GeyserMC

GeyserMC Developers

Table of contents

Introduction

Events

Plugins

Contributing

Introduction

Geyser is a bridge between Minecraft: Bedrock Edition and Minecraft: Java Edition, closing the gap from those wanting to play true cross-platform.

Geyser is an open collaboration project by CubeCraft Games.

What is Geyser?¶

Geyser is a proxy, bridging the gap between Minecraft: Bedrock Edition and Minecraft: Java Edition servers. The ultimate goal of this project is to allow Minecraft: Bedrock Edition users to join Minecraft: Java Edition servers as seamlessly as possible. Please note, this project is still a work in progress and should not be used on production. Expect bugs!

Special thanks to the DragonProxy project for being a trailblazer in protocol translation and for all the team members who have now joined us here!

> **Note**
>
> Currently supporting Minecraft Bedrock v1.14.6(0) and Minecraft Java v1.15.2.

Setting Up¶

Take a look here for how to set up Geyser.

Links:¶

- Website: https://geysermc.org
- Docs: https://github.com/GeyserMC/Geyser/wiki
- Download: http://ci.geysermc.org
- Discord: http://discord.geysermc.org/
- Donate: https://patreon.com/GeyserMC

Libraries Used:¶

- NukkitX Bedrock Protocol Library
- Steveice10's Java Protocol Library
- TerminalConsoleAppender
- Simple Logging Facade for Java (slf4j)

Last update:

Events¶

Geyser has an Event Manager that allows one to listen to or trigger an event easily. Events can be easily defined either by defining a method to be an event handler or by providing a lambda. Each event handler will be executed in turn based upon their priority.

Triggering an Event¶

An event is derived from either `GeyserEvent` or `CancellableGeyserEvent`.

Example

```
public class MyCustomEvent extends GeyserEvent {
}
```

The event is triggered through the `triggerEvent` method of the Event Manager.

Example

```
eventManager.triggerEvent(new MyCustomEvent());
```

This returns an EventResult which can be used to chain additional commands based upon the result. They include `onNotCancelled()`, `onCancelled()` and `orElse()` to more easily execute based upon the result and they can chain together.

Example

```
eventManager.triggerEvent(new MyCustomEvent())
    .onNotCancelled((result) -> {
        // Code executed if events were not cancelled
    })
    .orElse((result) -> {
        // Code executed if the above condition was not satisfied
    });
```

`triggerEvent()` can have an optional extra parameter passing in a `Class<?>`. If set then the handler will only execute if it has a filter list containing this filter.

Listening to an Event¶

There are two ways to listen for an event. One can either create an event handler method or one can create an anonymous lamda to be executed when the event is triggered.

Class Event Handler¶

An event handler method is a method that is annotated with `@Event`. The class it belongs to must also be registered with the event manager.

Example

```
public class MyClass {

    @Event
    public void onEnable(MyCustomEvent event) {
        System.err.println("Hello World");
    }
}

...

GeyserConnecter.getInstance().getEventManager.registerEvents(new MyClass());
```

Important

Plugins should use the `registerEvents` method inherited from `GeyserPlugin`.

The `@Event` annotation has the following optional parameters:

- priority - `Integer` from 0 - 100. Default `50`. Event Handlers are executed in order from lowest to highest priority.
- ignoreCancelled - `Boolean`. Default `true`. If true then if an event is cancelled the handler will not be executed.
- filter - `List<Class<?>>`. Default {}. If set will only execute the event handler if the passed in filter to `triggerEvent` is null or matches any on this list.

`MyCustomEvent` is the event defined previously.

Anonymous Lambda¶

An event can be hooked through the `on` method of the EventManager provided with an anonymous function. This allows code to be placed logically close to where it is related in the code instead of having to set up a separate class and method listeners.

Example

```
GeyserConnector.getInstance().getEventManager().on(MyCustomEvent.class, (handler, e
    System.err.println("Hello World");
}).build();
```

Important

Plugins should use the `on` method inherited from `GeyserPlugin`.

You'll note the `build()` on the end. `on()` returns a Builder that can add optional parameters. This must be finalized with a `build()` that generates the EventHandler and registers it with the EventManager.

The following additional parameters are available:

- `priority(int)` - Set the event priority. Default `EventHandler.PRIORITY.NORMAL`
- `ignoreCancelled(boolean)` - If true the handler will not execute if cancelled. Default `true`.
- `filter(Class<?>[])` - List of filters the handler will accept. Default `{}`

Example

```
GeyserConnector.getInstance().getEventManager().on(MyCustomEvent.class, (handler, e
    System.err.println("Hello World");
})
    .priority(30)
    .ignoreCancelled(false)
    .build();
```

Events¶

Geyser has the following predefined Events.

DownstreamPacketReceiveEvent<T>¶

cancellable

Modifier and Type	Method	Description
GeyserSession	getSession()	Gets the current session
T	getPacket()	Gets the Packet

Triggered for each packet received from downstream. If cancelled then regular processing of the packet will not occur.

The type of packet should be passed in as a Type. The filter should also be set to limit what packets you want otherwise every Downstream packet will trigger this handler.

Example

```
@Event(filter = ClientChatPacket.class)
public void onTextPacket(DownstreamPacketReceiveEvent<ClientChatPacket> event) {
    getLogger().warning("Got packet: " + event.getPacket());
}
```

Example

```
@Event
public void onGenericPacket(DownstreamPacketReceiveEvent<Packet> event) {
    getLogger().warning("Got generic packet: " + event.getPacket());
}
```

DownstreamPacketSendEvent<T>¶

cancellable

Modifier and Type	Method	Description
GeyserSession	getSession()	Gets the current session
T	getPacket()	Gets the Packet

Triggered for each packet sent to downstream. If cancelled then the packet will not be sent.

The type of packet should be passed in as a Type. The filter should also be set to limit what packets you want otherwise every Downstream packet will trigger this handler.

Example

```
@Event(filter = ServerChatPacket.class)
public void onTextPacket(DownstreamPacketSendEvent<ServerChatPacket> event) {
    getLogger().warning("Sending packet: " + event.getPacket());
}
```

Example

```
@Event
public void onGenericPacket(DownstreamPacketSendEvent<Packet> event) {
    getLogger().warning("Sending generic packet: " + event.getPacket());
}
```

GeyserStartEvent¶

Triggered after Geyser has finished starting.

GeyserStopEvent¶

Triggered when Geyser is about to stop.

PluginDisableEvent¶

Modifier and Type	Method	Description
GeyserPlugin	getPlugin()	Gets the Plugin

Triggered each time a plugin is disabled.

PluginEnableEvent¶

cancellable

Modifier and Type	Method	Description
GeyserPlugin	getPlugin()	Gets the Plugin

Triggered each time a plugin is enabled.

PluginMessageEvent¶

cancellable

Modifier and Type	Method	Description
String	getChannel()	Gets the message channel
byte[]	getData()	Gets the message data
GeyserSession	getSession()	Gets the current session

UpstreamPacketReceiveEvent<T>¶

cancellable

Modifier and Type	Method	Description
GeyserSession	getSession()	Gets the current session
T	getPacket()	Gets the Packet

Triggered for each packet received from upstream. If cancelled then regular processing of the packet will not occur.

The type of packet should be passed in as a Type. The filter should also be set to limit what packets you want otherwise every Upstream packet will trigger this handler.

Example

```
@Event(filter = TextPacket.class)
public void onTextPacket(UpstreamPacketReceiveEvent<TextPacket> event) {
    getLogger().warning("Got packet: " + event.getPacket());
}
```

Example

```
@Event
public void onGenericPacket(UpstreamPacketReceiveEvent<BedrockPacket> event) {
    getLogger().warning("Got generic packet: " + event.getPacket());
}
```

UpstreamPacketSendEvent<T>¶

cancellable

Modifier and Type	Method	Description
GeyserSession	getSession()	Gets the current session
T	getPacket()	Gets the Packet

Triggered for each packet sent upstream. If cancelled then the packet will not be sent.

The type of packet should be passed in as a Type. The filter should also be set to limit what packets you want otherwise every Upstream packet will trigger this handler.

Example

```
@Event(filter = TextPacket.class)
public void onTextPacket(UpstreamPacketSendEvent<TextPacket> event) {
    getLogger().warning("Sending packet: " + event.getPacket());
}
```

Example

```
@Event
public void onGenericPacket(UpstreamPacketSendEvent<BedrockPacket> event) {
    getLogger().warning("Sending generic packet: " + event.getPacket());
}
```

Last update:

Plugins¶

Geyser provides support for third party plugins which can be placed into a `plugins` folder under the Geyser data folder.

Plugins provide a way to extend the features of Geyser without needing to deal with the Geyser code. It is hoped that developers will be able to create plugins that would be of use to others.

This page describes how to write a plugin.

Maven¶

Add the following to the relevant section of your `pom.xml`

```
<repositories>
    <!-- Bundabrg's Repo -->
    <repository>
        <id>bundabrg-repo</id>
        <url>https://repo.worldguard.com.au/repository/maven-public<
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>

<dependencies>
    <!-- Geyser -->
    <dependency>
        <groupId>org.geysermc</groupId>
        <artifactId>connector</artifactId>
        <version>1.1.0-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Info

Plugin support is presently only available through a feature branch and thus there will be references to a third party maven repository that holds a build of this branch. This will change in the future.

Example Plugin¶

```
@Plugin(
    name = "ExamplePlugin",
    version = "1.1.0-dev",
    authors = {"bundabrg"},
    description = "Provides an example plugin"
)
public class MyPlugin extends GeyserPlugin {
    public MyPlugin(PluginManager pluginManager, PluginClassLoad
        super(pluginManager, pluginClassLoader);
    }

    @Event
    public void onEnable(PluginEnableEvent event) {
        if (event.getPlugin() == this) {
            System.err.println("I'm alive");

            // Register another class with event handlers
            registerEvents(new MyAdditionalClass());

            // Example of lambda event hook
            on(PluginDisableEvent.class, (handler, event) -> {
                if (event.getPlugin() == MyPlugin.this) {
                    System.err.println("I'm also dead");
                }
            })
                .priority(EventHandler.PRIORITY.HIGH)
                .build();
        }
    }

    @Event
    public void onDisable(PluginDisableEvent event) {
        if (event.getPlugin() == this) {
            System.err.println("I'm dead");
        }
    }

}
```

Plugin EntryPoint¶

A plugin must at a minimum define a class that extends `GeyserPlugin` and be annotated with `@Plugin`. The annotation provides details about the plugin such as its version and author(s).

The following fields are available for `@Plugin`:

- name - Name of the plugin. Used in the logs.
- version - Version of the plugin.
- authors - A list of authors
- description - A short description of the plugin
- global - Should the plugin make its classes available to other plugins (default: true)

Plugin Events¶

A plugin will generally hook into several events and provides its own event registration inherited from `GeyserPlugin`.

A plugin class will look for any methods annotated with `@Event` and will treat them as Event Handlers, using reflection to determine which event is being trapped. In the previous example the plugin has trapped both the `PluginEnableEvent` and `PluginDisableEvent`.

Please refer to events for more information about the event system.

Note

There is no need to register the plugin class for events as it will be registered by default.

Note

The plugin class itself provides many of the registration methods found in the Event Manager to track which events belong to the plugin. You should use the plugins own registration methods in preference to those in the Event Manger. This includes `registerEvents` and `on`.

Plugin Messages¶

A plugin can communicate with a plugin on the downstream server through the use of plugin message channels. More information about this can be found here.

Sending a Plugin Message¶

To send a plugin message use `GeyserSession#sendPluginMessage.`

Example

```
ByteArrayDataOutput out = ByteStreams.newDataOutput();
out.writeUTF("Data1");
out.writeUTF("Data2");
session.sendPluginMessage("myplugin:channel", out.toByteArray());
```

Receiving Plugin Messages¶

To receive plugin messages you need to first register to receive the message then listen for the `PluginMessageEvent`.

Example

```
GeyserConnector.getInstance().registerPluginChannel("myplugin:channelname");

'''

@Event
void public onPluginMessageEvent(PluginMessageEvent event) {
    if (!event.getChannel().equals("myplugin:channelname")) {
        return;
    }

    ...
}
```

Logging¶

Plugin classes use their `getLogger()` to retrieve a logging interface. This will log messages to the regular log with the plugin name prefixed in the messages.

Example

```
@Plugin(...)
public class MyPlugin extends GeyserPlugin {
    public void myMethod() {
        ...
        getLogger().info("This is an informative message!");
    }
}
```

Last update:

Contributing¶

Here are some ways that you can help contribute to this project.

New ideas or Bug Reports¶

Need something? Found a bug? Or just have a brilliant idea? Head to the Issues and create a new one.

Please feel free to reach out to us on Discord if you're interested in helping out with Geyser.

Contributing Code¶

If you know Java then take a look at open issues and create a pull request.

Do the following to build the code:

```
git clone https://github.com/GeyserMC/Geyser
cd Geyser
git submodule update --init --recursive
mvn clean install
```

Contributing Documentation¶

If you can help improve the documentation it would be highly appreciated. Have a look under the `docs` folder for the existing documentation.

The documentation is built using `mkdocs`. You can set up a hot-build dev environment that will auto-refresh changes as they are made.

Requirements¶

- python3
- pip3
- npm (only if changing themes)

Install dependencies by running:

```
pip3 install -r requirements.txt
```

Note

It's recommended to use a `virtualenv` so that you don't pollute your system environment.

Dev Environment¶

To start a http document server on `http://127.0.0.1:8000` execute:

```
mkdocs serve
```

Change PDF Theme¶

Edit the PDF theme under `docs/theme/pdf`. Rebuild by doing the following:

```
cd docs/theme/pdf
npm install
npm run build-compressed
```

This will update `pdf.css` under `docs/css/pdf.css`. Rebuilding the docs will now use the new theme.

Last update: