LEARNING minecraft

#minecraft

Table of Contents

About

Chapter 1: Getting started with minecraft

Remarks

This section provides an overview of what minecraft is, and why a developer might want to use it.

It should also mention any large subjects within minecraft, and link out to the related topics. Since the Documentation for minecraft is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
1.0.0	2011-11-18
1.2.5	2012-04-04
1.4.7	2013-01-09
1.6.4	2013-09-19
1.7.10	2014-06-26
1.8.9	2015-12-09
1.9.4	2016-05-10
1.10.2	2016-06-23
1.11	2016-11-14

Examples

Installing Minecraft

1. Buy Minecraft from here
2. Create a Mojang account or sign in if you already have one.
3. If you created a new account, verify your email.
4. Fill in your Payment Details. Make sure your on minecraft.net and you're on a secure connection (HTTPS)
5. Download and Run Minecraft

6. Open the Minecraft launcher

7. Log in with your email address if you purchased Minecraft after November 2012, or migrated your username to a Mojang account.

   Log in with your username if you have an older Minecraft account, and have not yet migrated to the new account format. You can migrate to a Mojang account from the Mojang accounts page.

8. Click on "Play" if you want to play on the newest version.

9. Enjoy the game.

Read Getting started with minecraft online: https://riptutorial.com/minecraft/topic/7952/getting-started-with-minecraft

Chapter 2: Creating a basic block with Forge

Introduction

Creating a simple, decorative block with Forge is one of the first tasks an aspiring modder will have to learn. How to do that has changed over the various versions of Minecraft and is probably at a "moderate" difficulty post 1.7.10 due to the sheer number of easy to make mistakes.

Remarks

If something goes wrong and your custom block (either when placed or held) has a missing texture (black/purple) or model (default cube that's too large when held) check the log. Problems of this sort will almost *always* show up in the log, provided that you have registered things correctly.

The first thing that will show up is `"Exception loading model for variant..."` line or similar, telling you which block or item failed to load properly. After a dozen or so lines beginning with `at...` will be another line starting `Caused by...`

This "Caused By" line is the important one, it will tell you which file failed to load properly and can be one of several errors, such as:

- Malformed JSON (your JSON file isn't valid and has a typo)
- File Not Found (the file Minecraft is looking for is not properly named or in the right place)
- Missing Variant Exception (your Blockstate JSON is incomplete)

You may even get more than one failing error for a single mistake! If you don't kno which block is the problem, comment out every block and item you know works, reduce the error list down to the block (or blocks) in question. You may have to do this several times.

Additionally, a mixing texture shows up as differently as a simple list all of the missing resources for a given domain (mod ID).

Examples

The Block Class

First we need a class that represents the block

```
public class CustomBlock extends Block {

    public CustomBlock () {
        super(Material.ROCK);
        setHardness(1.0f);
        setHarvestLevel("pickaxe", 0);
        setResistance(1.0f);
        setCreativeTab(CreativeTabs.DECORATIONS);
        this.setSoundType(SoundType.STONE);
    }
}
```

Even here there are several modifications available: material (which governs some properties such as being able to be pushed by pistons and whether it can be broken by hand), hardness (how long it takes to break), harvest level (appropriate tool and tool material: in this case wooden pickaxe), resistance (vs. explosions), the tab it shows up in the creative menu, and what step sound it has.

This is where any fancy functionality blocks will have will need to go, but for now we're making a block that just looks nice, so we're done.

The Block Model JSON

Next we need to tell Minecraft what we want our block to look like.

```
{
    "parent": "block/cube_all",
    "textures": {
        "all": "example:blocks/decorative"
    }
}
```

That's pretty much all that's needed for it to work once the block is registered. The only important thing is that the **filename** match the **registry name** used to register the block and should be in all lowercase (1.11+ file names are *required* to be lowercase, prior to that it is just case sensitive).

Name the model JSON file `my_block.json` (matching the registry name we're going to give it later) and save it at `src\main\resources\assets\example\models\block\` (where `example` is the mod ID specified in the @Mod annotation of your main mod class).

The block model here uses a parent of block/cube_all, which means that the single supplied texture will be used on all faces. There are other default models as well, such as:

- block/cube (all six faces assigned independently)
- block/cube_bottom_top (top and bottom faces independent from the sides)
- block/orientable (directional facing block, eg. furnace)
- block/cross (flowers, tall grass)
- block/crop (wheat, carrots)

Do note that each model specifies the textures it uses by a name ID (e.g. `"all"` or `"top"`). Look at the parent model to determine what those names are if you are uncertain. Incorrectly specified textures may lead to *non-error-reporting* missing texture issues.

It is also possible to create a wholly custom model or to create a custom parent model. But for now, this will be enough.

Don't forget to create a texture, name it `decorative.png` (as that's what the JSON file specified) and save it to `src\main\resources\assets\example\textures\blocks\`

Block Registration

Registering blocks is done from your main mod class, or a ModBlocks class method invoked from the main mod class during preInit.

```
Block myBlock = new CustomBlock();
string registryname = "my_block";
block.setRegistryName(registryname);
block.setUnlocalizedName(block.getRegistryName().toString());
GameRegistry.register(block);
```

There is an important reason to use `block.setUnlocalizedName(block.getRegistryName().toString());` as well! It insures that your block (and item) unlocalized names contain the mod ID to avoid language file conflicts between mods.

Oh you want an item version too so it can exist in your inventory too? That's created separately post 1.7.10 and done like so:

```
ItemBlock ib = new ItemBlock(block);
ib.setRegistryName(registryname);
GameRegistry.register(ib);
```

Notice that we set the ItemBlock's registry name to the same string as our block. This is how Forge and match blocks to their ItemBlock counterpart and vice versa.

But wait, there's more!

Your block might have an *item form*, but that item doesn't have a model or texture yet! Models are automatically registered for blocks, but not items. This *may only be called from the Client Proxy* and does not cover blocks with variants (such as wool or leaves).

```
ModelLoader.setCustomModelResourceLocation(
    ib , 0, new ModelResourceLocation(ib.getRegistryName(),"normal"));
```

Generally speaking you will not also need an Item Model JSON as Forge and vanilla will fall back on the block's model instead, however this is not always the case. If you do find that you need an Item model JSON, just parent it to your block JSON and save it to `src\main\resources\assets\example\models\item\` with the same file name as the block's registry name.

```
{
    "parent": "example:block/my_block"
}
```

Read Creating a basic block with Forge online: https://riptutorial.com/minecraft/topic/9748/creating-a-basic-block-with-forge

Chapter 3: Creating a Basic Item with Forge

Introduction

Creating a simple item with Forge is one of the first tasks an aspiring modder will have to learn. How to do that has changed over the various versions of Minecraft and is probably at a "moderate" difficulty post 1.7.10 due to the sheer number of easy to make mistakes, particularly with making it render properly.

Remarks

If something goes wrong and your custom item has a missing texture (black/purple) or model (default cube that's too large when held) check the log. Problems of this sort will almost *always* show up in the log, provided that you have registered things correctly.

The first thing that will show up is `"Exception loading model for variant..."` line or similar, telling you which block or item failed to load properly. After a dozen or so lines beginning with `at...` will be another line starting `Caused by...`

This "Caused By" line is the important one, it will tell you which file failed to load properly and can be one of several errors, such as:

- Malformed JSON (your JSON file isn't valid and has a typo)
- File Not Found (the file Minecraft is looking for is not properly named or in the right place)
- Missing Variant Exception (your Blockstate JSON is incomplete)

You may even get more than one failing error for a single mistake! If you don't know which item is the problem, comment out every block and item you know works, reduce the error list down to the block (or blocks) in question. You may have to do this several times.

Additionally, a mixing texture shows up as differently as a simple list all of the missing resources for a given domain (mod ID).

Examples

Item Class

This part hasn't changed over the versions of Minecraft a whole lot, although there have been some mutations in the exact method signatures as well as class hierarchy. A basic item (one that has no functionality, such as a stick or ingot: that's right, both are do-nothing items!)

```
public class CustomItem extends Item {

    public CustomItem () {
        super();
        this.setMaxDamage(0);
        this.setCreativeTab(CreativeTabs.MISC);
    }
}
```

Not much room for customization at this point, unlike blocks. About all we can do is change whether or not the item can take damage (tools use this) and what creative tab it will exist in. Name and texture we'll handle when we register the item with the GameRegistry.

However, this is all that is needed in order to hold, carry, drop, craft, smelt and otherwise utilize the item. Some items in Minecraft (such as sticks) don't even have a unique class and simply use `new Item()`. We could do that here, however any item with additional functionality will need a class.

Item Model

As with blocks, items need models too.

```
{
    "parent": "item/generated",
    "textures": {
        "layer0": "example:items/basic"
    }
}
```

That's pretty much all that's needed for it to work once the item is registered. The only important thing is that the filename match the registry name used to register the block and should be in all lowercase (1.11+ file names are required to be lowercase, prior to that it is just case sensitive).

Note that "layer0" is the only texture needed and it is highly unlikely that any other texture will be specified at all (although some items like potions and leather armor do have a "layer1"). All names are defined by `item/builtin` (the internal top-most parent model for items) unlike with blocks.

Name the model JSON file `my_item.json` (matching the registry name we're going to give it later) and save it at `src\main\resources\assets\example\models\item\` (where `example` is the mod ID specified in the @Mod annotation of your main mod class).

Additionally create a texture for your item, name it `basic.png` and save it to `src\main\resources\assets\example\textures\items\`

The item model here uses a parent of item/generated, which means that the single supplied texture will be used (as with most non-block items) and will be held in the player's hand in a default orientation. There is also item/handheld which specifies different display orientations (for tools). Items may also supply their own "display" attribute, overriding those from the parent, but is not needed in 99.9% of uses.

Item Registration

Registering items is done from your main mod class, or a ModItems class method invoked from the main mod class during preInit.

```
Item item = new CustomItem();
```

```
string registryname = "my_item";
item.setRegistryName(registryname);
item.setUnlocalizedName(item.getRegistryName().toString());
GameRegistry.register(item);
```

There is an important reason to use `item.setUnlocalizedName(item.getRegistryName().toString());` as well! It insures that your item's unlocalized name contain the mod ID to avoid language file conflicts between mods.

Now the item needs a model too, and this is where things got difficult post 1.7.10, which just involved telling Minecraft the name of the texture to load and could be specified in the item's constructor.

```
final ModelResourceLocation fullModelLocation = new
ModelResourceLocation(item.getRegistryName().toString(), "inventory");
ModelBakery.registerItemVariants(item, fullModelLocation);
ModelLoader.setCustomMeshDefinition(item, new ItemMeshDefinition()
    {
        public ModelResourceLocation getModelLocation(ItemStack stack)
        {
            return fullModelLocation;
        }
    });
```

Note that this section *must* be located client-side only (i.e. the client proxy) as many of the referenced classes do not exist on the dedicated server.

Registering items with *variants* e.g. Saplings, has to be done a different way, using `ModelLoader.setCustomModelResourceLocation(item, metadata, resourceLocation)` although it lets us use a Blockstate file to specify our variants (which is *much* preferred to the alternative). Our item doesn't use variants, so we're done.

Read Creating a Basic Item with Forge online: https://riptutorial.com/minecraft/topic/9850/creating-a-basic-item-with-forge

Chapter 4: Creating first Spigot plugin

Examples

First Plugin in Eclipse

Prerequisite

This guide assumes you have already used BuildTools and have run the Spigot server at least once. It also assumes that you have the Spigot-API jar file which we will use.

1) Start Eclipse; you may change the workspace location if desired.

2) Create a new Project

1. Set the project name to whatever you wish. Here, we chose MyFirstPlugin.
2. Click next.
3. Select Add External JARs under the Libraries tab. In the JAR Selection dialogue box, select the spigot-api-shaded jar file, which can be found in Spigot/Spigot-API/target/ inside your BuildTools folder.
4. Select Finish

3) Add a new package

Right click on **src** and click on **New > Package**. You may use any namespace convention you wish, just be consistent. (e.g: com.google.android).

4) Create a new class

1. Right-click on the newly created package and select **New > Class**.

2. Give it any name; often the same name as the project. Inside the editor, the newly created Java class will open. The code should look somewhat like this:

```
package yourpackage;
public class MyFirstPlugin {
}
```

5) Modify class declaration

1. Your class must extend from JavaPlugin. Eclipse will produce an error as it does not know what JavaPlugin is. If you have successfully imported the Spigot-API, you will be able to import JavaPlugin by adding the import statement. You do not need to manually type that line, simply click the error and select the appropriate action. Your code should now look like:

```
package yourpackage;
```

```
import org.bukkit.plugin.java.JavaPlugin;

public class MyFirstPlugin extends JavaPlugin {

}
```

6) Implement the necessary methods

The JavaPlugin class has some abstract methods which must be implemented by your plugin. Hence, add the onEnable and onDisable functions which will be triggered when the plugin is disabled or enabled in the console. You can leave these blank for now. You are also required to write `@Override` above the method.

Note: You do not need to add a getLogger when your plugin is enabled or disabled, Bukkit already does that for you.

```
package com.meeku.tutorialPlugin;
import org.bukkit.plugin.java.JavaPlugin;

public class MyFirstPlugin extends JavaPlugin {
    // Fired when plugin is enabled
    @Override
    public void onEnable() {
    }
    // Fired when plugin is disabled
    @Override
    public void onDisable() {

    }
}
```

7) Create plugin.yml file

Right click the project and create a file **New > File**. Name it **plugin.yml**. Paste in the following:

```
name: MyFirstPlugin
main: yourpackage.MyFirstPlugin
version: 1.0
commands:
```

8) Export

Since there are no errors, we can export this project as a JAR. Right-click on the project name, select Export. In the consequential dialogue box, select JAR file. Click Next. You can uncheck the classpath and project include and change the export destination to your plugins folder

9) Running

Start the server and you should see that your plugin was enabled.

Read Creating first Spigot plugin online: https://riptutorial.com/minecraft/topic/9766/creating-first-spigot-plugin

Chapter 5: Event Listeners in Bukkit

Examples

Creating an event listener

To register your methods, the class containing the EventHandler(s) must implement the Listener interface.

```
import org.bukkit.event.Listener;

public final class ListenerClass implements Listener {
}
```

You need to register the event listener by adding the following call to your onEnable method in the class that extends JavaPlugin:

```
getServer().getPluginManager().registerEvents(new ListenerClass(), this);
```

To listen to any given event in your listener class, you must create a method with @EventHandler annotation on the method. The event type is specified by the Type in the method's only argument. The method may be named whatever you wish.

```
import org.bukkit.event.Listener;
import org.bukkit.event.EventHandler;
import org.bukkit.event.player.PlayerLoginEvent;

public class ListenerClass implements Listener {
    @EventHandler
    public void onPlayerLogin(PlayerLoginEvent event) {
        event.getPlayer().sendMessage("Welcome to the server!");
    }
}
```

EventHandler Parameters

The `org.bukkit.event.EventHandler` annotation accepts a couple parameters.

priority - Indicates the priority of your listener. There are the six different priorities, in order of execution: LOWEST,LOW,NORMAL[default],HIGH,HIGHEST,MONITOR. These constants refer to the `org.bukkit.event.EventPriority` enum.

If you want to change the outcome of an event, choose very carefully from LOWEST to HIGHEST. Suggested generalized protection plugins on LOWEST, more specific plugins on NORMAL, and override plugins on HIGH. If you want to act when an event happens, but not change the outcome, use MONITOR.

Note: The MONITOR priority should only be used for reading only. This priority is useful for

logging plugins to see the results of an event and modifying values may interfere with those types of plugins.

ignoreCancelled - A boolean which indicates whether or not your listener should fire if the event has been cancelled before it is the listener's turn to handle the event. False by default.

```
import org.bukkit.event.Listener;
import org.bukkit.event.EventHandler;
import org.bukkit.event.EventPriority;
import org.bukkit.event.player.PlayerLoginEvent;

public final class LoginListener implements Listener {
    @EventHandler
    public void normalLogin(PlayerLoginEvent event) {
        // Some code here
    }

    @EventHandler(priority = EventPriority.HIGH)
    public void highLogin(PlayerLoginEvent event) {
        // Some code here
    }
}
```

Creating Custom Events

Sometimes you need to create your own Event, one that other plugins can listen to (Vault, among other plugins, does this) and even cancel. Bukkit's Event API allows this to be possible. All you need to do is make a new class, have it extend `Event`, add the handlers and the attributes your event needs (like Player or message).

```
import org.bukkit.event.Event;
import org.bukkit.event.HandlerList;

public final class CustomEvent extends Event {
    private static final HandlerList handlers = new HandlerList();
    private String message;

    public CustomEvent(String example) {
        message = example;
    }

    public String getMessage() {
        return message;
    }

    public HandlerList getHandlers() {
        return handlers;
    }

    public static HandlerList getHandlerList() {
        return handlers;
    }
}
```

Calling your Custom Event

You are in control of creating and calling your events, where you call it is completely up to you. Here's an example

```
// Create the event here
CustomEvent event = new CustomEvent("Sample Message");
// Call the event
Bukkit.getServer().getPluginManager().callEvent(event);
Bukkit.getServer().broadcastMessage(event.getMessage());
```

Remember: You are in control of your events. If you don't call it, and act upon it, it doesn't happen!

Listening to a Custom Event

Listening to a custom event is the same as listening to a normal event.

```
import org.bukkit.event.Listener;
import org.bukkit.event.EventHandler;

public final class CustomListener implements Listener {

    @EventHandler
    public void onCustomEvent(CustomEvent event) {
    // Some code here
    }
}
```

Making your CustomEvent Cancellable

If you ever want to make your event cancellable, just add `implements Cancellable`, `boolean cancelled` and a getter and setter:

```
import org.bukkit.event.Event;
import org.bukkit.event.HandlerList;
import org.bukkit.event.Cancellable;

public final class CustomEvent extends Event implements Cancellable {
    private static final HandlerList handlers = new HandlerList();
    private String message;
    private boolean cancelled;

    public CustomEvent(String example) {
        message = example;
    }

    public String getMessage() {
        return message;
    }
    public boolean isCancelled() {
        return cancelled;
    }

    public void setCancelled(boolean cancel) {
        cancelled = cancel;
    }

    public HandlerList getHandlers() {
        return handlers;
    }

    public static HandlerList getHandlerList() {
        return handlers;
    }
}
```

Afterwards, you would check if a plugin had cancelled the custom event.

```
// Create the event here
CustomEvent event = new CustomEvent("Sample Message");
// Call the event
Bukkit.getServer().getPluginManager().callEvent(event);
// Check if the event is not cancelled
if (!event.isCancelled()) {
    Bukkit.getServer().broadcastMessage(event.getMessage());
}
```

Un-registering events or listeners

You can un-register individual events, entire listener classes or all events registered by your plugin or even by other plugins!

Un-register specific event

Each event class has the getHandlerList() static method, call that and then you can use .unregister() method.

```
PlayerInteractEvent.getHandlerList().unregister(plugin);
// this will unregister all PlayerInteractEvent instances from the plugin
// you can also specify a listener class instead of plugin.
```

Now you know why you'll need the getHandlerList() in your custom events.

Un-register all events

Using the HandlerList class and its unregisterAll() static method you can easily unregister events from listener classes or plugins.

```
HandlerList.unregisterAll(plugin);
```

```
// this will unregister all events from the specified plugin
// you can also specify a listener class instead of plugin.
```

Read Event Listeners in Bukkit online: https://riptutorial.com/minecraft/topic/9739/event-listeners-in-bukkit

Chapter 6: Event Listeners in Forge

Examples

Creating an event listener in Forge

Creating an event listener in Forge is very similar to creating one in Bukket.

Creating the listener class requires a lot less. There's no interface to implement or other imports.

```
public class ListenerClass { } //perfectly valid event listener class
```

Registering it requires passing the instance to the Forge event bus:

```
MinecraftForge.EVENT_BUS.register(new ListenerClass());
```

There are a couple of different event buses depending on the event. For example, ore generation events are fired on the `ORE_GEN_BUS`. You can call this registration from anywhere, but it is advised to call it from either your main mod class (with the @Mod annotation) or from a proxy class (some events are client side only and a client side event handler must only be called from the client proxy, otherwise the dedicated server will crash!)

To listen to any given event in your listener class, you must create a method with @SubscribeEvent annotation on the method. The event type is specified by the Type in the method's only argument. The method may be named whatever you wish.

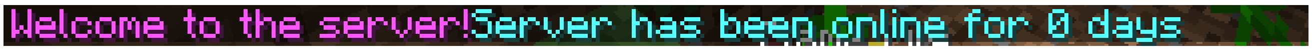
Note that some event types are subtypes (which should be referenced by their enclosing type, e.g. `CropGrowEvent.Pre`) and that some events may have a Phase as it is fired in more than one place (such as all `TickEvent`s which are fired both before and after all vanilla code). As a modder you should always check for both of these things and only run your code when needed.

```
public class ListenerClass {
    @SubscribeEvent
    public void onPlayerLogin(PlayerLoggedInEvent event) {
        event.player.addChatMessage(new TextComponentString("Welcome to the server!"));
    }
}
```

As Forge mods interact directly with Minecraft internals, a lot of power is given to the modder to be able to affect things, but similarly, the code must follow the vanilla framework: there's no shortcuts for sending messages, the message has to be constructed from ITextComponents manually, but the ability to manipulate these objects (such as applying color formatting) is much easier. For example:

```
TextComponentString txt = new TextComponentString(
    TextFormatting.LIGHT_PURPLE + "Welcome to the server!");
txt.appendSibling(new TextComponentString(
    TextFormatting.AQUA + "Server has been online for " + x + " days"));
event.player.addChatMessage(txt);
```

Which gives the following result:


Welcome to the server!Server has been online for 0 days

Read Event Listeners in Forge online: https://riptutorial.com/minecraft/topic/9744/event-listeners-in-forge

Chapter 7: Installing a Spigot server

Examples

BuildTools

What is it?

BuildTools.jar is a solution to building Bukkit, CraftBukkit, Spigot, and the Spigot-API. All of which is done on your computer! A few prerequisite programs are necessary, but the instructions below will guide you through everything you need to do.

Prerequisites

There are two applications necessary to use BuildTools: Git and Java.

Windows

Git

In order for BuildTools to run on Windows, you will need to install Git. For Windows it is distributed via git-scm, which can be downloaded here. Install it where you like, it will provide git bash, which will be used to run the BuildTools jar. Just keep hitting next when running the installer.

Java

Download JRE 8 from here and install. Just keep hitting next when running the installer.

Linux

Both git and Java, as well as util commands, can be installed using a single command via your package manager.

Debian/Ubuntu: `sudo apt-get install git openjdk-7-jre-headless tar`

CentOS/RHEL: `sudo dnf install git java-1.7.0-openjdk-devel tar`

Arch: `pacman -S jdk8-openjdk git`

Mac

Git can be downloaded from: http://sourceforge.net/projects/git-osx-installer/files/

Java may need to be updated from the Apple distributed version, and even if previously updated, may need to be linked for shell use. Please follow steps found here: https://gist.github.com/johan/10590467

Running BuildTools

1. Download BuildTools.jar from https://hub.spigotmc.org/jenkins/job/BuildTools/lastSuccessfulBuild/artifact/target/BuildTools.jar .

2. Open your terminal if you are on Linux, or git bash on Windows.

    1. Git bash can be found on the desktop or in the Start menu under the name "git bash". It's also possible to open it by right-clicking on anything, as it is now an item in your context menu.

3. Navigate to where you downloaded BuildTools.jar, or use the command line way to download the jar to your current directory.

    1. On Windows, you can either use the cd command to change directories, or you can right click the blank space of the folder where BuildTools.jar is (DO NOT click BuildTools.jar itself) and click "git bash", which will open it in your current directory.

4. Run BuildTools.jar from the terminal (Do not double-click BuildTools.jar) by doing the following:

    1. On Linux run git config --global --unset core.autocrlf, then run java -jar BuildTools.jar in bash or another appropriate shell.
    2. On Windows run the below command inside the git bash window that opened: java -jar BuildTools.jar Please be aware that it is required that you have BuildTools #35 or later, older versions will not work.
    3. On Mac run the below commands, export MAVEN_OPTS="-Xmx2G" java -Xmx2G -jar BuildTools.jar

5. Wait as it builds your jars. In a few minutes you should have freshly compiled jars!

6. You can find CraftBukkit and Spigot in the same directory you ran the the BuildTools.jar in (craftbukkit-1.10.jar and spigot-1.10.jar). You can find Spigot-API in \Spigot\Spigot-API\target\ (spigot-api-1.10-R0.1-SNAPSHOT.jar).

Spigot Installation

Windows

1. Get spigot.jar using BuildTools or from here.
2. Paste the following text into a text document. Save it as start.bat in the same directory as spigot.jar: You will need to rename your jar to spigot.jar, or modify the file in the bat file to point to the correct file. nb: Windows (by default) will hide the .jar extension of the file.

```
@echo off
java -Xmx1G -jar spigot.jar
pause
```

3. Double click the batch file.

Linux

1. Get spigot.jar using BuildTools or from here.
2. Create a new startup script (start.sh) in the directory to launch the the JAR: #!/bin/sh

```
java -Xmx1G -jar spigot.jar
```

3. Open your terminal and execute the following in the directory: chmod +x start.sh
4. Run your start up script:

```
./start.sh
```

Mac

1. Get spigot.jar using BuildTools or from here.
2. Create a new startup script (start.command) to launch the JAR: #!/bin/sh

```
cd "$( dirname "$0" )"
java -Xmx1G -jar spigot.jar
```

3. Open Terminal and type into it: (Don't hit enter!)

```
chmod a+x
```

4. Drag your startup script file into the Terminal window. (Be sure to put a space between chmod a+x and your startup script!)
5. Double click your startup script.

Read Installing a Spigot server online: https://riptutorial.com/minecraft/topic/9702/installing-a-spigot-server

Chapter 8: Modding with Forge

Syntax

- MODID = represents the Identifier of the MOD
- MODPath = stands for the full qualified directory path to your mod folder

Remarks

This topic should contain most used patterns/examples and well tested code for modding the Minecraft application with forge. Maybe this can replace the official documentation one day.

Examples

Implementation pattern for Initialization Proxies

This example shows you how to implement proxy classes for your Minecraft Mod Application, which are used to initialize your mod.

First of all you will need to implement the base CommonProxy.java class which contains the 3 mainly used method:

```
public class CommonProxy {
    public void preInit(FMLPreInitializationEvent e) {}
    public void init(FMLInitializationEvent e) {}
    public void postInit(FMLPostInitializationEvent e) {}
}
```

Normally your mod has 2 different packages for Client and Server Code, so you will need in each package a child class of CommonProxy.java like:

```
public class ClientProxy extends CommonProxy {
    @Override
    public void preInit(FMLPreInitializationEvent e) {
        super.preInit(e);
    }

    @Override
    public void init(FMLInitializationEvent e) {
        super.init(e);
    }

    @Override
    public void postInit(FMLPostInitializationEvent e) {
        super.postInit(e);
    }
}
```

and for the server:

```
public class ServerProxy extends CommonProxy {
    @Override
    public void preInit(FMLPreInitializationEvent e) {
        super.preInit(e);
    }

    @Override
    public void init(FMLInitializationEvent e) {
        super.init(e);
    }

    @Override
    public void postInit(FMLPostInitializationEvent e) {
        super.postInit(e);
    }
}
```

After you have created this classes you are able to extend them by methods which have to run only on client or server side, but can also attach them to both if you call the methods in the 'base' class.

Lastly you have to define which proxy is taken at runtime. You have to extend your main mod class with the `@Mod` annotation, by:

```
private static final String CLIENTPROXY = "com.yourpackage.client.ClientProxy";
private static final String SERVERPROXY = "com.yourpackage.client.ServerProxy";

@SidedProxy(clientSide = CLIENTPROXY, serverSide = SERVERPROXY)
public static CommonProxy PROXY;
```

This will enable Forge to detect which class should be taken at runtime. In the initialization methods of your Mod you can now use this static PROXY property.

```
@EventHandler
public void init(FMLInitializationEvent event) {
    PROXY.init(event);
}

@Mod.EventHandler
public void preInit(FMLPreInitializationEvent event) {
    PROXY.preInit(event);
}

@EventHandler
public void postInit(FMLPostInitializationEvent event) {
    PROXY.postInit(event);
}
```

Adding custom sounds to your MOD

This example show you how you add new sounds to your MOD and play them. First of all you need a sound file which has the format `*.ogg`. Any other format is not allowed by the Minecraft application and will be rejected.

The soundfile has the name: sound1.ogg

Put the sound file under the following path:

> /YourPath/src/main/resources/assets/MODID/sounds/sound1.ogg

Replace 'MODID' by the identifier you defined for your MOD

Next you have to create a `sounds.json` in UTF-8 (Standard) encoding which defines name, resource,... and other things for your custom sound. This file will look like:

```
{
  "sound1": {
    "category" : "player",
    "sounds": [{
        "name": "MODID:sound1",
        "stream": false
      }
    ]
  },
  "sound2": {
    "category" : "ambient",
    "sounds": [{
        "name": "MODID:subfolder/sound2",
        "stream": true
      }
    ]
  }
}
```

As explanation for this sounds.json.

There are defined 2 sounds defined, as I added an example that you can investigate how to add multiply sounds. `sound1` has the category `player` the second is of category `ambient` which means the volume of the sound is affected by the volume settings the user has set for player/ambient sounds. `name` is the most important property as it is pointing to the resource of the sound. The `MODID` is the identifier of your MOD and is mandatory as the Application will search in the resources of your mod for the file, otherwise it will search in the Minecraft resources and will find nothing. The `stream` property means that the sound will be streamed from file system, which is only needed for sounds longer than 4 seconds.

Your custom `sounds.json` file has to go under the following path:

> /YourPath/src/main/resources/assets/MODID/sounds.json

Now you will be able to load the sounds to the registry of the game. So you have to create a class which is initializing `SoundEvent`s and handling the registration.

```
public class SoundRegistrator {
    public static final SoundEvent SOUND_1;
    public static final SoundEvent SOUND_2;

    static {
        SOUND_1 = addSoundsToRegistry("sound1");
        SOUND_2 = addSoundsToRegistry("sound2");
    }

    private static SoundEvent addSoundsToRegistry(String soundId) {
        ResourceLocation shotSoundLocation = new ResourceLocation("MODID", soundId);
        SoundEvent soundEvent = new SoundEvent(shotSoundLocation);
        soundEvent.setRegistryName(shotSoundLocation);
        return soundEvent;
    }
}
```

Afterward you have to create a `SoundRegisterListener`:

```
public class SoundRegisterListener {
        @SubscribeEvent(priority = EventPriority.NORMAL, receiveCanceled = true)
        public void registerSoundEvents(RegistryEvent.Register<SoundEvent> event) {
           event.getRegistry().registerAll(SoundRegistrator.SOUND_1,SoundRegistrator.SOUND_2);
        }
}
```

and register it to the `MinecraftForge.EVENT_BUS` like:

```
MinecraftForge.EVENT_BUS.register(new SoundRegisterListener());
```

Finally you will be able to play your sounds:

```
void playSound(SoundEvent sound) {
    try {
        if (Minecraft.getMinecraft().world.isRemote) {
            EntityPlayerSP player = Minecraft.getMinecraft().player;
            Minecraft.getMinecraft().world.playSound(player, player.getPosition(), sound,
SoundCategory.PLAYERS, RandomGenerator.getNextRandomVolumeLoud(), 1.0F);
        }
    } catch (Exception ex) {
       //Error happened
    }
}
```

Sending a command

This example shows you different ways to execute 'commands' for Minecraft from code:

```
EntityPlayerSP player = Minecraft.getMinecraft().player;
player.sendChatMessage("/Command here");
```

to send a command in SinglePlayer

Read Modding with Forge online: https://riptutorial.com/minecraft/topic/9956/modding-with-forge

Chapter 9: Plugin Commands

Examples

Hello Command

In the code below you can see how to add a command to your plugin.

MainClass.java

```java
package yourpackage;

import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.plugin.java.JavaPlugin;

public class MainClass extends JavaPlugin {

@Override
public boolean onCommand(CommandSender sender, Command command, String label, String[] args) {
    if (command.getName().equalsIgnoreCase("hello")) {
        sender.sendMessage("Hey!");
    }
    return false;
}
}
```

Plugin.yml

```yaml
name: HelloCommand
main: yourpackage.MainClass
version: 1.0
commands:
  hello:
    description: Hello
```

Read Plugin Commands online: https://riptutorial.com/minecraft/topic/9708/plugin-commands

Chapter 10: Writing Minecraft Mods

Introduction

Understanding writing your own mods for Minecraft.

The first thing to know is that there are primarily two platforms which mods are build on. Forge and Bukkit.

Each platform has its advantages and disadvantages and are generally not compatible with each other. Forge mods run the gamut and are generally game-play oriented. Bukkit mods are entirely server-based and generally admin tool oriented. Bukkit mods are called plugins.

Examples

Basic Bukkit Plugins

- Writing a Hello command
- Writing an Event Listener

Basic Forge Mods

- Creating a basic block
- Creating a basic item
- Writing a Hello command
- Writing an Event Handler
- Getting started with Capabilities [1.8+]

Read Writing Minecraft Mods online: https://riptutorial.com/minecraft/topic/9740/writing-minecraft-mods

Credits

S. No	Chapters	Contributors
1	Getting started with minecraft	Community, Martin W, SoniEx2
2	Creating a basic block with Forge	Draco18s
3	Creating a Basic Item with Forge	Draco18s
4	Creating first Spigot plugin	Martin W
5	Event Listeners in Bukkit	Draco18s, Martin W
6	Event Listeners in Forge	Draco18s, Martin W
7	Installing a Spigot server	Martin W
8	Modding with Forge	Code.IT
9	Plugin Commands	Martin W
10	Writing Minecraft Mods	Draco18s, Martin W