Contents

[General organization 1](#_Toc533539464)

[src/main/java 1](#_Toc533539465)

[init 1](#_Toc533539466)

[items 1](#_Toc533539467)

[blocks 1](#_Toc533539468)

[proxy 1](#_Toc533539469)

[util 1](#_Toc533539470)

[Main 1](#_Toc533539471)

[src/main/resources 2](#_Toc533539472)

[assets.modid 2](#_Toc533539473)

[Quick Instructionals 4](#_Toc533539474)

[Adding an Item: 4](#_Toc533539475)

[Adding a block: 4](#_Toc533539476)

# General organization

src/main/java – holds all java files for mod

init –

ModItems – initializes items in the mod and materials

ModBlocks – initializes blocks from the mod

ModFluids – initializes fluids from the mod and contains methods for registering all fluids with forge

ModRecipes – initializes smelting recipes. All recipes should be added in the init function and init should be called from Main.init

items – holds classes describing each item, special items get their own class

ItemBase – extends Item and implements IHasModel

* Basic item class with constructor that sets UnlocalizedName, RegistryName, CreativeTab, and adds to ITEMS list
* registerModels registers the model with Minecraft: Main.proxy.registerItemRenderer(this, 0, “inventory”)

blocks – holds classes describing each block, special blocks get their own class

BlockBase – extends Block and implements IHasModel

* Basic block class with constructor that calls super, sets UnlocalizedName, RegistryName, CreativeTab, and adds the block to BLOCKS and ITEMS
* registerModels registers the model with Minecraft: Main.proxy.registerItemRenderer(Item.getItemFromBlock(this), 0, "inventory");

BlockFluidBase – extends BlockFluidClassic/BlockFluidFinite and implements IHasModel

* Basic fluid block class that calls super and sets name and creative tab as well as adding to BLOCKS and ITEMS
* Register models registers the model with Minecraft with two lines:
  + Main.proxy.registerItemRenderer(Item.getItemFromBlock(this), 0, “inventory”);
  + ModelLoader.setCustomStateMapper(this, new StateMap.Builder().ignore(LEVEL).build());

fluids – holds classes describing each fluid, special fluids get their own class

FluidBase – extends Fluid

* Basic fluid class with constructor that calls super with name and resource locations and adds to FLUIDS list

proxy – holds Client and Common Proxy classes with registerItemRenderer, only client proxy actually does anything this this method

util – utility package

handler – package to hold all classes in charge on handling, i.e. registrations

* onItemRegister registers all mod items with Minecraft
* onBlockRegister registers all mod blocks with Minecraft
* onModelRegister calls registerModels of all items and blocks in ITEMS and BLOCKS that implements IHasModel, i.e. anything from this mod with a model

IHasModel – interface that declares registerModels method

Reference is a data holding class that defines MOD\_ID, NAME, VERSION, ACCEPTED\_VERSIONS, CLIENT\_PROXY\_CLASS, and COMMON\_PROXY\_CLASS

World – world package

ModWorldGen – class that controls generation of blocks in world

* generate: method that is called when new chunk is generated and is in charge of calling each dimensions ore generator
* generateOverworld: method this called by generate when new chunk is in the overworld. generateOre should be called for each ore that should be generated in the overworld.
* generateOre: generates chances number of veins of the given ore in the given chunk

Main – main class of mod, not quite sure exactly what is supposed to be here

* creates a public instance and proxy variables
* also contains three event handlers (uses are not exhaustive):
  + preInit: register worldgen and fluids
  + init: initialize recipes
  + postInit

## src/main/resources

### assets.modid

blockstates – package for json file describing blockstates of blocks. A new file for each block is needed

//for normal blocks

{

"variants": {

"normal": { "model": "<MOD\_ID>:<blockName>" }

}

}

//for fluid blocks

{

"forge\_marker": 1,

"variants": {

"normal": {

"model": "forge:fluid",

"transform": "forge:default-item",

"custom": { "fluid": "ruby\_fluid" }

}

}

}

lang – holds language files with formats as follows:

Items: item.<itemName>.name=<nameToDisplay>

Blocks: tile.<blockName>.name=<nameToDisplay>

Fluids: fluid.<fluidName>=<nameToDisplay>

#### models

block – holds model jsons for blocks added by mod. A new file for each block is needed.

{

"parent": "block/cube\_all",

"textures": { "all": "<MOD\_ID>:<texturesFolder>/<blockName>" }

}

item – holds model jsons for items added by mod. A new file for each item is needed including blocks

//for items

{

"parent": "item/generated",

"textures": { "layer0": "<MOD\_ID>:items/<itemName>" }

}

//for blocks

{

"parent": "sbmm:block/ruby\_block"

}

//for tools

{

"parent": "item/handheld",

"textures": { "layer0": "<MOD\_ID>:items/<toolName>" }

}

#### Textures

items – holds all png files for items in mod, must be named as <itemName>.png and be 16x16 pixels

blocks – holds all png files for blocks in mod, must be name as <blockName>.png

fluids – holds all png files for fluids in mod, must be named identical to resource location

Recipes – package to hold all json files for crafting recipes using the following templates

{

"type": "minecraft:crafting\_shaped",

"pattern":

[

"RRR",

"RRR",

"RRR"

],

"key":

{

"R": { "item": “<MOD\_ID>:<itemName>" }

},

"result":

{

"item": “<MOD\_ID>:<itemName>"

"count": 1

}

}

{

"type": "minecraft:crafting\_shapeless",

"ingredients":

[

{ "item": “<MOD\_ID>:<itemName>" },

{ "item": “<MOD\_ID>:<itemName>" }

],

"result":

{

"item": “<MOD\_ID>:<itemName>",

"count": 1

}

}

# Quick Instructionals

## Adding an Item

1. add item instantiation to ModItems in init
   1. public static final Item <ITEMNAME> = new <itemClass>(“<itemName>”);
      1. ITEMNAME: all caps itemName
      2. itemClass: class that inherits Item, ItemBase if no properties need to be changed
      3. itemName: same item name used elsewhere, needs to be unique among items
2. OPTIONAL: create class in items package to set specific properties
3. Add line to each language file
4. Add item json to models.item with itemName.json format as described above
5. Add item png to textures.items with itemName.png format as described above

## Adding a block

1. Optional: create a class in blocks package that extends BlockBase to set specific properties
2. Add item instantiation to ModBlocks in init
   1. Public static final Block <BLOCKNAME> = new <blockClass>(“<blockName>”, Material.<material> )
      1. BLOCKNAME: all caps blockName
      2. blockClass: class that inherits Block, BlockBase if no properties need to be changed
      3. blockName: same item name used elsewhere, needs to be unique among blocks. NOTE: good practice to add \_block to end so as to distinguish from items
      4. material: material category for block to inherit properties from
3. Add line to each language filea
4. OPTIONAL: if block is to be automatically generated on world gen, add generateOre to appropriate method, i.e. generateOverworld, generateUnderworld, and/or generateEnd
5. Create blockstates json, models.block json, and models.item json with blockName.json format as described above
6. Add block png to textures.blocks with blockName.png format as described above

## Adding Crafting Recipes

1. Create json file in recipes package with format shown above. NOTE: naming is not important for compilation, but PLEASE name descriptively.

## Adding Smelting recipes

1. Add appropriate line to init function in ModRecipes using the following template:
   1. GameRegistry.addSmelting(<ISourceClass>.<IItem/Block>, new ItemStack(<OSourceClass>.<OItem/Block>, <#>), <xp>)
      1. ISourceClass – class in which the input item is defined. For example: ModItems or ModBlocks for stuff from the mod and Items or Blocks for vanilla stuff
      2. IItem/Block – input item or block defined in ISourceClass
      3. OSourceClass – same as ISourceClass except should contain output item/block
      4. OItem/Block – output item or block defined in OSourceClass
      5. # - number of OItem/Blocks to ouput for each IItem/Block
      6. Xp – number of xp points to get each time this smelting recipe is used

## Adding Tools

1. Add tool material to ModItems. Only needed if material is not already created.
   1. public static final ToolMaterial MATERIAL\_<MATERIAL> = EnumHelper.addToolMaterial(<name>, <harvestLevel>, <maxUses>, <efficiency>, <damage>, <enchantability>);
      1. Reference values can be found in ToolMaterial (ctrl + left click)
2. Add tool Class. Only needed if basic Tool<toolType> class is insufficient.
   1. Constructor should take name and material and call super(material) as well as set UnlocalizedName, RegistryName, CreativeTab and add to ITEMS.
      1. Axe needs additional values when calling super: ATTACK\_DAMAGES and ATTACK\_SPEEDS
   2. registerModels should be called same as in ItemBase
3. Add tool to ModItems creating a new version of the previously implemented class.
   1. public static final Item<toolType> RUBY\_SWORD = new <toolClass>(<name>, <material>);
4. Add tool json to models.item and png to textures.items both named as <name>.<extension>
5. Add name line to language files: item.<toolName>.name=<Tool Name>
6. OPTIONAL: add recipes to recipe package

## Adding Fluids

1. OPTIONAL: create fluid specific class in fluids that extends FluidBase
2. Add fluid to ModFluids static declarations
   1. public static final Fluid <FLUID\_NAME>= new <FluidClass("<fluidName", new ResourceLocation(Reference.MOD\_ID, "fluids/<fluidName>\_still"), new ResourceLocation(Reference.MOD\_ID, "fluids/<fluidName>\_flow"));
3. OPTIONAL: add fluid block to ModBlocks static declarations
   1. public static final BlockFluidBase <FLUIDNAME>\_BLOCK = new BlockFluidBase("<fluidName>", ModFluids.<FLUID\_NAME>, Material.<MATERIAL>);
4. add blockstates json to blockstates package and textures to fluids package
5. OPTIONAL: add block and item jsons to appropriate models packages if fluid block was made