Datalogue

Release 0.1

Olivier Boucard

May 14, 2022

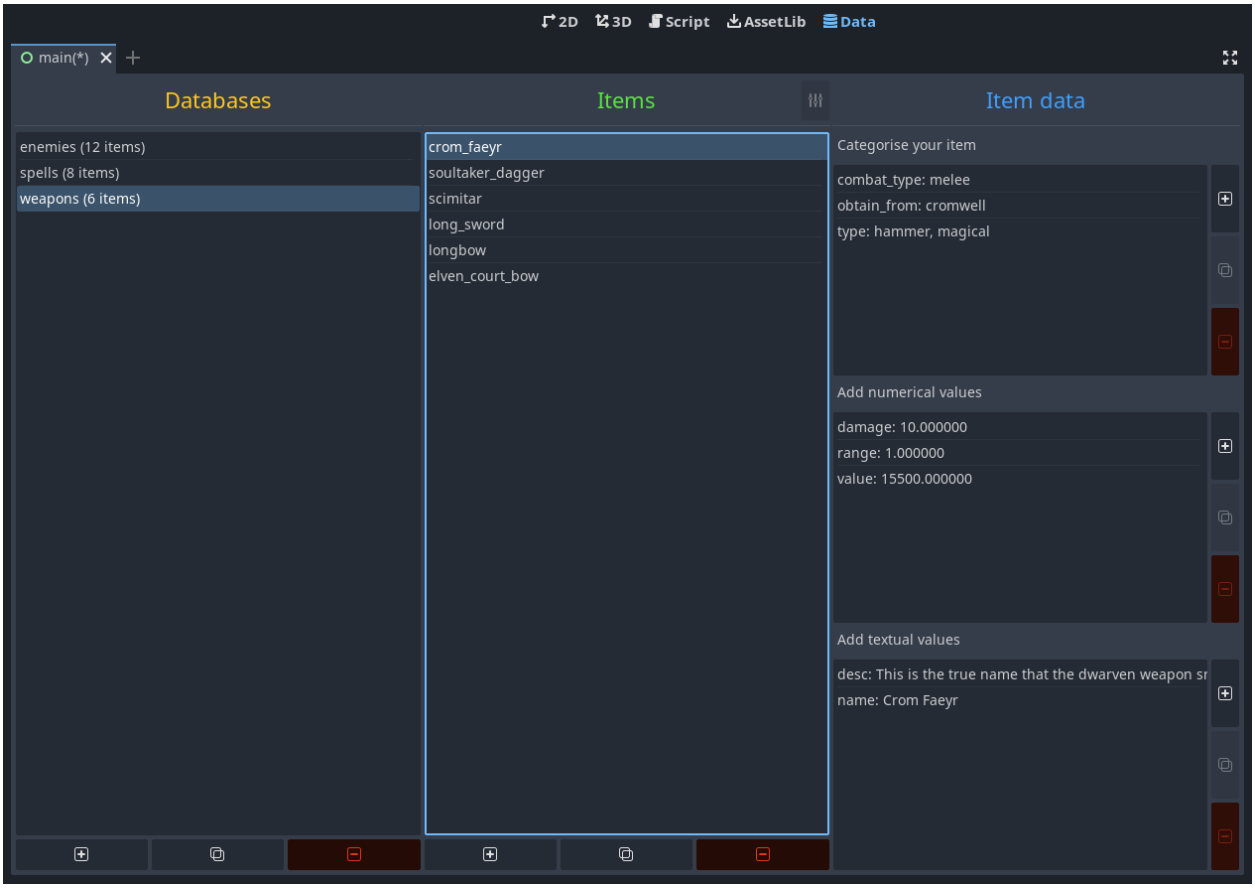
CONTENTS

CHAPTER

ONE

INTRODUCTION

Attention: This plugin is **not** compatible with Godot 3. It uses features that are only available to the incoming Godot 4.



Datalogue is a data management plugin for the Godot Engine. It originates from my previous plugin, CardEngine, and the way cards are managed in it. I took the card database concept, made it better and working with all kind of data. The idea is to provide a database that can be queried like a SQL database but with the flexibility of a format like JSON.

1.1 Why using Datalogue?

When storing static data, there is multiple approaches possible. One can use a SQL database, like SQLite, use text files like JSON or use structure in the source code like a Dictionary. All this solutions come with advantages and drawbacks, Datalogue tries to keep the firsts while avoiding the lasts.

Here's why you would want to use Datalogue:

- Flexibility of the data structure, you don't need to define what the data should look like
- Query language, you can quickly and easily retrieve the data you are looking for
- GDScript API, you can interact with your data directly from your scripts
- Integrated UI, you can manage your data without leaving the Godot Editor
- The database files are VCS friendly

1.2 When to use Datalogue?

Datalogue can be use in all kind of games and projects. Use it when you want to store static data.

Datalogue databases are not meant to be modified once the project is exported. It however offers the possibility to manipulate data in memory, which can be saved and loaded to/from a user folder (not yet implemented).

1.3 How to use Datalogue?

Datalogue can be installed by downloading the code from the repository (https://github.com/BraindeadBZH/datalogue) or from the Godot's AssetLib. You just need the files under `/addons/datalogue` to be copied into any Godot project and then you need to enable the plugin inside the project settings.

Quick start:

1. The Datalogue UI is accessible by switching to the "Data" view at the top of the window
2. Add a database by pressing the + button at the bottom of the first column
3. Select the newly created database in the list
4. Add an item by pressing the + button at the bottom of the middle column
5. Select the newly created item
6. Add some data using the + buttons on the side of the last column
7. You can retrieve a database in your GDScript by using `Datalogue.get_database("database_id")`
8. You can retrieve item data by using `item.get_classification("classification_id")`, `item.get_value("value_id")` or `item.get_text("text_id")`

CHAPTER

TWO

USAGE

Datalogue is designed to be straightforward to use. So this should be a short but important read.

2.1 Installation

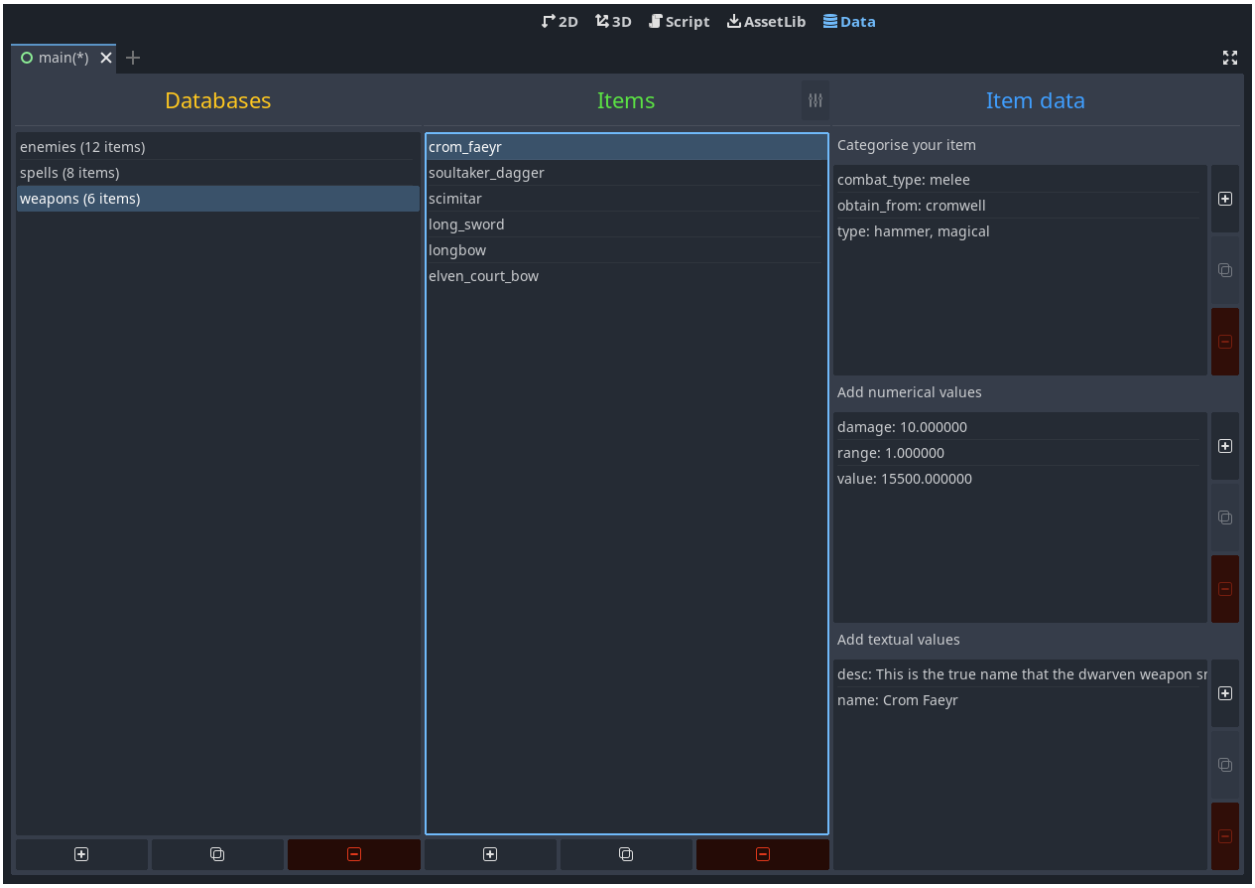
There is two way to install:

- Using the Godot's AssetLib:
  - Search for "Datalogue" directly from the Godot Editor AssetLib tool
  - Press "Download"
- From Github:
  - Go to the repository: https://github.com/BraindeadBZH/datalogue
  - Download the code as an archive
  - Decompress the archive
  - Copy the addons folder to your project

In both case, you need to enable the plugin in your project's settings, in the "Plugins" tab.

2.2 Using the UI

You can access the Datalogue UI at any time by choosing the "Data" tool at the top-center of the project viewport.



The UI is split into 3 mains sections displayed in columns:

- The first column is to manage your databases
- The second column is to manage your database's items
- The third column is to manage your item's data, it is split into 3 sub-sections, displayed in rows:
  - The first row is to manage classifications for your item
  - The second row is to manage numerical values for your item
  - The third row is to manage textual values for your items

All the sections work on the same model:

- A list which display the content:
  - A single click to select an object from the list
  - A double click to edit the selected object
- Three Buttons to manage the content:
  - The first button is to create a new object
  - The second button is to duplicate an object
  - The third button is to delete an object

(Note: be careful all modifications are definitive, there is no undo/redo)

In addition you can press the button at the top-right of the "Items" section to filter the displayed items.

2.3 Using the API

Accessing the data in your game is meant to be done using GDScript. The main entry point is a Autoload singleton *Datalogue*. For a detailed API reference: go to *the API reference page*.

CHAPTER

THREE

API

Datalogue offers multiple GDScript classes to access your data in scripts.
Some of the offered functions can only be accessed while in tool mode (i.e. editor scripts).

3.1 Datalogue

`Datalogue` is the entry point to the API, it is an autoload singleton.

At runtime you can use:

- `databases() -> Dictionary[string][DlDatabase]` to retrieve all the databases as map using the id as key
- `get_database(id: String) -> DlDatabase` to retrieve the database with the given id or null if it does not exist

In tool mode you can use:

- `validate_id(id: String) -> String` to check if the given string can be used for as a database id, if it can, the returned string is empty, otherwise it contains an error message
- `update_database(modified_db: DlDatabase, old_id: String = "") -> void` to save to disk a modification made to the given database, if you change the database id the old id should also given.
- `copy_database(origin: DlDatabase, new_id: String) -> void` to create a copy on disk of the given database with the given id
- `delete_database(id: String) -> void` to remove a database from disk

In tool mode you can also connect to the following signals:

- `database_added()` emitted when a database is inserted into the list
- `database_updated(db: DlDatabase)` emitted when a database has been modified in the list
- `database_removed()` emitted when a database is removed from the list

3.2 DlDatabase

`DlDatabase` represents 1 database designed by its id and containing items.

At runtime you can use:

- `id() -> String` to retrieve the database's id
- `count() -> int` to retrieve the number of items in the database
- `statistics() -> Dictionary` to retrieve some statistics about this database (more details below)
- `items() -> Dictionary` to retrieve all the items in the form `[<item id>: <item>]`
- `item_exists(id: String) -> bool` to check if an item is in the database
- `get_item(id: String) -> DlItem` to retrieve the item with the given id or null if it does not exist

In tool mode you can use:

- `set_id(new_id: String) -> void` to change the database's id
- `duplicate(new_id: String = "") -> DlDatabase` to create a copy of this database with the given `new_id`. If `new_id` is empty then the copy has the same id as this database. Note: the copy is not written to disk, contrary to the `Datalogue.copy_database function`
- `validate_item_id(id: String) -> String` to check if the given string can be used for as an item id, if it can, the returned string is empty, otherwise it contains an error message
- `add_item(item: DlItem) -> void` to add the given item to the database. Note: the change is not written to disk until you call `Datalogue.update_database`
- `copy_item(origin: DlItem, new_id: String) -> void` to create a copy of the given item with the given `new_id`. Note: the change is not written to disk until you call `Datalogue.update_database`
- `update_item(modified_item: DlItem, old_id: String = "") -> void` to update the given `modified_item`. If you have changed the item id, the `old_id` should be also given. Note: the change is not written to disk until you call `Datalogue.update_database`
- `remove_item(id: String) -> void` to remove the item with the given id from the database. Note: the change is not written to disk until you call `Datalogue.update_database`

In tool mode you can also connect to the following signals:

- `changed()` emitted when the database has changed

The statistics are given as a Dictionary with the following structure:

- `["items"]` contains the total number of item in the database
- `["classes"]["<classif id>"]` contains the number of items with the given classification
- `["classes_values"]["<classif id>"]["<value>"]` contains the number of items with the given classification value
- `["values"]["<value id>"]` contains the number of items with the given value
- `["values_min"]["<value id>"]` contains the minimum of the given value
- `["values_max"]["<value id>"]` contains the maximum of the given value
- `["texts"]["<text id>"]` contains the number of items with the given text