FREE eBook

LEARNING bukkit

Free unaffiliated eBook created from
Stack Overflow contributors.

#bukkit

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: bukkit

It is an unofficial and free bukkit ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official bukkit.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with bukkit

Remarks

Bukkit is a simple API that allows for modifying the normal Minecraft multiplayer experience using plugins.

Bukkit is now discontinued, and is no longer available for newer versions of Minecraft. Spigot, a version of Bukkit that boasts improving server performance is available. The API for Spigot is essentially the same as Bukkit.

Versions

Minecraft Version	Spigot Download Link	Release Date
1.10.2	Link	2016-11-03
1.10	Link	2016-06-26
1.9.4	Link	2016-06-09
1.9.2	Link	2016-03-30
1.9	Link	2016-02-29
1.8.8	Link	2015-07-28
1.8.7	Link	2015-06-05
1.8.6	Link	2015-05-25
1.8.5	Link	2015-05-22
1.8.4	Link	2015-04-17
1.8.3	Link	2015-02-20
1.8	Link	2014-09-02
1.7.10	Link	2014-06-26
1.7.9	Link	2014-04-14
1.7.8	--	2014-04-11
1.7.5	Link	2014-02-26
1.7.2	Link	2013-10-25

Minecraft Version	Spigot Download Link	Release Date
1.6.4	Link	2013-09-19
1.6.2	Link	2013-07-08
1.5.2	Link	2013-05-02
1.5.1	Link	2013-03-21
1.4.7	Link	2013-01-09
1.4.6	--	2012-12-20

Examples

Creating a Plugin

Prerequisites

- JDK 7 or Higher (Recommended: JDK 8+)

Adding Bukkit as a Dependency

The simplest method to add the Bukkit API to your project is to download the Bukkit.jar directly from the Spigot Repository and add it to your project's classpath. Legacy versions of Bukkit can be found at the Bukkit Repository.

The other is to add it as a Maven dependency, by adding the following lines to your `pom.xml`:

```
<repositories>
    <repository>
        <id>spigot-repo</id>
        <url>https://hub.spigotmc.org/nexus/content/repositories/snapshots/</url>
    </repository>
</repositories>
<dependencies>
    <!--Bukkit API-->
    <dependency>
        <groupId>org.bukkit</groupId>
        <artifactId>bukkit</artifactId>
        <version>{VERSION}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Main Class

The plugin's main class is the entry point for Bukkit to load an interact with your plugin. It is a class that extends `JavaPlugin` and only one instance of it should be created by your plugin. By convention it is good to give this class the same name as your plugin.

Here is an example of a main plugin class for the plugin "MyPlugin":

```
package com.example.myplugin; //{$TopLevelDomain}.{$Domain}.{$PluginName};

import org.bukkit.plugin.java.JavaPlugin;

public final class MyPlugin extends JavaPlugin {

    @Override
    public void onEnable() {
        //Called when the plugin is enabled
        getLogger().info("onEnable has been invoked!");
    }

    @Override
    public void onDisable() {
        //Called when the plugin is disabled
        getLogger().info("onDisable has been invoked!");
    }

}
```

To access your plugin instance from another class, you'll need to store the instance of your MyPlugin class created by Bukkit so that is accessible from outside of the class.

```
public class MyPlugin extends JavaPlugin {

    private static MyPlugin instance; //Effectively final variable containing your plugin's
instance

    public MyPlugin(){
        if(MyPlugin.instance != null) { //Unnecessary check but ensures your plugin is only
initialized once.
            throw new Error("Plugin already initialized!");
        }

        MyPlugin.instance = this; //A plugin's constructor should only be called once
    }

    public static MyPlugin getInstance(){ //Get's your plugin's instance
        return instance;
    }

    //your other code...
}
```

Then, to access your main class from another class, simply use `MyPlugin.getInstance()`

```
public class MyOtherClass {

    public void doSomethingWithMainClass(){
        MyPlugin.getInstance().getLogger().info("We just used MyPlugin");
    }

}
```

Creating a plugin.yml

The plugin.yml file goes in the root of your final jar file and provides essential information to Bukkit to load your plugin. The most simple plugin.yml looks like this

```
name: {$PluginName}               //The name of the plugin
main: {$PackageName}.{$MainClass} //The fully qualified name of the main class.
version: {$Version}               //The plugin's version
```

For example with the above MyPlugin class

```
name: MyPlugin
main: com.example.myplugin.MyPlugin
version: 1.0
```

Create a test server on Windows

To be able to create a server, you need to have the spigot or the bukkit jar file. Reefer to the versions topic to select your jar

1. First, create a new folder. In that folder, put the spigot/bukkit jar file.
2. Right click in the folder, and choose New > Text Document.
3. Name the new document start.bat, right click on it and click Edit.
4. Add the following code:

```
@echo off
java -Xms512M -Xmx1G -XX:+UseConcMarkSweepGC -jar {YOUR_JAR.jar}
pause
```

Don't forget to change {YOUR_JAR.jar} for the jar you downloaded before starting these topics.

5. You can edit `-Xms` to change the minimum allowed RAM (Ex: -Xms1024M = 1024MB, -Xms1G = 1GB). You can also edit `-Xmx` to change the maximum allowed RAM. Make sure the maximum is bigger than the minimum.
6. Save the file, close the window and start your `start.bat` file. Your server should now open. To run the server, you must accept the EULA.
7. If you agree to the EULA, open `eula.txt` change `eula=false` to `eula=true` Click "Save" and then you should now be able to start your server.
8. To connect to your server, run the `start.bat`, open Minecraft, add a server and put `localhost`

as the IP.

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
    4. If you need older version, you can specify the version using --rev argument to BuildTools, for example for 1.8.8: java -jar BuildTools.jar --rev 1.8.8

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 (for minecraft version 1.10, they would be craftbukkit-1.10.jar and spigot-1.10.jar). You can find Spigot-API in \Spigot\Spigot-API\target\ (for minecraft version 1.10, it would be spigot-api-1.10-R0.1-SNAPSHOT.jar).

Read Getting started with bukkit online: https://riptutorial.com/bukkit/topic/5400/getting-started-with-bukkit

Chapter 2: Commands

Syntax

- `@Override public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args)`

Examples

Handling a Command

To handle a command, you must have a class that implements the CommandExecutor interface. The JavaPlugin class (your plugin's main class) already implements this.

When implementing the CommandExecutor interface, the following method must be implemented:

```
public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args) {
//Handle your command in here
return true;      ///Should return false if you want to show the usage

}
```

Sender is the one who sent the command. It can be a Player or the Console.

CMD is the command you're listening to, as declared in plugin.yml. Not to be confused with label.

label is the alias used to execute this command, it's what the sender types after the slash.

and finally, args are the arguments the sender may have used to send your command.

A possible command might go as

    /tell Kerooker Hi, Kerooker!

Tell would be your label, and may also be defined as your command if you said so in plugin.yml;

'Kerooker', 'Hi,' , 'Kerooker!' are your args 0, 1 and 2, respectively

As a return, you will probably always want to return true when you expected everything to happen that way. You should return false if you want to show the sender the the command usage defined in your plugin.yml

A simple set GameMode command (/gm)

This example shows a very basic example of how to utilize onCommand. I don't suggest processing your commands directly in onCommand, but this does the trick for this simple case.

In this example we attempt to set the player's gamemode.

The first thing we need to do is make sure that the sender isn't a ConsoleCommandSender, because we can't set a console's gamemode. This is done with (sender instanceof Player).

Next, we want the player to type /gm CREATIVE (or what ever other gamemode) so we have to check 2 things:

1. make sure that they pass in 1 argument (CREATIVE)
2. make sure that their command was "gm"

We accomplished these checks with: args.length == 1 && label.equalsIgnoreCase("gm")

Now we know for sure the player typed "/gm x".

The next thing we need to do is turn args[0] into a GameMode object so that we can apply it to the player. This can be done with GameMode.valueOf(String) However, according to the Java enumeration documentation, if a string is passed into valueOf() that doesn't match an enumeration, it will throw an IllegalArgumentException - so we have to make sure to catch that.

Once we have the gamemode, we can go ahead and simply use p.setGameMode(gm) and the player's gamemode will change. In the case that we caught an exception - we simply print out a statement and return false.

```
@Override
public boolean onCommand(CommandSender sender, Command command, String label, String[] args) {
    if (sender instanceof Player) {
        final Player p = (Player) sender;

        if (args.length == 1 && label.equalsIgnoreCase("gm")) {
            try {
                GameMode gm = GameMode.valueOf(args[0]);
                p.setGameMode(gm);
                p.sendMessage(ChatColor.GREEN + "Your gamemode has been set to: " +
gm.toString());
                return true;
            } catch (IllegalArgumentException e) {
                p.sendMessage(ChatColor.RED + "Invalid gamemode option!");
                return false;
            }

        }
    }
    return false;
}
```

Command not in main class

If you have a lot of commands, you shouldn't put them all in the main class.

1. Make a new class and have it implement `CommandExecutor`

2. Add the following to the class:

```
@Override
```

```
 public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args)
{

 }
```

3. In your main class add in the onEnable (replace commandName with the name of the command and CommandExecutor with the name of the class):

```
 getCommand("commandName").setExecutor(new CommandExecutor());
```

Read Commands online: https://riptutorial.com/bukkit/topic/6880/commands

Chapter 3: Configuration Files

Syntax

- `String s = config.getString("path.to.string");`
- `int i = config.getInt("path.to.int");`
- `double d = config.getDouble("path.to.double");`
- `List<String> sl = config.getStringList("path.to.stringlist");`
- `List<Double> dl = config.getDoubleList("path.to.doublelist");`
- `List<Integer> il = config.getIntegerList("path.to.integerlist");`

Remarks

The Bukkit configuration files are straight-forward Y.A.M.L (Yet Another Markup Language) files, and are implemented as so.

Examples

Plugin Config.yml

You can have a config.yml file that loads directly from your jar file. It must be added to your project's folder, the same way as the plugin.yml file is.

In this file you have the default values for your configuration.

Example config:

```
# This is an YML comment
adminName: "Kerooker"
moderators: ["Romario", "Pelé", "Cafú"]
```

The example config file must be added to the project folder.

To load the default configuration file to your plugin's folder, the following code must be added to your onEnable():

```
saveDefaultConfig();
```

This will make your config.yml file from the project to be your plugin's configuration file, and will add it to your plugin's folder.

From there, you can access your config file from anywhere, by using your plugin instance:

```
JavaPlugin plugin; // Your plugin instance
FileConfiguration config = plugin.getConfig();  //Accessing the config file
```

From there, we can access anything that was set on the plugin's config.

Note: The default config file may have it's values changed, if the user wants to edit the config.yml file generated to the folder.

```
String adminName = config.getString("adminName");
List<String> moderators =  config.getStringList("moderators");
```

Multiple Paths Section

What can happen in your config file is having a path to a variable that goes through multiple sections.

Example Config

```
admins:
 first-tier: "Kerooker"
 second-tier: "Mordekaiser"
 third-tier: "Yesh4"
```

The name "Kerooker" is from section "first-tier", which is from section "admins". To access the inner paths of our file, we use a simple '.' as a way of saying that we want the next section. So, for us to access "Kerooker", we go:

```
config.getString("admins.first-tier");
```

Read Configuration Files online: https://riptutorial.com/bukkit/topic/6824/configuration-files

Chapter 4: Entities

Examples

Teleporting an entity to another entity

```
Entity entity;     //The entity you want to teleport
Entity teleportTo; //The entity where you want <entity> to be teleported to

boolean success = entity.teleport(teleportTo);   //Attempting to teleport.

if(success) {
//Teleport was successful
}else {
//Teleport wasn't successful
}
```

You can also add a cause to your teleport, so you can customize how your cause will be treated by your plugin or by others:

```
Entity entity;     //The entity you want to teleport
Entity teleportTo; //The entity where you want <entity> to be teleported to
PlayerTeleportEvent.TeleportCause cause;    //The cause you want the teleport to be of

boolean success = entity.teleport(teleportTo, cause);   //Attempting to teleport.

if(success) {
//Teleport was successful
}else {
//Teleport wasn't successful
}
```

Teleporting an entity to a location

```
Entity toBeTeleported;    //The entity you want to teleport
Location teleportTo = new Location(world,x,y,z,yaw,pitch);    //The location to teleport to


boolean success = toBeTeleported.teleport(teleportTo);

if(success) {
    //Teleport was successful
}else {
    //Teleport wasn't successful
}
```

You can also add a cause to your teleport, so you can customize how your cause will be treated by your plugin or by others:

```
Entity toBeTeleported;    //The entity you want to teleport
Location teleportTo = new Location(world,x,y,z,yaw,pitch);    //The location to teleport to
```

```
PlayerTeleportEvent.TeleportCause cause;    //The cause you want the teleport to be of


boolean success = toBeTeleported.teleport(teleportTo, cause);

if(success) {
    //Teleport was successful
}else {
    //Teleport wasn't successful
}
```

EntityType

The EntityType enum represents all the entities from Bukkit/Spigot.

All its values can be found below.

Value	Description
AREA_EFFECT_CLOUD	N/A
ARMOR_STAND	Mechanical entity with an inventory for placing weapons / armor into.
ARROW	An arrow projectile; may get stuck in the ground.
BAT	N/A
BLAZE	N/A
BOAT	A place boat
CAVE_SPIDER	N/A
CHICKEN	N/A
COMPLEX_PART	N/A
COW	N/A
CREEPER	N/A
DRAGON_FIREBALL	Like FIREBALL, but with extra effects
DROPPED_ITEM	An item resting on the ground.
EGG	A flying chicken egg.
ENDER_CRYSTAL	N/A
ENDER_DRAGON	N/A

Value	Description
ENDER_PEARL	A flying ender pearl.
ENDER_SIGNAL	An ender eye signal.
ENDERMAN	N/A
ENDERMITE	N/A
EXPERIENCE_ORB	An experience orb.
FALLING_BLOCK	A block that is going to or is about to fall.
FIREBALL	A flying large fireball, as thrown by a Ghast for example.
FIREWORK	Internal representation of a Firework once it has been launched.
FISHING_HOOK	A fishing line and bobber.
GHAST	N/A
GIANT	N/A
GUARDIAN	N/A
HORSE	N/A
IRON_GOLEM	N/A
ITEM_FRAME	An item frame on a wall.
LEASH_HITCH	A leash attached to a fencepost.
LIGHTNING	A bolt of lightning.
LINGERING_POTION	A flying lingering potion
MAGMA_CUBE	N/A
MINECART	N/A
MINECART_CHEST	N/A
MINECART_COMMAND	N/A
MINECART_FURNACE	N/A
MINECART_HOPPER	N/A
MINECART_MOB_SPAWNER	N/A

Value	Description
MINECART_TNT	N/A
MUSHROOM_COW	N/A
OCELOT	N/A
PAINTING	A painting on a wall.
PIG	N/A
PIG_ZOMBIE	N/A
PLAYER	N/A
POLAR_BEAR	N/A
PRIMED_TNT	Primed TNT that is about to explode.
RABBIT	N/A
SHEEP	N/A
SHULKER	N/A
SHULKER_BULLET	Bullet fired by SHULKER.
SILVERFISH	N/A
SKELETON	N/A
SLIME	N/A
SMALL_FIREBALL	A flying small fireball, such as thrown by a Blaze or player.
SNOWBALL	A flying snowball.
SNOWMAN	N/A
SPECTRAL_ARROW	Like TIPPED_ARROW but causes the PotionEffectType.GLOWING effect on all team members.
SPIDER	N/A
SPLASH_POTION	A flying splash potion
SQUID	N/A
THROWN_EXP_BOTTLE	A flying experience bottle.
TIPPED_ARROW	Like ARROW but tipped with a specific potion which is

Value	Description
	applied on contact.
UNKNOWN	An unknown entity without an Entity Class
VILLAGER	N/A
WEATHER	N/A
WITCH	N/A
WITHER	N/A
WITHER_SKULL	A flying wither skull projectile.
WOLF	N/A
ZOMBIE	N/A

Passenger

Entities can have passengers. A good example of a passenger is a Player riding a saddled pig, or a zombie inside a minecart.

Although there are specific vehicles, any entity can be a vehicle for any other entity with the SetPassenger method.

```
Entity vehicle;
Entity passenger;
boolean result = vehicle.setPassenger(passenger);   //False if couldn't be done for whatever
reason
```

The passenger should now be attached to the vehicle

You can check if an entity has a passenger using

```
boolean hasPassenger =  entity.isEmpty()
```

If the entity has a passenger, you can retrieve the passenger entity with

```
Entity passenger = entity.getPassenger();
```

Will only return the primary passenger if the vehicle can have multiples.

Finally, you can eject an entity's passenger with

```
boolean b = entity.eject();   //Eject all passengers - returns true if there was a passenger
to be ejected
```

Nearby Entities

To retrieve a list of nearby entities of an entity, one can use

```
List<Entity> nearby = entity.getNearbyEntities(double x, double y, double z);
```

Bukkit will then calculate a bounding box centered around entity, having as parameters:

- x: 1/2 the size of the box along x axis
- y: 1/2 the size of the box along y axis
- z: 1/2 the size of the box along z axis

The list may be empty, meaning that there are no nearby entities with the parameters.

This approach can be used to detect entities near custom projectiles, for example launching an Itemstack and detecting when it collides with a player

Read Entities online: https://riptutorial.com/bukkit/topic/7886/entities

Chapter 5: Entity Events

Introduction

All Entity Events extends EntityEvent, the superclass for EntityEvents.

All known EntityEvents can be found below, and will be covered in this documentation.

Examples

EntityDamage Event

The EntityDamage event is thrown when an Entity is damaged.

EntityDamageEvent

```
@EventHandler
public void onEntityDamage(EntityDamageEvent e) {
    DamageCause cause = e.getCause();    //Get the event DamageCause
    double rawDamage = e.getDamage();    //Returns the damage before any calculation
    double damageModified = e.getDamage(DamageModifier.X);     //Returns the damage that would
be caused with the specified modifier
    double finalDamage = e.getFinalDamage();    //Gets the final damage of this event, with
all the calculations included

    e.setCancelled(boolean x);    //If for any reasons you want the event to not happen, you
can cancel it
    e.setDamage(double damage);    //You can change the full damage the event will cause
    e.setDamage(DamageModifier modifier, double damage);    //Changes the damage considering
any possible modifier
}
```

Most of the times, the EntityDamageEvent will not be used. Instead, one of it's subclasses will be used, such as **EntityDamageByEntityEvent** or **EntityDamageByBlockEvent**. Both can be seen below.

EntityDamageByEntityEvent

```
@EventHandler
public void onEntityDamageByEntity(EntityDamageByEntityEvent e) {
    //Retrieving the Entity that dealt damage
    Entity damageDealer = e.getDamager();

    //Retrieving the Entity that took damage
    Entity damageTaker = e.getEntity();

    //Retrieving the cause of the damage
```

```
    DamageCause cause = e.getDamageCause();

    //damage is the double value of the damage before all the resistances and modifiers have
been applied
    double damage = e.getDamage();

    //FinalDamage is the double value of the damage after all the resistances and modifiers
have been applied
    double finalDamage = e.getFinalDamage();

   //You can also set the raw damage (before modifiers) for the event to a different value
   e.setDamage(20.0);
}
```

EntityDamageByBlockEvent

A simple extension to the EntityDamageEvent, but with one different method:

```
Block b = event.getDamager();    //Returns the block that dealt damage to the entity
```

EntityEvent (Superclass)

The known sub-classes for Entity Events are:

Sub-Classes	Sub-Classes	Sub-Classes
CreatureSpawnEvent	CreeperPowerEvent	EntityChangeBlockEvent
EntityCombustEvent	EntityCreatePortalEvent	EntityDamageEvent
EntityDeathEvent	EntityExplodeEvent	EntityInteractEvent
EntityPortalEnterEvent	EntityRegainHealthEvent	EntityShootBowEvent
EntityTameEvent	EntityTargetEvent	EntityTeleportEvent
EntityUnleashEvent	ExplosionPrimeEvent	FoodLevelChangeEvent
HorseJumpEvent	ItemDespawnEvent	ItemSpawnEvent
PigZapEvent	ProjectileHitEvent	ProjectileLaunchEvent
SheepDyeWoolEvent	SheepRegrowWoolEvent	SlimeSplitEvent

In addition to this, all the sub-classes inherit the following methods:

```
Entity getEntity();            //Entity who is involved in this event
EntityType getEntityType();    //EntityType of the Entity involved in this event
```

Read Entity Events online: https://riptutorial.com/bukkit/topic/6486/entity-events

Chapter 6: Event Handling

Introduction

When something happens inside Bukkit, an Event is called so every plugin can decide what to do whenever something happens.

An Event is called when a player tries to play a block, when an entity despawn, when someone logs in... Plugins can listen to specific events and deal with it in many different ways, for example, sending a message to an admin when a Player logs in, via the PlayerLoginEvent.

Syntax

- Bukkit.getPluginManager().registerEvents(Listener l, Plugin p);

Remarks

When registering an event, take a look if you're not registering it twice! Or your plugin will act twice for the registered event.

Take an extra look for how to handle specific events:

- Player Events
- Entity Events

Examples

Register Events inside the Listener class

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

public class MyEventListener implements Listener {

    /**
     * Constructor
     */
    public MyEventListener(Main plugin){
        //register Events of this class
        //with methode: registerEvents(Listener, Plugin);
        plugin.getServer().getPluginManager().registerEvents(this, plugin);
    }

    /**
     * A Event with HIGH priority
     */
```

```
    @EventHandler(priority = EventPriority.HIGH) //An EventHandler annotation
    public void onPlayerLogin(PlayerLoginEvent event){  //A bukkit event
        event.getPlayer().sendMessage("Welcome.");
    }
    /**
     * A Event with NORMAL (default) priority
     */
    @EventHandler
    public void onPlayerQuit(PlayerQuitEvent event){
        Bukkit.broadcastMessage(event.getPlayer().getName() + " left the Server.");
    }


}

/**
 * Main class
 */
public class Main extends JavaPlugin {
    public void onEnable(){
        //Register Events
        new MyEventListener(this);
    }
}
```

Registering Events to your Main class

```
public class Main extends JavaPlugin {

    @Override
    public void onEnable() {
        Bukkit.getPluginManager().registerEvents(this, this);
    }

    @EventHandler
    public void yourEvent(Event e) {
    //...
    }
}
```

Listening to Events

Bukkit uses an event based system that allows plugin developers to interact with and modify the server and specific actions that occur in the world.

Creating an Event Handler

Event handlers are methods that get called when their event occurs. They are generally public and void as well as named `on{EventNameStem}` by convention. All handlers however, must have the `@EventHandler` annotation, as well as contain its event as the ONLY parameter. Here is an example of an event handler for the `PlayerJoinEvent`

```
@EventHandler
public void onPlayerJoin(PlayerJoinEvent event){
    //Run when a player joins
}
```

Note: The naming format for Bukkit events is `{Source}{Action}({Target})Event`. Some examples of these event names are: `PlayerInteractEvent` or `BlockBreakEvent` or `PlayerJoinEvent`. A list of all events can be found on the Spigot Javadocs

Registering Events

Merely creating an event handler is not enough to allow Bukkit to start sending event calls to your method. You must also register it through the PluginManager interface.

The most common way to register events is to create a class that implements the Listener interface and use it wrap your event handlers.

```
public class EventListener implements Listener { //Implements the Listener interface

    @EventHandler
    public void onPlayerJoin(PlayerJoinEvent event){
        //Run when a player joins
    }

}
```

This listener class and all of its events can then be registered in your main plugin class like this:

```
@Override
public void onEnable(){
    Bukkit.getPluginManager().registerEvents(new EventListener(), this); //Register your
listener and its event handlers
}
```

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());
}
```

Basic Event Handling

Bukkit uses an event based system that allows plugin developers to interact with and modify the server and specific actions that occur in the world.

Creating an Event Handler

Event handlers are methods that get called when their event occurs. They are generally public and

void as well as named `on{EventNameStem}` by convention. All handlers however, must have the `@EventHandler` annotation, as well as contain its event as the ONLY parameter. Here is an example of an event handler for the `PlayerJoinEvent`

```
@EventHandler
public void onPlayerJoin(PlayerJoinEvent event){
    //Run when a player joins
}
```

Note: The naming format for Bukkit events is `{Source}{Action}({Target})Event`. Some examples of these event names are: `PlayerInteractEvent` or `BlockBreakEvent` or `PlayerJoinEvent`. A list of all events can be found on the Spigot Javadocs

Registering Events

Merely creating an event handler is not enough to allow Bukkit to start sending event calls to your method. You must also register it through the PluginManager interface.

The most common way to register events is to create a class that implements the Listener interface and use it wrap your event handlers.

```
public class EventListener implements Listener { //Implements the Listener interface

    @EventHandler
    public void onPlayerJoin(PlayerJoinEvent event){
        //Run when a player joins
    }

}
```

This listener class and all of its events can then be registered in your main plugin class like this:

```
@Override
public void onEnable(){
    Bukkit.getPluginManager().registerEvents(new EventListener(), this); //Register your
listener and its event handlers
}
```

Event Priorities

Bukkit has a system called **Event Priorities** to help plugins handle events in the correct older. The seven priorities are (in older from first executed to last):

- Lowest
- Low
- Normal (default)
- High

- Highest
- Monitor

If you are planning to cancel a lot of events (e.g. protection plugin) it would be a good idea to use lower priority (or lowest) to avoid problems.

You should never modify outcome of an event at MONITOR.

```
@EventHandler //same as @EventHandler(priority = EventPriority.NORMAL)
public void onLogin(PlayerLoginEvent event) {
    // normal login
}

@EventHandler(priority = EventPriority.HIGH)
public void onLogin(PlayerLoginEvent event) {
    // high login
}
```

More info:

- EventPriority at spigot javadocs
- Event Priorities at BukkitWiki

Read Event Handling online: https://riptutorial.com/bukkit/topic/5743/event-handling

Chapter 7: Falling

Remarks

There currently isn't any consistent way to avoid an entity gravity suffering, even if you cancel it's movement, the client-side of the player would still try to fall before the event is cancelled.

Examples

Entity Falling Distance

Entity falling distance is the distance the entity have fallen without reaching a block.

It can be used to calculate different damage from falling, or activating an effect after a big fall.

Retrieving the falling distance

```
float distanceFell = entity.getFallingDistance();
```

Setting the falling distance

This can be used to simulate a different falling distance than the real one. Bukkit will calculate the damage using the new falling distance.

```
entity.setFallingDistance(float distance);
```

Cancelling Damage

You can cancel a fall damage by using the `EntityDamageEvent`

```
@EventHandler
public void onEntityDamage(EntityDamageEvent e) {
Entity tookDamage = e.getEntity();

DamageCause cause = e.getCause();

if (cause == DamageCause.FALL){
   //Damage was caused by falling, cancel it
   e.setCancelled(true);
}
```

Read Falling online: https://riptutorial.com/bukkit/topic/7899/falling

Chapter 8: Hiding Players

Syntax

- void hide(Player toHide);
- void show(Player toShow);
- boolean canSee(Player toBeSeen);

Remarks

Events are better covered in StackOverflow's List of Events documentation

Examples

Hiding a Player from other Players

```
Player playerToHide;
Player playerToNotSee;


playerToNotSee.hide(playerToHide);
//playerToHide will no longer be seen by playerToNotSee.
```

If player is already hidden, nothing happens

Showing a Player to another Player

```
Player toUnhide;
Player toSeeAgain

toSeeAgain.show(toUnhide);
//Player toSeeAgain will now see Player toUnhide again.
```

If player is already visible, nothing happens.

Checking if player can be seen

```
Player playerToCheck;
Player playerSeeing;

boolean isVisible = playerSeeing.canSee(playerToCheck);
//isVisible returns true if playerSeeing can see playerToCheck and false otherwhise
```

Hiding Player from an Entity

This can be done by using the event EntityTargetEvent

Entities won't target the player if you cancel the event:

```
@EventHandler
public void onEntityTarget(EntityTargetEvent e) {
    Entity target = e.getEntity();
    if(target instanceof Player) {
        Player playerTargetted = (Player) target;
        if (shouldBeInvisible(playerTargetted) {
            e.setCancelled(true);    //Cancel the target event
        }
    }
}
```

Read Hiding Players online: https://riptutorial.com/bukkit/topic/7697/hiding-players

Chapter 9: Logging

Examples

Using the Bukkit Logger

```
public class MyClass {

public void foo() {
    Logger logger = Bukkit.getLogger();

    logger.info("A log message");
    logger.log(Level.INFO, "Another message");
    logger.fine("A fine message");

        // logging an exception
        try {
            // code might throw an exception
        } catch (SomeException ex) {
            // log a warning printing "Something went wrong"
            // together with the exception message and stacktrace
            logger.log(Level.WARNING, "Something went wrong", ex);
        }

        String s = "Hello World!";

        // logging an object
        LOG.log(Level.FINER, "String s: {0}", s);

        // logging several objects
        LOG.log(Level.FINEST, "String s: {0} has length {1}", new Object[]{s, s.length()});
    }

}
```

Logging Levels

Java Logging Api has 7 levels. The levels in descending order are:

- `SEVERE` (highest value)
- `WARNING`
- `INFO`
- `CONFIG`
- `FINE`
- `FINER`
- `FINEST` (lowest value)

The default level is `INFO` (but this depends on the system and used a virtual machine).

Note: There are also levels `OFF` (can be used to turn logging off) and `ALL` (the oposite of `OFF`).

Code example for this:

```
import java.util.logging.Logger;

public class Levels {
    private static final Logger logger = Bukkit.getLogger();

    public static void main(String[] args) {

        logger.severe("Message logged by SEVERE");
        logger.warning("Message logged by WARNING");
        logger.info("Message logged by INFO");
        logger.config("Message logged by CONFIG");
        logger.fine("Message logged by FINE");
        logger.finer("Message logged by FINER");
        logger.finest("Message logged by FINEST");

        // All of above methods are really just shortcut for
        // public void log(Level level, String msg):
        logger.log(Level.FINEST, "Message logged by FINEST");
    }
}
```

By default running this class will output only messages with level higher then `CONFIG`:

```
Jul 23, 2016 9:16:11 PM LevelsExample main
SEVERE: Message logged by SEVERE
Jul 23, 2016 9:16:11 PM LevelsExample main
WARNING: Message logged by WARNING
Jul 23, 2016 9:16:11 PM LevelsExample main
INFO: Message logged by INFO
```

Read Logging online: https://riptutorial.com/bukkit/topic/7202/logging

Chapter 10: Manipulating Achievements

Syntax

- player.awardAchievement(Achievement ach);

Examples

Awarding Achievements

```
Player player;     //The player you want to award your achievement with
Achievement achievement;     //The achievement you want to award your player

player.awardAchievement(achievement);    //Awarding the achievement
```

Checking if player has achievement:

```
Player player;
Achievement achievement;
boolean hasAchievement = player.hasAchievement(achievement);
```

The code will award the achievement and any parent achievement.

Possible achievements are:

- ACQUIRE_IRON (Acquire Hardware)
- BAKE_CAKE (The Lie)
- BOOKCASE (Librarian)
- BREED_COW (Repopulation)
- BREW_POTION (Local Brewery)
- BUILD_BETTER_PICKAXE (Getting an Upgrade)
- BUILD_FURNACE (Hot Topic)
- BUILD_HOE (Time to Farm!)
- BUILD_PICKAXE (Time to Mine!)
- BUILD_SWORD (Time to Strike!)
- BUILD_WORKBENCH (Benchmarking)
- COOK_FISH (Delicious Fish)
- DIAMONDS_TO_YOU (Diamonds to you!)
- ENCHANTMENTS (Enchanter)
- END_PORTAL (The End?)
- EXPLORE_ALL_BIOMES (Adventuring Time)
- FLY_PIG (When Pigs Fly)
- FULL_BEACON (Beaconator)
- GET_BLAZE_ROD (Into Fire)
- GET_DIAMONDS (DIAMONDS!)

- GHAST_RETURN (Return to Sender)
- KILL_COW (Cow Tipper)
- KILL_ENEMY (Monster Hunter)
- KILL_WITHER (The Beginning.)
- MAKE_BREAD (Bake Bread)
- MINE_WOOD (Getting Wood)
- NETHER_PORTAL (We Need to Go Deeper)
- ON_A_RAIL (On A Rail)
- OPEN_INVENTORY (Taking Inventory)
- OVERKILL (Overkill)
- OVERPOWERED (Overpowered)
- SNIPE_SKELETON (Sniper Duel)
- SPAWN_WITHER (The Beginning?)
- THE_END (The End.)

Reference

Read Manipulating Achievements online: https://riptutorial.com/bukkit/topic/7690/manipulating-achievements

Chapter 11: NMS

Introduction

NMS, also known as **N**et.**M**inecraft.**S**erver is the package which contains core Minecraft server code. Classes in this package were made by Mojang (not Bukkit) and are therefore mostly obfuscated and not meant to be used or altered. However, interacting with the Minecraft server code at this level allows you to modify almost every aspect of it. This is significant because there are numerous modifications that Bukkit does not support.

Remarks

The Bukkit API is a wrapper or abstraction layer for NMS that allows plugin developers to interact with the server without worrying about changes made to the internal codebase.

Use of NMS code is discouraged as it breaks often between Minecraft version changes and cannot be supported by Bukkit or Spigot as they do not create, own, or maintain it.

Examples

Accessing the Current Minecraft Version

One of the most critical parts of dealing with NMS code is being able to support mulitple Minecraft versions. There are numerous ways to do this, but a simple solution is to use this code to store the version as a public static field:

```
public static final String NMS_VERSION =
Bukkit.getServer().getClass().getPackage().getName().substring(23);
```

This code snippet works by taking the CraftServer class:

```
org.bukkit.craftbukkit.VERSION.CraftServer.class
```

Getting its package:

```
org.bukkit.craftbukkit.VERSION
```

And taking the substring of the package name starting at the index 23 which will always be after 'org.bukkit.craftbukkit.' (which has a length of 23 characters). Resulting in the final VERSION string:

```
VERSION
```

There are a number of reasons why it is so important to be able to access the current Minecraft version. Mostly because any accessing of a class on a server running a different Minecraft version than what the plugin was coding with will throw an Error.

Here is an example that demonstrates how to solve that issue by using the `NMS_VERSION` field to retrieve an instance of CraftPlayer (which is a NMS class) on any Minecraft version.

```
/**
 * Invokes the getHandle() method on the player's CraftPlayer instance to
 * retrieve the EntityPlayer representation of the player as an Object to
 * avoid package version change issues
 *
 * @param player
 *            the player to cast
 * @return the NMS EnityPlayer representation of the player
 */
public static Object getCraftPlayer(Player player) {
    try {
        return Class.forName("org.bukkit.craftbukkit." + NMS_VERSION + ".entity.CraftPlayer")
                .getMethod("getHandle")
                .invoke(player);
    } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException |
NoSuchMethodException | SecurityException | ClassNotFoundException e) {
        throw new Error(e);
    }
}
```

The resulting object can then be manipulated using reflection to perform NMS based tasks without worrying about trying to access the wrong version of the class.

Even this method is not foolproof however, as NMS field and method names change easily, so the only thing you're guaranteeing by doing this is that your code wont definitely break each time Minecraft updates.

Getting a Player's Ping

One very simple thing that you might want to do with NMS that Bukkit doesn't support is get the player's ping. This can be done like this:

```
/**
 * Gets the players ping by using NMS to access the internal 'ping' field in
 * EntityPlayer
 *
 * @param player
 *            the player whose ping to get
 * @return the player's ping
 */
public static int getPing(Player player) {
    EntityPlayer entityPlayer = ((CraftPlayer) player).getHandle();
    return entityPlayer.ping;
}
```

If you're using a method like getCraftPlayer(Player) which returns an instance of the Player's corresponding CraftPlayer instance as an Object. You can access the data without importing the version dependent classes by using reflection like this:

```
/**
 * Gets the player's ping using reflection to avoid breaking on a Minecraft
```

```
 * update
 *
 * @param player
 *            the player whose ping to get
 * @return the player's ping
 */
public static int getPing(Player player) {
    try {
        Object craftPlayer = getCraftPlayer(player);
        return (int) craftPlayer.getClass().getField("ping").get(craftPlayer);
    } catch (IllegalArgumentException | IllegalAccessException | NoSuchFieldException |
SecurityException e) {
        throw new Error(e);
    }
}
```

Read NMS online: https://riptutorial.com/bukkit/topic/9576/nms

Chapter 12: Player Events

Introduction

This is a List of Player Events and an example on how to use them.

Examples

PlayerJoinEvent

```
public class PlayerJoinListener implements Listener {
    @EventHandler
    public void onPlayerJoin(PlayerJoinEvent evt) {
        Player joined = evt.getPlayer();
        String joinedName = joined.getName();

       //RETRIEVING THE JOIN MESSAGE ALREADY SET
       String joinMessage = evt.getJoinMessage();

       //SETTING THE JOIN MESSAGE
       evt.setJoinMessage(joinedName + " has joined the game");

       //CLEARING THE JOIN MESSAGE
       evt.setJoinMessage(null);
    }
}
```

PlayerMoveListener

```
public class PlayerMoveListener implements Listener {
    @EventHandler
    public void onPlayerMove(PlayerMoveEvent evt) {
        Location from = evt.getFrom();
        Location to = evt.getTo();
        double xFrom = from.getX();
        double yFrom = from.getY();
        double zFrom = from.getZ();
        double xTo = to.getX();
        double yTo = to.getY();
        double zTo = to.getZ();

        Bukkit.getLogger().info("Player " + evt.getPlayer().getName()
                        +  " has moved from x: " + xFrom + " y: " + yFrom + " z: "
                        + zFrom + " to x: " + xTo + " y: " + yTo + " z: " + zTo);
    }
}
```

PlayerLoginEvent

Event stores details for players attempting to log in

```
@EventHandler
public void onPlayerLogin(PlayerLoginEvent e) {
    Player tryingToLogin = e.getPlayer();

    //Disallowing a player login
    e.disallow(PlayerLoginEvent.Result.KICK_FULL , "The server is reserved and is full for
you!");

    //Allowing a player login
    if (e.getResult() != PlayerLoginEvent.Result.ALLOW) {
        if (isVip(tryingToLogin) ){
            e.allow();
        }
    }

    //Getting player IP
    String ip = e.getAddress();

    //Get the hostname player used to login to the server
    String ipJoined = e.getHostname();

    //Get current result from the login attempt
    PlayerLoginEvent.Result result = e.getResult();

    //Set kick message if Result wasn't ALLOW
    e.setKickMessage("You were kicked!");

    //Retrieve the kick message
    String s = e.getKickMessage();

}
```

PlayerLoginEvent.Result ENUM:

- ALLOWED - The player is allowed to log in
- KICK_BANNED - The player is not allowed to log in, due to them being banned
- KICK_FULL - The player is not allowed to log in, due to the server being full
- KICK_OTHER - The player is not allowed to log in, for reasons undefined
- KICK_WHITELIST - The player is not allowed to log in, due to them not being on the white list

Player Bed Events

Event fired when player enters a bed: **PlayerBedEnterEvent**

PlayerBedEnterEvent(Player who, Block bed)

```
@EventHandler
public void onPlayerBedEnter(PlayerBedEnterEvent e) {
    Player entered = e.getPlayer();

   Block bedEntered = e.getBed();
}
```

Event fired when player leaves a bed: **PlayerBedLeaveEvent**

PlayerBedLeaveEvent(Player who, Block bed)

```
@EventHandler
public void onPlayerBedEnter(PlayerBedEnterEvent e) {
    Player entered = e.getPlayer();

   Block bedEntered = e.getBed();
}
```

Read Player Events online: https://riptutorial.com/bukkit/topic/8905/player-events

Chapter 13: Scala

Introduction

How to implement Bukkit plugins in the Scala programming language

Examples

Project setup (Scala Eclipse)

Creating a project in scala is very similar to creating one in java. Here is what the entry class should look like:

```
package com.example.myplugin; //{$TopLevelDomain}.{$Domain}.{$PluginName}

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

class PluginName extends JavaPlugin {

  override def onEnable() {

  }

  override def onDisable() {

  }

  override def onCommand(sender: CommandSender, cmd: Command, label: String, args:
Array[String]): Boolean = {

    false
  }

}
```

First, make sure you have installed the latest Scala version located here: https://www.scala-lang.org/download/

Next, you'll want to download Scala Eclipse, available here: http://scala-ide.org/ and extract the download to a folder of your choice.

Once these are both installed, simply open Scala Eclipse.

Lastly, in order for your plugin to work - you need to have some sort of runtime plugin to load the scala library for you, I use this one: https://dev.bukkit.org/projects/scala-loader (place this jar in your plugins folder just like any other plugin)

From here on out, the process is almost identical to java:

1. Press `Alt+Shift+N` -> click `Scala Project`
2. Right-click on your project - click `Properties`
3. Click `Java Build Path`, then click on the `Libraries` tab
4. Click `Add External Jars`, and select your spigot-api jar file
5. Click `Apply` and then `OK`

For the project setup, you'll want to create a package so:

Right-click on project -> `New` -> `Package`

Name it how you'd like, typically: `com.yourdomain.pluginname`

Inside of this package, create a Scala Class and name it how you'd like, typically: `PluginName`

Make the class `extends JavaPlugin` and override the provided functions for a basic setup as shown above.

Lastly, Right-Click on the folder called "src" and select New File. Name the file plugin.yml (NOT the name of your plugin, but explicitly plugin.yml) and open it.

A basic implementation should look like this:

```
name: PluginName
main: com.example.pluginname.PluginName
version: 0.1
```

And there you have it! After you're done writing your plugin, click `File` -> `Export` -> `Java` -> `Jar file` -> Select your project and specify your server's pluginfolder as the destination -> click `Finish`

Typically you can simply reload your server to see the changes after export, however **some plugins will break on reload, so be careful!** I advise always **restarting** the server unless you know that reloading will not break other plugins.

Read Scala online: https://riptutorial.com/bukkit/topic/9259/scala

Chapter 14: Scheduler Programming

Syntax

- `Bukkit.getScheduler().scheduleSyncRepeatingTask(Plugin plugin, Runnable task, int initialDelay, int repeatingDelay)`
- `Bukkit.getScheduler().scheduleSyncDelayedTask(Plugin plugin, Runnable task, int initialDelay)`
- `Bukkit.getScheduler().runTaskAsynchronously(Plugin plugin, Runnable task)`
- `Bukkit.getScheduler().runTask(Plugin plugin, Runnable task)`
- `new BukkitRunnable() { @Override public void run() { /* CODE */ } }.runTaskLater(Plugin plugin, long delay);`
- `new BukkitRunnable() { @Override public void run() { /* CODE */ } }.runTaskTimer(Plugin plugin, long initialDelay, long repeatingDelay);`

Remarks

Few Bukkit API methods are thread-safe and can be called asynchronously. For this reason, Bukkit API methods should *only*, with a few exceptions, be run on the main thread.

Code run inside of `scheduleSync` methods, as well as the `runTask` method will be run on the main thread.

Code run inside of `runTaskAsynchronously` will be run asynchronously from the main thread. Asynchronous methods are very useful for doing large math or database operations without lagging the server, yet will cause undefined behavior if used to call Bukkit API methods. For this reason, Bukkit API methods that should be run after the asynchronous code should always be put in a `runTask` method.

Examples

Scheduler Repeating task

The time for Scheduler Tasks are measured in Ticks. Under normal conditions, there are 20 ticks per second.

Tasks scheduled with `.scheduleSyncRepeatingTask` will be run on the Main Thread

```
Bukkit.getScheduler().scheduleSyncRepeatingTask(plugin, new Runnable() {
    @Override
    public void run() {
        Bukkit.broadcastMessage("This message is shown immediately and then repeated every
second");
    }
}, 0L, 20L); //0 Tick initial delay, 20 Tick (1 Second) between repeats
```

Scheduler Delayed Task

The time for Scheduler Tasks are measured in Ticks. Under normal conditions, there are 20 ticks

per second.

Tasks scheduled with `.scheduleSyncDelayedTask` will be run on the Main Thread

```
Bukkit.getScheduler().scheduleSyncDelayedTask(plugin, new Runnable() {
    @Override
    public void run() {
        Bukkit.broadcastMessage("This message is shown after one second");
    }
}, 20L); //20 Tick (1 Second) delay before run() is called
```

Running Tasks Asynchronously

You can make code run asynchronously from the main thread using `runTaskAsynchronously`. This is useful for doing intensive math or database operations, as they will prevent the main thread from freezing (and the server from lagging).

Few Bukkit API methods are thread-safe, so many will cause undefined behavior if called asynchronously from the main thread.

```
Bukkit.getScheduler().runTaskAsynchronously(plugin, new Runnable() {
    @Override
    public void run() {
        Bukkit.getLogger().info("This message was printed to the console asynchronously");
        //Bukkit.broadcastMessage is not thread-safe
    }
});
```

Running Tasks on the Main Thread

You can also make code run synchronously with the main thread using `runTask`. This is useful when you want to call Bukkit API methods after running code asynchronously from the main thread.

Code called inside of this Runnable will be executed on the main thread, making it safe to call Bukkit API methods.

```
Bukkit.getScheduler().runTask(plugin, new Runnable() {
    @Override
    public void run() {
        Bukkit.broadcastMessage("This message is displayed to the server on the main thread");
        //Bukkit.broadcastMessage is thread-safe
    }
});
```

Running a BukkitRunnable

The BukkitRunnable is a Runnable found in Bukkit. It's possible to schedule a task directly from a BukkitRunnable, and also cancel it from inside itself.

Important: The time on the tasks is measured in Ticks. A second has 20 ticks.

Non-RepeatingTask:

```
JavaPlugin plugin;    //Your plugin instance
Long timeInSeconds = 10;
Long timeInTicks = 20 * timeInSeconds;
new BukkitRunnable() {

    @Override
    public void run() {
        //The code inside will be executed in {timeInTicks} ticks.

    }
}.runTaskLater(plugin, timeInTicks);   // Your plugin instance, the time to be delayed.
```

Repeating Task:

```
JavaPlugin plugin;    //Your plugin instance
Long timeInSeconds = 10;
Long timeInTicks = 20 * timeInSeconds;
new BukkitRunnable() {

    @Override
    public void run() {
        //The code inside will be executed in {timeInTicks} ticks.
       //After that, it'll be re-executed every {timeInTicks} ticks;
      //Task can also cancel itself from running, if you want to.

       if (boolean) {
           this.cancel();
       }

    }
}.runTaskTimer(plugin, timeInTicks, timeInTicks);   //Your plugin instance,
                                                     //the time to wait until first execution,
                                                    //the time inbetween executions.
```

Running Thread Safe Code from an Asynchronous Task

Sometimes you'll need to execute synchronous code from within an asynchronous task. To do this, simply schedule a synchronous task from within the asynchronous block.

```
Bukkit.getScheduler().runTaskTimerAsynchronously(VoidFlame.getPlugin(), () -> {

    Bukkit.getScheduler().runTask(VoidFlame.getPlugin(), () -> {
        World world = Bukkit.getWorld("world");
        world.spawnEntity(new Location(world, 0, 100, 0), EntityType.PRIMED_TNT);
    });

}, 0L, 20L);
```

Read Scheduler Programming online: https://riptutorial.com/bukkit/topic/5436/scheduler-programming

Chapter 15: Spawn Eggs

Remarks

Refer to Entities Documentation to understand EntityType better

Examples

Creating an ItemStack of a SpawnEgg

For anything below 1.9

```
SpawnEgg egg = new SpawnEgg(EntityType.CREEPER);
ItemStack creeperEgg = egg.toItemStack(5);
```

For 1.9 and above

In versions 1.9 and higher, Spigot does not have an implementation for creating spawn eggs without using NMS. To accomplish this, you can use a small custom class/wrapper to make it happen:

```
public ItemStack toItemStack(int amount, EntityType type) {
    ItemStack item = new ItemStack(Material.MONSTER_EGG, amount);
    net.minecraft.server.v1_9_R1.ItemStack stack = CraftItemStack.asNMSCopy(item);
    NBTTagCompound tagCompound = stack.getTag();
    if(tagCompound == null){
        tagCompound = new NBTTagCompound();
    }
    NBTTagCompound id = new NBTTagCompound();
    id.setString("id", type.getName());
    tagCompound.set("EntityTag", id);
    stack.setTag(tagCompound);
    return CraftItemStack.asBukkitCopy(stack);
}
```

Read Spawn Eggs online: https://riptutorial.com/bukkit/topic/7737/spawn-eggs

Chapter 16: Versions

Examples

Getting Version on Runtime

```
@Override
public void onEnable() {
    String version = Bukkit.getBukkitVersion();    //The string version of this bukkit server
}
```

Accessing NMS across implementation versions

```
// gets the Class objects from the net.mminecraft.server package with the given name
public Class<?> getNmsClass(String name) throws ClassNotFoundException {
    // explode the Server interface implementation's package name into its components
    String[] packageArray = Bukkit.getServer().getClass().getPackage().getName().split("\\.");

    // pick out the component representing the package version if it's present
    String packageVersion = array.length == 4 ? array[3] + "." : "";

    // construct the qualified class name from the obtained package version
    String qualName = "net.minecraft.server." + packageVersion + name;

    // simple call to get the Class object
    return Class.forName(qualName);
}
```

Read Versions online: https://riptutorial.com/bukkit/topic/7730/versions

Chapter 17: World generation

Examples

Void Generator

The void generator class:

```
public class VoidGenerator extends ChunkGenerator
{

    @SuppressWarnings("deprecation")
    public byte[] generate(World w, Random rand, int x, int z)
    {
        byte[] result = new byte[32768]; //chunksized array filled with 0 - Air
        //Build a platform with Bedrock where the player shall spawn later
        if(x == 0 && z == 0)
        {
            result[xyz(0, 64, 0)] = (byte)Material.BEDROCK.getId();
            result[xyz(1, 64, 0)] = (byte)Material.BEDROCK.getId();
            result[xyz(0, 64, 1)] = (byte)Material.BEDROCK.getId();
        }
        return result;
    }

    private Integer xyz(int x, int y, int z)
    {
        return (x * 16 + z)*128+y; //position inside the chunk
    }

}
```

Any class where you want to generate a new World:

```
public void generateWorld(String mapName)
{
    try
    {
        WorldCreator w = new WorldCreator(mapName);
        w.generateStructures(false); //no trees, etc.
        w.generator(new VoidGenerator()); //use the VoidGenerator
        w.environment(Environment.NORMAL); //no nether, etc.
        w.createWorld(); //create the world
    }
    catch(Exception ex)
    {
      ex.printStackTrace();
    }
}
```

Read World generation online: https://riptutorial.com/bukkit/topic/6652/world-generation

Chapter 18: World Manipulation

Remarks

Refer to World Generation for world generation topics

Examples

Creating Explosions

To create an explosion, the following method signatures may be used:

```
boolean createExplosion(double x, double y, double z, float power);
boolean createExplosion(double x, double y, double z, float power, boolean setFire);
boolean createExplosion(double x, double y, double z, float power,
                        boolean setFire, boolean breakBlocks);
boolean createExplosion(Location loc, float power);
boolean createExplosion(Location loc, float power, boolean setFire);
```

- x, y, z and loc represent the location where you want the explosion to happen.
- power represents the power of your explosion, TnT power is 4F.
- setFire represents the capability of the explosion to set blocks on fire
- breakBlocks represents the capability of the explosion to destroy blocks around it.
- all the methods return true if the explosion happenned, and return false if a plugin cancelled the explosion event.

Simulating a TnT explosion that break blocks and set fire at x=0, y=0 and z=0

```
createExplosion(0.0, 0.0, 0.0, 4F, true, true);
```

Dropping an Item

The following methods can be used to drop an Item somewhere in the world:

```
Item dropItem(Location loc, ItemStack is);
Item dropItemNaturally(Location loc, ItemStack is);
```

`dropItem` means dropping an Item exactly at the location, returning an Item object.

`dropItemNaturally` means dropping the Item at the location, but with a random offset, meaning it won't be exactly at the location, but very close nearby. This is made to simulate an item being dropped by an entity or a block such as a Dispenser.

Generating a Tree

The following methods can be used to generate a tree naturally (as if it was grown from a sapling) into the world.

```
boolean generateTree(Location loc, TreeType type);
boolean generateTree(Location loc, TreeType type, BlockChangeDelegate delegate);
```

- Location is where you want the tree to spawn
- TreeType is the type of the tree you want to spawn, and can be one of the following

TreeType enum

Type	Description
ACACIA	Acacia tree
BIG_TREE	Regular tree, extra tall with branches
BIRCH	Birch tree
BROWN_MUSHROOM	Big brown mushroom; tall and umbrella-like
CHORUS_PLANT	Large plant native to The End
COCOA_TREE	Jungle tree with cocoa plants; 1 block wide
DARK_OAK	Dark Oak tree.
JUNGLE	Standard jungle tree; 4 blocks wide and tall
JUNGLE_BUSH	Small bush that grows in the jungle
MEGA_REDWOOD	Mega redwood tree; 4 blocks wide and tall
RED_MUSHROOM	Big red mushroom; short and fat
REDWOOD	Redwood tree, shaped like a pine tree
SMALL_JUNGLE	Smaller jungle tree; 1 block wide
SWAMP	Swamp tree (regular with vines on the side)
TALL_BIRCH	Tall birch tree
TALL_REDWOD	Tall redwood tree with just a few leaves at the top
TREE	Regular tree, no branches

- delegate may be used if you want a class to call for each block changed as a result of this method

Both signatures will return true if the tree was sucessfully generated, false otherwise.

Spawning Rules

There are some spawning rules in Worlds in Bukkit. They are:

- Animal Spawning
- Creature Spawning
- Amount of the above that can be spawned

Animal Spawning

Animal spawning can be split into the following categories:

- Water Animals
- Land Animals

To get the amount of animals that can be spawned inside the World at runtime, you can use the method

```
int getAnimalSpawnLimit()
```

For land animals and

```
int getWaterAnimalSpawnLimit();
```

For water animals.

Both limits can be set with the methods

```
void setAnimalSpawnLimit(int limit);
void setWaterAnimalSpawnLimit(int limit);
```

Note: If set to numbers below 0, world's default amount will be used instead.

Minecraft makes an attempt to spawn animals every 400 ticks (default). That can be changed if you desire, using the following signatures:

```
void setTicksPerAnimalSpawns(int ticks);
void setTicksPerWaterAnimalSpawns(int ticks);
```

- A value of 1 will mean the server will attempt to spawn animals in this world every tick.
- A value of 400 will mean the server will attempt to spawn animals in this world every 400th tick.
- A value below 0 will be reset back to Minecraft's default.

Note: If set to 0, animal spawning will be disabled for this world. It's recommended to use

setSpawnFlags(boolean, boolean) to control this instead.

Read World Manipulation online: https://riptutorial.com/bukkit/topic/7926/world-manipulation

Credits

S. No	Chapters	Contributors
1	Getting started with bukkit	Community, Drayke, ItzPam, Jojodmo, Keenan Thompson, Kerooker, kmecpp, Martin W, RamenChef, Tassu, Unihedron
2	Commands	annon, Kerooker, Martin W
3	Configuration Files	Kerooker
4	Entities	Kerooker
5	Entity Events	Alw7SHxD, Ferrybig, Kerooker, kmecpp
6	Event Handling	Drayke, Jarrod Dixon, Kerooker, kmecpp, Martin W, Tassu
7	Falling	Kerooker
8	Hiding Players	Kerooker
9	Logging	Kerooker
10	Manipulating Achievements	Kerooker
11	NMS	Alw7SHxD, kmecpp
12	Player Events	Ferrybig, Kerooker, Tassu
13	Scala	annon
14	Scheduler Programming	Ferrybig, Jojodmo, Kerooker, kmecpp, Pokechu22
15	Spawn Eggs	Infuzed guy, Kerooker, RamenChef
16	Versions	caseif, Kerooker
17	World generation	Drayke
18	World Manipulation	Kerooker