipmapping is used.

29.131.4 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.

29.131.5 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 *BitmapFont*s for the ability to change font parameters like size and spacing during runtime. *DynamicFontData* is used for referencing the font file paths.

29.131.6 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`.

29.132 DynamicFontData

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.132.1 Brief Description

Used with *DynamicFont* to describe the location of a font file.

29.132.2 Member Variables

- *String* **font_path** - The path to the vector font file.
- *Hinting* **hinting** - The font hinting mode used by FreeType.

29.132.3 Enums

enum **Hinting**

- **HINTING_NONE = 0** — Disable font hinting (smoother but less crisp).
- **HINTING_LIGHT = 1** — Use the light font hinting mode.
- **HINTING_NORMAL = 2** — Use the default font hinting mode (crisper but less smooth).

29.132.4 Description

Used with *DynamicFont* to describe the location of a vector font file for dynamic rendering at runtime.

29.133 EditorExportPlugin

Inherits: *Reference* < *Object*

Category: Core

29.133.1 Brief Description

29.133.2 Member Functions

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* ( )

29.133.3 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** ( )

29.134 EditorFileDialog

Inherits: *ConfirmationDialog* < *AcceptDialog* < *WindowDialog* < *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.134.1 Brief Description

29.134.2 Member Functions

void	*add_filter* (*String* filter)
void	*clear_filters* ( )
VBoxContainer	*get_vbox* ( )
void	*invalidate* ( )

29.134.3 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.

29.134.4 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`.

29.134.5 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.

29.134.6 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.

29.135 EditorFileSystem

Inherits: *Node* < *Object*

Category: Core

29.135.1 Brief Description

Resource filesystem, as the editor sees it.

29.135.2 Member Functions

String	*get_file_type* (*String* path) const
EditorFileSystemDirectory	*get_filesystem* ( )
EditorFileSystemDirectory	*get_filesystem_path* (*String* path)
float	*get_scanning_progress* ( ) const
bool	*is_scanning* ( ) const
void	*scan* ( )
void	*scan_sources* ( )
void	*update_file* (*String* path)

29.135.3 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.

29.135.4 Description

This object holds information of all resources in the filesystem, their types, etc.

29.135.5 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 of 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.

29.136 EditorFileSystemDirectory

Inherits: *Object*

Category: Core

29.136.1 Brief Description

A diretory for the resource filesystem.

29.136.2 Member Functions

int	*find_dir_index* (*String* name) const
int	*find_file_index* (*String* name) const
String	*get_file* (*int* idx) const
int	*get_file_count* ( ) const
bool	*get_file_import_is_valid* (*int* idx) const
String	*get_file_path* (*int* idx) const
String	*get_file_type* (*int* idx) const
String	*get_name* ( )
EditorFileSystemDirectory	*get_parent* ( )
String	*get_path* ( ) const
EditorFileSystemDirectory	*get_subdir* (*int* idx)
int	*get_subdir_count* ( ) const

29.136.3 Description

A more generalized, low-level variation of the directory concept.

29.136.4 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.

29.137 EditorImportPlugin

Inherits: *Reference* < *Object*

Category: Core

29.137.1 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.

29.137.2 Member Functions

Array	*get_import_options* (*int* preset) virtual
int	*get_import_order* ( ) virtual
String	*get_importer_name* ( ) virtual
bool	*get_option_visibility* (*String* option, *Dictionary* options) virtual
int	*get_preset_count* ( ) virtual
String	*get_preset_name* (*int* preset) virtual
float	*get_priority* ( ) virtual
Array	*get_recognized_extensions* ( ) virtual
String	*get_resource_type* ( ) virtual
String	*get_save_extension* ( ) virtual
String	*get_visible_name* ( ) virtual
int	*import* (*String* source_file, *String* save_path, *Dictionary* options, *Array* r_platform_variants, *Array* r_gen_files) virtual

29.137.3 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
```

29.137.4 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).

- *int* **get_import_order** ( ) virtual

Get the order of this importer to be run when importing resources. Higher values will be called later. Use this to ensure the importer runs after the dependencies are already imported.

- *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.

- *float* **get_priority** ( ) virtual

Get the priority of this plugin for the recognized extension. Higher priority plugins will be preferred. Default value is 1.0.

- *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

29.138 EditorInspector

Inherits: *ScrollContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.138.1 Brief Description

29.138.2 Signals

- **object_id_selected** (*int* id)
- **property_keyed** (*String* property)
- **resource_selected** (*Object* res, *String* prop)

29.139 EditorInspectorPlugin

Inherits: *Reference* < *Object*

Category: Core

29.139.1 Brief Description

29.139.2 Member Functions

void	*add_custom_control* (*Control* control)
void	*add_property_editor* (*String* property, *Control* editor)
void	*add_property_editor_for_multiple_properties* (*String* label, *PoolStringArray* properties, *Control* editor)
bool	*can_handle* (*Object* object) virtual
void	*parse_begin* (*Object* object) virtual
void	*parse_category* (*Object* object, *String* category) virtual
void	*parse_end* ( ) virtual
bool	*parse_property* (*Object* object, *int* type, *String* path, *int* hint, *String* hint_text, *int* usage) virtual

29.139.3 Member Function Description

- void **add_custom_control** (*Control* control)
- void **add_property_editor** (*String* property, *Control* editor)
- void **add_property_editor_for_multiple_properties** (*String* label, *PoolStringArray* properties, *Control* editor)
- *bool* **can_handle** (*Object* object) virtual
- void **parse_begin** (*Object* object) virtual
- void **parse_category** (*Object* object, *String* category) virtual
- void **parse_end** ( ) virtual
- *bool* **parse_property** (*Object* object, *int* type, *String* path, *int* hint, *String* hint_text, *int* usage) virtual

29.140 EditorInterface

Inherits: *Node* < *Object*

Category: Core

29.140.1 Brief Description

Editor interface and main components.

29.140.2 Member Functions

void	*edit_resource* (*Resource* resource)
Control	*get_base_control* ( )
Node	*get_edited_scene_root* ( )
EditorSettings	*get_editor_settings* ( )
Control	*get_editor_viewport* ( )
Array	*get_open_scenes* ( ) const
EditorFileSystem	*get_resource_filesystem* ( )
EditorResourcePreview	*get_resource_previewer* ( )
ScriptEditor	*get_script_editor* ( )
String	*get_selected_path* ( ) const
EditorSelection	*get_selection* ( )
void	*inspect_object* (*Object* object, *String* for_property="")
bool	*is_plugin_enabled* (*String* plugin) const
Array	*make_mesh_previews* (*Array* meshes, *int* preview_size)
void	*open_scene_from_path* (*String* scene_filepath)
void	*reload_scene_from_path* (*String* scene_filepath)
Error	*save_scene* ( )
void	*save_scene_as* (*String* path, *bool* with_preview=true)
void	*select_file* (*String* p_file)
void	*set_plugin_enabled* (*String* plugin, *bool* enabled)

29.140.3 Description

Editor interface. Allows saving and (re-)loading scenes, rendering mesh previews, inspecting and editing resources and objects and provides access to *EditorSettings*, *EditorFileSystem*, *EditorResourcePreview*, *ScriptEditor*, the editor viewport, as well as information about scenes. Also see *EditorPlugin* and *EditorScript*.

29.140.4 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 *EditorResourcePreview*er.

- *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.

- *bool* **is_plugin_enabled (*String* plugin)** const

Returns the enabled status of a plugin. The plugin name is the same as its directory name.

- *Array* **make_mesh_previews (*Array* meshes, *int* preview_size)**

Returns mesh previews rendered at the given size as an *Array* of *Texture*s.

- 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.

- *Error* **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)**
- void **set_plugin_enabled (*String* plugin, *bool* enabled)**

Sets the enabled status of a plugin. The plugin name is the same as its directory name.

29.141 EditorPlugin

Inherits: *Node* **<** *Object*

Category: Core

29.141.1 Brief Description

Used by the editor to extend its functionality.

29.141.2 Member Functions

void	*add_autoload_singleton* (*String* name, *String* path)
ToolButton	*add_control_to_bottom_panel* (*Control* control, *String* title)
void	*add_control_to_container* (*CustomControlContainer* container, *Control* control)
void	*add_control_to_dock* (*DockSlot* slot, *Control* control)
void	*add_custom_type* (*String* type, *String* base, *Script* script, *Texture* icon)
void	*add_export_plugin* (*EditorExportPlugin* plugin)
void	*add_import_plugin* (*EditorImportPlugin* importer)
void	*add_inspector_plugin* (*EditorInspectorPlugin* plugin)
void	*add_scene_import_plugin* (*EditorSceneImporter* scene_importer)
void	*add_tool_menu_item* (*String* name, *Object* handler, *String* callback, *Variant* ud=null)
void	*add_tool_submenu_item* (*String* name, *Object* submenu)
void	*apply_changes* ( ) virtual
bool	*build* ( ) virtual
void	*clear* ( ) virtual
EditorSpatialGizmo	*create_spatial_gizmo* (*Spatial* for_spatial) virtual
void	*edit* (*Object* object) virtual
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
PoolStringArray	*get_breakpoints* ( ) virtual
EditorInterface	*get_editor_interface* ( )
Object	*get_plugin_icon* ( ) virtual
String	*get_plugin_name* ( ) virtual
Dictionary	*get_state* ( ) virtual
UndoRedo	*get_undo_redo* ( )
void	*get_window_layout* (*ConfigFile* layout) virtual
bool	*handles* (*Object* object) virtual
bool	*has_main_screen* ( ) virtual
void	*hide_bottom_panel* ( )
void	*make_bottom_panel_item_visible* (*Control* item)
void	*make_visible* (*bool* visible) virtual
void	*queue_save_layout* ( ) const
void	*remove_autoload_singleton* (*String* name)
void	*remove_control_from_bottom_panel* (*Control* control)
void	*remove_control_from_container* (*CustomControlContainer* container, *Control* control)
void	*remove_control_from_docks* (*Control* control)
void	*remove_custom_type* (*String* type)
void	*remove_export_plugin* (*EditorExportPlugin* plugin)
void	*remove_import_plugin* (*EditorImportPlugin* importer)
void	*remove_inspector_plugin* (*EditorInspectorPlugin* plugin)
void	*remove_scene_import_plugin* (*EditorSceneImporter* scene_importer)
void	*remove_tool_menu_item* (*String* name)
void	*save_external_data* ( ) virtual
void	*set_force_draw_over_forwarding_enabled* ( )
void	*set_input_event_forwarding_always_enabled* ( )
void	*set_state* (*Dictionary* state) virtual
void	*set_window_layout* (*ConfigFile* layout) virtual
int	*update_overlays* ( ) const

29.141.3 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.

- **resource_saved** (*Object* resource)
- **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.

29.141.4 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**

29.141.5 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.

29.141.6 Member Function Description

- void **add_autoload_singleton** (*String* name, *String* path)

Add a script at `path` to the Autoload list as `name`.

- *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** (*CustomControlContainer* 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** (*DockSlot* 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* plugin)
- void **add_import_plugin** (*EditorImportPlugin* importer)
- void **add_inspector_plugin** (*EditorInspectorPlugin* plugin)
- void **add_scene_import_plugin** (*EditorSceneImporter* scene_importer)
- void **add_tool_menu_item** (*String* name, *Object* handler, *String* callback, *Variant* ud=null)

Adds a custom menu to 'Project > Tools' as `name` that calls `callback` on an instance of `handler` with a parameter `ud` when user activates it.

- 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.

- *bool* **build** ( ) virtual
- 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 gets 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_autoload_singleton** (*String* name)

Remove an Autoload `name` from the list.

- 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_container** (*CustomControlContainer* container, *Control* control)

Remove the control from the specified container. Use it when cleaning up after adding a control with *add_control_to_container*. Note that you can simply free the control if you won't use it anymore.

- 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* plugin)
- void **remove_import_plugin** (*EditorImportPlugin* importer)
- void **remove_inspector_plugin** (*EditorInspectorPlugin* plugin)
- void **remove_scene_import_plugin** (*EditorSceneImporter* scene_importer)
- void **remove_tool_menu_item** (*String* name)

Removes a menu `name` from 'Project > Tools'.

- 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

29.142 EditorProperty

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.142.1 Brief Description

29.142.2 Member Functions

Object	*get_edited_object* ( )
String	*get_edited_property* ( )
void	*update_property* ( ) virtual

29.142.3 Signals

- **multiple_properties_changed** (*PoolStringArray* properties, *Array* value)
- **object_id_selected** (*String* property, *int* id)
- **property_changed** (*String* property, *Nil* value)
- **property_checked** (*String* property, *String* bool)
- **property_keyed** (*String* property)
- **property_keyed_with_value** (*String* property, *Nil* value)
- **resource_selected** (*String* path, *Object* resource)
- **selected** (*String* path, *int* focusable_idx)

29.142.4 Member Variables

- *bool* **checkable**
- *bool* **checked**
- *bool* **draw_red**
- *bool* **keying**
- *String* **label**
- *bool* **read_only**

29.142.5 Member Function Description

- *Object* **get_edited_object** ( )
- *String* **get_edited_property** ( )
- void **update_property** ( ) virtual

29.143 EditorResourceConversionPlugin

Inherits: *Reference* < *Object*

Category: Core

29.143.1 Brief Description

29.143.2 Member Functions

Resource	*_convert* (*Resource* resource) virtual
String	*_converts_to* ( ) virtual

29.143.3 Member Function Description

- *Resource* **_convert** (*Resource* resource) virtual
- *String* **_converts_to** ( ) virtual

29.144 EditorResourcePreview

Inherits: *Node* < *Object*

Category: Core

29.144.1 Brief Description

Helper to generate previews of resources or files.

29.144.2 Member Functions

void	*add_preview_generator* (*EditorResourcePreviewGenerator* generator)
void	*check_for_invalidation* (*String* path)
void	*queue_edited_resource_preview* (*Resource* resource, *Object* receiver, *String* receiver_func, *Variant* user-data)
void	*queue_resource_preview* (*String* path, *Object* receiver, *String* receiver_func, *Variant* userdata)
void	*remove_preview_generator* (*EditorResourcePreviewGenerator* generator)

29.144.3 Signals

- **preview_invalidated** (*String* path)

If a preview was invalidated (changed) this signal will emit (using the path of the preview)

29.144.4 Description

This object is used to generate previews for resources of files.

29.144.5 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.

29.145 EditorResourcePreviewGenerator

Inherits: *Reference* < *Object*

Category: Core

29.145.1 Brief Description

Custom generator of previews.

29.145.2 Member Functions

Texture	*generate* (*Resource* from) virtual
Texture	*generate_from_path* (*String* path) virtual
bool	*handles* (*String* type) virtual

29.145.3 Description

Custom code to generate previews. Please check "file_dialog/thumbnail_size" in EditorSettings to find out the right size to do previews at.

29.145.4 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.

29.146 EditorSceneImporter

Inherits: *Reference* < *Object*

Category: Core

29.146.1 Brief Description

29.146.2 Member Functions

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)

29.146.3 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**

29.146.4 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)

29.147 EditorScenePostImport

Inherits: *Reference* < *Object*

Category: Core

29.147.1 Brief Description

29.147.2 Member Functions

void	*post_import* (*Object* scene) virtual

29.147.3 Member Function Description

- void **post_import** (*Object* scene) virtual

29.148 EditorScript

Inherits: *Reference* < *Object*

Category: Core

29.148.1 Brief Description

Base script that can be used to add extension functions to the editor.

29.148.2 Member Functions

void	*_run* ( ) virtual
void	*add_root_node* (*Node* node)
EditorInterface	*get_editor_interface* ( )
Node	*get_scene* ( )

29.148.3 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 *EditorPlugin*s 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.

29.148.4 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.

29.149 EditorSelection

Inherits: *Object*

Category: Core

29.149.1 Brief Description

Manages the SceneTree selection in the editor.

29.149.2 Member Functions

void	*add_node* (*Node* node)
void	*clear* ( )
Array	*get_selected_nodes* ( )
Array	*get_transformable_selected_nodes* ( )
void	*remove_node* (*Node* node)

29.149.3 Signals

- **selection_changed** ( )

Emitted when the selection changes.

29.149.4 Description

This object manages the SceneTree selection in the editor.

29.149.5 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 chid/grandchild.

- void **remove_node** (*Node* node)

Remove a node from the selection.

29.150 EditorSettings

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.150.1 Brief Description

Object that holds the project-independent editor settings.

29.150.2 Member Functions

void	*add_property_info* (*Dictionary* info)
void	*erase* (*String* property)
PoolStringArray	*get_favorite_dirs* ( ) const
Variant	*get_project_metadata* (*String* section, *String* key, *Variant* default=null) const
String	*get_project_settings_dir* ( ) const
PoolStringArray	*get_recent_dirs* ( ) const
Variant	*get_setting* (*String* name) const
String	*get_settings_dir* ( ) const
bool	*has_setting* (*String* name) const
bool	*property_can_revert* (*String* name)
Variant	*property_get_revert* (*String* name)
void	*set_favorite_dirs* (*PoolStringArray* dirs)
void	*set_initial_value* (*String* name, *Variant* value, *bool* update_current)
void	*set_project_metadata* (*String* section, *String* key, *Variant* data)
void	*set_recent_dirs* (*PoolStringArray* dirs)
void	*set_setting* (*String* name, *Variant* value)

29.150.3 Signals

- **settings_changed** ( )

29.150.4 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()
```

29.150.5 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"
}
```

(continues on next page)

(continued from previous page)

```
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.

- *Variant* **get_project_metadata** (*String* section, *String* key, *Variant* default=null) const
- *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_project_metadata** (*String* section, *String* key, *Variant* data)
- 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)

29.151 EditorSpatialGizmo

Inherits: *SpatialGizmo* < *Reference* < *Object*

Category: Core

29.151.1 Brief Description

Custom gizmo for editing Spatial objects.

29.151.2 Member Functions

void	*add_collision_segments* (*PoolVector3Array* segments)
void	*add_collision_triangles* (*TriangleMesh* triangles, *AABB* bounds)
void	*add_handles* (*PoolVector3Array* handles, *bool* billboard=false, *bool* secondary=false)
void	*add_lines* (*PoolVector3Array* lines, *Material* material, *bool* billboard=false)
void	*add_mesh* (*ArrayMesh* mesh, *bool* billboard=false, *RID* skeleton)
void	*add_unscaled_billboard* (*Material* material, *float* default_scale=1)
void	*clear* ( )
void	*commit_handle* (*int* index, *Variant* restore, *bool* cancel=false) virtual
String	*get_handle_name* (*int* index) virtual
Variant	*get_handle_value* (*int* index) virtual
void	*redraw* ( ) virtual
void	*set_handle* (*int* index, *Camera* camera, *Vector2* point) virtual
void	*set_spatial_node* (*Node* node)

29.151.3 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*.

29.151.4 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)

29.152 EncodedObjectAsID

Inherits: *Reference* < *Object*

Category: Core

29.152.1 Brief Description

29.152.2 Member Functions

int	*get_object_id* ( ) const
void	*set_object_id* (*int* id)

29.152.3 Member Function Description

- *int* **get_object_id** ( ) const
- void **set_object_id** (*int* id)

29.153 Engine

Inherits: *Object*

Category: Core

29.153.1 Brief Description

Access to basic engine properties.

29.153.2 Member Functions

Dictionary	*get_author_info* ( ) const
Array	*get_copyright_info* ( ) const
Dictionary	*get_donor_info* ( ) const
int	*get_frames_drawn* ( )
float	*get_frames_per_second* ( ) const
Dictionary	*get_license_info* ( ) const
String	*get_license_text* ( ) const
MainLoop	*get_main_loop* ( ) const
Object	*get_singleton* (*String* name) const
Dictionary	*get_version_info* ( ) const
bool	*has_singleton* (*String* name) const
bool	*is_in_physics_frame* ( ) const

29.153.3 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).
- *float* **physics_jitter_fix**
- *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.

29.153.4 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.

29.153.5 Member Function Description

- *Dictionary* **get_author_info** ( ) const

Returns engine author information in a Dictionary.

"lead_developers" - Array of Strings, lead developer names

"founders" - Array of Strings, founder names

"project_managers" - Array of Strings, project manager names

"developers" - Array of Strings, developer names

- *Array* **get_copyright_info** ( ) const

Returns an Array of copyright information Dictionaries.

"name" - String, component name

“parts” - Array of Dictionaries {“files”, “copyright”, “license”} describing subsections of the component

- *Dictionary* **get_donor_info** **(** **)** const

Returns a Dictionary of Arrays of donor names.

{“platinum_sponsors”, “gold_sponsors”, “mini_sponsors”, “gold_donors”, “silver_donors”, “bronze_donors”}

- *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.

- *Dictionary* **get_license_info** **(** **)** const

Returns Dictionary of licenses used by Godot and included third party components.

- *String* **get_license_text** **(** **)** const

Returns Godot license text.

- *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 an int

“minor” - Holds the minor version number as an int

“patch” - Holds the patch version number as an int

“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.

29.154 Environment

Inherits: *Resource* **<** *Reference* **<** *Object*

Category: Core

29.154.1 Brief Description

Resource for environment nodes (like *WorldEnvironment*) that define multiple rendering options.

29.154.2 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.

29.154.3 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: color = color / (1 + 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**

29.154.4 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

29.155 File

Inherits: *Reference* < *Object*

Category: Core

29.155.1 Brief Description

Type to handle file reading and writing operations.

29.155.2 Member Functions

void	*close* ( )
bool	*eof_reached* ( ) const
bool	*file_exists* (*String* path) const
int	*get_16* ( ) const
int	*get_32* ( ) const
int	*get_64* ( ) const
int	*get_8* ( ) const
String	*get_as_text* ( ) const
PoolByteArray	*get_buffer* (*int* len) const
PoolStringArray	*get_csv_line* (*String* delim=”,”) const
float	*get_double* ( ) const
Error	*get_error* ( ) const
float	*get_float* ( ) const
int	*get_len* ( ) const
String	*get_line* ( ) const
String	*get_md5* (*String* path) const
int	*get_modified_time* (*String* file) const
String	*get_pascal_string* ( )
String	*get_path* ( ) const
String	*get_path_absolute* ( ) const
int	*get_position* ( ) const
float	*get_real* ( ) const
String	*get_sha256* (*String* path) const
Variant	*get_var* ( ) const
bool	*is_open* ( ) const
Error	*open* (*String* path, *int* flags)
Error	*open_compressed* (*String* path, *int* mode_flags, *int* compression_mode=0)
Error	*open_encrypted* (*String* path, *int* mode_flags, *PoolByteArray* key)
Error	*open_encrypted_with_pass* (*String* path, *int* mode_flags, *String* pass)
void	*seek* (*int* position)
void	*seek_end* (*int* position=0)
void	*store_16* (*int* value)
void	*store_32* (*int* value)
void	*store_64* (*int* value)
void	*store_8* (*int* value)
void	*store_buffer* (*PoolByteArray* buffer)
void	*store_double* (*float* value)
void	*store_float* (*float* value)
void	*store_line* (*String* line)
void	*store_pascal_string* (*String* string)
void	*store_real* (*float* value)
void	*store_string* (*String* string)
void	*store_var* (*Variant* value)

29.155.3 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 reset to `false` whenever you open the file.

29.155.4 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.

29.155.5 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
```

29.155.6 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.

- *Error* **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.

- *String* **get_path** ( ) const

Returns the path as a *String* for the current open file.

- *String* **get_path_absolute** ( ) const

Returns the absolute path as a *String* for the current open 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.

- *Error* **open (** *String* path, *int* flags **)**

Opens the file for writing or reading, depending on the flags.

- *Error* **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`.

- *Error* **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.

- *Error* **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.

29.156 FileDialog

Inherits: *ConfirmationDialog* < *AcceptDialog* < *WindowDialog* < *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.156.1 Brief Description

Dialog for selecting files or directories in the filesystem.

29.156.2 Member Functions

void	*add_filter* (*String* filter)
void	*clear_filters* ( )
void	*deselect_items* ( )
VBoxContainer	*get_vbox* ( )
void	*invalidate* ( )

29.156.3 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.

29.156.4 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**

29.156.5 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.

29.156.6 Description

FileDialog is a preset dialog used to choose files and directories in the filesystem. It supports filter masks.

29.156.7 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.

29.157 float

Category: Built-In Types

29.157.1 Brief Description

Float built-in type

29.157.2 Member Functions

float	float (*bool* from)
float	float (*int* from)
float	float (*String* from)

29.157.3 Description

Float built-in type.

29.157.4 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.

29.158 Font

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *DynamicFont*, *BitmapFont*

Category: Core

29.158.1 Brief Description

Internationalized font and text drawing support.

29.158.2 Member Functions

void	*draw* (*RID* canvas_item, *Vector2* position, *String* string, *Color* modulate=Color(1, 1, 1, 1), *int* clip_w=-1, *Color* outline_modulate=Color(1, 1, 1, 1)) const
float	*draw_char* (*RID* canvas_item, *Vector2* position, *int* char, *int* next=-1, *Color* modulate=Color(1, 1, 1, 1), *bool* outline=false) const
float	*get_ascent* ( ) const
float	*get_descent* ( ) const
float	*get_height* ( ) const
Vector2	*get_string_size* (*String* string) const
bool	*has_outline* ( ) const
bool	*is_distance_field_hint* ( ) const
void	*update_changes* ( )

29.158.3 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.

29.158.4 Member Function Description

- void **draw** (*RID* canvas_item, *Vector2* position, *String* string, *Color* modulate=Color(1, 1, 1, 1), *int* clip_w=-1, *Color* outline_modulate=Color(1, 1, 1, 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), *bool* outline=false) 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* **has_outline** ( ) const
- *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.

29.159 FuncRef

Inherits: *Reference* < *Object*

Category: Core

29.159.1 Brief Description

Reference to a function in an object.

29.159.2 Member Functions

Variant	*call_func* ( ) vararg
void	*set_function* (*String* name)
void	*set_instance* (*Object* instance)

29.159.3 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.

29.159.4 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*.

29.160 GDNative

Inherits: *Reference* < *Object*

Category: Core

29.160.1 Brief Description

29.160.2 Member Functions

Variant	*call_native* (*String* calling_type, *String* procedure_name, *Array* arguments)
bool	*initialize* ( )
bool	*terminate* ( )

29.160.3 Member Variables

- *GDNativeLibrary* **library**

29.160.4 Member Function Description

- *Variant* **call_native** (*String* calling_type, *String* procedure_name, *Array* arguments)
- *bool* **initialize** ( )
- *bool* **terminate** ( )

29.161 GDNativeLibrary

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.161.1 Brief Description

29.161.2 Member Functions

PoolStringArray	*get_current_dependencies* ( ) const
String	*get_current_library_path* ( ) const

29.161.3 Member Variables

- *ConfigFile* **config_file**
- *bool* **load_once**

- *bool* **reloadable**
- *bool* **singleton**
- *String* **symbol_prefix**

29.161.4 Member Function Description

- *PoolStringArray* **get_current_dependencies** ( ) const
- *String* **get_current_library_path** ( ) const

29.162 GDScript

Inherits: *Script* < *Resource* < *Reference* < *Object*

Category: Core

29.162.1 Brief Description

A script implemented in the GDScript programming language.

29.162.2 Member Functions

PoolByteArray	*get_as_byte_code* ( ) const
Object	*new* ( ) vararg

29.162.3 Description

A script implemented in the GDScript programming language. The script exends 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.

29.162.4 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)
```

29.163 GDScriptFunctionState

Inherits: *Reference* < *Object*

Category: Core

29.163.1 Brief Description

State of a function call after yielding.

29.163.2 Member Functions

bool	*is_valid* (*bool* extended_check=false) const
Variant	*resume* (*Variant* arg=null)

29.163.3 Signals

- **completed** (*Nil* result)

29.163.4 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.

29.163.5 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.

29.164 GDScriptNativeClass

Inherits: *Reference* < *Object*

Category: Core

29.164.1 Brief Description

29.164.2 Member Functions

Variant	*new* **( )**

29.164.3 Member Function Description

- *Variant* **new ( )**

29.165 Generic6DOFJoint

Inherits: *Joint* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.165.1 Brief Description

The generic 6 degrees of freedom joint can implement a variety of joint-types by locking certain axes' rotation or translation.

29.165.2 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 limited.
- *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 limited.

- *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 limited.
- *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 limited.
- *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 limited.
- *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 limited.
- *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.
- *bool* **linear_motor_x/enabled** - If `true` then there is a linear motor on the x-axis. It will attempt to reach the target velocity while staying within the force limits.
- *float* **linear_motor_x/force_limit** - The maximum force the linear motor can apply on the x-axis while trying to reach the target velocity.
- *float* **linear_motor_x/target_velocity** - The speed that the linear motor will attempt to reach on the x-axis.
- *bool* **linear_motor_y/enabled** - If `true` then there is a linear motor on the y-axis. It will attempt to reach the target velocity while staying within the force limits.
- *float* **linear_motor_y/force_limit** - The maximum force the linear motor can apply on the y-axis while trying to reach the target velocity.
- *float* **linear_motor_y/target_velocity** - The speed that the linear motor will attempt to reach on the y-axis.
- *bool* **linear_motor_z/enabled** - If `true` then there is a linear motor on the z-axis. It will attempt to reach the target velocity while staying within the force limits.
- *float* **linear_motor_z/force_limit** - The maximum force the linear motor can apply on the z-axis while trying to reach the target velocity.
- *float* **linear_motor_z/target_velocity** - The speed that the linear motor will attempt to reach on the z-axis.

29.165.3 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_ENABLE_LINEAR_MOTOR = 3**

- **FLAG_MAX = 4** — 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_LINEAR_MOTOR_TARGET_VELOCITY = 5** — The velocity the linear motor will try to reach.
- **PARAM_LINEAR_MOTOR_FORCE_LIMIT = 6** — The maximum force the linear motor will apply while trying to reach the velocity target.
- **PARAM_ANGULAR_LOWER_LIMIT = 7** — The minimum rotation in negative direction to break loose and rotate around the axes.
- **PARAM_ANGULAR_UPPER_LIMIT = 8** — The minimum rotation in positive direction to break loose and rotate around the axes.
- **PARAM_ANGULAR_LIMIT_SOFTNESS = 9** — The speed of all rotations across the axes.
- **PARAM_ANGULAR_DAMPING = 10** — The amount of rotational damping across the axes. The lower, the more dampening occurs.
- **PARAM_ANGULAR_RESTITUTION = 11** — The amount of rotational restitution across the axes. The lower, the more restitution occurs.
- **PARAM_ANGULAR_FORCE_LIMIT = 12** — The maximum amount of force that can occur, when rotating around the axes.
- **PARAM_ANGULAR_ERP = 13** — 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 = 14** — Target speed for the motor at the axes.
- **PARAM_ANGULAR_MOTOR_FORCE_LIMIT = 15** — Maximum acceleration for the motor at the axes.
- **PARAM_MAX = 16** — End flag of PARAM_* constants, used internally.

29.165.4 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.

29.166 Geometry

Inherits: *Object*

Category: Core

29.166.1 Brief Description

29.166.2 Member Functions

Array	*build_box_planes* (*Vector3* extents)
Array	*build_capsule_planes* (*float* radius, *float* height, *int* sides, *int* lats, Axis axis=2)
Array	*build_cylinder_planes* (*float* radius, *float* height, *int* sides, Axis axis=2)
PoolVector3Array	*clip_polygon* (*PoolVector3Array* points, *Plane* plane)
PoolVector2Array	*convex_hull_2d* (*PoolVector2Array* points)
Vector3	*get_closest_point_to_segment* (*Vector3* point, *Vector3* s1, *Vector3* s2)
Vector2	*get_closest_point_to_segment_2d* (*Vector2* point, *Vector2* s1, *Vector2* s2)
Vector3	*get_closest_point_to_segment_uncapped* (*Vector3* point, *Vector3* s1, *Vector3* s2)
Vector2	*get_closest_point_to_segment_uncapped_2d* (*Vector2* point, *Vector2* s1, *Vector2* s2)
PoolVector3Array	*get_closest_points_between_segments* (*Vector3* p1, *Vector3* p2, *Vector3* q1, *Vector3* q2)
PoolVector2Array	*get_closest_points_between_segments_2d* (*Vector2* p1, *Vector2* q1, *Vector2* p2, *Vector2* q2)
int	*get_uv84_normal_bit* (*Vector3* normal)
Variant	*line_intersects_line_2d* (*Vector2* from_a, *Vector2* dir_a, *Vector2* from_b, *Vector2* dir_b)
Dictionary	*make_atlas* (*PoolVector2Array* sizes)
bool	*point_is_inside_triangle* (*Vector2* point, *Vector2* a, *Vector2* b, *Vector2* c) const
Variant	*ray_intersects_triangle* (*Vector3* from, *Vector3* dir, *Vector3* a, *Vector3* b, *Vector3* c)
float	*segment_intersects_circle* (*Vector2* segment_from, *Vector2* segment_to, *Vector2* circle_position, *float* circle_radius)
PoolVector3Array	*segment_intersects_convex* (*Vector3* from, *Vector3* to, *Array* planes)
PoolVector3Array	*segment_intersects_cylinder* (*Vector3* from, *Vector3* to, *float* height, *float* radius)
Variant	*segment_intersects_segment_2d* (*Vector2* from_a, *Vector2* to_a, *Vector2* from_b, *Vector2* to_b)
PoolVector3Array	*segment_intersects_sphere* (*Vector3* from, *Vector3* to, *Vector3* sphere_position, *float* sphere_radius)
Variant	*segment_intersects_triangle* (*Vector3* from, *Vector3* to, *Vector3* a, *Vector3* b, *Vector3* c)
PoolIntArray	*triangulate_polygon* (*PoolVector2Array* polygon)

29.166.3 Member Function Description

- *Array* **build_box_planes** (*Vector3* extents)

Returns an array with 6 *Plane*s 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, Axis axis=2)

Returns an array of *Plane*s 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, Axis axis=2)

Returns an array of *Plane*s 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)

- *Variant* **line_intersects_line_2d** (*Vector2* from_a, *Vector2* dir_a, *Vector2* from_b, *Vector2* dir_b)

Checks if the two lines (`from_a`, `dir_a`) and (`from_b`, `dir_b`) intersect. If yes, return the point of intersection as *Vector2*. If no intersection takes place, returns an empty *Variant*. Note that the lines are specified using direction vectors, not end points.

- *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 though the *Plane*s 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 * 3` elements, with `n` being the number of found triangles). If the triangulation did not succeed, an empty *PoolIntArray* is returned.

29.167 GeometryInstance

Inherits: *VisualInstance* < *Spatial* < *Node* < *Object*

Inherited By: *MultiMeshInstance*, *MeshInstance*, *Particles*, *SpriteBase3D*, *ImmediateGeometry*

Category: Core

29.167.1 Brief Description

Base node for geometry based visual instances.

29.167.2 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.

29.167.3 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.

29.167.4 Description

Base node for geometry based visual instances. Shares some common functionality like visibility and custom materials.

29.168 GIProbe

Inherits: *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.168.1 Brief Description

29.168.2 Member Functions

void	*bake* (*Node* from_node=null, *bool* create_visual_debug=false)
void	*debug_bake* ( )

29.168.3 Member Variables

- *float* **bias**
- *bool* **compress**
- *GIProbeData* **data**
- *int* **dynamic_range**
- *float* **energy**
- *Vector3* **extents**
- *bool* **interior**
- *float* **normal_bias**
- *float* **propagation**
- *Subdiv* **subdiv**

29.168.4 Enums

enum **Subdiv**

- **SUBDIV_64 = 0**
- **SUBDIV_128 = 1**
- **SUBDIV_256 = 2**
- **SUBDIV_512 = 3**
- **SUBDIV_MAX = 4**

29.168.5 Member Function Description

- void **bake** (*Node* from_node=null, *bool* create_visual_debug=false)
- void **debug_bake** ( )

29.169 GIProbeData

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.169.1 Brief Description

29.169.2 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**

29.170 GodotSharp

Inherits: *Object*

Category: Core

29.170.1 Brief Description

29.170.2 Member Functions

void	*attach_thread* ( )
void	*detach_thread* ( )
bool	*is_domain_loaded* ( )
bool	*is_finalizing_domain* ( )

29.170.3 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.

29.171 Gradient

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.171.1 Brief Description

Color interpolator node.

29.171.2 Member Functions

void	*add_point* (*float* offset, *Color* color)
Color	*get_color* (*int* point) const
float	*get_offset* (*int* point) const
int	*get_point_count* ( ) const
Color	*interpolate* (*float* offset)
void	*remove_point* (*int* offset)
void	*set_color* (*int* point, *Color* color)
void	*set_offset* (*int* point, *float* offset)

29.171.3 Member Variables

- *PoolColorArray* **colors** - Gradient's colors returned as a *PoolColorArray*.
- *PoolRealArray* **offsets** - Gradient's offsets returned as a *PoolRealArray*.

29.171.4 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.

29.171.5 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*

29.172 GradientTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.172.1 Brief Description

Gradient filled texture.

29.172.2 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*.

29.172.3 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).

29.173 GraphEdit

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.173.1 Brief Description

GraphEdit is an area capable of showing various GraphNodes. It manages connection events between them.

29.173.2 Member Functions

void	*add_valid_connection_type* (*int* from_type, *int* to_type)
void	*add_valid_left_disconnect_type* (*int* type)
void	*add_valid_right_disconnect_type* (*int* type)
void	*clear_connections* ( )
Error	*connect_node* (*String* from, *int* from_port, *String* to, *int* to_port)
void	*disconnect_node* (*String* from, *int* from_port, *String* to, *int* to_port)
Array	*get_connection_list* ( ) const
bool	*is_node_connected* (*String* from, *int* from_port, *String* to, *int* to_port)
bool	*is_valid_connection_type* (*int* from_type, *int* to_type) const
void	*remove_valid_connection_type* (*int* from_type, *int* to_type)
void	*remove_valid_left_disconnect_type* (*int* type)
void	*remove_valid_right_disconnect_type* (*int* type)
void	*set_selected* (*Node* node)

29.173.3 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)

Emitted when a GraphNode is selected.

- **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)

29.173.4 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.

29.173.5 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.

29.173.6 Member Function Description

- void **add_valid_connection_type** **(** *int* from_type, *int* to_type **)**

Makes possible the connection between two different slot types. The type is defined with the *GraphNode.set_slot* method.

- void **add_valid_left_disconnect_type** **(** *int* type **)**

Makes possible to disconnect nodes when dragging from the slot at the left if it has the specified type.

- void **add_valid_right_disconnect_type** **(** *int* type **)**

Makes possible to disconnect nodes when dragging from the slot at the right if it has the specified type.

- void **clear_connections** **( )**

Remove all connections between nodes.

- *Error* **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.

- *bool* **is_valid_connection_type** **(** *int* from_type, *int* to_type **)** const

Returns whether it's possible to connect slots of the specified types.

- void **remove_valid_connection_type** (*int* from_type, *int* to_type)

Makes it not possible to connect between two different slot types. The type is defined with the *GraphNode.set_slot* method.

- void **remove_valid_left_disconnect_type** (*int* type)

Removes the possibility to disconnect nodes when dragging from the slot at the left if it has the specified type.

- void **remove_valid_right_disconnect_type** (*int* type)

Removes the possibility to disconnect nodes when dragging from the slot at the right if it has the specified type.

- void **set_selected** (*Node* node)

Sets the specified `node` as the one selected.

29.174 GraphNode

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.174.1 Brief Description

A GraphNode is a container with several input and output slots allowing connections between GraphNodes. Slots can have different, incompatible types.

29.174.2 Member Functions

void	*clear_all_slots* ( )
void	*clear_slot* (*int* idx)
Color	*get_connection_input_color* (*int* idx)
int	*get_connection_input_count* ( )
Vector2	*get_connection_input_position* (*int* idx)
int	*get_connection_input_type* (*int* idx)
Color	*get_connection_output_color* (*int* idx)
int	*get_connection_output_count* ( )
Vector2	*get_connection_output_position* (*int* idx)
int	*get_connection_output_type* (*int* idx)
Color	*get_slot_color_left* (*int* idx) const
Color	*get_slot_color_right* (*int* idx) const
int	*get_slot_type_left* (*int* idx) const
int	*get_slot_type_right* (*int* idx) const
bool	*is_slot_enabled_left* (*int* idx) const
bool	*is_slot_enabled_right* (*int* idx) const
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)

29.174.3 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 **)**

29.174.4 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**

29.174.5 Enums

enum **Overlay**

- **OVERLAY_DISABLED = 0**
- **OVERLAY_BREAKPOINT = 1**
- **OVERLAY_POSITION = 2**

29.174.6 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.

29.174.7 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)

29.175 GridContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.175.1 Brief Description

Grid container used to arrange elements in a grid like layout.

29.175.2 Member Functions

Control	*get_child_control_at_cell* (*int* row, *int* column)

29.175.3 Member Variables

- *int* **columns** - The number of columns in the `GridContainer`. If modified, `GridContainer` reorders its children to accommodate the new layout.

29.175.4 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.

29.175.5 Member Function Description

- *Control* **get_child_control_at_cell** (*int* row, *int* column)

29.176 GridMap

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.176.1 Brief Description

Node for 3D tile-based maps.

29.176.2 Member Functions

void	*clear* ( )
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
int	*get_cell_item_orientation* (*int* x, *int* y, *int* z) const
bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
Array	*get_meshes* ( )
Array	*get_used_cells* ( ) const
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)
void	*set_clip* (*bool* enabled, *bool* clipabove=true, *int* floor=0, Axis 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

29.176.3 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*.

29.176.4 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*).

29.176.5 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.

29.176.6 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, Axis 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

29.177 GrooveJoint2D

Inherits: *Joint2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.177.1 Brief Description

Groove constraint for 2D physics.

29.177.2 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`

29.177.3 Description

Groove constraint for 2D physics. This is useful for making a body "slide" through a segment placed in another.

29.178 HBoxContainer

Inherits: *BoxContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.178.1 Brief Description

Horizontal box container.

29.178.2 Description

Horizontal box container. See *BoxContainer*.

29.179 HingeJoint

Inherits: *Joint* < *Spatial* < *Node* < *Object*

Category: Core

29.179.1 Brief Description

A hinge between two 3D bodies.

29.179.2 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.

29.179.3 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.

29.179.4 Description

Normally uses the z-axis of body A as the hinge axis, another axis can be specified when adding it manually though.

29.180 HScrollBar

Inherits: *ScrollBar* < *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.180.1 Brief Description

Horizontal scroll bar.

29.180.2 Description

Horizontal scroll bar. See *ScrollBar*. This one goes from left (min) to right (max).

29.181 HSeparator

Inherits: *Separator* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.181.1 Brief Description

Horizontal separator.

29.181.2 Description

Horizontal separator. See *Separator*. It is used to separate objects vertically, though (but it looks horizontal!).

29.182 HSlider

Inherits: *Slider* < *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.182.1 Brief Description

Horizontal slider.

29.182.2 Description

Horizontal slider. See *Slider*. This one goes from left (min) to right (max).

29.183 HSplitContainer

Inherits: *SplitContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.183.1 Brief Description

Horizontal split container.

29.183.2 Description

Horizontal split container. See *SplitContainer*. This goes from left to right.

29.184 HTTPClient

Inherits: *Reference* < *Object*

Category: Core

29.184.1 Brief Description

Hyper-text transfer protocol client.

29.184.2 Member Functions

void	*close* ( )
Error	*connect_to_host* (*String* host, *int* port=-1, *bool* use_ssl=false, *bool* verify_host=true)
int	*get_response_body_length* ( ) const
int	*get_response_code* ( ) const
PoolStringArray	*get_response_headers* ( )
Dictionary	*get_response_headers_as_dictionary* ( )
Status	*get_status* ( ) const
bool	*has_response* ( ) const
bool	*is_response_chunked* ( ) const
Error	*poll* ( )
String	*query_string_from_dict* (*Dictionary* fields)
PoolByteArray	*read_response_body_chunk* ( )
Error	*request* (*Method* method, *String* url, *PoolStringArray* headers, *String* body="")
Error	*request_raw* (*Method* method, *String* url, *PoolStringArray* headers, *PoolByteArray* body)
void	*set_read_chunk_size* (*int* bytes)

29.184.3 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.

29.184.4 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.

29.184.5 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).

29.184.6 Member Function Description

- void **close** **(** **)**

Closes the current connection, allowing reuse of this `HTTPClient`.

- *Error* **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)

- *Status* **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.

- *Error* **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"
```

Furthermore, if a key has a null value, only the key itself is added, without equal sign and value. If the value is an array, for each value in it a pair with the same key is added.

```
var fields = {"single": 123, "not_valued": null, "multiple": [22, 33, 44]}
String queryString = httpClient.query_string_from_dict(fields)
returns:= "single=123&not_valued&multiple=22&multiple=33&multiple=44"
```

- *PoolByteArray* **read_response_body_chunk ( )**

Reads one chunk from the response.

- *Error* **request** (*Method* 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)
```

- *Error* **request_raw** (*Method* 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*

29.185 HTTPRequest

Inherits: *Node* < *Object*

Category: Core

29.185.1 Brief Description

A node with the ability to send HTTP requests.

29.185.2 Member Functions

void	*cancel_request* ( )
int	*get_body_size* ( ) const
int	*get_downloaded_bytes* ( ) const
Status	*get_http_client_status* ( ) const
Error	*request* (*String* url, *PoolStringArray* custom_headers=PoolStringArray( ), *bool* ssl_validate_domain=true, *Method* method=0, *String* request_data="")

29.185.3 Signals

- **request_completed** (*int* result, *int* response_code, *PoolStringArray* headers, *PoolByteArray* body)

This signal is emitted upon request completion.

29.185.4 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.

29.185.5 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.

29.185.6 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.

29.185.7 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.

- *Status* **get_http_client_status** ( ) const

Returns the current status of the underlying *HTTPClient*. See `STATUS_*` enum on *HTTPClient*.

- *Error* **request** (*String* url, *PoolStringArray* custom_headers=PoolStringArray( ), *bool* ssl_validate_domain=true, *Method* method=0, *String* request_data="")

Creates request on the underlying *HTTPClient*. If there is no configuration errors, it tries to connect using *HTTPClient.connect_to_host* and passes parameters onto *HTTPClient.request*.

Returns `OK` if request is successfully created. (Does not imply that the server has responded), `ERR_UNCONFIGURED` if not in the tree, `ERR_BUSY` if still processing previous request, `ERR_INVALID_PARAMETER` if given string is not a valid URL format, or `ERR_CANT_CONNECT` if not using thread and the *HTTPClient* cannot connect to host.

29.186 Image

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.186.1 Brief Description

Image datatype.

29.186.2 Member Functions

void	*blend_rect* (*Image* src, *Rect2* src_rect, *Vector2* dst)
void	*blend_rect_mask* (*Image* src, *Image* mask, *Rect2* src_rect, *Vector2* dst)
void	*blit_rect* (*Image* src, *Rect2* src_rect, *Vector2* dst)
void	*blit_rect_mask* (*Image* src, *Image* mask, *Rect2* src_rect, *Vector2* dst)
void	*bumpmap_to_normalmap* (*float* bump_scale=1.0)
void	*clear_mipmaps* ( )
Error	*compress* (*CompressMode* mode, *CompressSource* source, *float* lossy_quality)
void	*convert* (*Format* format)

Continued on next page

Table 10 – continued from previous page

void	*copy_from* (*Image* src)
void	*create* (*int* width, *int* height, *bool* use_mipmaps, *Format* format)
void	*create_from_data* (*int* width, *int* height, *bool* use_mipmaps, *Format* format, *PoolByteArray* data)
void	*crop* (*int* width, *int* height)
Error	*decompress* ( )
AlphaMode	*detect_alpha* ( ) const
void	*expand_x2_hq2x* ( )
void	*fill* (*Color* color)
void	*fix_alpha_edges* ( )
void	*flip_x* ( )
void	*flip_y* ( )
Error	*generate_mipmaps* (*bool* renormalize=false)
PoolByteArray	*get_data* ( ) const
Format	*get_format* ( ) const
int	*get_height* ( ) const
int	*get_mipmap_offset* (*int* mipmap) const
Color	*get_pixel* (*int* x, *int* y) const
Color	*get_pixelv* (*Vector2* src) const
Image	*get_rect* (*Rect2* rect) const
Vector2	*get_size* ( ) const
Rect2	*get_used_rect* ( ) const
int	*get_width* ( ) const
bool	*has_mipmaps* ( ) const
bool	*is_compressed* ( ) const
bool	*is_empty* ( ) const
bool	*is_invisible* ( ) const
Error	*load* (*String* path)
Error	*load_jpg_from_buffer* (*PoolByteArray* buffer)
Error	*load_png_from_buffer* (*PoolByteArray* buffer)
void	*lock* ( )
void	*normalmap_to_xy* ( )
void	*premultiply_alpha* ( )
void	*resize* (*int* width, *int* height, *Interpolation* interpolation=1)
void	*resize_to_po2* (*bool* square=false)
Error	*save_png* (*String* path) const
void	*set_pixel* (*int* x, *int* y, *Color* color)
void	*set_pixelv* (*Vector2* dst, *Color* color)
void	*shrink_x2* ( )
void	*srgb_to_linear* ( )
void	*unlock* ( )

29.186.3 Member Variables

- *Dictionary* **data** - Holds all of the image's color data in a given format. See `FORMAT_*` constants.

29.186.4 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** — OpenGL texture format RED with a single component and a bitdepth of 8.
- **FORMAT_RG8 = 3** — OpenGL texture format RG with two components and a bitdepth of 8 for each.
- **FORMAT_RGB8 = 4** — OpenGL texture format RGB with three components, each with a bitdepth of 8.
- **FORMAT_RGBA8 = 5** — OpenGL texture format RGBA with four components, each with a bitdepth of 8.
- **FORMAT_RGBA4444 = 6** — OpenGL texture format RGBA with four components, each with a bitdepth of 4.
- **FORMAT_RGBA5551 = 7** — OpenGL texture format GL_RGB5_A1 where 5 bits of depth for each component of RGB and one bit for alpha.
- **FORMAT_RF = 8** — OpenGL texture format GL_R32F where there's one component, a 32-bit floating-point value.
- **FORMAT_RGF = 9** — OpenGL texture format GL_RG32F where there are two components, each a 32-bit floating-point values.
- **FORMAT_RGBF = 10** — OpenGL texture format GL_RGB32F where there are three components, each a 32-bit floating-point values.
- **FORMAT_RGBAF = 11** — OpenGL texture format GL_RGBA32F where there are four components, each a 32-bit floating-point values.

- **FORMAT_RH = 12** — OpenGL texture format GL_R32F where there's one component, a 16-bit "half-precision" floating-point value.
- **FORMAT_RGH = 13** — OpenGL texture format GL_RG32F where there's two components, each a 16-bit "half-precision" floating-point value.
- **FORMAT_RGBH = 14** — OpenGL texture format GL_RGB32F where there's three components, each a 16-bit "half-precision" floating-point value.
- **FORMAT_RGBAH = 15** — OpenGL texture format GL_RGBA32F where there's four components, each a 16-bit "half-precision" floating-point value.
- **FORMAT_RGBE9995 = 16** — A special OpenGL texture format where the three color components have 9 bits of precision and all three share a single exponent.
- **FORMAT_DXT1 = 17** — The S3TC texture format that uses Block Compression 1, and is the smallest variation of S3TC, only providing 1 bit of alpha and color data being premultiplied with alpha. More information can be found at https://www.khronos.org/opengl/wiki/S3_Texture_Compression.
- **FORMAT_DXT3 = 18** — The S3TC texture format that uses Block Compression 2, and color data is interpreted as not having been premultiplied by alpha. Well suited for images with sharp alpha transitions between translucent and opaque areas.
- **FORMAT_DXT5 = 19** — The S3TC texture format also known as Block Compression 3 or BC3 that contains 64 bits of alpha channel data followed by 64 bits of DXT1-encoded color data. Color data is not premultiplied by alpha, same as DXT3. DXT5 generally produces superior results for transparency gradients than DXT3.
- **FORMAT_RGTC_R = 20** — Texture format that uses Red Green Texture Compression, normalizing the red channel data using the same compression algorithm that DXT5 uses for the alpha channel. More information can be found here https://www.khronos.org/opengl/wiki/Red_Green_Texture_Compression.
- **FORMAT_RGTC_RG = 21** — Texture format that uses Red Green Texture Compression, normalizing the red and green channel data using the same compression algorithm that DXT5 uses for the alpha channel.
- **FORMAT_BPTC_RGBA = 22** — Texture format that uses BPTC compression with unsigned normalized RGBA components. More information can be found at https://www.khronos.org/opengl/wiki/BPTC_Texture_Compression.
- **FORMAT_BPTC_RGBF = 23** — Texture format that uses BPTC compression with signed floating-point RGB components.
- **FORMAT_BPTC_RGBFU = 24** — Texture format that uses BPTC compression with unsigned floating-point RGB components.
- **FORMAT_PVRTC2 = 25** — Texture format used on PowerVR-supported mobile platforms, uses 2 bit color depth with no alpha. More information on PVRTC can be found here https://en.wikipedia.org/wiki/PVRTC.
- **FORMAT_PVRTC2A = 26** — Same as PVRTC2, but with an alpha component.
- **FORMAT_PVRTC4 = 27** — Similar to PVRTC2, but with 4 bit color depth and no alpha.
- **FORMAT_PVRTC4A = 28** — Same as PVRTC4, but with an alpha component.
- **FORMAT_ETC = 29** — Ericsson Texture Compression format, also referred to as 'ETC1', and is part of the OpenGL ES graphics standard. An overview of the format is given at https://en.wikipedia.org/wiki/Ericsson_Texture_Compression#ETC1.
- **FORMAT_ETC2_R11 = 30** — Ericsson Texture Compression format 2 variant R11_EAC, which provides one channel of unsigned data.
- **FORMAT_ETC2_R11S = 31** — Ericsson Texture Compression format 2 variant SIGNED_R11_EAC, which provides one channel of signed data.

- **FORMAT_ETC2_RG11 = 32** — Ericsson Texture Compression format 2 variant RG11_EAC, which provides two channels of unsigned data.
- **FORMAT_ETC2_RG11S = 33** — Ericsson Texture Compression format 2 variant SIGNED_RG11_EAC, which provides two channels of signed data.
- **FORMAT_ETC2_RGB8 = 34** — Ericsson Texture Compression format 2 variant RGB8, which is a followup of ETC1 and compresses RGB888 data.
- **FORMAT_ETC2_RGBA8 = 35** — Ericsson Texture Compression format 2 variant RGBA8, which compresses RGBA8888 data with full alpha support.
- **FORMAT_ETC2_RGB8A1 = 36** — Ericsson Texture Compression format 2 variant RGB8_PUNCHTHROUGH_ALPHA1, which compresses RGBA data to make alpha either fully transparent or fully opaque.
- **FORMAT_MAX = 37**

29.186.5 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.

29.186.6 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 `dest`.

- 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 **bumpmap_to_normalmap** (*float* bump_scale=1.0)
- void **clear_mipmaps** ( )

Removes the image's mipmaps.

- *Error* **compress** (*CompressMode* mode, *CompressSource* 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** (*Format* 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, *Format* 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, *Format* 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.

- *Error* **decompress** ( )

Decompresses the image if it is compressed. Returns an error if decompress function is not available.

- *AlphaMode* **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.

- *Error* **generate_mipmaps** (*bool* renormalize=false)

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.

- *Format* **get_format** ( ) const

Returns the image's format. See `FORMAT_*` constants.

- *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)`.

- *Color* **get_pixelv** (*Vector2* src) const
- *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.

- *Error* **load** (*String* path)

Loads an image from file `path`.

- *Error* **load_jpg_from_buffer** (*PoolByteArray* buffer)

Loads an image from the binary contents of a JPEG file.

- *Error* **load_png_from_buffer** (*PoolByteArray* buffer)

Loads an image from the binary contents of a PNG file.

- 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, *Interpolation* interpolation=1)

Resizes the image to the given `width` and `height`. New pixels are calculated using `interpolation`. See `interpolation` constants.

- void **resize_to_po2** (*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.

- *Error* **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 **set_pixelv** (*Vector2* dst, *Color* color)
- 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.

29.187 ImageTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.187.1 Brief Description

A *Texture* based on an *Image*.

29.187.2 Member Functions

void	*create* (*int* width, *int* height, *Format* format, *int* flags=7)
void	*create_from_image* (*Image* image, *int* flags=7)
Format	*get_format* ( ) const
Error	*load* (*String* path)
void	*set_data* (*Image* image)
void	*set_size_override* (*Vector2* size)

29.187.3 Member Variables

- *float* **lossy_quality** - The storage quality for `ImageTexture`.STORAGE_COMPRESS_LOSSY.
- *Storage* **storage** - The storage type (raw, lossy, or compressed).

29.187.4 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.

29.187.5 Description

A *Texture* based on an *Image*. Can be created from an *Image* with *create_from_image*.

29.187.6 Member Function Description

- void **create** (*int* width, *int* height, *Format* 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_*.

- *Format* **get_format** ( ) const

Return the format of the `ImageTexture`, one of *Image*.FORMAT_*.

- *Error* **load** (*String* path)

Load an `ImageTexture` from a file path.

- void **set_data** (*Image* image)

Set the *Image* of this `ImageTexture`.

- void **set_size_override** (*Vector2* size)

Resizes the `ImageTexture` to the specified dimensions.

29.188 ImmediateGeometry

Inherits: *GeometryInstance* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.188.1 Brief Description

Draws simple geometry from code.

29.188.2 Member Functions

void	*add_sphere* (*int* lats, *int* lons, *float* radius, *bool* add_uv=true)
void	*add_vertex* (*Vector3* position)
void	*begin* (*PrimitiveType* primitive, *Texture* texture=null)
void	*clear* ( )
void	*end* ( )
void	*set_color* (*Color* color)
void	*set_normal* (*Vector3* normal)
void	*set_tangent* (*Plane* tangent)
void	*set_uv* (*Vector2* uv)
void	*set_uv2* (*Vector2* uv)

29.188.3 Description

Draws simple geometry from code. Uses a drawing mode similar to OpenGL 1.x.

29.188.4 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** (*PrimitiveType* 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.

29.189 Input

Inherits: *Object*

Inherited By: *InputDefault*

Category: Core

29.189.1 Brief Description

A Singleton that deals with inputs.

29.189.2 Member Functions

void	*action_press* (*String* action)
void	*action_release* (*String* action)
void	*add_joy_mapping* (*String* mapping, *bool* update_existing=false)
Vector3	*get_accelerometer* ( ) const
float	*get_action_strength* (*String* action) const
Array	*get_connected_joypads* ( )
Vector3	*get_gravity* ( ) const
Vector3	*get_gyroscope* ( ) const
float	*get_joy_axis* (*int* device, *int* axis) const
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
String	*get_joy_name* (*int* device)
float	*get_joy_vibration_duration* (*int* device)
Vector2	*get_joy_vibration_strength* (*int* device)
Vector2	*get_last_mouse_speed* ( ) const
Vector3	*get_magnetometer* ( ) const
int	*get_mouse_button_mask* ( ) const
MouseMode	*get_mouse_mode* ( ) const
bool	*is_action_just_pressed* (*String* action) const
bool	*is_action_just_released* (*String* action) const
bool	*is_action_pressed* (*String* action) const
bool	*is_joy_button_pressed* (*int* device, *int* button) const
bool	*is_joy_known* (*int* device)
bool	*is_key_pressed* (*int* scancode) const
bool	*is_mouse_button_pressed* (*int* button) const

Continued on next page

Table 11 – continued from previous page

void	joy_connection_changed (*int* device, *bool* connected, *String* name, *String* guid)
void	*parse_input_event* (*InputEvent* event)
void	*remove_joy_mapping* (*String* guid)
void	*set_custom_mouse_cursor* (*Resource* image, *CursorShape* shape=0, *Vector2* hotspot=Vector2(0, 0))
void	*set_default_cursor_shape* (*CursorShape* shape=0)
void	*set_mouse_mode* (*MouseMode* mode)
void	*start_joy_vibration* (*int* device, *float* weak_magnitude, *float* strong_magnitude, *float* duration=0)
void	*stop_joy_vibration* (*int* device)
void	*warp_mouse_position* (*Vector2* to)

29.189.3 Signals

- **joy_connection_changed** (*int* device, *bool* connected)

Emitted when a joypad device has been connected or disconnected.

29.189.4 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** — Makes the mouse cursor visible but confines it to the game window.

enum **CursorShape**

- **CURSOR_ARROW = 0** — Arrow cursor. Standard, default pointing cursor.
- **CURSOR_IBEAM = 1** — I-beam cursor. Usually used to show where the text cursor will appear when the mouse is clicked.
- **CURSOR_POINTING_HAND = 2** — Pointing hand cursor. Usually used to indicate the pointer is over a link or other interactable item.
- **CURSOR_CROSS = 3** — Cross cursor. Typically appears over regions in which a drawing operation can be performance or for selections.
- **CURSOR_WAIT = 4** — Wait cursor. Indicates that the application is busy performing an operation.
- **CURSOR_BUSY = 5** — Busy cursor. See `CURSOR_WAIT`.
- **CURSOR_DRAG = 6** — Drag cursor. Usually displayed when dragging something.
- **CURSOR_CAN_DROP = 7** — Can drop cursor. Usually displayed when dragging something to indicate that it can be dropped at the current position.
- **CURSOR_FORBIDDEN = 8** — Forbidden cursor. Indicates that the current action is forbidden (for example, when dragging something) or that the control at a position is disabled.
- **CURSOR_VSIZE = 9** — Vertical resize mouse cursor. A double headed vertical arrow. It tells the user they can resize the window or the panel vertically.

- **CURSOR_HSIZE = 10** — Horizontal resize mouse cursor. A double headed horizontal arrow. It tells the user they can resize the window or the panel horizontally.
- **CURSOR_BDIAGSIZE = 11** — Window resize mouse cursor. 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** — Window resize mouse cursor. 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** — Move cursor. Indicates that something can be moved.
- **CURSOR_VSPLIT = 14** — Vertical split mouse cursor. On Windows, it's the same as `CURSOR_VSIZE`.
- **CURSOR_HSPLIT = 15** — Horizontal split mouse cursor. On Windows, it's the same as `CURSOR_HSIZE`.
- **CURSOR_HELP = 16** — Help cursor. Usually a question mark.

29.189.5 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*.

29.189.6 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 acceleration. Otherwise, it returns an empty *Vector3*.

- *float* **get_action_strength** (*String* action) const
- *Array* **get_connected_joypads** ( )

Returns an *Array* containing the device IDs of all currently connected joypads.

- *Vector3* **get_gravity** ( ) const

If the device has an accelerometer, this will return the gravity. Otherwise, it returns an empty *Vector3*.

- *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. Otherwise, it returns an empty *Vector3*.

- *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.

- *MouseMode* **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 the user starts pressing the action event, meaning it's true only on the frame that the user pressed down the button.

This is useful for code that needs to run only once when an action is pressed, instead of every frame while it's pressed.

- *bool* **is_action_just_released** (*String* action) const

Returns `true` when the user stops pressing the action event, meaning it's true only on the frame that the user released the button.

- *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)

Feeds an *InputEvent* to the game. Can be used to artificially trigger input events from code.

- 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, *CursorShape* 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. Passing `null` to the image parameter resets to the system cursor. See enum `CURSOR_*` for the list of shapes.

- void **set_default_cursor_shape** (*CursorShape* shape=0)
- void **set_mouse_mode** (*MouseMode* 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.

29.190 InputDefault

Inherits: *Input* < *Object*

Category: Core

29.190.1 Brief Description

Default implementation of the *Input* class.

29.190.2 Description

Default implementation of the *Input* class, used internally by the editor and games for default input management.

29.191 InputEvent

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *InputEventScreenTouch*, *InputEventWithModifiers*, *InputEventScreenDrag*, *InputEventJoypadMotion*, *InputEventJoypadButton*, *InputEventAction*

Category: Core

29.191.1 Brief Description

Generic input event

29.191.2 Member Functions

String	*as_text* ( ) const
float	*get_action_strength* (*String* action) const
bool	*is_action* (*String* action) const
bool	*is_action_pressed* (*String* action) const
bool	*is_action_released* (*String* action) const
bool	*is_action_type* ( ) const
bool	*is_echo* ( ) const
bool	*is_pressed* ( ) const
bool	*shortcut_match* (*InputEvent* event) const
InputEvent	*xformed_by* (*Transform2D* xform, *Vector2* local_ofs=Vector2(0, 0)) const

29.191.3 Member Variables

- *int* **device** - The event's device ID.

29.191.4 Description

Base class of all sort of input event. See *Node._input*.

29.191.5 Member Function Description

- *String* **as_text** ( ) const

Returns a *String* representation of the event.

- *float* **get_action_strength** (*String* action) const
- *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

29.192 InputEventAction

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.192.1 Brief Description

Input event type for actions.

29.192.2 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.

29.192.3 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*.

29.193 InputEventGesture

Inherits: *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Inherited By: *InputEventPanGesture*, *InputEventMagnifyGesture*

Category: Core

29.193.1 Brief Description

29.193.2 Member Variables

- *Vector2* **position**

29.194 InputEventJoypadButton

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.194.1 Brief Description

Input event for gamepad buttons.

29.194.2 Member Variables

- *int* **button_index** - Button identifier. One of the `JOY_BUTTON_*` constants from *@GlobalScope*.
- *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`.

29.194.3 Description

Input event type for gamepad buttons. For joysticks see *InputEventJoypadMotion*.

29.195 InputEventJoypadMotion

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.195.1 Brief Description

Input event type for gamepad joysticks and other motions. For buttons see `InputEventJoypadButton`.

29.195.2 Member Variables

- *int* **axis** - Axis identifier. Use one of the `JOY_AXIS_*` constants in *@GlobalScope*.

- *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.

29.195.3 Description

Stores information about joystick motions. One `InputEventJoypadMotion` represents one axis at a time.

29.196 InputEventKey

Inherits: *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.196.1 Brief Description

Input event type for keyboard events.

29.196.2 Member Functions

int	*get_scancode_with_modifiers* ( ) const

29.196.3 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 *@GlobalScope*.
- *int* **unicode** - Key unicode identifier when relevant.

29.196.4 Description

Stores key presses on the keyboard. Supports key presses, key releases and *echo* events.

29.196.5 Member Function Description

- *int* **get_scancode_with_modifiers** ( ) const

29.197 InputEventMagnifyGesture

Inherits: *InputEventGesture* < *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.197.1 Brief Description

29.197.2 Member Variables

- *float* **factor**

29.198 InputEventMouse

Inherits: *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Inherited By: *InputEventMouseMotion*, *InputEventMouseButton*

Category: Core

29.198.1 Brief Description

Base input event type for mouse events.

29.198.2 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.

29.198.3 Description

Stores general mouse events information.

29.199 InputEventMouseButton

Inherits: *InputEventMouse* < *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.199.1 Brief Description

Input event type for mouse button events.

29.199.2 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.

29.199.3 Description

Contains mouse click information. See *Node._input*.

29.200 InputEventMouseMotion

Inherits: *InputEventMouse* < *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.200.1 Brief Description

Input event type for mouse motion events.

29.200.2 Member Variables

- *Vector2* **relative** - Mouse position relative to the previous position (position at the last frame).
- *Vector2* **speed** - Mouse speed.

29.200.3 Description

Contains mouse motion information. Supports relative, absolute positions and speed. See *Node._input*.

29.201 InputEventPanGesture

Inherits: *InputEventGesture* < *InputEventWithModifiers* < *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.201.1 Brief Description

29.201.2 Member Variables

- *Vector2* **delta**

29.202 InputEventScreenDrag

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.202.1 Brief Description

Input event type for screen drag events.

(only available on mobile devices)

29.202.2 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.

29.202.3 Description

Contains screen drag information. See *Node._input*.

29.203 InputEventScreenTouch

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Category: Core

29.203.1 Brief Description

Input event type for screen touch events.

(only available on mobile devices)

29.203.2 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.

29.203.3 Description

Stores multi-touch press/release information. Supports touch press, touch release and *index* for multi-touch count and order.

29.204 InputEventWithModifiers

Inherits: *InputEvent* < *Resource* < *Reference* < *Object*

Inherited By: *InputEventKey*, *InputEventMouse*, *InputEventGesture*

Category: Core

29.204.1 Brief Description

Base class for keys events with modifiers.

29.204.2 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.

29.204.3 Description

Contains keys events information with modifiers support like `SHIFT` or `ALT`. See *Node._input*.

29.205 InputMap

Inherits: *Object*

Category: Core

29.205.1 Brief Description

Singleton that manages *InputEventAction*.

29.205.2 Member Functions

void	*action_add_event* (*String* action, *InputEvent* event)
void	*action_erase_event* (*String* action, *InputEvent* event)
void	*action_erase_events* (*String* action)
bool	*action_has_event* (*String* action, *InputEvent* event)
void	*action_set_deadzone* (*String* deadzone, *float* arg1)
void	*add_action* (*String* action, *float* deadzone=0.5)
void	*erase_action* (*String* action)
bool	*event_is_action* (*InputEvent* event, *String* action) const
Array	*get_action_list* (*String* action)
Array	*get_actions* ( )
bool	*has_action* (*String* action) const
void	*load_from_globals* ( )

29.205.3 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*.

29.205.4 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.

- void **action_erase_events** (*String* action)
- *bool* **action_has_event** (*String* action, *InputEvent* event)

Returns true if an action has an *InputEvent* associated with it.

- void **action_set_deadzone** (*String* deadzone, *float* arg1)
- void **add_action** (*String* action, *float* deadzone=0.5)

Adds an (empty) action to the `InputMap`, with a configurable `deadzone`.

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 *InputEvent*s 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*.

29.206 InstancePlaceholder

Inherits: *Node* < *Object*

Category: Core

29.206.1 Brief Description

Placeholder for the root *Node* of a *PackedScene*.

29.206.2 Member Functions

Node	*create_instance* (*bool* replace=false, *PackedScene* custom_scene=null)
String	*get_instance_path* ( ) const
Dictionary	*get_stored_values* (*bool* with_order=false)
void	*replace_by_instance* (*PackedScene* custom_scene=null)

29.206.3 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.

29.206.4 Member Function Description

- *Node* **create_instance** (*bool* replace=false, *PackedScene* custom_scene=null)
- *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.

29.207 int

Category: Built-In Types

29.207.1 Brief Description

Integer built-in type.

29.207.2 Member Functions

int	int (*bool* from)
int	int (*float* from)
int	int (*String* from)

29.207.3 Description

Integer built-in type.

29.207.4 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.

29.208 InterpolatedCamera

Inherits: *Camera* < *Spatial* < *Node* < *Object*

Category: Core

29.208.1 Brief Description

Camera which moves toward another node.

29.208.2 Member Functions

void	*set_target* (*Object* target)

29.208.3 Member Variables

- *bool* **enabled** - If `true` and a target is set, the camera will move automatically.
- *float* **speed** - How quickly the camera moves toward its target. Higher values will result in tighter camera motion.
- *NodePath* **target** - The target's *NodePath*.

29.208.4 Description

InterpolatedCamera is a *Camera* which smoothly moves to match a target node's position and rotation.

If it is not *enabled* or does not have a valid target set, InterpolatedCamera acts like a normal Camera.

29.208.5 Member Function Description

- void **set_target** (*Object* target)

Sets the node to move toward and orient with.

29.209 IP

Inherits: *Object*

Inherited By: *IP_Unix*

Category: Core

29.209.1 Brief Description

Internet protocol (IP) support functions like DNS resolution.

29.209.2 Member Functions

void	*clear_cache* (*String* hostname="")
void	*erase_resolve_item* (*int* id)
Array	*get_local_addresses* ( ) const
String	*get_resolve_item_address* (*int* id) const
ResolverStatus	*get_resolve_item_status* (*int* id) const
String	*resolve_hostname* (*String* host, *Type* ip_type=3)
int	*resolve_hostname_queue_item* (*String* host, *Type* ip_type=3)

29.209.3 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.

29.209.4 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.

29.209.5 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.

29.209.6 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*).

- *ResolverStatus* **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, *Type* 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, *Type* 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.

29.210 IP_Unix

Inherits: *IP* < *Object*

Category: Core

29.210.1 Brief Description

Unix IP support. See *IP*.

29.210.2 Description

Unix-specific implementation of IP support functions. See *IP*.

29.211 ItemList

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.211.1 Brief Description

Control that provides a list of selectable items (and/or icons) in a single column, or optionally in multiple columns.

29.211.2 Member Functions

void	*add_icon_item* (*Texture* icon, *bool* selectable=true)
void	*add_item* (*String* text, *Texture* icon=null, *bool* selectable=true)
void	*clear* ( )
void	*ensure_current_is_visible* ( )
int	*get_item_at_position* (*Vector2* position, *bool* exact=false) const
int	*get_item_count* ( ) const
Color	*get_item_custom_bg_color* (*int* idx) const
Texture	*get_item_icon* (*int* idx) const
Color	*get_item_icon_modulate* (*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
String	*get_item_tooltip* (*int* idx) const

Continued on next page

Table 12 – continued from previous page

PoolIntArray	*get_selected_items* ( )
VScrollBar	*get_v_scroll* ( )
bool	*is_anything_selected* ( )
bool	*is_item_disabled* (*int* idx) const
bool	*is_item_selectable* (*int* idx) const
bool	*is_item_tooltip_enabled* (*int* idx) const
bool	*is_selected* (*int* idx) const
void	*move_item* (*int* from_idx, *int* to_idx)
void	*remove_item* (*int* idx)
void	*select* (*int* idx, *bool* single=true)
void	*set_item_custom_bg_color* (*int* idx, *Color* custom_bg_color)
void	*set_item_disabled* (*int* idx, *bool* disabled)
void	*set_item_icon* (*int* idx, *Texture* icon)
void	*set_item_icon_modulate* (*int* idx, *Color* modulate)
void	*set_item_icon_region* (*int* idx, *Rect2* rect)
void	*set_item_metadata* (*int* idx, *Variant* metadata)
void	*set_item_selectable* (*int* idx, *bool* selectable)
void	*set_item_text* (*int* idx, *String* text)
void	*set_item_tooltip* (*int* idx, *String* tooltip)
void	*set_item_tooltip_enabled* (*int* idx, *bool* enable)
void	*sort_items_by_text* ( )
void	*unselect* (*int* idx)
void	*unselect_all* ( )

29.211.3 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)

29.211.4 Member Variables

- *bool* **allow_reselect** - If `true` the currently selected item may be selected again.
- *bool* **allow_rmb_select** - If `true` a right mouse button click can select items.

- *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** - Allow single or multiple selection. See the `SELECT_*` constants.

29.211.5 Enums

enum **IconMode**

- **ICON_MODE_TOP = 0**
- **ICON_MODE_LEFT = 1**

enum **SelectMode**

- **SELECT_SINGLE = 0**
- **SELECT_MULTI = 1**

29.211.6 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).

29.211.7 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
- *Color* **get_item_icon_modulate** (*int* idx) const

Returns a *Color* modulating item's icon at the specified index.

- *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_anything_selected** ( )

Returns `true` if one or more items are selected.

- *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 **move_item** (*int* from_idx, *int* to_idx)

Moves item at index `from_idx` to `to_idx`.

- 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_modulate** (*int* idx, *Color* modulate)

Sets a modulating *Color* for item's icon 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.

- void **unselect_all** ( )

Ensure there are no items selected.

29.212 JavaScript

Inherits: *Object*

Category: Core

29.212.1 Brief Description

Singleton that connects the engine with the browser's JavaScript context in HTML5 export.

29.212.2 Member Functions

Variant	*eval* (*String* code, *bool* use_global_execution_context=false)

29.212.3 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.

29.212.4 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.

29.213 Joint

Inherits: *Spatial* < *Node* < *Object*

Inherited By: *ConeTwistJoint*, *Generic6DOFJoint*, *SliderJoint*, *HingeJoint*, *PinJoint*

Category: Core

29.213.1 Brief Description

Base class for all 3D joints

29.213.2 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.

29.213.3 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

29.214 Joint2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Inherited By: *PinJoint2D*, *DampedSpringJoint2D*, *GrooveJoint2D*

Category: Core

29.214.1 Brief Description

Base node for all joint constraints in 2D physics.

29.214.2 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*.

29.214.3 Description

Base node for all joint constraints in 2D physics. Joints take 2 bodies and apply a custom constraint.

29.215 JSON

Inherits: *Object*

Category: Core

29.215.1 Brief Description

Helper class for parsing JSON data.

29.215.2 Member Functions

JSONParseResult	*parse* (*String* json)
String	*print* (*Variant* value, *String* indent="", *bool* sort_keys=false)

29.215.3 Description

Helper class for parsing JSON data. For usage example and other important hints, see *JSONParseResult*.

29.215.4 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.

29.216 JSONParseResult

Inherits: *Reference* < *Object*

Category: Core

29.216.1 Brief Description

Data class wrapper for decoded JSON.

29.216.2 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.

Note that JSON objects do not preserve key order like Godot dictionaries, thus you should not rely on keys being in a certain order if a dictionary is constructed from JSON. In contrast, JSON arrays retain the order of their elements:*

```
var p = JSON.parse('["hello", "world", "!"]')
if typeof(p.result) == TYPE_ARRAY:
    print(p.result[0]) # prints 'hello'
else:
    print("unexpected results")
```

29.216.3 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 == OK`.

29.217 KinematicBody

Inherits: *PhysicsBody* < *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

29.217.1 Brief Description

Kinematic body 3D node.

29.217.2 Member Functions

Vector3	*get_floor_velocity* ( ) const
KinematicCollision	*get_slide_collision* (*int* slide_idx)
int	*get_slide_count* ( ) const
bool	*is_on_ceiling* ( ) const
bool	*is_on_floor* ( ) const
bool	*is_on_wall* ( ) const
KinematicCollision	*move_and_collide* (*Vector3* rel_vec, *bool* infinite_inertia=true)
Vector3	*move_and_slide* (*Vector3* linear_velocity, *Vector3* floor_normal=Vector3(0, 0, 0), *bool* infinite_inertia=true, *float* slope_stop_min_velocity=0.05, *int* max_slides=4, *float* floor_max_angle=0.785398)
bool	*test_move* (*Transform* from, *Vector3* rel_vec, *bool* infinite_inertia)

29.217.3 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.

29.217.4 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.

29.217.5 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, *bool* infinite_inertia=true)

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), *bool* infinite_inertia=true, *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_slides` 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, *bool* infinite_inertia)

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.

29.218 KinematicBody2D

Inherits: *PhysicsBody2D* < *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.218.1 Brief Description

Kinematic body 2D node.

29.218.2 Member Functions

Vector2	*get_floor_velocity* ( ) const
KinematicCollision2D	*get_slide_collision* (*int* slide_idx)
int	*get_slide_count* ( ) const
bool	*is_on_ceiling* ( ) const
bool	*is_on_floor* ( ) const
bool	*is_on_wall* ( ) const
KinematicCollision2D	*move_and_collide* (*Vector2* rel_vec, *bool* infinite_inertia=true)
Vector2	*move_and_slide* (*Vector2* linear_velocity, *Vector2* floor_normal=Vector2(0, 0), *bool* infinite_inertia=true, *float* slope_stop_min_velocity=5, *int* max_bounces=4, *float* floor_max_angle=0.785398)
bool	*test_move* (*Transform2D* from, *Vector2* rel_vec, *bool* infinite_inertia)

29.218.3 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.

29.218.4 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.

29.218.5 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, *bool* infinite_inertia=true)

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), *bool* infinite_inertia=true, *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, *bool* infinite_inertia)

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.

29.219 KinematicCollision

Inherits: *Reference* < *Object*

Category: Core

29.219.1 Brief Description

Collision data for KinematicBody collisions.

29.219.2 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.

29.219.3 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.

29.220 KinematicCollision2D

Inherits: *Reference* **<** *Object*

Category: Core

29.220.1 Brief Description

Collision data for KinematicBody2D collisions.

29.220.2 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.

29.220.3 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.

29.221 Label

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.221.1 Brief Description

Displays plain text in a line or wrapped inside a rectangle. For formatted text, use *RichTextLabel*.

29.221.2 Member Functions

int	*get_line_count* ( ) const
int	*get_line_height* ( ) const
int	*get_total_character_count* ( ) const
int	*get_visible_line_count* ( ) const

29.221.3 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.

29.221.4 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.

29.221.5 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 *Control*s, Label's *Control.mouse_filter* defaults to MOUSE_FILTER_IGNORE (i.e. it doesn't react to mouse input events).

29.221.6 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.

29.222 LargeTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.222.1 Brief Description

A Texture capable of storing many smaller Textures with offsets.

29.222.2 Member Functions

int	*add_piece* (*Vector2* ofs, *Texture* texture)
void	*clear* ( )
int	*get_piece_count* ( ) const
Vector2	*get_piece_offset* (*int* idx) const
Texture	*get_piece_texture* (*int* idx) const
void	*set_piece_offset* (*int* idx, *Vector2* ofs)
void	*set_piece_texture* (*int* idx, *Texture* texture)
void	*set_size* (*Vector2* size)

29.222.3 Description

A Texture capable of storing many smaller Textures with offsets.

You can dynamically add pieces(*Texture*) to this `LargeTexture` using different offsets.

29.222.4 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 "ofs".

- void **set_size (*Vector2* size)**

Sets the size of this `LargeTexture`.

29.223 Light

Inherits: *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Inherited By: *SpotLight*, *OmniLight*, *DirectionalLight*

Category: Core

29.223.1 Brief Description

Provides a base class for different kinds of light nodes.

29.223.2 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**

29.223.3 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**

29.223.4 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.

29.224 Light2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.224.1 Brief Description

Casts light in a 2D environment.

29.224.2 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.

29.224.3 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 inversed 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*.

29.224.4 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.

29.225 LightOccluder2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.225.1 Brief Description

Occludes light cast by a Light2D, casting shadows.

29.225.2 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.

29.225.3 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.

29.226 Line2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.226.1 Brief Description

A 2D line.

29.226.2 Member Functions

void	*add_point* (*Vector2* position)
int	*get_point_count* ( ) const
Vector2	*get_point_position* (*int* i) const
void	*remove_point* (*int* i)
void	*set_point_position* (*int* i, *Vector2* position)

29.226.3 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.

29.226.4 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.

29.226.5 Description

A line through several points in 2D space.

29.226.6 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`.

29.227 LineEdit

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.227.1 Brief Description

Control that provides single line string editing.

29.227.2 Member Functions

void	*append_at_cursor* (*String* text)
void	*clear* ( )
void	*deselect* ( )
PopupMenu	*get_menu* ( ) const
void	*menu_option* (*int* option)
void	*select* (*int* from=0, *int* to=-1)
void	*select_all* ( )

29.227.3 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`.

29.227.4 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 replaced with the secret character (see *secret_character*).
- *String* **secret_character** - The character to use to mask secret input (defaults to "*"). Only a single character can be used as the secret character.
- *String* **text** - String value of the *LineEdit*.

29.227.5 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**

29.227.6 Description

LineEdit provides a single line string editor, used for text fields.

29.227.7 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*.

29.228 LineShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.228.1 Brief Description

Line shape for 2D collisions.

29.228.2 Member Variables

- *float* **d** - The line's distance from the origin.
- *Vector2* **normal** - The line's normal.

29.228.3 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.

29.229 LinkButton

Inherits: *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.229.1 Brief Description

Simple button used to represent a link to some resource.

29.229.2 Member Variables

- *String* **text**
- *UnderlineMode* **underline**

29.229.3 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.

29.229.4 Description

This kind of buttons are primarily used when the interaction with the button causes a context change (like linking to a web page).

29.230 Listener

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.230.1 Brief Description

29.230.2 Member Functions

void	*clear_current* ( )
Transform	*get_listener_transform* ( ) const
bool	*is_current* ( ) const
void	*make_current* ( )

29.230.3 Member Function Description

- void **clear_current** ( )
- *Transform* **get_listener_transform** ( ) const
- *bool* **is_current** ( ) const
- void **make_current** ( )

29.231 MainLoop

Inherits: *Object*

Inherited By: *SceneTree*

Category: Core

29.231.1 Brief Description

Main loop is the abstract main loop base class.

29.231.2 Member Functions

void	*_drop_files* (*PoolStringArray* files, *int* screen) virtual
void	*_finalize* ( ) virtual
void	*_idle* (*float* delta) virtual
void	*_initialize* ( ) virtual
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)

29.231.3 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**

29.231.4 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`.

29.231.5 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)

29.232 MarginContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.232.1 Brief Description

Simple margin container.

29.232.2 Description

Simple margin container. Adds a left margin to anything contained.

29.233 Marshalls

Inherits: *Reference* < *Object*

Category: Core

29.233.1 Brief Description

Data transformation (marshalling) and encoding helpers.

29.233.2 Member Functions

PoolByteArray	*base64_to_raw* (*String* base64_str)
String	*base64_to_utf8* (*String* base64_str)
Variant	*base64_to_variant* (*String* base64_str)
String	*raw_to_base64* (*PoolByteArray* array)
String	*utf8_to_base64* (*String* utf8_str)
String	*variant_to_base64* (*Variant* variant)

29.233.3 Description

Provides data transformation and encoding utility functions.

29.233.4 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*.

29.234 Material

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *ParticlesMaterial*, *ShaderMaterial*, *SpatialMaterial*, *CanvasItemMaterial*

Category: Core

29.234.1 Brief Description

Abstract base *Resource* for coloring and shading geometry.

29.234.2 Member Variables

- *Material* **next_pass**
- *int* **render_priority**

29.234.3 Numeric Constants

- **RENDER_PRIORITY_MAX = 127**
- **RENDER_PRIORITY_MIN = -128**

29.234.4 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.

29.235 MenuButton

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.235.1 Brief Description

Special button that brings up a *PopupMenu* when clicked.

29.235.2 Member Functions

PopupMenu	*get_popup* ( ) const
void	*set_disable_shortcuts* (*bool* disabled)

29.235.3 Signals

- **about_to_show** ( )

Emitted when *PopupMenu* of this MenuButton is about to show.

29.235.4 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.

29.235.5 Member Function Description

- *PopupMenu* **get_popup** ( ) const

Return the *PopupMenu* contained in this button.

- void **set_disable_shortcuts** (*bool* disabled)

29.236 Mesh

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *ArrayMesh*, *PrimitiveMesh*

Category: Core

29.236.1 Brief Description

A *Resource* that contains vertex-array based geometry.

29.236.2 Member Functions

Shape	*create_convex_shape* ( ) const
Mesh	*create_outline* (*float* margin) const
Shape	*create_trimesh_shape* ( ) const
TriangleMesh	*generate_triangle_mesh* ( ) const
PoolVector3Array	*get_faces* ( ) const

29.236.3 Member Variables

- *Vector2* **lightmap_size_hint**

29.236.4 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.

29.236.5 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.

29.236.6 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.

29.237 MeshDataTool

Inherits: *Reference* < *Object*

Category: Core

29.237.1 Brief Description

29.237.2 Member Functions

void	*clear* ( )
Error	*commit_to_surface* (*ArrayMesh* mesh)
Error	*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

Continued on next page

Table 13 – continued from previous page

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)

29.237.3 Member Function Description

- void **clear** ( )
- *Error* **commit_to_surface** (*ArrayMesh* mesh)
- *Error* **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)

29.238 MeshInstance

Inherits: *GeometryInstance* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.238.1 Brief Description

Node that instances meshes into a scenario.

29.238.2 Member Functions

void	*create_convex_collision* ( )
void	*create_debug_tangents* ( )
void	*create_trimesh_collision* ( )
Material	*get_surface_material* (*int* surface) const
void	*set_surface_material* (*int* surface, *Material* material)

29.238.3 Member Variables

- *Mesh* **mesh** - The *Mesh* resource for the instance.
- *NodePath* **skeleton** - *NodePath* to the *Skeleton* associated with the instance.

29.238.4 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.

29.238.5 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.

29.239 MeshInstance2D

Inherits: *Node2D* **<** *CanvasItem* **<** *Node* **<** *Object*

Category: Core

29.239.1 Brief Description

29.239.2 Member Variables

- *Mesh* **mesh**
- *Texture* **normal_map**
- *Texture* **texture**

29.240 MeshLibrary

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.240.1 Brief Description

Library of meshes.

29.240.2 Member Functions

void	*clear* ( )
void	*create_item* (*int* id)
int	*find_item_by_name* (*String* name) const
PoolIntArray	*get_item_list* ( ) const
Mesh	*get_item_mesh* (*int* id) const
String	*get_item_name* (*int* id) const
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
void	*remove_item* (*int* id)
void	*set_item_mesh* (*int* id, *Mesh* mesh)
void	*set_item_name* (*int* id, *String* name)
void	*set_item_navmesh* (*int* id, *NavigationMesh* navmesh)
void	*set_item_preview* (*int* id, *Texture* texture)
void	*set_item_shapes* (*int* id, *Array* shapes)

29.240.3 Description

Library of meshes. Contains a list of *Mesh* resources, each with name and ID. Useful for GridMap or painting Terrain.

29.240.4 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)

29.241 MobileVRInterface

Inherits: *ARVRInterface* < *Reference* < *Object*

Category: Core

29.241.1 Brief Description

Generic mobile VR implementation

29.241.2 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 then the screen can natively handle. A value between 1.5 and 2.0 often provides good results but at the cost of performance.

29.241.3 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.

29.242 MultiMesh

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.242.1 Brief Description

Provides high performance mesh instancing.

29.242.2 Member Functions

AABB	*get_aabb* ( ) const
Color	*get_instance_color* (*int* instance) const
Transform	*get_instance_transform* (*int* instance) const
void	*set_instance_color* (*int* instance, *Color* color)
void	*set_instance_transform* (*int* instance, *Transform* transform)

29.242.3 Member Variables

- *ColorFormat* **color_format**
- *int* **instance_count**
- *Mesh* **mesh**
- *TransformFormat* **transform_format**

29.242.4 Enums

enum **TransformFormat**

- **TRANSFORM_2D = 0**
- **TRANSFORM_3D = 1**

enum **ColorFormat**

- **COLOR_NONE = 0**
- **COLOR_8BIT = 1**

- **COLOR_FLOAT = 2**

29.242.5 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.

29.242.6 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.

29.243 MultiMeshInstance

Inherits: *GeometryInstance* **<** *VisualInstance* **<** *Spatial* **<** *Node* **<** *Object*

Category: Core

29.243.1 Brief Description

Node that instances a *MultiMesh*.

29.243.2 Member Variables

- *MultiMesh* **multimesh** - The *MultiMesh* resource that will be used and shared among all instances of the `MultiMeshInstance`.

29.243.3 Description

`MultiMeshInstance` is a specialized node to instance *GeometryInstance*s 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).

29.244 MultiplayerAPI

Inherits: *Reference* < *Object*

Category: Core

29.244.1 Brief Description

High Level Multiplayer API.

29.244.2 Member Functions

void	*clear* ( )
PoolIntArray	*get_network_connected_peers* ( ) const
int	*get_network_unique_id* ( ) const
int	*get_rpc_sender_id* ( ) const
bool	*has_network_peer* ( ) const
bool	*is_network_server* ( ) const
void	*poll* ( )
Error	*send_bytes* (*PoolByteArray* bytes, *int* id=0)
void	*set_root_node* (*Node* node)

29.244.3 Signals

- **connected_to_server** ( )

Emitted whenever this MultiplayerAPI's *network_peer* successfully connected to a server. Only emitted on clients.

- **connection_failed** ( )

Emitted whenever this MultiplayerAPI's *network_peer* fails to establish a connection to a server. Only emitted on clients.

- **network_peer_connected** (*int* id)

Emitted whenever this MultiplayerAPI's *network_peer* connects with a new peer. ID is the peer ID of the new peer. Clients get notified when other clients connect to the same server. Upon connecting to a server, a client also receives this signal for the server (with ID being 1).

- **network_peer_disconnected** (*int* id)

Emitted whenever this MultiplayerAPI's *network_peer* disconnects from a peer. Clients get notified when other clients disconnect from the same server.

- **network_peer_packet** (*int* id, *PoolByteArray* packet)

Emitted whenever this MultiplayerAPI's *network_peer* receive a `packet` with custom data (see *send_bytes*). ID is the peer ID of the peer that sent the packet.

- **server_disconnected ( )**

Emitted whenever this MultiplayerAPI's *network_peer* disconnects from server. Only emitted on clients.

29.244.4 Member Variables

- *NetworkedMultiplayerPeer* **network_peer** - The peer object to handle the RPC system (effectively enabling networking when set). Depending on the peer itself, the MultiplayerAPI 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 MultiplayerAPI's signals.
- *bool* **refuse_new_network_connections** - If `true` the MultiplayerAPI's *network_peer* refuses new incoming connections.

29.244.5 Enums

enum **RPCMode**

- **RPC_MODE_DISABLED = 0** — Used with *Node.rpc_config* or *Node.rset_config* to disable a method or property for all RPC calls, making it unavailable. Default for all methods.
- **RPC_MODE_REMOTE = 1** — Used with *Node.rpc_config* or *Node.rset_config* to set a method to be called or a property to be changed only on the remote end, not locally. Analogous to the `remote` keyword. Calls and property changes are accepted from all remote peers, no matter if they are node's master or slaves.
- **RPC_MODE_SYNC = 2** — Behave like `RPC_MODE_REMOTE` but also make the call or property change locally. Analogous to the `sync` keyword.
- **RPC_MODE_MASTER = 3** — Used with *Node.rpc_config* or *Node.rset_config* to set a method to be called or a property to be changed only on the network master for this node. Analogous to the `master` keyword. Only accepts calls or property changes from the node's network slaves, see *Node.set_network_master*.
- **RPC_MODE_SLAVE = 4** — Used with *Node.rpc_config* or *Node.rset_config* to set a method to be called or a property to be changed only on slaves for this node. Analogous to the `slave` keyword. Only accepts calls or property changes from the node's network master, see *Node.set_network_master*.
- **RPC_MODE_REMOTESYNC = 5** — Behave like `RPC_MODE_REMOTE` but also make the call or property change locally. Same as `RPC_MODE_SYNC` which is only kept for compatibility. Analogous to the `remotesync` keyword.
- **RPC_MODE_MASTERSYNC = 6** — Behave like `RPC_MODE_MASTER` but also make the call or property change locally. Analogous to the `mastersync` keyword.
- **RPC_MODE_SLAVESYNC = 7** — Behave like `RPC_MODE_SLAVE` but also make the call or property change locally. Analogous to the `slavesync` keyword.

29.244.6 Description

This class implements most of the logic behind the high level multiplayer API.

By default, *SceneTree* has a reference to this class that is used to provide multiplayer capabilities (i.e. RPC/RSET) across the whole scene.

It is possible to override the MultiplayerAPI instance used by specific Nodes by setting the *Node.custom_multiplayer* property, effectively allowing to run both client and server in the same scene.

29.244.7 Member Function Description

- void **clear** ( )

Clears the current MultiplayerAPI network state (you shouldn't call this unless you know what you are doing).

- *PoolIntArray* **get_network_connected_peers** ( ) const

Returns the peer IDs of all connected peers of this MultiplayerAPI's *network_peer*.

- *int* **get_network_unique_id** ( ) const

Returns the unique peer ID of this MultiplayerAPI's *network_peer*.

- *int* **get_rpc_sender_id** ( ) const

Returns the sender's peer ID for the RPC currently being executed.

NOTE: If not inside an RPC this method will return 0.

- *bool* **has_network_peer** ( ) const

Returns `true` if there is a *network_peer* set.

- *bool* **is_network_server** ( ) const

Returns `true` if this MultiplayerAPI's *network_peer* is in server mode (listening for connections).

- void **poll** ( )

Method used for polling the MultiplayerAPI. You only need to worry about this if you are using *Node.custom_multiplayer* override or you set *SceneTree.multiplayer_poll* to `false`. By default *SceneTree* will poll its MultiplayerAPI for you.

NOTE: This method results in RPCs and RSETs being called, so they will be executed in the same context of this function (e.g. `_process`, `physics`, *Thread*).

- *Error* **send_bytes** (*PoolByteArray* bytes, *int* id=0)

Sends the given raw `bytes` to a specific peer identified by `id` (see *NetworkedMultiplayerPeer.set_target_peer*). Default ID is `0`, i.e. broadcast to all peers.

- void **set_root_node** (*Node* node)

Sets the base root node to use for RPCs. Instead of an absolute path, a relative path will be used to find the node upon which the RPC should be executed.

This effectively allows to have different branches of the scene tree to be managed by different MultiplayerAPI, allowing for example to run both client and server in the same scene.

29.245 Mutex

Inherits: *Reference* < *Object*

Category: Core

29.245.1 Brief Description

A synchronization Mutex.

29.245.2 Member Functions

void	*lock* ( )
Error	*try_lock* ( )
void	*unlock* ( )

29.245.3 Description

A synchronization Mutex. Element used to synchronize multiple *Thread*s. 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.

29.245.4 Member Function Description

- void **lock** ( )

Lock this `Mutex`, blocks until it is unlocked by the current owner.

- *Error* **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.

29.246 NativeScript

Inherits: *Script* < *Resource* < *Reference* < *Object*

Category: Core

29.246.1 Brief Description

29.246.2 Member Functions

String	*get_class_documentation* ( ) const
String	*get_method_documentation* (*String* method) const
String	*get_property_documentation* (*String* path) const
String	*get_signal_documentation* (*String* signal_name) const
Object	*new* ( ) vararg

29.246.3 Member Variables

- *String* **class_name**
- *GDNativeLibrary* **library**

29.246.4 Member Function Description

- *String* **get_class_documentation** ( ) const

Returns the documentation string that was previously set with `godot_nativescript_set_class_documentation`.

- *String* **get_method_documentation** (*String* method) const

Returns the documentation string that was previously set with `godot_nativescript_set_method_documentation`.

- *String* **get_property_documentation** (*String* path) const

Returns the documentation string that was previously set with `godot_nativescript_set_property_documentation`.

- *String* **get_signal_documentation** (*String* signal_name) const

Returns the documentation string that was previously set with `godot_nativescript_set_signal_documentation`.

- *Object* **new** ( ) vararg

Constructs a new object of the base type with a script of this type already attached.

Note: Any arguments passed to this function will be ignored and not passed to the native constructor function. This will change with in a future API extension.

29.247 Navigation

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.247.1 Brief Description

Mesh-based navigation and pathfinding node.

29.247.2 Member Functions

Vector3	*get_closest_point* (*Vector3* to_point)
Vector3	*get_closest_point_normal* (*Vector3* to_point)
Object	*get_closest_point_owner* (*Vector3* to_point)
Vector3	*get_closest_point_to_segment* (*Vector3* start, *Vector3* end, *bool* use_collision=false)
PoolVector3Array	*get_simple_path* (*Vector3* start, *Vector3* end, *bool* optimize=true)
int	*navmesh_add* (*NavigationMesh* mesh, *Transform* xform, *Object* owner=null)
void	*navmesh_remove* (*int* id)
void	*navmesh_set_transform* (*int* id, *Transform* xform)

29.247.3 Member Variables

- *Vector3* **up_vector** - Defines which direction is up. By default this is `(0, 1, 0)`, which is the world up direction.

29.247.4 Description

Provides navigation and pathfinding within a collection of *NavigationMesh*es. By default these will be automatically collected from child *NavigationMeshInstance* nodes, but they can also be added on the fly with *navmesh_add*. In addition to basic pathfinding, this class also assists with aligning navigation agents with the meshes they are navigating on.

29.247.5 Member Function Description

- *Vector3* **get_closest_point** (*Vector3* to_point)

Returns the navigation point closest to the point given. Points are in local coordinate space.

- *Vector3* **get_closest_point_normal** (*Vector3* to_point)

Returns the surface normal at the navigation point closest to the point given. Useful for rotating a navigation agent according to the navigation mesh it moves on.

- *Object* **get_closest_point_owner** (*Vector3* to_point)

Returns the owner of the *NavigationMesh* which contains the navigation point closest to the point given. This is usually a NavigtionMeshInstance. For meshes added via *navmesh_add*, returns the owner that was given (or `null` if the `owner` parameter was omitted).

- *Vector3* **get_closest_point_to_segment** (*Vector3* start, *Vector3* end, *bool* use_collision=false)

Returns the navigation point closest to the given line segment. When enabling `use_collision`, only considers intersection points between segment and navigation meshes. If multiple intersection points are found, the one closest to the segment start point is returned.

- *PoolVector3Array* **get_simple_path** (*Vector3* start, *Vector3* end, *bool* optimize=true)

Returns the path between two given points. Points are in local coordinate space. If `optimize` is `true` (the default), the agent properties associated with each *NavigationMesh* (raidus, height, etc.) are considered in the path calculation, otherwise they are ignored.

- *int* **navmesh_add** (*NavigationMesh* mesh, *Transform* xform, *Object* owner=null)

Adds a *NavigationMesh*. Returns an ID for use with *navmesh_remove* or *navmesh_set_transform*. If given, a *Transform2D* is applied to the polygon. The optional `owner` is used as return value for *get_closest_point_owner*.

- void **navmesh_remove** (*int* id)

Removes the *NavigationMesh* with the given ID.

- void **navmesh_set_transform** (*int* id, *Transform* xform)

Sets the transform applied to the *NavigationMesh* with the given ID.

29.248 Navigation2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.248.1 Brief Description

2D navigation and pathfinding node.

29.248.2 Member Functions

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)

29.248.3 Description

Navigation2D provides navigation and pathfinding within a 2D area, specified as a collection of *NavigationPolygon* resources. By default these are automatically collected from child *NavigationPolygonInstance* nodes, but they can also be added on the fly with *navpoly_add*.

29.248.4 Member Function Description

- *Vector2* **get_closest_point** (*Vector2* to_point)

Returns the navigation point closest to the point given. Points are in local coordinate space.

- *Object* **get_closest_point_owner** (*Vector2* to_point)

Returns the owner of the *NavigationPolygon* which contains the navigation point closest to the point given. This is usually a NavigtionPolygonInstance. For polygons added via *navpoly_add*, returns the owner that was given (or `null` if the `owner` parameter was omitted).

- *PoolVector2Array* **get_simple_path** (*Vector2* start, *Vector2* end, *bool* optimize=true)

Returns the path between two given points. Points are in local coordinate space. If `optimize` is `true` (the default), the path is smoothed by merging path segments where possible.

- *int* **navpoly_add** (*NavigationPolygon* mesh, *Transform2D* xform, *Object* owner=null)

Adds a *NavigationPolygon*. Returns an ID for use with *navpoly_remove* or *navpoly_set_transform*. If given, a *Transform2D* is applied to the polygon. The optional `owner` is used as return value for *get_closest_point_owner*.

- void **navpoly_remove** (*int* id)

Removes the *NavigationPolygon* with the given ID.

- void **navpoly_set_transform** (*int* id, *Transform2D* xform)

Sets the transform applied to the *NavigationPolygon* with the given ID.

29.249 NavigationMesh

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.249.1 Brief Description

29.249.2 Member Functions

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)

29.249.3 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**

29.249.4 Numeric Constants

- **SAMPLE_PARTITION_WATERSHED = 0**
- **SAMPLE_PARTITION_MONOTONE = 1**
- **SAMPLE_PARTITION_LAYERS = 2**

29.249.5 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)

29.250 NavigationMeshInstance

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.250.1 Brief Description

29.250.2 Member Variables

- *bool* **enabled**
- *NavigationMesh* **navmesh**

29.251 NavigationPolygon

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.251.1 Brief Description

29.251.2 Member Functions

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)

29.251.3 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)

29.252 NavigationPolygonInstance

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.252.1 Brief Description

29.252.2 Member Variables

- *bool* **enabled**
- *NavigationPolygon* **navpoly**

29.253 NetworkedMultiplayerENet

Inherits: *NetworkedMultiplayerPeer* < *PacketPeer* < *Reference* < *Object*

Category: Core

29.253.1 Brief Description

PacketPeer implementation using the ENet library.

29.253.2 Member Functions

void	*close_connection* (*int* wait_usec=100)
Error	*create_client* (*String* address, *int* port, *int* in_bandwidth=0, *int* out_bandwidth=0, *int* client_port=0)
Error	*create_server* (*int* port, *int* max_clients=32, *int* in_bandwidth=0, *int* out_bandwidth=0)
void	*disconnect_peer* (*int* id, *bool* now=false)
int	*get_last_packet_channel* ( ) const
int	*get_packet_channel* ( ) const
String	*get_peer_address* (*int* id) const
int	*get_peer_port* (*int* id) const
void	*set_bind_ip* (*String* ip)

29.253.3 Member Variables

- *bool* **always_ordered** - Always use `TRANSFER_MODE_ORDERED` in place of `TRANSFER_MODE_UNRELIABLE`. This is the only way to use ordering with the RPC system.
- *int* **channel_count** - The number of channels to be used by ENet. Default: `3`. Channels are used to separate different kinds of data. In realiable or ordered mode, for example, the packet delivery order is ensured on a per channel basis.
- *CompressionMode* **compression_mode** - The compression method used for network packets. Default is no compression. These have different tradeoffs of compression speed versus bandwidth, you may need to test which one works best for your use case if you use compression at all.
- *int* **transfer_channel** - Set the default channel to be used to transfer data. By default this value is `-1` which means that ENet will only use 2 channels, one for reliable and one for unreliable packets. Channel `0` is reserved, and cannot be used. Setting this member to any value between `0` and *channel_count* (excluded) will force ENet to use that channel for sending data.

29.253.4 Enums

enum **CompressionMode**

- **COMPRESS_NONE = 0** — No compression.
- **COMPRESS_RANGE_CODER = 1** — ENet's buildin range encoding.
- **COMPRESS_FASTLZ = 2** — FastLZ compression.
- **COMPRESS_ZLIB = 3** — zlib compression.
- **COMPRESS_ZSTD = 4** — ZStandard compression.

29.253.5 Description

A PacketPeer implementation that should be passed to SceneTree.set_network_peer after being initialized as either a client or server. Events can then be handled by connecting to *SceneTree* signals.

29.253.6 Member Function Description

- void **close_connection** (*int* wait_usec=100)

Closes the connection. Ignored if no connection is currently established. If this is a server it tries to notify all clients before forcibly disconnecting them. If this is a client it simply closes the connection to the server.

- *Error* **create_client** (*String* address, *int* port, *int* in_bandwidth=0, *int* out_bandwidth=0, *int* client_port=0)

Create client that connects to a server at `address` using specified `port`. The given address needs to be either a fully qualified domain nome (e.g. `www.example.com`) or an IP address in IPv4 or IPv6 format (e.g. `192.168.1.1`). The `port` is the port the server is listening on. The `in_bandwidth` and `out_bandwidth` parameters can be used to limit the incoming and outgoing bandwidth to the given number of bytes per second. The default of 0 means unlimited bandwidth. Note that ENet will strategically drop packets on specific sides of a connection between peers to ensure the peer's bandwidth is not overwhelmed. The bandwidth parameters also determine the window size of a connection which limits the amount of reliable packets that may be in transit at any given time. Returns `OK` if a client was created, `ERR_ALREADY_IN_USE` if this NetworkedMultiplayerEnet instance already has an open connection (in which case you need to call *close_connection* first) or `ERR_CANT_CREATE` if the client could not be created. If `client_port` is specified, the client will also listen to the given port, this is useful in some NAT traveral technique.

- *Error* **create_server** (*int* port, *int* max_clients=32, *int* in_bandwidth=0, *int* out_bandwidth=0)

Create server that listens to connections via `port`. The port needs to be an available, unused port between 0 and 65535. Note that ports below 1024 are privileged and may require elevated permissions depending on the platform. To change the interface the server listens on, use *set_bind_ip*. The default IP is the wildcard `*`, which listens on all available interfaces. `max_clients` is the maximum number of clients that are allowed at once, any number up to 4096 may be used, although the achievable number of simultaneous clients may be far lower and depends on the application. For additional details on the bandwidth parameters, see *create_client*. Returns `OK` if a server was created, `ERR_ALREADY_IN_USE` if this NetworkedMultiplayerEnet instance already has an open connection (in which case you need to call *close_connection* first) or `ERR_CANT_CREATE` if the server could not be created.

- void **disconnect_peer** (*int* id, *bool* now=false)

Disconnect the given peer. If "now" is set to true, the connection will be closed immediately without flushing queued messages.

- *int* **get_last_packet_channel** ( ) const

Returns the channel of the last packet fetched via *PacketPeer.get_packet*

- *int* **get_packet_channel** ( ) const

Returns the channel of the next packet that will be retrieved via PacketPeer.get_packet_peer

- *String* **get_peer_address** (*int* id) const

Returns the IP address of the given peer.

- *int* **get_peer_port** (*int* id) const

Returns the remote port of the given peer.

- void **set_bind_ip** (*String* ip)

The IP used when creating a server. This is set to the wildcard `*` by default, which binds to all available interfaces. The given IP needs to be in IPv4 or IPv6 address format, for example: `192.168.1.1`.

29.254 NetworkedMultiplayerPeer

Inherits: *PacketPeer* < *Reference* < *Object*

Inherited By: *NetworkedMultiplayerENet*, *WebSocketMultiplayerPeer*

Category: Core

29.254.1 Brief Description

A high-level network interface to simplify multiplayer interactions.

29.254.2 Member Functions

ConnectionStatus	*get_connection_status* ( ) const
int	*get_packet_peer* ( ) const
int	*get_unique_id* ( ) const
void	*poll* ( )
void	*set_target_peer* (*int* id)

29.254.3 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.

29.254.4 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 TransferMode.

29.254.5 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.

29.254.6 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.

29.254.7 Description

Manages the connection to network peers. Assigns unique IDs to each client connected to the server.

29.254.8 Member Function Description

- *ConnectionStatus* **get_connection_status ( )** const

Returns the current state of the connection. See 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)**

Sets the peer to which packets will be sent.

The `id` can be one of: `TARGET_PEER_BROADCAST` to send to all connected peers, `TARGET_PEER_SERVER` to send to the peer acting as server, a valid peer ID to send to that specific peer, a negative peer ID to send to all peers except that one. Default: `TARGET_PEER_BROADCAST`

29.255 Nil

Category: Built-In Types

29.255.1 Brief Description

29.255.2 Member Functions

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)

29.255.3 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)

29.256 NinePatchRect

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.256.1 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.

29.256.2 Signals

- **texture_changed** ( )

Fired when the node's texture changes.

29.256.3 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.

29.256.4 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.

29.256.5 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.

29.257 Node

Inherits: *Object*

Inherited By: *Viewport*, *AudioStreamPlayer*, *EditorFileSystem*, *CanvasLayer*, *Spatial*, *AnimationPlayer*, *EditorPlugin*, *ResourcePreloader*, *AnimationTreePlayer*, *WorldEnvironment*, *InstancePlaceholder*, *HTTPRequest*, *EditorInterface*, *EditorResourcePreview*, *CanvasItem*, *Timer*, *Tween*

Category: Core

29.257.1 Brief Description

Base class for all *scene* objects.

29.257.2 Member Functions

void	*_enter_tree* ( ) virtual
void	*_exit_tree* ( ) virtual
void	*_input* (*InputEvent* event) virtual
void	*_physics_process* (*float* delta) virtual
void	*_process* (*float* delta) virtual
void	*_ready* ( ) virtual
void	*_unhandled_input* (*InputEvent* event) virtual
void	*_unhandled_key_input* (*InputEventKey* event) virtual
void	*add_child* (*Node* node, *bool* legible_unique_name=false)
void	*add_child_below_node* (*Node* node, *Node* child_node, *bool* legible_unique_name=false)
void	*add_to_group* (*String* group, *bool* persistent=false)
bool	*can_process* ( ) const
Node	*duplicate* (*int* flags=15) const
Node	*find_node* (*String* mask, *bool* recursive=true, *bool* owned=true) const
Node	*get_child* (*int* idx) const
int	*get_child_count* ( ) const
Array	*get_children* ( ) const
Array	*get_groups* ( ) const
int	*get_index* ( ) const
int	*get_network_master* ( ) const
Node	*get_node* (*NodePath* path) const
Array	*get_node_and_resource* (*NodePath* path)
Node	*get_parent* ( ) const
NodePath	*get_path* ( ) const
NodePath	*get_path_to* (*Node* node) const
float	*get_physics_process_delta_time* ( ) const
int	*get_position_in_parent* ( ) const
float	*get_process_delta_time* ( ) const
bool	*get_scene_instance_load_placeholder* ( ) const
SceneTree	*get_tree* ( ) const
Viewport	*get_viewport* ( ) const
bool	*has_node* (*NodePath* path) const
bool	*has_node_and_resource* (*NodePath* path) const
bool	*is_a_parent_of* (*Node* node) const
bool	*is_displayed_folded* ( ) const
bool	*is_greater_than* (*Node* node) const
bool	*is_in_group* (*String* group) const
bool	*is_inside_tree* ( ) const
bool	*is_network_master* ( ) const
bool	*is_physics_processing* ( ) const
bool	*is_physics_processing_internal* ( ) const
bool	*is_processing* ( ) const
bool	*is_processing_input* ( ) const
bool	*is_processing_internal* ( ) const

Continued on next page

Table 14 – continued from previous page

bool	*is_processing_unhandled_input* ( ) const
bool	*is_processing_unhandled_key_input* ( ) const
void	*move_child* (*Node* child_node, *int* to_position)
void	*print_stray_nodes* ( )
void	*print_tree* ( )
void	*print_tree_pretty* ( )
void	*propagate_call* (*String* method, *Array* args=[ ], *bool* parent_first=false)
void	*propagate_notification* (*int* what)
void	*queue_free* ( )
void	*raise* ( )
void	*remove_and_skip* ( )
void	*remove_child* (*Node* node)
void	*remove_from_group* (*String* group)
void	*replace_by* (*Node* node, *bool* keep_data=false)
void	*request_ready* ( )
Variant	*rpc* (*String* method) vararg
void	*rpc_config* (*String* method, *RPCMode* mode)
Variant	*rpc_id* (*int* peer_id, *String* method) vararg
Variant	*rpc_unreliable* (*String* method) vararg
Variant	*rpc_unreliable_id* (*int* peer_id, *String* method) vararg
void	*rset* (*String* property, *Variant* value)
void	*rset_config* (*String* property, *RPCMode* mode)
void	*rset_id* (*int* peer_id, *String* property, *Variant* value)
void	*rset_unreliable* (*String* property, *Variant* value)
void	*rset_unreliable_id* (*int* peer_id, *String* property, *Variant* value)
void	*set_display_folded* (*bool* fold)
void	*set_network_master* (*int* id, *bool* recursive=true)
void	*set_physics_process* (*bool* enable)
void	*set_physics_process_internal* (*bool* enable)
void	*set_process* (*bool* enable)
void	*set_process_input* (*bool* enable)
void	*set_process_internal* (*bool* enable)
void	*set_process_unhandled_input* (*bool* enable)
void	*set_process_unhandled_key_input* (*bool* enable)
void	*set_scene_instance_load_placeholder* (*bool* load_placeholder)

29.257.3 Signals

- **ready** ( )

Emitted when the node is ready.

- **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 (or a "destructor", if you will).

29.257.4 Member Variables

- *MultiplayerAPI* **custom_multiplayer** - The override to the default *MultiplayerAPI*. Set to null to use the default SceneTree one.
- *String* **filename** - When a scene is instanced from a file, its topmost node contains the filename from which it was loaded.
- *MultiplayerAPI* **multiplayer** - The *MultiplayerAPI* instance associated with this node. Either the *custom_multiplayer*, or the default SceneTree one (if inside tree).
- *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 *PackedScene*) all the nodes it owns will be saved with it. This allows for the creation of complex *SceneTree*s, with instancing and subinstancing.
- *PauseMode* **pause_mode** - Pause mode. How the node will behave if the *SceneTree* is paused.

29.257.5 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 unpaused.
- **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 received when translations may have changed. Can be triggered by the user changing the locale. Can be used to respond to language changes, for

example to change the UI strings on the fly. Useful when working with the built-in translation support, like *Object.tr*.

- **NOTIFICATION_INTERNAL_PROCESS = 25** — Notification received every frame when the internal process flag is set (see *set_process_internal*).
- **NOTIFICATION_INTERNAL_PHYSICS_PROCESS = 26** — Notification received every frame when the internal physics process flag is set (see *set_physics_process_internal*).

29.257.6 Enums

enum **PauseMode**

- **PAUSE_MODE_INHERIT = 0** — Inherits pause mode from the node's parent. For the root node, it is equivalent to PAUSE_MODE_STOP. Default.
- **PAUSE_MODE_STOP = 1** — Stop processing when the *SceneTree* is paused.
- **PAUSE_MODE_PROCESS = 2** — Continue to process regardless of the *SceneTree* pause state.

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.

29.257.7 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.

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.

Groups: Nodes can be added to as many groups as you want to be easy to manage, you could create groups like "enemies" or "collectables" for example, depending on your game. See *add_to_group*, *is_in_group* and *remove_from_group*. You can then retrieve all nodes in these groups, iterate them and even call methods on groups via the methods on *SceneTree*.

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). To identify which node receives the RPC call Godot will use its *NodePath* (make sure node names are the same on all peers). Also take a look at the high-level networking tutorial and corresponding demos.

29.257.8 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*.

To consume the input event and stop it propagating further to other nodes, *SceneTree.set_input_as_handled* can be called.

For gameplay input, *_unhandled_input* and *_unhandled_key_input* are usually a better fit as they allow the GUI to intercept the events first.

- 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*. See also the `onready` keyword for variables.

Usually used for initialization. For even earlier initialization, *Object._init* may be used. Also see *_enter_tree*.

- 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*.

To consume the input event and stop it propagating further to other nodes, *SceneTree.set_input_as_handled* can be called.

For gameplay input, this and *_unhandled_key_input* are usually a better fit than *_input* as they allow the GUI to intercept the events first.

- void **_unhandled_key_input** **(** *InputEventKey* event **)** virtual

Propagated to all nodes when the previous *InputEventKey* is not consumed by any nodes.

It is only called if unhandled key input processing is enabled, which is done automatically if this method is overridden, and can be toggled with *set_process_unhandled_key_input*.

To consume the input event and stop it propagating further to other nodes, *SceneTree.set_input_as_handled* can be called.

For gameplay input, this and *_unhandled_input* are usually a better fit than *_input* as they allow the GUI to intercept the events first.

- 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*). See notes in the description, and the group methods in *SceneTree*.

- *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. See *set_network_master*.

- *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 *NodePath* from this node to the specified `node`. 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

Returns `true` if this is an instance load placeholder. See *InstancePlaceholder*.

- *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. See notes in the description, and the group methods in *SceneTree*.

- *bool* **is_inside_tree** ( ) const

Returns `true` if this node is currently inside a *SceneTree*.

- *bool* **is_network_master** ( ) const

Returns `true` if the local system is the master of this node.

- *bool* **is_physics_processing** ( ) const

Returns `true` if physics processing is enabled (see *set_physics_process*).

- *bool* **is_physics_processing_internal** ( ) const

Returns `true` if internal physics processing is enabled (see *set_physics_process_internal*).

- *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

Returns `true` if internal processing is enabled (see *set_process_internal*).

- *bool* **is_processing_unhandled_input** ( ) const

Returns `true` if the node is processing unhandled input (see *set_process_unhandled_input*).

- *bool* **is_processing_unhandled_key_input** ( ) 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** ( )

Prints all stray nodes (nodes outside the *SceneTree*). Used for debugging. Works only in debug builds.

- void **print_tree** ( )

Prints the tree to stdout. Used mainly for debugging purposes. This version displays the path relative to the current node, and is good for copy/pasting into the *get_node* function. Example output:

```
TheGame
TheGame/Menu
TheGame/Menu/Label
TheGame/Menu/Camera2D
TheGame/SplashScreen
TheGame/SplashScreen/Camera2D
```

- void **print_tree_pretty** ( )

Similar to *print_tree*, this prints the tree to stdout. This version displays a more graphical representation similar to what is displayed in the scene inspector. It is useful for inspecting larger trees. Example output:

```
TheGame
   Menu
      Label
      Camera2D
   SplashScreen
      Camera2D
```

- 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. See notes in the description, and the group methods in *SceneTree*.

- 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 for the given `method` to peers on the network (and locally), optionally sending all additional arguments as arguments to the method called by the RPC. The call request will only be received by nodes with the same *NodePath*, including the exact same node name. Behaviour depends on the RPC configuration for the given method, see *rpc_config*. Methods are not exposed to RPCs by default. Also see *rset* and *rset_config* for properties. Returns an empty *Variant*. Note that you can only safely use RPCs on clients after you received the `connected_to_server` signal from the *SceneTree*. You also need to keep track of the connection state, either by the *SceneTree* signals like `server_disconnected` or by checking `SceneTree.network_peer.get_connection_status() == CONNECTION_CONNECTED`.

- void **rpc_config** (*String* method, *RPCMode* mode)

Changes the RPC mode for the given `method` to the given `mode`. See *RPCMode*. An alternative is annotating methods and properties with the corresponding keywords (`remote`, `sync`, `master`, `slave`). By default, methods are not exposed to networking (and RPCs). Also see *rset* and *rset_config* for properties.

- *Variant* **rpc_id** (*int* peer_id, *String* method) vararg

Sends a *rpc* to a specific peer identified by `peer_id` (see *NetworkedMultiplayerPeer.set_target_peer*). Returns an empty *Variant*.

- *Variant* **rpc_unreliable** (*String* method) vararg

Sends a *rpc* using an unreliable protocol. Returns an empty *Variant*.

- *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 (see *NetworkedMultiplayerPeer.set_target_peer*). Returns an empty *Variant*.

- void **rset** (*String* property, *Variant* value)

Remotely changes a property's value on other peers (and locally). Behaviour depends on the RPC configuration for the given property, see *rset_config*. Also see *rpc* for RPCs for methods, most information applies to this method as well.

- void **rset_config** (*String* property, *RPCMode* mode)

Changes the RPC mode for the given `property` to the given `mode`. See *RPCMode*. An alternative is annotating methods and properties with the corresponding keywords (`remote`, `sync`, `master`, `slave`). By default, properties are not exposed to networking (and RPCs). Also see *rpc* and *rpc_config* for methods.

- void **rset_id** (*int* peer_id, *String* property, *Variant* value)

Remotely changes the property's value on a specific peer identified by `peer_id` (see *NetworkedMultiplayerPeer.set_target_peer*).

- void **rset_unreliable** (*String* property, *Variant* value)

Remotely changes the 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 (see *NetworkedMultiplayerPeer.set_target_peer*).

- 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's network master to the peer with the given peer ID. The network master is the peer that has authority over the node on the network. Useful in conjunction with the `master` and `slave` keywords. Inherited from the parent node by default, which ultimately defaults to peer ID 1 (the server). If `recursive`, the given peer is recursively set as the master for all children of this node.

- 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)

Enables or disables internal physics for this node. Internal physics processing happens in isolation from the normal *_physics_process* calls and is used by some nodes internally to guarantee proper functioning even if the node is paused or physics processing is disabled for scripting (*set_physics_process*). Only useful for advanced uses to manipulate built-in nodes behaviour.

- 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)

Enables or disabled internal processing for this node. Internal processing happens in isolation from the normal *_process* calls and is used by some nodes internally to guarantee proper functioning even if the node is paused or processing is disabled for scripting (*set_process*). Only useful for advanced uses to manipulate built-in nodes behaviour.

- void **set_process_unhandled_input** (*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 **set_process_unhandled_key_input** (*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 **set_scene_instance_load_placeholder** (*bool* load_placeholder)

Sets whether this is an instance load placeholder. See *InstancePlaceholder*.

29.258 Node2D

Inherits: *CanvasItem* < *Node* < *Object*

Inherited By: *RemoteTransform2D*, *Joint2D*, *Skeleton2D*, *VisibilityNotifier2D*, *Navigation2D*, *CollisionPolygon2D*, *TouchScreenButton*, *Particles2D*, *AnimatedSprite*, *RayCast2D*, *Light2D*, *Path2D*, *Line2D*, *AudioStreamPlayer2D*, *CanvasModulate*, *Sprite*, *CollisionShape2D*, *NavigationPolygonInstance*, *PathFollow2D*, *ParallaxLayer*, *Polygon2D*, *Position2D*, *LightOccluder2D*, *CollisionObject2D*, *BackBufferCopy*, *MeshInstance2D*, *YSort*, *TileMap*, *Bone2D*, *Camera2D*

Category: Core

29.258.1 Brief Description

A 2D game object, parent of all 2D related nodes. Has a position, rotation, scale and Z-index.

29.258.2 Member Functions

void	*apply_scale* (*Vector2* ratio)
float	*get_angle_to* (*Vector2* point) const
Transform2D	*get_relative_transform_to_parent* (*Node* parent) const
void	*global_translate* (*Vector2* offset)
void	*look_at* (*Vector2* point)
void	*move_local_x* (*float* delta, *bool* scaled=false)
void	*move_local_y* (*float* delta, *bool* scaled=false)
void	*rotate* (*float* radians)
Vector2	*to_global* (*Vector2* local_point) const
Vector2	*to_local* (*Vector2* global_point) const
void	*translate* (*Vector2* offset)

29.258.3 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.

29.258.4 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.

29.258.5 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.

29.259 NodePath

Category: Built-In Types

29.259.1 Brief Description

Pre-parsed scene tree path.

29.259.2 Member Functions

NodePath	*NodePath* (*String* from)
NodePath	*get_as_property_path* ( )
String	*get_concatenated_subnames* ( )
String	*get_name* (*int* idx)
int	*get_name_count* ( )
String	*get_subname* (*int* idx)
int	*get_subname_count* ( )
bool	*is_absolute* ( )
bool	*is_empty* ( )

29.259.3 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.

29.259.4 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.

29.260 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

29.260.1 Brief Description

Base class for all non built-in types.

29.260.2 Member Functions

void	*_get* (*String* property) virtual
Array	*_get_property_list* ( ) virtual
void	*_init* ( ) virtual
void	*_notification* (*int* what) virtual
bool	*_set* (*String* property, *Variant* value) virtual

Continued on next page

Table 15 – continued from previous page

void	*add_user_signal* (*String* signal, *Array* arguments=[ ])
Variant	*call* (*String* method) vararg
Variant	*call_deferred* (*String* method) vararg
Variant	*callv* (*String* method, *Array* arg_array)
bool	*can_translate_messages* ( ) const
Error	*connect* (*String* signal, *Object* target, *String* method, *Array* binds=[ ], *int* flags=0)
void	*disconnect* (*String* signal, *Object* target, *String* method)
Variant	*emit_signal* (*String* signal) vararg
void	*free* ( )
Variant	*get* (*String* property) const
String	*get_class* ( ) const
Array	*get_incoming_connections* ( ) const
Variant	*get_indexed* (*NodePath* property) const
int	*get_instance_id* ( ) const
Variant	*get_meta* (*String* name) const
PoolStringArray	*get_meta_list* ( ) const
Array	*get_method_list* ( ) const
Array	*get_property_list* ( ) const
Reference	*get_script* ( ) const
Array	*get_signal_connection_list* (*String* signal) const
Array	*get_signal_list* ( ) const
bool	*has_meta* (*String* name) const
bool	*has_method* (*String* method) const
bool	*has_user_signal* (*String* signal) const
bool	*is_blocking_signals* ( ) const
bool	*is_class* (*String* type) const
bool	*is_connected* (*String* signal, *Object* target, *String* method) const
bool	*is_queued_for_deletion* ( ) const
void	*notification* (*int* what, *bool* reversed=false)
void	*property_list_changed_notify* ( )
void	*set* (*String* property, *Variant* value)
void	*set_block_signals* (*bool* enable)
void	*set_indexed* (*NodePath* property, *Variant* value)
void	*set_message_translation* (*bool* enable)
void	*set_meta* (*String* name, *Variant* value)
void	*set_script* (*Reference* script)
String	*tr* (*String* message) const

29.260.3 Signals

- **script_changed** ( )

Emitted whenever the script of the Object is changed.

29.260.4 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.

29.260.5 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.

29.260.6 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.

29.260.7 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.

- *Error* **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*.

29.261 OccluderPolygon2D

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.261.1 Brief Description

Defines a 2D polygon for LightOccluder2D.

29.261.2 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.

29.261.3 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*.

29.261.4 Description

Editor facility that helps you draw a 2D polygon used as resource for *LightOccluder2D*.

29.262 OmniLight

Inherits: *Light* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.262.1 Brief Description

OmniDirectional Light, such as a light bulb or a candle.

29.262.2 Member Variables

- *float* **omni_attenuation**
- *float* **omni_range**
- *ShadowDetail* **omni_shadow_detail**
- *ShadowMode* **omni_shadow_mode**

29.262.3 Enums

enum **ShadowDetail**

- **SHADOW_DETAIL_VERTICAL = 0**
- **SHADOW_DETAIL_HORIZONTAL = 1**

enum **ShadowMode**

- **SHADOW_DUAL_PARABOLOID = 0**
- **SHADOW_CUBE = 1**

29.262.4 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*.

29.263 OptionButton

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.263.1 Brief Description

Button control that provides selectable options when pressed.

29.263.2 Member Functions

void	*add_icon_item* (*Texture* texture, *String* label, *int* id=-1)
void	*add_item* (*String* label, *int* id=-1)
void	*add_separator* ( )
void	*clear* ( )
int	*get_item_count* ( ) const
Texture	*get_item_icon* (*int* idx) const
int	*get_item_id* (*int* idx) const
Variant	*get_item_metadata* (*int* idx) const
String	*get_item_text* (*int* idx) const
PopupMenu	*get_popup* ( ) const
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)
void	*set_item_disabled* (*int* idx, *bool* disabled)
void	*set_item_icon* (*int* idx, *Texture* texture)
void	*set_item_id* (*int* idx, *int* id)
void	*set_item_metadata* (*int* idx, *Variant* metadata)
void	*set_item_text* (*int* idx, *String* text)

29.263.3 Signals

- **item_focused** (*int* ID)

This signal is emitted when user navigated to an item using `ui_up` or `ui_down` action. ID of the item selected is passed as argument (if no IDs were added, ID will be just the item index).

- **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).

29.263.4 Member Variables

- *int* **selected**

29.263.5 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.

29.263.6 Member Function Description

- void **add_icon_item** (*Texture* texture, *String* label, *int* id=-1)

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".

29.264 OrientedPathFollow

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.264.1 Brief Description

Oriented point sampler for a *Path*.

29.264.2 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.
- *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.

29.264.3 Description

This node behaves like *PathFollow*, except it uses its parent *Path* up vector information to enforce orientation.

Make sure to check if the curve of this node's parent *Path* has up vectors enabled. See *PathFollow* and *Curve3D* for further information.

29.265 OS

Inherits: *Object*

Category: Core

29.265.1 Brief Description

Operating System functions.

29.265.2 Member Functions

void	*alert* (*String* text, *String* title="Alert!")
bool	*can_draw* ( ) const
bool	*can_use_threads* ( ) const
void	*center_window* ( )
void	*delay_msec* (*int* msec) const
void	*delay_usec* (*int* usec) const
void	*dump_memory_to_file* (*String* file)
void	*dump_resources_to_file* (*String* file)
int	*execute* (*String* path, *PoolStringArray* arguments, *bool* blocking, *Array* output=[ ])
int	*find_scancode_from_string* (*String* string) const
int	*get_audio_driver_count* ( ) const
String	*get_audio_driver_name* (*int* driver) const
PoolStringArray	*get_cmdline_args* ( )
Dictionary	*get_date* (*bool* utc=false) const
Dictionary	*get_datetime* (*bool* utc=false) const
Dictionary	*get_datetime_from_unix_time* (*int* unix_time_val) const
int	*get_dynamic_memory_usage* ( ) const
String	*get_environment* (*String* environment) const
String	*get_executable_path* ( ) const
String	*get_latin_keyboard_variant* ( ) const
String	*get_locale* ( ) const
String	*get_model_name* ( ) const
String	*get_name* ( ) const
int	*get_power_percent_left* ( )
int	*get_power_seconds_left* ( )
PowerState	*get_power_state* ( )
int	*get_process_id* ( ) const
int	*get_processor_count* ( ) const
Vector2	*get_real_window_size* ( ) const
String	*get_scancode_string* (*int* code) const
int	*get_screen_count* ( ) const
int	*get_screen_dpi* (*int* screen=-1) const
Vector2	*get_screen_position* (*int* screen=-1) const
Vector2	*get_screen_size* (*int* screen=-1) const
int	*get_splash_tick_msec* ( ) const
int	*get_static_memory_peak_usage* ( ) const
int	*get_static_memory_usage* ( ) const
String	*get_system_dir* (*SystemDir* dir) const
int	*get_system_time_secs* ( ) const
int	*get_ticks_msec* ( ) const
Dictionary	*get_time* (*bool* utc=false) const
Dictionary	*get_time_zone_info* ( ) const
String	*get_unique_id* ( ) const
int	*get_unix_time* ( ) const

Continued on next page

Table 16 – continued from previous page

int	*get_unix_time_from_datetime* (*Dictionary* datetime) const
String	*get_user_data_dir* ( ) const
int	*get_video_driver_count* ( ) const
String	*get_video_driver_name* (*int* driver) const
int	*get_virtual_keyboard_height* ( )
Rect2	*get_window_safe_area* ( ) const
bool	*has_environment* (*String* environment) const
bool	*has_feature* (*String* tag_name) const
bool	*has_touchscreen_ui_hint* ( ) const
bool	*has_virtual_keyboard* ( ) const
void	*hide_virtual_keyboard* ( )
bool	*is_debug_build* ( ) const
bool	*is_ok_left_and_cancel_right* ( ) const
bool	*is_scancode_unicode* (*int* code) const
bool	*is_stdout_verbose* ( ) const
bool	*is_userfs_persistent* ( ) const
bool	*is_window_always_on_top* ( ) const
Error	*kill* (*int* pid)
bool	*native_video_is_playing* ( )
void	*native_video_pause* ( )
Error	*native_video_play* (*String* path, *float* volume, *String* audio_track, *String* subtitle_track)
void	*native_video_stop* ( )
void	*native_video_unpause* ( )
void	*print_all_resources* (*String* tofile="")
void	*print_all_textures_by_size* ( )
void	*print_resources_by_type* (*PoolStringArray* types)
void	*print_resources_in_use* (*bool* short=false)
void	*request_attention* ( )
void	*set_icon* (*Image* icon)
void	*set_ime_position* (*Vector2* position)
Error	*set_thread_name* (*String* name)
void	*set_use_file_access_save_and_swap* (*bool* enabled)
void	*set_window_always_on_top* (*bool* enabled)
void	*set_window_title* (*String* title)
Error	*shell_open* (*String* uri)
void	*show_virtual_keyboard* (*String* existing_text="")

29.265.3 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** - The exit code passed to the OS when the main loop exits.
- *bool* **keep_screen_on** - If `true` the engine tries to keep the screen on while the game is running. Useful on mobile.
- *bool* **low_processor_usage_mode** - If `true` the engine optimizes for low processor usage by only refreshing the screen if needed. Can improve battery consumption on mobile.
- _OS.ScreenOrientation **screen_orientation** - The current screen orientation.

- *bool* **vsync_enabled** - If `true` vertical synchronization (Vsync) is 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.
- *bool* **window_per_pixel_transparency_enabled**
- *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).

29.265.4 Enums

enum **SystemDir**

- **SYSTEM_DIR_DESKTOP = 0** — Desktop directory path.
- **SYSTEM_DIR_DCIM = 1** — DCIM (Digital Camera Images) directory path.
- **SYSTEM_DIR_DOCUMENTS = 2** — Documents directory path.
- **SYSTEM_DIR_DOWNLOADS = 3** — Downloads directory path.
- **SYSTEM_DIR_MOVIES = 4** — Movies directory path.
- **SYSTEM_DIR_MUSIC = 5** — Music directory path.
- **SYSTEM_DIR_PICTURES = 6** — Pictures directory path.
- **SYSTEM_DIR_RINGTONES = 7** — Ringtones directory path.

enum **ScreenOrientation**

- **SCREEN_ORIENTATION_LANDSCAPE = 0** — Landscape screen orientation.
- **SCREEN_ORIENTATION_PORTRAIT = 1** — Portrait screen orientation.
- **SCREEN_ORIENTATION_REVERSE_LANDSCAPE = 2** — Reverse landscape screen orientation.
- **SCREEN_ORIENTATION_REVERSE_PORTRAIT = 3** — Reverse portrait screen orientation.
- **SCREEN_ORIENTATION_SENSOR_LANDSCAPE = 4** — Uses landscape or reverse landscape based on the hardware sensor.
- **SCREEN_ORIENTATION_SENSOR_PORTRAIT = 5** — Uses portrait or reverse portrait based on the hardware sensor.
- **SCREEN_ORIENTATION_SENSOR = 6** — Uses most suitable orientation based on the hardware sensor.

enum **PowerState**

- **POWERSTATE_UNKNOWN = 0** — Unknown powerstate.
- **POWERSTATE_ON_BATTERY = 1** — Unplugged, running on battery.
- **POWERSTATE_NO_BATTERY = 2** — Plugged in, no battery available.
- **POWERSTATE_CHARGING = 3** — Plugged in, battery charging.

- **POWERSTATE_CHARGED = 4** — Plugged in, battery fully charged.

enum **Weekday**

- **DAY_SUNDAY = 0** — Sunday.
- **DAY_MONDAY = 1** — Monday.
- **DAY_TUESDAY = 2** — Tuesday.
- **DAY_WEDNESDAY = 3** — Wednesday.
- **DAY_THURSDAY = 4** — Thursday.
- **DAY_FRIDAY = 5** — Friday.
- **DAY_SATURDAY = 6** — Saturday.

enum **Month**

- **MONTH_JANUARY = 1** — January.
- **MONTH_FEBRUARY = 2** — February.
- **MONTH_MARCH = 3** — March.
- **MONTH_APRIL = 4** — April.
- **MONTH_MAY = 5** — May.
- **MONTH_JUNE = 6** — June.
- **MONTH_JULY = 7** — July.
- **MONTH_AUGUST = 8** — August.
- **MONTH_SEPTEMBER = 9** — September.
- **MONTH_OCTOBER = 10** — October.
- **MONTH_NOVEMBER = 11** — November.
- **MONTH_DECEMBER = 12** — December.

29.265.5 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.

29.265.6 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 **center_window** ( )

Centers the window on the screen if in windowed mode.

- void **delay_msec** (*int* msec) const

Delay execution of the current thread by given milliseconds.

- void **delay_usec** (*int* usec) const

Delay execution 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 with the arguments passed as an array of strings. Platform path resolution will take place. The resolved file must exist and be executable.

The arguments are used in the given order and separated by a space, so `OS.execute('ping', ['-c', '3', 'godotengine.org'])` will resolve to `ping -c 3 godotengine.org` in the system's shell.

This method has slightly different behaviour based on whether the `blocking` mode is enabled.

When `blocking` is enabled, the Godot thread will pause its execution while waiting for the process to terminate. The shell output of the process will be written to the `output` array as a single string. When the process terminates, the Godot thread will resume execution.

When `blocking` is disabled, the Godot thread will continue while the new process runs. It is not possible to retrieve the shell output in non-blocking mode, so `output` will be empty.

The return value also depends on the blocking mode. When blocking, the method will return -2 (no process ID information is available in blocking mode). When non-blocking, the method returns a process ID, which you can use to monitor the process (and potentially terminate it with *kill*). If the process forking (non-blocking) or opening (blocking) fails, the method will return -1.

Example of blocking mode and retrieving the shell output:

```
var output = []
OS.execute('ls', ['-l', '/tmp'], true, output)
```

Example of non-blocking mode, running another instance of the project and storing its process ID:

```
var pid = OS.execute(OS.get_executable_path(), [], false)
```

If you wish to access a shell built-in or perform a composite command, a platform-specific shell can be invoked. For example:

```
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")

- *int* **get_audio_driver_count** ( ) const

Returns the total number of available audio drivers.

- *String* **get_audio_driver_name** (*int* driver) const

Returns the audio driver name for the given index.

- *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.

- *PowerState* **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.

- *Vector2* **get_real_window_size** ( ) const

Returns the window size including decorations like window borders.

- *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** (*SystemDir* 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.

Returns empty string on HTML5 and UWP which are not supported yet.

- *int* **get_unix_time ( )** const

Returns the current unix epoch 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://`).

On Linux, this is `~/.local/share/godot/app_userdata/[project_name]`, or `~/.local/share/[custom_name]` if `use_custom_user_dir` is set.

On macOS, this is `~/Library/Application Support/Godot/app_userdata/[project_name]`, or `~/Library/Application Support/[custom_name]` if `use_custom_user_dir` is set.

On Windows, this is `%APPDATA%/Godot/app_userdata/[project_name]`, or `%APPDATA%/[custom_name]` if `use_custom_user_dir` is set.

If the project name is empty, `user://` falls back to `res://`.

- *int* **get_video_driver_count ( )** const
- *String* **get_video_driver_name (** *int* driver **)** const
- *int* **get_virtual_keyboard_height ( )**

Returns the on-screen keyboard's height in pixels. Returns 0 if there is no keyboard or it is currently hidden.

- *Rect2* **get_window_safe_area ( )** const
- *bool* **has_environment (** *String* environment **)** const

Returns `true` if an environment variable exists.

- *bool* **has_feature (** *String* tag_name **)** const

Returns `true` if the feature for the given feature tag is supported in the currently running instance, depending on platform, build etc. Can be used to check whether you're currently running a debug build, on a certain platform or arch, etc. See feature tags documentation.

- *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.

- *bool* **is_window_always_on_top** **(** **)** const

Returns `true` if the window should always be on top of other windows.

- *Error* **kill** **(** *int* pid **)**

Kill (terminate) the process identified by the given process ID (`pid`), e.g. the one returned by *execute* in non-blocking mode.

Note that this method can also 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.

- *Error* **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)
- *Error* **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_always_on_top** (*bool* enabled)

Sets whether the window should always be on top.

- void **set_window_title** (*String* title)

Sets the window title to the specified string.

- *Error* **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).

29.266 PackedDataContainer

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.266.1 Brief Description

29.266.2 Member Functions

Error	*pack* (*Variant* value)
int	*size* ( ) const

29.266.3 Member Variables

- *PoolByteArray* **__data__**

29.266.4 Member Function Description

- *Error* **pack** (*Variant* value)
- *int* **size** ( ) const

29.267 PackedDataContainerRef

Inherits: *Reference* < *Object*

Category: Core

29.267.1 Brief Description

29.267.2 Member Functions

int	*size* ( ) const

29.267.3 Member Function Description

- *int* **size** ( ) const

29.268 PackedScene

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.268.1 Brief Description

An abstraction of a serialized scene.

29.268.2 Member Functions

bool	*can_instance* ( ) const
SceneState	*get_state* ( )
Node	*instance* (*GenEditState* edit_state=0) const
Error	*pack* (*Node* path)

29.268.3 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.

29.268.4 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.

29.268.5 Description

A simplified interface to a scene file. Provides access to operations and checks that can be performed on the scene resource itself.

Can be used to save a node to a file. When saving, the node as well as all the node it owns get saved (see `owner` property on *Node*). Note that the node doesn't need to own itself.

Example of saving a node:

```
var scene = PackedScene.new()
var result = scene.pack(child)
if result == OK:
    ResourceSaver.save("res://path/name.scn", scene) // or user://...
```

29.268.6 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** **(** *GenEditState* edit_state=0 **)** const

Instantiates the scene's node hierarchy. Triggers child scene instantiation(s). Triggers the NOTIFICATION_INSTANCED notification on the root node.

- *Error* **pack** **(** *Node* path **)**

Pack will ignore any sub-nodes not owned by given node. See Node.set_owner.

29.269 PacketPeer

Inherits: *Reference* < *Object*

Inherited By: *PacketPeerStream*, *PacketPeerUDP*, *WebSocketPeer*, *NetworkedMultiplayerPeer*

Category: Core

29.269.1 Brief Description

Abstraction and base class for packet-based protocols.

29.269.2 Member Functions

int	*get_available_packet_count* ( ) const
PoolByteArray	*get_packet* ( )
Error	*get_packet_error* ( ) const
Variant	*get_var* ( )
Error	*put_packet* (*PoolByteArray* buffer)
Error	*put_var* (*Variant* var)

29.269.3 Member Variables

- *bool* **allow_object_decoding**

29.269.4 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.

29.269.5 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.

- *Error* **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.

- *Error* **put_packet** (*PoolByteArray* buffer)

Send a raw packet.

- *Error* **put_var** (*Variant* var)

Send a Variant as a packet.

29.270 PacketPeerStream

Inherits: *PacketPeer* < *Reference* < *Object*

Category: Core

29.270.1 Brief Description

Wrapper to use a PacketPeer over a StreamPeer.

29.270.2 Member Variables

- *int* **input_buffer_max_size**
- *int* **output_buffer_max_size**
- *StreamPeer* **stream_peer** - The wrapped *StreamPeer* object.

29.270.3 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.

29.271 PacketPeerUDP

Inherits: *PacketPeer* < *Reference* < *Object*

Category: Core

29.271.1 Brief Description

UDP packet peer.

29.271.2 Member Functions

void	*close* ( )
String	*get_packet_ip* ( ) const
int	*get_packet_port* ( ) const
bool	*is_listening* ( ) const
Error	*listen* (*int* port, *String* bind_address="*", *int* recv_buf_size=65536)
Error	*set_dest_address* (*String* host, *int* port)
Error	*wait* ( )

29.271.3 Description

UDP packet peer. Can be used to send raw UDP packets as well as *Variant*s.

29.271.4 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 *PacketPeer.get_packet* or *PacketPeer.get_var*).

- *int* **get_packet_port** ( ) const

Return the port of the remote peer that sent the last packet(that was received with *PacketPeer.get_packet* or *PacketPeer.get_var*).

- *bool* **is_listening** ( ) const

Return whether this `PacketPeerUDP` is listening.

- *Error* **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).

- *Error* **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.

- *Error* **wait** ( )

Wait for a packet to arrive on the listening port, see *listen*.

29.272 Panel

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.272.1 Brief Description

Provides an opaque background for *Control* children.

29.272.2 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.

29.273 PanelContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *ScriptEditor*

Category: Core

29.273.1 Brief Description

Panel container type.

29.273.2 Description

Panel container type. This container fits controls inside of the delimited area of a stylebox. It's useful for giving controls an outline.

29.274 PanoramaSky

Inherits: *Sky* < *Resource* < *Reference* < *Object*

Category: Core

29.274.1 Brief Description

A type of *Sky* used to draw a background texture.

29.274.2 Member Variables

- *Texture* **panorama** - *Texture* to be applied to the PanoramaSky.

29.274.3 Description

A resource referenced in an *Environment* that is used to draw a background. The Panorama sky functions similar to skyboxes in other engines except it uses a equirectangular sky map instead of a cube map.

29.275 ParallaxBackground

Inherits: *CanvasLayer* < *Node* < *Object*

Category: Core

29.275.1 Brief Description

A node used to create a parallax scrolling background.

29.275.2 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.

29.275.3 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*.

29.276 ParallaxLayer

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.276.1 Brief Description

A parallax scrolling layer to be used with *ParallaxBackground*.

29.276.2 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.

29.276.3 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.

29.277 Particles

Inherits: *GeometryInstance* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.277.1 Brief Description

3D particle emitter.

29.277.2 Member Functions

AABB	*capture_aabb* ( ) const
void	*restart* ( )

29.277.3 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**

29.277.4 Numeric Constants

- **MAX_DRAW_PASSES = 4**

29.277.5 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.

29.277.6 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.

29.277.7 Member Function Description

- *AABB* **capture_aabb** ( ) const
- void **restart** ( )

29.278 Particles2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.278.1 Brief Description

2D particle emitter.

29.278.2 Member Functions

Rect2	*capture_rect* ( ) const
void	*restart* ( )

29.278.3 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.

29.278.4 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.

29.278.5 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.

29.278.6 Member Function Description

- *Rect2* **capture_rect ( )** const
- void **restart ( )**

29.279 ParticlesMaterial

Inherits: *Material* < *Resource* < *Reference* < *Object*

Category: Core

29.279.1 Brief Description

Particle properties for *Particles* and *Particles2D* nodes.

29.279.2 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*.

29.279.3 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**

29.279.4 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.

29.280 Path

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.280.1 Brief Description

Container for a *Curve3D*.

29.280.2 Signals

- **curve_changed ( )**

29.280.3 Member Variables

- *Curve3D* **curve**

29.280.4 Description

This class is a container/Node-ification of a *Curve3D*, so it can have *Spatial* properties and *Node* info.

29.281 Path2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.281.1 Brief Description

Contains a *Curve2D* path for *PathFollow2D* nodes to follow.

29.281.2 Member Variables

- *Curve2D* **curve** - A *Curve2D* describing the path.

29.281.3 Description

Can have *PathFollow2D* child-nodes moving along the *Curve2D*. See *PathFollow2D* for more information on this usage.

29.282 PathFollow

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.282.1 Brief Description

Point sampler for a *Path*.

29.282.2 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.

29.282.3 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.

29.282.4 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.

29.283 PathFollow2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.283.1 Brief Description

Point sampler for a *Path2D*.

29.283.2 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.

29.283.3 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.

29.284 PCKPacker

Inherits: *Reference* < *Object*

Category: Core

29.284.1 Brief Description

29.284.2 Member Functions

Error	*add_file* (*String* pck_path, *String* source_path)
Error	*flush* (*bool* verbose)
Error	*pck_start* (*String* pck_name, *int* alignment)

29.284.3 Member Function Description

- *Error* **add_file** (*String* pck_path, *String* source_path)
- *Error* **flush** (*bool* verbose)
- *Error* **pck_start** (*String* pck_name, *int* alignment)

29.285 Performance

Inherits: *Object*

Category: Core

29.285.1 Brief Description

Exposes performance related data.

29.285.2 Member Functions

float	*get_monitor* (*Monitor* monitor) const

29.285.3 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**

29.285.4 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.

29.285.5 Member Function Description

- *float* **get_monitor** (*Monitor* 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
```

29.286 PHashTranslation

Inherits: *Translation* < *Resource* < *Reference* < *Object*

Category: Core

29.286.1 Brief Description

Optimized translation.

29.286.2 Member Functions

void	*generate* (*Translation* from)

29.286.3 Description

Optimized translation. Uses real-time compressed translations, which results in very small dictionaries.

29.286.4 Member Function Description

- void **generate** (*Translation* from)

29.287 PhysicalBone

Inherits: *PhysicsBody* < *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

29.287.1 Brief Description

29.287.2 Member Functions

bool	*get_simulate_physics* ( )
bool	*is_simulating_physics* ( )
bool	*is_static_body* ( )

29.287.3 Member Variables

- *Transform* **body_offset**
- *float* **bounce**
- *float* **friction**
- *float* **gravity_scale**
- *Transform* **joint_offset**
- *JointType* **joint_type**
- *float* **mass**
- *float* **weight**

29.287.4 Enums

enum **JointType**

- **JOINT_TYPE_NONE = 0**
- **JOINT_TYPE_PIN = 1**
- **JOINT_TYPE_CONE = 2**
- **JOINT_TYPE_HINGE = 3**
- **JOINT_TYPE_SLIDER = 4**
- **JOINT_TYPE_6DOF = 5**

29.287.5 Member Function Description

- *bool* **get_simulate_physics** ( )
- *bool* **is_simulating_physics** ( )
- *bool* **is_static_body** ( )

29.288 Physics2DDirectBodyState

Inherits: *Object*

Inherited By: *Physics2DDirectBodyStateSW*

Category: Core

29.288.1 Brief Description

Direct access object to a physics body in the *Physics2DServer*.

29.288.2 Member Functions

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
Vector2	*get_contact_collider_position* (*int* contact_idx) const
int	*get_contact_collider_shape* (*int* contact_idx) const
Variant	*get_contact_collider_shape_metadata* (*int* contact_idx) const
Vector2	*get_contact_collider_velocity_at_position* (*int* contact_idx) const
int	*get_contact_count* ( ) const
Vector2	*get_contact_local_normal* (*int* contact_idx) const
Vector2	*get_contact_local_position* (*int* contact_idx) const
int	*get_contact_local_shape* (*int* contact_idx) const
Physics2DDirectSpaceState	*get_space_state* ( )
void	*integrate_forces* ( )

29.288.3 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.

29.288.4 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.

29.288.5 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.

29.289 Physics2DDirectBodyStateSW

Inherits: *Physics2DDirectBodyState* < *Object*

Category: Core

29.289.1 Brief Description

Software implementation of *Physics2DDirectBodyState*.

29.289.2 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.

29.290 Physics2DDirectSpaceState

Inherits: *Object*

Category: Core

29.290.1 Brief Description

Direct access object to a space in the *Physics2DServer*.

29.290.2 Member Functions

Array	*cast_motion* (*Physics2DShapeQueryParameters* shape)
Array	*collide_shape* (*Physics2DShapeQueryParameters* shape, *int* max_results=32)
Dictionary	*get_rest_info* (*Physics2DShapeQueryParameters* shape)
Array	*intersect_point* (*Vector2* point, *int* max_results=32, *Array* exclude=[ ], *int* collision_layer=2147483647)
Dictionary	*intersect_ray* (*Vector2* from, *Vector2* to, *Array* exclude=[ ], *int* collision_layer=2147483647)
Array	*intersect_shape* (*Physics2DShapeQueryParameters* shape, *int* max_results=32)

29.290.3 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.

29.290.4 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 *RID*s 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 *RID*s 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.

29.291 Physics2DServer

Inherits: *Object*

Inherited By: *Physics2DServerSW*

Category: Core

29.291.1 Brief Description

Physics 2D Server.

29.291.2 Member Functions

void	*area_add_shape* (*RID* area, *RID* shape, *Transform2D* transform=Transform2D(1, 0, 0, 1, 0, 0))
void	*area_attach_object_instance_id* (*RID* area, *int* id)
void	*area_clear_shapes* (*RID* area)
RID	*area_create* ( )
int	*area_get_object_instance_id* (*RID* area) const
Variant	*area_get_param* (*RID* area, *AreaParameter* param) const
RID	*area_get_shape* (*RID* area, *int* shape_idx) const

Table 17 – continued from previous page

int	*area_get_shape_count* (*RID* area) const
Transform2D	*area_get_shape_transform* (*RID* area, *int* shape_idx) const
RID	*area_get_space* (*RID* area) const
AreaSpaceOverrideMode	*area_get_space_override_mode* (*RID* area) const
Transform2D	*area_get_transform* (*RID* area) const
void	*area_remove_shape* (*RID* area, *int* shape_idx)
void	*area_set_area_monitor_callback* (*RID* area, *Object* receiver, *String* method)
void	*area_set_collision_layer* (*RID* area, *int* layer)
void	*area_set_collision_mask* (*RID* area, *int* mask)
void	*area_set_monitor_callback* (*RID* area, *Object* receiver, *String* method)
void	*area_set_monitorable* (*RID* area, *bool* monitorable)
void	*area_set_param* (*RID* area, *AreaParameter* param, *Variant* value)
void	*area_set_shape* (*RID* area, *int* shape_idx, *RID* shape)
void	*area_set_shape_disabled* (*RID* area, *int* shape_idx, *bool* disable)
void	*area_set_shape_transform* (*RID* area, *int* shape_idx, *Transform2D* transform)
void	*area_set_space* (*RID* area, *RID* space)
void	*area_set_space_override_mode* (*RID* area, *AreaSpaceOverrideMode* mode)
void	*area_set_transform* (*RID* area, *Transform2D* transform)
void	*body_add_collision_exception* (*RID* body, *RID* excepted_body)
void	*body_add_force* (*RID* body, *Vector2* offset, *Vector2* force)
void	*body_add_shape* (*RID* body, *RID* shape, *Transform2D* transform=Transform2D(1, 0, 0, 1, 0, 0))
void	*body_apply_impulse* (*RID* body, *Vector2* position, *Vector2* impulse)
void	*body_attach_object_instance_id* (*RID* body, *int* id)
void	*body_clear_shapes* (*RID* body)
RID	*body_create* ( )
int	*body_get_collision_layer* (*RID* body) const
int	*body_get_collision_mask* (*RID* body) const
CCDMode	*body_get_continuous_collision_detection_mode* (*RID* body) const
Physics2DDirectBodyState	*body_get_direct_state* (*RID* body)
int	*body_get_max_contacts_reported* (*RID* body) const
BodyMode	*body_get_mode* (*RID* body) const
int	*body_get_object_instance_id* (*RID* body) const
float	*body_get_param* (*RID* body, *BodyParameter* param) const
RID	*body_get_shape* (*RID* body, *int* shape_idx) const
int	*body_get_shape_count* (*RID* body) const
Variant	*body_get_shape_metadata* (*RID* body, *int* shape_idx) const
Transform2D	*body_get_shape_transform* (*RID* body, *int* shape_idx) const
RID	*body_get_space* (*RID* body) const
Variant	*body_get_state* (*RID* body, *BodyState* state) const
bool	*body_is_omitting_force_integration* (*RID* body) const
void	*body_remove_collision_exception* (*RID* body, *RID* excepted_body)
void	*body_remove_shape* (*RID* body, *int* shape_idx)
void	*body_set_axis_velocity* (*RID* body, *Vector2* axis_velocity)
void	*body_set_collision_layer* (*RID* body, *int* layer)
void	*body_set_collision_mask* (*RID* body, *int* mask)
void	*body_set_continuous_collision_detection_mode* (*RID* body, *CCDMode* mode)
void	*body_set_force_integration_callback* (*RID* body, *Object* receiver, *String* method, *Variant* userdata=nul
void	*body_set_max_contacts_reported* (*RID* body, *int* amount)
void	*body_set_mode* (*RID* body, *BodyMode* mode)
void	*body_set_omit_force_integration* (*RID* body, *bool* enable)

Table 17 – continued from previous page

void	*body_set_param* (*RID* body, *BodyParameter* param, *float* value)
void	*body_set_shape* (*RID* body, *int* shape_idx, *RID* shape)
void	*body_set_shape_as_one_way_collision* (*RID* body, *int* shape_idx, *bool* enable)
void	*body_set_shape_disabled* (*RID* body, *int* shape_idx, *bool* disable)
void	*body_set_shape_metadata* (*RID* body, *int* shape_idx, *Variant* metadata)
void	*body_set_shape_transform* (*RID* body, *int* shape_idx, *Transform2D* transform)
void	*body_set_space* (*RID* body, *RID* space)
void	*body_set_state* (*RID* body, *BodyState* state, *Variant* value)
bool	*body_test_motion* (*RID* body, *Transform2D* from, *Vector2* motion, *bool* infinite_inertia, *float* margin=0
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)
float	*damped_string_joint_get_param* (*RID* joint, *DampedStringParam* param) const
void	*damped_string_joint_set_param* (*RID* joint, *DampedStringParam* param, *float* value)
void	*free_rid* (*RID* rid)
int	*get_process_info* (*ProcessInfo* process_info)
RID	*groove_joint_create* (*Vector2* groove1_a, *Vector2* groove2_a, *Vector2* anchor_b, *RID* body_a, *RID* bod
float	*joint_get_param* (*RID* joint, *JointParam* param) const
JointType	*joint_get_type* (*RID* joint) const
void	*joint_set_param* (*RID* joint, *JointParam* param, *float* value)
RID	*line_shape_create* ( )
RID	*pin_joint_create* (*Vector2* anchor, *RID* body_a, *RID* body_b)
RID	*ray_shape_create* ( )
RID	*rectangle_shape_create* ( )
RID	*segment_shape_create* ( )
void	*set_active* (*bool* active)
Variant	*shape_get_data* (*RID* shape) const
ShapeType	*shape_get_type* (*RID* shape) const
void	*shape_set_data* (*RID* shape, *Variant* data)
RID	*space_create* ( )
Physics2DDirectSpaceState	*space_get_direct_state* (*RID* space)
float	*space_get_param* (*RID* space, *SpaceParameter* param) const
bool	*space_is_active* (*RID* space) const
void	*space_set_active* (*RID* space, *bool* active)
void	*space_set_param* (*RID* space, *SpaceParameter* param, *float* value)

29.291.3 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 to 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.

29.291.4 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.

29.291.5 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, *AreaParameter* 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.

- *AreaSpaceOverrideMode* **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_area_monitor_callback** (*RID* area, *Object* receiver, *String* method)
- 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_monitorable** (*RID* area, *bool* monitorable)
- void **area_set_param** (*RID* area, *AreaParameter* 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, *AreaSpaceOverrideMode* 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.

- *CCDMode* **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*.

- *BodyMode* **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, *BodyParameter* 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, *BodyState* 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, *CCDMode* 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, *BodyMode* 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, *BodyParameter* 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 *space_create*).

- void **body_set_state** (*RID* body, *BodyState* state, *Variant* value)

Sets a body state (see BODY_STATE* constants).

- *bool* **body_test_motion** (*RID* body, *Transform2D* from, *Vector2* motion, *bool* infinite_inertia, *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, *DampedStringParam* param) const

Returns the value of a damped spring joint parameter.

- void **damped_string_joint_set_param** (*RID* joint, *DampedStringParam* 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** (*ProcessInfo* 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 bodyies are assumed to be the joint itself.

- *float* **joint_get_param** (*RID* joint, *JointParam* param) const

Returns the value of a joint parameter.

- *JointType* **joint_get_type** (*RID* joint) const

Returns the type of a joint (see JOINT_* constants).

- void **joint_set_param** (*RID* joint, *JointParam* 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.

- *ShapeType* **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, *SpaceParameter* 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, *SpaceParameter* param, *float* value)

Sets the value for a space parameter. A list of available parameters is on the SPACE_PARAM_* constants.

29.292 Physics2DServerSW

Inherits: *Physics2DServer* < *Object*

Category: Core

29.292.1 Brief Description

Software implementation of *Physics2DServer*.

29.292.2 Description

This class exposes no new methods or properties and should not be used, as *Physics2DServer* automatically selects the best implementation available.

29.293 Physics2DShapeQueryParameters

Inherits: *Reference* < *Object*

Category: Core

29.293.1 Brief Description

Parameters to be sent to a 2D shape physics query.

29.293.2 Member Functions

void	*set_shape* (*Resource* shape)

29.293.3 Member Variables

- *int* **collision_layer** - The physics layer the query should be made on.
- *Array* **exclude** - The list of objects or object *RID*s, 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.

29.293.4 Description

This class contains the shape and other parameters for intersection/collision queries.

29.293.5 Member Function Description

- void **set_shape** (*Resource* shape)

Set the *Shape2D* that will be used for collision/intersection queries.

29.294 Physics2DShapeQueryResult

Inherits: *Reference* < *Object*

Category: Core

29.294.1 Brief Description

29.294.2 Member Functions

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

29.294.3 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

29.295 Physics2DTestMotionResult

Inherits: *Reference* < *Object*

Category: Core

29.295.1 Brief Description

29.295.2 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**

29.296 PhysicsBody

Inherits: *CollisionObject* < *Spatial* < *Node* < *Object*

Inherited By: *PhysicalBone*, *StaticBody*, *KinematicBody*, *RigidBody*

Category: Core

29.296.1 Brief Description

Base class for all objects affected by physics in 3D space.

29.296.2 Member Functions

void	*add_collision_exception_with* (*Node* body)
bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
void	*remove_collision_exception_with* (*Node* body)
void	*set_collision_layer_bit* (*int* bit, *bool* value)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.296.3 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.

29.296.4 Description

PhysicsBody is an abstract base class for implementing a physics body. All *Body types inherit from it.

29.296.5 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)

29.297 PhysicsBody2D

Inherits: *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Inherited By: *RigidBody2D*, *StaticBody2D*, *KinematicBody2D*

Category: Core

29.297.1 Brief Description

Base class for all objects affected by physics in 2D space.

29.297.2 Member Functions

void	*add_collision_exception_with* (*Node* body)
bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
void	*remove_collision_exception_with* (*Node* body)
void	*set_collision_layer_bit* (*int* bit, *bool* value)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.297.3 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.

29.297.4 Description

PhysicsBody2D is an abstract base class for implementing a physics body. All *Body2D types inherit from it.

29.297.5 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.

29.298 PhysicsDirectBodyState

Inherits: *Object*

Inherited By: *BulletPhysicsDirectBodyState*

Category: Core

29.298.1 Brief Description

29.298.2 Member Functions

void	*add_central_force* (*Vector3* force)
void	*add_force* (*Vector3* force, *Vector3* position)
void	*add_torque* (*Vector3* torque)
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* ( )

29.298.3 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.

29.298.4 Member Function Description

- void **add_central_force** (*Vector3* force)
- void **add_force** (*Vector3* force, *Vector3* position)
- void **add_torque** (*Vector3* torque)
- 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** ( )

29.299 PhysicsDirectSpaceState

Inherits: *Object*

Category: Core

29.299.1 Brief Description

Direct access object to a space in the *PhysicsServer*.

29.299.2 Member Functions

Array	*cast_motion* (*PhysicsShapeQueryParameters* shape, *Vector3* motion)
Array	*collide_shape* (*PhysicsShapeQueryParameters* shape, *int* max_results=32)
Dictionary	*get_rest_info* (*PhysicsShapeQueryParameters* shape)
Dictionary	*intersect_ray* (*Vector3* from, *Vector3* to, *Array* exclude=[ ], *int* collision_layer=2147483647)
Array	*intersect_shape* (*PhysicsShapeQueryParameters* shape, *int* max_results=32)

29.299.3 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.

29.299.4 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 *RID*s 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.

29.300 PhysicsServer

Inherits: *Object*

Inherited By: *BulletPhysicsServer*

Category: Core

29.300.1 Brief Description

Server interface for low level physics access.

29.300.2 Member Functions

void	*area_add_shape* (*RID* area, *RID* shape, *Transform* transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)
void	*area_attach_object_instance_id* (*RID* area, *int* id)
void	*area_clear_shapes* (*RID* area)
RID	*area_create* ( )
int	*area_get_object_instance_id* (*RID* area) const
Variant	*area_get_param* (*RID* area, *AreaParameter* param) const
RID	*area_get_shape* (*RID* area, *int* shape_idx) const
int	*area_get_shape_count* (*RID* area) const
Transform	*area_get_shape_transform* (*RID* area, *int* shape_idx) const
RID	*area_get_space* (*RID* area) const
AreaSpaceOverrideMode	*area_get_space_override_mode* (*RID* area) const
Transform	*area_get_transform* (*RID* area) const
bool	*area_is_ray_pickable* (*RID* area) const
void	*area_remove_shape* (*RID* area, *int* shape_idx)
void	*area_set_area_monitor_callback* (*RID* area, *Object* receiver, *String* method)
void	*area_set_collision_layer* (*RID* area, *int* layer)
void	*area_set_collision_mask* (*RID* area, *int* mask)
void	*area_set_monitor_callback* (*RID* area, *Object* receiver, *String* method)
void	*area_set_monitorable* (*RID* area, *bool* monitorable)
void	*area_set_param* (*RID* area, *AreaParameter* param, *Variant* value)
void	*area_set_ray_pickable* (*RID* area, *bool* enable)
void	*area_set_shape* (*RID* area, *int* shape_idx, *RID* shape)
void	*area_set_shape_transform* (*RID* area, *int* shape_idx, *Transform* transform)
void	*area_set_space* (*RID* area, *RID* space)

Continued on next pag

Table 18 – continued from previous page

void	*area_set_space_override_mode* (*RID* area, *AreaSpaceOverrideMode* mode)
void	*area_set_transform* (*RID* area, *Transform* transform)
void	*body_add_collision_exception* (*RID* body, *RID* excepted_body)
void	*body_add_shape* (*RID* body, *RID* shape, *Transform* transform=Transform(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)
void	*body_apply_impulse* (*RID* body, *Vector3* position, *Vector3* impulse)
void	*body_apply_torque_impulse* (*RID* body, *Vector3* impulse)
void	*body_attach_object_instance_id* (*RID* body, *int* id)
void	*body_clear_shapes* (*RID* body)
RID	*body_create* (*BodyMode* mode=2, *bool* init_sleeping=false)
int	*body_get_collision_layer* (*RID* body) const
int	*body_get_collision_mask* (*RID* body) const
PhysicsDirectBodyState	*body_get_direct_state* (*RID* body)
float	*body_get_kinematic_safe_margin* (*RID* body) const
int	*body_get_max_contacts_reported* (*RID* body) const
BodyMode	*body_get_mode* (*RID* body) const
int	*body_get_object_instance_id* (*RID* body) const
float	*body_get_param* (*RID* body, *BodyParameter* param) const
RID	*body_get_shape* (*RID* body, *int* shape_idx) const
int	*body_get_shape_count* (*RID* body) const
Transform	*body_get_shape_transform* (*RID* body, *int* shape_idx) const
RID	*body_get_space* (*RID* body) const
Variant	*body_get_state* (*RID* body, *BodyState* state) const
bool	*body_is_axis_locked* (*RID* body, *BodyAxis* axis) const
bool	*body_is_continuous_collision_detection_enabled* (*RID* body) const
bool	*body_is_omitting_force_integration* (*RID* body) const
bool	*body_is_ray_pickable* (*RID* body) const
void	*body_remove_collision_exception* (*RID* body, *RID* excepted_body)
void	*body_remove_shape* (*RID* body, *int* shape_idx)
void	*body_set_axis_lock* (*RID* body, *BodyAxis* axis, *bool* lock)
void	*body_set_axis_velocity* (*RID* body, *Vector3* axis_velocity)
void	*body_set_collision_layer* (*RID* body, *int* layer)
void	*body_set_collision_mask* (*RID* body, *int* mask)
void	*body_set_enable_continuous_collision_detection* (*RID* body, *bool* enable)
void	*body_set_force_integration_callback* (*RID* body, *Object* receiver, *String* method, *Variant* userdata=null)
void	*body_set_kinematic_safe_margin* (*RID* body, *float* margin)
void	*body_set_max_contacts_reported* (*RID* body, *int* amount)
void	*body_set_mode* (*RID* body, *BodyMode* mode)
void	*body_set_omit_force_integration* (*RID* body, *bool* enable)
void	*body_set_param* (*RID* body, *BodyParameter* param, *float* value)
void	*body_set_ray_pickable* (*RID* body, *bool* enable)
void	*body_set_shape* (*RID* body, *int* shape_idx, *RID* shape)
void	*body_set_shape_transform* (*RID* body, *int* shape_idx, *Transform* transform)
void	*body_set_space* (*RID* body, *RID* space)
void	*body_set_state* (*RID* body, *BodyState* state, *Variant* value)
float	*cone_twist_joint_get_param* (*RID* joint, *ConeTwistJointParam* param) const
void	*cone_twist_joint_set_param* (*RID* joint, *ConeTwistJointParam* param, *float* value)
void	*free_rid* (*RID* rid)
bool	*generic_6dof_joint_get_flag* (*RID* joint, Axis axis, *G6DOFJointAxisFlag* flag)
float	*generic_6dof_joint_get_param* (*RID* joint, Axis axis, *G6DOFJointAxisParam* param)
void	*generic_6dof_joint_set_flag* (*RID* joint, Axis axis, *G6DOFJointAxisFlag* flag, *bool* enable)

Continued on next pag

Table 18 – continued from previous page

void	*generic_6dof_joint_set_param* (*RID* joint, Axis axis, *G6DOFJointAxisParam* param, *float* value)
int	*get_process_info* (*ProcessInfo* process_info)
bool	*hinge_joint_get_flag* (*RID* joint, *HingeJointFlag* flag) const
float	*hinge_joint_get_param* (*RID* joint, *HingeJointParam* param) const
void	*hinge_joint_set_flag* (*RID* joint, *HingeJointFlag* flag, *bool* enabled)
void	*hinge_joint_set_param* (*RID* joint, *HingeJointParam* param, *float* value)
RID	*joint_create_cone_twist* (*RID* body_A, *Transform* local_ref_A, *RID* body_B, *Transform* local_ref_B)
RID	*joint_create_generic_6dof* (*RID* body_A, *Transform* local_ref_A, *RID* body_B, *Transform* local_ref_B)
RID	*joint_create_hinge* (*RID* body_A, *Transform* hinge_A, *RID* body_B, *Transform* hinge_B)
RID	*joint_create_pin* (*RID* body_A, *Vector3* local_A, *RID* body_B, *Vector3* local_B)
RID	*joint_create_slider* (*RID* body_A, *Transform* local_ref_A, *RID* body_B, *Transform* local_ref_B)
int	*joint_get_solver_priority* (*RID* joint) const
JointType	*joint_get_type* (*RID* joint) const
void	*joint_set_solver_priority* (*RID* joint, *int* priority)
Vector3	*pin_joint_get_local_a* (*RID* joint) const
Vector3	*pin_joint_get_local_b* (*RID* joint) const
float	*pin_joint_get_param* (*RID* joint, *PinJointParam* param) const
void	*pin_joint_set_local_a* (*RID* joint, *Vector3* local_A)
void	*pin_joint_set_local_b* (*RID* joint, *Vector3* local_B)
void	*pin_joint_set_param* (*RID* joint, *PinJointParam* param, *float* value)
void	*set_active* (*bool* active)
RID	*shape_create* (*ShapeType* type)
Variant	*shape_get_data* (*RID* shape) const
ShapeType	*shape_get_type* (*RID* shape) const
void	*shape_set_data* (*RID* shape, *Variant* data)
float	*slider_joint_get_param* (*RID* joint, *SliderJointParam* param) const
void	*slider_joint_set_param* (*RID* joint, *SliderJointParam* param, *float* value)
RID	*space_create* ( )
PhysicsDirectSpaceState	*space_get_direct_state* (*RID* space)
float	*space_get_param* (*RID* space, *SpaceParameter* param) const
bool	*space_is_active* (*RID* space) const
void	*space_set_active* (*RID* space, *bool* active)
void	*space_set_param* (*RID* space, *SpaceParameter* param, *float* value)

29.300.3 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_LINEAR_MOTOR_TARGET_VELOCITY = 5** — The velocity that the joint's linear motor will attempt to reach.
- **G6DOF_JOINT_LINEAR_MOTOR_FORCE_LIMIT = 6** — The maximum force that the linear motor can apply while trying to reach the target velocity.
- **G6DOF_JOINT_ANGULAR_LOWER_LIMIT = 7** — The minimum rotation in negative direction to break loose and rotate around the axes.
- **G6DOF_JOINT_ANGULAR_UPPER_LIMIT = 8** — The minimum rotation in positive direction to break loose and rotate around the axes.
- **G6DOF_JOINT_ANGULAR_LIMIT_SOFTNESS = 9** — A factor that gets multiplied onto all rotations across the axes.
- **G6DOF_JOINT_ANGULAR_DAMPING = 10** — The amount of rotational damping across the axes. The lower, the more dampening occurs.
- **G6DOF_JOINT_ANGULAR_RESTITUTION = 11** — The amount of rotational restitution across the axes. The lower, the more restitution occurs.
- **G6DOF_JOINT_ANGULAR_FORCE_LIMIT = 12** — The maximum amount of force that can occur, when rotating around the axes.
- **G6DOF_JOINT_ANGULAR_ERP = 13** — 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 = 14** — Target speed for the motor at the axes.
- **G6DOF_JOINT_ANGULAR_MOTOR_FORCE_LIMIT = 15** — 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_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 **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.
- **G6DOF_JOINT_FLAG_ENABLE_LINEAR_MOTOR = 3** — If `set` there is a linear motor on this axis that targets a specific velocity.

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.

29.300.4 Description

Everything related to physics in 3D.

29.300.5 Member Function Description

- void **area_add_shape** (*RID* area, *RID* shape, *Transform* transform=Transform(1, 0, 0, 0, 1, 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, *AreaParameter* 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.

- *AreaSpaceOverrideMode* **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_area_monitor_callback** (*RID* area, *Object* receiver, *String* method)
- 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_monitorable** (*RID* area, *bool* monitorable)
- void **area_set_param** (*RID* area, *AreaParameter* 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, *AreaSpaceOverrideMode* 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** (*BodyMode* 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*.

- *BodyMode* **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, *BodyParameter* 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, *BodyState* state) const

Returns a body state.

- *bool* **body_is_axis_locked** (*RID* body, *BodyAxis* 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, *BodyAxis* 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, *BodyMode* 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, *BodyParameter* 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, *BodyState* state, *Variant* value)

Sets a body state (see BODY_STATE* constants).

- *float* **cone_twist_joint_get_param** (*RID* joint, *ConeTwistJointParam* param) const

Gets a cone_twist_joint parameter (see CONE_TWIST_JOINT* constants).

- void **cone_twist_joint_set_param** (*RID* joint, *ConeTwistJointParam* 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, Axis axis, *G6DOFJointAxisFlag* flag)

Gets a generic_6_DOF_joint flag (see G6DOF_JOINT_FLAG* constants).

- *float* **generic_6dof_joint_get_param** (*RID* joint, Axis axis, *G6DOFJointAxisParam* 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, Axis axis, *G6DOFJointAxisFlag* flag, *bool* enable)

Sets a generic_6_DOF_joint flag (see G6DOF_JOINT_FLAG* constants).

- void **generic_6dof_joint_set_param** (*RID* joint, Axis axis, *G6DOFJointAxisParam* param, *float* value)

Sets a generic_6_DOF_joint parameter (see G6DOF_JOINT* constants without the G6DOF_JOINT_FLAG*).

- *int* **get_process_info** (*ProcessInfo* process_info)

Returns an Info defined by the ProcessInfo input given.

- *bool* **hinge_joint_get_flag** (*RID* joint, *HingeJointFlag* flag) const

Gets a hinge_joint flag (see HINGE_JOINT_FLAG* constants).

- *float* **hinge_joint_get_param** (*RID* joint, *HingeJointParam* param) const

Gets a hinge_joint parameter (see HINGE_JOINT* constants without the HINGE_JOINT_FLAG*).

- void **hinge_joint_set_flag** (*RID* joint, *HingeJointFlag* flag, *bool* enabled)

Sets a hinge_joint flag (see HINGE_JOINT_FLAG* constants).

- void **hinge_joint_set_param** (*RID* joint, *HingeJointParam* 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.

- *JointType* **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, *PinJointParam* 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, *PinJointParam* 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** (*ShapeType* 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.

- *ShapeType* **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, *SliderJointParam* param) const

Gets a slider_joint parameter (see SLIDER_JOINT* constants).

- void **slider_joint_set_param** (*RID* joint, *SliderJointParam* 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, *SpaceParameter* 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, *SpaceParameter* param, *float* value)

Sets the value for a space parameter. A list of available parameters is on the SPACE_PARAM_* constants.

29.301 PhysicsShapeQueryParameters

Inherits: *Reference* < *Object*

Category: Core

29.301.1 Brief Description

29.301.2 Member Functions

void	*set_shape* (*Resource* shape)

29.301.3 Member Variables

- *int* **collision_mask**
- *Array* **exclude**
- *float* **margin**

- *RID* **shape_rid**
- *Transform* **transform**

29.301.4 Member Function Description

- void **set_shape** (*Resource* shape)

29.302 PhysicsShapeQueryResult

Inherits: *Reference* < *Object*

Category: Core

29.302.1 Brief Description

Result of a shape query in Physics2DServer.

29.302.2 Member Functions

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

29.302.3 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

29.303 PinJoint

Inherits: *Joint* < *Spatial* < *Node* < *Object*

Category: Core

29.303.1 Brief Description

Pin Joint for 3D Shapes.

29.303.2 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.

29.303.3 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.

29.303.4 Description

Pin Joint for 3D Rigid Bodies. It pins 2 bodies (rigid or static) together.

29.304 PinJoint2D

Inherits: *Joint2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.304.1 Brief Description

Pin Joint for 2D Shapes.

29.304.2 Member Variables

- *float* **softness** - The higher this value, the more the bond to the pinned partner can flex.

29.304.3 Description

Pin Joint for 2D Rigid Bodies. It pins two bodies (rigid or static) together.

29.305 Plane

Category: Built-In Types

29.305.1 Brief Description

Plane in hessian form.

29.305.2 Member Functions

Plane	Plane (*float* a, *float* b, *float* c, *float* d)
Plane	Plane (*Vector3* v1, *Vector3* v2, *Vector3* v3)
Plane	Plane (*Vector3* normal, *float* d)
Vector3	*center* ( )
float	*distance_to* (*Vector3* point)
Vector3	*get_any_point* ( )
bool	*has_point* (*Vector3* point, *float* epsilon=0.00001)
Vector3	*intersect_3* (*Plane* b, *Plane* c)
Vector3	*intersects_ray* (*Vector3* from, *Vector3* dir)
Vector3	*intersects_segment* (*Vector3* begin, *Vector3* end)
bool	*is_point_over* (*Vector3* point)
Plane	*normalized* ( )
Vector3	*project* (*Vector3* point)

29.305.3 Member Variables

- *float* **d**
- *Vector3* **normal**
- *float* **x**
- *float* **y**
- *float* **z**

29.305.4 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.

29.305.5 Member Function Description

- *Plane* **Plane** (*float* a, *float* b, *float* c, *float* d)

Creates a plane from the four 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.

29.306 PlaneMesh

Inherits: *PrimitiveMesh* **<** *Mesh* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.306.1 Brief Description

Class representing a planar *PrimitiveMesh*.

29.306.2 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.

29.306.3 Description

Class representing a planar *PrimitiveMesh*. This flat mesh does not have a thickness.

29.307 PlaneShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.307.1 Brief Description

29.307.2 Member Variables

- *Plane* **plane**

29.308 PluginScript

Inherits: *Script* < *Resource* < *Reference* < *Object*

Category: Core

29.308.1 Brief Description

29.309 Polygon2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.309.1 Brief Description

A 2D polygon.

29.309.2 Member Functions

void	*add_bone* (*NodePath* path, *PoolRealArray* weights)
void	*clear_bones* ( )
void	*erase_bone* (*int* index)
int	*get_bone_count* ( ) const
NodePath	*get_bone_path* (*int* index) const
PoolRealArray	*get_bone_weights* (*int* index) const
void	*set_bone_path* (*int* index, *NodePath* path)
void	*set_bone_weights* (*int* index, *PoolRealArray* weights)

29.309.3 Member Variables

- *bool* **antialiased** - If `true` polygon edges will be anti-aliased. Default value: `false`.
- *Array* **bones**
- *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.
- *NodePath* **skeleton**
- *PoolIntArray* **splits**
- *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`.

29.309.4 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.

29.309.5 Member Function Description

- void **add_bone** (*NodePath* path, *PoolRealArray* weights)
- void **clear_bones** ( )
- void **erase_bone** (*int* index)
- *int* **get_bone_count** ( ) const
- *NodePath* **get_bone_path** (*int* index) const
- *PoolRealArray* **get_bone_weights** (*int* index) const
- void **set_bone_path** (*int* index, *NodePath* path)
- void **set_bone_weights** (*int* index, *PoolRealArray* weights)

29.310 PolygonPathFinder

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.310.1 Brief Description

29.310.2 Member Functions

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)

29.310.3 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)

29.311 PoolByteArray

Category: Built-In Types

29.311.1 Brief Description

Raw byte array.

29.311.2 Member Functions

PoolByteArray	*PoolByteArray* (*Array* from)
void	*append* (*int* byte)
void	*append_array* (*PoolByteArray* array)
PoolByteArray	*compress* (*int* compression_mode=0)
PoolByteArray	*decompress* (*int* buffer_size, *int* compression_mode=0)
String	*get_string_from_ascii* ( )
String	*get_string_from_utf8* ( )
int	*insert* (*int* idx, *int* byte)
void	*invert* ( )
void	*push_back* (*int* byte)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *int* byte)
int	*size* ( )
PoolByteArray	*subarray* (*int* from, *int* to)

29.311.3 Description

Raw byte array. Contains bytes. Optimized for memory usage, can't fragment the memory. Note that this type is passed by value and not by reference.

29.311.4 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.

29.312 PoolColorArray

Category: Built-In Types

29.312.1 Brief Description

Array of Colors

29.312.2 Member Functions

PoolColorArray	*PoolColorArray* (*Array* from)
void	*append* (*Color* color)
void	*append_array* (*PoolColorArray* array)
int	*insert* (*int* idx, *Color* color)
void	*invert* ( )
void	*push_back* (*Color* color)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *Color* color)
int	*size* ( )

29.312.3 Description

Array of Color, Contains colors. Optimized for memory usage, can't fragment the memory. Note that this type is passed by value and not by reference.

29.312.4 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.

29.313 PoolIntArray

Category: Built-In Types

29.313.1 Brief Description

Integer Array.

29.313.2 Member Functions

PoolIntArray	*PoolIntArray* (*Array* from)
void	*append* (*int* integer)
void	*append_array* (*PoolIntArray* array)
int	*insert* (*int* idx, *int* integer)
void	*invert* ( )
void	*push_back* (*int* integer)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *int* integer)
int	*size* ( )

29.313.3 Description

Integer Array. Contains integers. Optimized for memory usage, can't fragment the memory. Note that this type is passed by value and not by reference.

29.313.4 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.

29.314 PoolRealArray

Category: Built-In Types

29.314.1 Brief Description

Real Array.

29.314.2 Member Functions

PoolRealArray	*PoolRealArray* (*Array* from)
void	*append* (*float* value)
void	*append_array* (*PoolRealArray* array)
int	*insert* (*int* idx, *float* value)
void	*invert* ( )
void	*push_back* (*float* value)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *float* value)
int	*size* ( )

29.314.3 Description

Real Array. Array of floating point values. Can only contain floats. Optimized for memory usage, can't fragment the memory. Note that this type is passed by value and not by reference.

29.314.4 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.

29.315 PoolStringArray

Category: Built-In Types

29.315.1 Brief Description

String Array.

29.315.2 Member Functions

PoolStringArray	*PoolStringArray* (*Array* from)
void	*append* (*String* string)
void	*append_array* (*PoolStringArray* array)
int	*insert* (*int* idx, *String* string)
void	*invert* ( )
String	*join* (*String* delimiter)
void	*push_back* (*String* string)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *String* string)
int	*size* ( )

29.315.3 Description

String Array. Array of strings. Can only contain strings. Optimized for memory usage, can't fragment the memory. Note that this type is passed by value and not by reference.

29.315.4 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.

29.316 PoolVector2Array

Category: Built-In Types

29.316.1 Brief Description

An Array of Vector2.

29.316.2 Member Functions

PoolVector2Array	*PoolVector2Array* (*Array* from)
void	*append* (*Vector2* vector2)
void	*append_array* (*PoolVector2Array* array)
int	*insert* (*int* idx, *Vector2* vector2)
void	*invert* ( )
void	*push_back* (*Vector2* vector2)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *Vector2* vector2)
int	*size* ( )

29.316.3 Description

An Array specifically designed to hold Vector2. Note that this type is passed by value and not by reference.

29.316.4 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.

29.317 PoolVector3Array

Category: Built-In Types

29.317.1 Brief Description

An Array of Vector3.

29.317.2 Member Functions

PoolVector3Array	*PoolVector3Array* (*Array* from)
void	*append* (*Vector3* vector3)
void	*append_array* (*PoolVector3Array* array)
int	*insert* (*int* idx, *Vector3* vector3)
void	*invert* ( )
void	*push_back* (*Vector3* vector3)
void	*remove* (*int* idx)
void	*resize* (*int* idx)
void	*set* (*int* idx, *Vector3* vector3)
int	*size* ( )

29.317.3 Description

An Array specifically designed to hold Vector3. Note that this type is passed by value and not by reference.

29.317.4 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.

29.318 Popup

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *PopupPanel*, *PopupDialog*, *PopupMenu*, *WindowDialog*

Category: Core

29.318.1 Brief Description

Base container control for popups and dialogs.

29.318.2 Member Functions

void	*popup* (*Rect2* bounds=Rect2(0, 0, 0, 0))
void	*popup_centered* (*Vector2* size=Vector2(0, 0))
void	*popup_centered_minsize* (*Vector2* minsize=Vector2(0, 0))
void	*popup_centered_ratio* (*float* ratio=0.75)

29.318.3 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.

29.318.4 Member Variables

- *bool* **popup_exclusive** - If `true` the popup will not be hidden when a click event occurs outside of it, or when it receives the `ui_cancel` action event.

29.318.5 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.

29.318.6 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.

29.318.7 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.

29.319 PopupDialog

Inherits: *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.319.1 Brief Description

Base class for Popup Dialogs.

29.319.2 Description

PopupDialog is a base class for popup dialogs, along with *WindowDialog*.

29.320 PopupMenu

Inherits: *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.320.1 Brief Description

PopupMenu displays a list of options.

29.320.2 Member Functions

void	*add_check_item* (*String* label, *int* id=-1, *int* accel=0)
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)
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)
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)
void	*add_radio_check_item* (*String* label, *int* id=-1, *int* accel=0)
void	*add_radio_check_shortcut* (*ShortCut* shortcut, *int* id=-1, *bool* global=false)
void	*add_separator* ( )
void	*add_shortcut* (*ShortCut* shortcut, *int* id=-1, *bool* global=false)
void	*add_submenu_item* (*String* label, *String* submenu, *int* id=-1)
void	*clear* ( )
int	*get_item_accelerator* (*int* idx) const
int	*get_item_count* ( ) const
Texture	*get_item_icon* (*int* idx) const
int	*get_item_id* (*int* idx) const
int	*get_item_index* (*int* id) const
Variant	*get_item_metadata* (*int* idx) const
ShortCut	*get_item_shortcut* (*int* idx) const
String	*get_item_submenu* (*int* idx) const
String	*get_item_text* (*int* idx) const
String	*get_item_tooltip* (*int* idx) const
bool	*is_item_checkable* (*int* idx) const
bool	*is_item_checked* (*int* idx) const
bool	*is_item_disabled* (*int* idx) const
bool	*is_item_radio_checkable* (*int* idx) const
bool	*is_item_separator* (*int* idx) const
void	*remove_item* (*int* idx)
void	*set_item_accelerator* (*int* idx, *int* accel)
void	*set_item_as_checkable* (*int* idx, *bool* enable)
void	*set_item_as_radio_checkable* (*int* idx, *bool* enable)
void	*set_item_as_separator* (*int* idx, *bool* enable)
void	*set_item_checked* (*int* idx, *bool* checked)
void	*set_item_disabled* (*int* idx, *bool* disabled)
void	*set_item_icon* (*int* idx, *Texture* icon)
void	*set_item_id* (*int* idx, *int* id)
void	*set_item_metadata* (*int* idx, *Variant* metadata)
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)
void	*set_item_text* (*int* idx, *String* text)
void	*set_item_tooltip* (*int* idx, *String* tooltip)
void	*toggle_item_checked* (*int* idx)

Continued on next page

Table 19 – continued from previous page

void	*toggle_item_multistate* (*int* idx)

29.320.3 Signals

- **id_focused** (*int* ID)

This event is emitted when user navigated to an item of some id using `ui_up` or `ui_down` action.

- **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.

29.320.4 Member Variables

- *bool* **hide_on_checkable_item_selection**
- *bool* **hide_on_item_selection**
- *bool* **hide_on_state_item_selection**

29.320.5 Description

PopupMenu is the typical Control that displays a list of options. They are popular in toolbars or context menus.

29.320.6 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_radio_check_item** (*String* label, *int* id=-1, *int* accel=0)

The same as *add_check_item* but the inserted item will look as a radio button. Remember this is just cosmetic and you have to add the logic for checking/unchecking items in radio groups.

- void **add_radio_check_shortcut** (*ShortCut* shortcut, *int* id=-1, *bool* global=false)
- 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 is 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" is checkable in some way, i.e., whether has a checkbox or radio button. Note that checkable items just display a checkmark or radio button, but don't have any built-in checking behavior and must be checked/unchecked manually.

- *bool* **is_item_checked** (*int* idx) const

Return whether the item at index "idx" is checked.

- *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_radio_checkable** (*int* idx) const

Return whether the item at index "idx" has radio-button-style checkability. Remember this is just cosmetic and you have to add the logic for checking/unchecking items in radio groups.

- *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_radio_checkable** (*int* idx, *bool* enable)

The same as *set_item_as_checkable* but placing a radio button in case of enabling. If used for disabling, it's the same.

Remember this is just cosmetic and you have to add the logic for checking/unchecking items in radio groups.

- 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)

29.321 PopupPanel

Inherits: *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.321.1 Brief Description

Class for displaying popups with a panel background.

29.321.2 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*.

29.322 Position2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.322.1 Brief Description

Generic 2D Position hint for editing.

29.322.2 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.

29.323 Position3D

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.323.1 Brief Description

Generic 3D Position hint for editing.

29.323.2 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.

29.324 PrimitiveMesh

Inherits: *Mesh* < *Resource* < *Reference* < *Object*

Inherited By: *PlaneMesh*, *CubeMesh*, *SphereMesh*, *CylinderMesh*, *CapsuleMesh*, *QuadMesh*, *PrismMesh*

Category: Core

29.324.1 Brief Description

Base class for all primitive meshes. Handles applying a *Material* to a primitive mesh.

29.324.2 Member Functions

Array	*get_mesh_arrays* **( )** const

29.324.3 Member Variables

- *AABB* **custom_aabb**
- *bool* **flip_faces**
- *Material* **material** - The current *Material* of the primitive mesh.

29.324.4 Description

Base class for all primitive meshes. Handles applying a *Material* to a primitive mesh.

29.324.5 Member Function Description

- *Array* **get_mesh_arrays ( )** const

29.325 PrismMesh

Inherits: *PrimitiveMesh* < *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.325.1 Brief Description

Class representing a prism-shaped *PrimitiveMesh*.

29.325.2 Member Variables

- *float* **left_to_right** - Displacement of of the upper edge along the x-axis. 0.0 positions edge straight above the bottome 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.

29.325.3 Description

Class representing a prism-shaped *PrimitiveMesh*.

29.326 ProceduralSky

Inherits: *Sky* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.326.1 Brief Description

Type of *Sky* that is generated procedurally based on user input parameters.

29.326.2 Member Variables

- *Color* **ground_bottom_color** - Color of the ground at the bottom.
- *float* **ground_curve** - How quickly the *ground_horizon_color* fades into the *ground_bottom_color*.
- *float* **ground_energy** - Amount of energy contribution from the ground.
- *Color* **ground_horizon_color** - Color of the ground at the horizon.
- *float* **sky_curve** - How quickly the *sky_horizon_color* fades into the *sky_top_color*.
- *float* **sky_energy** - Amount of energy contribution from the sky.
- *Color* **sky_horizon_color** - Color of the sky at the horizon.
- *Color* **sky_top_color** - Color of the sky at the top.
- *float* **sun_angle_max** - Distance from center of sun where it fades out completely.
- *float* **sun_angle_min** - Distance from sun where it goes from solid to starting to fade.
- *Color* **sun_color** - Color of the sun.
- *float* **sun_curve** - How quickly the sun fades away between *sun_angle_min* and *sun_angle_max*
- *float* **sun_energy** - Amount of energy contribution from the sun.
- *float* **sun_latitude** - The suns height using polar coordinates.

- *float* **sun_longitude** - The direction of the sun using polar coordinates.
- *TextureSize* **texture_size** - Size of *Texture* that the ProceduralSky will generate.

29.326.3 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**

29.326.4 Description

ProceduralSky provides a way to create an effective background quickly by defining procedural parameters for the sun, the sky and the ground. The sky and ground are very similar, they are defined by a color at the horizon, another color, and finally an easing curve to interpolate between these two colors. Similarly the sun is described by a position in the sky, a color, and an easing curve. However, the sun also defines a minimum and maximum angle, these two values define at what distance the easing curve begins and ends from the sun, and thus end up defining the size of the sun in the sky.

The ProceduralSky is updated on the CPU after the parameters change and stored in a texture and then displayed as a background in the scene. This makes it relatively unsuitable for realtime updates during gameplay. But with a small texture size it is still feasible to update relatively frequently becuase it is updated on a background thread when multi-threading is available.

29.327 ProgressBar

Inherits: *Range* **<** *Control* **<** *CanvasItem* **<** *Node* **<** *Object*

Category: Core

29.327.1 Brief Description

General purpose progress bar.

29.327.2 Member Variables

- *bool* **percent_visible**

29.327.3 Description

General purpose progress bar. Shows fill percentage from right to left.

29.328 ProjectSettings

Inherits: *Object*

Category: Core

29.328.1 Brief Description

Contains global variables accessible from everywhere.

29.328.2 Member Functions

void	*add_property_info* (*Dictionary* hint)
void	*clear* (*String* name)
int	*get_order* (*String* name) const
Variant	*get_setting* (*String* name) const
String	*globalize_path* (*String* path) const
bool	*has_setting* (*String* name) const
bool	*load_resource_pack* (*String* pack)
String	*localize_path* (*String* path) const
bool	*property_can_revert* (*String* name)
Variant	*property_get_revert* (*String* name)
Error	*save* ( )
Error	*save_custom* (*String* file)
void	*set_initial_value* (*String* name, *Variant* value)
void	*set_order* (*String* name, *int* position)
void	*set_setting* (*String* name, *Variant* value)

29.328.3 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.

29.328.4 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)
- *Error* **save** ( )
- *Error* **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)

29.329 ProximityGroup

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.329.1 Brief Description

General purpose proximity-detection node.

29.329.2 Member Functions

void	broadcast (*String* name, *Variant* parameters)

29.329.3 Signals

- **broadcast** (*String* group_name, *Array* parameters)

29.329.4 Member Variables

- *DispatchMode* **dispatch_mode**
- *Vector3* **grid_radius**
- *String* **group_name**

29.329.5 Enums

enum **DispatchMode**

- **MODE_PROXY = 0**
- **MODE_SIGNAL = 1**

29.329.6 Description

General purpose proximity-detection node.

29.329.7 Member Function Description

- void **broadcast** (*String* name, *Variant* parameters)

29.330 ProxyTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.330.1 Brief Description

29.330.2 Member Variables

- *Texture* **base**

29.331 QuadMesh

Inherits: *PrimitiveMesh* < *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.331.1 Brief Description

Class representing a square mesh.

29.331.2 Member Variables

- *Vector2* **size**

29.331.3 Description

Class representing a square mesh with size (2,2,0). Consider using a *PlaneMesh* if you require a differently sized plane.

29.332 Quat

Category: Built-In Types

29.332.1 Brief Description

Quaternion.

29.332.2 Member Functions

Quat	Quat (*Basis* from)
Quat	Quat (*Vector3* euler)
Quat	Quat (*Vector3* axis, *float* angle)
Quat	Quat (*float* x, *float* y, *float* z, *float* w)
Quat	*cubic_slerp* (*Quat* b, *Quat* pre_a, *Quat* post_b, *float* t)
float	*dot* (*Quat* b)
Vector3	*get_euler* ( )
Quat	*inverse* ( )
bool	*is_normalized* ( )
float	*length* ( )
float	*length_squared* ( )
Quat	*normalized* ( )
void	*set_axis_angle* (*Vector3* axis, *float* angle)
void	*set_euler* (*Vector3* euler)
Quat	*slerp* (*Quat* b, *float* t)
Quat	*slerpni* (*Quat* b, *float* t)
Vector3	*xform* (*Vector3* v)

29.332.3 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`

29.332.4 Description

A unit quaternion used for representing 3D rotations.

It is similar to *Basis*, which implements matrix representation of rotations, and can be parametrized using both an axis-angle pair or Euler angles. But due to its compactness and the way it is stored in memory, certain operations (obtaining axis-angle and performing SLERP, in particular) are more efficient and robust against floating point errors.

Quaternions need to be (re)normalized.

29.332.5 Member Function Description

- *Quat* **Quat** (*Basis* from)

Returns the rotation matrix corresponding to the given quaternion.

- *Quat* **Quat** (*Vector3* euler)

Returns a quaternion that will perform a rotation specified by Euler angles (in the YXZ convention: first Z, then X, and Y last), given in the vector format as (X-angle, Y-angle, Z-angle).

- *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 (***float* x, *float* y, *float* z, *float* w **)**

Returns a quaternion defined by these values.

- *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.

- *Vector3* **get_euler ( )**

Return Euler angles (in the YXZ convention: first Z, then X, and Y last) corresponding to the rotation represented by the unit quaternion. Returned vector contains the rotation angles in the format (X-angle, Y-angle, Z-angle).

- *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.

- void **set_axis_angle (***Vector3* axis, *float* angle **)**

Set the quaternion to a rotation which rotates around axis by the specified angle, in radians. The axis must be a normalized vector.

- void **set_euler (***Vector3* euler **)**

Set the quaternion to a rotation specified by Euler angles (in the YXZ convention: first Z, then X, and Y last), given in the vector format as (X-angle, Y-angle, Z-angle).

- *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 quaterion without checking if the rotation path is not bigger than 90°.

- *Vector3* **xform (***Vector3* v **)**

Transforms the vector `v` by this quaternion.

29.333 Range

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *SpinBox*, *ScrollBar*, *ProgressBar*, *TextureProgress*, *Slider*

Category: Core

29.333.1 Brief Description

Abstract base class for range-based controls.

29.333.2 Member Functions

void	*share* (*Node* with)
void	*unshare* ( )

29.333.3 Signals

- **changed** ( )

This signal is emitted when min, max, range or step change.

- **value_changed** (*float* value)

This signal is emitted when value changes.

29.333.4 Member Variables

- *bool* **allow_greater**
- *bool* **allow_lesser**
- *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.

29.333.5 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*.

29.333.6 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.

29.334 RayCast

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.334.1 Brief Description

Query the closest object intersecting a ray.

29.334.2 Member Functions

void	*add_exception* (*Object* node)
void	*add_exception_rid* (*RID* rid)
void	*clear_exceptions* ( )
void	*force_raycast_update* ( )
Object	*get_collider* ( ) const
int	*get_collider_shape* ( ) const
bool	*get_collision_mask_bit* (*int* bit) const
Vector3	*get_collision_normal* ( ) const
Vector3	*get_collision_point* ( ) const
bool	*is_colliding* ( ) const
void	*remove_exception* (*Object* node)
void	*remove_exception_rid* (*RID* rid)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.334.3 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`.

29.334.4 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.

29.334.5 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.

29.335 RayCast2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.335.1 Brief Description

Query the closest object intersecting a ray.

29.335.2 Member Functions

void	*add_exception* (*Object* node)
void	*add_exception_rid* (*RID* rid)
void	*clear_exceptions* ( )
void	*force_raycast_update* ( )
Object	*get_collider* ( ) const
int	*get_collider_shape* ( ) const
bool	*get_collision_mask_bit* (*int* bit) const
Vector2	*get_collision_normal* ( ) const
Vector2	*get_collision_point* ( ) const
bool	*is_colliding* ( ) const
void	*remove_exception* (*Object* node)
void	*remove_exception_rid* (*RID* rid)
void	*set_collision_mask_bit* (*int* bit, *bool* value)

29.335.3 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`.

29.335.4 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.

29.335.5 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.

29.336 RayShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.336.1 Brief Description

Ray shape for 3D collisions.

29.336.2 Member Variables

- *float* **length** - The ray's length.
- *bool* **slips_on_slope** - If `true` allow the shape to return the correct normal. Default value: `false`.

29.336.3 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.

29.337 RayShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.337.1 Brief Description

Ray shape for 2D collisions.

29.337.2 Member Variables

- *float* **length** - The ray's length.
- *bool* **slips_on_slope** - If `true` allow the shape to return the correct normal. Default value: `false`.

29.337.3 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.

29.338 Rect2

Category: Built-In Types

29.338.1 Brief Description

2D Axis-aligned bounding box.

29.338.2 Member Functions

Rect2	Rect2 (*Vector2* position, *Vector2* size)
Rect2	Rect2 (*float* x, *float* y, *float* width, *float* height)
Rect2	*abs* ( )
Rect2	*clip* (*Rect2* b)
bool	*encloses* (*Rect2* b)
Rect2	*expand* (*Vector2* to)
float	*get_area* ( )
Rect2	*grow* (*float* by)
Rect2	*grow_individual* (*float* left, *float* top, *float* right, *float* bottom)
Rect2	*grow_margin* (*int* margin, *float* by)
bool	*has_no_area* ( )
bool	*has_point* (*Vector2* point)
bool	*intersects* (*Rect2* b)
Rect2	*merge* (*Rect2* b)

29.338.3 Member Variables

- *Vector2* **end** - Ending corner.
- *Vector2* **position** - Position (starting corner).
- *Vector2* **size** - Size from position to end.

29.338.4 Description

Rect2 consists of a position, a size, and several utility functions. It is typically used for fast overlap tests.

29.338.5 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`.

29.339 RectangleShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.339.1 Brief Description

Rectangle shape for 2D collisions.

29.339.2 Member Variables

- *Vector2* **extents** - The rectangle's half extents. The width and height of this shape is twice the half extents.

29.339.3 Description

Rectangle shape for 2D collisions. This shape is useful for modeling box-like 2D objects.

29.340 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*, *EditorInspectorPlugin*, *Resource*, *KinematicCollision*, *SurfaceTool*, *JSONParseResult*, *SpatialVelocityTracker*, *EditorResourcePreviewGenerator*, *Physics2DShapeQueryParameters*, *EditorExportPlugin*, *ARVRInterface*, *EditorScript*, *MultiplayerAPI*, *Mutex*, *PacketPeer*, *Semaphore*, *XMLParser*, *EditorImportPlugin*, *Directory*, *WeakRef*, *GDScriptFunctionState*, *Marshalls*, *SceneState*, *PCKPacker*, *MeshDataTool*, *AStar*, *ResourceImporter*, *EditorResourceConversionPlugin*, *SceneTreeTimer*, *Thread*, *ResourceInteractiveLoader*, *PackedDataContainerRef*, *KinematicCollision2D*, *GDNative*, *PhysicsShapeQueryParameters*

Category: Core

29.340.1 Brief Description

Base class for anything that keeps a reference count.

29.340.2 Member Functions

bool	*init_ref* ( )
bool	*reference* ( )
bool	*unreference* ( )

29.340.3 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.

29.340.4 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.

29.341 ReferenceRect

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.341.1 Brief Description

Reference frame for GUI.

29.341.2 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.

29.342 ReflectionProbe

Inherits: *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.342.1 Brief Description

29.342.2 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**

29.342.3 Enums

enum **UpdateMode**

- **UPDATE_ONCE = 0**
- **UPDATE_ALWAYS = 1**

29.343 RegEx

Inherits: *Reference* **<** *Object*

Category: Core

29.343.1 Brief Description

Class for searching text for patterns using regular expressions.

29.343.2 Member Functions

void	*clear* ( )
Error	*compile* (*String* pattern)
int	*get_group_count* ( ) const
Array	*get_names* ( ) const
String	*get_pattern* ( ) const
bool	*is_valid* ( ) const
RegExMatch	*search* (*String* subject, *int* offset=0, *int* end=-1) const
Array	*search_all* (*String* subject, *int* offset=0, *int* end=-1) const
String	*sub* (*String* subject, *String* replacement, *bool* all=false, *int* offset=0, *int* end=-1) const

29.343.3 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
```

29.343.4 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.

- *Error* **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.

29.344 RegExMatch

Inherits: *Reference* < *Object*

Category: Core

29.344.1 Brief Description

Contains the results of a regex search.

29.344.2 Member Functions

int	*get_end* (*Variant* name=0) const
int	*get_group_count* ( ) const
int	*get_start* (*Variant* name=0) const
String	*get_string* (*Variant* name=0) const

29.344.3 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.

29.344.4 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.

29.344.5 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.

29.345 RemoteTransform

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.345.1 Brief Description

RemoteTransform leads the *Transform* of another *Spatial* derived Node in the scene.

29.345.2 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`.

29.345.3 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.

29.346 RemoteTransform2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.346.1 Brief Description

RemoteTransform2D leads the *Transform2D* of another *CanvasItem* derived Node in the scene.

29.346.2 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`.

29.346.3 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.

29.347 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

29.347.1 Brief Description

Base class for all resources.

29.347.2 Member Functions

void	*_setup_local_to_scene* ( ) virtual
Resource	*duplicate* (*bool* subresources=false) const
Node	*get_local_scene* ( ) const
RID	*get_rid* ( ) const
void	*setup_local_to_scene* ( )
void	*take_over_path* (*String* path)

29.347.3 Signals

- **changed** ( )

29.347.4 Member Variables

- *bool* **resource_local_to_scene**
- *String* **resource_name**

- *String* **resource_path**

29.347.5 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.

29.347.6 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.

29.348 ResourceImporter

Inherits: *Reference* < *Object*

Inherited By: *ResourceImporterTheora*, *ResourceImporterOGGVorbis*, *ResourceImporterWebm*

Category: Core

29.348.1 Brief Description

29.349 ResourceImporterOGGVorbis

Inherits: *ResourceImporter* < *Reference* < *Object*

Category: Core

29.349.1 Brief Description

29.350 ResourceImporterTheora

Inherits: *ResourceImporter* < *Reference* < *Object*

Category: Core

29.350.1 Brief Description

29.351 ResourceImporterWebm

Inherits: *ResourceImporter* < *Reference* < *Object*

Category: Core

29.351.1 Brief Description

29.352 ResourceInteractiveLoader

Inherits: *Reference* < *Object*

Category: Core

29.352.1 Brief Description

Interactive Resource Loader.

29.352.2 Member Functions

Resource	*get_resource* ( )
int	*get_stage* ( ) const
int	*get_stage_count* ( ) const
Error	*poll* ( )
Error	*wait* ( )

29.352.3 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.

29.352.4 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.

- *Error* **poll** ( )

Poll the load. If OK is returned, this means poll will have to be called again. If ERR_FILE_EOF is returned, them the load has finished and the resource can be obtained by calling *get_resource*.

- *Error* **wait ( )**

29.353 ResourceLoader

Inherits: *Object*

Category: Core

29.353.1 Brief Description

Resource Loader.

29.353.2 Member Functions

PoolStringArray	*get_dependencies* (*String* path)
PoolStringArray	*get_recognized_extensions_for_type* (*String* 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="")
void	*set_abort_on_missing_resources* (*bool* abort)

29.353.3 Description

Resource Loader. This is a static object accessible as `ResourceLoader`. GDScript has a simplified load() function, though.

29.353.4 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.

29.354 ResourcePreloader

Inherits: *Node* < *Object*

Category: Core

29.354.1 Brief Description

Resource Preloader Node.

29.354.2 Member Functions

void	*add_resource* (*String* name, *Resource* resource)
Resource	*get_resource* (*String* name) const
PoolStringArray	*get_resource_list* ( ) const
bool	*has_resource* (*String* name) const
void	*remove_resource* (*String* name)
void	*rename_resource* (*String* name, *String* newname)

29.354.3 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.

29.354.4 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.

29.355 ResourceSaver

Inherits: *Object*

Category: Core

29.355.1 Brief Description

Resource Saving Interface.

29.355.2 Member Functions

PoolStringArray	*get_recognized_extensions* (*Resource* type)
Error	*save* (*String* path, *Resource* resource, *int* flags=0)

29.355.3 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**

29.355.4 Description

Resource Saving Interface. This interface is used for saving resources to disk.

29.355.5 Member Function Description

- *PoolStringArray* **get_recognized_extensions** (*Resource* type)

Return the list of extensions available for saving a resource of a given type.

- *Error* **save** (*String* path, *Resource* resource, *int* flags=0)

Save a resource to disk, to a given path.

29.356 RichTextLabel

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.356.1 Brief Description

Label that displays rich text.

29.356.2 Member Functions

void	*add_image* (*Texture* image)
void	*add_text* (*String* text)
Error	*append_bbcode* (*String* bbcode)
void	*clear* ( )
int	*get_content_height* ( )
int	*get_line_count* ( ) const
int	*get_total_character_count* ( ) const
VScrollBar	*get_v_scroll* ( )
int	*get_visible_line_count* ( ) const
void	*newline* ( )
Error	*parse_bbcode* (*String* bbcode)
void	*pop* ( )
void	*push_align* (*Align* align)
void	*push_cell* ( )
void	*push_color* (*Color* color)
void	*push_font* (*Font* font)
void	*push_indent* (*int* level)
void	*push_list* (*ListType* type)
void	*push_meta* (*Variant* data)
void	*push_table* (*int* columns)
void	*push_underline* ( )
bool	*remove_line* (*int* line)
void	*scroll_to_line* (*int* line)
void	*set_table_column_expand* (*int* column, *bool* expand, *int* ratio)

29.356.3 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.

29.356.4 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 *bbcode_text*.

- *int* **visible_characters** - The restricted number of characters to display in the label.

29.356.5 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**

29.356.6 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.

29.356.7 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.

- *Error* **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_content_height** ( )

Returns the height of the content.

- *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.

- *Error* **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** (*Align* 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** (*ListType* 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.

29.357 RID

Category: Built-In Types

29.357.1 Brief Description

Handle for a *Resource*'s unique ID.

29.357.2 Member Functions

RID	*RID* (*Object* from)
int	*get_id* ( )

29.357.3 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*.

29.357.4 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.

29.358 RigidBody

Inherits: *PhysicsBody* < *CollisionObject* < *Spatial* < *Node* < *Object*

Inherited By: *VehicleBody*

Category: Core

29.358.1 Brief Description

Physics Body whose position is determined through physics simulation in 3D space.

29.358.2 Member Functions

void	*_integrate_forces* (*PhysicsDirectBodyState* state) virtual
void	*apply_impulse* (*Vector3* position, *Vector3* impulse)
void	*apply_torque_impulse* (*Vector3* impulse)
Array	*get_colliding_bodies* ( ) const
void	*set_axis_velocity* (*Vector3* axis_velocity)

29.358.3 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.

29.358.4 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".

29.358.5 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.

29.358.6 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.

A RigidBody has 4 behavior *mode*s: Rigid, Static, Character, and Kinematic.

Note: Don't change a 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. 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*.

29.358.7 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.

- void **apply_torque_impulse** (*Vector3* impulse)

Apply a torque impulse (which will be affected by the body mass and shape). This will rotate the body around the passed in vector.

- *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.

29.359 RigidBody2D

Inherits: *PhysicsBody2D* < *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.359.1 Brief Description

A body that is controlled by the 2D physics engine.

29.359.2 Member Functions

void	*_integrate_forces* (*Physics2DDirectBodyState* state) virtual
void	*add_force* (*Vector2* offset, *Vector2* force)
void	*apply_impulse* (*Vector2* offset, *Vector2* impulse)
Array	*get_colliding_bodies* ( ) const
void	*set_axis_velocity* (*Vector2* axis_velocity)
bool	*test_motion* (*Vector2* motion, *bool* infinite_inertia=true, *float* margin=0.08, *Physics2DTestMotionResult* result=null)

29.359.3 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.

29.359.4 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 *Area2D*s. 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".

29.359.5 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.

29.359.6 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 *mode*s: Rigid, Static, Character, and Kinematic.

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*.

29.359.7 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, *bool* infinite_inertia=true, *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).

29.360 SceneState

Inherits: *Reference* < *Object*

Category: Core

29.360.1 Brief Description

A script interface to a scene file's data.

29.360.2 Member Functions

Array	*get_connection_binds* (*int* idx) const
int	*get_connection_count* ( ) const
int	*get_connection_flags* (*int* idx) const
String	*get_connection_method* (*int* idx) const
String	*get_connection_signal* (*int* idx) const
NodePath	*get_connection_source* (*int* idx) const
NodePath	*get_connection_target* (*int* idx) const
int	*get_node_count* ( ) const
PoolStringArray	*get_node_groups* (*int* idx) const
int	*get_node_index* (*int* idx) const
PackedScene	*get_node_instance* (*int* idx) const
String	*get_node_instance_placeholder* (*int* idx) const
String	*get_node_name* (*int* idx) const
NodePath	*get_node_owner_path* (*int* idx) const
NodePath	*get_node_path* (*int* idx, *bool* for_parent=false) const
int	*get_node_property_count* (*int* idx) const
String	*get_node_property_name* (*int* idx, *int* prop_idx) const
Variant	*get_node_property_value* (*int* idx, *int* prop_idx) const
String	*get_node_type* (*int* idx) const
bool	*is_node_instance_placeholder* (*int* idx) const

29.360.3 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.

29.360.4 Description

Maintains a list of resources, nodes, exported and overridden properties, and built-in scripts associated with a scene.

29.360.5 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*.

29.361 SceneTree

Inherits: *MainLoop* < *Object*

Category: Core

29.361.1 Brief Description

SceneTree manages a hierarchy of nodes.

29.361.2 Member Functions

Variant	*call_group* (*String* group, *String* method) vararg
Variant	*call_group_flags* (*int* flags, *String* group, *String* method) vararg
Error	*change_scene* (*String* path)
Error	*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
bool	*is_input_handled* ( )
bool	*is_network_server* ( ) const
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* ( )
Error	*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* (*StretchMode* mode, *StretchAspect* aspect, *Vector2* minsize, *float* shrink=1)

29.361.3 Signals

- **connected_to_server** ( )

Emitted whenever this SceneTree's *network_peer* successfully connected to a server. Only emitted on clients.

- **connection_failed** ( )

Emitted whenever this SceneTree's *network_peer* fails to establish a connection to a server. Only emitted on clients.

- **files_dropped** (*PoolStringArray* files, *int* screen)

Emitted whenever files are drag-and-dropped onto the window.

- **idle_frame ( )**

Emitted immediately before *Node._process* is called on every node in the SceneTree.

- **network_peer_connected (*int* id)**

Emitted whenever this SceneTree's *network_peer* connects with a new peer. ID is the peer ID of the new peer. Clients get notified when other clients connect to the same server. Upon connecting to a server, a client also receives this signal for the server (with ID being 1).

- **network_peer_disconnected (*int* id)**

Emitted whenever this SceneTree's *network_peer* disconnects from a peer. Clients get notified when other clients disconnect from the same server.

- **node_added (*Object* node)**

Emitted whenever a node is added to the SceneTree.

- **node_configuration_warning_changed (*Object* node)**

Emitted when a node's configuration changed. Only emitted in tool mode.

- **node_removed (*Object* node)**

Emitted whenever a node is removed from the SceneTree.

- **physics_frame ( )**

Emitted immediately before *Node._physics_process* is called on every node in the SceneTree.

- **screen_resized ( )**

Emitted whenever the screen resolution (fullscreen) or window size (windowed) changes.

- **server_disconnected ( )**

Emitted whenever this SceneTree's *network_peer* disconnected from server. Only emitted on clients.

- **tree_changed ( )**

Emitted whenever the SceneTree hierarchy changed (children being moved or renamed, etc.).

29.361.4 Member Variables

- *Node* **current_scene** - The current scene.
- *bool* **debug_collisions_hint**
- *bool* **debug_navigation_hint**
- *Node* **edited_scene_root** - The root of the edited scene.
- *MultiplayerAPI* **multiplayer** - The default *MultiplayerAPI* instance for this SceneTree.
- *bool* **multiplayer_poll** - If `true` (default) enable the automatic polling of the *MultiplayerAPI* for this SceneTree during *idle_frame*.

When `false` you need to manually call *MultiplayerAPI.poll* for processing network packets and delivering RPCs/RSETs. This allows to run RPCs/RSETs in a different loop (e.g. physics, thread, specific time step) and for manual *Mutex* protecion when accessing the *MultiplayerAPI* from threads.

- *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** - If `true` the SceneTree is paused.
- *bool* **refuse_new_network_connections** - If `true` the SceneTree's *network_peer* refuses new incoming connections.
- *Viewport* **root** - The SceneTree's *Viewport*.
- *bool* **use_font_oversampling** - If `true` font oversampling is used.

29.361.5 Enums

enum **GroupCallFlags**

- **GROUP_CALL_DEFAULT = 0** — Call a group with no flags (default).
- **GROUP_CALL_REVERSE = 1** — Call a group in reverse scene order.
- **GROUP_CALL_REALTIME = 2** — Call a group immediately (calls are normally made on idle).
- **GROUP_CALL_UNIQUE = 4** — Call a group only once even if the call is executed many times.

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**

29.361.6 Description

As one of the most important classes, the `SceneTree` manages the hierarchy of nodes in a scene as well as scenes themselves. Nodes can be added, retrieved and removed. The whole scene tree (and thus the current scene) can be paused. Scenes can be loaded, switched and reloaded. You can also use the SceneTree to organize your nodes into groups: every node can be assigned as many groups as you want to create, e.g. a "enemy" group. You can then iterate these groups or even call methods and set properties on all the group's members at once.

29.361.7 Member Function Description

- *Variant* **call_group** (*String* group, *String* method) vararg

Calls `method` on each member of the given group.

- *Variant* **call_group_flags** (*int* flags, *String* group, *String* method) vararg

Calls `method` on each member of the given group, respecting the given GroupCallFlags.

- *Error* **change_scene** (*String* path)

Changes to the scene at the given `path`.

- *Error* **change_scene_to** (*PackedScene* packed_scene)

Changes to the given *PackedScene*.

- *SceneTreeTimer* **create_timer** (*float* time_sec, *bool* pause_mode_process=true)

Returns a *SceneTreeTimer* which will *SceneTreeTimer.timeout* after the given time in seconds elapsed in this SceneTree. If `pause_mode_process` is set to false, pausing the SceneTree will also pause the timer.

- *int* **get_frame** ( ) const
- *PoolIntArray* **get_network_connected_peers** ( ) const

Returns the peer IDs of all connected peers of this SceneTree's *network_peer*.

- *int* **get_network_unique_id** ( ) const

Returns the unique peer ID of this SceneTree's *network_peer*.

- *int* **get_node_count** ( ) const

Returns the number of nodes in this SceneTree.

- *Array* **get_nodes_in_group** (*String* group)

Returns all nodes assigned to the given group.

- *int* **get_rpc_sender_id** ( ) const

Returns the sender's peer ID for the most recently received RPC call.

- *bool* **has_group** (*String* name) const

Returns `true` if the given group exists.

- *bool* **has_network_peer** ( ) const

Returns `true` if there is a *network_peer* set.

- *bool* **is_input_handled** ( )

Returns `true` if the most recent InputEvent was marked as handled with *set_input_as_handled*.

- *bool* **is_network_server** ( ) const

Returns `true` if this SceneTree's *network_peer* is in server mode (listening for connections).

- void **notify_group** (*String* group, *int* notification)

Sends the given notification to all members of the `group`.

- void **notify_group_flags** (*int* call_flags, *String* group, *int* notification)

Sends the given notification to all members of the `group`, respecting the given GroupCallFlags.

- void **queue_delete** (*Object* obj)

Queues the given object for deletion, delaying the call to *Object.free* to after the current frame.

- void **quit** ( )

Quits the application.

- *Error* **reload_current_scene** ( )

Reloads the currently active scene.

- void **set_auto_accept_quit** (*bool* enabled)

If `true` the application automatically accepts quitting.

- void **set_group** (*String* group, *String* property, *Variant* value)

Sets the given `property` to `value` on all members of the given group.

- void **set_group_flags** (*int* call_flags, *String* group, *String* property, *Variant* value)

Sets the given `property` to `value` on all members of the given group, respecting the given GroupCallFlags.

- void **set_input_as_handled** ( )

Marks the most recent input event as handled.

- void **set_quit_on_go_back** (*bool* enabled)

If `true` the application quits automatically on going back (e.g. on Android).

- void **set_screen_stretch** (*StretchMode* mode, *StretchAspect* aspect, *Vector2* minsize, *float* shrink=1)

Configures screen stretching to the given StretchMode, StretchAspect, minimum size and `shrink`.

29.362 SceneTreeTimer

Inherits: *Reference* < *Object*

Category: Core

29.362.1 Brief Description

29.362.2 Signals

- **timeout** ( )

29.362.3 Member Variables

- *float* **time_left**

29.363 Script

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *VisualScript*, *GDScript*, *CSharpScript*, *NativeScript*, *PluginScript*

Category: Core

29.363.1 Brief Description

A class stored as a resource.

29.363.2 Member Functions

bool	*can_instance* ( ) const
Script	*get_base_script* ( ) const
String	*get_instance_base_type* ( ) const
bool	*has_script_signal* (*String* signal_name) const
bool	*has_source_code* ( ) const
bool	*instance_has* (*Object* base_object) const
bool	*is_tool* ( ) const
Error	*reload* (*bool* keep_state=false)

29.363.3 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.

29.363.4 Description

A class stored as a resource. The script exends 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.

29.363.5 Member Function Description

- *bool* **can_instance** ( ) const

Returns true if the script can be instanced.

- *Script* **get_base_script** ( ) const

Returns the script directly inherited by this script.

- *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.

- *Error* **reload** (*bool* keep_state=false)

Reloads the script's class implementation. Returns an error code.

29.364 ScriptEditor

Inherits: *PanelContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.364.1 Brief Description

29.364.2 Member Functions

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* ( )
Variant	*get_drag_data_fw* (*Vector2* point, *Control* from)
Array	*get_open_scripts* ( ) const
void	*open_script_create_dialog* (*String* base_name, *String* base_path)

29.364.3 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.

29.364.4 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)

29.365 ScrollBar

Inherits: *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *HScrollBar*, *VScrollBar*

Category: Core

29.365.1 Brief Description

Base class for scroll bars.

29.365.2 Signals

- **scrolling ( )**

Emitted whenever the scrollbar is being scrolled.

29.365.3 Member Variables

- *float* **custom_step**

29.365.4 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.

29.366 ScrollContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *EditorInspector*

Category: Core

29.366.1 Brief Description

A helper node for displaying scrollable elements (e.g. lists).

29.366.2 Signals

- **scroll_ended ( )**

Emitted whenever scrolling stops.

- **scroll_started ( )**

Emitted whenever scrolling is started.

29.366.3 Member Variables

- *int* **scroll_deadzone**
- *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.

29.366.4 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).

29.367 SegmentShape2D

Inherits: *Shape2D* < *Resource* < *Reference* < *Object*

Category: Core

29.367.1 Brief Description

Segment shape for 2D collisions.

29.367.2 Member Variables

- *Vector2* **a** - The segment's first point position.
- *Vector2* **b** - The segment's second point position.

29.367.3 Description

Segment shape for 2D collisions. Consists of two points, `a` and `b`.

29.368 Semaphore

Inherits: *Reference* < *Object*

Category: Core

29.368.1 Brief Description

A synchronization Semaphore.

29.368.2 Member Functions

Error	*post* ( )
Error	*wait* ( )

29.368.3 Description

A synchronization Semaphore. Element used to synchronize multiple *Thread*s. Initialized to zero on creation. Be careful to avoid deadlocks. For a binary version, see *Mutex*.

29.368.4 Member Function Description

- *Error* **post** ( )

Lowers the `Semaphore`, allowing one more thread in. Returns OK on success, ERR_BUSY otherwise.

- *Error* **wait** ( )

Tries to wait for the `Semaphore`, if its value is zero, blocks until non-zero. Returns OK on success, ERR_BUSY otherwise.

29.369 Separator

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *HSeparator*, *VSeparator*

Category: Core

29.369.1 Brief Description

Base class for separators.

29.369.2 Description

Separator is a *Control* used for separating other controls. It's purely a visual decoration. Horizontal (*HSeparator*) and Vertical (*VSeparator*) versions are available.

29.370 Shader

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.370.1 Brief Description

A custom shader program.

29.370.2 Member Functions

Texture	*get_default_texture_param* (*String* param) const
Mode	*get_mode* ( ) const
bool	*has_param* (*String* name) const
void	*set_default_texture_param* (*String* param, *Texture* texture)

29.370.3 Member Variables

- *String* **code**

29.370.4 Enums

enum **Mode**

- **MODE_SPATIAL = 0**
- **MODE_CANVAS_ITEM = 1**
- **MODE_PARTICLES = 2**

29.370.5 Description

This class allows you to define a custom shader program that can be used for various materials to render objects.

29.370.6 Member Function Description

- *Texture* **get_default_texture_param** (*String* param) const
- *Mode* **get_mode** ( ) const

Returns the shader mode for the shader, eiter `MODE_CANVAS_ITEM`, `MODE_SPATIAL` or `MODE_PARTICLES`

- *bool* **has_param** (*String* name) const
- void **set_default_texture_param** (*String* param, *Texture* texture)

29.371 ShaderMaterial

Inherits: *Material* < *Resource* < *Reference* < *Object*

Category: Core

29.371.1 Brief Description

A material that uses a custom *Shader* program.

29.371.2 Member Functions

Variant	*get_shader_param* (*String* param) const
void	*set_shader_param* (*String* param, *Variant* value)

29.371.3 Member Variables

- *Shader* **shader** - The *Shader* program used to render this material.

29.371.4 Description

A material that uses a custom *Shader* program to render either items to screen or process particles. You can create multiple materials for the same shader but configure different values for the uniforms defined in the shader.

29.371.5 Member Function Description

- *Variant* **get_shader_param** (*String* param) const

Returns the current value set for this material of a uniform in the shader.

- void **set_shader_param** (*String* param, *Variant* value)

Changes the value set for this material of a uniform in the shader.

29.372 Shape

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *PlaneShape*, *SphereShape*, *CapsuleShape*, *BoxShape*, *ConvexPolygonShape*, *RayShape*, *ConcavePolygonShape*

Category: Core

29.372.1 Brief Description

Base class for all 3D shape resources.

29.372.2 Description

Base class for all 3D shape resources. All 3D shapes that inherit from this can be set into a *PhysicsBody* or *Area*.

29.373 Shape2D

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *RayShape2D*, *CapsuleShape2D*, *LineShape2D*, *CircleShape2D*, *ConcavePolygonShape2D*, *ConvexPolygonShape2D*, *RectangleShape2D*, *SegmentShape2D*

Category: Core

29.373.1 Brief Description

Base class for all 2D Shapes.

29.373.2 Member Functions

bool	*collide* (*Transform2D* local_xform, *Shape2D* with_shape, *Transform2D* shape_xform)
Variant	*collide_and_get_contacts* (*Transform2D* local_xform, *Shape2D* with_shape, *Transform2D* shape_xform)
bool	*collide_with_motion* (*Transform2D* local_xform, *Vector2* local_motion, *Shape2D* with_shape, *Transform2D* shape_xform, *Vector2* shape_motion)
Variant	*collide_with_motion_and_get_contacts* (*Transform2D* local_xform, *Vector2* local_motion, *Shape2D* with_shape, *Transform2D* shape_xform, *Vector2* shape_motion)

29.373.3 Member Variables

- *float* **custom_solver_bias**

29.373.4 Description

Base class for all 2D Shapes. All 2D shape types inherit from this.

29.373.5 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`).

29.374 ShortCut

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.374.1 Brief Description

A shortcut for binding input.

29.374.2 Member Functions

String	*get_as_text* ( ) const
bool	*is_shortcut* (*InputEvent* event) const
bool	*is_valid* ( ) const

29.374.3 Member Variables

- *InputEvent* **shortcut** - The Shortcut's *InputEvent*.

Generally the *InputEvent* is a keyboard key, though it can be any *InputEvent*.

29.374.4 Description

A shortcut for binding input.

Shortcuts are commonly used for interacting with a *Control* element from a *InputEvent*.

29.374.5 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.

29.375 Skeleton

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.375.1 Brief Description

Skeleton for characters and animated objects.

29.375.2 Member Functions

void	*add_bone* (*String* name)
void	*bind_child_node_to_bone* (*int* bone_idx, *Node* node)
void	*clear_bones* ( )
int	*find_bone* (*String* name) const
int	*get_bone_count* ( ) const
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
int	*get_bone_parent* (*int* bone_idx) const
Transform	*get_bone_pose* (*int* bone_idx) const
Transform	*get_bone_rest* (*int* bone_idx) const
Transform	*get_bone_transform* (*int* bone_idx) const
Array	*get_bound_child_nodes_to_bone* (*int* bone_idx) const
bool	*is_bone_rest_disabled* (*int* bone_idx) const
void	*physical_bones_add_collision_exception* (*RID* exception)
void	*physical_bones_remove_collision_exception* (*RID* exception)
void	*physical_bones_start_simulation* (*Array* bones=[ ])
void	*physical_bones_stop_simulation* ( )
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)
void	*set_bone_pose* (*int* bone_idx, *Transform* pose)
void	*set_bone_rest* (*int* bone_idx, *Transform* rest)
void	*unbind_child_node_from_bone* (*int* bone_idx, *Node* node)
void	*unparent_bone_and_rest* (*int* bone_idx)

29.375.3 Numeric Constants

- **NOTIFICATION_UPDATE_SKELETON = 50**

29.375.4 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.

29.375.5 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 **physical_bones_add_collision_exception** (*RID* exception)
- void **physical_bones_remove_collision_exception** (*RID* exception)
- void **physical_bones_start_simulation** (*Array* bones=[ ])
- void **physical_bones_stop_simulation** ( )
- 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)

29.376 Skeleton2D

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.376.1 Brief Description

29.376.2 Member Functions

Bone2D	*get_bone* (*int* arg0)
int	*get_bone_count* ( ) const
RID	*get_skeleton* ( ) const

29.376.3 Member Function Description

- *Bone2D* **get_bone** (*int* arg0)
- *int* **get_bone_count** ( ) const
- *RID* **get_skeleton** ( ) const

29.377 Sky

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *PanoramaSky*, *ProceduralSky*

Category: Core

29.377.1 Brief Description

The base class for *PanoramaSky* and *ProceduralSky*.

29.377.2 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.

29.377.3 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.

29.377.4 Description

The base class for *PanoramaSky* and *ProceduralSky*.

29.378 Slider

Inherits: *Range* **<** *Control* **<** *CanvasItem* **<** *Node* **<** *Object*

Inherited By: *HSlider*, *VSlider*

Category: Core

29.378.1 Brief Description

Base class for GUI Sliders.

29.378.2 Member Variables

- *bool* **editable**
- *FocusMode* **focus_mode**
- *int* **tick_count**

- *bool* **ticks_on_borders**

29.378.3 Description

Base class for GUI Sliders.

29.379 SliderJoint

Inherits: *Joint* < *Spatial* < *Node* < *Object*

Category: Core

29.379.1 Brief Description

Piston kind of slider between two bodies in 3D.

29.379.2 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.

29.379.3 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.

29.379.4 Description

Slides across the x-axis of the Pivot object.

29.380 Spatial

Inherits: *Node* < *Object*

Inherited By: *Joint*, *RayCast*, *Camera*, *BoneAttachment*, *CollisionShape*, *AudioStreamPlayer3D*, *ARVRController*, *Path*, *VisualInstance*, *VehicleWheel*, *Position3D*, *ProximityGroup*, *ARVRAnchor*, *RemoteTransform*, *CollisionObject*, *OrientedPathFollow*, *PathFollow*, *NavigationMeshInstance*, *Listener*, *VisibilityNotifier*, *Navigation*, *CollisionPolygon*, *GridMap*, *Skeleton*, *ARVROrigin*

Category: Core

29.380.1 Brief Description

Most basic 3D game object, parent of all 3D related nodes.

29.380.2 Member Functions

Spatial	*get_parent_spatial* ( ) const
World	*get_world* ( ) const
void	*global_rotate* (*Vector3* axis, *float* angle)
void	*global_scale* (*Vector3* scale)
void	*global_translate* (*Vector3* offset)
void	*hide* ( )
bool	*is_local_transform_notification_enabled* ( ) const
bool	*is_set_as_toplevel* ( ) const
bool	*is_transform_notification_enabled* ( ) const
bool	*is_visible_in_tree* ( ) const
void	*look_at* (*Vector3* target, *Vector3* up)
void	*look_at_from_position* (*Vector3* position, *Vector3* target, *Vector3* up)
void	*orthonormalize* ( )
void	*rotate* (*Vector3* axis, *float* angle)
void	*rotate_object_local* (*Vector3* axis, *float* angle)
void	*rotate_x* (*float* angle)
void	*rotate_y* (*float* angle)
void	*rotate_z* (*float* angle)
void	*scale_object_local* (*Vector3* scale)
void	*set_as_toplevel* (*bool* enable)
void	*set_identity* ( )
void	*set_ignore_transform_notification* (*bool* enabled)
void	*set_notify_local_transform* (*bool* enable)
void	*set_notify_transform* (*bool* enable)
void	*show* ( )
Vector3	*to_global* (*Vector3* local_point) const
Vector3	*to_local* (*Vector3* global_point) const
void	*translate* (*Vector3* offset)
void	*translate_object_local* (*Vector3* offset)
void	*update_gizmo* ( )

29.380.3 Signals

- **visibility_changed** ( )

Emitted when node visibility changes.

29.380.4 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 indepdent 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.

29.380.5 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.

29.380.6 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.

29.380.7 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 so that the local -Z axis points towards the `target` position.

The transform will first be rotated around the given `up` vector, and then fully aligned to the target by a further rotation around an axis perpendicular to both the `target` and `up` vectors.

Operations take place in global space.

- void **look_at_from_position (*Vector3* position, *Vector3* target, *Vector3* up)**

Moves the node to the specified `position`, and then rotates itself to point toward the `target` as per *look_at*. 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.

29.381 SpatialGizmo

Inherits: *Reference* < *Object*

Inherited By: *EditorSpatialGizmo*

Category: Core

29.381.1 Brief Description

29.382 SpatialMaterial

Inherits: *Material* < *Resource* < *Reference* < *Object*

Category: Core

29.382.1 Brief Description

29.382.2 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_do_not_receive_shadows**
- *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**

29.382.3 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_SUBSURACE_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_DONT_RECEIVE_SHADOWS = 14**
- **FLAG_MAX = 15**

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**

29.383 SpatialVelocityTracker

Inherits: *Reference* < *Object*

Category: Core

29.383.1 Brief Description

29.383.2 Member Functions

Vector3	*get_tracked_linear_velocity* ( ) const
void	*reset* (*Vector3* position)
void	*update_position* (*Vector3* position)

29.383.3 Member Variables

- *bool* **track_physics_step**

29.383.4 Member Function Description

- *Vector3* **get_tracked_linear_velocity** ( ) const
- void **reset** (*Vector3* position)
- void **update_position** (*Vector3* position)

29.384 SphereMesh

Inherits: *PrimitiveMesh* < *Mesh* < *Resource* < *Reference* < *Object*

Category: Core

29.384.1 Brief Description

Class representing a spherical *PrimitiveMesh*.

29.384.2 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.

29.384.3 Description

Class representing a spherical *PrimitiveMesh*.

29.385 SphereShape

Inherits: *Shape* < *Resource* < *Reference* < *Object*

Category: Core

29.385.1 Brief Description

Sphere shape for 3D collisions.

29.385.2 Member Variables

- *float* **radius** - The sphere's radius. The shape's diameter is double the radius.

29.385.3 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.

29.386 SpinBox

Inherits: *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.386.1 Brief Description

Numerical input text field.

29.386.2 Member Functions

LineEdit	*get_line_edit* **( )**

29.386.3 Member Variables

- *bool* **editable**
- *String* **prefix**
- *String* **suffix**

29.386.4 Description

SpinBox is a numerical input text field. It allows entering integers and floats.

29.386.5 Member Function Description

- *LineEdit* **get_line_edit ( )**

29.387 SplitContainer

Inherits: *Container* **<** *Control* **<** *CanvasItem* **<** *Node* **<** *Object*

Inherited By: *HSplitContainer*, *VSplitContainer*

Category: Core

29.387.1 Brief Description

Container for splitting and adjusting.

29.387.2 Signals

- **dragged (** *int* offset **)**

Emitted when the dragger is dragged by user.

29.387.3 Member Variables

- *bool* **collapsed**
- *DraggerVisibility* **dragger_visibility** - Determines whether the dragger is visible.
- *int* **split_offset**

29.387.4 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.

29.387.5 Description

Container for splitting two controls vertically or horizontally, with a grabber that allows adjusting the split offset or ratio.

29.388 SpotLight

Inherits: *Light* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.388.1 Brief Description

Spotlight *Light*, such as a reflector spotlight or a lantern.

29.388.2 Member Variables

- *float* **spot_angle**
- *float* **spot_angle_attenuation**
- *float* **spot_attenuation**
- *float* **spot_range**

29.388.3 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.

29.389 Sprite

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.389.1 Brief Description

General purpose Sprite node.

29.389.2 Member Functions

Rect2	*get_rect* ( ) const

29.389.3 Signals

- **frame_changed** ( )

Emitted when the *frame* changes.

- **texture_changed** ( )

Emitted when the *texture* changes.

29.389.4 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.

29.389.5 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.

29.389.6 Member Function Description

- *Rect2* **get_rect ( )** const

Returns a Rect2 representing the Sprite's boundary relative to its local coordinates.

29.390 Sprite3D

Inherits: *SpriteBase3D* < *GeometryInstance* < *VisualInstance* < *Spatial* < *Node* < *Object*

Category: Core

29.390.1 Brief Description

2D Sprite node in 3D world.

29.390.2 Signals

- **frame_changed ( )**

Emitted when the *frame* changes.

29.390.3 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.

29.390.4 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.

29.391 SpriteBase3D

Inherits: *GeometryInstance* < *VisualInstance* < *Spatial* < *Node* < *Object*

Inherited By: *AnimatedSprite3D*, *Sprite3D*

Category: Core

29.391.1 Brief Description

2D Sprite node in 3D environment.

29.391.2 Member Functions

Rect2	*get_item_rect* ( ) const

29.391.3 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 objects 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`.

29.391.4 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.

29.391.5 Description

A node that displays 2D texture information in a 3D environment.

29.391.6 Member Function Description

- *Rect2* **get_item_rect** ( ) const

29.392 SpriteFrames

Inherits: *Resource* **<** *Reference* **<** *Object*

Category: Core

29.392.1 Brief Description

Sprite frame library for AnimatedSprite.

29.392.2 Member Functions

void	*add_animation* (*String* anim)
void	*add_frame* (*String* anim, *Texture* frame, *int* at_position=-1)
void	*clear* (*String* anim)
void	*clear_all* ( )
bool	*get_animation_loop* (*String* anim) const
float	*get_animation_speed* (*String* anim) const
Texture	*get_frame* (*String* anim, *int* idx) const
int	*get_frame_count* (*String* anim) const
bool	*has_animation* (*String* anim) const
void	*remove_animation* (*String* anim)
void	*remove_frame* (*String* anim, *int* idx)
void	*rename_animation* (*String* anim, *String* newname)
void	*set_animation_loop* (*String* anim, *bool* loop)
void	*set_animation_speed* (*String* anim, *float* speed)
void	*set_frame* (*String* anim, *int* idx, *Texture* txt)

29.392.3 Member Variables

- *Array* **frames**

29.392.4 Description

Sprite frame library for *AnimatedSprite*. Contains frames and animation data for playback.

29.392.5 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.

29.393 StaticBody

Inherits: *PhysicsBody* < *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

29.393.1 Brief Description

Static body for 3D Physics.

29.393.2 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).

29.393.3 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).

29.394 StaticBody2D

Inherits: *PhysicsBody2D* < *CollisionObject2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.394.1 Brief Description

Static body for 2D Physics.

29.394.2 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).

29.394.3 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).

29.395 StreamPeer

Inherits: *Reference* < *Object*

Inherited By: *StreamPeerBuffer*, *StreamPeerSSL*, *StreamPeerTCP*

Category: Core

29.395.1 Brief Description

Abstraction and base class for stream-based protocols.

29.395.2 Member Functions

int	*get_16* ( )
int	*get_32* ( )
int	*get_64* ( )
int	*get_8* ( )
int	*get_available_bytes* ( ) const
Array	*get_data* (*int* bytes)
float	*get_double* ( )
float	*get_float* ( )
Array	*get_partial_data* (*int* bytes)
String	*get_string* (*int* bytes)
int	*get_u16* ( )
int	*get_u32* ( )
int	*get_u64* ( )
int	*get_u8* ( )
String	*get_utf8_string* (*int* bytes)
Variant	*get_var* ( )
void	*put_16* (*int* value)
void	*put_32* (*int* value)
void	*put_64* (*int* value)
void	*put_8* (*int* value)
Error	*put_data* (*PoolByteArray* data)
void	*put_double* (*float* value)
void	*put_float* (*float* value)
Array	*put_partial_data* (*PoolByteArray* data)
void	*put_u16* (*int* value)
void	*put_u32* (*int* value)
void	*put_u64* (*int* value)
void	*put_u8* (*int* value)
void	*put_utf8_string* (*String* value)
void	*put_var* (*Variant* value)

29.395.3 Member Variables

- *bool* **big_endian** - If `true`, this `StreamPeer` will using big-endian format for encoding and decoding.

29.395.4 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.

29.395.5 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.

- *Error* **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.

29.396 StreamPeerBuffer

Inherits: *StreamPeer* **<** *Reference* **<** *Object*

Category: Core

29.396.1 Brief Description

29.396.2 Member Functions

void	*clear* ( )
StreamPeerBuffer	*duplicate* ( ) const
int	*get_position* ( ) const
int	*get_size* ( ) const
void	*resize* (*int* size)
void	*seek* (*int* position)

29.396.3 Member Variables

- *PoolByteArray* **data_array**

29.396.4 Member Function Description

- void **clear** ( )
- *StreamPeerBuffer* **duplicate** ( ) const
- *int* **get_position** ( ) const
- *int* **get_size** ( ) const
- void **resize** (*int* size)
- void **seek** (*int* position)

29.397 StreamPeerSSL

Inherits: *StreamPeer* < *Reference* < *Object*

Category: Core

29.397.1 Brief Description

SSL Stream peer.

29.397.2 Member Functions

Error	*accept_stream* (*StreamPeer* stream)
Error	*connect_to_stream* (*StreamPeer* stream, *bool* validate_certs=false, *String* for_hostname="")
void	*disconnect_from_stream* ( )
Status	*get_status* ( ) const
void	*poll* ( )

29.397.3 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 errot 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.

29.397.4 Description

SSL Stream peer. This object can be used to connect to SSL servers.

29.397.5 Member Function Description

- *Error* **accept_stream** (*StreamPeer* stream)
- *Error* **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.

- *Status* **get_status** ( ) const

Return the status of the connection, one of STATUS_* enum.

- void **poll** ( )

Poll the connection to check for incoming bytes. Call this right before "get_available_bytes()" for it to work properly.

29.398 StreamPeerTCP

Inherits: *StreamPeer* **<** *Reference* **<** *Object*

Category: Core

29.398.1 Brief Description

TCP Stream peer.

29.398.2 Member Functions

Error	*connect_to_host* (*String* host, *int* port)
void	*disconnect_from_host* ( )
String	*get_connected_host* ( ) const
int	*get_connected_port* ( ) const
Status	*get_status* ( ) const
bool	*is_connected_to_host* ( ) const
void	*set_no_delay* (*bool* enabled)

29.398.3 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 staus representing a `StreamPeerTCP` in error state.

29.398.4 Description

TCP Stream peer. This object can be used to connect to TCP servers, or also is returned by a tcp server.

29.398.5 Member Function Description

- *Error* **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.

- *Status* **get_status** ( ) const

Return the status of the connection, one of STATUS_* enum.

- *bool* **is_connected_to_host** ( ) const
- void **set_no_delay** (*bool* enabled)

Disable Nagle algorithm to improve latency for small packets.

Note that for applications that send large packets, or need to transfer a lot of data, this can reduce total bandwidth.

29.399 StreamTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.399.1 Brief Description

A .stex texture.

29.399.2 Member Variables

- *String* **load_path** - The StreamTexture's filepath to a .stex file.

29.399.3 Description

A texture that is loaded from a .stex file.

29.400 String

Category: Built-In Types

29.400.1 Brief Description

Built-in string class.

29.400.2 Member Functions

String	String (*bool* from)
String	String (*int* from)
String	String (*float* from)
String	String (*Vector2* from)
String	String (*Rect2* from)
String	String (*Vector3* from)
String	String (*Transform2D* from)
String	String (*Plane* from)
String	String (*Quat* from)
String	String (*AABB* from)
String	String (*Basis* from)
String	String (*Transform* from)
String	String (*Color* from)
String	String (*NodePath* from)
String	String (*RID* from)
String	String (*Dictionary* from)

Continued on next page

Table 20 – continued from previous page

String	String (*Array* from)
String	String (*PoolByteArray* from)
String	String (*PoolIntArray* from)
String	String (*PoolRealArray* from)
String	String (*PoolStringArray* from)
String	String (*PoolVector2Array* from)
String	String (*PoolVector3Array* from)
String	String (*PoolColorArray* from)
bool	*begins_with* (*String* text)
PoolStringArray	*bigrams* ( )
String	*c_escape* ( )
String	*c_unescape* ( )
String	*capitalize* ( )
int	*casecmp_to* (*String* to)
String	*dedent* ( )
bool	*empty* ( )
bool	*ends_with* (*String* text)
void	*erase* (*int* position, *int* chars)
int	*find* (*String* what, *int* from=0)
int	*find_last* (*String* what)
int	*findn* (*String* what, *int* from=0)
String	*format* (var values, *String* placeholder={_})
String	*get_base_dir* ( )
String	*get_basename* ( )
String	*get_extension* ( )
String	*get_file* ( )
int	*hash* ( )
int	*hex_to_int* ( )
String	*insert* (*int* position, *String* what)
bool	*is_abs_path* ( )
bool	*is_rel_path* ( )
bool	*is_subsequence_of* (*String* text)
bool	*is_subsequence_ofi* (*String* text)
bool	*is_valid_float* ( )
bool	*is_valid_html_color* ( )
bool	*is_valid_identifier* ( )
bool	*is_valid_integer* ( )
bool	*is_valid_ip_address* ( )
String	*json_escape* ( )
String	*left* (*int* position)
int	*length* ( )
String	*lstrip* (*String* chars)
bool	*match* (*String* expr)
bool	*matchn* (*String* expr)
PoolByteArray	*md5_buffer* ( )
String	*md5_text* ( )
int	*nocasecmp_to* (*String* to)
int	*ord_at* (*int* at)
String	*pad_decimals* (*int* digits)
String	*pad_zeros* (*int* digits)

Continued on next page

Table 20 – continued from previous page

String	*percent_decode* ( )
String	*percent_encode* ( )
String	*plus_file* (*String* file)
String	*replace* (*String* what, *String* forwhat)
String	*replacen* (*String* what, *String* forwhat)
int	*rfind* (*String* what, *int* from=-1)
int	*rfindn* (*String* what, *int* from=-1)
String	*right* (*int* position)
PoolStringArray	*rsplit* (*String* divisor, *bool* allow_empty=True, *int* maxsplit=0)
String	*rstrip* (*String* chars)
PoolByteArray	*sha256_buffer* ( )
String	*sha256_text* ( )
float	*similarity* (*String* text)
PoolStringArray	*split* (*String* divisor, *bool* allow_empty=True, *int* maxsplit=0)
PoolRealArray	*split_floats* (*String* divisor, *bool* allow_empty=True)
String	*strip_edges* (*bool* left=True, *bool* right=True)
String	*substr* (*int* from, *int* len)
PoolByteArray	*to_ascii* ( )
float	*to_float* ( )
int	*to_int* ( )
String	*to_lower* ( )
String	*to_upper* ( )
PoolByteArray	*to_utf8* ( )
String	*trim_prefix* (*String* prefix)
String	*trim_suffix* (*String* suffix)
String	*xml_escape* ( )
String	*xml_unescape* ( )

29.400.3 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.

29.400.4 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 lowercase, 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.

- *String* **lstrip** (*String* chars)

Returns a copy of the string with characters removed from the left.

- *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.

- *PoolStringArray* **rsplit** (*String* divisor, *bool* allow_empty=True, *int* maxsplit=0)

Splits the string by a `divisor` string and returns an array of the substrings, starting from right. Example "One,Two,Three" will return "One","Two","Three" if split by ",".

If `maxsplit` is specified, then it is number of splits to do, default is 0 which splits all the items.

- *String* **rstrip** (*String* chars)

Returns a copy of the string with characters removed from the right.

- *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* **trim_prefix** (*String* prefix)

Removes a given string from the start if it starts with it or leaves the string unchanged.

- *String* **trim_suffix** (*String* suffix)

Removes a given string from the end if it ends with it or leaves the string unchanged.

- *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.

29.401 StyleBox

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *StyleBoxLine*, *StyleBoxFlat*, *StyleBoxTexture*, *StyleBoxEmpty*

Category: Core

29.401.1 Brief Description

Base class for drawing stylized boxes for the UI.

29.401.2 Member Functions

void	*draw* (*RID* canvas_item, *Rect2* rect) const
Vector2	*get_center_size* ( ) const
float	*get_margin* (*Margin* margin) const
Vector2	*get_minimum_size* ( ) const
Vector2	*get_offset* ( ) const
bool	*test_mask* (*Vector2* point, *Rect2* rect) const

29.401.3 Member Variables

- *float* **content_margin_bottom**
- *float* **content_margin_left**
- *float* **content_margin_right**
- *float* **content_margin_top**

29.401.4 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.

29.401.5 Member Function Description

- void **draw** (*RID* canvas_item, *Rect2* rect) const
- *Vector2* **get_center_size** ( ) const
- *float* **get_margin** (*Margin* 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.

29.402 StyleBoxEmpty

Inherits: *StyleBox* < *Resource* < *Reference* < *Object*

Category: Core

29.402.1 Brief Description

Empty stylebox (does not display anything).

29.402.2 Description

Empty stylebox (really does not display anything).

29.403 StyleBoxFlat

Inherits: *StyleBox* < *Resource* < *Reference* < *Object*

Category: Core

29.403.1 Brief Description

Customizable Stylebox with a given set of parameters. (no texture required)

29.403.2 Member Functions

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)

29.403.3 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** - Toggels 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.

29.403.4 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
```

29.403.5 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)

29.404 StyleBoxLine

Inherits: *StyleBox* < *Resource* < *Reference* < *Object*

Category: Core

29.404.1 Brief Description

29.404.2 Member Variables

- *Color* **color**
- *float* **grow**
- *int* **thickness**
- *bool* **vertical**

29.405 StyleBoxTexture

Inherits: *StyleBox* < *Resource* < *Reference* < *Object*

Category: Core

29.405.1 Brief Description

Texture Based 3x3 scale style.

29.405.2 Member Functions

void	*set_expand_margin_all* (*float* size)
void	*set_expand_margin_individual* (*float* size_left, *float* size_top, *float* size_right, *float* size_bottom)

29.405.3 Signals

- **texture_changed** ( )

29.405.4 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**
- *Texture* **normal_map**
- *Rect2* **region_rect**
- *Texture* **texture**

29.405.5 Enums

enum **AxisStretchMode**

- **AXIS_STRETCH_MODE_STRETCH = 0**
- **AXIS_STRETCH_MODE_TILE = 1**
- **AXIS_STRETCH_MODE_TILE_FIT = 2**

29.405.6 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.

29.405.7 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)

29.406 SurfaceTool

Inherits: *Reference* < *Object*

Category: Core

29.406.1 Brief Description

Helper tool to create geometry.

29.406.2 Member Functions

void	*add_bones* (*PoolIntArray* bones)
void	*add_color* (*Color* color)
void	*add_index* (*int* index)
void	*add_normal* (*Vector3* normal)
void	*add_smooth_group* (*bool* smooth)
void	*add_tangent* (*Plane* tangent)
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=[ ])
void	*add_uv* (*Vector2* uv)
void	*add_uv2* (*Vector2* uv2)
void	*add_vertex* (*Vector3* vertex)
void	*add_weights* (*PoolRealArray* weights)
void	*append_from* (*Mesh* existing, *int* surface, *Transform* transform)
void	*begin* (*PrimitiveType* primitive)
void	*clear* ( )
ArrayMesh	*commit* (*ArrayMesh* existing=null, *int* flags=97792)
void	*create_from* (*Mesh* existing, *int* surface)
void	*deindex* ( )
void	*generate_normals* (*bool* flip=false)
void	*generate_tangents* ( )
void	*index* ( )
void	*set_material* (*Material* material)

29.406.3 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.

29.406.4 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** (*PrimitiveType* 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** (*bool* flip=false)

Generates normals from Vertices so you do not have to do it manually.

Setting "flip" `true` inverts resulting normals.

- 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.

29.407 TabContainer

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.407.1 Brief Description

Tabbed Container.

29.407.2 Member Functions

Control	*get_current_tab_control* ( ) const
Popup	*get_popup* ( ) const
int	*get_previous_tab* ( ) const
Control	*get_tab_control* (*int* idx) const
int	*get_tab_count* ( ) const
bool	*get_tab_disabled* (*int* tab_idx) const
Texture	*get_tab_icon* (*int* tab_idx) const
String	*get_tab_title* (*int* tab_idx) const
int	*get_tabs_rearrange_group* ( ) const
void	*set_popup* (*Node* popup)
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)
void	*set_tabs_rearrange_group* (*int* group_id)

29.407.3 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.

29.407.4 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`.
- *bool* **drag_to_rearrange_enabled**
- *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`.

29.407.5 Enums

enum **TabAlign**

- **ALIGN_LEFT = 0**
- **ALIGN_CENTER = 1**
- **ALIGN_RIGHT = 2**

29.407.6 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.

29.407.7 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*.

- *int* **get_tabs_rearrange_group** ( ) const
- 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*.

- void **set_tabs_rearrange_group** (*int* group_id)

29.408 Tabs

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.408.1 Brief Description

Tabs Control.

29.408.2 Member Functions

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
String	*get_tab_title* (*int* tab_idx) const
int	*get_tabs_rearrange_group* ( ) const
void	*move_tab* (*int* from, *int* to)
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)
void	*set_tabs_rearrange_group* (*int* group_id)

29.408.3 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)

29.408.4 Member Variables

- *int* **current_tab**
- *bool* **drag_to_rearrange_enabled**
- *bool* **scrolling_enabled**
- *TabAlign* **tab_align**
- *CloseButtonDisplayPolicy* **tab_close_display_policy**

29.408.5 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**

29.408.6 Description

Simple tabs control, similar to *TabContainer* but is only in charge of drawing tabs, not interact with children.

29.408.7 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
- *int* **get_tabs_rearrange_group** ( ) 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)
- void **set_tabs_rearrange_group** (*int* group_id)

29.409 TCP_Server

Inherits: *Reference* < *Object*

Category: Core

29.409.1 Brief Description

TCP Server.

29.409.2 Member Functions

bool	*is_connection_available* ( ) const
Error	*listen* (*int* port, *String* bind_address="*")
void	*stop* ( )
StreamPeerTCP	*take_connection* ( )

29.409.3 Description

TCP Server class. Listens to connections on a port and returns a *StreamPeerTCP* when got a connection.

29.409.4 Member Function Description

- *bool* **is_connection_available** ( ) const

Return true if a connection is available for taking.

- *Error* **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/

29.410 TextEdit

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.410.1 Brief Description

Multiline text editing control.

29.410.2 Member Functions

void	*add_color_region* (*String* begin_key, *String* end_key, *Color* color, *bool* line_only=false)
void	*add_keyword_color* (*String* keyword, *Color* color)
bool	*can_fold* (*int* line) const
void	*clear_colors* ( )
void	*clear_undo_history* ( )
void	*copy* ( )
int	*cursor_get_column* ( ) const
int	*cursor_get_line* ( ) const
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, *int* wrap_index=0)
void	*cut* ( )
void	*deselect* ( )
void	*fold_all_lines* ( )
void	*fold_line* (*int* line)
Color	*get_keyword_color* (*String* keyword) const
String	*get_line* (*int* line) const
int	*get_line_count* ( ) const
PopupMenu	*get_menu* ( ) const
int	*get_selection_from_column* ( ) const
int	*get_selection_from_line* ( ) const
String	*get_selection_text* ( ) const
int	*get_selection_to_column* ( ) const
int	*get_selection_to_line* ( ) const
String	*get_word_under_cursor* ( ) const
bool	*has_keyword_color* (*String* keyword) const
void	*insert_text_at_cursor* (*String* text)
bool	*is_folded* (*int* line) const
bool	*is_line_hidden* (*int* line) const
bool	*is_selection_active* ( ) const
void	*menu_option* (*int* option)
void	*paste* ( )
void	*redo* ( )
PoolIntArray	*search* (*String* key, *int* flags, *int* from_line, *int* from_column) const
void	*select* (*int* from_line, *int* from_column, *int* to_line, *int* to_column)
void	*select_all* ( )
void	*set_line_as_hidden* (*int* line, *bool* enable)
void	*toggle_fold_line* (*int* line)
void	*undo* ( )
void	*unfold_line* (*int* line)
void	*unhide_all_lines* ( )

29.410.3 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.

29.410.4 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 he edge of what is visible.
- *bool* **wrap_enabled**

29.410.5 Enums

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 *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.

29.410.6 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.

29.410.7 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, *int* wrap_index=0)
- void **cut** ( )

Cut the current selection.

- void **deselect** ( )

Clears the current selection.

- void **fold_all_lines** ( )

- void **fold_line** (*int* line)
- *Color* **get_keyword_color** (*String* keyword) const
- *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
- *bool* **has_keyword_color** (*String* keyword) 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** ( )

29.411 Texture

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *CurveTexture*, *AtlasTexture*, *ProxyTexture*, *GradientTexture*, *ViewportTexture*, *StreamTexture*, *ImageTexture*, *LargeTexture*

Category: Core

29.411.1 Brief Description

Texture for 2D and 3D.

29.411.2 Member Functions

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
Vector2	*get_size* ( ) const
int	*get_width* ( ) const
bool	*has_alpha* ( ) const

29.411.3 Member Variables

- *int* **flags** - The texture's flags.

29.411.4 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 horrizon 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.

29.411.5 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*.

29.411.6 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

29.412 TextureButton

Inherits: *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.412.1 Brief Description

Texture-based button. Supports Pressed, Hover, Disabled and Focused states.

29.412.2 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.

29.412.3 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.

29.412.4 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.

29.413 TextureProgress

Inherits: *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.413.1 Brief Description

Texture-based progress bar. Useful for loading screens and life or stamina bars.

29.413.2 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.
- *Color* **tint_over**
- *Color* **tint_progress**
- *Color* **tint_under**

29.413.3 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 bototm.
- **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.

29.413.4 Description

TextureProgress works like *ProgressBar* but it uses up to 3 textures instead of Godot's *Theme* resource. Works horizontally, vertically, and radially.

29.414 TextureRect

Inherits: *Control* **<** *CanvasItem* **<** *Node* **<** *Object*

Category: Core

29.414.1 Brief Description

Control for drawing textures.

29.414.2 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 resizing the node's bounding rectangle. See StretchMode.
- *Texture* **texture** - The node's *Texture* resource.

29.414.3 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.

29.414.4 Description

Used to draw icons and sprites in a user interface. The texture's placement can be controlled with the *stretch_mode* property. It can scale, tile, or stay centered inside its bounding rectangle.

29.415 Theme

Inherits: *Resource* **<** *Reference* **<** *Object*

Category: Core

29.415.1 Brief Description

Theme for controls.

29.415.2 Member Functions

void	*clear_color* (*String* name, *String* type)
void	*clear_constant* (*String* name, *String* type)
void	*clear_font* (*String* name, *String* type)
void	*clear_icon* (*String* name, *String* type)
void	*clear_stylebox* (*String* name, *String* type)
void	*copy_default_theme* ( )
Color	*get_color* (*String* name, *String* type) const
PoolStringArray	*get_color_list* (*String* type) const
int	*get_constant* (*String* name, *String* type) const
PoolStringArray	*get_constant_list* (*String* type) const
Font	*get_font* (*String* name, *String* type) const
PoolStringArray	*get_font_list* (*String* type) const
Texture	*get_icon* (*String* name, *String* type) const
PoolStringArray	*get_icon_list* (*String* type) const
StyleBox	*get_stylebox* (*String* name, *String* type) const
PoolStringArray	*get_stylebox_list* (*String* type) const
PoolStringArray	*get_stylebox_types* ( ) const
PoolStringArray	*get_type_list* (*String* type) const
bool	*has_color* (*String* name, *String* type) const
bool	*has_constant* (*String* name, *String* type) const
bool	*has_font* (*String* name, *String* type) const
bool	*has_icon* (*String* name, *String* type) const
bool	*has_stylebox* (*String* name, *String* type) const
void	*set_color* (*String* name, *String* type, *Color* color)
void	*set_constant* (*String* name, *String* type, *int* constant)
void	*set_font* (*String* name, *String* type, *Font* font)
void	*set_icon* (*String* name, *String* type, *Texture* texture)
void	*set_stylebox* (*String* name, *String* type, *StyleBox* texture)

29.415.3 Member Variables

- *Font* **default_font** - The theme's default font.

29.415.4 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.

29.415.5 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 *Color*s 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 *Font*s 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`.

29.416 Thread

Inherits: *Reference* **<** *Object*

Category: Core

29.416.1 Brief Description

A unit of execution in a process.

29.416.2 Member Functions

String	*get_id* ( ) const
bool	*is_active* ( ) const
Error	*start* (*Object* instance, *String* method, *Variant* userdata=null, *int* priority=1)
Variant	*wait_to_finish* ( )

29.416.3 Enums

enum **Priority**

- **PRIORITY_LOW = 0**
- **PRIORITY_NORMAL = 1**
- **PRIORITY_HIGH = 2**

29.416.4 Description

A unit of execution in a process. Can run methods on *Object*s simultaneously. The use of synchronization via *Mutex*, *Semaphore* is advised if working with shared objects.

29.416.5 Member Function Description

- *String* **get_id** ( ) const

Returns the current `Thread`s 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*.

- *Error* **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.

29.417 TileMap

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.417.1 Brief Description

Node for 2D tile-based maps.

29.417.2 Member Functions

void	*clear* ( )
void	*fix_invalid_tiles* ( )
int	*get_cell* (*int* x, *int* y) const
int	*get_cellv* (*Vector2* position) const
bool	*get_collision_layer_bit* (*int* bit) const
bool	*get_collision_mask_bit* (*int* bit) const
Array	*get_used_cells* ( ) const
Array	*get_used_cells_by_id* (*int* id) const
Rect2	*get_used_rect* ( )
bool	*is_cell_transposed* (*int* x, *int* y) const
bool	*is_cell_x_flipped* (*int* x, *int* y) const
bool	*is_cell_y_flipped* (*int* x, *int* y) const
Vector2	*map_to_world* (*Vector2* map_position, *bool* ignore_half_ofs=false) const
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))
void	*set_cellv* (*Vector2* position, *int* tile, *bool* flip_x=false, *bool* flip_y=false, *bool* transpose=false)
void	*set_collision_layer_bit* (*int* bit, *bool* value)
void	*set_collision_mask_bit* (*int* bit, *bool* value)
void	*update_bitmask_area* (*Vector2* position)
void	*update_bitmask_region* (*Vector2* start=Vector2(0, 0), *Vector2* end=Vector2(0, 0))
Vector2	*world_to_map* (*Vector2* world_position) const

29.417.3 Signals

- **settings_changed** ( )

Emitted when a tilemap setting has changed.

29.417.4 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*.

29.417.5 Numeric Constants

- **INVALID_CELL = -1** — Returned when a cell doesn't exist.

29.417.6 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** — Tile origin at its bottom-left corner.

enum **Mode**

- **MODE_SQUARE = 0** — Orthogonal orientation mode.
- **MODE_ISOMETRIC = 1** — Isometric orientation mode.

- **MODE_CUSTOM = 2** — Custom orientation mode.

29.417.7 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.

29.417.8 Member Function Description

- void **clear** ( )

Clears all cells.

- void **fix_invalid_tiles** ( )

Clears cells that do not exist in the tileset.

- *int* **get_cell** (*int* x, *int* y) const

Returns the tile index of the given cell.

- *int* **get_cellv** (*Vector2* position) const

Returns the tile index of the cell given by a Vector2.

- *bool* **get_collision_layer_bit** (*int* bit) const

Returns `true` if the given collision layer bit is set.

- *bool* **get_collision_mask_bit** (*int* bit) const

Returns `true` if the given collision mask bit is set.

- *Array* **get_used_cells** ( ) const

Returns 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

Returns an array of all cells with the given tile id.

- *Rect2* **get_used_rect** ( )

Returns a rectangle enclosing the used (non-empty) tiles of the map.

- *bool* **is_cell_transposed** (*int* x, *int* y) const

Returns `true` if the given cell is transposed, i.e. the x and y axes are swapped.

- *bool* **is_cell_x_flipped** (*int* x, *int* y) const

Returns `true` if the given cell is flipped in the x axis.

- *bool* **is_cell_y_flipped** (*int* x, *int* y) const

Returns `true` if the given cell is flipped in the y axis.

- *Vector2* **map_to_world** (*Vector2* map_position, *bool* ignore_half_ofs=false) const

Returns the global position corresponding to the given tilemap (grid-based) coordinates.

Optionally, the tilemap's 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))

Sets the tile index for the cell given by a Vector2.

An index of `-1` clears the cell.

Optionally, the tile can also be flipped, transposed, or given autotile coordinates.

- void **set_cellv** (*Vector2* position, *int* tile, *bool* flip_x=false, *bool* flip_y=false, *bool* transpose=false)

Sets the tile index for the given cell.

An index of `-1` clears the cell.

Optionally, the tile can also be flipped or transposed.

- void **set_collision_layer_bit** (*int* bit, *bool* value)

Sets the given collision layer bit.

- void **set_collision_mask_bit** (*int* bit, *bool* value)

Sets the given collision mask bit.

- 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

Returns the tilemap (grid-based) coordinatescorresponding to the given global position.

29.418 TileSet

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.418.1 Brief Description

Tile library for tilemaps.

29.418.2 Member Functions

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
BitmaskMode	*autotile_get_bitmask_mode* (*int* id) const
void	*autotile_set_bitmask_mode* (*int* id, *BitmaskMode* mode)
void	*clear* ( )
void	*create_tile* (*int* id)
int	*find_tile_by_name* (*String* name) const
int	*get_last_unused_tile_id* ( ) const
Array	*get_tiles_ids* ( ) const

Co

Table 22 – continued from previous page

void	*remove_tile* (*int* id)
void	*tile_add_shape* (*int* id, *Shape2D* shape, *Transform2D* shape_transform, *bool* one_way=false, *Vector2* autotile_
OccluderPolygon2D	*tile_get_light_occluder* (*int* id) const
ShaderMaterial	*tile_get_material* (*int* id) const
String	*tile_get_name* (*int* id) const
NavigationPolygon	*tile_get_navigation_polygon* (*int* id) const
Vector2	*tile_get_navigation_polygon_offset* (*int* id) const
Texture	*tile_get_normal_map* (*int* id) const
Vector2	*tile_get_occluder_offset* (*int* id) const
Rect2	*tile_get_region* (*int* id) const
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
Texture	*tile_get_texture* (*int* id) const
Vector2	*tile_get_texture_offset* (*int* id) const
TileMode	*tile_get_tile_mode* (*int* id) const
int	*tile_get_z_index* (*int* id) const
void	*tile_set_light_occluder* (*int* id, *OccluderPolygon2D* light_occluder)
void	*tile_set_material* (*int* id, *ShaderMaterial* material)
void	*tile_set_name* (*int* id, *String* name)
void	*tile_set_navigation_polygon* (*int* id, *NavigationPolygon* navigation_polygon)
void	*tile_set_navigation_polygon_offset* (*int* id, *Vector2* navigation_polygon_offset)
void	*tile_set_normal_map* (*int* id, *Texture* normal_map)
void	*tile_set_occluder_offset* (*int* id, *Vector2* occluder_offset)
void	*tile_set_region* (*int* id, *Rect2* region)
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)
void	*tile_set_texture* (*int* id, *Texture* texture)
void	*tile_set_texture_offset* (*int* id, *Vector2* texture_offset)
void	*tile_set_tile_mode* (*int* id, *TileMode* tilemode)
void	*tile_set_z_index* (*int* id, *int* z_index)

29.418.3 Enums

enum **BitmaskMode**

- **BITMASK_2X2 = 0**
- **BITMASK_3X3_MINIMAL = 1**
- **BITMASK_3X3 = 2**

enum **TileMode**

- **SINGLE_TILE = 0**
- **AUTO_TILE = 1**
- **ANIMATED_TILE = 2**

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**

29.418.4 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.

29.418.5 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
- *BitmaskMode* **autotile_get_bitmask_mode** (*int* id) const
- void **autotile_set_bitmask_mode** (*int* id, *BitmaskMode* 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.

- *TileMode* **tile_get_tile_mode** (*int* id) const
- *int* **tile_get_z_index** (*int* id) const
- 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.

- void **tile_set_tile_mode** (*int* id, *TileMode* tilemode)
- void **tile_set_z_index** (*int* id, *int* z_index)

29.419 Timer

Inherits: *Node* < *Object*

Category: Core

29.419.1 Brief Description

A countdown timer.

29.419.2 Member Functions

bool	*is_stopped* ( ) const
void	*start* (*float* time_sec=-1)
void	*stop* ( )

29.419.3 Signals

- **timeout** ( )

Emitted when the Timer reaches 0.

29.419.4 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 unpaused 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.

29.419.5 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.

29.419.6 Description

Counts down a specified interval and emits a signal on reaching 0. Can be set to repeat or "one shot" mode.

29.419.7 Member Function Description

- *bool* **is_stopped** **(** **)** const

Returns `true` if the timer is stopped.

- void **start** **(** *float* time_sec=-1 **)**

Starts the timer. Sets `wait_time` to `time_sec` if `time_sec` > 0. 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.

29.420 ToolButton

Inherits: *Button* < *BaseButton* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.420.1 Brief Description

Flat button helper class.

29.420.2 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)
```

29.421 TouchScreenButton

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.421.1 Brief Description

Button for touch screen devices.

29.421.2 Member Functions

bool	*is_pressed* **( )** const

29.421.3 Signals

- **pressed ( )**

Emitted when the button is pressed (down).

- **released ( )**

Emitted when the button is released (up).

29.421.4 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.

29.421.5 Enums

enum **VisibilityMode**

- **VISIBILITY_ALWAYS = 0** — Always visible.
- **VISIBILITY_TOUCHSCREEN_ONLY = 1** — Visible on touch screens only.

29.421.6 Description

Button for touch screen devices. You can set it to be visible on all screens, or only on touch devices.

29.421.7 Member Function Description

- *bool* **is_pressed** ( ) const

Returns `true` if this button is currently pressed.

29.422 Transform

Category: Built-In Types

29.422.1 Brief Description

3D Transformation. 3x4 matrix.

29.422.2 Member Functions

Transform	Transform (*Vector3* x_axis, *Vector3* y_axis, *Vector3* z_axis, *Vector3* origin)
Transform	Transform (*Basis* basis, *Vector3* origin)
Transform	Transform (*Transform2D* from)
Transform	Transform (*Quat* from)
Transform	Transform (*Basis* from)
Transform	*affine_inverse* ( )
Transform	*interpolate_with* (*Transform* transform, *float* weight)
Transform	*inverse* ( )
Transform	*looking_at* (*Vector3* target, *Vector3* up)
Transform	*orthonormalized* ( )
Transform	*rotated* (*Vector3* axis, *float* phi)
Transform	*scaled* (*Vector3* scale)
Transform	*translated* (*Vector3* ofs)
var	*xform* (var v)
var	*xform_inv* (var v)

29.422.3 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.

29.422.4 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.

29.422.5 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)

Returns a copy of the transform rotated such that its -Z axis points towards the `target` position.

The transform will first be rotated around the given `up` vector, and then fully aligned to the target by a further rotation around an axis perpendicular to both the `target` and `up` vectors.

Operations take place in global space.

- *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 *Vector3*, *Plane*, or *AABB* by this transform.

- var **xform_inv** (var v)

Inverse-transforms the given *Vector3*, *Plane*, or *AABB* by this transform.

29.423 Transform2D

Category: Built-In Types

29.423.1 Brief Description

2D Transformation. 3x2 matrix.

29.423.2 Member Functions

Transform2D	Transform2D (*Transform* from)
Transform2D	Transform2D (*Vector2* x_axis, *Vector2* y_axis, *Vector2* origin)
Transform2D	Transform2D (*float* rotation, *Vector2* position)
Transform2D	*affine_inverse* ( )
Vector2	*basis_xform* (*Vector2* v)
Vector2	*basis_xform_inv* (*Vector2* v)
Vector2	*get_origin* ( )
float	*get_rotation* ( )
Vector2	*get_scale* ( )
Transform2D	*interpolate_with* (*Transform2D* transform, *float* weight)
Transform2D	*inverse* ( )
Transform2D	*orthonormalized* ( )
Transform2D	*rotated* (*float* phi)
Transform2D	*scaled* (*Vector2* scale)
Transform2D	*translated* (*Vector2* offset)
var	*xform* (var v)
var	*xform_inv* (var v)

29.423.3 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.

29.423.4 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.

29.423.5 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.

- *Vector2* **basis_xform** (*Vector2* v)

Transforms the given vector by this transform's basis (no translation).

- *Vector2* **basis_xform_inv** (*Vector2* 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.

- var **xform** (var v)

Transforms the given *Vector2* or *Rect2* by this transform.

- var **xform_inv** (var v)

Inverse-transforms the given *Vector2* or *Rect2* by this transform.

29.424 Translation

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *PHashTranslation*

Category: Core

29.424.1 Brief Description

Language Translation.

29.424.2 Member Functions

void	*add_message* (*String* src_message, *String* xlated_message)
void	*erase_message* (*String* src_message)
String	*get_message* (*String* src_message) const
int	*get_message_count* ( ) const
PoolStringArray	*get_message_list* ( ) const

29.424.3 Member Variables

- *String* **locale**

29.424.4 Description

Translations are resources that can be loaded/unloaded on demand. They map a string to another string.

29.424.5 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).

29.425 TranslationServer

Inherits: *Object*

Category: Core

29.425.1 Brief Description

Server that manages all translations. Translations can be set to it and removed from it.

29.425.2 Member Functions

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

29.425.3 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

29.426 Tree

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.426.1 Brief Description

Control to show a tree of items.

29.426.2 Member Functions

bool	*are_column_titles_visible* ( ) const
void	*clear* ( )
Object	*create_item* (*Object* parent=null, *int* idx=-1)
void	*ensure_cursor_is_visible* ( )
int	*get_column_at_position* (*Vector2* position) const
String	*get_column_title* (*int* column) const
int	*get_column_width* (*int* column) const
Rect2	*get_custom_popup_rect* ( ) const
int	*get_drop_section_at_position* (*Vector2* position) const
TreeItem	*get_edited* ( ) const
int	*get_edited_column* ( ) const
Rect2	*get_item_area_rect* (*Object* item, *int* column=-1) const
TreeItem	*get_item_at_position* (*Vector2* position) const
TreeItem	*get_next_selected* (*Object* from)
int	*get_pressed_button* ( ) const
TreeItem	*get_root* ( )
Vector2	*get_scroll* ( ) const
TreeItem	*get_selected* ( ) const
int	*get_selected_column* ( ) const
void	*set_column_expand* (*int* column, *bool* expand)
void	*set_column_min_width* (*int* column, *int* min_width)
void	*set_column_title* (*int* column, *String* title)
void	*set_column_titles_visible* (*bool* visible)

29.426.3 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's label is 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's icon 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** ( )

29.426.4 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.

29.426.5 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.

29.426.6 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")
```

29.426.7 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 the 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.

29.427 TreeItem

Inherits: *Object*

Category: Core

29.427.1 Brief Description

Control for a single item inside a *Tree*.

29.427.2 Member Functions

void	*add_button* (*int* column, *Texture* button, *int* button_idx=-1, *bool* disabled=false, *String* tooltip="")
void	*clear_custom_bg_color* (*int* column)
void	*clear_custom_color* (*int* column)
void	*deselect* (*int* column)
void	*erase_button* (*int* column, *int* button_idx)
Texture	*get_button* (*int* column, *int* button_idx) const
int	*get_button_count* (*int* column) const
TreeCellMode	*get_cell_mode* (*int* column) const
TreeItem	*get_children* ( )
Color	*get_custom_bg_color* (*int* column) const
bool	*get_expand_right* (*int* column) const
Texture	*get_icon* (*int* column) const
int	*get_icon_max_width* (*int* column) const
Rect2	*get_icon_region* (*int* column) const
Variant	*get_metadata* (*int* column) const
TreeItem	*get_next* ( )
TreeItem	*get_next_visible* ( )
TreeItem	*get_parent* ( )
TreeItem	*get_prev* ( )
TreeItem	*get_prev_visible* ( )
float	*get_range* (*int* column) const
Dictionary	*get_range_config* (*int* column)
String	*get_text* (*int* column) const
TextAlign	*get_text_align* (*int* column) const
String	*get_tooltip* (*int* column) const
bool	*is_button_disabled* (*int* column, *int* button_idx) const
bool	*is_checked* (*int* column) const
bool	*is_custom_set_as_button* (*int* column) const
bool	*is_editable* (*int* column)
bool	*is_selectable* (*int* column) const
bool	*is_selected* (*int* column)
void	*move_to_bottom* ( )
void	*move_to_top* ( )
void	*remove_child* (*Object* child)
void	*select* (*int* column)
void	*set_button* (*int* column, *int* button_idx, *Texture* button)
void	*set_cell_mode* (*int* column, *TreeCellMode* mode)
void	*set_checked* (*int* column, *bool* checked)
void	*set_custom_as_button* (*int* column, *bool* enable)
void	*set_custom_bg_color* (*int* column, *Color* color, *bool* just_outline=false)
void	*set_custom_color* (*int* column, *Color* color)
void	*set_custom_draw* (*int* column, *Object* object, *String* callback)
void	*set_editable* (*int* column, *bool* enabled)
void	*set_expand_right* (*int* column, *bool* enable)
void	*set_icon* (*int* column, *Texture* texture)
void	*set_icon_max_width* (*int* column, *int* width)
void	*set_icon_region* (*int* column, *Rect2* region)
void	*set_metadata* (*int* column, *Variant* meta)

Continued on next page

Table 23 – continued from previous page

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)
void	*set_text* (*int* column, *String* text)
void	*set_text_align* (*int* column, *TextAlign* text_align)
void	*set_tooltip* (*int* column, *String* tooltip)

29.427.3 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.

29.427.4 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()`.

29.427.5 Description

Control for a single item inside a *Tree*. May have child `TreeItem`s and be styled as well as contain buttons.

29.427.6 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_buton_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.

- *TreeCellMode* **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.

- *TextAlign* **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, *TreeCellMode* 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, *TextAlign* 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.

29.428 TriangleMesh

Inherits: *Reference* < *Object*

Category: Core

29.428.1 Brief Description

29.429 Tween

Inherits: *Node* < *Object*

Category: Core

29.429.1 Brief Description

Smoothly animates a node's properties over time.

29.429.2 Member Functions

bool	*follow_method* (*Object* object, *String* method, *Variant* initial_val, *Object* target, *String* target_method, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
bool	*follow_property* (*Object* object, *NodePath* property, *Variant* initial_val, *Object* target, *NodePath* target_property, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
float	*get_runtime* ( ) const
bool	*interpolate_callback* (*Object* object, *float* duration, *String* callback, *Variant* arg1=null, *Variant* arg2=null, *Variant* arg3=null, *Variant* arg4=null, *Variant* arg5=null)
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)
bool	*interpolate_method* (*Object* object, *String* method, *Variant* initial_val, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
bool	*interpolate_property* (*Object* object, *NodePath* property, *Variant* initial_val, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
bool	*is_active* ( ) const
bool	*remove* (*Object* object, *String* key="")
bool	*remove_all* ( )
bool	*reset* (*Object* object, *String* key="")
bool	*reset_all* ( )
bool	*resume* (*Object* object, *String* key="")
bool	*resume_all* ( )
bool	*seek* (*float* time)
void	*set_active* (*bool* active)
bool	*start* ( )
bool	*stop* (*Object* object, *String* key="")
bool	*stop_all* ( )
bool	*targeting_method* (*Object* object, *String* method, *Object* initial, *String* initial_method, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
bool	*targeting_property* (*Object* object, *NodePath* property, *Object* initial, *NodePath* initial_val, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)
float	*tell* ( ) const

29.429.3 Signals

- **tween_completed** (*Object* object, *NodePath* key)

Emitted when a tween ends.

- **tween_started** (*Object* object, *NodePath* key)

Emitted when a tween starts.

- **tween_step** (*Object* object, *NodePath* key, *float* elapsed, *Object* value)

Emitted at each step of the animation.

29.429.4 Member Variables

- *TweenProcessMode* **playback_process_mode** - The tween's animation process thread. See TweenProcessMode. Default value: TWEEN_PROCESS_IDLE.
- *float* **playback_speed** - The tween's speed multiplier. For example, set it to `1.0` for normal speed, `2.0` for two times normal speed, or `0.5` for half of the normal speed. A value of `0` pauses the animation, but see also *set_active* or *stop_all* for this.
- *bool* **repeat** - If `true` the tween loops.

29.429.5 Enums

enum **TweenProcessMode**

- **TWEEN_PROCESS_PHYSICS = 0** — The tween updates with the `_physics_process` callback.
- **TWEEN_PROCESS_IDLE = 1** — The tween updates with the `_process` callback.

enum **EaseType**

- **EASE_IN = 0** — The interpolation starts slowly and speeds up towards the end.
- **EASE_OUT = 1** — The interpolation starts quickly and slows down towards the end.
- **EASE_IN_OUT = 2** — A combination of EASE_IN and EASE_OUT. The interpolation is slowest at both ends.
- **EASE_OUT_IN = 3** — A combination of EASE_IN and EASE_OUT. The interpolation is fastest at both ends.

enum **TransitionType**

- **TRANS_LINEAR = 0** — The animation is interpolated linearly.
- **TRANS_SINE = 1** — The animation is interpolated using a sine function.
- **TRANS_QUINT = 2** — The animation is interpolated with a quintic (to the power of 5) function.
- **TRANS_QUART = 3** — The animation is interpolated with a quartic (to the power of 4) function.
- **TRANS_QUAD = 4** — The animation is interpolated with a quadratic (to the power of 2) function.
- **TRANS_EXPO = 5** — The animation is interpolated with an exponential (to the power of x) function.
- **TRANS_ELASTIC = 6** — The animation is interpolated with elasticity, wiggling around the edges.
- **TRANS_CUBIC = 7** — The animation is interpolated with a cubic (to the power of 3) function.
- **TRANS_CIRC = 8** — The animation is interpolated with a function using square roots.
- **TRANS_BOUNCE = 9** — The animation is interpolated by bouncing at the end.
- **TRANS_BACK = 10** — The animation is interpolated backing out at ends.

29.429.6 Description

Tweens are useful for animations requiring a numerical property to be interpolated over a range of values. The name *tween* comes from *in-betweening*, an animation technique where you specify *keyframes* and the computer interpolates the frames that appear between them.

Here is a brief usage example that causes a 2D node to move smoothly between two positions:

```
var tween = get_node("Tween")
tween.interpolate_property($Node2D, "position",
                Vector2(0, 0), Vector2(100, 100), 1,
                Tween.TRANS_LINEAR, Tween.EASE_IN_OUT)
tween.start()
```

Many methods require a property name, such as "position" above. You can find the correct property name by hovering over the property in the Inspector.

Many of the methods accept `trans_type` and `ease_type`. The first accepts an TransitionType constant, and refers to the way the timing of the animation is handled (see `http://easings.net/` for some examples). The second accepts an EaseType 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 TransitionType constants with EASE_IN_OUT, and use the one that looks best.

29.429.7 Member Function Description

- *bool* **follow_method** (*Object* object, *String* method, *Variant* initial_val, *Object* target, *String* target_method, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Follows `method` of `object` and applies the returned value on `target_method` of `target`, beginning from `initial_val` for `duration` seconds, `delay` later. Methods are called with consecutive values.

> Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *bool* **follow_property** (*Object* object, *NodePath* property, *Variant* initial_val, *Object* target, *NodePath* target_property, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Follows `property` of `object` and applies it on `target_property` of `target`, beginning from `initial_val` for `duration` seconds, `delay` seconds later.

> Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *float* **get_runtime** ( ) const

Returns the total time needed for all tweens to end. If you have two tweens, one lasting 10 seconds 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)

Calls `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)

Calls `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, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Animates `method` of `object` from `initial_val` to `final_val` for `duration` seconds, `delay` seconds later. Methods are called with consecutive values.

> Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *bool* **interpolate_property** (*Object* object, *NodePath* property, *Variant* initial_val, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Animates `property` of `object` from `initial_val` to `final_val` for `duration` seconds, `delay` seconds later.

> Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *bool* **is_active** ( ) const

Returns `true` if any tweens are currently running. Note that this method doesn't consider tweens that have ended.

- *bool* **remove** (*Object* object, *String* key="")

Stops animation and removes a tween, given its object and property/method pair. By default, all tweens are removed, unless `key` is specified.

- *bool* **remove_all** ( )

Stops animation and removes all tweens.

- *bool* **reset** (*Object* object, *String* key="")

Resets a tween to its initial value (the one given, not the one before the tween), given its object and property/method pair. By default, all tweens are removed, unless `key` is specified.

- *bool* **reset_all** ( )

Resets all tweens to their initial values (the ones given, not those before the tween).

- *bool* **resume** (*Object* object, *String* key="")

Continues animating a stopped tween, given its object and property/method pair. By default, all tweens are resumed, unless `key` is specified.

- *bool* **resume_all** ( )

Continues animating all stopped tweens.

- *bool* **seek** (*float* time)

Sets the interpolation to the given `time` in seconds.

- void **set_active** (*bool* active)

Activates/deactivates the tween. See also *stop_all* and *resume_all*.

- *bool* **start** ( )

Starts the tween. You can define animations both before and after this.

- *bool* **stop** (*Object* object, *String* key="")

Stops a tween, given its object and property/method pair. By default, all tweens are stopped, unless `key` is specified.

- *bool* **stop_all** ( )

Stops animating all tweens.

- *bool* **targeting_method** (*Object* object, *String* method, *Object* initial, *String* initial_method, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Animates `method` of `object` from the value returned by `initial_method` to `final_val` for `duration` seconds, `delay` seconds later. Methods are animated by calling them with consecutive values.

> Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *bool* **targeting_property** (*Object* object, *NodePath* property, *Object* initial, *NodePath* initial_val, *Variant* final_val, *float* duration, *TransitionType* trans_type, *EaseType* ease_type, *float* delay=0)

Animates `property` of `object` from the current value of the `initial_val` property of `initial` to `final_val` for `duration` seconds, `delay` seconds later.

Use TransitionType for `trans_type` and EaseType for `ease_type` parameters. These values control the timing and direction of the interpolation. See the class description for more information

- *float* **tell** ( ) const

Returns the current time of the tween.

29.430 UndoRedo

Inherits: *Object*

Category: Core

29.430.1 Brief Description

Helper to manage UndoRedo in the editor or custom tools.

29.430.2 Member Functions

Variant	*add_do_method* (*Object* object, *String* method) vararg
void	*add_do_property* (*Object* object, *String* property, *Variant* value)
void	*add_do_reference* (*Object* object)
Variant	*add_undo_method* (*Object* object, *String* method) vararg
void	*add_undo_property* (*Object* object, *String* property, *Variant* value)
void	*add_undo_reference* (*Object* object)
void	*clear_history* ( )
void	*commit_action* ( )
void	*create_action* (*String* name, *MergeMode* merge_mode=0)
String	*get_current_action_name* ( ) const
int	*get_version* ( ) const
void	*redo* ( )
void	*undo* ( )

29.430.3 Enums

enum **MergeMode**

- **MERGE_DISABLE = 0**
- **MERGE_ENDS = 1**
- **MERGE_ALL = 2**

29.430.4 Description

Helper to manage UndoRedo in the editor or custom tools. It works by storing calls to functions in both 'do' an 'undo' lists.

Common behavior is to create an action, then add do/undo calls to functions or property changes, then committing the action.

29.430.5 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, *MergeMode* 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** ( )

29.431 Variant

Category: Core

29.431.1 Brief Description

The most important data type in Godot.

29.431.2 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.

29.432 VBoxContainer

Inherits: *BoxContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.432.1 Brief Description

Vertical box container.

29.432.2 Description

Vertical box container. See *BoxContainer*.

29.433 Vector2

Category: Built-In Types

29.433.1 Brief Description

Vector used for 2D math.

29.433.2 Member Functions

Vector2	*Vector2* (*float* x, *float* y)
Vector2	*abs* ( )
float	*angle* ( )
float	*angle_to* (*Vector2* to)
float	*angle_to_point* (*Vector2* to)
float	*aspect* ( )
Vector2	*bounce* (*Vector2* n)
Vector2	*ceil* ( )
Vector2	*clamped* (*float* length)
float	*cross* (*Vector2* with)
Vector2	*cubic_interpolate* (*Vector2* b, *Vector2* pre_a, *Vector2* post_b, *float* t)
float	*distance_squared_to* (*Vector2* to)
float	*distance_to* (*Vector2* to)
float	*dot* (*Vector2* with)
Vector2	*floor* ( )
bool	*is_normalized* ( )
float	*length* ( )
float	*length_squared* ( )
Vector2	*linear_interpolate* (*Vector2* b, *float* t)
Vector2	*normalized* ( )
Vector2	*reflect* (*Vector2* n)
Vector2	*rotated* (*float* phi)
Vector2	*round* ( )
Vector2	*slerp* (*Vector2* b, *float* t)
Vector2	*slide* (*Vector2* n)
Vector2	*snapped* (*Vector2* by)
Vector2	*tangent* ( )

29.433.3 Member Variables

- *float* **x** - The vector's x component.
- *float* **y** - The vector's y component.

29.433.4 Description

2-element structure that can be used to represent positions in 2d space or any other pair of numeric values.

29.433.5 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 vector's angle in radians with respect to the x-axis, or `(1, 0)` vector.

Equivalent to the result of atan2 when called with the vector's x and y as parameters: `atan2(x, y)`.

- *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 **)**

Returns the vector "bounced off" from a plane defined by the given normal.

- *Vector2* **ceil ( )**
- *Vector2* **clamped (** *float* length **)**

Returns the vector with a maximum length.

- *float* **cross (** *Vector2* with **)**

Returns the 2 dimensional analog of the cross product with the given vector.

- *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 returns the result at position `t`. `t` is in the range of `0.0 - 1.0`, or 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 `true` if the vector is normalized.

- *float* **length ( )**

Returns the vector's length.

- *float* **length_squared ( )**

Returns the vector's length squared. 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` is in the range of `0.0 - 1.0`, a percentage of how far along the interpolation is.

- *Vector2* **normalized ( )**

Returns the vector scaled to unit length. Equivalent to `v / v.length()`.

- *Vector2* **reflect (** *Vector2* n **)**

Returns the vector reflected from a plane defined by the given normal.

- *Vector2* **rotated (** *float* phi **)**

Returns the vector rotated by `phi` radians.

- *Vector2* **round ( )**
- *Vector2* **slerp (** *Vector2* b, *float* t **)**

Returns the result of SLERP 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.

Both vectors need to be normalized.

- *Vector2* **slide (** *Vector2* n **)**

Returns the component of the vector along a plane defined by the given normal.

- *Vector2* **snapped (** *Vector2* by **)**

Returns the vector snapped to a grid with the given size.

- *Vector2* **tangent ( )**

Returns a perpendicular vector.

29.434 Vector3

Category: Built-In Types

29.434.1 Brief Description

Vector class, which performs basic 3D vector math operations.

29.434.2 Member Functions

Vector3	*Vector3* (*float* x, *float* y, *float* z)
Vector3	*abs* ( )
float	*angle_to* (*Vector3* to)
Vector3	*bounce* (*Vector3* n)
Vector3	*ceil* ( )
Vector3	*cross* (*Vector3* b)
Vector3	*cubic_interpolate* (*Vector3* b, *Vector3* pre_a, *Vector3* post_b, *float* t)
float	*distance_squared_to* (*Vector3* b)
float	*distance_to* (*Vector3* b)
float	*dot* (*Vector3* b)
Vector3	*floor* ( )
Vector3	*inverse* ( )
bool	*is_normalized* ( )
float	*length* ( )
float	*length_squared* ( )
Vector3	*linear_interpolate* (*Vector3* b, *float* t)
int	*max_axis* ( )
int	*min_axis* ( )
Vector3	*normalized* ( )
Basis	*outer* (*Vector3* b)
Vector3	*reflect* (*Vector3* n)
Vector3	*rotated* (*Vector3* axis, *float* phi)
Vector3	*round* ( )
Vector3	*slerp* (*Vector3* b, *float* t)
Vector3	*slide* (*Vector3* n)
Vector3	*snapped* (*Vector3* by)
Basis	*to_diagonal_matrix* ( )

29.434.3 Member Variables

- *float* **x** - The vector's x component.
- *float* **y** - The vector's y component.
- *float* **z** - The vector's z component.

29.434.4 Numeric Constants

- **AXIS_X = 0** — Enumerated value for the X axis. Returned by *max_axis* and *min_axis*.
- **AXIS_Y = 1** — Enumerated value for the Y axis.
- **AXIS_Z = 2** — Enumerated value for the Z axis.

29.434.5 Description

Vector3 is one of the core classes of the engine, and includes several built-in helper functions to perform basic vector math operations.

29.434.6 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 minimum angle to the given vector.

- *Vector3* **bounce (** *Vector3* n **)**

Returns the vector "bounced off" from a plane defined by the given normal.

- *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 `true` if the vector is normalized.

- *float* **length ( )**

Returns the vector's length.

- *float* **length_squared ( )**

Returns the vector's length 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 **)**

Returns the result of the linear interpolation between this vector and `b` by amount `t`. `t` is in the range of `0.0 - 1.0`, a percentage of how far along the interpolation is.

- *int* **max_axis ( )**

Returns the axis of the vector's largest value. See `AXIS_*` constants.

- *int* **min_axis ( )**

Returns the axis of the vector's smallest value. See `AXIS_*` constants.

- *Vector3* **normalized ( )**

Returns the vector scaled to unit length. Equivalent to `v / v.length()`.

- *Basis* **outer (*Vector3* b)**

Returns the outer product with `b`.

- *Vector3* **reflect (*Vector3* n)**

Returns the vector reflected from a plane defined by the given normal.

- *Vector3* **rotated (*Vector3* axis, *float* phi)**

Rotates the vector around a given axis by `phi` radians. The axis must be a normalized vector.

- *Vector3* **round ( )**
- *Vector3* **slerp (*Vector3* b, *float* t)**

Returns the result of SLERP 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.

Both vectors need to be normalized.

- *Vector3* **slide (*Vector3* n)**

Returns the component of the vector along a plane defined by the given normal.

- *Vector3* **snapped (*Vector3* 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.

29.435 VehicleBody

Inherits: *RigidBody* < *PhysicsBody* < *CollisionObject* < *Spatial* < *Node* < *Object*

Category: Core

29.435.1 Brief Description

Physics body that simulates the behaviour of a car.

29.435.2 Member Variables

- *float* **brake** - Slows down the vehicle by applying a braking force. The vehicle is only slowed down if the wheels are in contact with a surface. The force you need to apply to adequately slow down your vehicle depends on the *RigidBody.mass* of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 30 range for hard braking.
- *float* **engine_force** - Accelerates the vehicle by applying an engine force. The vehicle is only speed up if the wheels that have VehicleWheel.set_use_as_traction set to true and are in contact with a surface. The *RigidBody.mass* of the vehicle has an effect on the acceleration of the vehicle. For a vehicle with a mass set to 1000, try a value in the 25 - 50 range for acceleration. Note that the simulation does not take the effect of gears into account, you will need to add logic for this if you wish to simulate gears.

A negative value will result in the vehicle reversing.

- *float* **steering** - The steering angle for the vehicle. Setting this to a non-zero value will result in the vehicle turning when it's moving. Wheels that have VehicleWheel.set_use_as_steering set to true will automatically be rotated.

29.435.3 Description

This nodes implements all the physics logic needed to simulate a car. It is based on the raycast vehicle system commonly found in physics engines. You will need to add a *CollisionShape* for the main body of your vehicle and add *VehicleWheel* nodes for the wheels. You should also add a *MeshInstance* to this node for the 3D model of your car but this model should not include meshes for the wheels. You should control the vehicle by using the *brake*, *engine_force*, and *steering* properties and not change the position or orientation of this node directly.

Note that the origin point of your VehicleBody will determine the center of gravity of your vehicle so it is better to keep this low and move the *CollisionShape* and *MeshInstance* upwards.

29.436 VehicleWheel

Inherits: *Spatial* < *Node* < *Object*

Category: Core

29.436.1 Brief Description

Physics object that simulates the behaviour of a wheel.

29.436.2 Member Functions

float	*get_skidinfo* ( ) const
bool	*is_in_contact* ( ) const

29.436.3 Member Variables

- *float* **damping_compression** - The damping applied to the spring when the spring is being compressed. This value should be between 0.0 (no damping) and 1.0. A value of 0.0 means the car will keep bouncing as the spring keeps its energy. A good value for this is around 0.3 for a normal car, 0.5 for a race car.
- *float* **damping_relaxation** - The damping applied to the spring when relaxing. This value should be between 0.0 (no damping) and 1.0. This value should always be slightly higher than the *damping_compression* property. For a *damping_compression* value of 0.3, try a relaxation value of 0.5
- *float* **suspension_max_force** - The maximum force the spring can resist. This value should be higher than a quarter of the *RigidBody.mass* of the *VehicleBody* or the spring will not carry the weight of the vehicle. Good results are often obtained by a value that is about 3x to 4x this number.
- *float* **suspension_stiffness** - This value defines the stiffness of the suspension. Use a value lower than 50 for an off-road car, a value between 50 and 100 for a race car and try something around 200 for something like a Formula 1 car.
- *float* **suspension_travel** - This is the distance the suspension can travel. As Godot measures are in meters keep this setting relatively low. Try a value between 0.1 and 0.3 depending on the type of car .
- *bool* **use_as_steering** - If true this wheel will be turned when the car steers.
- *bool* **use_as_traction** - If true this wheel transfers engine force to the ground to propel the vehicle forward.
- *float* **wheel_friction_slip** - This determines how much grip this wheel has. It is combined with the friction setting of the surface the wheel is in contact with. 0.0 means no grip, 1.0 is normal grip. For a drift car setup, try setting the grip of the rear wheels slightly lower than the front wheels, or use a lower value to simulate tire wear.

It's best to set this to 1.0 when starting out.

- *float* **wheel_radius** - The radius of the wheel in meters.
- *float* **wheel_rest_length** - This is the distance in meters the wheel is lowered from its origin point. Don't set this to 0.0 and move the wheel into position, instead move the origin point of your wheel (the gizmo in Godot) to the position the wheel will take when bottoming out, then use the rest length to move the wheel down to the position it should be in when the car is in rest.
- *float* **wheel_roll_influence** - This value effects the roll of your vehicle. If set to 0.0 for all wheels your vehicle will be prone to rolling over while a value of 1.0 will resist body roll.

29.436.4 Description

This node needs to be used as a child node of *VehicleBody* and simulates the behaviour of one of its wheels. This node also acts as a collider to detect if the wheel is touching a surface.

29.436.5 Member Function Description

- *float* **get_skidinfo** **(** **)** const

Returns a value between 0.0 and 1.0 that indicates whether this wheel is skidding. 0.0 is not skidding, 1.0 means the wheel has lost grip.

- *bool* **is_in_contact** **(** **)** const

Returns true if this wheel is in contact with a surface.

29.437 VideoPlayer

Inherits: *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.437.1 Brief Description

Control for playing video streams.

29.437.2 Member Functions

String	*get_stream_name* ( ) const
Texture	*get_video_texture* ( )
bool	*is_playing* ( ) const
void	*play* ( )
void	*stop* ( )

29.437.3 Signals

- **finished** ( )

Emitted when playback is finished.

29.437.4 Member Variables

- *int* **audio_track** - The embedded audio track to play.
- *bool* **autoplay** - If `true` playback starts when the scene loads. Default value: `false`.
- *int* **buffering_msec** - Amount of time in milliseconds to store in buffer while playing.
- *String* **bus** - Audio bus to use for sound playback.
- *bool* **expand** - If `true` the video scales to the control size. Default value: `true`.
- *bool* **paused** - If `true` the video is paused.
- *VideoStream* **stream**
- *float* **stream_position** - The current position of the stream, in seconds.
- *float* **volume** - Audio volume as a linear value.
- *float* **volume_db** - Audio volume in dB.

29.437.5 Description

Control node for playing video streams. Supported formats are WebM and OGV Theora.

29.437.6 Member Function Description

- *String* **get_stream_name** ( ) const

Returns the video stream's name.

- *Texture* **get_video_texture** ( )

Returns the current frame as a *Texture*.

- *bool* **is_playing** ( ) const

Returns `true` if the video is playing.

- void **play** ( )

Starts the video playback.

- void **stop** ( )

Stops the video playback.

29.438 VideoStream

Inherits: *Resource* < *Reference* < *Object*

Inherited By: *VideoStreamWebm*, *VideoStreamTheora*

Category: Core

29.438.1 Brief Description

29.439 VideoStreamTheora

Inherits: *VideoStream* < *Resource* < *Reference* < *Object*

Category: Core

29.439.1 Brief Description

29.439.2 Member Functions

String	*get_file* ( )
void	*set_file* (*String* file)

29.439.3 Member Function Description

- *String* **get_file** ( )
- void **set_file** (*String* file)

29.440 VideoStreamWebm

Inherits: *VideoStream* < *Resource* < *Reference* < *Object*

Category: Core

29.440.1 Brief Description

29.440.2 Member Functions

String	*get_file* ( )
void	*set_file* (*String* file)

29.440.3 Member Function Description

- *String* **get_file** ( )
- void **set_file** (*String* file)

29.441 Viewport

Inherits: *Node* < *Object*

Category: Core

29.441.1 Brief Description

Creates a sub-view into the screen.

29.441.2 Member Functions

World	*find_world* ( ) const
World2D	*find_world_2d* ( ) const
Camera	*get_camera* ( ) const
Transform2D	*get_final_transform* ( ) const
Vector2	*get_mouse_position* ( ) const
int	*get_render_info* (*RenderInfo* info)
Vector2	*get_size_override* ( ) const
ViewportTexture	*get_texture* ( ) const
RID	*get_viewport_rid* ( ) const
Rect2	*get_visible_rect* ( ) const
Variant	*gui_get_drag_data* ( ) const
bool	*gui_has_modal_stack* ( ) const
void	*input* (*InputEvent* local_event)
bool	*is_size_override_enabled* ( ) const
bool	*is_size_override_stretch_enabled* ( ) const
void	*set_attach_to_screen_rect* (*Rect2* rect)
void	*set_size_override* (*bool* enable, *Vector2* size=Vector2(-1, -1), *Vector2* margin=Vector2(0, 0))
void	*set_size_override_stretch* (*bool* enabled)
void	*unhandled_input* (*InputEvent* local_event)
void	*update_worlds* ( )
void	*warp_mouse* (*Vector2* to_position)

29.441.3 Signals

- **size_changed** ( )

Emitted when the size of the viewport is changed, whether by *set_size_override*, resize of window, or some other means.

29.441.4 Member Variables

- *bool* **arvr** - If `true` the viewport will be used in AR/VR process. Default value: `false`.
- *bool* **audio_listener_enable_2d** - If `true` the viewport will process 2D audio streams. Default value: `false`.
- *bool* **audio_listener_enable_3d** - If `true` the viewport will process 3D audio streams. Default value: `false`.
- *Transform2D* **canvas_transform** - The canvas transform of the viewport, useful for changing the on-screen positions of all child *CanvasItem*s. This is relative to the global canvas transform of the viewport.
- *DebugDraw* **debug_draw** - The overlay mode for test rendered geometry in debug purposes. Default value: `DEBUG_DRAW_DISABLED`.
- *bool* **disable_3d** - If `true` the viewport will disable 3D rendering. For actual disabling use `usage`. Default value: `false`.
- *Transform2D* **global_canvas_transform** - The global canvas transform of the viewport. The canvas transform is relative to this.

- *bool* **gui_disable_input** - If `true` the viewport will not receive input event. Default value: `false`.
- *bool* **gui_snap_controls_to_pixels** - If `true` the GUI controls on the viewport will lay pixel perfectly. Default value: `true`.
- *bool* **hdr** - If `true` the viewport rendering will receive benefits from High Dynamic Range algorithm. Default value: `true`.
- *bool* **keep_3d_linear** - If `true` the result after 3D rendering will not have a linear to sRGB color conversion applied. This is important when the viewport is used as a render target where the result is used as a texture on a 3D object rendered in another viewport. It is also important if the viewport is used to create data that is not color based (noise, heightmaps, pickmaps, etc.). Do not enable this when the viewport is used as a texture on a 2D object or if the viewport is your final output.
- *MSAA* **msaa** - The multisample anti-aliasing mode. Default value: `MSAA_DISABLED`.
- *bool* **own_world** - If `true` the viewport will use *World* defined in `world` property. Default value: `false`.
- *bool* **physics_object_picking** - If `true` the objects rendered by viewport become subjects of mouse picking process. Default value: `false`.
- *ClearMode* **render_target_clear_mode** - The clear mode when viewport used as a render target. Default value: `CLEAR_MODE_ALWAYS`.
- *UpdateMode* **render_target_update_mode** - The update mode when viewport used as a render target. Default value: `UPDATE_WHEN_VISIBLE`.
- *bool* **render_target_v_flip** - If `true` the result of rendering will be flipped vertically. Default value: `false`.
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_0** - The subdivision amount of first quadrant on shadow atlas. Default value: `SHADOW_ATLAS_QUADRANT_SUBDIV_4`.
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_1** - The subdivision amount of second quadrant on shadow atlas. Default value: `SHADOW_ATLAS_QUADRANT_SUBDIV_4`.
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_2** - The subdivision amount of third quadrant on shadow atlas. Default value: `SHADOW_ATLAS_QUADRANT_SUBDIV_16`.
- *ShadowAtlasQuadrantSubdiv* **shadow_atlas_quad_3** - The subdivision amount of fourth quadrant on shadow atlas. Default value: `SHADOW_ATLAS_QUADRANT_SUBDIV_64`.
- *int* **shadow_atlas_size** - The resolution of shadow atlas. Both width and height is equal to one value.
- *Vector2* **size** - The width and height of viewport.
- *bool* **transparent_bg** - If `true` the viewport should render its background as transparent. Default value: `false`.
- *Usage* **usage** - The rendering mode of viewport. Default value: `USAGE_3D`.
- *World* **world** - The custom *World* which can be used as 3D environment source.
- *World2D* **world_2d** - The custom *World2D* which can be used as 2D environment source.

29.441.5 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** — Always update the render target.

enum **RenderInfo**

- **RENDER_INFO_OBJECTS_IN_FRAME = 0** — Amount of objects in frame.
- **RENDER_INFO_VERTICES_IN_FRAME = 1** — Amount of vertices in frame.
- **RENDER_INFO_MATERIAL_CHANGES_IN_FRAME = 2** — Amount of material changes in frame.
- **RENDER_INFO_SHADER_CHANGES_IN_FRAME = 3** — Amount of shader changes in frame.
- **RENDER_INFO_SURFACE_CHANGES_IN_FRAME = 4** — Amount of surface changes in frame.
- **RENDER_INFO_DRAW_CALLS_IN_FRAME = 5** — Amount of draw calls in frame.
- **RENDER_INFO_MAX = 6** — Enum limiter. Do not use it directly.

enum **MSAA**

- **MSAA_DISABLED = 0** — Multisample anti-aliasing mode disabled. This is the default value.
- **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** — Objects are displayed normally.
- **DEBUG_DRAW_UNSHADED = 1** — Objects are displayed without light information.
- **DEBUG_DRAW_OVERDRAW = 2**
- **DEBUG_DRAW_WIREFRAME = 3** — Objects are displayed in wireframe style.

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** — Enum limiter. Do not use it directly.

29.441.6 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.

29.441.7 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** (*RenderInfo* info)

Get the specific information about the viewport from rendering pipeline.

- *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.

29.442 ViewportContainer

Inherits: *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.442.1 Brief Description

29.442.2 Member Variables

- *bool* **stretch**
- *int* **stretch_shrink**

29.443 ViewportTexture

Inherits: *Texture* < *Resource* < *Reference* < *Object*

Category: Core

29.443.1 Brief Description

29.443.2 Member Variables

- *NodePath* **viewport_path**

29.444 VisibilityEnabler

Inherits: *VisibilityNotifier* < *Spatial* < *Node* < *Object*

Category: Core

29.444.1 Brief Description

Enable certain nodes only when visible.

29.444.2 Member Variables

- *bool* **freeze_bodies**
- *bool* **pause_animations**

29.444.3 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**

29.444.4 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.

29.445 VisibilityEnabler2D

Inherits: *VisibilityNotifier2D* < *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.445.1 Brief Description

Enable certain nodes only when visible.

29.445.2 Member Variables

- *bool* **freeze_bodies**
- *bool* **pause_animated_sprites**
- *bool* **pause_animations**
- *bool* **pause_particles**
- *bool* **physics_process_parent**
- *bool* **process_parent**

29.445.3 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**

29.445.4 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.

29.446 VisibilityNotifier

Inherits: *Spatial* **<** *Node* **<** *Object*

Inherited By: *VisibilityEnabler*

Category: Core

29.446.1 Brief Description

Detects when the node is visible on screen.

29.446.2 Member Functions

bool	*is_on_screen* **(** **)** const

29.446.3 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.

29.446.4 Member Variables

- *AABB* **aabb** - The VisibilityNotifier's bounding box.

29.446.5 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.

29.446.6 Member Function Description

- *bool* **is_on_screen** **(** **)** const

If `true` the bounding box is on the screen.

29.447 VisibilityNotifier2D

Inherits: *Node2D* **<** *CanvasItem* **<** *Node* **<** *Object*

Inherited By: *VisibilityEnabler2D*

Category: Core

29.447.1 Brief Description

Detects when the node is visible on screen.

29.447.2 Member Functions

bool	*is_on_screen* ( ) const

29.447.3 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.

29.447.4 Member Variables

- *Rect2* **rect** - The VisibilityNotifier2D's bounding rectangle.

29.447.5 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.

29.447.6 Member Function Description

- *bool* **is_on_screen** ( ) const

If `true` the bounding rectangle is on the screen.

29.448 VisualInstance

Inherits: *Spatial* < *Node* < *Object*

Inherited By: *BakedLightmap*, *Light*, *ReflectionProbe*, *CSGShape*, *GIProbe*, *GeometryInstance*

Category: Core

29.448.1 Brief Description

29.448.2 Member Functions

AABB	*get_aabb* ( ) const
AABB	*get_transformed_aabb* ( ) const
void	*set_base* (*RID* base)

29.448.3 Member Variables

- *int* **layers** - The render layer(s) this VisualInstance is drawn on.

This object will only be visible for *Camera*s whose cull mask includes the render object this VisualInstance is set to.

29.448.4 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.

29.449 VisualScript

Inherits: *Script* < *Resource* < *Reference* < *Object*

Category: Core

29.449.1 Brief Description

A script implemented in the Visual Script programming environment.

29.449.2 Member Functions

void	*add_custom_signal* (*String* name)
void	*add_function* (*String* name)
void	*add_node* (*String* func, *int* id, *VisualScriptNode* node, *Vector2* position=Vector2(0, 0))
void	*add_variable* (*String* name, *Variant* default_value=null, *bool* export=false)

Continued on next page

Table 24 – continued from previous page

void	*custom_signal_add_argument* (*String* name, *Variant.Type* type, *String* argname, *int* index=-1)
int	*custom_signal_get_argument_count* (*String* name) const
String	*custom_signal_get_argument_name* (*String* name, *int* argidx) const
Variant.Type	*custom_signal_get_argument_type* (*String* name, *int* argidx) const
void	*custom_signal_remove_argument* (*String* name, *int* argidx)
void	*custom_signal_set_argument_name* (*String* name, *int* argidx, *String* argname)
void	*custom_signal_set_argument_type* (*String* name, *int* argidx, *Variant.Type* type)
void	*custom_signal_swap_argument* (*String* name, *int* argidx, *int* withidx)
void	*data_connect* (*String* func, *int* from_node, *int* from_port, *int* to_node, *int* to_port)
void	*data_disconnect* (*String* func, *int* from_node, *int* from_port, *int* to_node, *int* to_port)
int	*get_function_node_id* (*String* name) const
Vector2	*get_function_scroll* (*String* name) const
VisualScriptNode	*get_node* (*String* func, *int* id) const
Vector2	*get_node_position* (*String* func, *int* id) const
Variant	*get_variable_default_value* (*String* name) const
bool	*get_variable_export* (*String* name) const
Dictionary	*get_variable_info* (*String* name) const
bool	*has_custom_signal* (*String* name) const
bool	*has_data_connection* (*String* func, *int* from_node, *int* from_port, *int* to_node, *int* to_port) const
bool	*has_function* (*String* name) const
bool	*has_node* (*String* func, *int* id) const
bool	*has_sequence_connection* (*String* func, *int* from_node, *int* from_output, *int* to_node) const
bool	*has_variable* (*String* name) const
void	*remove_custom_signal* (*String* name)
void	*remove_function* (*String* name)
void	*remove_node* (*String* func, *int* id)
void	*remove_variable* (*String* name)
void	*rename_custom_signal* (*String* name, *String* new_name)
void	*rename_function* (*String* name, *String* new_name)
void	*rename_variable* (*String* name, *String* new_name)
void	*sequence_connect* (*String* func, *int* from_node, *int* from_output, *int* to_node)
void	*sequence_disconnect* (*String* func, *int* from_node, *int* from_output, *int* to_node)
void	*set_function_scroll* (*String* name, *Vector2* ofs)
void	*set_instance_base_type* (*String* type)
void	*set_node_position* (*String* func, *int* id, *Vector2* position)
void	*set_variable_default_value* (*String* name, *Variant* value)
void	*set_variable_export* (*String* name, *bool* enable)
void	*set_variable_info* (*String* name, *Dictionary* value)

29.449.3 Signals

- **node_ports_changed** (*String* function, *int* id)

Emitted when the ports of a node are changed.

29.449.4 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.

29.449.5 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, *Variant.Type* 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.

- *Variant.Type* **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, *Variant.Type* 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*.

29.450 VisualScriptBasicTypeConstant

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.450.1 Brief Description

A Visual Script node representing a constant from the base types.

29.450.2 Member Variables

- *Variant.Type* **basic_type** - The type to get the constant from.
- *String* **constant** - The name of the constant to return.

29.450.3 Description

A Visual Script node representing a constant from base types, such as Vector3.AXIS_X.

29.451 VisualScriptBuiltinFunc

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.451.1 Brief Description

A Visual Script node used to call built-in functions.

29.451.2 Member Variables

- *BuiltinFunc* **function** - The function to be executed.

29.451.3 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 the mathematical constant **e** raised to the specified power of the input. **e** has an approximate value of 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) * 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 `th`) 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(input, range_low), 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 *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.

29.451.4 Description

A built-in function used inside a *VisualScript*. It is usually a math function or an utility function.

See also *@GDScript*, for the same functions in the GDScript language.

29.452 VisualScriptClassConstant

Inherits: *VisualScriptNode* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.452.1 Brief Description

Gets a constant from a given class.

29.452.2 Member Variables

- *String* **base_type** - The constant's parent class.
- *String* **constant** - The constant to return. See the given class for its available constants.

29.452.3 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`

29.453 VisualScriptComment

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.453.1 Brief Description

A Visual Script node used to annotate the script.

29.453.2 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.

29.453.3 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.

29.454 VisualScriptCondition

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.454.1 Brief Description

A Visual Script node which branches the flow.

29.454.2 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`

29.455 VisualScriptConstant

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.455.1 Brief Description

Gets a contant's value.

29.455.2 Member Variables

- *Variant.Type* **type** - The constant's type.
- *Variant* **value** - The constant's value.

29.455.3 Description

This node returns a constant's value.

Input Ports:

none

Output Ports:

- Data (variant): `get`

29.456 VisualScriptConstructor

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.456.1 Brief Description

A Visual Script node which calls a base type constructor.

29.456.2 Member Functions

Dictionary	*get_constructor* ( ) const
Variant.Type	*get_constructor_type* ( ) const
void	*set_constructor* (*Dictionary* constructor)
void	*set_constructor_type* (*Variant.Type* type)

29.456.3 Description

A Visual Script node which calls a base type constructor. It can be used for type conversion as well.

29.456.4 Member Function Description

- *Dictionary* **get_constructor** ( ) const
- *Variant.Type* **get_constructor_type** ( ) const
- void **set_constructor** (*Dictionary* constructor)
- void **set_constructor_type** (*Variant.Type* type)

29.457 VisualScriptCustomNode

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.457.1 Brief Description

A scripted Visual Script node.

29.457.2 Member Functions

String	*_get_caption* ( ) virtual
String	*_get_category* ( ) virtual
int	*_get_input_value_port_count* ( ) virtual
String	*_get_input_value_port_name* (*int* idx) virtual
int	*_get_input_value_port_type* (*int* idx) virtual
int	*_get_output_sequence_port_count* ( ) virtual
String	*_get_output_sequence_port_text* (*int* idx) virtual
int	*_get_output_value_port_count* ( ) virtual
String	*_get_output_value_port_name* (*int* idx) virtual
int	*_get_output_value_port_type* (*int* idx) virtual
String	*_get_text* ( ) virtual
int	*_get_working_memory_size* ( ) virtual
bool	*_has_input_sequence_port* ( ) virtual
Variant	*_step* (*Array* inputs, *Array* outputs, *int* start_mode, *Array* working_mem) virtual

29.457.3 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*.

29.457.4 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.

29.457.5 Description

A custom Visual Script node which can be scripted in powerful ways.

29.457.6 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.

29.458 VisualScriptDeconstruct

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.458.1 Brief Description

A Visual Script node which deconstructs a base type instance into its parts.

29.458.2 Member Variables

- *Variant.Type* **type** - The type to deconstruct.

29.458.3 Description

A Visual Script node which deconstructs a base type instance into its parts.

29.459 VisualScriptEditor

Inherits: *Object*

Category: Core

29.459.1 Brief Description

29.459.2 Member Functions

void	*add_custom_node* (*String* name, *String* category, *Script* script)
void	*remove_custom_node* (*String* name, *String* category)

29.459.3 Signals

- **custom_nodes_updated** ( )

Emitted when a custom Visual Script node is added or removed.

29.459.4 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.

29.460 VisualScriptEmitSignal

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.460.1 Brief Description

Emits a specified signal.

29.460.2 Member Variables

- *String* **signal** - The signal to emit.

29.460.3 Description

Emits a specified signal when it is executed.

Input Ports:

- Sequence: `emit`

Output Ports:

- Sequence

29.461 VisualScriptEngineSingleton

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.461.1 Brief Description

A Visual Script node returning a singleton from *@GlobalScope*

29.461.2 Member Variables

- *String* **constant** - The singleton's name.

29.461.3 Description

A Visual Script node returning a singleton from *@GlobalScope*

29.462 VisualScriptExpression

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.462.1 Brief Description

29.463 VisualScriptFunction

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.463.1 Brief Description

29.464 VisualScriptFunctionCall

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.464.1 Brief Description

29.464.2 Member Variables

- *String* **base_script**
- *String* **base_type**
- *Variant.Type* **basic_type**
- *CallMode* **call_mode**
- *String* **function**
- *NodePath* **node_path**
- *RPCCallMode* **rpc_call_mode**
- *String* **singleton**
- *int* **use_default_args**
- *bool* **validate**

29.464.3 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**

29.465 VisualScriptFunctionState

Inherits: *Reference* < *Object*

Category: Core

29.465.1 Brief Description

29.465.2 Member Functions

void	*connect_to_signal* (*Object* obj, *String* signals, *Array* args)
bool	*is_valid* ( ) const
Variant	*resume* (*Array* args=null)

29.465.3 Member Function Description

- void **connect_to_signal** (*Object* obj, *String* signals, *Array* args)
- *bool* **is_valid** ( ) const
- *Variant* **resume** (*Array* args=null)

29.466 VisualScriptGlobalConstant

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.466.1 Brief Description

29.466.2 Member Variables

- *int* **constant**

29.467 VisualScriptIndexGet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.467.1 Brief Description

29.468 VisualScriptIndexSet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.468.1 Brief Description

29.469 VisualScriptInputAction

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.469.1 Brief Description

29.469.2 Member Variables

- *String* **action**
- *Mode* **mode**

29.469.3 Enums

enum **Mode**

- **MODE_PRESSED = 0**
- **MODE_RELEASED = 1**
- **MODE_JUST_PRESSED = 2**
- **MODE_JUST_RELEASED = 3**

29.470 VisualScriptIterator

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.470.1 Brief Description

Steps through items in a given input.

29.470.2 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 passed out the `exit` Sequence port.

Input Ports:

- Sequence: `for (elem) in (input)`
- Data (variant): `input`

Output Ports:

- Sequence: `each`
- Sequence: `exit`
- Data (variant): `elem`

29.471 VisualScriptLocalVar

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.471.1 Brief Description

Gets a local variable's value.

29.471.2 Member Variables

- *Variant.Type* **type** - The local variable's type.
- *String* **var_name** - The local variable's name.

29.471.3 Description

Returns a local variable's value. "Var Name" must be supplied, with an optional type.

Input Ports:

none

Output Ports:

- Data (variant): `get`

29.472 VisualScriptLocalVarSet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.472.1 Brief Description

Changes a local variable's value.

29.472.2 Member Variables

- *Variant.Type* **type** - The local variable's type.
- *String* **var_name** - The local variable's name.

29.472.3 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`

29.473 VisualScriptMathConstant

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.473.1 Brief Description

Commonly used mathematical constants.

29.473.2 Member Variables

- *MathConstant* **constant** - The math constant.

29.473.3 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** — Mathematical constant `e`, the natural log base: `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**

29.473.4 Description

Provides common math constants, such as Pi, on an output Data port.

Input Ports:

none

Output Ports:

- Data (variant): `get`

29.474 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

29.474.1 Brief Description

A node which is part of a *VisualScript*.

29.474.2 Member Functions

Variant	*get_default_input_value* (*int* port_idx) const
VisualScript	*get_visual_script* ( ) const
void	*ports_changed_notify* ( )
void	*set_default_input_value* (*int* port_idx, *Variant* value)

29.474.3 Signals

- **ports_changed** ( )

Emitted when the available input/output ports are changed.

29.474.4 Description

A node which is part of a *VisualScript*. Not to be confused with *Node*, which is a part of a *SceneTree*.

29.474.5 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.

29.475 VisualScriptOperator

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.475.1 Brief Description

29.475.2 Member Variables

- *Variant.Operator* **operator**
- *Variant.Type* **type**

29.475.3 Description

Input Ports:

- Data (variant): `A`
- Data (variant): `B`

Output Ports:

- Data (variant): `result`

29.476 VisualScriptPreload

Inherits: *VisualScriptNode* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.476.1 Brief Description

Creates a new *Resource* or loads one from the filesystem.

29.476.2 Member Variables

- *Resource* **resource** - The *Resource* to load.

29.476.3 Description

Creates a new *Resource* or loads one from the filesystem.

Input Ports:

none

Output Ports:

- Data (object): `res`

29.477 VisualScriptPropertyGet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.477.1 Brief Description

29.477.2 Member Variables

- *String* **base_script**
- *String* **base_type**
- *Variant.Type* **basic_type**
- *String* **index**
- *NodePath* **node_path**
- *String* **property**
- *CallMode* **set_mode**

29.477.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**
- **CALL_MODE_NODE_PATH = 1**
- **CALL_MODE_INSTANCE = 2**

29.478 VisualScriptPropertySet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.478.1 Brief Description

29.478.2 Member Variables

- *AssignOp* **assign_op**
- *String* **base_script**
- *String* **base_type**
- *Variant.Type* **basic_type**

- *String* **index**
- *NodePath* **node_path**
- *String* **property**
- *CallMode* **set_mode**

29.478.3 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**

29.479 VisualScriptResourcePath

Inherits: *VisualScriptNode* **<** *Resource* **<** *Reference* **<** *Object*

Category: Core

29.479.1 Brief Description

29.479.2 Member Variables

- *String* **path**

29.480 VisualScriptReturn

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.480.1 Brief Description

Exits a function and returns an optional value.

29.480.2 Member Variables

- *bool* **return_enabled** - If `true` the `return` input port is available.
- *Variant.Type* **return_type** - The return value's data type.

29.480.3 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

29.481 VisualScriptSceneNode

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.481.1 Brief Description

Node reference.

29.481.2 Member Variables

- *NodePath* **node_path** - The node's path in the scene tree.

29.481.3 Description

A direct reference to a node.

Input Ports:

none

Output Ports:

- Data: `node` (obj)

29.482 VisualScriptSceneTree

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.482.1 Brief Description

29.483 VisualScriptSelect

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.483.1 Brief Description

Chooses between two input values.

29.483.2 Member Variables

- *Variant.Type* **type** - The input variables' type.

29.483.3 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`

29.484 VisualScriptSelf

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.484.1 Brief Description

Outputs a reference to the current instance.

29.484.2 Description

Provides a reference to the node running the visual script.

Input Ports:

none

Output Ports:

- Data (object): `instance`

29.485 VisualScriptSequence

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.485.1 Brief Description

Executes a series of Sequence ports.

29.485.2 Member Variables

- *int* **steps** - The number of steps in the sequence.

29.485.3 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`

29.486 VisualScriptSubCall

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.486.1 Brief Description

29.486.2 Member Functions

Variant	*_subcall* (*Variant* arguments) virtual

29.486.3 Member Function Description

- *Variant* **_subcall** (*Variant* arguments) virtual

29.487 VisualScriptSwitch

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.487.1 Brief Description

Branches program flow based on a given input's value.

29.487.2 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`

29.488 VisualScriptTypeCast

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.488.1 Brief Description

29.488.2 Member Variables

- *String* **base_script**
- *String* **base_type**

29.489 VisualScriptVariableGet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.489.1 Brief Description

Gets a variable's value.

29.489.2 Member Variables

- *String* **var_name** - The variable's name.

29.489.3 Description

Returns a variable's value. "Var Name" must be supplied, with an optional type.

Input Ports:

none

Output Ports:

- Data (variant): `value`

29.490 VisualScriptVariableSet

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.490.1 Brief Description

Changes a variable's value.

29.490.2 Member Variables

- *String* **var_name** - The variable's name.

29.490.3 Description

Changes a variable's value to the given input.

Input Ports:

- Sequence
- Data (variant): `set`

Output Ports:

- Sequence

29.491 VisualScriptWhile

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.491.1 Brief Description

Conditional loop.

29.491.2 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`

29.492 VisualScriptYield

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.492.1 Brief Description

29.492.2 Member Variables

- *YieldMode* **mode**
- *float* **wait_time**

29.492.3 Enums

enum **YieldMode**

- **YIELD_FRAME = 1**
- **YIELD_PHYSICS_FRAME = 2**
- **YIELD_WAIT = 3**

29.493 VisualScriptYieldSignal

Inherits: *VisualScriptNode* < *Resource* < *Reference* < *Object*

Category: Core

29.493.1 Brief Description

29.493.2 Member Variables

- *String* **base_type**
- *CallMode* **call_mode**
- *NodePath* **node_path**
- *String* **signal**

29.493.3 Enums

enum **CallMode**

- **CALL_MODE_SELF = 0**

- **CALL_MODE_NODE_PATH = 1**
- **CALL_MODE_INSTANCE = 2**

29.494 VisualServer

Inherits: *Object*

Category: Core

29.494.1 Brief Description

Server for anything visible.

29.494.2 Member Functions

void	*black_bars_set_images* (*RID* left, *RID* top, *RID* right, *RID* bottom)
void	*black_bars_set_margins* (*int* left, *int* top, *int* right, *int* bottom)
RID	*camera_create* ( )
void	*camera_set_cull_mask* (*RID* camera, *int* layers)
void	*camera_set_environment* (*RID* camera, *RID* env)
void	*camera_set_orthogonal* (*RID* camera, *float* size, *float* z_near, *float* z_far)
void	*camera_set_perspective* (*RID* camera, *float* fovy_degrees, *float* z_near, *float* z_far)
void	*camera_set_transform* (*RID* camera, *Transform* transform)
void	*camera_set_use_vertical_aspect* (*RID* camera, *bool* enable)
RID	*canvas_create* ( )
void	*canvas_item_add_circle* (*RID* item, *Vector2* pos, *float* radius, *Color* color)
void	*canvas_item_add_clip_ignore* (*RID* item, *bool* ignore)
void	*canvas_item_add_line* (*RID* item, *Vector2* from, *Vector2* to, *Color* color, *float* width=1.0, *bool* antialiased=false)
void	*canvas_item_add_mesh* (*RID* item, *RID* mesh, *RID* texture, *RID* normal_map)
void	*canvas_item_add_multimesh* (*RID* item, *RID* mesh, *RID* texture, *RID* normal_map)
void	*canvas_item_add_nine_patch* (*RID* item, *Rect2* rect, *Rect2* source, *RID* texture, *Vector2* topleft, *Vector2* bottomri
void	*canvas_item_add_particles* (*RID* item, *RID* particles, *RID* texture, *RID* normal_map, *int* h_frames, *int* v_frames
void	*canvas_item_add_polygon* (*RID* item, *PoolVector2Array* points, *PoolColorArray* colors, *PoolVector2Array* uvs=I
void	*canvas_item_add_polyline* (*RID* item, *PoolVector2Array* points, *PoolColorArray* colors, *float* width=1.0, *bool* an
void	*canvas_item_add_primitive* (*RID* item, *PoolVector2Array* points, *PoolColorArray* colors, *PoolVector2Array* uvs,
void	*canvas_item_add_rect* (*RID* item, *Rect2* rect, *Color* color)
void	*canvas_item_add_set_transform* (*RID* item, *Transform2D* transform)
void	*canvas_item_add_texture_rect* (*RID* item, *Rect2* rect, *RID* texture, *bool* tile=false, *Color* modulate=Color(1, 1, 1
void	*canvas_item_add_texture_rect_region* (*RID* item, *Rect2* rect, *RID* texture, *Rect2* src_rect, *Color* modulate=Color
void	*canvas_item_add_triangle_array* (*RID* item, *PoolIntArray* indices, *PoolVector2Array* points, *PoolColorArray* co
void	*canvas_item_clear* (*RID* item)
RID	*canvas_item_create* ( )
void	*canvas_item_set_clip* (*RID* item, *bool* clip)
void	*canvas_item_set_copy_to_backbuffer* (*RID* item, *bool* enabled, *Rect2* rect)
void	*canvas_item_set_custom_rect* (*RID* item, *bool* use_custom_rect, *Rect2* rect=Rect2(0, 0, 0, 0))
void	*canvas_item_set_distance_field_mode* (*RID* item, *bool* enabled)
void	*canvas_item_set_draw_behind_parent* (*RID* item, *bool* enabled)
void	*canvas_item_set_draw_index* (*RID* item, *int* index)

T

void	*canvas_item_set_light_mask* (*RID* item, *int* mask)
void	*canvas_item_set_material* (*RID* item, *RID* material)
void	*canvas_item_set_modulate* (*RID* item, *Color* color)
void	*canvas_item_set_parent* (*RID* item, *RID* parent)
void	*canvas_item_set_self_modulate* (*RID* item, *Color* color)
void	*canvas_item_set_sort_children_by_y* (*RID* item, *bool* enabled)
void	*canvas_item_set_transform* (*RID* item, *Transform2D* transform)
void	*canvas_item_set_use_parent_material* (*RID* item, *bool* enabled)
void	*canvas_item_set_visible* (*RID* item, *bool* visible)
void	*canvas_item_set_z_as_relative_to_parent* (*RID* item, *bool* enabled)
void	*canvas_item_set_z_index* (*RID* item, *int* z_index)
void	*canvas_light_attach_to_canvas* (*RID* light, *RID* canvas)
RID	*canvas_light_create* ( )
void	*canvas_light_occluder_attach_to_canvas* (*RID* occluder, *RID* canvas)
RID	*canvas_light_occluder_create* ( )
void	*canvas_light_occluder_set_enabled* (*RID* occluder, *bool* enabled)
void	*canvas_light_occluder_set_light_mask* (*RID* occluder, *int* mask)
void	*canvas_light_occluder_set_polygon* (*RID* occluder, *RID* polygon)
void	*canvas_light_occluder_set_transform* (*RID* occluder, *Transform2D* transform)
void	*canvas_light_set_color* (*RID* light, *Color* color)
void	*canvas_light_set_enabled* (*RID* light, *bool* enabled)
void	*canvas_light_set_energy* (*RID* light, *float* energy)
void	*canvas_light_set_height* (*RID* light, *float* height)
void	*canvas_light_set_item_cull_mask* (*RID* light, *int* mask)
void	*canvas_light_set_item_shadow_cull_mask* (*RID* light, *int* mask)
void	*canvas_light_set_layer_range* (*RID* light, *int* min_layer, *int* max_layer)
void	*canvas_light_set_mode* (*RID* light, *CanvasLightMode* mode)
void	*canvas_light_set_scale* (*RID* light, *float* scale)
void	*canvas_light_set_shadow_buffer_size* (*RID* light, *int* size)
void	*canvas_light_set_shadow_color* (*RID* light, *Color* color)
void	*canvas_light_set_shadow_enabled* (*RID* light, *bool* enabled)
void	*canvas_light_set_shadow_filter* (*RID* light, *CanvasLightShadowFilter* filter)
void	*canvas_light_set_shadow_gradient_length* (*RID* light, *float* length)
void	*canvas_light_set_shadow_smooth* (*RID* light, *float* 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)
void	*canvas_light_set_z_range* (*RID* light, *int* min_z, *int* max_z)
RID	*canvas_occluder_polygon_create* ( )
void	*canvas_occluder_polygon_set_cull_mode* (*RID* occluder_polygon, *CanvasOccluderPolygonCullMode* mode)
void	*canvas_occluder_polygon_set_shape* (*RID* occluder_polygon, *PoolVector2Array* shape, *bool* closed)
void	*canvas_occluder_polygon_set_shape_as_lines* (*RID* occluder_polygon, *PoolVector2Array* shape)
void	*canvas_set_item_mirroring* (*RID* canvas, *RID* item, *Vector2* mirroring)
void	*canvas_set_modulate* (*RID* canvas, *Color* color)
RID	*directional_light_create* ( )
void	*draw* (*bool* swap_buffers=true)
RID	*environment_create* ( )
void	*environment_set_adjustment* (*RID* env, *bool* enable, *float* brightness, *float* contrast, *float* saturation, *RID* ramp)
void	*environment_set_ambient_light* (*RID* env, *Color* color, *float* energy=1.0, *float* sky_contibution=0.0)
void	*environment_set_background* (*RID* env, *EnvironmentBG* bg)

void	*environment_set_bg_color* (*RID* env, *Color* color)
void	*environment_set_bg_energy* (*RID* env, *float* energy)
void	*environment_set_canvas_max_layer* (*RID* env, *int* max_layer)
void	*environment_set_dof_blur_far* (*RID* env, *bool* enable, *float* distance, *float* transition, *float* far_amount, *Environme*
void	*environment_set_dof_blur_near* (*RID* env, *bool* enable, *float* distance, *float* transition, *float* far_amount, *Environ*
void	*environment_set_fog* (*RID* env, *bool* enable, *Color* color, *Color* sun_color, *float* sun_amount)
void	*environment_set_fog_depth* (*RID* env, *bool* enable, *float* depth_begin, *float* depth_curve, *bool* transmit, *float* trans
void	*environment_set_fog_height* (*RID* env, *bool* enable, *float* min_height, *float* max_height, *float* height_curve)
void	*environment_set_glow* (*RID* env, *bool* enable, *int* level_flags, *float* intensity, *float* strength, *float* bloom_threshold
void	*environment_set_sky* (*RID* env, *RID* sky)
void	*environment_set_sky_custom_fov* (*RID* env, *float* scale)
void	*environment_set_ssao* (*RID* env, *bool* enable, *float* radius, *float* intensity, *float* radius2, *float* intensity2, *float* bias,
void	*environment_set_ssr* (*RID* env, *bool* enable, *int* max_steps, *float* fade_in, *float* fade_out, *float* depth_tolerance, *b*
void	*environment_set_tonemap* (*RID* env, *EnvironmentToneMapper* tone_mapper, *float* exposure, *float* white, *bool* aut
void	*finish* ( )
void	*force_draw* (*bool* swap_buffers=true)
void	*force_sync* ( )
void	*free_rid* (*RID* rid)
int	*get_render_info* (*RenderInfo* info)
RID	*get_test_cube* ( )
RID	*get_test_texture* ( )
RID	*get_white_texture* ( )
RID	*gi_probe_create* ( )
float	*gi_probe_get_bias* (*RID* arg0) const
AABB	*gi_probe_get_bounds* (*RID* probe) const
float	*gi_probe_get_cell_size* (*RID* probe) const
PoolIntArray	*gi_probe_get_dynamic_data* (*RID* arg0) const
int	*gi_probe_get_dynamic_range* (*RID* arg0) const
float	*gi_probe_get_energy* (*RID* arg0) const
float	*gi_probe_get_normal_bias* (*RID* arg0) const
float	*gi_probe_get_propagation* (*RID* arg0) const
Transform	*gi_probe_get_to_cell_xform* (*RID* arg0) const
bool	*gi_probe_is_compressed* (*RID* arg0) const
bool	*gi_probe_is_interior* (*RID* arg0) const
void	*gi_probe_set_bias* (*RID* bias, *float* arg1)
void	*gi_probe_set_bounds* (*RID* probe, *AABB* bounds)
void	*gi_probe_set_cell_size* (*RID* probe, *float* range)
void	*gi_probe_set_compress* (*RID* enable, *bool* arg1)
void	*gi_probe_set_dynamic_data* (*RID* data, *PoolIntArray* arg1)
void	*gi_probe_set_dynamic_range* (*RID* range, *int* arg1)
void	*gi_probe_set_energy* (*RID* energy, *float* arg1)
void	*gi_probe_set_interior* (*RID* enable, *bool* arg1)
void	*gi_probe_set_normal_bias* (*RID* bias, *float* arg1)
void	*gi_probe_set_propagation* (*RID* propagation, *float* arg1)
void	*gi_probe_set_to_cell_xform* (*RID* xform, *Transform* arg1)
bool	*has_changed* ( ) const
bool	*has_feature* (*Features* feature) const
bool	*has_os_feature* (*String* feature) const
void	*immediate_begin* (*RID* immediate, *PrimitiveType* primitive, *RID* texture)
void	*immediate_clear* (*RID* immediate)

T

void	*immediate_color* (*RID* immediate, *Color* color)
RID	*immediate_create* ( )
void	*immediate_end* (*RID* immediate)
RID	*immediate_get_material* (*RID* immediate) const
void	*immediate_normal* (*RID* immediate, *Vector3* normal)
void	*immediate_set_material* (*RID* immediate, *RID* material)
void	*immediate_tangent* (*RID* immediate, *Plane* tangent)
void	*immediate_uv* (*RID* immediate, *Vector2* tex_uv)
void	*immediate_uv2* (*RID* immediate, *Vector2* tex_uv)
void	*immediate_vertex* (*RID* immediate, *Vector3* vertex)
void	*immediate_vertex_2d* (*RID* immediate, *Vector2* vertex)
void	*init* ( )
void	*instance_attach_object_instance_id* (*RID* instance, *int* id)
void	*instance_attach_skeleton* (*RID* instance, *RID* skeleton)
RID	*instance_create* ( )
RID	*instance_create2* (*RID* base, *RID* scenario)
void	*instance_geometry_set_as_instance_lod* (*RID* instance, *RID* as_lod_of_instance)
void	*instance_geometry_set_cast_shadows_setting* (*RID* instance, *ShadowCastingSetting* shadow_casting_setting)
void	*instance_geometry_set_draw_range* (*RID* instance, *float* min, *float* max, *float* min_margin, *float* max_margin)
void	*instance_geometry_set_flag* (*RID* instance, *InstanceFlags* flag, *bool* enabled)
void	*instance_geometry_set_material_override* (*RID* instance, *RID* material)
void	*instance_set_base* (*RID* instance, *RID* base)
void	*instance_set_blend_shape_weight* (*RID* instance, *int* shape, *float* weight)
void	*instance_set_custom_aabb* (*RID* instance, *AABB* aabb)
void	*instance_set_exterior* (*RID* instance, *bool* enabled)
void	*instance_set_extra_visibility_margin* (*RID* instance, *float* margin)
void	*instance_set_layer_mask* (*RID* instance, *int* mask)
void	*instance_set_scenario* (*RID* instance, *RID* scenario)
void	*instance_set_surface_material* (*RID* instance, *int* surface, *RID* material)
void	*instance_set_transform* (*RID* instance, *Transform* transform)
void	*instance_set_use_lightmap* (*RID* instance, *RID* lightmap_instance, *RID* lightmap)
void	*instance_set_visible* (*RID* instance, *bool* visible)
Array	*instances_cull_aabb* (*AABB* aabb, *RID* scenario) const
Array	*instances_cull_convex* (*Array* convex, *RID* scenario) const
Array	*instances_cull_ray* (*Vector3* from, *Vector3* to, *RID* scenario) const
void	*light_directional_set_blend_splits* (*RID* light, *bool* enable)
void	*light_directional_set_shadow_depth_range_mode* (*RID* light, *LightDirectionalShadowDepthRangeMode* range_
void	*light_directional_set_shadow_mode* (*RID* light, *LightDirectionalShadowMode* mode)
void	*light_omni_set_shadow_detail* (*RID* light, *LightOmniShadowDetail* detail)
void	*light_omni_set_shadow_mode* (*RID* light, *LightOmniShadowMode* mode)
void	*light_set_color* (*RID* light, *Color* color)
void	*light_set_cull_mask* (*RID* light, *int* mask)
void	*light_set_negative* (*RID* light, *bool* enable)
void	*light_set_param* (*RID* light, *LightParam* param, *float* value)
void	*light_set_projector* (*RID* light, *RID* texture)
void	*light_set_reverse_cull_face_mode* (*RID* light, *bool* enabled)
void	*light_set_shadow* (*RID* light, *bool* enabled)
void	*light_set_shadow_color* (*RID* light, *Color* color)
RID	*lightmap_capture_create* ( )
AABB	*lightmap_capture_get_bounds* (*RID* capture) const

float	*lightmap_capture_get_energy* (*RID* capture) const
PoolByteArray	*lightmap_capture_get_octree* (*RID* capture) const
int	*lightmap_capture_get_octree_cell_subdiv* (*RID* capture) const
Transform	*lightmap_capture_get_octree_cell_transform* (*RID* capture) const
void	*lightmap_capture_set_bounds* (*RID* capture, *AABB* bounds)
void	*lightmap_capture_set_energy* (*RID* capture, *float* energy)
void	*lightmap_capture_set_octree* (*RID* capture, *PoolByteArray* octree)
void	*lightmap_capture_set_octree_cell_subdiv* (*RID* capture, *int* subdiv)
void	*lightmap_capture_set_octree_cell_transform* (*RID* capture, *Transform* xform)
RID	*make_sphere_mesh* (*int* latitudes, *int* longitudes, *float* radius)
RID	*material_create* ( )
Variant	*material_get_param* (*RID* material, *String* parameter) const
RID	*material_get_shader* (*RID* shader_material) const
void	*material_set_line_width* (*RID* material, *float* width)
void	*material_set_next_pass* (*RID* material, *RID* next_material)
void	*material_set_param* (*RID* material, *String* parameter, *Variant* value)
void	*material_set_render_priority* (*RID* material, *int* priority)
void	*material_set_shader* (*RID* shader_material, *RID* shader)
void	*mesh_add_surface_from_arrays* (*RID* mesh, *PrimitiveType* primtive, *Array* arrays, *Array* blend_shapes=[ ], *int* c
void	*mesh_clear* (*RID* mesh)
RID	*mesh_create* ( )
int	*mesh_get_blend_shape_count* (*RID* mesh) const
BlendShapeMode	*mesh_get_blend_shape_mode* (*RID* mesh) const
AABB	*mesh_get_custom_aabb* (*RID* mesh) const
int	*mesh_get_surface_count* (*RID* mesh) const
void	*mesh_remove_surface* (*RID* mesh, *int* index)
void	*mesh_set_blend_shape_count* (*RID* mesh, *int* amount)
void	*mesh_set_blend_shape_mode* (*RID* mesh, *BlendShapeMode* mode)
void	*mesh_set_custom_aabb* (*RID* mesh, *AABB* aabb)
AABB	*mesh_surface_get_aabb* (*RID* mesh, *int* surface) const
PoolByteArray	*mesh_surface_get_array* (*RID* mesh, *int* surface) const
int	*mesh_surface_get_array_index_len* (*RID* mesh, *int* surface) const
int	*mesh_surface_get_array_len* (*RID* mesh, *int* surface) const
Array	*mesh_surface_get_arrays* (*RID* mesh, *int* surface) const
Array	*mesh_surface_get_blend_shape_arrays* (*RID* mesh, *int* surface) const
int	*mesh_surface_get_format* (*RID* mesh, *int* surface) const
PoolByteArray	*mesh_surface_get_index_array* (*RID* mesh, *int* surface) const
RID	*mesh_surface_get_material* (*RID* mesh, *int* surface) const
PrimitiveType	*mesh_surface_get_primitive_type* (*RID* mesh, *int* surface) const
Array	*mesh_surface_get_skeleton_aabb* (*RID* mesh, *int* surface) const
void	*mesh_surface_set_material* (*RID* mesh, *int* surface, *RID* material)
void	*multimesh_allocate* (*RID* multimesh, *int* instances, *MultimeshTransformFormat* transform_format, *MultimeshCo*
AABB	*multimesh_get_aabb* (*RID* multimesh) const
int	*multimesh_get_instance_count* (*RID* multimesh) const
RID	*multimesh_get_mesh* (*RID* multimesh) const
int	*multimesh_get_visible_instances* (*RID* multimesh) const
Color	*multimesh_instance_get_color* (*RID* multimesh, *int* index) const
Transform	*multimesh_instance_get_transform* (*RID* multimesh, *int* index) const
Transform2D	*multimesh_instance_get_transform_2d* (*RID* multimesh, *int* index) const
void	*multimesh_instance_set_color* (*RID* multimesh, *int* index, *Color* color)

void	*multimesh_instance_set_transform* (*RID* multimesh, *int* index, *Transform* transform)
void	*multimesh_instance_set_transform_2d* (*RID* multimesh, *int* index, *Transform2D* transform)
void	*multimesh_set_mesh* (*RID* multimesh, *RID* mesh)
void	*multimesh_set_visible_instances* (*RID* multimesh, *int* visible)
RID	*omni_light_create* ( )
RID	*particles_create* ( )
AABB	*particles_get_current_aabb* (*RID* particles)
bool	*particles_get_emitting* (*RID* particles)
void	*particles_restart* (*RID* particles)
void	*particles_set_amount* (*RID* particles, *int* amount)
void	*particles_set_custom_aabb* (*RID* particles, *AABB* aabb)
void	*particles_set_draw_order* (*RID* particles, *ParticlesDrawOrder* order)
void	*particles_set_draw_pass_mesh* (*RID* particles, *int* pass, *RID* mesh)
void	*particles_set_draw_passes* (*RID* particles, *int* count)
void	*particles_set_emission_transform* (*RID* particles, *Transform* transform)
void	*particles_set_emitting* (*RID* particles, *bool* emitting)
void	*particles_set_explosiveness_ratio* (*RID* particles, *float* ratio)
void	*particles_set_fixed_fps* (*RID* particles, *int* fps)
void	*particles_set_fractional_delta* (*RID* particles, *bool* enable)
void	*particles_set_lifetime* (*RID* particles, *float* lifetime)
void	*particles_set_one_shot* (*RID* particles, *bool* one_shot)
void	*particles_set_pre_process_time* (*RID* particles, *float* time)
void	*particles_set_process_material* (*RID* particles, *RID* material)
void	*particles_set_randomness_ratio* (*RID* particles, *float* ratio)
void	*particles_set_speed_scale* (*RID* particles, *float* scale)
void	*particles_set_use_local_coordinates* (*RID* particles, *bool* enable)
RID	*reflection_probe_create* ( )
void	*reflection_probe_set_as_interior* (*RID* probe, *bool* enable)
void	*reflection_probe_set_cull_mask* (*RID* probe, *int* layers)
void	*reflection_probe_set_enable_box_projection* (*RID* probe, *bool* enable)
void	*reflection_probe_set_enable_shadows* (*RID* probe, *bool* enable)
void	*reflection_probe_set_extents* (*RID* probe, *Vector3* extents)
void	*reflection_probe_set_intensity* (*RID* probe, *float* intensity)
void	*reflection_probe_set_interior_ambient* (*RID* probe, *Color* color)
void	*reflection_probe_set_interior_ambient_energy* (*RID* probe, *float* energy)
void	*reflection_probe_set_interior_ambient_probe_contribution* (*RID* probe, *float* contrib)
void	*reflection_probe_set_max_distance* (*RID* probe, *float* distance)
void	*reflection_probe_set_origin_offset* (*RID* probe, *Vector3* offset)
void	*reflection_probe_set_update_mode* (*RID* probe, *ReflectionProbeUpdateMode* mode)
void	*request_frame_drawn_callback* (*Object* where, *String* method, *Variant* userdata)
RID	*scenario_create* ( )
void	*scenario_set_debug* (*RID* scenario, *ScenarioDebugMode* debug_mode)
void	*scenario_set_environment* (*RID* scenario, *RID* environment)
void	*scenario_set_fallback_environment* (*RID* scenario, *RID* environment)
void	*scenario_set_reflection_atlas_size* (*RID* scenario, *int* p_size, *int* subdiv)
void	*set_boot_image* (*Image* image, *Color* color, *bool* scale)
void	*set_debug_generate_wireframes* (*bool* generate)
void	*set_default_clear_color* (*Color* color)
RID	*shader_create* ( )
String	*shader_get_code* (*RID* shader) const

T

RID	*shader_get_default_texture_param* (*RID* shader, *String* name) const
Array	*shader_get_param_list* (*RID* shader) const
void	*shader_set_code* (*RID* shader, *String* code)
void	*shader_set_default_texture_param* (*RID* shader, *String* name, *RID* texture)
void	*skeleton_allocate* (*RID* skeleton, *int* bones, *bool* is_2d_skeleton=false)
Transform	*skeleton_bone_get_transform* (*RID* skeleton, *int* bone) const
Transform2D	*skeleton_bone_get_transform_2d* (*RID* skeleton, *int* bone) const
void	*skeleton_bone_set_transform* (*RID* skeleton, *int* bone, *Transform* transform)
void	*skeleton_bone_set_transform_2d* (*RID* skeleton, *int* bone, *Transform2D* transform)
RID	*skeleton_create* ( )
int	*skeleton_get_bone_count* (*RID* skeleton) const
RID	*sky_create* ( )
void	*sky_set_texture* (*RID* sky, *RID* cube_map, *int* radiance_size)
RID	*spot_light_create* ( )
void	*sync* ( )
void	*texture_allocate* (*RID* texture, *int* width, *int* height, *Format* format, *int* flags=7)
RID	*texture_create* ( )
RID	*texture_create_from_image* (*Image* image, *int* flags=7)
Array	*texture_debug_usage* ( )
Image	*texture_get_data* (*RID* texture, *CubeMapSide* cube_side=0) const
int	*texture_get_flags* (*RID* texture) const
Format	*texture_get_format* (*RID* texture) const
int	*texture_get_height* (*RID* texture) const
String	*texture_get_path* (*RID* texture) const
int	*texture_get_texid* (*RID* texture) const
int	*texture_get_width* (*RID* texture) const
void	*texture_set_data* (*RID* texture, *Image* image, *CubeMapSide* cube_side=0)
void	*texture_set_data_partial* (*RID* texture, *Image* image, *int* src_x, *int* src_y, *int* src_w, *int* src_h, *int* dst_x, *int* dst_y,
void	*texture_set_flags* (*RID* texture, *int* flags)
void	*texture_set_path* (*RID* texture, *String* path)
void	*texture_set_shrink_all_x2_on_set_data* (*bool* shrink)
void	*texture_set_size_override* (*RID* texture, *int* width, *int* height)
void	*textures_keep_original* (*bool* enable)
void	*viewport_attach_camera* (*RID* viewport, *RID* camera)
void	*viewport_attach_canvas* (*RID* viewport, *RID* canvas)
void	*viewport_attach_to_screen* (*RID* viewport, *Rect2* rect=Rect2(0, 0, 0, 0), *int* screen=0)
RID	*viewport_create* ( )
void	*viewport_detach* (*RID* viewport)
int	*viewport_get_render_info* (*RID* viewport, *ViewportRenderInfo* info)
RID	*viewport_get_texture* (*RID* viewport) const
void	*viewport_remove_canvas* (*RID* viewport, *RID* canvas)
void	*viewport_set_active* (*RID* viewport, *bool* active)
void	*viewport_set_canvas_layer* (*RID* viewport, *RID* canvas, *int* layer)
void	*viewport_set_canvas_transform* (*RID* viewport, *RID* canvas, *Transform2D* offset)
void	*viewport_set_clear_mode* (*RID* viewport, *ViewportClearMode* clear_mode)
void	*viewport_set_debug_draw* (*RID* viewport, *ViewportDebugDraw* draw)
void	*viewport_set_disable_3d* (*RID* viewport, *bool* disabled)
void	*viewport_set_disable_environment* (*RID* viewport, *bool* disabled)
void	*viewport_set_global_canvas_transform* (*RID* viewport, *Transform2D* transform)
void	*viewport_set_hdr* (*RID* viewport, *bool* enabled)

T

void	*viewport_set_hide_canvas* (*RID* viewport, *bool* hidden)
void	*viewport_set_hide_scenario* (*RID* viewport, *bool* hidden)
void	*viewport_set_msaa* (*RID* viewport, *ViewportMSAA* msaa)
void	*viewport_set_parent_viewport* (*RID* viewport, *RID* parent_viewport)
void	*viewport_set_scenario* (*RID* viewport, *RID* scenario)
void	*viewport_set_shadow_atlas_quadrant_subdivision* (*RID* viewport, *int* quadrant, *int* subdivision)
void	*viewport_set_shadow_atlas_size* (*RID* viewport, *int* size)
void	*viewport_set_size* (*RID* viewport, *int* width, *int* height)
void	*viewport_set_transparent_background* (*RID* viewport, *bool* enabled)
void	*viewport_set_update_mode* (*RID* viewport, *ViewportUpdateMode* update_mode)
void	*viewport_set_usage* (*RID* viewport, *ViewportUsage* usage)
void	*viewport_set_use_arvr* (*RID* viewport, *bool* use_arvr)
void	*viewport_set_vflip* (*RID* viewport, *bool* enabled)

29.494.3 Signals

- **frame_drawn_in_thread** ( )

29.494.4 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.

29.494.5 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 an 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 **ShadowCastingSetting**

- **SHADOW_CASTING_SETTING_OFF = 0**
- **SHADOW_CASTING_SETTING_ON = 1**
- **SHADOW_CASTING_SETTING_DOUBLE_SIDED = 2**
- **SHADOW_CASTING_SETTING_SHADOWS_ONLY = 3**

enum **CanvasOccluderPolygonCullMode**

- **CANVAS_OCCLUDER_POLYGON_CULL_DISABLED = 0** — Culling of the canvas occluder is disabled.
- **CANVAS_OCCLUDER_POLYGON_CULL_CLOCKWISE = 1** — Culling of the canvas occluder is clockwise.
- **CANVAS_OCCLUDER_POLYGON_CULL_COUNTER_CLOCKWISE = 2** — Culling of the canvas occluder is counterclockwise.

enum **ParticlesDrawOrder**

- **PARTICLES_DRAW_ORDER_INDEX = 0**

- **PARTICLES_DRAW_ORDER_LIFETIME = 1**
- **PARTICLES_DRAW_ORDER_VIEW_DEPTH = 2**

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 **MultimeshColorFormat**

- **MULTIMESH_COLOR_NONE = 0**
- **MULTIMESH_COLOR_8BIT = 1**
- **MULTIMESH_COLOR_FLOAT = 2**

enum **LightDirectionalShadowMode**

- **LIGHT_DIRECTIONAL_SHADOW_ORTHOGONAL = 0**
- **LIGHT_DIRECTIONAL_SHADOW_PARALLEL_2_SPLITS = 1**
- **LIGHT_DIRECTIONAL_SHADOW_PARALLEL_4_SPLITS = 2**

enum **LightOmniShadowMode**

- **LIGHT_OMNI_SHADOW_DUAL_PARABOLOID = 0**
- **LIGHT_OMNI_SHADOW_CUBE = 1**

enum **EnvironmentSSAOQuality**

- **ENV_SSAO_QUALITY_LOW = 0**
- **ENV_SSAO_QUALITY_MEDIUM = 1**
- **ENV_SSAO_QUALITY_HIGH = 2**

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 **LightDirectionalShadowDepthRangeMode**

- **LIGHT_DIRECTIONAL_SHADOW_DEPTH_RANGE_STABLE = 0**
- **LIGHT_DIRECTIONAL_SHADOW_DEPTH_RANGE_OPTIMIZED = 1**

enum **EnvironmentGlowBlendMode**

- **GLOW_BLEND_MODE_ADDITIVE = 0**
- **GLOW_BLEND_MODE_SCREEN = 1**
- **GLOW_BLEND_MODE_SOFTLIGHT = 2**
- **GLOW_BLEND_MODE_REPLACE = 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 **EnvironmentDOFBlurQuality**

- **ENV_DOF_BLUR_QUALITY_LOW = 0**
- **ENV_DOF_BLUR_QUALITY_MEDIUM = 1**
- **ENV_DOF_BLUR_QUALITY_HIGH = 2**

enum **InstanceFlags**

- **INSTANCE_FLAG_USE_BAKED_LIGHT = 0**
- **INSTANCE_FLAG_MAX = 1**

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 **LightOmniShadowDetail**

- **LIGHT_OMNI_SHADOW_DETAIL_VERTICAL = 0**
- **LIGHT_OMNI_SHADOW_DETAIL_HORIZONTAL = 1**

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 **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 **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 **MultimeshTransformFormat**

- **MULTIMESH_TRANSFORM_2D = 0**
- **MULTIMESH_TRANSFORM_3D = 1**

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 **EnvironmentSSAOBlur**

- **ENV_SSAO_BLUR_DISABLED = 0**
- **ENV_SSAO_BLUR_1x1 = 1**
- **ENV_SSAO_BLUR_2x2 = 2**
- **ENV_SSAO_BLUR_3x3 = 3**

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 **ReflectionProbeUpdateMode**

- **REFLECTION_PROBE_UPDATE_ONCE = 0**
- **REFLECTION_PROBE_UPDATE_ALWAYS = 1**

enum **EnvironmentBG**

- **ENV_BG_CLEAR_COLOR = 0**
- **ENV_BG_COLOR = 1**
- **ENV_BG_SKY = 2**
- **ENV_BG_COLOR_SKY = 3**
- **ENV_BG_CANVAS = 4**
- **ENV_BG_KEEP = 5**
- **ENV_BG_MAX = 6**

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 horrizon 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 cubemap. - **TEXTURE_FLAG_USED_FOR_STREAMING = 4096** — Texture is a video surface. - **TEXTURE_FLAGS_DEFAULT = 7** — Default flags. Generate mipmaps, repeat, and filter are enabled.

enum **EnvironmentToneMapper**

- **ENV_TONE_MAPPER_LINEAR = 0**
- **ENV_TONE_MAPPER_REINHARDT = 1**
- **ENV_TONE_MAPPER_FILMIC = 2**
- **ENV_TONE_MAPPER_ACES = 3**

29.494.6 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.

29.494.7 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* **camera_create** ( )

- void **camera_set_cull_mask** (*RID* camera, *int* layers)
- void **camera_set_environment** (*RID* camera, *RID* env)
- void **camera_set_orthogonal** (*RID* camera, *float* size, *float* z_near, *float* z_far)
- void **camera_set_perspective** (*RID* camera, *float* fovy_degrees, *float* z_near, *float* z_far)
- void **camera_set_transform** (*RID* camera, *Transform* transform)
- void **camera_set_use_vertical_aspect** (*RID* camera, *bool* enable)
- *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* texture, *RID* normal_map)

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* texture, *RID* normal_map)

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, *NinePatchAxisMode* x_axis_mode=0, *NinePatchAxisMode* 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 commands 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( ), *PoolIntArray* bones=PoolIntArray( ), *PoolRealArray* weights=PoolRealArray( ), *RID* texture, *int* count=-1, *RID* normal_map)
- 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, *CanvasLightMode* 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, *CanvasLightShadowFilter* 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, *CanvasOccluderPolygonCullMode* 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.

- *RID* **directional_light_create** ( )
- void **draw** (*bool* swap_buffers=true)

Draws a frame.

- *RID* **environment_create** ( )
- void **environment_set_adjustment** (*RID* env, *bool* enable, *float* brightness, *float* contrast, *float* saturation, *RID* ramp)
- void **environment_set_ambient_light** (*RID* env, *Color* color, *float* energy=1.0, *float* sky_contibution=0.0)
- void **environment_set_background** (*RID* env, *EnvironmentBG* bg)
- void **environment_set_bg_color** (*RID* env, *Color* color)
- void **environment_set_bg_energy** (*RID* env, *float* energy)
- void **environment_set_canvas_max_layer** (*RID* env, *int* max_layer)
- void **environment_set_dof_blur_far** (*RID* env, *bool* enable, *float* distance, *float* transition, *float* far_amount, *EnvironmentDOFBlurQuality* quality)
- void **environment_set_dof_blur_near** (*RID* env, *bool* enable, *float* distance, *float* transition, *float* far_amount, *EnvironmentDOFBlurQuality* quality)
- void **environment_set_fog** (*RID* env, *bool* enable, *Color* color, *Color* sun_color, *float* sun_amount)
- void **environment_set_fog_depth** (*RID* env, *bool* enable, *float* depth_begin, *float* depth_curve, *bool* transmit, *float* transmit_curve)
- void **environment_set_fog_height** (*RID* env, *bool* enable, *float* min_height, *float* max_height, *float* height_curve)
- void **environment_set_glow** (*RID* env, *bool* enable, *int* level_flags, *float* intensity, *float* strength, *float* bloom_threshold, *EnvironmentGlowBlendMode* blend_mode, *float* hdr_bleed_threshold, *float* hdr_bleed_scale, *bool* bicubic_upscale)
- void **environment_set_sky** (*RID* env, *RID* sky)
- void **environment_set_sky_custom_fov** (*RID* env, *float* scale)
- void **environment_set_ssao** (*RID* env, *bool* enable, *float* radius, *float* intensity, *float* radius2, *float* intensity2, *float* bias, *float* light_affect, *Color* color, *EnvironmentSSAOQuality* quality, *EnvironmentSSAOBlur* blur, *float* bilateral_sharpness)
- void **environment_set_ssr** (*RID* env, *bool* enable, *int* max_steps, *float* fade_in, *float* fade_out, *float* depth_tolerance, *bool* roughness)
- void **environment_set_tonemap** (*RID* env, *EnvironmentToneMapper* tone_mapper, *float* exposure, *float* white, *bool* auto_exposure, *float* min_luminance, *float* max_luminance, *float* auto_exp_speed, *float* auto_exp_grey)
- 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** (*RenderInfo* 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.

- *RID* **gi_probe_create** ( )
- *float* **gi_probe_get_bias** (*RID* arg0) const
- *AABB* **gi_probe_get_bounds** (*RID* probe) const
- *float* **gi_probe_get_cell_size** (*RID* probe) const
- *PoolIntArray* **gi_probe_get_dynamic_data** (*RID* arg0) const
- *int* **gi_probe_get_dynamic_range** (*RID* arg0) const
- *float* **gi_probe_get_energy** (*RID* arg0) const
- *float* **gi_probe_get_normal_bias** (*RID* arg0) const
- *float* **gi_probe_get_propagation** (*RID* arg0) const
- *Transform* **gi_probe_get_to_cell_xform** (*RID* arg0) const
- *bool* **gi_probe_is_compressed** (*RID* arg0) const
- *bool* **gi_probe_is_interior** (*RID* arg0) const
- void **gi_probe_set_bias** (*RID* bias, *float* arg1)
- void **gi_probe_set_bounds** (*RID* probe, *AABB* bounds)
- void **gi_probe_set_cell_size** (*RID* probe, *float* range)
- void **gi_probe_set_compress** (*RID* enable, *bool* arg1)
- void **gi_probe_set_dynamic_data** (*RID* data, *PoolIntArray* arg1)
- void **gi_probe_set_dynamic_range** (*RID* range, *int* arg1)
- void **gi_probe_set_energy** (*RID* energy, *float* arg1)
- void **gi_probe_set_interior** (*RID* enable, *bool* arg1)
- void **gi_probe_set_normal_bias** (*RID* bias, *float* arg1)
- void **gi_probe_set_propagation** (*RID* propagation, *float* arg1)
- void **gi_probe_set_to_cell_xform** (*RID* xform, *Transform* arg1)
- *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** (*Features* 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 pvrtc,

- void **immediate_begin** (*RID* immediate, *PrimitiveType* primitive, *RID* texture)
- void **immediate_clear** (*RID* immediate)
- void **immediate_color** (*RID* immediate, *Color* color)
- *RID* **immediate_create** ( )
- void **immediate_end** (*RID* immediate)
- *RID* **immediate_get_material** (*RID* immediate) const
- void **immediate_normal** (*RID* immediate, *Vector3* normal)
- void **immediate_set_material** (*RID* immediate, *RID* material)
- void **immediate_tangent** (*RID* immediate, *Plane* tangent)
- void **immediate_uv** (*RID* immediate, *Vector2* tex_uv)
- void **immediate_uv2** (*RID* immediate, *Vector2* tex_uv)
- void **immediate_vertex** (*RID* immediate, *Vector3* vertex)
- void **immediate_vertex_2d** (*RID* immediate, *Vector2* vertex)
- void **init** ( )

Initializes the visual server.

- void **instance_attach_object_instance_id** (*RID* instance, *int* id)
- void **instance_attach_skeleton** (*RID* instance, *RID* skeleton)
- *RID* **instance_create** ( )
- *RID* **instance_create2** (*RID* base, *RID* scenario)
- void **instance_geometry_set_as_instance_lod** (*RID* instance, *RID* as_lod_of_instance)
- void **instance_geometry_set_cast_shadows_setting** (*RID* instance, *ShadowCastingSetting* shadow_casting_setting)
- void **instance_geometry_set_draw_range** (*RID* instance, *float* min, *float* max, *float* min_margin, *float* max_margin)
- void **instance_geometry_set_flag** (*RID* instance, *InstanceFlags* flag, *bool* enabled)
- void **instance_geometry_set_material_override** (*RID* instance, *RID* material)
- void **instance_set_base** (*RID* instance, *RID* base)
- void **instance_set_blend_shape_weight** (*RID* instance, *int* shape, *float* weight)
- void **instance_set_custom_aabb** (*RID* instance, *AABB* aabb)
- void **instance_set_exterior** (*RID* instance, *bool* enabled)
- void **instance_set_extra_visibility_margin** (*RID* instance, *float* margin)
- void **instance_set_layer_mask** (*RID* instance, *int* mask)
- void **instance_set_scenario** (*RID* instance, *RID* scenario)
- void **instance_set_surface_material** (*RID* instance, *int* surface, *RID* material)

- void **instance_set_transform** (*RID* instance, *Transform* transform)
- void **instance_set_use_lightmap** (*RID* instance, *RID* lightmap_instance, *RID* lightmap)
- void **instance_set_visible** (*RID* instance, *bool* visible)
- *Array* **instances_cull_aabb** (*AABB* aabb, *RID* scenario) const
- *Array* **instances_cull_convex** (*Array* convex, *RID* scenario) const
- *Array* **instances_cull_ray** (*Vector3* from, *Vector3* to, *RID* scenario) const
- void **light_directional_set_blend_splits** (*RID* light, *bool* enable)
- void **light_directional_set_shadow_depth_range_mode** (*RID* light, *LightDirectionalShadowDepthRangeMode* range_mode)
- void **light_directional_set_shadow_mode** (*RID* light, *LightDirectionalShadowMode* mode)
- void **light_omni_set_shadow_detail** (*RID* light, *LightOmniShadowDetail* detail)
- void **light_omni_set_shadow_mode** (*RID* light, *LightOmniShadowMode* mode)
- void **light_set_color** (*RID* light, *Color* color)
- void **light_set_cull_mask** (*RID* light, *int* mask)
- void **light_set_negative** (*RID* light, *bool* enable)
- void **light_set_param** (*RID* light, *LightParam* param, *float* value)
- void **light_set_projector** (*RID* light, *RID* texture)
- void **light_set_reverse_cull_face_mode** (*RID* light, *bool* enabled)
- void **light_set_shadow** (*RID* light, *bool* enabled)
- void **light_set_shadow_color** (*RID* light, *Color* color)
- *RID* **lightmap_capture_create** ( )
- *AABB* **lightmap_capture_get_bounds** (*RID* capture) const
- *float* **lightmap_capture_get_energy** (*RID* capture) const
- *PoolByteArray* **lightmap_capture_get_octree** (*RID* capture) const
- *int* **lightmap_capture_get_octree_cell_subdiv** (*RID* capture) const
- *Transform* **lightmap_capture_get_octree_cell_transform** (*RID* capture) const
- void **lightmap_capture_set_bounds** (*RID* capture, *AABB* bounds)
- void **lightmap_capture_set_energy** (*RID* capture, *float* energy)
- void **lightmap_capture_set_octree** (*RID* capture, *PoolByteArray* octree)
- void **lightmap_capture_set_octree_cell_subdiv** (*RID* capture, *int* subdiv)
- void **lightmap_capture_set_octree_cell_transform** (*RID* capture, *Transform* xform)
- *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, *PrimitiveType* primtive, *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.

- *BlendShapeMode* **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, *BlendShapeMode* 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.

- *PrimitiveType* **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 **multimesh_allocate** (*RID* multimesh, *int* instances, *MultimeshTransformFormat* transform_format, *MultimeshColorFormat* color_format)
- *AABB* **multimesh_get_aabb** (*RID* multimesh) const
- *int* **multimesh_get_instance_count** (*RID* multimesh) const
- *RID* **multimesh_get_mesh** (*RID* multimesh) const
- *int* **multimesh_get_visible_instances** (*RID* multimesh) const
- *Color* **multimesh_instance_get_color** (*RID* multimesh, *int* index) const
- *Transform* **multimesh_instance_get_transform** (*RID* multimesh, *int* index) const
- *Transform2D* **multimesh_instance_get_transform_2d** (*RID* multimesh, *int* index) const
- void **multimesh_instance_set_color** (*RID* multimesh, *int* index, *Color* color)
- void **multimesh_instance_set_transform** (*RID* multimesh, *int* index, *Transform* transform)
- void **multimesh_instance_set_transform_2d** (*RID* multimesh, *int* index, *Transform2D* transform)

- void **multimesh_set_mesh** (*RID* multimesh, *RID* mesh)
- void **multimesh_set_visible_instances** (*RID* multimesh, *int* visible)
- *RID* **omni_light_create** ( )
- *RID* **particles_create** ( )
- *AABB* **particles_get_current_aabb** (*RID* particles)
- *bool* **particles_get_emitting** (*RID* particles)
- void **particles_restart** (*RID* particles)
- void **particles_set_amount** (*RID* particles, *int* amount)
- void **particles_set_custom_aabb** (*RID* particles, *AABB* aabb)
- void **particles_set_draw_order** (*RID* particles, *ParticlesDrawOrder* order)
- void **particles_set_draw_pass_mesh** (*RID* particles, *int* pass, *RID* mesh)
- void **particles_set_draw_passes** (*RID* particles, *int* count)
- void **particles_set_emission_transform** (*RID* particles, *Transform* transform)
- void **particles_set_emitting** (*RID* particles, *bool* emitting)
- void **particles_set_explosiveness_ratio** (*RID* particles, *float* ratio)
- void **particles_set_fixed_fps** (*RID* particles, *int* fps)
- void **particles_set_fractional_delta** (*RID* particles, *bool* enable)
- void **particles_set_lifetime** (*RID* particles, *float* lifetime)
- void **particles_set_one_shot** (*RID* particles, *bool* one_shot)
- void **particles_set_pre_process_time** (*RID* particles, *float* time)
- void **particles_set_process_material** (*RID* particles, *RID* material)
- void **particles_set_randomness_ratio** (*RID* particles, *float* ratio)
- void **particles_set_speed_scale** (*RID* particles, *float* scale)
- void **particles_set_use_local_coordinates** (*RID* particles, *bool* enable)
- *RID* **reflection_probe_create** ( )
- void **reflection_probe_set_as_interior** (*RID* probe, *bool* enable)
- void **reflection_probe_set_cull_mask** (*RID* probe, *int* layers)
- void **reflection_probe_set_enable_box_projection** (*RID* probe, *bool* enable)
- void **reflection_probe_set_enable_shadows** (*RID* probe, *bool* enable)
- void **reflection_probe_set_extents** (*RID* probe, *Vector3* extents)
- void **reflection_probe_set_intensity** (*RID* probe, *float* intensity)
- void **reflection_probe_set_interior_ambient** (*RID* probe, *Color* color)
- void **reflection_probe_set_interior_ambient_energy** (*RID* probe, *float* energy)
- void **reflection_probe_set_interior_ambient_probe_contribution** (*RID* probe, *float* contrib)
- void **reflection_probe_set_max_distance** (*RID* probe, *float* distance)
- void **reflection_probe_set_origin_offset** (*RID* probe, *Vector3* offset)

- void **reflection_probe_set_update_mode** (*RID* probe, *ReflectionProbeUpdateMode* mode)
- 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'.

- *RID* **scenario_create** ( )
- void **scenario_set_debug** (*RID* scenario, *ScenarioDebugMode* debug_mode)
- void **scenario_set_environment** (*RID* scenario, *RID* environment)
- void **scenario_set_fallback_environment** (*RID* scenario, *RID* environment)
- void **scenario_set_reflection_atlas_size** (*RID* scenario, *int* p_size, *int* subdiv)
- 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.

- void **skeleton_allocate** (*RID* skeleton, *int* bones, *bool* is_2d_skeleton=false)
- *Transform* **skeleton_bone_get_transform** (*RID* skeleton, *int* bone) const
- *Transform2D* **skeleton_bone_get_transform_2d** (*RID* skeleton, *int* bone) const
- void **skeleton_bone_set_transform** (*RID* skeleton, *int* bone, *Transform* transform)
- void **skeleton_bone_set_transform_2d** (*RID* skeleton, *int* bone, *Transform2D* transform)
- *RID* **skeleton_create** ( )
- *int* **skeleton_get_bone_count** (*RID* skeleton) const
- *RID* **sky_create** ( )

Creates an empty sky.

- void **sky_set_texture** (*RID* sky, *RID* cube_map, *int* radiance_size)

Sets a sky's texture.

- *RID* **spot_light_create ( )**
- void **sync ( )**
- void **texture_allocate (*RID* texture, *int* width, *int* height, *Format* 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, *CubeMapSide* 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.

- *Format* **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, *CubeMapSide* 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_data_partial (*RID* texture, *Image* image, *int* src_x, *int* src_y, *int* src_w, *int* src_h, *int* dst_x, *int* dst_y, *int* dst_mip, *CubeMapSide* cube_side=0)**
- void **texture_set_flags (*RID* texture, *int* flags)**

Sets the texture's flags. See 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, *ViewportRenderInfo* 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, *ViewportClearMode* clear_mode)

Sets the clear mode of a viewport. See VIEWPORT_CLEAR_MODE_* constants for options.

- void **viewport_set_debug_draw** (*RID* viewport, *ViewportDebugDraw* 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, *ViewportMSAA* msaa)

Sets the anti-aliasing mode. see 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 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, *ViewportUpdateMode* update_mode)

Sets when the viewport should be updated. See VIEWPORT_UPDATE_MODE_* constants for options.

- void **viewport_set_usage** (*RID* viewport, *ViewportUsage* 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.

29.495 VScrollBar

Inherits: *ScrollBar* < *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.495.1 Brief Description

Vertical version of *ScrollBar*, which goes from left (min) to right (max).

29.496 VSeparator

Inherits: *Separator* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.496.1 Brief Description

Vertical version of *Separator*.

29.496.2 Description

Vertical version of *Separator*. It is used to separate objects horizontally, though (but it looks vertical!).

29.497 VSlider

Inherits: *Slider* < *Range* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.497.1 Brief Description

Vertical slider.

29.497.2 Description

Vertical slider. See *Slider*. This one goes from left (min) to right (max).

29.498 VSplitContainer

Inherits: *SplitContainer* < *Container* < *Control* < *CanvasItem* < *Node* < *Object*

Category: Core

29.498.1 Brief Description

Vertical split container.

29.498.2 Description

Vertical split container. See *SplitContainer*. This goes from left to right.

29.499 WeakRef

Inherits: *Reference* < *Object*

Category: Core

29.499.1 Brief Description

Holds an *Object*, but does not contribute to the reference count if the object is a reference.

29.499.2 Member Functions

Variant	*get_ref* ( ) const

29.499.3 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.

29.499.4 Member Function Description

- *Variant* **get_ref** ( ) const

Returns the *Object* this weakref is referring to.

29.500 WebSocketClient

Inherits: *WebSocketMultiplayerPeer* < *NetworkedMultiplayerPeer* < *PacketPeer* < *Reference* < *Object*

Category: Core

29.500.1 Brief Description

A WebSocket client implementation

29.500.2 Member Functions

Error	*connect_to_url* (*String* url, *PoolStringArray* protocols=PoolStringArray( ), *bool* gd_mp_api=false)
void	*disconnect_from_host* ( )

29.500.3 Signals

- **connection_closed ( )**

Emitted when the connection to the server is closed.

- **connection_error ( )**

Emitted when the connection to the server fails.

- **connection_established (** *String* protocol **)**

Emitted when a connection with the server is established, `protocol` will contain the sub-protocol agreed with the server.

- **data_received ( )**

Emitted when a WebSocket message is received. Note: This signal is NOT emitted when used as high level multiplayer peer.

29.500.4 Member Variables

- *bool* **verify_ssl** - Enable or disable SSL certificate verification. Note: You must specify the certificates to be used in the project settings for it to work when exported.

29.500.5 Description

This class implements a WebSocket client compatible with any RFC 6455 complaint WebSocket server.

This client can be optionally used as a network peer for the *MultiplayerAPI*.

After starting the client (*connect_to_url*), you will need to *NetworkedMultiplayerPeer.poll* it at regular intervals (e.g. inside *Node._process*).

You will received appropriate signals when connecting, disconnecting, or when new data is available.

29.500.6 Member Function Description

- *Error* **connect_to_url (** *String* url, *PoolStringArray* protocols=PoolStringArray( ), *bool* gd_mp_api=false **)**

Connect to the given URL requesting one of the given `protocols` as sub-protocol.

If `true` is passed as `gd_mp_api`, the client will behave like a network peer for the *MultiplayerAPI*. Note: connnections to non Godot servers will not work, and *data_received* will not be emitted when this option is true.

- void **disconnect_from_host ( )**

Disconnect from the server if currently connected.

29.501 WebSocketMultiplayerPeer

Inherits: *NetworkedMultiplayerPeer* < *PacketPeer* < *Reference* < *Object*

Inherited By: *WebSocketClient*, *WebSocketServer*

Category: Core

29.501.1 Brief Description

Base class for WebSocket server and client.

29.501.2 Member Functions

WebSocketPeer	*get_peer* (*int* peer_id) const

29.501.3 Signals

- **peer_packet** (*int* peer_source)

Emitted when a packet is received from a peer. Note: this signal is only emitted when the client or server is configured to use Godot multiplayer API.

29.501.4 Description

Base class for WebSocket server and client, allowing them to be used as network peer for the *MultiplayerAPI*.

29.501.5 Member Function Description

- *WebSocketPeer* **get_peer** (*int* peer_id) const

Returns the *WebSocketPeer* associated to the given `peer_id`.

29.502 WebSocketPeer

Inherits: *PacketPeer* < *Reference* < *Object*

Category: Core

29.502.1 Brief Description

A class representing a specific WebSocket connection.

29.502.2 Member Functions

void	*close* ( )
String	*get_connected_host* ( ) const
int	*get_connected_port* ( ) const
WriteMode	*get_write_mode* ( ) const
bool	*is_connected_to_host* ( ) const
void	*set_write_mode* (*WriteMode* mode)
bool	*was_string_packet* ( ) const

29.502.3 Enums

enum **WriteMode**

- **WRITE_MODE_TEXT = 0** — Specify that WebSockets messages should be transferred as text payload (only valid UTF-8 is allowed).
- **WRITE_MODE_BINARY = 1** — Specify that WebSockets messages should be transferred as binary payload (any byte combination is allowed).

29.502.4 Description

This class represent a specific WebSocket connection, you can do lower level operations with it.

You can choose to write to the socket in binary or text mode, and you can recognize the mode used for writing by the other peer.

29.502.5 Member Function Description

- void **close** **(** **)**

Close this WebSocket connection, actively disconnecting the peer.

- *String* **get_connected_host** **(** **)** const

Returns the IP Address of the connected peer. (Not available in HTML5 export)

- *int* **get_connected_port** **(** **)** const

Returns the remote port of the connected peer. (Not available in HTML5 export)

- *WriteMode* **get_write_mode** **(** **)** const

Get the current selected write mode. See WriteMode.

- *bool* **is_connected_to_host** **(** **)** const

Returns `true` if this peer is currently connected.

- void **set_write_mode** **(** *WriteMode* mode **)**

Sets the socket to use the given WriteMode.

- *bool* **was_string_packet** **(** **)** const

Returns `true` if the last received packet was sent as a text payload. See WriteMode

29.503 WebSocketServer

Inherits: *WebSocketMultiplayerPeer* **<** *NetworkedMultiplayerPeer* **<** *PacketPeer* **<** *Reference* **<** *Object*

Category: Core

29.503.1 Brief Description

A WebSocket server implementation

29.503.2 Member Functions

void	*disconnect_peer* (*int* id)
String	*get_peer_address* (*int* id) const
int	*get_peer_port* (*int* id) const
bool	*has_peer* (*int* id) const
bool	*is_listening* ( ) const
Error	*listen* (*int* port, *PoolStringArray* protocols=PoolStringArray( ), *bool* gd_mp_api=false)
void	*stop* ( )

29.503.3 Signals

- **client_connected** (*int* id, *String* protocol)

Emitted when a new client connects. "protocol" will be the sub-protocol agreed with the client.

- **client_disconnected** (*int* id)

Emitted when a client disconnects.

- **data_received** (*int* id)

Emitted when a new message is received. Note: This signal is NOT emitted when used as high level multiplayer peer.

29.503.4 Description

This class implements a WebSocket server that can also support the high level multiplayer API.

After starting the server (*listen*), you will need to *NetworkedMultiplayerPeer.poll* it at regular intervals (e.g. inside *Node._process*). When clients connect, disconnect, or send data, you will receive the appropriate signal.

Note: This class will not work in HTML5 exports due to browser restrictions.

29.503.5 Member Function Description

- void **disconnect_peer** (*int* id)

Disconnects the given peer.

- *String* **get_peer_address** (*int* id) const

Returns the IP address of the given peer.

- *int* **get_peer_port** (*int* id) const

Returns the remote port of the given peer.

- *bool* **has_peer** (*int* id) const

Returns `true` if a peer with the given ID is connected.

- *bool* **is_listening** ( ) const

Returns `true` if the server is actively listening on a port.

- *Error* **listen** (*int* port, *PoolStringArray* protocols=PoolStringArray( ), *bool* gd_mp_api=false)

Start listening on the given port.

You can specify the desired subprotocols via the "protocols" array. If the list empty (default), "binary" will be used.

You can use this server as a network peer for *MultiplayerAPI* by passing true as "gd_mp_api". Note: *data_received* will not be fired and clients other than Godot will not work in this case.

- void **stop** ( )

Stop the server and clear its state.

29.504 WindowDialog

Inherits: *Popup* < *Control* < *CanvasItem* < *Node* < *Object*

Inherited By: *AcceptDialog*

Category: Core

29.504.1 Brief Description

Base class for window dialogs.

29.504.2 Member Functions

TextureButton	*get_close_button* **( )**

29.504.3 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".

29.504.4 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.

29.504.5 Member Function Description

- *TextureButton* **get_close_button** **( )**

Return the close *TextureButton*.

29.505 World

Inherits: *Resource* < *Reference* < *Object*

Category: Core

29.505.1 Brief Description

Class that has everything pertaining to a world.

29.505.2 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.

29.505.3 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.

29.506 World2D

Inherits: *Resource* **<** *Reference* **<** *Object*

Category: Core

29.506.1 Brief Description

Class that has everything pertaining to a 2D world.

29.506.2 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.

29.506.3 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.

29.507 WorldEnvironment

Inherits: *Node* < *Object*

Category: Core

29.507.1 Brief Description

Default environment properties for the entire scene (post-processing effects, lightning and background settings).

29.507.2 Member Variables

- *Environment* **environment** - The *Environment* resource used by this `WorldEnvironment`, defining the default properties.

29.507.3 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.

29.508 XMLParser

Inherits: *Reference* < *Object*

Category: Core

29.508.1 Brief Description

Low-level class for creating parsers for XML files.

29.508.2 Member Functions

int	*get_attribute_count* ( ) const
String	*get_attribute_name* (*int* idx) const
String	*get_attribute_value* (*int* idx) const
int	*get_current_line* ( ) const
String	*get_named_attribute_value* (*String* name) const
String	*get_named_attribute_value_safe* (*String* name) const
String	*get_node_data* ( ) const
String	*get_node_name* ( ) const
int	*get_node_offset* ( ) const
NodeType	*get_node_type* ( )
bool	*has_attribute* (*String* name) const
bool	*is_empty* ( ) const
Error	*open* (*String* file)
Error	*open_buffer* (*PoolByteArray* buffer)
Error	*read* ( )
Error	*seek* (*int* position)
void	*skip_section* ( )

29.508.3 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

29.508.4 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.

29.508.5 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.

- *NodeType* **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** **(** **)** const

Check whether the current element is empty (this only works for completely empty tags, e.g. <element \>).

- *Error* **open** **(** *String* file **)**

Open a XML file for parsing. This returns an error code.

- *Error* **open_buffer** **(** *PoolByteArray* buffer **)**

Open a XML raw buffer for parsing. This returns an error code.

- *Error* **read** **(** **)**

Read the next node of the file. This returns an error code.

- *Error* **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.

29.509 YSort

Inherits: *Node2D* < *CanvasItem* < *Node* < *Object*

Category: Core

29.509.1 Brief Description

Sort all child nodes based on their Y positions.

29.509.2 Member Variables

- *bool* **sort_enabled**

29.509.3 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.