Amulet Map Editor Documentation

The Amulet Map Editor Team

Feb 01, 2019

Contents:

CHAPTER 1

api module

1.1 data_structures module

1.2 paths module

COMMANDS_DIR
Points to the directory where 3rd party commands are loaded from when running in command-line mode

FORMATS_DIR
Points to the directory where world format loaders are loaded from

DEFINITIONS_DIR
Points to the directory where block/entity/tile entity definitions are loaded from

1.3 box module

1.4 world module

1.5 world_loader module

CHAPTER 2

Design Notes

2.1 Block Definitions

Relevant classes: `version_definitions.definition_manager.DefinitionManager`

2.1.1 Intro

One of the biggest problems with *Minecraft: Java Edition 1.13* is that it switched the structure of the world save data from using pairs of numerical IDs to define blocks in the world. Previously these IDs were constant and represented a single block. However, 1.13 changed it so blocks were identified by blockstates as strings. Some blocks, like Noteblocks for example, went from storing their data as NBT to using blockstates, this caused an issue where not all blocks have a clear `(ID, data) <-> blockstate` translation. This has prompted us to use blockstate strings as block identifiers in the editor and only numerical IDs where absolutely needed.

2.1.2 Versioned Blocks

Each supported Minecraft version must define all blocks and provide the following information:

- The blockstate string that the version uses (IE: `minecraft:stone[variant=stone]` for Java 1.12) as a dictionary/JSON key for the rest of the following information. If there's information inside the square brackets, the blockstate identifiers should be a child key with the base blockstate (`stone`) being the parent key.
- The ID of that blockstate for that version (IE: `[1,0]` for Java 1.12, `minecraft:stone` for Java 1.13)
- The `map_to` key that links the version defined block to a block that we have internally defined
- *Optional*: The `nbt` key if the block originally stored it's data as NBT before 1.13 and switched to blockstates in 1.13+ (IE: Noteblocks) **!!Incomplete!!**

2.1.3 Identifying and Loading Worlds

Each version definition is required to have a `identify()` and `load()` function in a python file with the same name as the directory containing it and the version definitions, but with underscores instead of dots (IE: `1.12` definitions would have `1_12.py`). The identify function doesn't do any loading but is given the directory path to the world and using the directory structure and NBT structure in the *level.dat*. This function returns `True` if it matches criteria to be loaded by that format loader, if not, `False` is to be returned.

`load()`'s function is to load a world with the appropriate format loader for the version definitions (IE: `1.12` loads via `anvil` and `1.13` loads via `anvil2`). When calling `load()`, the path to the world directory is given (it can be assumed that the accompanying `identify()` function has been called and has returned `True`) and the method is to return the resulting `api.world.World` object created from the world format loader.

This python file is also expected to have a global variable named `FORMAT`, which allows Amulet to output what format loader will be used when loading the world to the user.

2.2 World Design

Relevant Classes: `api.world.World`, `api.world.WorldFormat`

2.2.1 Intro

In order to make world loading simpler and more flexible to future changes with Minecraft, the Amulet Map Editor only loads/edits a proprietary format. By doing this, to support a Minecraft world format, the only things required are separate block/entity/tile entity definitions and a "conversion" wrapper, which converts the world data on disc to the "Amulet format"

2.2.2 The "Amulet Format"

The "Amulet Format" in a basic form is a wrapper and temporary storage of world data for editing. For blocks, the format expects the blocks to be integer-based, however, these integers are dynamic and are assigned to blocks as new ones are found with newly loaded chunks. IE: `minecraft:stone` won't always have an integer ID of 1, but may have one of 20 if it isn't encountered any time earlier when loading other chunks.

Despite using integer based IDs, these IDs are used only internally, any method that exposes blocks will normalize the ID to the internal string ID used by the editor (these string IDs are based off of the blockstates present in 1.13). The only way to reference a block will be through this way, integer IDs are entirely dynamic and should never be used except for loading/saving chunks.

2.2.3 Format Loaders

Each world format loader is separated into their own containers and don't do any editing of their own. Each format loader handles reading the world data then converting the blocks/entities/ tile entities into a format that the Amulet Format expects. Each format loader must inherit from `api.world.WorldFormat`

When loading, format loaders are first called via a class method `load()` which is expected to return a `api.world.World` instance. The method receives the path to directory of the world. Once the world is loaded, the format loader is only used to load and translate new data from the disc.

When saving, format loaders shouldn't make any assumptions about the previous format of the world since the Amulet Format doesn't keep track of that data. Due to this, while saving various attributes should be check and either saved or ignored depending on what data is present.

CHAPTER 3

command_line module

CHAPTER 4

version_definitions module

CHAPTER 5

world_utils module

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index