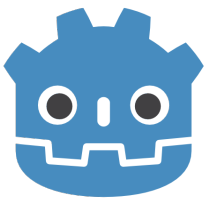
inkgd

Release 0.5.0

Frédéric Maquin

Aug 28, 2023

GETTING STARTED



inkgd

Welcome to the official documentation of *inkgd*, an implementation of Ink's runtime for Godot, in pure GDScript.

The table of contents below, as well as the sidebar, should let you easily find the topic you're looking for. If it's your first time using *inkgd*, we recommend you start with *Introduction* first.

If you prefer a more hands-on approach, feel free to tinker with the example project (*inkgd*'s repository is the example project itself).

While looking for an implementation of **ink** in Godot, you may have come across godot-ink. *inkgd* and *godot-ink* have different philosophies and purposes. If you are not certain which one you should use, *Choosing between inkgd and godot-ink* offers a breakdown of their differences.

CHAPTER

ONE

INTRODUCTION

inkgd is an implementation of the Ink runtime, in pure GDScript.

If you are not familiar with Ink, it's a good idea to take a quick look at two documents from the original documentation:

- Writing with Ink – which describes how to write **ink** stories;
- Running your Ink – which describes how to integrate the **ink** runtime in a C# environment.

The GDScript API is 100% compatible with the original C# API, except for a couple of minor divergences accomodating the differences between the two languages. For more information, see *Differences between the GDScript and C# APIs*.

Note: *inkgd* provides no GUI, only text. It's up to you to decide how to present the content to your players.

Playing a story typically involves repeating the following three steps until the story reaches its end:

1. calling `InkPlayer.continue_story()` until a branch is found;
2. presenting an array of choices to the player;
3. reporting back the selected choice through `InkPlayer.choose_choice_index()`.

Click on the *Next* button to discover how to install *inkgd*!

Note: Nicholas O'Brien created step-by-step video tutorial showing how to create a visual-novel-like game using Godot and Ink. Feel free to check it out!

CHAPTER

TWO

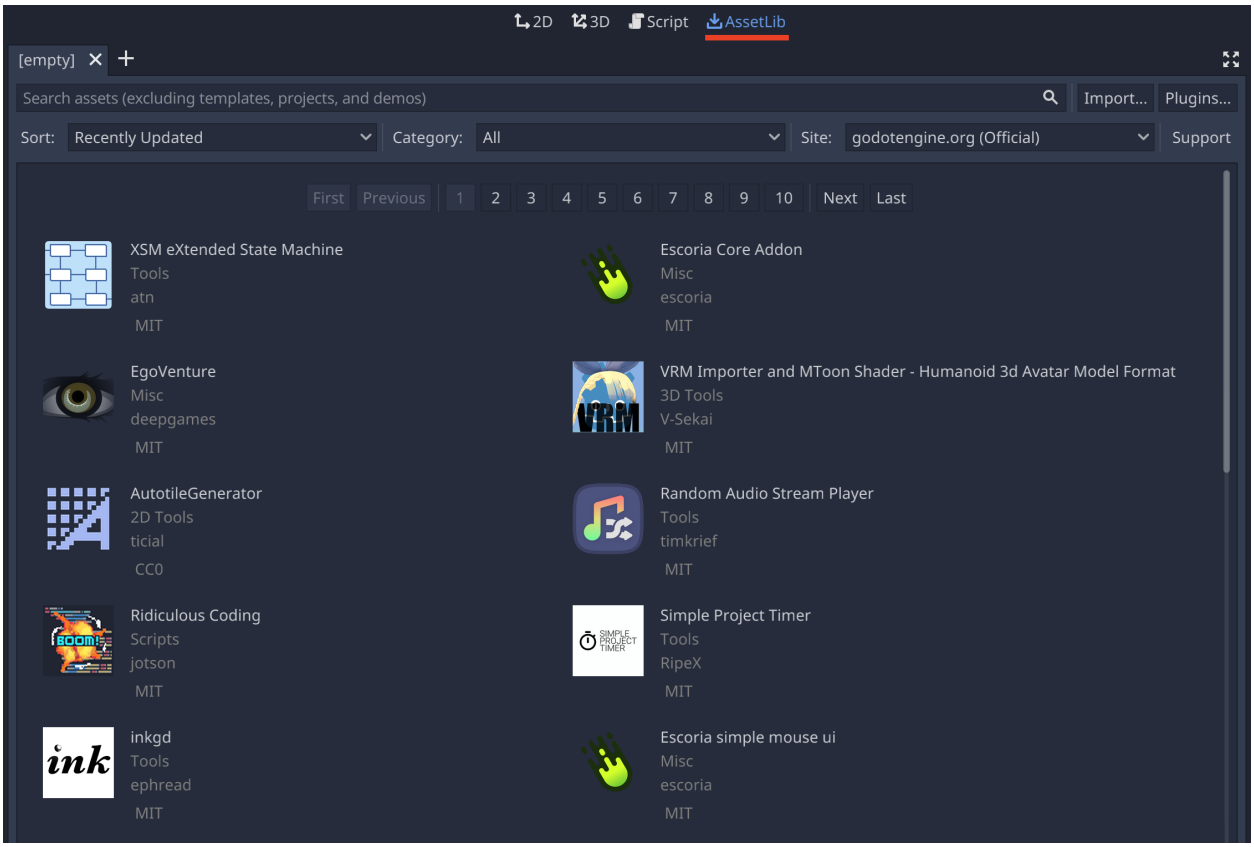
INSTALLATION

You can either install *inkgd* manually (more difficult) or through the Asset Library (simpler).

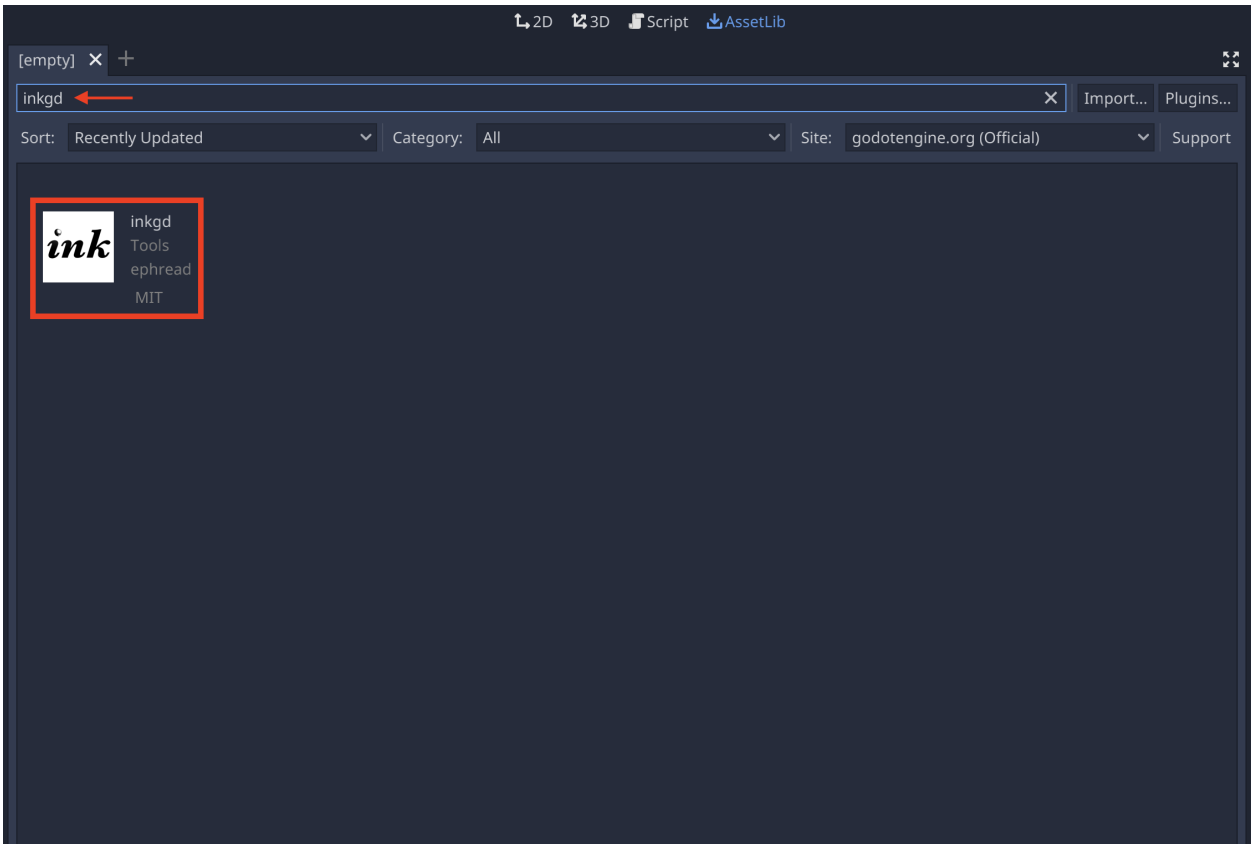
2.1 Asset Library Installation

Note: Due to review processes, there is often a delay between the moment a new version of the documentation is published and the new version of *inkgd* is available in the Asset Library. Always double-check whether the version in the library matches the version of the documentation you are reading.

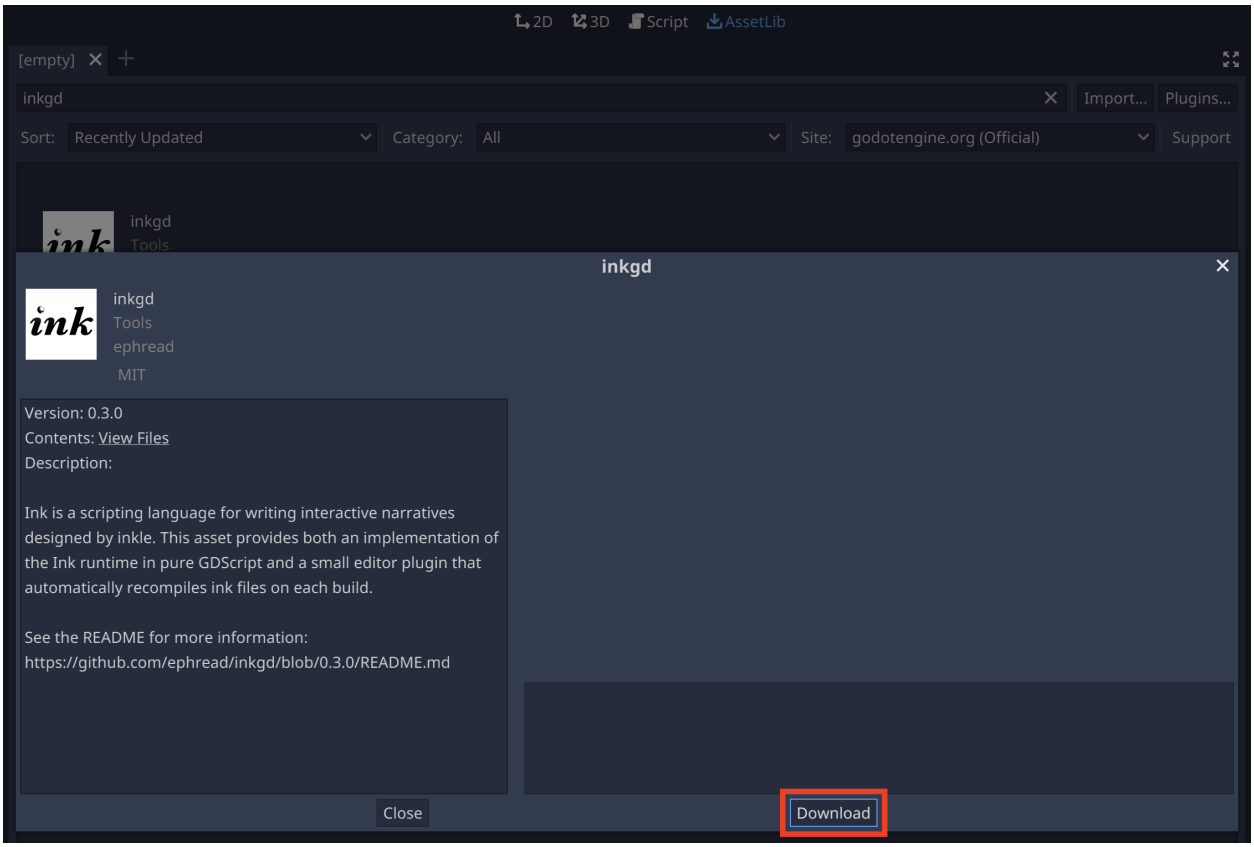
Open a Godot project, click on the to *AssetLib* tab, at the top of the screen, then search for *inkgd*.



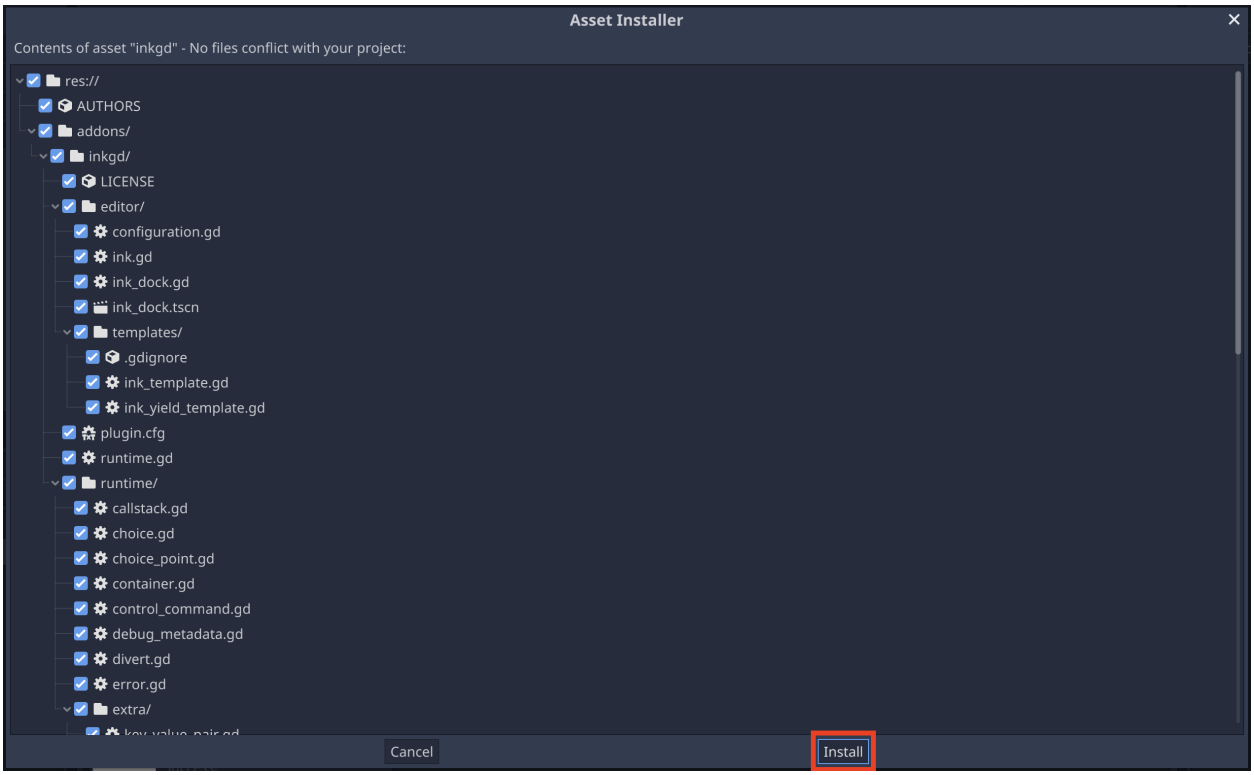
Select *inkgd*.



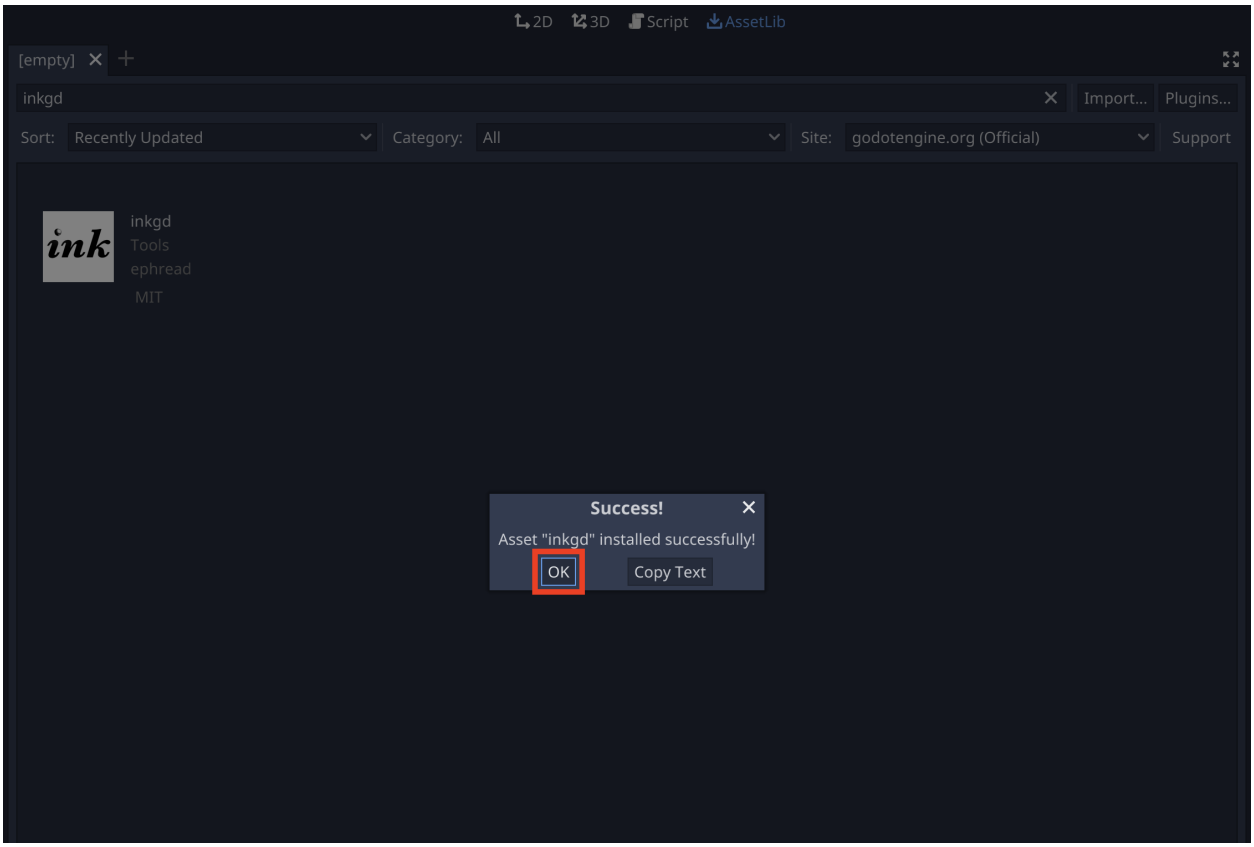
In the popup window, click on *Download*.



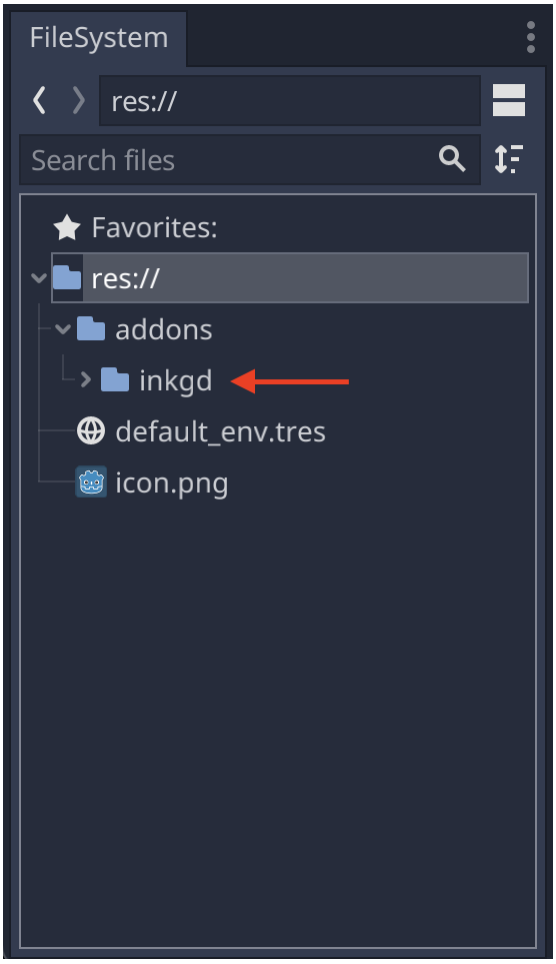
Once the plugin is downloaded, another window will pop up. This window displays the new file expected to be added to the project. Select all files except *examples/* then click on *Install*.



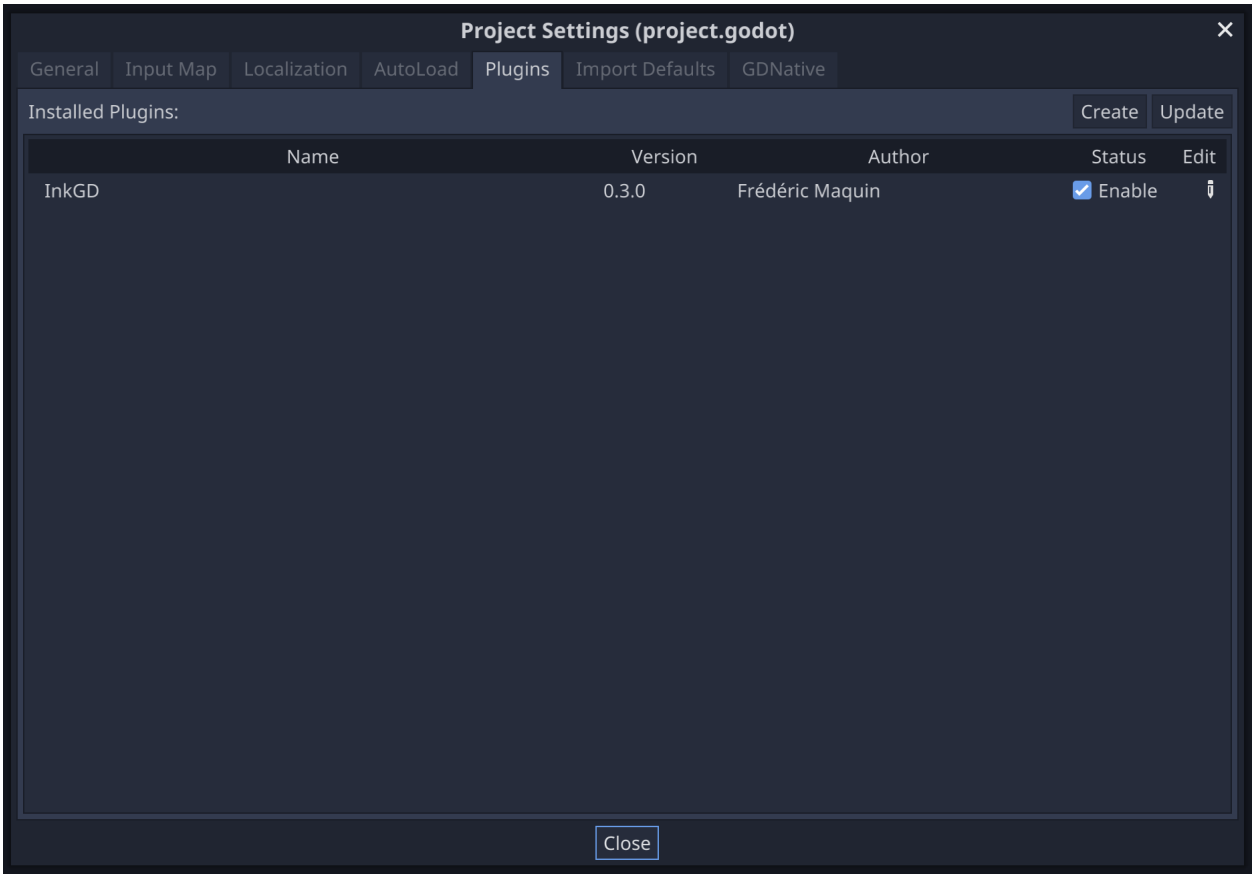
After the installation is completed, a confirmation dialog should pop up. Click on *OK* to close it.



The new files should appear in the FileSystem dock, under the *addons* folder.



inkgd also provides with an editor plugin to manage **ink** stories. The plugin should be enabled by default, but it can be disabled from the project settings. (*Project* > *Project Settings* > *Plugins*).



The editor plugin is not required to use the runtime. **ink** stories can be compiled through `inklecate` directly or other editors, such as Inky. The resulting `.ink.json` file can be loaded in the project manually.

> **Warning:** If you do not enable the plugin, you will not be able to import JSON files as **ink** resources. Make sure to include a filter rule in the export settings to prevent JSON files from being discarded during export.

2.2 Manual Installation

Use Git to clone this repository:

```
$ git clone https://github.com/ephread/inkgd.git
```

Or download the latest stable version of *inkgd*, then extract the content of the archive.

Once you have the content of the repository on your computer, copy the folder `addons/inkgd` to `res://addons/` in your project.

Well done, *inkgd* is installed. Next stop, basic configuration!

CHAPTER

THREE

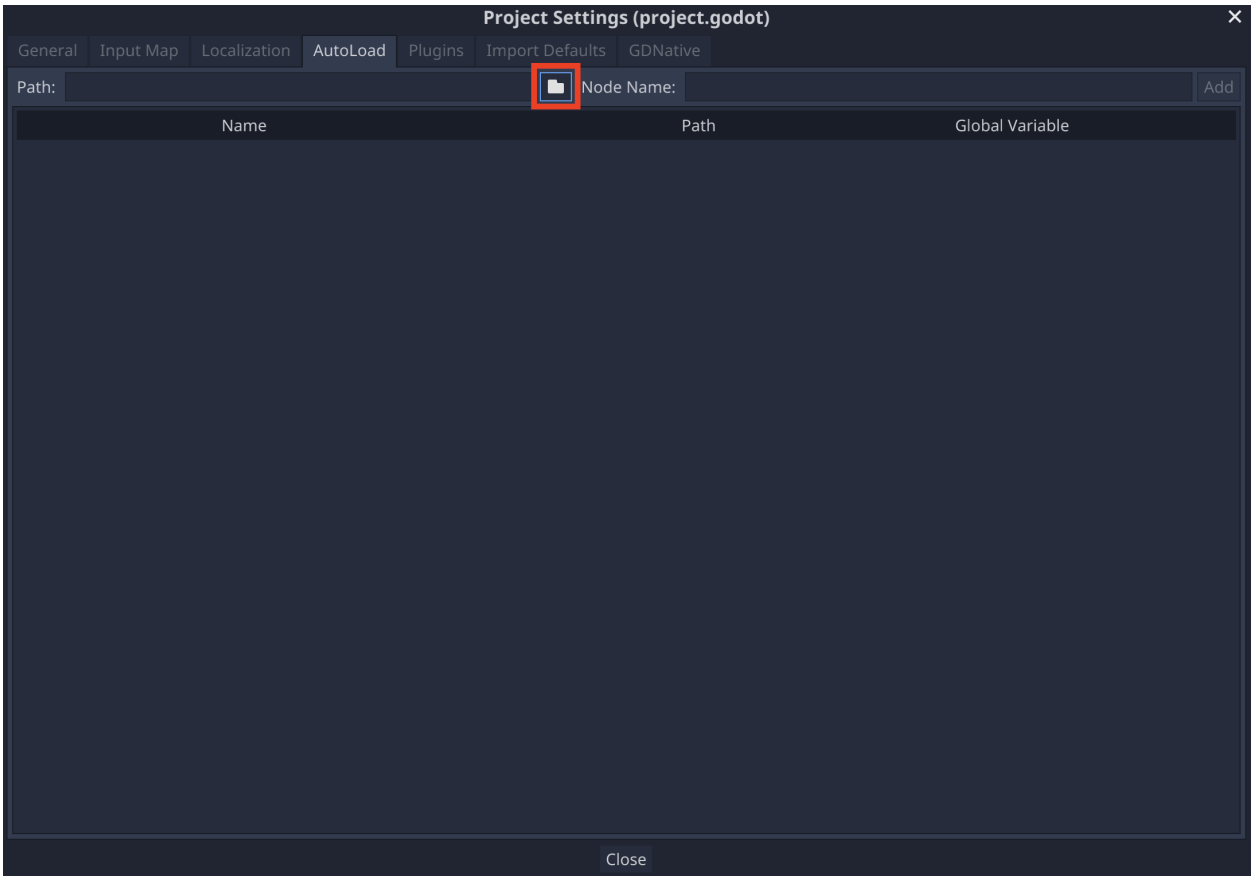
INITIAL CONFIGURATION

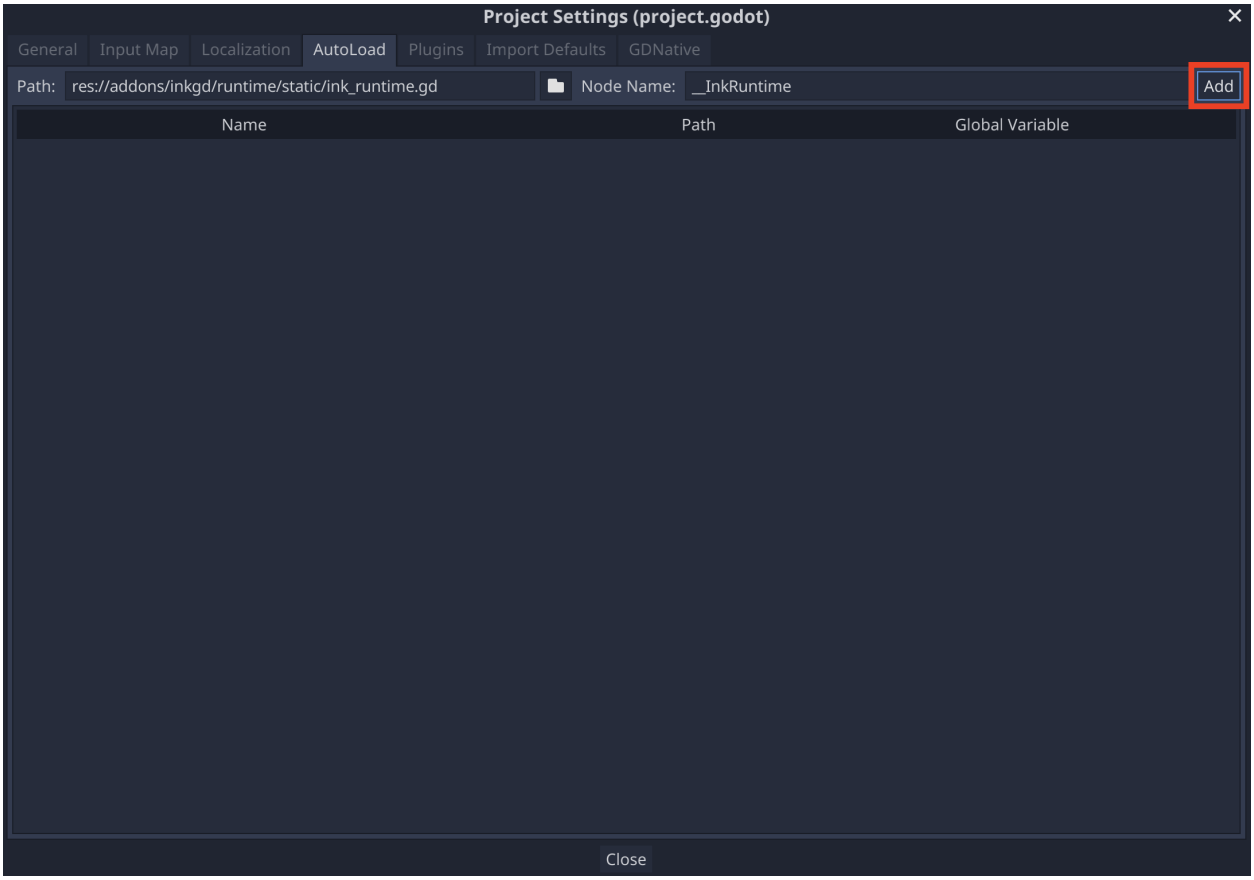
One of the divergences in API mentioned above relates to *static members*. The original C# implementation makes heavy use of static variables, but since GDScript doesn't support them, it uses a singleton node called *InkRuntime* instead. This runtime node must be added to the scene tree before executing any of the methods of the GDScript API.

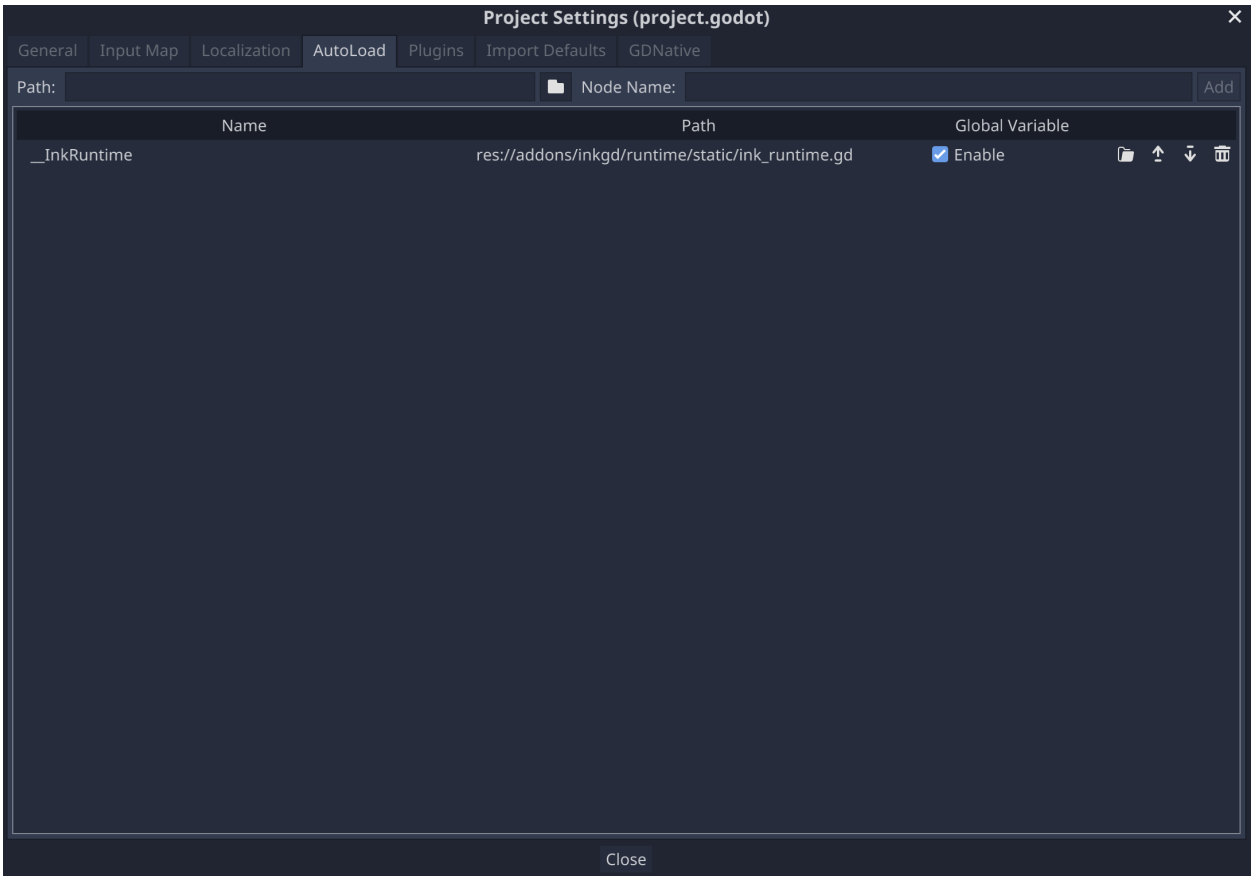
The singleton node is autoconfigured an AutoLoad singleton as long as the editor plugin is enabled.

It's also possible to add `res://addons/inkgd/runtime/static/ink_runtime.gd` to the AutoLoad list manually if it doesn't appear in the list or was previously removed.

Warning: The singleton must be named "__InkRuntime".




Project Settings (project.godot)
General
Input Map
Localization
AutoLoad
Plugins
Import Defaults
GDNative
Path:
res://addons/inkgd/runtime/static/ink_runtime.gd
Node Name:
__InkRuntime
Add
Name
Path
Global Variable
Close



When added as an AutoLoad singleton, the node will remain in the scene tree even when the current scene changes.

That's it! You can now start using *inkgd* in your Godot scripts. From here you can:

- learn how to add *InkPlayer* to your projets;
- learn how to use all the features offered by the *editor plugin*.

CHAPTER

FOUR

CHOOSING BETWEEN *INKGD* AND *GODOT-INK*

Important: The decision to pick *inkgd* over *godot-ink* boils down to the programming language you want to use in your game. If your game uses Godot Mono and is primarily written in C#, you should use *godot-ink*. On the other hand, if you have a GDScript codebase, *inkgd* is the best option. For more information, see below.

There are two ports of the **ink** runtime for Godot, *godot-ink* and *inkgd*.

godot-ink wraps the original C# runtime and is geared towards Godot Mono. *inkgd* is written in GDScript and is more at home in Godot vanilla.

4.1 Differences, strengths and weaknesses

godot-ink provides a C# API that can be awkward when used in GDScript. For instance, methods are written in PascalCase. Additionally, some types —such as InkList— can't be easily bridged to GDScript.

inkgd provides a snake-cased API that integrates well with other GDScript scripts but suffers from poor performance. The GDScript implementation is about 50 times slower than *godot-ink*. These performance limitations are detailed in *Performance*.

If you're still interested in using *inkgd*, head over to the *Introduction*. Otherwise, godot-ink is an excelent choice!

CHAPTER

FIVE

USING INKPLAYER

InkPlayer is a custom node that greatly simplifies the use of **ink** in Godot.

`story.gd` is a direct port of `Story.cs`, to use it in any engine, a bit of boilerplate code is necessary. InkPlayer takes care of that boilerplate so you can focus on building your game.

While it's possible to instantiate `story.gd` directly, it's highly recommended that you use InkPlayer instead.

In addition to reading this document, feel free to glance at story_player.tscn and story_player.gd, which use InkPlayer to run different stories.

Tip: InkPlayer's API documentation is available *here*.

5.1 Main differences with *story.gd*

1. InkPlayer takes a resource as its input, rather than a string containing the JSON bytecode.
2. It unifies Ink's original handlers and *inkgd* custom signals under the same set of consistent signals.
3. It adds convenient methods to save and load the story state.
4. It simplifies certain APIs, such as *evaluate_function* or *remove_variable_observer*.

5.2 Loop-based vs. signal-based flow

InkPlayer can be used in two different ways.

Warning: The examples below are not complete. For a working example, refer to story_player.gd.

5.2.1 Loop-based

This is the traditional way to use Ink.

```
onready var _ink_player = $InkPlayer


func _ready():
    _ink_player.connect("loaded", self, "_story_loaded")
    _ink_player.create_story()


func _story_loaded(successfully: bool):
    if !successfully:
        return

    _continue_story()


func _continue_story():
    while _ink_player.can_continue:
        var text = _ink_player.continue_story()

        # This text is a line of text from the ink story.
        # Set the text of a Label to this value to display it in your game.
        print(text)

    if _ink_player.has_choices:
        # 'current_choices' contains a list of the choices, as strings.
        for choice in _ink_player.current_choices:
            print(choice)

        # '_select_choice' is a function that will take the index of
        # your selection and continue the story by calling again
        # `_continue_story()`.
        _select_choice(0)

    else:
        # This code runs when the story reaches it's end.
        print("The End")
```

5.2.2 Signal-based

Using signals makes the code a little bit more idiomatic for Godot. It's also more flexible.

```
onready var _ink_player = $InkPlayer


func _ready():
    _ink_player.connect("loaded", self, "_story_loaded")
    _ink_player.connect("continued", self, "_continued")
    _ink_player.connect("prompt_choices", self, "_prompt_choices")
```

(continues on next page)

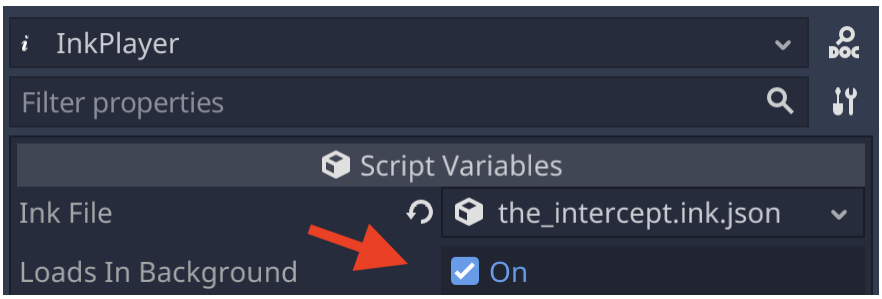
(continued from previous page)

```
    _ink_player.connect("ended", self, "_ended")

    _ink_player.create_story()


func _story_loaded(successfully: bool):
    if !successfully:
        return

    _ink_player.continue_story()


func _continued(text, tags):
    print(text)
    _ink_player.continue_story()


func _prompt_choices(choices):
    if !choices.empty():
        print(choices)

        # In a real-world scenario, _select_choice' could be
        # connected to a signal, like 'Button.pressed'.
        _select_choice(0)


func _ended():
    print("The End")


func _select_choice(index):
    _ink_player.choose_choice_index(index)
    _continue_story()
```

5.3 Loading the story from a background thread

For bigger stories, loading the compiled story into the runtime can take a long time (more than a second). To avoid blocking the main thread, you may want to load the story from a background thread and display a loading indicator.

Fortunately, `InkPlayer` supports loading the story in a thread out of the box. Either tick *Loads In Background* in the inspector or set *loads_in_background* to `true` in code.



On platforms that don't support threads, the feature is automatically disabled regardless of the value of *loads_in_background*.

CHAPTER

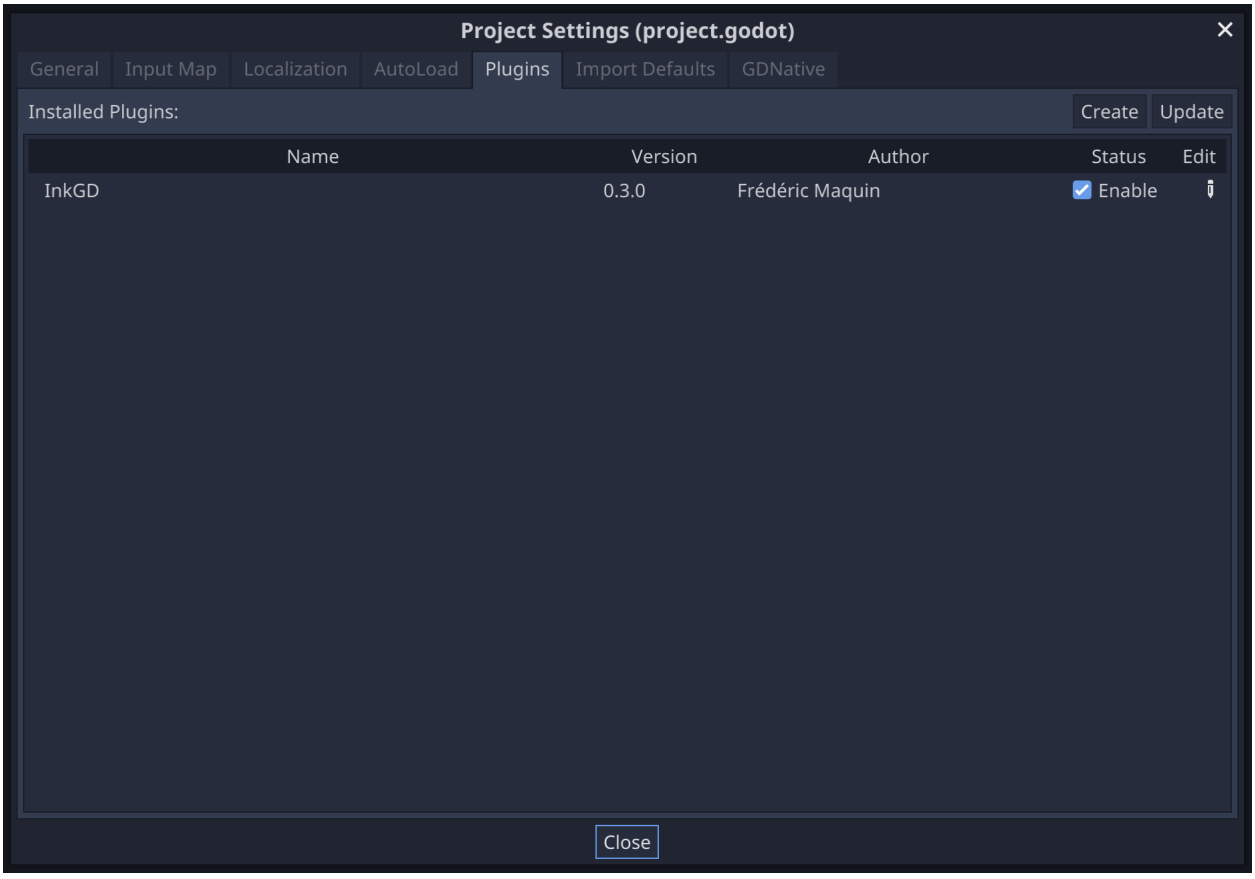
SIX

EDITOR PLUGIN

inkgd ships with an editor plugin providing two major features:

1. a fully-featured bottom panel which can manage and preview stories for you;
2. two import plugins to treat **ink** files as resources.

If the plugin is not already available, navigate to *Project* > *Project Settings* and, in the *Plugins* tab, enable *InkGD*.



Note: To take full advantage of the panel, it's recommended that you download the latest compatible version of

inklecate for your system here.

For more information, see one of the documents listed below.

6.1 Import Plugins

The editor plugin bundles two import plugins. One can import **.ink* files while the other can import **.json* files.

6.1.1 Ink Importer

The **.ink* importer converts all **ink** files into dummy resources and is only used to enable the automatic recompilation of managed stories. Internally, the plugin gets notified any time an **ink** file has been reimported and can trigger a recompilation when appropriate. For more information, refer to the section about *automatic recompilation*.

Note: If you store **ink** files in your project, it's recommended that you exclude them from exports, as they serve no purpose in the final game.

6.1.2 JSON Importer

The **.json* importer converts compiled stories into instances of *InkResource* that can be passed to *InkPlayer*.

After loading an *InkResource*, you can retrieve its JSON content through the `json` property.

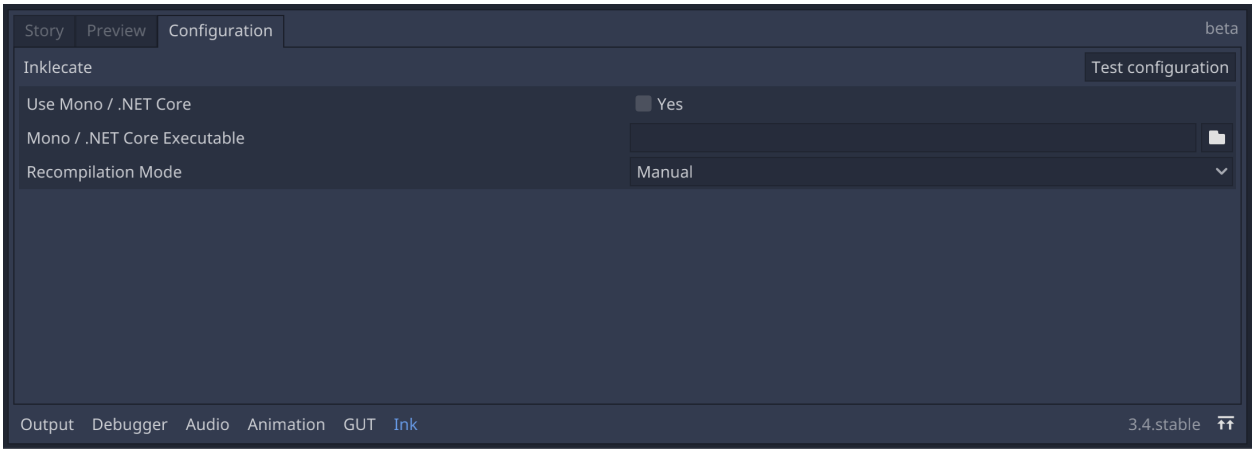
```
var bytecode = load("res://examples/ink/the_intercept.ink.json")

print(bytecode.json)
```

Since JSON files are very common and might be imported by multiple plugins, make sure to use the appropriate importer when reimporting them.

6.2 Ink Panel

6.2.1 Configuration tab

In the Ink panel, select the *Configuration* tab to change inklecate's configuration settings.

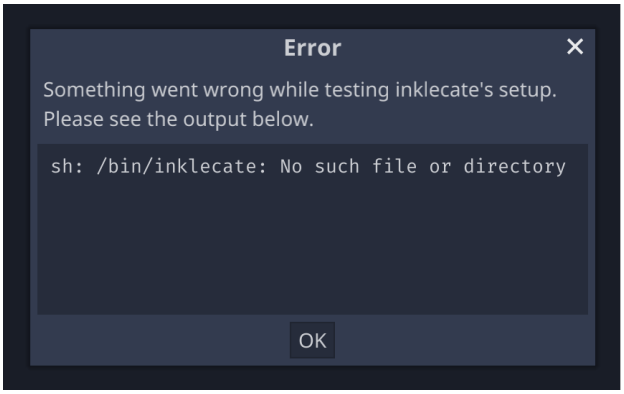


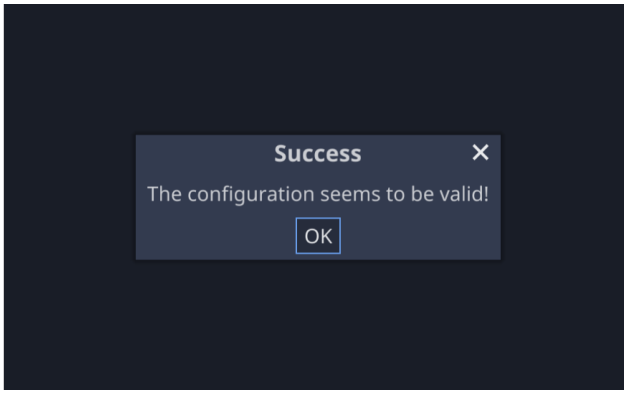
Note: The configuration settings defined in this tab are saved in `.inkgd_compiler.cfg`. If you work in a team, it's recommended to keep this file out of version control, as the environment may differ between team members.

Basic Configuration

To let the plugin know where inklecate is located, click on the folder icon opposite of the *Executable* field and navigate to the desired binary. You can also paste the path directly in the field.

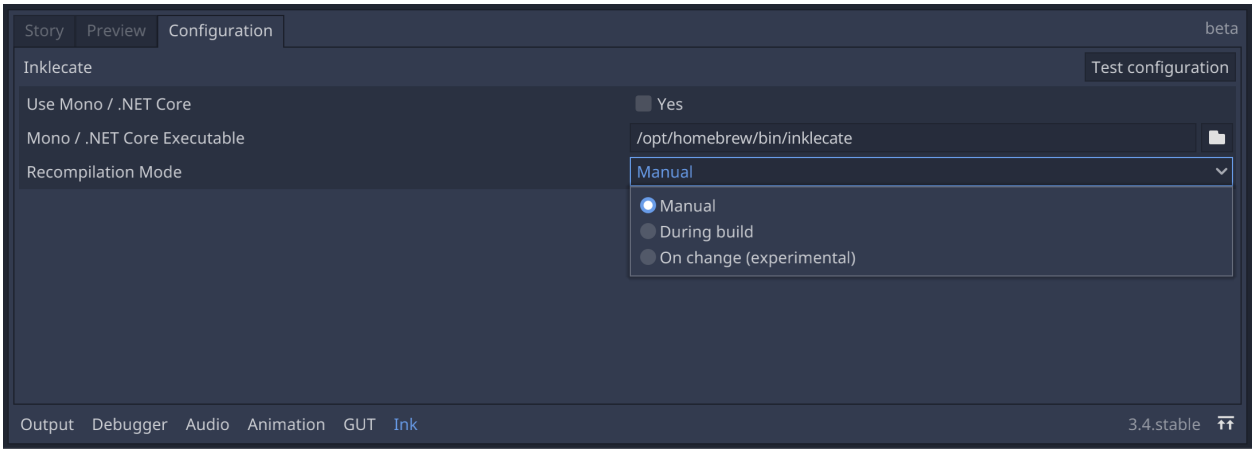
Click on *Test configuration*. If the selected executable is valid, a dialog will appear and confirm that inklecate was successfully executed by the plugin. Otherwise, an error dialog will appear, containing the output of the command that inklecate tried to execute.





Recompilation Modes

The editor plugin supports three recompilation mode.



Manual the stories are never recompiled by the plugin. This is the default mode.

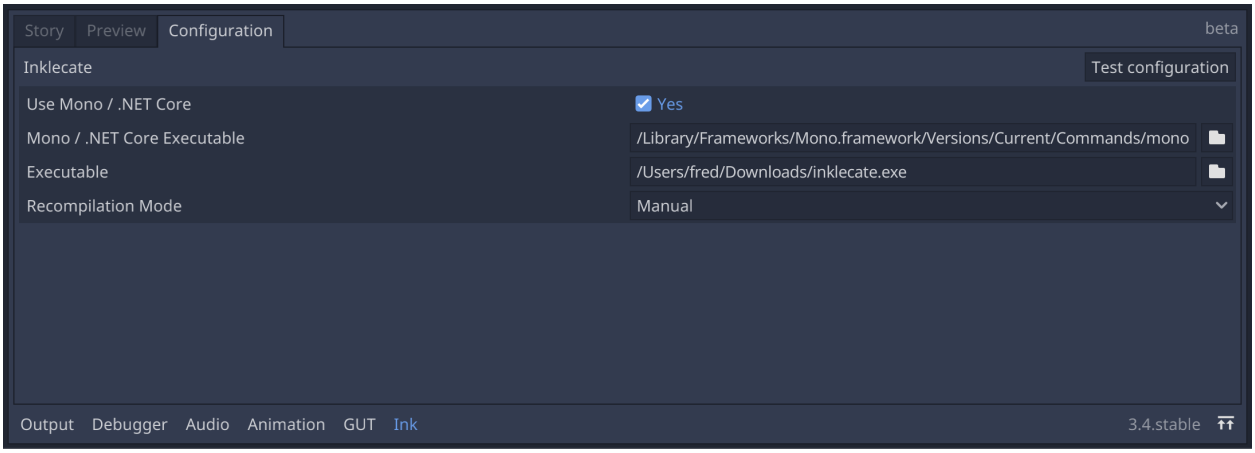
During build the stories are recompiled each time the project is run.

On change an experimental feature, recompiling the stories when changes are detected in **ink** files. For more information, see *below*.

> **Warning:** *On change* is an experimental mode, use with caution.

Custom Mono / .NET Core Runtimes

On platforms other than Windows, advanced users can use specific versions of .NET runtimes with custom inklecates that don't come bundled with a Mono runtime. By ticking the *Use Mono / .NET Core* checkbox, a new configuration field will appear. This field expects a path to the .NET runtime you want to use.

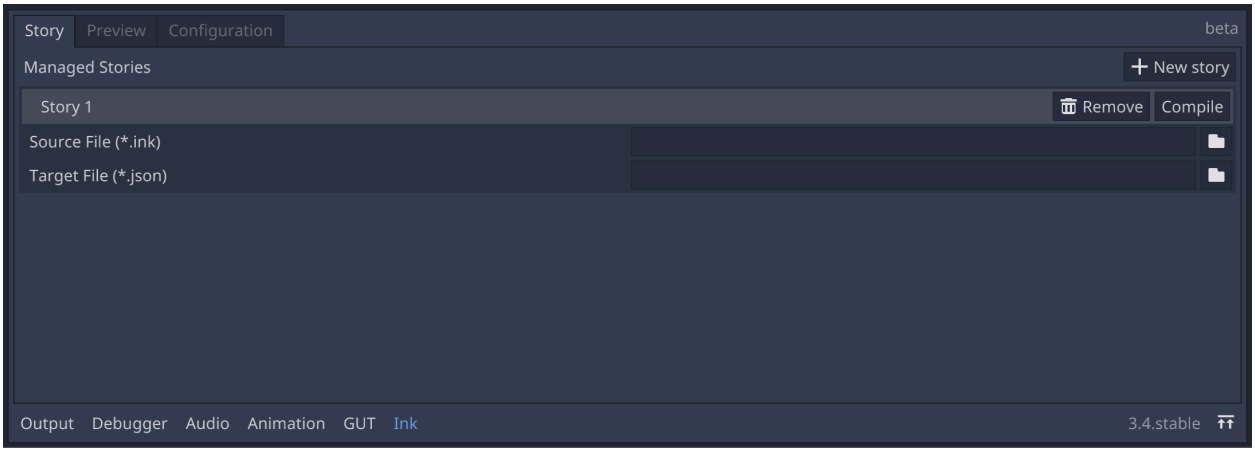


6.2.2 Story tab

In the Ink panel, select the *Story* tab to manage **ink** stories in the project.

Note: The configuration settings defined in this tab are saved in `.inkgd_ink.cfg`. If you work in a team, it's important to commit this file.

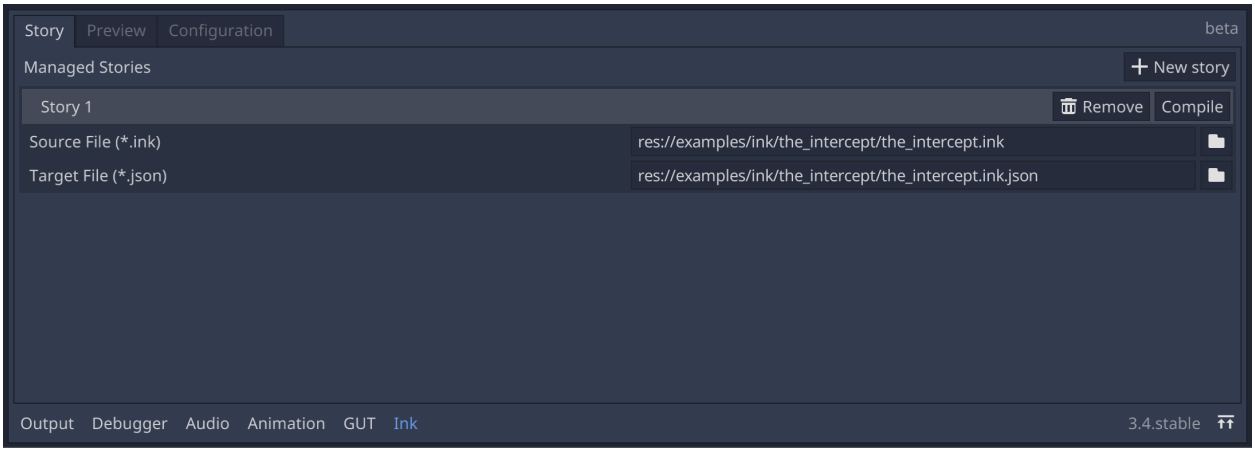
There should be no stories yet, so click on *New Story* to start registering a new story.



To manage a story, the plugin expect two paths. The path to the main **ink** file that should be compiled (*Source File*) and where to write the compiled story (*Target File*). Both files need to be inside the project's file system, thus they can

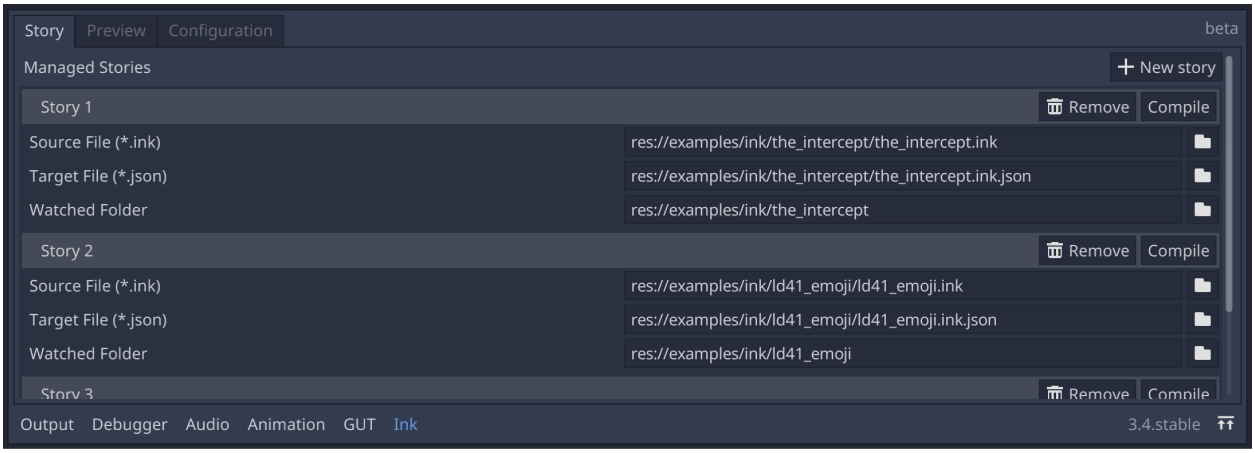
be accessed through the regular `res://` scheme. To set the path of the source file, either click on the folder icon or paste the path in the field directly.

When the source file path is defined, the plugin populates the *Target File* field with a default location.



You can add as many stories as you want and compile them individually clicking on each *Compile* button.

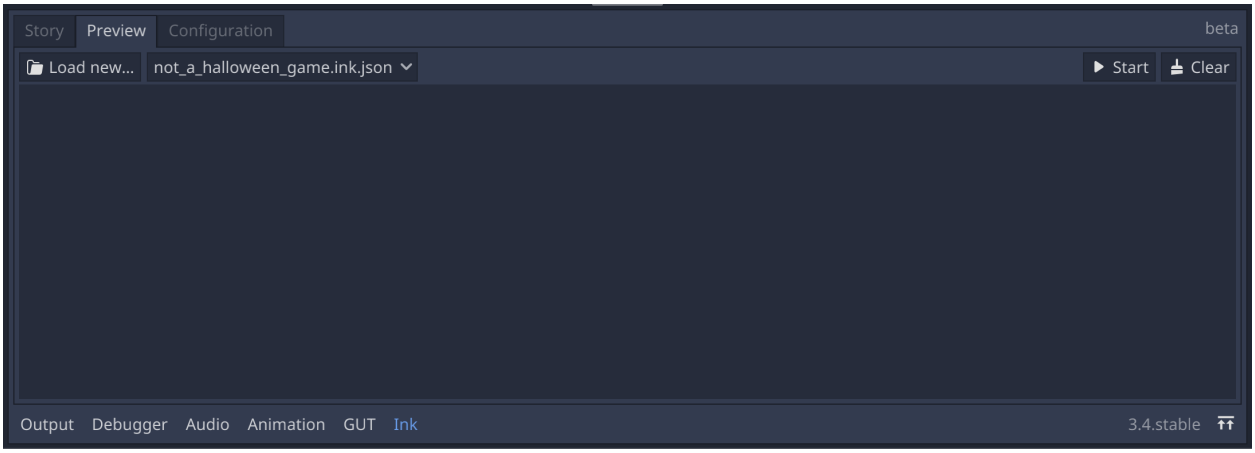
When the selected recompilation mode is *On change*, each entry shows an additional field specifying which folder to watch. Any **.ink* file under that folder that is reimported by Godot (i. e. changed externally) will trigger a recompilation.



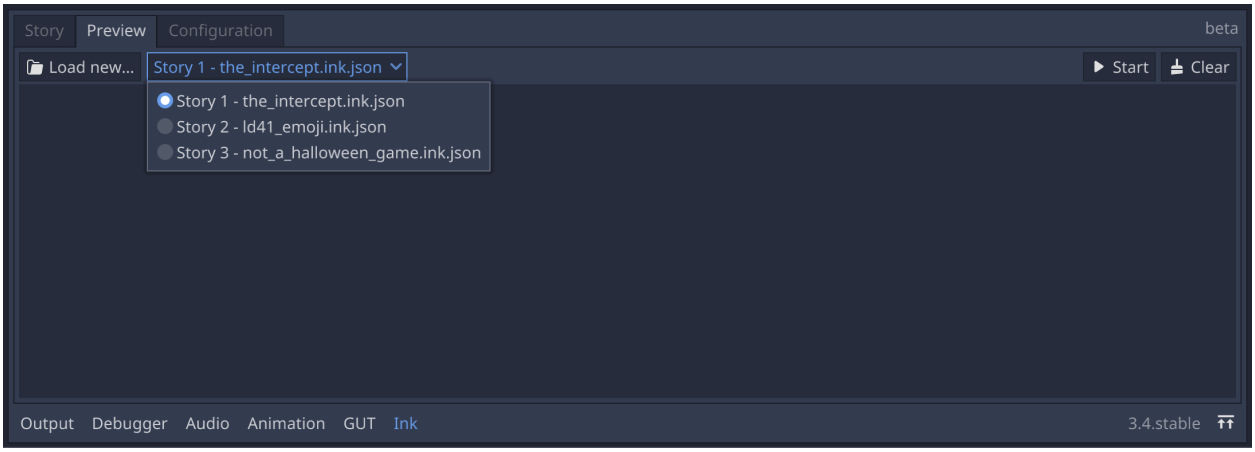
6.2.3 The Preview tab

In the Ink panel, select the *Preview* tab to preview a story. You can load any valid JSON through *Load new...*.

Unmanaged stories appear in a dropdown menu once they are loaded.

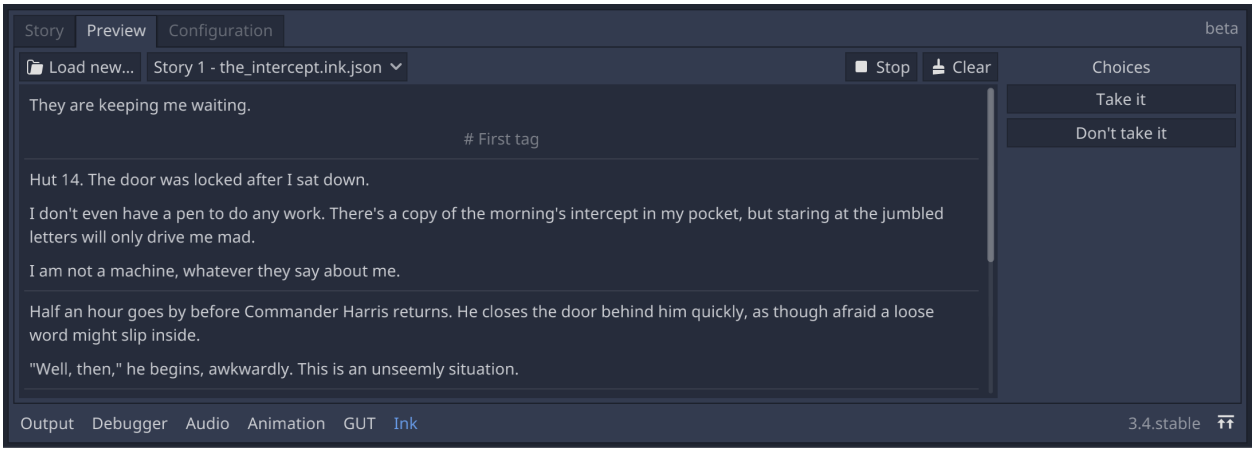


Managed stories appear automatically in the drop down menu as soon as they are registered.



To start a preview, select a story from the drop down menu and press *Start*. At any point during the execution, you can press *Clear* to remove all the previous lines.

The previewer automatically unfolds the story until it encounters a branch. When input in required, choices are displayed on the right side of the panel.



Any error encountered during the execution of the preview is printed to the output console.

CHAPTER

SEVEN

DIFFERENCES BETWEEN THE GDSCRIPT AND C# APIS

There are subtle differences between the original C# runtime and the GDScript version, but since the two APIs are mostly compatible, it's a good idea to take a look at the original documentation.

7.1 Style

Functions are all snake_cased rather than CamelCased. For instance `ContinueMaximally` becomes `continue_maximally`.

7.2 *inkgd*'s runtime node

Since GDScript doesn't support static properties, any static property was moved into a singleton node called *InkRuntime* which needs to be added to the current tree before starting the story.

This singleton node is added to the AutoLoad list of your project automatically when the editor plugin is activated. If you don't want to use the plugin, the runtime node can be registered manually, see *here* for more information.

Alternatively, you can also manage the singleton in code. Import `res://addons/inkgd/runtime.gd` in your script, then call the appropriate methods in `_ready()` and `_exit_tree()` to add/remove `res://addons/inkgd/runtime/static/ink_runtime.gd` to/from the tree.

```
var InkRuntime = load("res://addons/inkgd/runtime.gd")


# Since the tree is locked during '_enter_tree' and '_exit_tree',
# the node has to be added and removed through deferred calls.
func _ready():
    call_deferred("_add_runtime")


func _exit_tree():
    call_deferred("_remove_runtime")


func _add_runtime():
    InkRuntime.init(get_tree().root)
```

(continues on next page)

(continued from previous page)

```
func _remove_runtime():
    InkRuntime.deinit(get_tree().root)
```

InkRuntime contains a few configuration settings you may want to tweak, see the *API documentation*.

Note: When using InkPlayer, you don't need to manually add the runtime node to the tree. All the properties defined on InkRuntime are also available on InkPlayer, use them instead if you did not instantiate the node by yourself.

7.3 Getting and setting variables

Since the `[]` operator can't be overloaded in GDScript, simple `get` and `set` calls replace it.

7.3.1 GDScript API

```
story.variables_state.get("player_health")
story.variables_state.set("player_health", 10)
```

7.3.2 Original C# API

```
_inkStory.VariablesState["player_health"]
_inkStory.VariablesState["player_health"] = 10
```

7.4 Variable Observers

The event / delegate mechanism found in C# is translated into a signal-based logic in the GDScript runtime.

7.4.1 GDScript API

```
story.observe_variable("health", self, "_observe_health")

func _observe_health(variable_name, new_value):
    set_health_in_ui(int(new_value))
```

7.4.2 Original C# API

```
_inkStory.ObserveVariable("health", (string varName, object newValue) => {
   SetHealthInUI((int)newValue);
});
```

7.5 External Functions

The event / delegate mechanism found in C# is again translated into a signal-based logic.

7.5.1 GDScript API

```
# GDScript API

story.bind_external_function("multiply", self, "_multiply", true)

func _multiply(arg1, arg2):
    return arg1 * arg2
```

7.5.2 Original C# API

```
// Original C# API

_inkStory.BindExternalFunction ("multiply", (int arg1, float arg2) => {
    return arg1 * arg2;
}, true);
```

7.6 Handlers

Starting with **ink** version 1.0.0, it's possible to attach different types of handlers to a story to receive callbacks. In C#, they are implemented using events. In *inkgd*, they are again implemented using signals.

7.6.1 GDScript API

```
signal on_error(message, type)
signal on_did_continue()
signal on_make_choice(choice)
signal on_evaluate_function(function_name, arguments)
signal on_complete_evaluate_function(function_name, arguments, text_output, result)
signal on_choose_path_string(path, arguments)
```

7.6.2 Original C# API

```
public event Ink.ErrorHandler onError;
public event Action onDidContinue;
public event Action<Choice> onMakeChoice;
public event Action<string, object[]> onEvaluateFunction;
public event Action<string, object[], string, object> onCompleteEvaluateFunction;
public event Action<string, object[]> onChoosePathString;
```

The new handler system also supports reporting errors and warnings. It's recommended that you connect a handler to `on_error` to receive them.

7.7 Error Management

The original implementation relies on C#'s exceptions to report and recover from inconsistent states. Exceptions are not available in GDScript, so the runtime may behave slightly differently. In particular, if an error or an exception is encountered during `story.continue()`, the story may be inconsistent state even though it can still move forward after calling `story.reset_errors()`.

Runtime exceptions are emitted through *exception_raised*. For more information, refer to *this document*.

Note: *InkPlayer* has a different API regarding handlers and signals and fowards *exception_raised*.

7.8 Getting the output of `evaluate_function`

`evaluate_function` evaluates an **ink** function from GDScript. Since it's not possible to have in-out variables in GDScript you need to pass `true` to `return_text_output` to retrieve the text output of the function. `evaluate_function` will then return a dictionary containing both the return value and the output text.

```
# story.ink
#
# === function multiply(x, y) ===
#     Hello World
#     ~ return x * y
#

var result = story.evaluate_function("multiply", [5, 3])
# result == 15

var result = story.evaluate_function("multiply", [5, 3], true)
# result == {
#     "result": 15,
#     "output": "Hello World"
# }
```

Note: *InkPlayer* splits this function into two different functions, `evaluate_function` and `evaluate_function_and_get_output`, instead of a boolean flag.

7.9 Observing Variables

To be added.

CHAPTER

EIGHT

ERROR MANAGEMENT

8.1 C# Runtime

The original implementation relies on two mechanisms to report issues: runtime exceptions and story error/warnings.

8.1.1 Runtime Exceptions

The runtime can raise three types of exceptions.

1. Regular System Exceptions, which are usually thrown when the runtime encountered an error it can't recover from. They result from a misconfiguration or the misuse of an API.
2. Argument exceptions, which are very similar to the ones above. They may occur when executing Ink functions from C# while providing unsupported arguments.
3. Story Exception, which are **ink** errors that can't be caught during the compilation and mean that the author made a mistake when writing the ink.

8.1.2 Story Error / Warnings

These errors are usually recoverable. Recent versions of the runtime allow hooking a handler to receive the second kind of error in real-time. If no callbacks are provided, they are raised as Story Exceptions.

8.2 GDScript Runtime

GDscript doesn't use exceptions, thus *inkgd* can't map the C# runtime's behavior. In *inkgd*, errors are split into two categories: *Exceptions* and *Errors*.

8.2.1 Vanilla inkgd

Exceptions

Exceptions are reported through the `exception_raised` signal declared on `__InkRuntime`. Depending on the build type the runtime will behave differently.

1. This is a debug build:
    - if the runtime is running in the editor (for instance, in an editor plugin), exceptions and their stack traces are printed to the console;

- if the runtime is running in a standalone executable (a game), exceptions are pushed to the editor/terminal using `push_error`;
- if `__InkRuntime.stop_execution_on_exception` is `true`, exceptions are reported using `assert` instead, which pauses the execution and makes them explicit.

2. This is a release build:
   - nothing happens; the error is ignored.

If a handler is connected to `exception_raised`, the exception is also sent to the handler.

Errors

Errors are reported through the `on_error` signal declared on `story.gd` and stored in `current_errors` / `current_warnings`. If no handlers are connected to `on_error`, errors are raised as exceptions and reported through `__InkRuntime.exception_raised` instead.

If the runtime is running in a debug build, there are no handlers connected to `on_error` and `__InkRuntime.stop_execution_on_error` is set to `true`, the execution will pause each time a warning/error is encountered.

8.2.2 InkPlayer

Using `InkPlayer` simplifies error management. Errors and warnings can be observed through `exception_raised`; while exceptions can be monitored through `error_encountered`.

CHAPTER

NINE

PERFORMANCE

inkgd is overall slow. The high number of allocations that **ink** requires makes GDScript choke.

For instance, loading The Intercept will create about 30,000 objects. *ipsuminious 12* —a generated story based on The Intercept that contains about 90,000 words and 4,000 choices— will create a whopping 360,000 objects.

GDScript struggles to keep the performances at acceptable levels when allocating such a large number of objects, making the creation of stories the major bottleneck.

There's also a significant memory footprint. *ipsuminious 12* takes about 250 MB of RAM.

9.1 Benchmarks

The time required to create and allocate a story is benchmarked below.

All benchmarks use variations of *ipsuminious*, a story based on the Intercept, which can replicate itself to artificially increase the number of words and constructs. The growth is linear. Size 12 is 12 times bigger than size 1.

ipsuminious doesn't reflect real-world conditions, but should still give you a rough idea of the time budget you need to consider.

9.1.1 Size 1

About the size of *The Intercept*.

Table 1: Statistics

Bytecode	Words	Knots	Stitches	Functions	Choices	Gathers	Diverts
120 KB	7,650	34	30	3	340	95	230

Table 2: Performances

CPU	OS	Time
AMD Ryzen 7 5800X (3.80 GHz)	Windows 11	600 milliseconds
Apple M1 (3.20 GHz)	macOS Monterey	360 milliseconds
Intel i5-6267U (2.90 GHz)	macOS Monterey	1.52 seconds
Apple A14 Bionic	iOS 15	430 milliseconds
Apple A9	iOS 13	2.57 seconds

9.1.2 Size 6

Table 3: Statistics

Bytecode	Words	Knots	Stitches	Functions	Choices	Gathers	Diverts
714 KB	45,900	199	180	18	2040	570	1370

Table 4: Performances

CPU	OS	Time
AMD Ryzen 7 5800X (3.80 GHz)	Windows 11	3.32 seconds
Apple M1 (3.20 GHz)	macOS Monterey	2.12 seconds
Intel i5-6267U (2.90 GHz)	macOS Monterey	8.91 seconds
Apple A14 Bionic	iOS 15	2.80 seconds
Apple A9	iOS 13	42.65 seconds

9.1.3 Size 12

Average novels contain about 90,000 words.

Table 5: Statistics

Bytecode	Words	Knots	Stitches	Functions	Choices	Gathers	Diverts
1.4 MB	91,800	397	360	36	4,080	1,140	2,738

Table 6: Performances

CPU	OS	Time
AMD Ryzen 7 5800X (3.80 GHz)	Windows 11	6.65 seconds
Apple M1 (3.20 GHz)	macOS Monterey	4.42 seconds
Intel i5-6267U (2.90 GHz)	macOS Monterey	17.64 seconds
Apple A14 Bionic	iOS 15	6.10 seconds
Apple A9	iOS 13	1 minute 44 seconds

9.2 Alternatives to *inkgd*

If vanilla *inkgd* proves too slow for your needs, there are two options involving Godot Mono.

1. use the (undocumented) compatibility layer of inkgd, that exposes the official C# implementation while keeping the same API;
2. use the official C# implementation through godot-ink.

CHAPTER

TEN

MIGRATING TO GODOT MONO

To be added. In the meantime, see the example project and story_player.gd.

CHAPTER

ELEVEN

INKGD API

11.1 InkList

Inherits: Reference

11.1.1 Description

The underlying type that's used to store an instance of a list in ink. It's not used for the *definition* of the list, but for a list value that's stored in a variable. Somewhat confusingly, it's backed by a Dictionary, and has nothing to do with an Array.

11.1.2 Properties

To be added.

11.1.3 Property Descriptions

To be added.

11.2 InkPlayer

Inherits: Node

11.2.1 Description

A convenience node to run *inkgd*. Additional information on how to use it is available in *Using InkPlayer*.

11.2.2 Properties

Exported Properties

Resource	*ink_player*	
bool	*loads_in_background*	`true`

Read/Write Properties

bool	*allow_external_function_fallbacks*	`false`
bool	*do_not_save_default_values*	`false`
bool	*stop_execution_on_exception*	`false`
bool	*stop_execution_on_error*	`false`

Read Only Properties

bool	*can_continue*	`false`
bool	*async_continue_complete*	`false`
String	*current_text*	`""`
Array	*current_choices*	`[]`
Array	*current_tags*	`[]`
Array	*global_tags*	`[]`
bool	*has_choices*	`false`
bool	*current_flow_name*	`"DEFAULT_FLOW"`
bool	*alive_flow_names*	`[]`
bool	*current_flow_is_default_flow*	`true`
bool	*current_current_path*	`""`

11.2.3 Methods

Story Creation

void	*create_story* ( )
void	*reset* ( )
void	*destroy* ( )

Story Flow

String	*continue_story* ( )
String	*continue_story_async* (float millisecs_limit_async)
String	*continue_story_maximally* ( )
void	*choose_choice_index* (int index)
void	*choose_path* (String path_string)
void	*switch_flow* (String flow_name)
void	*switch_to_default_flow* ( )
void	*remove_flow* (String flow_name)
Array	*tags_for_content_at_path* (String path)
int	*visit_count_at_path* (String path)

State Management

String	*get_state* ( )
String	*copy_state_for_background_thread_save* ( )
void	*background_save_complete* ( )
void	*set_state* (String state)
void	*save_state_to_path* (String path)
void	*save_state_to_file* (File file)
void	*load_state_from_path* (String path)
void	*load_state_from_file* (File file)

Variables

Variant	*get_variable* (String name)
void	*set_variable* (String name, Variant value)
void	*observe_variables* (Array variable_names, Object object, String method_name)
void	*observe_variable* (String variable_name, Object object, String method_name)
void	*remove_variable_observer* (Object object, String method_name, String specific_variable_name)
void	*remove_variable_observer_for_all_variables* (Object object, String method_name)
void	*remove_all_variable_observers* (String specific_variable_name)

Functions

void	*bind_external_function* (String func_name, Object object, String method_name, bool lookahead_safe=false)
void	*unbind_external_function* (String func_name)
InkFunctionResult	*evaluate_function* (String function_name, Array arguments)
InkList	*create_ink_list_with_origin* (String origin_list_name)
InkList	*create_ink_list_from_item_name* (String item_name,)

11.2.4 Signals

- **exception (** String message, PoolStringArray stack_trace **)**

Emitted when the **ink** runtime encountered an exception. Exception are usually not recoverable as they corrupt the state. `stack_trace` is optional and contains each line of the stack trace leading to the exception for logging purposes.

- **loaded (** bool successfully **)**

Emitted with `true` when the runtime had loaded the JSON content and created the story. If an error was encountered, `successfully` will be `false` and error will appear in Godot's output.

- **continued (** String text, Array tags **)**

Emitted with the text and tags of the current line when the story successfully continued.

- **interrupted ( )**

Emitted when using *continue_async*, if the time spent evaluating the ink exceeded the alloted time.

- **prompt_choices (** Array choices **)**

Emitted when the player should pick a choice. The choices are string values.

- **choice_made (** String choice **)**

Emitted when a choice was reported back to the runtime.

- **function_evaluating (** String function_name, Array arguments **)**

Emitted when an external function is about to evaluate.

- **function_evaluated (** String function_name, Array arguments, *InkFunctionResult* function_result **)**

Emitted when an external function evaluated.

- **path_string_choosen (** String path, Array arguments **)**

Emitted when an external function evaluated.

- **ended ( )**

Emitted when the story ended.

11.2.5 Property Descriptions

- Resource **ink_file**

The compiled **ink** file (.json) to play. While you can set this property to any resource, it should be an instance of *InkResource*.

- bool **loads_in_background**

Default	`true`

When `true` the story will be created in a separate threads, to prevent the UI from freezing if the story is too big. Note that on platforms where threads aren't available, the value of this property is ignored.

- bool **allow_external_function_fallbacks**

Default	`true`
Setter	set_aeff(value)
Getter	get_aeff()

`true` to allow external function fallbacks, `false` otherwise. If this property is `false` and the appropriate function hasn't been binded, the story will output an error.

- bool **do_not_save_default_values**

Default	`true`
Setter	set_dnsdv(value)
Getter	get_dnsdv()

When set to `true`, *inkgd* skips saving global values that remain equal to the initial values that were declared in ink. This property matches the static property declared in VariablesState.cs.

- bool **stop_execution_on_exception**

Default	`true`
Setter	set_speoex(value)
Getter	get_speoex()

When set to `true`, *inkgd* uses `assert()` instead of `push_error` to report exceptions, thus making them more explicit during development.

- bool **stop_execution_on_error**

Default	`true`
Setter	set_speoer(value)
Getter	get_speoer()

When set to `true`, *inkgd* uses `assert()` instead of `push_error` to report errors, thus making them more explicit during development.

- bool **story**

Default	`null`
Getter	get_can_story()

The underlying story, exposed for convenience. For instance, you may want to create a new InkList, which in certain acses needs a reference to the story to be constructed.

- bool **can_continue**

Default	`false`
Getter	get_can_continue()

`true` if the story can continue (i. e. is not expecting a choice to be choosen and hasn't reached the end).

- bool **async_continue_complete**

Default	`false`
Getter	get_async_continue_complete()

If `continue_async` was called (with milliseconds limit > 0) then this property will return false if the ink evaluation isn't yet finished, and you need to call it again in order for the continue to fully complete.

- String **current_text**

Default	`""`
Getter	get_current_text()

The content of the current line.

- Array **current_choices**

Default	`""`
Getter	get_current_choices()

The current choices. Empty is there are no choices for the current line.

- Array **current_tags**

Default	`[]`
Getter	get_current_tags()

The current tags. Empty is there are no tags for the current line.

- Array **global_tags**

Default	`[]`
Getter	get_global_tags()

The global tags for the story. Empty if none have been declared.

- bool **has_choices**

Default	`false`
Getter	get_has_choices()

`true` if the story currently has choices, `false` otherwise.

- bool **current_flow_name**

Default	`"DEFAULT_FLOW"`
Getter	get_current_flow_name()

The name of the current flow.

- bool **alive_flow_names**

Default	`[]`
Getter	get_alive_flow_names()

The names of all flows currently alive.

- bool **current_flow_is_default_flow**

Default	`true`
Getter	get_current_flow_is_default_flow()

`true` if the current flow is the default flow.

- bool **current_current_path**

Default	`""`
Getter	get_current_path()

The current story path.

11.2.6 Method Descriptions

- void **create_story ( )**

Creates the story, based on the value of *ink_player*. The result of this method is reported through *loaded*.

- void **reset ( )**

Reset the story back to its initial state as it was when it was first constructed.

- void **destroy ( )**

Destroys the current story. Always call this method first if you want to recreate the story.

- String **continue_story ( )**

Continues the story.

- String **continue_story_async ( )**

An "asynchronous" version of `continue_story` that only partially evaluates the ink, with a budget of a certain time limit. It will exit **ink** evaluation early if the evaluation isn't complete within the time limit, with the `async_continue_complete` property being false. This is useful if the evaluation takes a long time, and you want to distribute it over multiple game frames for smoother animation. If you pass a limit of zero, then it will fully evaluate the **ink** in the same way as calling `continue_story`.

To get notified when the evaluation is exited early, you can connect to the `interrupted` signal.

- String **continue_story_maximally ( )**

Continue the story until the next choice point or until it runs out of content. This is as opposed to `continue` which only evaluates one line of output at a time.

- void **choose_choice_index (** int index **)**

Chooses a choice. If the story is not currently expected choices or the index is out of bounds, this method does nothing.

- void **choose_path (** String path_string **)**

Moves the story to the specified knot/stitch/gather. This method will throw an error through *exception* if the path string does not match any known path.

- void **switch_flow (** String flow_name **)**

Switches the flow, creating a new flow if it doesn't exist.

- void **switch_to_default_flow ( )**

Switches the the default flow.

- void **remove_flow** (String flow_name)

Remove the given flow.

- Array **tags_for_content_at_path** (String path)

Returns the tags declared at the given path.

- int **visit_count_at_path** (String path)

Returns the visit count of the given path.

- String **get_state** ( )

Gets the current state as a JSON string. It can then be saved somewhere.

- String **copy_state_for_background_thread_save** ( )

If you have a large story, and saving state to JSON takes too long for your framerate, you can temporarily freeze a copy of the state for saving on a separate thread. Internally, the engine maintains a "diff patch". When you've finished saving your state, call `background_save_complete` and that diff patch will be applied, allowing the story to continue in its usual mode.

- void **background_save_complete** ( )

See `copy_state_for_background_thread_save`. This method releases the "frozen" save state, applying its patch that it was using internally.

- void **set_state** (String state)

Sets the state from a JSON string.

- void **save_state_to_path** (String path)

Saves the current state to the given path.

- void **save_state_to_file** (File file)

Saves the current state to the file.

- void **load_state_from_path** (String path)

Loads the state from the given path.

- void **load_state_from_file** (File file)

Loads the state from the given file.

- Variant **get_variable** (String name)

Returns the value of variable named 'name' or 'null' if it doesn't exist.

- void **set_variable** (String name, Variant value)

Sets the value of variable named 'name'.

- void **observe_variables** (Array variable_names, Object object, String method_name)

Registers an observer for the given variables.

- void **observe_variable** (String variable_name, Object object, String method_name)

Registers an observer for the given variable.

- void **remove_variable_observer** (Object object, String method_name, String specific_variable_name)

Removes an observer for the given variable name. This method is highly specific and will only remove one observer.

- void **remove_variable_observer_for_all_variables** (Object object, String method_name)

Removes all observers registered with the couple object/method_name, regardless of which variable they observed.

- void **remove_all_variable_observers** (String specific_variable_name)

Removes all observers observing the given variable.

- void **bind_external_function** (String func_name, Object object, String method_name, bool lookahead_safe=false)

Binds an external function.

- void **unbind_external_function** (String func_name)

Unbinds an external function.

- *InkFunctionResult* **evaluate_function** (String function_name, Array arguments)

Evaluate a given **ink** function, returning both its return value and its text output.

- *InkList* **create_ink_list_with_origin** (String origin_list_name,)

Creates a new empty InkList that's intended to hold items from a particular origin list definition.

- *InkList* **create_ink_list_from_item_name** (String item_name,)

Creates a new InkList from the name of a preexisting item.

11.3 InkRuntime

Inherits: Node

11.3.1 Description

A node encapsulating the static properties of the runtime and managing exceptions.

Exceptions don't exists in GDScript, but they are *emulated* by the runtime and reported through *exception_raised*.

11.3.2 Properties

bool	*do_not_save_default_values*	`false`
bool	*stop_execution_on_exception*	`false`
bool	*stop_execution_on_error*	`false`

11.3.3 Signals

- **exception_raised** (String message, PoolStringArray stack_trace)

Emitted when the runtime encounters an exception. Exceptions are not recoverable and may corrupt the state. They are the consequence of either a programmer error or a bug in the runtime.

11.3.4 Property Descriptions

- bool **do_not_save_default_values**

Default	`true`
Setter	set_dnsdv(value)
Getter	get_dnsdv()

When set to `true`, *inkgd* skips saving global values that remain equal to the initial values that were declared in ink. This property matches the static property declared in VariablesState.cs.

- bool **stop_execution_on_exception**

Default	`true`
Setter	set_speoex(value)
Getter	get_speoex()

When set to `true`, *inkgd* uses `assert()` instead of `push_error` to report exceptions, thus making them more explicit during development.

- bool **stop_execution_on_error**

Default	`true`
Setter	set_speoer(value)
Getter	get_speoer()

When set to `true`, *inkgd* uses `assert()` instead of `push_error` to report errors, thus making them more explicit during development.

11.4 InkFunctionResult

Inherits: Reference

11.4.1 Description

A plain object that encapsulates the result of evaluating an function from Ink.

11.4.2 Properties

String	*text_output*
Variant	*return_value*

11.4.3 Property Descriptions

- String **text_output**

The text output generated during the function's evaluation.

- Variant **return_value**

The return value of the function.