Huomautus

Release 0.1.0

Laura F. Dickinson

Jan 03, 2020

CONTENTS:

Huomautus (lit: notice, annotation, note) is an annotation processor for Kotlin-based Fabric Minecraft mods. It contains helpers for mixins and auto-registration.

Huomautus does not include any dependency in your final code; it does not use any reflection; all is achieved through the power of code generation.

CHAPTER

ONE

SETUP

Huomautus is a compile-time annotation processor, so it requires some setup in your gradle buildscript. The code below is for Kotlin DSL buildscripts.

1.1 Maven Repo

TODO: publish this somewhere

1.2 Adding kapt

Huomautus is powered by `kapt`, the Kotlin annotation processor tool. Add the following to your plugins block:

```
plugins {
    kotlin("kapt")
}
```

Next, you need to configure kapt:

```
kapt {
    // Required line!
    annotationProcessor("green.sailor.mc.huomautus.Processor")
    arguments {
        // Set the package for generated code to be outputted here.
        arg("sailor.huomautus.package", "green.sailor.mc.testmod.generated")
        // Set the prefix for your Blocks, Items, etc objects to have.
        arg("sailor.huomautus.prefix", "TestMod")
    }
}
```

Finally, add the generated sourceset:

```
sourceSets {
    main {
        java {
            srcDir("${buildDir.absolutePath}/generated/source/kaptKotlin/")
        }
    }
}
```

1.3 Adding Huomautus

As Huomautus is an annotation processor, you only want to use it at compile time. None of the annotations provided are retained at runtime, so there's no point adding it as a runtime dependency.

```
val huomautusVersion = "0.1.0"

compileOnly(group = "green.sailor.mc", name = "huomautus", verison = huomautusVersion)
kapt(group = "green.sailor.mc", name = "huomautus", verison = huomautusVersion)
```

CHAPTER

TWO

MIXIN HELPERS

Huomautus comes with some helpers for mixin generation.

2.1 Setting up mixins.json

Your mixins.json should point to the package name you specified with `mixins` added, as this is where the generated mixin objects will be.

```
"package": "green.sailor.mc.testmod.generated.mixin"
```

2.2 MixinImpl

Kotlin cannot be used directly with Mixins, so a pattern emerges which consists of writing the actual mixin class in Java and the implementation elsewhere in Kotlin. Huomautus can automate this, by writing your Mixin entirely in Kotlin and auto-generating a Java bridge.

```
import green.sailor.mc.huomautus.annotations.MixinImpl
import net.minecraft.client.gui.screen.TitleScreen
import org.spongepowered.asm.mixin.injection.At
import org.spongepowered.asm.mixin.injection.Inject
import org.spongepowered.asm.mixin.injection.callback.CallbackInfo

@MixinImpl(TitleScreen::class)
object MixinMinecraftClient {
    @Inject(at = [At(value = "HEAD")], method = ["init()V"])
    fun injectInit(info: CallbackInfo) {
        println("Hello, world!")
    }
}
```

This will generate a Java class that simply calls `MixinMinecraftClient.injectInit` automatically with the same params as you specify in your function.

Note: In `@Inject` and other similar annotations, you need to use the array literal format for some fields instead of just the literal, due to differences in annotation handling between Java and Kotlin.

Warning: Only methods are supported with MixinImpl. Shadowed fields are unsupported.

2.3 GenerateExtensions

`@GenerateExtensions` can be added to an accessor interface to automatically generate extension properties for that class, avoiding the need to cast.

```
@Mixin(MinecraftClient.class)
@GenerateExtensions
public interface MinecraftClientAccessor {
    @Accessor
    int getFpsCounter();
}
```

Then, in some other class:

```
println("FPS: ${MinecraftClient.getInstance().fpsCounter")
```

If only a getter is provided, these extensions are `val` properties; if a setter is provided too, then these properties will be `var` properties.