# Entwickler (API)

GWorld v2 bietet eine moderne API, um Welten zu verwalten, zu erstellen und Einstellungen (Flags) programmatisch zu ändern.

*Version v2.0.1*

# 1. Setup & Integration

Um GWorld v2 in deinem Plugin zu nutzen, musst du es als Abhängigkeit (Dependency) hinzufügen.

# 1. Maven Dependency

Füge das `core`-Modul von GWorld in deine `pom.xml` ein. Da die API zur Laufzeit vom Server bereitgestellt wird, nutzen wir den Scope `provided`.

```
<dependency>
  <groupId>de.gilljan</groupId>
  <artifactId>gworld-core</artifactId>
  <version>2.0.1-SNAPSHOT</version>
  <scope>provided</scope>
</dependency>
```

Wir verwenden die GitHub Maven Package Registry. Um diese zu verwenden, beachte bitte die Dokumentation von GitHub:

https://docs.github.com/de/packages/working-with-a-github-packages-registry/working-with-the-apache-maven-registry#installing-a-package

# 2. plugin.yml

Damit dein Plugin sicher auf GWorld zugreifen kann, muss es *nach* GWorld geladen werden. Füge dazu den Eintrag in deine `plugin.yml` hinzu:

```
depend: [GWorld]
# Alternativ, wenn GWorld optional ist:
# softdepend: [GWorld]
```

# 3. Zugriff auf die API

Der Haupteinstiegspunkt ist das Interface `GWorldAPI`. Du erhältst die Instanz über die statische Methode der Hauptklasse.

```
import de.gilljan.gworld.GWorld;
import de.gilljan.gworld.api.GWorldAPI;
import de.gilljan.gworld.api.IWorldManager;

public class MyPlugin extends JavaPlugin {

    private IWorldManager worldManager;

    @Override
    public void onEnable() {
        if (Bukkit.getPluginManager().isPluginEnabled("GWorld")) {
            // API Instanz abrufen
            GWorldAPI api = GWorld.getInstance();

            // Den Manager für Welten laden
            this.worldManager = api.getWorldManager();
        }
    }
}
```

# 2. Welten erstellen (Builder)

In GWorld v2 werden neue Welten über das **Builder-Pattern** erstellt. Dies trennt die **Konfiguration** (Registrierung) von der **Generierung** (Last).

# Schritt 1: Der Builder (Registrierung)

Mit dem `WorldCreationBuilder` legst du alle Eigenschaften der Welt fest. Der Aufruf von `.build()` registriert die Welt in der Datenbank, erstellt sie aber noch nicht physikalisch.

```
import de.gilljan.gworld.api.IManageableWorld;
import de.gilljan.gworld.enums.WorldTypeMapping;

// Startet den Builder
IManageableWorld newWorld = worldManager.createBuilder("MeineEventWelt")
    .worldType(WorldTypeMapping.NORMAL) // Umgebung: Normal, Nether, End, Large Biomes,
Amplified, Flat
    .generator("PlotSquared")           // Optional: Custom Generator Name
    .seed(987654321L)                   // Optional: Seed festlegen
    .build();                           // -> Speichert die Welt in die Config/DB
```

> **Hinweis:** Die Methode `.build()` ruft intern `addWorldFromBuilder` auf. Die Welt ist danach bekannt (`registered`), aber noch `unloaded`.

# Schritt 2: Laden (Generierung)

```
// Erstellt die Bukkit-Welt und lädt Chunks
boolean success = newWorld.createMap();

if (success) {
  getLogger().info("Welt ist bereit!");
}
```

> 

> Für einen Import gilt dies analog. Den Ordnernamen der Welt verwenden und anstatt `createMap()` die Methode `importMap()` verwenden.

# Warum zwei Schritte?

1. **Performance:** Du kannst Welten beim Serverstart registrieren, aber erst laden, wenn ein Spieler ein Minigame startet.
2. **Sicherheit:** Du kannst Flags (z. B. PvP aus) setzen, bevor der erste Spieler die Welt betritt.

# 3. Management & Steuerung

Der `IWorldManager` verwaltet alle Welten, während das `IManageableWorld`-Objekt die Kontrolle über eine spezifische Welt ermöglicht.

# Welten abrufen

```
// Einzelne Welt holen
Optional<IManageableWorld> worldOpt = worldManager.getWorld("Lobby");

// Alle Welten auflisten
List<IManageableWorld> allWorlds = worldManager.getWorlds();
```

# Welt-Aktionen

Jedes `IManageableWorld`-Objekt bietet Methoden zur Steuerung:

| Methode | Beschreibung |
|---|---|
| `loadMap()` | Lädt die Welt von der Festplatte (Bukkit World Init). |
| `unloadMap()` | Entlädt die Welt und teleportiert Spieler zum Hauptspawn. |
| `deleteMap()` | Löscht die Welt unwiderruflich (Dateien & Datenbank-Eintrag). |
| `reCreate(boolean save)` | Löscht die Welt und generiert sie neu (Reset). Optional mit Backup (true) oder ohne (false). |
| `clone(String name)` | Erstellt eine Kopie der Welt unter neuem Namen. |

# Beispiel: Welt-Reset durchführen

```
worldManager.getWorld("Farmwelt").ifPresent(world -> {
    // Welt zurücksetzen und Kopie der alten Welt behalten
    world.reCreate(true);
});
```

# Beispiel: Welt entfernen (aus dem System)

Wenn du eine Welt aus GWorld entfernen möchtest (inklusive Löschung der Dateien):

```
// Variante A: Direkt über das Objekt (Empfohlen)
world.deleteMap();

// Variante B: Über den Manager
worldManager.removeWorld(world);
```

# 4. Flags & Properties

GWorld erlaubt es, Welteinstellungen (Flags) programmatisch zu ändern. Diese Einstellungen werden persistent gespeichert.

# Flags setzen

Du kannst Flags direkt am `IManageableWorld`-Objekt ändern oder schon während der Erstellung im Builder setzen.

```
// Beispiel: PvP deaktivieren und Schwierigkeit ändern
world.setAllowPvP(false);
world.setDifficulty(Difficulty.HARD);

// WICHTIG: Speichern, damit es nach Neustart erhalten bleibt!
world.saveProperties();
```

# Verfügbare Einstellungen

Hier ist eine Übersicht der wichtigsten Methoden im `IManageableWorld` Interface:

| Kategorie | Methoden (Getter / Setter) | Beschreibung |
|---|---|---|
| **Kampf** | `isAllowPvP`, `setAllowPvP` | Globales PvP an/aus. |
| **Spawning** | `isMonsterSpawning`, `setMonsterSpawning` | Spawnen von Monstern. |
| | `isAnimalSpawning`, `setAnimalSpawning` | Spawnen von Tieren. |
| **Umgebung** | `isWeatherCycle`, `setWeatherCycle` | Ob sich das Wetter ändert. |
| | `isTimeCycle`, `setTimeCycle` | Ob die Tageszeit voranschreitet. |
| | `getTime`, `setTime` | Aktuelle Zeit in Ticks. |
| **Spieler** | `getGameMode`, `setGameMode` | Standard-Spielmodus der Welt. |
| **System** | `isLoadOnStartup`, `setLoadOnStartup` | Soll die Welt beim Serverstart geladen werden? |
| | `isKeepSpawnInMemory`, `setKeepSpawnInMemory` | Spawn-Chunks im RAM behalten. |

# Nutzung im Builder

Wenn du eine Welt erstellst, kannst du Flags auch generisch über `WorldProperty` setzen:

```
import de.gilljan.gworld.data.properties.WorldProperty;


manager.createBuilder("Lobby")
    .property(WorldProperty.PVP, false)
    .property(WorldProperty.ANIMALS, false)
    .build();
```