Bedrock Server Manager

Release 3.10.7.dev50

DMedina559

Sep 28, 2026

GENERAL INFORMATION

Welcome to the Bedrock Server Manager documentation!

This documentation is designed to help you understand and use the Bedrock Server Manager effectively.

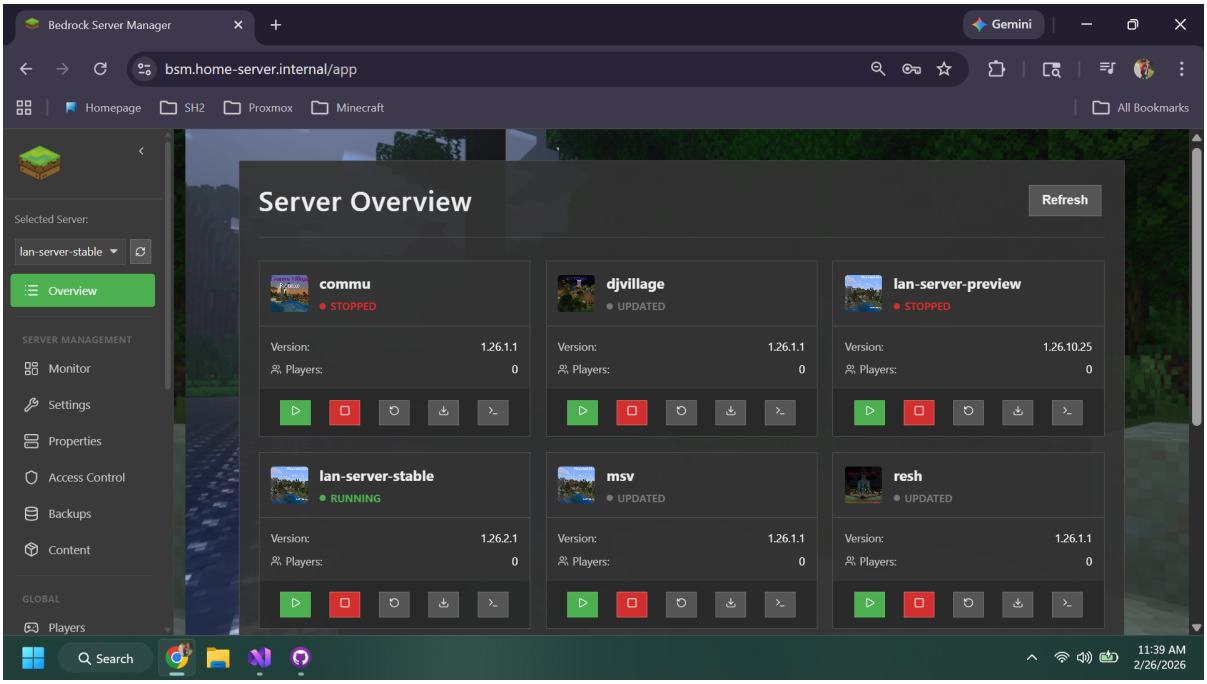
Browse through the sections below to find various information about Bedrock Server Manager, including general information, CLI commands, web interface, plugins, and developer documentation.

CHAPTER

ONE

INTRODUCTION

Bedrock Server Manager is a comprehensive python server designed for installing, managing, and maintaining Minecraft Bedrock Dedicated Servers with ease, compatible with Linux/Windows.



1.1 Features

- **Easiest Way to Manage Minecraft Bedrock Servers**: Easy to use manager for Minecraft Bedrock Dedicated Servers, with a simple and intuitive web interface.
- **Install New Servers**: Quickly set up a server with customizable options like version (LATEST, PREVIEW, or CUSTOM versions).
- **Update Existing Servers**: Seamlessly download and update server files while preserving critical configuration files and backups.
- **Backup Management**: Automatically backup worlds and configuration files, with pruning for older backups.
- **Server Configuration**: Easily modify server properties and the allow-list interactively.
- **Auto-Update**: Automatically update the server with a simple restart.

- **Content Management**: Easily import .mcworld/.mcpack files into your server.
- **Resource Monitoring**: View how much CPU and RAM your server is using.
- **Web Server**: Manage your Minecraft servers in your browser, even if you're on mobile!
- **Plugin Support**: Extend functionality with custom plugins that can listen to events, access the core app APIs, and trigger custom events.

1.2 Quick Start Guide

Bedrock Server Manager is also available as a docker image. See the *Docker Guide* for more information.

1.2.1 Step 1: Installation

> **Note**
>
> This app requires **Python 3.11** or later, and you will need **pip** installed.

First, install the main application package from PyPI:

```
pip install --upgrade bedrock-server-manager
```

(Optional) Install the CLI Client for CLI Management

To manage your servers from the command line, install the optional API client:

```
pip install --upgrade "bsm-cli"
```

This provides the `bsm-cli` command, which allows you to perform various tasks via the API.

See the *Installation Guide* for beta or development versions.

1.2.2 Step 2: Configure the Web Server

To get started with the web server, its recommended to run the setup command first:

```
bedrock-server-manager setup
```

This command will prompt you for the necessary configuration details, such as:

- **Data Directory**: The location where the application will store its data (default is `$HOME/bedrock-server-manager`).
- **Databse URL**: The URL for the database connection (default is `sqlite:///<data_dir>/bedrock_server_manager.db`).
- **Host**: The IP address the web server will listen on (default is `127.0.0.1`).
- **Port**: The port the web server will use (default is `11325`).
- **System Service**: Whether to install the web server as a system service (default is `no`).

If you choose not to run the setup command, default values will be used. This can be changed later by running the setup command.

1.2.3 Step 3: Run the Application

To start the web server, use the following command:

```
bedrock-server-manager web start
```

By default, the server listens on `127.0.0.1:11325`. Once running, you can access the web interface in your browser at this address.

Once the server is running, a one first-time setup will be required. This includes setting up an admin user account and configuring your first bedrock server.

> See the *Web Usage Guide* for more examples, like how to run the server on a different IP.

1.3 What's Next?

Bedrock Server Manager is a powerful tool for managing Minecraft Bedrock Dedicated Servers. To explore more about its capabilities, check out the following sections:

- *Web Usage*: Discover how to use the web interface for server management.
- *CLI Commands*: View what commands are available for the core application.
- *API Client CLI Commands*: View what commands are available in the `bsm-cli` package.
- *Plugins*: Explore how to extend the functionality of Bedrock Server Manager with custom plugins.
- *Changelog*: Stay updated with the latest changes and improvements in each release.
- *Troubleshooting*: Find solutions to common issues.
- Contributing: Find out how you can contribute to the project and help improve it.
- License: Understand the licensing terms under which Bedrock Server Manager is distributed.
- Join the Community: Connect with other users in the Bedrock Server Manager community on Reddit.

CHAPTER

TWO

CLI USAGE

For a complete list of commands, see *CLI Commands*.

> [!note] As of BSM 3.6.0, CLI commands have been migrated to the `bsm-api-client[cli]` package. Install with:
>
> `pip install --upgrade bsm-api-client[cli]`

2.1 Examples:

2.1.1 Start the server:

```
bedrock-server-manager web start
```

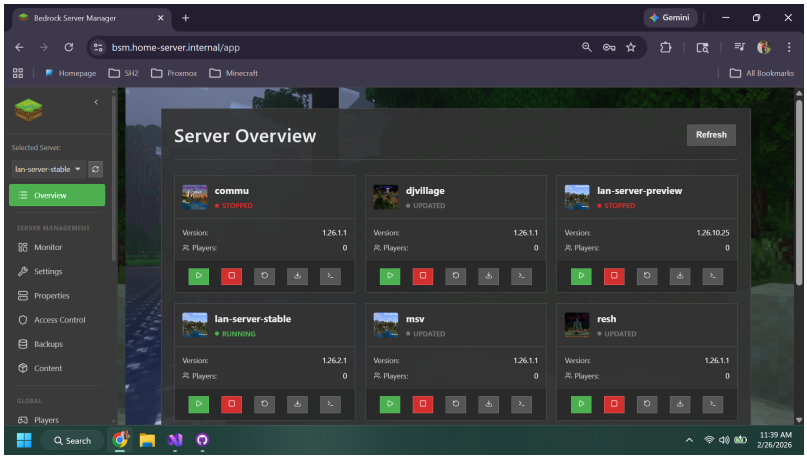
2.1.2 Generate password hash interactively

```
bedrock-server-manager generate-password
```

CHAPTER

THREE

WEB USAGE



Bedrock Server Manager includes a Web server you can run to easily manage your bedrock servers in your web browser, and is also mobile friendly!

With the web server you can:

- Install New Server
- Configure various server config files such as allowlist and permissions
- Start/Stop/Restart Bedrock server
- Update/Delete Bedrock server
- Monitor resource usage
- Install world/addons
- Backup and Restore all or individual files/worlds
- Manage Plugins

3.1 Hosts:

By Default Bedrock Server Manager will only listen to local host only interfaces `127.0.0.1`

To change which host to listen to start the web server with the specified host

Example: specify local host only ipv4:

```
bedrock-server-manager web start --host 127.0.0.1
```

Example: specify all ipv4:

```
bedrock-server-manager web start --host 0.0.0.0
```

You can also change the host by running the `setup` command, which will prompt you for the host to use.

```
bedrock-server-manager setup
```

3.1.1 Port:

By default Bedrock Server Manager will use port `11325`. This can be change with the `setup` command.

3.1.2 HTTP API:

An HTTP API is provided allowing tools like `curl` or `Invoke-RestMethod` to interact with server.

Obtaining a JWT token:

> **Note**
>
> The API relies on Bearer tokens for authentication using the `Authorization` header.

The API endpoints require authentication using a JSON Web Token (JWT). How: Obtain a token by sending a POST request to the `/auth/token` endpoint. Request Body: Include form data (`application/x-www-form-urlencoded`) with `username`, `password`, and an optional `remember_me` key (default is `false`). Setting `remember_me` to `true` extends the token expiration.

Response: On success, the API returns a JSON object containing the access_token:

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "bearer",
    "message": "Successfully authenticated."
}
```

Tokens expiration is configurable via web ui (default: 4 weeks).

`curl` Example (Bash):

Extract the token from the response. You can use tools like `jq` to parse the JSON output.

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" \
     -d "username=your_username&password=your_password&remember_me=true" \
     http://<your-manager-host>:<port>/auth/token
```

PowerShell Example:

Store the authentication token in a variable.

```
$body = @{ username = 'your_username'; password = 'your_password'; remember_me = $true }
$response = Invoke-RestMethod -Method Post -Uri "http://<your-manager-host>:<port>/auth/
↪token" -Body $body -ContentType 'application/x-www-form-urlencoded'
$token = $response.access_token
```

Using the API

Endpoints requiring authentication will need the obtained `access_token` included in the `Authorization` header as a Bearer token.

For requests sending data (like POST or PUT), set the Content-Type header to `application/json`.

Examples:

- Start/Stop server:

`curl` Example (Bash):

Using `-H "Authorization: Bearer <token>"` to pass the token.

```
curl -X POST -H "Authorization: Bearer <YOUR_ACCESS_TOKEN>" \
     http://<your-manager-host>:<port>/api/server/<server_name>/stop
```

PowerShell Example:

Using the previously saved authentication token in the Headers.

```
$headers = @{ 'Authorization' = "Bearer $token" }
Invoke-RestMethod -Method Post -Uri "http://<your-manager-host>:<port>/api/server/
↪<server_name>/stop" -Headers $headers
```

- Send Command:

`curl` Example (Bash):

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer <YOUR_ACCESS_
↪TOKEN>" \
     -d '{"command": "say Hello from API!"}' \
     http://<your-manager-host>:<port>/api/server/<server_name>/send_command
```

PowerShell Example:

```
$headers = @{ 'Content-Type' = 'application/json'; 'Authorization' = "Bearer $token" }
$body = @{ command = 'say Hello from API!' } | ConvertTo-Json
Invoke-RestMethod -Method Post -Uri "http://<your-manager-host>:<port>/api/server/
↪<server_name>/send_command" -Headers $headers -Body $body
```

CHAPTER

FOUR

ROLE ACCESS TABLE

Endpoint	Type	Admin	Moderator	User
`GET /`	HTML	Full Access	Full Access	Full Access
`GET /server/{server_name}/monitor`	HTML	Full Access	Full Access	Full Access
`GET /auth/login`	HTML	Public	Public	Public
`POST /auth/token`	API	Public	Public	Public
`GET /auth/logout`	API	Full Access	Full Access	Full Access
`POST /register/generate-token`	API	Full Access	No Access	No Access
`GET /register/{token}`	HTML	Public	Public	Public
`POST /register/{token}`	API	Public	Public	Public
`GET /account`	HTML	Full Access	Full Access	Full Access
`GET /api/account`	API	Full Access	Full Access	Full Access
`POST /api/account/theme`	API	Full Access	Full Access	Full Access
`GET /settings`	HTML	Full Access	No Access	No Access
`GET /api/settings`	API	Full Access	No Access	No Access
`POST /api/settings`	API	Full Access	No Access	No Access
`GET /api/themes`	API	Full Access	Full Access	Full Access
`POST /api/settings/reload`	API	Full Access	No Access	No Access
`GET /users`	HTML	Full Access	Read-only	No Access
`POST /users/create`	API	Full Access	No Access	No Access
`POST /users/{user_id}/delete`	API	Full Access	No Access	No Access
`POST /api/server/{server_name}/start`	API	Full Access	Full Access	No Access

continues on next page

Table 1 – continued from previous page

Endpoint	Type	Admin	Moderator	User
POST /api/server/{server_name}/stop	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/restart	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/send_command	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/update	API	Full Access	No Access	No Access
DELETE /api/server/{server_name}/delete	API	Full Access	No Access	No Access
GET /api/server/{server_name}/status	API	Full Access	Full Access	Full Access
GET /api/server/{server_name}/config_status	API	Full Access	Full Access	Full Access
GET /api/server/{server_name}/version	API	Full Access	Full Access	Full Access
GET /api/server/{server_name}/validate	API	Full Access	Full Access	Full Access
GET /api/server/{server_name}/process_info	API	Full Access	Full Access	Full Access
POST /api/players/scan	API	Full Access	Full Access	No Access
GET /api/players/get	API	Full Access	Full Access	No Access
POST /api/downloads/prune	API	Full Access	No Access	No Access
GET /api/servers	API	Full Access	Full Access	Full Access
GET /api/info	API	Public	Public	Public

continues on next page

Table 1 – continued from previous page

Endpoint	Type	Admin	Moderator	User
POST /api/players/add	API	Full Access	Full Access	No Access
GET /server/{server_name}/backup	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/backup/select	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/restore	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/restore/{restore_type}/select_file	HTML	Full Access	Full Access	No Access
POST /api/server/{server_name}/restore/select_backup_typ	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/backups/prune	API	Full Access	Full Access	No Access
GET /api/server/{server_name}/backup/list/{backup_type}	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/backup/action	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/restore/action	API	Full Access	Full Access	No Access
GET /server/{server_name}/install_world	HTML	Full Access	No Access	No Access
GET /server/{server_name}/install_addon	HTML	Full Access	No Access	No Access
GET /api/content/worlds	API	Full Access	Full Access	No Access
GET /api/content/addons	API	Full Access	Full Access	No Access

continues on next page

Table 1 – continued from previous page

Endpoint	Type	Admin	Moderator	User
POST /api/server/{server_name}/world/install	API	Full Access	No Access	No Access
POST /api/server/{server_name}/world/export	API	Full Access	No Access	No Access
DELETE /api/server/{server_name}/world/reset	API	Full Access	No Access	No Access
POST /api/server/{server_name}/addon/install	API	Full Access	No Access	No Access
GET /plugins	HTML	Full Access	No Access	No Access
GET /api/plugins	API	Full Access	No Access	No Access
POST /api/plugins/trigger_event	API	Full Access	No Access	No Access
POST /api/plugins/{plugin_name}	API	Full Access	No Access	No Access
PUT /api/plugins/reload	API	Full Access	No Access	No Access
GET /api/downloads/list	API	Full Access	Full Access	No Access
GET /install	HTML	Full Access	No Access	No Access
POST /api/server/install	API	Full Access	No Access	No Access
GET /server/{server_name}/configure_propert	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/configure_allowli	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/configure_permiss	HTML	Full Access	Full Access	No Access
GET /server/{server_name}/configure_service	HTML	Full Access	No Access	No Access
POST /api/server/{server_name}/properties/set	API	Full Access	Full Access	No Access

continues on next page

Table 1 – continued from previous page

Endpoint	Type	Admin	Moderator	User
GET /api/server/{server_name}/properties/get	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/allowlist/add	API	Full Access	Full Access	No Access
GET /api/server/{server_name}/allowlist/get	API	Full Access	Full Access	No Access
DELETE /api/server/{server_name}/allowlist/remove	API	Full Access	Full Access	No Access
PUT /api/server/{server_name}/permissions/set	API	Full Access	Full Access	No Access
GET /api/server/{server_name}/permissions/get	API	Full Access	Full Access	No Access
POST /api/server/{server_name}/service/update	API	Full Access	No Access	No Access
GET /setup	HTML	Public	Public	Public
POST /setup	API	Public	Public	Public
GET /api/tasks/status/{task_id}	API	Full Access	Full Access	Full Access
GET /api/panorama	API	Public	Public	Public
GET /api/server/{server_name}/world/icon	API	Full Access	Full Access	Full Access
GET /favicon.ico	API	Public	Public	Public
GET /{full_path:path}	HTML	Full Access	Full Access	Full Access

CHAPTER

FIVE

PLUGIN SUPPORT

Bedrock Server Manager 3.4.0 features a powerful plugin system that allows you to extend and customize its functionality. Whether you want to add new automations, integrate with other services, or introduce custom server management logic, plugins provide the framework to do so.

Key Capabilities:

- **Event Hooks:** Plugins can "listen" to various events within BSM (e.g., before a server starts, after a backup completes) and execute custom code in response.
- **API Access:** Plugins have safe access to core BSM functions, allowing them to perform actions like starting/stopping servers, sending commands, and more.
- **Custom Events:** Plugins can define and trigger their own events, enabling complex communication and collaboration between different plugins.

Managing Plugins:

You can manage your plugins directly from the Web Server:

- Enable/Disable Plugins
- Reload Plugins
- View Plugin Information such as version

Developing Plugins:

To learn how to create your own plugins, please refer to the comprehensive:

Plugin Documentation

This documentation covers everything from creating your first plugin, understanding the `PluginBase` class, using event hooks and the plugin API, to advanced topics like custom inter-plugin events.

CHAPTER

SIX

TROUBLESHOOTING

When you encounter an issue, please follow these steps to diagnose and resolve the problem. Following this guide helps you solve the issue quickly and ensures you have all the necessary information if you need to file a bug report.

6.1 Initial Checks & Common Fixes

Start with these simple steps, as they resolve many common problems.

6.1.1 1. Update to the Latest Version

The issue you're facing may have already been fixed in a newer release. Ensure both the Bedrock Server Manager and the `bsm-cli` (if you use it) are up to date.

You can check your version in the BSM web UI and compare it with the latest release on GitHub.

6.1.2 1.1. Check the Changelog for Breaking Changes

If you recently upated your BSM instance, read the changelog!

Major updates can introduce "breaking changes" that alter how BSM works. Before going further, review the changelog for the version you just installed.

> **Tip**
>
> The changelog often contains important warnings and instructions on how to migrate your setup or fix issues caused by an update. Look for BREAKING CHANGE notices relevant to your setup (e.g., Web UI, CLI, Plugins).

6.1.3 2. Clear Caches

Sometimes, outdated cache files can cause unexpected behavior. Run the cleanup command to clear the `__pycache__` directories.

```
bedrock-server-manager cleanup --cache
```

6.1.4 3. Isolate the Problem: Disable Plugins

A plugin could be the source of the issue. To check if a plugin is causing the problem, disable all plugins through the BSM web UI, restart the manager, and see if the issue persists. If the problem disappears, re-enable plugins one by one until you find the one causing the conflict.

6.2 Gathering More Information

If the initial steps don't resolve the issue, the next step is to gather more detailed logs.

6.2.1 4. Set Log Level to Debug

Enabling `DEBUG` level logging provides detailed operational information that is crucial for diagnosing complex problems.

1. Go to **Settings** in the BSM web UI.
2. Find the **Log Level** setting and change it to `DEBUG`.
3. Save the settings and restart the Bedrock Server Manager.
4. Reproduce the issue. The `.logs/bedrock_server_manager.log` file will now contain detailed trace information.

6.3 Reporting an Issue

If you've followed the steps above and are confident you've found a new bug, we encourage you to report it.

6.3.1 5. Search for Open Issues

Before creating a new report, please **search the open issues** on GitHub to see if someone has already reported the same problem. If you find a similar issue, you can add any additional information you've gathered to the existing thread.

6.3.2 6. Create a New Bug Report

If no existing issue describes your problem, please **open a new bug report**.

Important Issue Reporting Guidelines:

- **Strict Template Usage:** You **must** use the provided templates (Bug Report or Feature Request). Issues opened without a template or with deleted sections will be **closed without review**.
- **Fill All Fields:** We cannot debug issues without your **Environment Details** (OS, BSM Version) and **Logs**. If a section isn't relevant, mark it as "N/A" rather than removing it.
- **CLI Issues:** If your issue is related to the Command Line Interface, please post it in the **bsm-api-client repository**.
- **Attach Logs:** Copy and paste the debug logs gathered in Step 4. Please remove any sensitive information before posting.

6.4 For Developers

6.4.1 Consider Contributing a Fix

If you are a developer and have identified the source of the bug, consider contributing a fix! Pull requests are always welcome. Fork the repository, create a new branch for your fix, and submit a pull request for review.

6.5 Common Issues & Important Notes

This section contains solutions for common problems, often related to major updates.

- **Updates and Graceful Shutdowns**: As of version 3.6.0, BSM automatically attempts to shut down all running servers and unload plugins gracefully when the main application stops. While the old advice to "stop servers before updating" is less critical now, it remains a good practice, especially if upgrading from a version before 3.6.0.
- **Bedrock Updates**: Sometimes Mojang releases new versions of the server that requires new server properties. Before BSM version 3.10.0 BSM would preserve the original server properties file when updating the server. As of version 3.10.0, BSM will now **merge any new server properties with your existing ones**. This means that if Mojang adds a new property, it will be added to your existing `server.properties` file instead of being overwritten. If you updated your servers before 3.10.0, you may need to manually update your `server.properties` file to include any new properties introduced in the latest Bedrock server version using the Web UI.
- **Web UI Login Fails After v3.5.0 Update**: The migration from Flask to FastAPI in v3.5.0 required a new password hashing system (`bcrypt`). If you updated from a version prior to 3.5.0, **you must regenerate your password hash and auth tokens** to log in.
- **CLI Commands Don't Work After v3.10.0 Update**: In v3.10.0, most CLI commands were moved to the `bsm-cli` package. The core `bedrock-server-manager` package no longer handles most command-line actions directly.
  - Install the new CLI with: `pip install bedrock-server-manager[cli]`
  - Use the new command: `bsm-cli <command>`
- **Slow Web UI (3.6+)**: Using the default database has been observed to cause slow loadding times for the web server on some systems, if you experience this issue, consider using an external database such as `MySQL` or `PostgreSQL`.

6.6 Platform Information

6.6.1 Windows Service Integration

Running BSM as a service on Windows has specific requirements:

- **Administrator Privileges**: You must run your Command Prompt or PowerShell session **as an Administrator** to create, configure, or manage Windows services.
- **Alternative to Services (No Admin)**: If you don't have administrative rights, you can manually create a task in the **Windows Task Scheduler** to launch the BSM web server on startup.
- **Use Python from python.org**: It is **highly recommended** to use a Python installer from python.org. The version from the Microsoft Store is known to cause critical file-locking issues that can prevent BSM and other Python Environments from working correctly. ~~* **Set the "Log On As" Account**: A common reason services fail to start is incorrect permissions. After creating a service, go to the Services app (`services.msc`), find the service, and in its **Properties**, go to the **Log On** tab. Change "Log on as:" from "Local System account" to **"This account"** and enter your local Windows username and password. This ensures the service can access server files, backups, and content in your user directory.~~ (As of v3.6.0, this is no longer required as credentials are entered at creation.)

6.6.2 Linux Service Integration

- **Enable Startup on Boot (`linger`)**: Services are created with the `--user` flag by default which only run while you are logged in. To enable your BSM service to start automatically on boot and stay running, you must enable lingering for your user account:

```
sudo loginctl enable-linger $(whoami)
```

~~* **Environment Variables**: `systemd` user services do not inherit environment variables from your shell (`~/.bashrc`, etc.). You must handle them manually: 1. Create a separate environment file (e.g., `~/.config/bsm/bsm.env`) containing your `BEDROCK_SERVER_MANAGER_*` variables. 2. **Secure the file.** Since this file contains sensitive credentials, restrict its permissions so that only your user account can read and write to it: `bash chmod 600 ~/.config/bsm/bsm.env` 3. Add the `EnvironmentFile=` directive to your `systemd` service file to load the variables.~~ (As of v3.6.0, a database is used to store various configuration, including environment variables, so this is no longer necessary.)

6.6.3 Tested Systems

- Debian 12 (Bookworm)
- Ubuntu 24.04
- Windows 11 24H2
- Windows Subsystem for Linux (WSL2)

CHAPTER

SEVEN

INSTALLATION & UPDATES

Bedrock Server Manager streamlines the process of deploying new Bedrock Dedicated Server instances and keeping them up to date.

7.1 Installing a New Server

You can install multiple server instances on the same machine, each with its own configuration.

7.1.1 Installation Steps

1. **Navigate to Install**: Click **Install New Server** from the dashboard.
2. **Server Name**: Enter a unique name for your server (e.g., `Survival_World`, `Creative_Lobby`). This name will be used for the server's directory and identification.
3. **Server Version**:
   - **LATEST**: Automatically downloads the most recent stable version from Mojang.
   - **PREVIEW**: Installs the latest beta/preview build for testing new features.
   - **CUSTOM**: Allows you to select a custom server zip file that you have manually placed in the `downloads/custom` directory.
   - **SPECIFIC VERSION**: Enter a specific released version (e.g., `1.21.114.1`, `1.21.130.22-preview`).
4. **Install**: Click the **Install Server** button.
   - BSM will download the server software (if not already cached) and extract it to a new directory in `servers/<server_name>`.
   - Progress is tracked in real-time.

7.1.2 Post-Installation Wizard

Once the files are installed, BSM guides you through the initial setup:

1. **Properties**: Configure basic settings like Game Mode, Difficulty, and Port.
2. **Allowlist**: (Optional) Add your own Gamertag so you can join immediately.
3. **Permissions**: (Optional) Set yourself as an Operator.
4. **Service Settings**:
   - **Auto Start**: Check this if you want the server to start automatically when BSM starts (or the computer reboots).
   - **Auto Update**: Check this to enable automatic updates for this instance on start up.

5. **Finish**: Click **Save & Start Server** to boot up your new world.

7.2 Updating Servers

Mojang frequently releases updates for Minecraft Bedrock. BSM helps you stay current without losing data.

7.2.1 Manual Update

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Settings** page.
3. Click the **Update Server** button.
4. BSM will:
    - Stop the server safely (if it was running).
    - Perform a full backup (if enabled).
    - Download and extract the new binaries.
    - Restore your configuration files.
    - Restart the server (if it was running).

7.2.2 Automatic Updates

If **Auto Update** is enabled for a server instance:

- BSM checks for updates durring the bedrock server startup.
- If a new version is detected, it will perform the update process automatically.

Tip: Enable the `Update Before Start` default plugin to have BSM automatically check for updates each time the server starts. (Must have `autoupdate: true` set in the server configuration).

7.3 Custom Versions

You may want to run an older version of the server or a specific beta build.

1. Download the desired Bedrock Server `.zip` file manually.
2. Place it in the `downloads/custom` folder in your BSM installation directory.
3. During installation, select **CUSTOM** as the version.
4. Choose your zip file from the list.

CHAPTER

EIGHT

SERVER MANAGEMENT

Bedrock Server Manager (BSM) provides a comprehensive set of tools to manage the lifecycle and runtime of your Minecraft Bedrock Dedicated Servers. This section covers starting, stopping, restarting, sending commands, and monitoring server resources.

8.1 Basic Operations

The dashboard is the central hub for managing your servers. From here, you can perform basic operations on any configured server instance.

8.1.1 Starting a Server

To start a server:

1. Navigate to the **Dashboard** page.
2. Click the **Start** button on the server card.
3. The server status indicator will change from `Stopped` to `Starting`, and finally to `Running` once the server is fully operational.

8.1.2 Stopping a Server

To stop a running server:

1. Navigate to the **Dashboard** page.
2. Click the **Stop** button on the server card.
3. BSM will attempt to gracefully shut down the server.

Graceful Shutdown: BSM sends the `stop` command to the server console, allowing it to save chunks and player data before terminating, if that fails, it will forcibly kill the process after a timeout period.

8.1.3 Restarting a Server

To restart a server:

1. Navigate to the **Dashboard** page.
2. Click the **Restart** button on the server card.
3. The server will shut down and immediately start up again. This is useful for applying configuration changes or refreshing the server state.

8.2 Console Interaction

You can send commands directly to the server console without needing SSH access to the host machine.

1. Navigate to the **Dashboard** page.
2. Click the **Send Command** button on the server card.
3. A prompt will appear asking for the command.
4. Type your command (e.g., `say Hello World`, `op <player_name>`, `gamerule doDaylightCycle false`) and click **OK**.
5. The command is sent directly to the server.

Note: Do not include the leading `/` for commands sent via the console, although most commands will work with or without it.

8.3 Resource Monitoring

The **Monitor** page provides real-time visibility into the performance of your server instances.

8.3.1 Viewing Usage Stats

2. Select the server from the dropdown menu in the web dashboard.
3. Navigate to the **Monitor** page from the web dashboard.
4. Real-time metrics include:
    - **PID**: The process ID of the server instance.
    - **CPU Usage**: Percentage of CPU cores used by the server process.
    - **Memory Usage**: RAM consumed by the server (e.g., "350 MB").
    - **Uptime**: How long the server has been running.
    - **Server Log**: A live feed of the server console output, allowing you to see player activity, errors, and other events as they happen.
    - **Quick Actions**: Buttons to start, stop, restart, or send commands to the server directly from the monitor page.

8.3.2 Auto-Refresh

The monitor page automatically refreshes data periodically (typically every few seconds) to give you an up-to-date view of your infrastructure.

CHAPTER

NINE

SERVER CONFIGURATION

Configuring your Bedrock server correctly is crucial for gameplay, security, and performance. BSM provides an intuitive interface to manage `server.properties`, `allowlist.json`, and `permissions.json` directly from the web UI.

9.1 Server Properties

The **Server Properties** page allows you to modify the core configuration of the Minecraft server. This includes settings like the server name, game mode, difficulty, and port.

9.1.1 Accessing Properties

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Properties** page.
3. From here, you can view and edit various server properties, or even add custom properties.

9.1.2 Modifying a Setting

1. Locate the setting you wish to change (you can also search for specific properties using the search bar).
2. Edit the value in the input field.
    - **Text/Number**: Type the new value.
    - **Toggle**: Click the switch to enable or disable the option.
3. Navigate to the bottom of the page and click **Save Properties**.
4. **Important**: Most changes require a **Server Restart** to take effect.

9.2 Allowlist Management

The **Allowlist** (formerly whitelist) controls which players are permitted to join your server.

Note: The allowlist feature requires the bedrock server must have allowlist enabled in the `server.properties` file (`allow-list=true`).

9.2.1 Managing Players

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Access Control** page, and then click on the **Allowlist** tab.
3. **Add Player**:

- Enter a player's Gamertag.
- Optionally, enable the `Ignore Player Limit` checkbox to allow these players to join even if the server is full.
- Click **Add**.

4. **Remove Player**:
    - Find the player in the list at the bottom of the page.
    - Click the **Remove** button next to their name.

9.3 Permissions

The **Permissions** page allows you to assign operator roles and other permission levels to players.

Note: Only players who have joined any BSM server at least once will appear in the permissions list. (Assumes player scanning is enabled (default)).

9.3.1 Permission Levels

- **Visitor**: Can explore but cannot interact with blocks or entities.
- **Member**: Standard player permissions (break/place blocks).
- **Operator**: Admin permissions (use commands, manage server).

9.3.2 Managing Permissions

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Access Control** page and then click on the **Permissions** tab.
3. **Add/Update Player**:
    - Find the player in the list.
    - Select the desired permission level from the dropdown.
    - Click **Save Permissions**.

9.3.3 Manually Add Players

If a player has not joined the server yet, you can manually add them to the permissions list:

Note: You must know the player's XUID to add them manually. You can find a player's XUID using third-party tools.

1. Navigate to the **Access Control** page and then click on the **Permissions** tab.
2. At the top of the page enter the player's Gamertag and XUID in the respective input fields.
3. Select the desired permission level from the dropdown.
4. Click **Add Player**.

9.3.4 Other Operations:

- From the Permissions page you can also trigger a player scan to update the list of players who have joined the server at least once (or currently online).

CHAPTER

TEN

BACKUP & RESTORE

Data loss prevention is a critical aspect of server management. Bedrock Server Manager includes a robust backup and restore system that allows you to protect your worlds and configuration files.

10.1 Creating Backups

You can create backups manually at any time from the web interface.

10.1.1 Types of Backups

- **World Backup**: Archives the `worlds/` directory, saving all map data, player inventories, and structures.
- **Full Backup**: Archives both the world data and all configuration files (`server.properties`, `allowlist.json`, `permissions.json`).
- **Config Backup**: Backs up specific configuration files individually.

10.1.2 How to Create a Backup

1. Navigate to the **Backups** page for your server.
2. **For a World Backup**: Click **+ New World Backup**.
3. **For a Full Backup**: Click **Backup All Now**.
4. **For Config Backups**:
    - Same as world backup, but select the specific config type (Properties, Allowlist, Permissions) to backup.

Note: Backups are stored in the server's `backups/` directory, organized by date and type.

10.2 Restoring Backups

If something goes wrong, you can easily restore your server to a previous state.

10.2.1 Restore Process

1. Navigate to the **Backups** page.
2. BSM will display a list of available backups sorted by date and type.
3. Find the desired backup and click the **Restore** button next to it.
4. **Confirm the warning**: Restoring will **overwrite** the current files on the server. This action cannot be undone unless you made a fresh backup just before restoring.

10.2.2 Restore All

The **Restore All** function attempts to restore the most recent backup for *all* categories (World, Properties, Allowlist, Permissions). Use this with caution when performing a full server rollback.

10.3 Automated Backups

BSM supports automated backups to ensure you never lose progress.

1. Default behavior often includes a "Backup on Start" option to ensure a safe point before a new session begins.
2. Another option "Backup before Update" is enabled to create a backup before applying server updates.

10.4 Managing Backups

While the web interface allows you to create and restore backups, you may occasionally need to manage the backup files themselves (e.g., to free up disk space).

- Backups are standard `.mcworld` or plain `json/properties` files.
- They are located in the `backups/<server_name>` folder.
- You can manually delete old backup files using a file manager or the command line if the default pruning options are insufficient.

CHAPTER

ELEVEN

CONTENT MANAGEMENT

Bedrock Server Manager allows you to easily manage the content of your server, including uploading new worlds and installing addons (resource and behavior packs).

11.1 World Management

You can import existing worlds, export your current world, or reset the server to a fresh state.

11.1.1 Importing a World

To use a custom world (e.g., a downloaded map or a backup from another server):

1. **Upload the World**:
    - Place your world file (`.mcworld`) into the `content/worlds` directory on the server.
    - (Enable the content uploader plugin to easily upload content files).
2. Select the server from the dropdown menu in the web dashboard.
3. Navigate to the **Content** page and then click on the **Worlds** tab.
4. Locate the file in the file list.
5. Click the **Install** button next to the file.
6. **Warning**: This action will replace the currently active world on the server. Ensure you have backed up your current world if you wish to save it.

11.1.2 Exporting the Current World

To save a copy of your current world for download or transfer:

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Content** page and then click on the **Worlds** tab.
3. Click the **Export World** button.
4. BSM will compress the active world directory into a `.mcworld` file.
5. The file will be exported to the `content/worlds` directory for you to import into another server.

11.1.3 Resetting the World

To start fresh with a newly generated world:

1. Select the server from the dropdown menu in the web dashboard.
2. Navigate to the **Content** page and then click on the **Worlds** tab.
3. Click the **Reset World** button.
4. Confirm the action.
5. BSM will delete the current `worlds/` directory.
6. Upon the next server start, the Bedrock Dedicated Server software will generate a new world based on the `level-seed` defined in `server.properties`.

11.2 Addons (Resource & Behavior Packs)

BSM simplifies the process of installing, managing, and configuring addons to enhance your server.

11.2.1 Installing Addons

1. **Upload the Addon**:
   - Place your addon file (`.mcpack`/`.mcaddon`) into the `content/addons` directory.
   - (Enable the content uploader plugin to easily upload content files).
2. Select the server from the dropdown menu in the web dashboard.
3. Navigate to the **Content** page and then click on the **Addons** tab.
4. Locate the file in the **Available Addons** list.
5. Click the **Install** button.

11.2.2 Managing Installed Addons

Once addons are installed, you can manage their status and load order:

1. Navigate to the **Content** page and click the **Manage Installed Addons** button.
2. A window will appear with tabs for **Behavior Packs** and **Resource Packs**.
3. **Enabling/Disabling**: You can toggle addons on or off. Disabled addons remain installed on the server but are temporarily deactivated in the world configuration.
4. **Reordering**: You can drag and drop the packs to change their load order (packs higher in the list load first and take priority over those below them). Click **Save Order** to apply the changes.
5. **Removing**: You can click the delete/remove icon next to an installed pack to permanently remove it from the server.

11.2.3 How It Works

When you install an addon, BSM performs the following actions:

1. **Extraction**: Unzips the package.
2. **Placement**: Moves resource packs to `resource_packs/` and behavior packs to `behavior_packs/`.
3. **Activation**: Automatically updates `world_behavior_packs.json` and `world_resource_packs.json` to register the new content with the server.

CHAPTER

TWELVE

APPLICATION SETTINGS

Beyond managing individual servers, Bedrock Server Manager has global settings to control the application itself, manage users, and handle plugins.

12.1 Global Settings

The **Settings** page controls how BSM behaves.

12.1.1 Key Categories

- **Paths**: Defines where servers, backups, content, and downloads are stored on the host machine.
  - *Note: Changing paths requires moving existing data manually.*
- **Security**: Settings related to token timeouts and authentication.
- **System**: Logging levels and other backend behaviors.

12.1.2 Modifying Settings

1. Navigate to **BSM Settings**.
2. Locate the setting you wish to change.
3. Update the value.
4. Changes are auto-saved.
5. **Reload**: If you edited the configuration, click the **Reload from File** button to refresh the application state.

Note: You can also add custom settings here that may be used by plugins. Custom keys will automatically be saved under the `custom` tree.

12.2 User Management

BSM supports a multi-user environment, allowing you to grant access to other team members.

12.2.1 Inviting Users

New users are added via an invitation system.

1. Navigate to the **Users** page.
2. Click the **Invite User** button.
3. Select the **Role** for the new user (e.g., Admin, Moderator, User).

4. Click **Generate Token**.
5. A unique registration link will be generated (valid for 24 hours).
6. Copy this link and send it to the person you wish to invite. They will be prompted to create their own username and password.

Warning: The registration link will only be shown once. If you lose it, you will need to generate a new one.

12.2.2 User Roles

- **Admin**: Has full access to the system, including managing other users, installing plugins, and changing global settings.
- **Moderator**: Typically has access to manage servers (Start/Stop, Backups) but cannot alter global system settings or manage other users.
- **User**: Usually has read-only access to view server statuses and data without making changes.

12.2.3 Managing Existing Users

Admins can manage existing accounts from the user table:

- **Change Role**: Update a user from Moderator to Admin (or vice versa) using the dropdown.
- **Disable/Enable**: Temporarily revoke access without deleting the account.
- **Delete**: Permanently remove the user account.

12.2.4 The "One Admin" Rule

To prevent the system from becoming inaccessible, BSM enforces a strict safety requirement:

- **You cannot disable, delete, or demote the last active Administrator.**
- If you wish to remove the current admin, you must first promote another user to the Admin role.

12.3 Plugin Management

Plugins extend the functionality of BSM.

12.3.1 Managing Plugins

1. Navigate to the **Plugins** page.
2. **View**: See a list of installed plugins and their versions.
3. **Enable/Disable**: Toggle the switch next to a plugin to turn it on or off.
    - *Note*: Disabling a plugin prevents it from loading on the next server restart.
4. **Reload**: Click **Reload Plugins** to apply changes.

12.3.2 Installing Plugins

1. Download a BSM plugin.
2. Place it in the `plugins/` directory of your BSM installation.
3. Go to the **Plugins** page.
4. Enable the new plugin.

12.4 Global Player Management

The **Players** page provides a view of all players that have been discovered across all server instances. This list is used for permission management.

12.4.1 Manually Adding Players

If a player has not yet joined any server, you can add them manually:

Note: You must know the player's exact Gamertag and XUID to add them manually. This information can be obtained through third-party services.

1. Navigate to the **Players** page.
2. Enter the player's Gamertag and XUID in a :,:,… format.
3. Click **+ Add/Update**.

12.4.2 Trigering Player Scans

BSM automatically scans for players when a server starts, but you can also trigger a manual scan:

1. Navigate to the **Players** page.
2. Click the **Scan Logs** button to refresh the list of players who have joined any server at least once (or are currently online).

CHAPTER

THIRTEEN

DEFAULT PLUGINS

Bedrock Server Manager comes with a set of default plugins that handle many standard lifecycle automation tasks. Most of these plugins are enabled by default, but you can disable them individually if desired, or even override them with custom plugins.

13.1 Lifecycle Automation

These plugins automate server actions based on events like startup or shutdown.

13.1.1 Autostart Servers

- **Purpose**: Automatically starts servers that have the `autostart: true` setting enabled in their Service Configuration.
- **Trigger**: Application startup (when the BSM Web Server itself starts).
- **Key Behavior**: Iterates through all configured servers and boots them up if the flag is set.

13.1.2 Backup on Start

- **Purpose**: Ensures a safety backup exists before a server session begins.
- **Trigger**: On bedrock server start (`before_server_start`).
- **Key Behavior**: Performs a full backup of the server data. If the backup fails, the plugin logs a warning but allows the server to start (to prevent downtime loops).

13.1.3 Update Before Start

- **Purpose**: Checks for Bedrock Dedicated Server updates automatically.
- **Trigger**: On bedrock server start (`before_server_start`).
- **Key Behavior**: If `autoupdate: true` is set for the server, it checks Mojang's servers for a new version. If found, it updates the server binary before launching.

13.2 Notifications & Safety

These plugins provide feedback to players and ensure smoother operations.

13.2.1 Server Lifecycle Notifications

- **Purpose**: Warns players about impending server events.
- **Trigger**: Before major bedrock server events (`before_server_stop`, `before_delete_server_data`, `before_server_update`).
- **Key Behavior**:
  - **Shutdown**: Sends a "Server is stopping in X seconds..." message to the in-game chat and waits (default 10s) before killing the process.
  - **Delete**: Sends a critical warning if data is about to be erased.

13.2.2 World Operation Notifications

- **Purpose**: Warns players about major world changes.
- **Trigger**: Before major world operations (`before_world_export`, `before_world_import`, `before_world_reset`).
- **Key Behavior**: Sends chat messages like "World import starting... Current world will be replaced."

13.2.3 Auto Reload Config

- **Purpose**: Hot-reloads configuration files.
- **Trigger**: After configuration file changes (`after_allowlist_change`, `after_permission_change`).
- **Key Behavior**: If you modify the Allowlist or Permissions via the Web Server while the server is running, this plugin automatically sends the `allowlist reload` or `permission reload` command to the server console, so changes take effect instantly.

13.3 Content Tools

13.3.1 Content Uploader

- **Purpose**: Provides a web interface for uploading `.mcworld`, `.mcpack`, and `.mcaddon` files.
- **Access**: Adds a new route `/content/upload`.
- **Key Behavior**: Accepts file uploads and places them in the configured imports directory (`content/worlds` or `content/addons`), making them available for installation via the Content Management page.

13.3.2 Download Page

- **Purpose**: Provides a web interface for downloading BSM content like worlds, addons, and backups.
- **Access**: Adds a new entry in the web dashboard, and additional route `/api/download_page/download` to handle file downloads.
- **Key Behavior**: Lists available content/backup files and allows users to download them directly from the web interface, which is especially useful for remote server management.

CHAPTER

FOURTEEN

BEDROCK-SERVER-MANAGER

A comprehensive Web Server for managing Minecraft Bedrock Dedicated Servers.

Usage

```
bedrock-server-manager [OPTIONS] [COMMAND] [ARGS]...
```

Options

--config-dir <config_dir>
: Override the configuration directory.

--data-dir <data_dir>
: Override the application data directory.

--db-url <db_url>
: Override the database URL connection string.

--log-level <log_level>
: Set the logging level.

  Options
  DEBUG | INFO | WARNING | ERROR | CRITICAL

-v, --version
: Show the version and exit.

14.1 cleanup

Cleans up generated application files, such as logs and Python bytecode cache.

This maintenance command helps keep the project directory and application data areas tidy by removing temporary or accumulated files. At least one of the cleanup flags (*–cache* or *–logs*) must be provided for the command to perform an action.

Log Cleanup Behavior:
: When *–logs* is specified, this command will delete `.log` files from the configured log directory. **Crucially, it preserves the single most recent log file**, ensuring that the latest operational logs are not accidentally deleted.

Raises:

click.Abort: If log cleaning (*–logs*) is requested but no valid log
: directory can be determined (neither specified via *–log-dir* nor found in settings).

Usage

```
bedrock-server-manager cleanup [OPTIONS]
```

Options

-c, --cache

Clean up Python bytecode cache files (__pycache__).

-l, --logs

Clean up application log files (.log).

--log-dir <log_dir_override>

Override the default log directory from settings.

14.2 database

Database management commands.

Usage

```
bedrock-server-manager database [OPTIONS] COMMAND [ARGS]...
```

14.2.1 backup

Backups all database records to a JSON file.

Usage

```
bedrock-server-manager database backup [OPTIONS]
```

Options

-o, --output <output>

The output path for the JSON backup file. If not provided, it will be saved in the configured backups directory.

14.2.2 downgrade

Downgrades the database to a previous version.

Usage

```
bedrock-server-manager database downgrade [OPTIONS]
```

Options

-r, --revision <revision>

The revision to downgrade to (e.g., -1 for previous, or a specific revision ID).

14.2.3 restore

Restores database records from a JSON file. Wipes existing data!

Usage

```
bedrock-server-manager database restore [OPTIONS]
```

Options

-i, --input <input>

Required The input path for the JSON backup file.

14.2.4 status

Displays current database version and migration status.

Usage

```
bedrock-server-manager database status [OPTIONS]
```

14.2.5 upgrade

Upgrades the database to the latest version, stamping it if necessary.

Usage

```
bedrock-server-manager database upgrade [OPTIONS]
```

Options

-y, --yes

Skip confirmation prompts and automatically backup.

14.3 reset-password

Resets the password for a user.

Usage

```
bedrock-server-manager reset-password [OPTIONS] USERNAME
```

Arguments

USERNAME

Required argument

14.4 service

Manages the Bedrock Server Manager Web UI application's service.

This group of commands allows you to manage its integration as an OS-level system service for features like automatic startup.

Usage

```
bedrock-server-manager service [OPTIONS] COMMAND [ARGS]...
```

14.4.1 configure

Configures the OS-level system service for the Web UI application.

This command allows setting up the Web UI to run as a system service, enabling features like automatic startup on boot/login.

If run without any specific configuration flags (*–setup-service*, *–enable-autostart*), it launches an interactive wizard (`interactive_web_service_workflow()`) to guide the user.

If flags are provided, it applies those settings directly. The command respects system capabilities (e.g., won't attempt service setup if a service manager isn't available or *pywin32* is missing on Windows).

Calls internal helpers:

- `interactive_web_service_workflow()` (if no flags)
- `_perform_web_service_configuration()` (if flags are present)

Usage

```
bedrock-server-manager service configure [OPTIONS]
```

Options

--setup-service

Create or update the system service file for the Web UI.

--enable-autostart, --no-enable-autostart

Enable or disable Web UI service autostart.

-s, --system

Configure as a system-wide service (Linux only, requires sudo).

Default
`False`

--username <username>

Username to run the Windows service as.

--password <password>

Password for the user.

14.4.2 disable

Disables the Web UI system service from starting automatically.

Configures the OS service for the Web UI to not start automatically.

Requires a supported service manager (checked by decorator).

Calls API: *disable_web_ui_service()*.

Usage

```
bedrock-server-manager service disable [OPTIONS]
```

Options

-s, --system

Disable a system-wide service (Linux only, requires sudo).

Default
`False`

14.4.3 enable

Enables the Web UI system service for automatic startup.

Configures the OS service for the Web UI (systemd on Linux, Windows Service on Windows) to start automatically when the system boots or user logs in.

Requires a supported service manager (checked by decorator).

Calls API: *enable_web_ui_service()*.

Usage

```
bedrock-server-manager service enable [OPTIONS]
```

Options

-s, --system

Enable a system-wide service (Linux only, requires sudo).

Default
`False`

14.4.4 remove

Removes the Web UI system service definition from the OS.

Danger

This is a destructive operation. The service definition will be deleted from the system.

Prompts for confirmation before proceeding. Requires a supported service manager (checked by decorator).

Calls API: *remove_web_ui_service()*.

Usage

```
bedrock-server-manager service remove [OPTIONS]
```

Options

-s, --system

Remove a system-wide service (Linux only, requires sudo).

Default
`False`

14.4.5 status

Checks and displays the status of the Web UI system service.

Reports whether the service definition exists, if it's currently active (running), and if it's enabled for autostart.

Requires a supported service manager (checked by decorator).

Calls API: *get_web_ui_service_status()*.

Usage

```
bedrock-server-manager service status [OPTIONS]
```

Options

-s, --system

Check status of a system-wide service (Linux only, requires sudo).

Default
`False`

14.5 setup

Run an interactive setup to configure the manager.

Usage

```
bedrock-server-manager setup [OPTIONS]
```

14.6 web

Manages the Bedrock Server Manager Web UI application.

This group of commands allows you to start and stop the web server, and to manage its integration as an OS-level system service for features like automatic startup.

Usage

```
bedrock-server-manager web [OPTIONS] COMMAND [ARGS]...
```

14.6.1 start

Starts the Bedrock Server Manager web UI.

This command launches the Uvicorn server that hosts the FastAPI web application. It can run in 'direct' mode (blocking the terminal, useful for development or when managed by an external process manager) or 'detached' mode (running in the background as a new process).

The web server's listening host(s) and debug mode can be configured via options.

Calls API: *start_web_server_api()*.

Usage

```
bedrock-server-manager web start [OPTIONS]
```

Options

-H, --host <host>
: Host address to bind to.

-p, --port <port>
: Port to bind to. Overrides the value in settings.

-d, --debug
: Run in Flask's debug mode (NOT for production).

-m, --mode <mode>
: Run mode: 'direct' blocks the terminal, 'detached' runs in the background.

  Default
  : 'direct'

  Options
  : direct | detached

14.6.2 stop

Stops a detached Bedrock Server Manager web UI process.

This command attempts to find and terminate a web server process that was previously started in 'detached' mode. It typically relies on a PID file to identify the correct process.

This command does not affect web servers started in 'direct' mode or those managed by system services.

Calls API: *stop_web_server_api()*.

Usage

```
bedrock-server-manager web stop [OPTIONS]
```

CHAPTER

FIFTEEN

BSM-CLI

A CLI for managing Bedrock servers.

Usage

```
bsm-cli [OPTIONS] [COMMAND] [ARGS]...
```

15.1 account

Commands for managing your account.

Usage

```
bsm-cli account [OPTIONS] COMMAND [ARGS]...
```

15.1.1 change-password

Change your password.

Usage

```
bsm-cli account change-password [OPTIONS]
```

Options

--current-password <current_password>

Your current password.

--new-password <new_password>

Your new password.

15.1.2 details

Get your account details.

Usage

```
bsm-cli account details [OPTIONS]
```

15.1.3 update-profile

Update your profile.

Usage

```
bsm-cli account update-profile [OPTIONS]
```

Options

--full-name <full_name>

Your full name.

--email <email>

Your email address.

15.1.4 update-theme

Update your theme.

Usage

```
bsm-cli account update-theme [OPTIONS]
```

Options

--theme <theme>

The name of the theme to set.

15.2 addon

Manages server addons.

Usage

```
bsm-cli addon [OPTIONS] COMMAND [ARGS]...
```

15.2.1 install

Installs a behavior or resource pack addon to a specified server.

Usage

```
bsm-cli addon install [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the target server.

-f, --file <addon_file_path>

Path to the addon file (.mcpack, .mcaddon); skips interactive menu.

15.2.2 manage

Interactively manages installed addons on a specified server.

Usage

```
bsm-cli addon manage [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the target server.

15.3 allowlist

Manages a server's player allowlist.

Usage

```
bsm-cli allowlist [OPTIONS] COMMAND [ARGS]...
```

15.3.1 add

Adds one or more players to a server's allowlist.

Usage

```
bsm-cli allowlist add [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the server.

-p, --player <players>

Gamertag of the player to add. Use multiple times for multiple players.

--ignore-limit

Allow player(s) to join even if the server is full.

15.3.2 list

Lists all players currently on a server's allowlist.

Usage

```
bsm-cli allowlist list [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the server.

15.3.3 remove

Removes one or more players from a server's allowlist.

Usage

```
bsm-cli allowlist remove [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the server.

-p, --player <players>

Required Gamertag of the player to remove. Use multiple times for multiple players.

15.4 auth

Manages authentication.

Usage

```
bsm-cli auth [OPTIONS] COMMAND [ARGS]...
```

15.4.1 login

Logs in to the Bedrock Server Manager API.

Usage

```
bsm-cli auth login [OPTIONS]
```

Options

--base-url <base_url>

The base URL of the Bedrock Server Manager API.

--verify-ssl, --no-verify-ssl

Enable/disable SSL verification.

--username <username>

The username for authentication.

--password <password>

The password for authentication.

--token <token>

The JWT token for authentication.

15.4.2 logout

Logs out from the Bedrock Server Manager API.

Usage

```
bsm-cli auth logout [OPTIONS]
```

15.5 backup

Manages server backups.

Usage

```
bsm-cli backup [OPTIONS] COMMAND [ARGS]...
```

15.5.1 create

Creates a backup of specified server data.

Usage

```
bsm-cli backup create [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the target server.

-t, --type <backup_type>

Type of backup to create; skips interactive menu.

Options

world | config | all

-f, --file <file_to_backup>

Specific file to back up (required if –type=config).

15.5.2 prune

Deletes old backups for a server.

Usage

```
bsm-cli backup prune [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server whose backups to prune.

15.5.3 restore

Restores server data from a specified backup file.

Usage

```
bsm-cli backup restore [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the target server.

-f, --file <backup_file_path>

Path to the backup file to restore; skips interactive menu.

15.6 bans

Manages the server ban list.

Usage

```
bsm-cli bans [OPTIONS] COMMAND [ARGS]...
```

15.6.1 add

Adds a player to the server ban list interactively.

Usage

```
bsm-cli bans add [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server.

15.6.2 list

Lists all players on the server's ban list.

Usage

```
bsm-cli bans list [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server.

15.6.3 remove

Removes a player from the server ban list interactively.

Usage

```
bsm-cli bans remove [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server.

15.7 content

Commands for managing content.

Usage

```
bsm-cli content [OPTIONS] COMMAND [ARGS]...
```

15.7.1 upload

Upload a content file.

Usage

```
bsm-cli content upload [OPTIONS] FILE_PATH
```

Arguments

FILE_PATH

Required argument

15.8 permissions

Manages player permission levels on a server.

Usage

```
bsm-cli permissions [OPTIONS] COMMAND [ARGS]...
```

15.8.1 list

Lists all configured player permissions for a specific server.

Usage

```
bsm-cli permissions list [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the server.

15.8.2 set

Sets a permission level for a player on a specific server.

Usage

```
bsm-cli permissions set [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the target server.

-p, --player <player_name>

The gamertag of the player. Skips interactive mode.

-l, --level <level>

The permission level to grant. Skips interactive mode.

Options

visitor | member | operator

15.9 player

Manages the central player database.

Usage

```
bsm-cli player [OPTIONS] COMMAND [ARGS]...
```

15.9.1 add

Manually adds or updates player entries in the central player database.

Usage

```
bsm-cli player add [OPTIONS]
```

Options

-p, --player <players>

Required Player to add in 'Gamertag:XUID' format. Use multiple times for multiple players.

15.9.2 scan

Scans all server logs to discover player gamertags and XUIDs.

Usage

```
bsm-cli player scan [OPTIONS]
```

15.10 plugin

Manages plugins.

Usage

```
bsm-cli plugin [OPTIONS] [COMMAND] [ARGS]...
```

15.10.1 disable

Disables a plugin.

Usage

```
bsm-cli plugin disable [OPTIONS] PLUGIN_NAME
```

Arguments

PLUGIN_NAME

Required argument

15.10.2 enable

Enables a plugin.

Usage

```
bsm-cli plugin enable [OPTIONS] PLUGIN_NAME
```

Arguments

PLUGIN_NAME

Required argument

15.10.3 list

Lists all discoverable plugins.

Usage

```
bsm-cli plugin list [OPTIONS]
```

15.10.4 reload

Reloads all plugins.

Usage

```
bsm-cli plugin reload [OPTIONS]
```

15.10.5 trigger-event

Triggers a custom plugin event.

Usage

```
bsm-cli plugin trigger-event [OPTIONS] EVENT_NAME
```

Options

--payload-json <payload_json>

Optional JSON string to use as the event payload.

Arguments

EVENT_NAME

Required argument

15.11 properties

Manages a server's server.properties file.

Usage

```
bsm-cli properties [OPTIONS] COMMAND [ARGS]...
```

15.11.1 get

Displays server properties from a server's server.properties file.

Usage

```
bsm-cli properties get [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the target server.

-p, --prop <property_name>

Display a single property value.

15.11.2 set

Sets one or more properties in a server's server.properties file.

Usage

```
bsm-cli properties set [OPTIONS]
```

Options

-s, --server <server_name>

Required The name of the target server.

-p, --prop <properties>

A 'key=value' pair to set. Use multiple times for multiple properties.

15.12 server

Manages servers.

Usage

```
bsm-cli server [OPTIONS] COMMAND [ARGS]...
```

15.12.1 delete

Deletes all data for a server, including world, configs, and backups.

Usage

```
bsm-cli server delete [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server to delete.

-y, --yes

Bypass the confirmation prompt.

15.12.2 install

Guides you through installing and configuring a new Bedrock server instance.

Usage

```
bsm-cli server install [OPTIONS]
```

15.12.3 list

Lists all configured Bedrock servers and their current operational status.

Usage

```
bsm-cli server list [OPTIONS]
```

Options

--loop

Continuously refresh server statuses every 5 seconds.

--server-name <server_name>
Display status for only a specific server.

15.12.4 restart

Gracefully restarts a specific Bedrock server.

Usage

```
bsm-cli server restart [OPTIONS]
```

Options

-s, --server <server_name>
Required Name of the server to restart.

15.12.5 send-command

Sends a command to a running Bedrock server's console.

Usage

```
bsm-cli server send-command [OPTIONS] COMMAND_PARTS...
```

Options

-s, --server <server_name>
Required Name of the target server.

Arguments

COMMAND_PARTS
Required argument(s)

15.12.6 start

Starts a specific Bedrock server instance.

Usage

```
bsm-cli server start [OPTIONS]
```

Options

-s, --server <server_name>
Required Name of the server to start.

15.12.7 stop

Sends a graceful stop command to a running Bedrock server.

Usage

```
bsm-cli server stop [OPTIONS]
```

Options

-s, --server <server_name>
: **Required** Name of the server to stop.

15.12.8 update

Checks for and applies updates to an existing Bedrock server.

Usage

```
bsm-cli server update [OPTIONS]
```

Options

-s, --server <server_name>
: **Required** Name of the server to update.

15.13 system

Manages server OS-level resource monitoring and settings.

Usage

```
bsm-cli system [OPTIONS] COMMAND [ARGS]...
```

15.13.1 monitor

Continuously monitors CPU and memory usage of a specific server process.

Usage

```
bsm-cli system monitor [OPTIONS]
```

Options

-s, --server <server_name>
: **Required** Name of the server to monitor.

15.13.2 settings

Configures autostart and autoupdate settings for a Bedrock server.

Usage

```
bsm-cli system settings [OPTIONS]
```

Options

-s, --server <server_name>
: **Required** Name of the server to configure.

15.14 users

Commands for managing users.

Usage

```
bsm-cli users [OPTIONS] [COMMAND] [ARGS]...
```

15.14.1 delete

Delete a user.

Usage

```
bsm-cli users delete [OPTIONS] USER_ID
```

Options

--yes
: Skip confirmation prompt

Arguments

USER_ID
: Required argument

15.14.2 disable

Disable a user account.

Usage

```
bsm-cli users disable [OPTIONS] USER_ID
```

Arguments

USER_ID
: Required argument

15.14.3 enable

Enable a user account.

Usage

```
bsm-cli users enable [OPTIONS] USER_ID
```

Arguments

USER_ID

Required argument

15.14.4 invite

Generate an invite link for a new user.

Usage

```
bsm-cli users invite [OPTIONS] {user|moderator|admin}
```

Arguments

ROLE

Required argument

15.14.5 list

List all users.

Usage

```
bsm-cli users list [OPTIONS]
```

15.14.6 set-role

Set a user's role.

Usage

```
bsm-cli users set-role [OPTIONS] USER_ID {user|moderator|admin}
```

Arguments

USER_ID

Required argument

ROLE

Required argument

15.15 world

Manages server worlds.

Usage

```
bsm-cli world [OPTIONS] COMMAND [ARGS]...
```

15.15.1 export

Exports the server's current active world to a .mcworld file.

Usage

```
bsm-cli world export [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server whose world to export.

15.15.2 install

Installs a world from a .mcworld file, replacing the server's current world.

Usage

```
bsm-cli world install [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the target server.

-f, --file <world_file_path>

Path to the .mcworld file to install. Skips interactive menu.

15.15.3 reset

Deletes the current active world data for a server.

Usage

```
bsm-cli world reset [OPTIONS]
```

Options

-s, --server <server_name>

Required Name of the server whose world to reset.

-y, --yes

Bypass the confirmation prompt.

CHAPTER

SIXTEEN

INSTALLATION

There are three ways to install Bedrock Server Manager, depending on your needs. For most users, the stable version is recommended.

16.1 1. Stable Version (Recommended)

This is the latest official, stable release. It has been tested and is suitable for most use cases. This command will install or upgrade to the latest stable version available on PyPI (Python Package Index).

```
pip install --upgrade bedrock-server-manager
```

Installing a Specific Version

If you need to install a specific version, you can do so by specifying the version with ==.

```
# Example: Install exactly version 3.2.5
pip install bedrock-server-manager==3.2.5
```

You can find a list of all available versions in the **Release History on PyPI**.

16.2 2. Beta / Pre-Release Versions (For Testers)

Occasionally, pre-release versions will be published to PyPI for testing. These versions contain new features and are generally stable but may contain minor bugs.

To install the latest pre-release version, use the `--pre` flag with pip:

```
pip install --pre bedrock-server-manager
```

If you wish to return to the stable version later, you can run: `pip install --force-reinstall bedrock-server-manager`

Previewing the Next Release

The `dev` branch is where all beta developments are merged before being bundled into a new stable release. To see the latest changes that are being prepared, you can browse the code and documentation on the **dev branch**.

16.3 3. Advanced Installation (For Developers & Contributors)

These instructions are for advanced users who want to run the absolute latest code or contribute to the project. Since the project includes a compiled frontend, you cannot simply install directly from the git URL anymore.

Prerequisites:

- **Python:** Version 3.11 or higher.
- **Node.js:** Version 20 or higher (required for building the frontend).
- **Git:** To clone the repository.

Step-by-Step Build & Install:

1. **Clone the Repository:**

```
git clone https://github.com/DMedina559/bedrock-server-manager.git
cd bedrock-server-manager
```

2. **Checkout the Desired Branch:**
   - `main`: Stable code.
   - `dev`: Latest features and changes (unstable).

```
git checkout dev
```

3. **Build the Project:** The project uses a build script to compile the frontend assets (React/JS) and prepare the Python package.
   - **Linux/macOS:**

```
chmod +x build.sh
./build.sh
```

   - **Windows:**

```
build.bat
```

   This process will:
   - Install npm dependencies in `frontend/`.
   - Build the static assets to `src/bedrock_server_manager/web/static/js/dist/`.
   - Build the Python package using `build`.

4. **Install the Local Package:** Once built, you can install the package in editable mode (recommended for development) or as a standard package.

```
# Editable install (changes to python code reflect immediately)
pip install -e .

# OR Standard install
pip install .
```

Note on Versioning: The project uses `setuptools-scm` for dynamic versioning. The version number is automatically derived from the latest git tag. If you are working on a purely local copy without git metadata, you may need to set the `SETUPTOOLS_SCM_PRETEND_VERSION` environment variable during build.

16.4 4. Environment Variables

You can configure the application using environment variables. These variables take precedence over the configuration file.

- `BSM_DATA_DIR`: Overrides the default directory where the application stores data (servers, backups, etc.).
- `BSM_DB_URL`: Overrides the database connection URL found in the configuration file.

16.5 5. Database Configuration

Bedrock Server Manager uses SQLAlchemy to connect to a database. By default, it uses a SQLite database, but you can configure it to use other databases like MySQL, MariaDB, or PostgreSQL.

To use a different database, you need to:

1. Install the necessary database driver.
2. Set the `db_url` in your configuration file OR set the `BSM_DB_URL` environment variable to the correct database connection string.

16.5.1 Installing Database Drivers

You can install the required drivers as optional dependencies with `pip`.

- **For MySQL:**

```
pip install "bedrock-server-manager[mysql]"
```

  This will install both `mysqlclient` and `PyMySQL`.

- **For MariaDB:**

```
pip install "bedrock-server-manager[mariadb]"
```

  This will install both `mariadb` and `PyMySQL`.

- **For PostgreSQL:**

```
pip install "bedrock-server-manager[postgresql]"
```

  This will install `psycopg`.

16.5.2 Database Connection URLs

Here are some examples of connection URLs for different databases.

- **MySQL (using `mysqlclient`):**

```
mysql://user:password@host/dbname
```

- **MySQL (using `PyMySQL`):**

```
mysql+pymysql://user:password@host/dbname
```

- **MariaDB (using `mariadb`):**

```
mariadb://user:password@host/dbname
```

- **MariaDB (using PyMySQL):**

```
mariadb+pymysql://user:password@host/dbname
```

- **PostgreSQL (using psycopg):**

```
postgresql+psycopg://user:password@host/dbname
```

CHAPTER

SEVENTEEN

DOCKER INSTALL

This project includes a `Dockerfile` that allows you to build and run the Bedrock Server Manager in a containerized environment. This allows for easy deployment and management of the application.

17.1 Image Location

The official Docker image is hosted on both the GitHub Container Registry and Docker Hub.

- **Docker Hub**: `dmedina559/bedrock-server-manager:stable`
- **GitHub Container Registry**: `ghcr.io/dmedina559/bedrock-server-manager:stable`

You can pull it using the following command:

```
docker pull dmedina559/bedrock-server-manager:stable
```

17.2 Image Tags and Versioning

The Docker images for Bedrock Server Manager are tagged to help you choose the right version for your needs. Understanding the tagging strategy will help you manage your deployments effectively.

17.2.1 Available Tags

- **`latest`**: This tag always points to the most recent release, including stable, beta, and release candidates. It is a convenient way to stay up-to-date with the latest features and fixes, but it may not always be the most stable. Use this tag if you want the newest version and are comfortable with potentially running pre-release software.
- **`stable`**: This tag points to the most recent stable release. It does not include beta versions or release candidates. This is the recommended tag for most users, as it provides a balance of new features and stability. Use this tag for production environments or if you prefer to avoid pre-release versions.
- **Version Tags (e.g., `3.7.0`, `3.7.0b5`)**: Every release, whether stable or pre-release, has its own version-specific tag. These tags are useful for pinning your deployment to a specific version of the application, which ensures that your environment is predictable and does not change unexpectedly. This is a common practice for production deployments where stability and control are critical.

17.2.2 How to Use the Tags

To pull a specific tag, simply append it to the image name. For example:

- To pull the latest stable version:

  ```
  docker pull dmedina559/bedrock-server-manager:stable
  ```

- To pull a specific version:

```
docker pull dmedina559/bedrock-server-manager:3.7.0
```

By choosing the right tag, you can control when and how you update your Bedrock Server Manager instance, ensuring a smooth and stable experience.

17.3 Running the Container

The Docker image is configured to run the web server by default. To run the container, you need to map the port and, most importantly, provide volumes for persistent data storage.

17.3.1 Data Persistence

The application uses two main directories to store its data. To prevent data loss when the container is removed, you **must** mount volumes for both of these locations.

1. **Configuration Directory:** This directory stores the main `bedrock_server_manager.json` file, which contains the path to the data directory and the database URL.
   - Container Path: `/root/.config/bedrock-server-manager`
2. **Data Directory:** This directory stores everything else, including server files, plugins, backups, and the application's database.
   - Default Container Path: `/root/bedrock-server-manager`

Using a Named Volume (Recommended)

This is the easiest and most recommended way to manage the data. Docker will manage the volumes for you.

```
docker run -d \
  -p 11325:11325 \
  -p 19132:19132/udp \
  -p 19133:19133/udp \
  --name bsm-container \
  -v bsm_config:/root/.config/bedrock-server-manager \
  -v bsm_data:/root/bedrock-server-manager \
  dmedina559/bedrock-server-manager:stable
```

This command creates two named volumes, `bsm_config` and `bsm_data`, and mounts them to the correct locations.

Using Bind Mounts

Alternatively, you can mount directories from your host machine.

```
docker run -d \
  -p 11325:11325 \
  -p 19132:19132/udp \
  -p 19133:19133/udp \
  --name bsm-container \
  -v /path/on/host/bsm_config:/root/.config/bedrock-server-manager \
  -v /path/on/host/bsm_data:/root/bedrock-server-manager \
  dmedina559/bedrock-server-manager:stable
```

17.3.2 Overriding Environment Variables

You can configure the application by passing environment variables to the container.

- `HOST`: The host for the web server (default: `0.0.0.0`).
- `PORT`: The port for the web server (default: `11325`).
- `BSM_DATA_DIR`: The path to the data directory within the container (default: `/root/bedrock-server-manager`). **Note**: If you change this, make sure to update your volume mounts accordingly.
- `BSM_DB_URL`: The database connection URL. Setting this overrides the value in the configuration file.

For example, to change the web server port to `8080` and use a custom database:

```
docker run -d \
  -p 8080:8080 \
  -p 19132:19132/udp \
  -p 19133:19133/udp \
  --name bsm-container \
  -e PORT=8080 \
  -e HOST=0.0.0.0 \
  -e BSM_DB_URL="mysql://user:pass@host/db" \
  -v bsm_config:/root/.config/bedrock-server-manager \
  -v bsm_data:/root/bedrock-server-manager \
  dmedina559/bedrock-server-manager:stable
```

17.3.3 Exposing Minecraft Server Ports

For players to be able to connect to your Minecraft servers, you must expose the corresponding UDP ports from the container. The default Minecraft Bedrock ports are `19132/udp` (IPv4) and `19133/udp` (IPv6).

If you run multiple servers, you will need to map a port for each one. See the example above for how to add more `-p` flags.

Note on LAN Discovery (Broadcast): By default, Docker containers run in a bridge network which isolates broadcast traffic. This means servers running in Docker **will not appear in the "Worlds" tab** (LAN games) of Minecraft clients on the same network, even if the ports are mapped correctly. Players will need to add the server manually using the "Servers" tab -> "Add Server" button and entering the host's IP address and port.

Alternative: Host Networking (Required for LAN Discovery)

To enable automatic LAN discovery (so the server appears in the Worlds list), you must use **Host Networking**. This bypasses Docker's network isolation and allows broadcast packets to reach the LAN.

A simpler, but less isolated, approach is to use host networking by adding `--network host` to your `docker run` command. Note that when using host networking, you do not need to map individual ports.

17.4 Using Docker Compose

For an even easier setup, you can use Docker Compose. Create a `docker-compose.yml` file and add the following content:

```
version: '3.8'
services:
  bedrock-server-manager:
    image: dmedina559/bedrock-server-manager:stable     # Use desired tag here (e.g.,␣
```

(continues on next page)

(continued from previous page)

```
↪stable, latest, 3.7.0)
    container_name: bsm-container                    # Name of the container
    restart: unless-stopped                          # Restart policy
    # network_mode: "host"                           # Use host networking for LAN␣
↪discovery
    ports:
      - "11325:11325"                                # Web server port
      - "19132:19132/udp"                            # Default Minecraft Bedrock␣
↪server port (IPv4)
      - "19133:19133/udp"                            # Default Minecraft Bedrock␣
↪server port (IPv6)
      # - "19100:19100/udp"                          # Add more ports as needed for␣
↪additional servers
    environment: # Optional
      - HOST=0.0.0.0                                 # Which host to bind the web␣
↪server to
      - PORT=11325                                   # Port for the web server
      # - BSM_DB_URL=mysql://user:pass@host/db       # Custom database URL
    volumes:
      - bsm_config:/root/.config/bedrock-server-manager
      - bsm_data:/root/bedrock-server-manager

volumes:
  bsm_config:
  bsm_data:
```

You can then start the application with a single command: `docker-compose up -d`.

17.5 Advanced Usage

17.5.1 Accessing the CLI

You can access the `bedrock-server-manager` CLI using `docker exec` when the container is already running.

To run a single command:

```
docker exec bsm-container bedrock-server-manager <command>
```

To get an interactive shell:

```
docker exec -it bsm-container /bin/bash
```

17.5.2 Overriding the Startup Command (e.g., for Migrations)

If you need to perform maintenance tasks—such as running database migrations or setting up configurations—before starting the web server, you can override the default startup command.

When you start the container, passing a custom command will replace the default behavior of starting the web server. For example, to temporarily run database migrations interactively without starting the web server, use:

```
docker run -it --rm \
  -v bsm_config:/root/.config/bedrock-server-manager \
  -v bsm_data:/root/bedrock-server-manager \
```

(continues on next page)

(continued from previous page)

```
  dmedina559/bedrock-server-manager:stable \
  bedrock-server-manager database upgrade
```

Or, to launch an interactive bash shell instead of the web server:

```
docker run -it --rm dmedina559/bedrock-server-manager:stable bash
```

Note: The `--rm` flag automatically removes the container once the command finishes. Remove this flag if you want to keep the container.

17.5.3 Changing the Database URL

The easiest way to change the database URL is to set the `BSM_DB_URL` environment variable as described in the **Overriding Environment Variables** section.

Alternatively, you can edit the configuration file stored in the `bsm_config` volume:

1. Stop the container: `docker stop bsm-container`
2. Locate and edit the `bedrock_server_manager.json` file inside the `bsm_config` volume. The exact location on your host will depend on your Docker setup. You can use `docker volume inspect bsm_config` to find the mountpoint.
3. Change the `db_url` value in the JSON file.
4. Start the container again: `docker start bsm-container`

CHAPTER

EIGHTEEN

INSTALL CONTENT

Easily import addons and worlds into your servers. The app will look in the configured `CONTENT_DIR` directories for addon files.

Place .mcworld files in `CONTENT_DIR/worlds` or .mcpack/.mcaddon files in `CONTENT_DIR/addons`

Navigate to the Web UI to install content to your servers.

> **Tip**
>
> Enable the content_uploader_plugin to easily upload content to your content dir!

CHAPTER

NINETEEN

WEB SERVER

19.1 Themes

The Bedrock Server Manager web UI supports theming, allowing you to customize the look and feel of the application.

See the *Theming* documentation for more information on how to create and apply custom themes.

19.2 Custom Web Server Panorama

You can personalize the background panorama displayed on the main web server page.

Steps:

1. Choose your desired background image. It **must** be a JPEG file (with a `.jpeg` extension).
2. Navigate to the Bedrock Server Manager's configuration directory, typically located at `./.config/` relative to the manager's installation path.
3. Place your chosen image file into this `./.config/` directory.
4. Rename the image file to `panorama.jpeg`.
5. Refresh the web server page in your browser. The new panorama should now be displayed.

Default Behavior:

If the file `./.config/panorama.jpeg` is not found or is not a valid JPEG image, the default Bedrock Server Manager icon will be used as the background panorama.

19.3 World Icons

The "Servers List" within the web interface can display unique icons for each of your Minecraft worlds, making them easier to identify at a glance.

How it Works:

The server manager looks for a file named `world_icon.jpeg` inside each world's specific folder.

Adding Icons:

- **Imported Worlds / Client-Created Worlds:** Worlds that were imported or originally created using the Minecraft client (Bedrock Edition) often already include a `world_icon.jpeg` file within their folder structure. No extra steps may be needed.

- **Dedicated Server-Created Worlds:** Worlds generated directly by the Bedrock Dedicated Server software usually *do not* have this icon file by default. You can add one manually:
    1. Obtain or create an image you want to use as the icon (JPEG format, `.jpeg`). A square aspect ratio often looks best.
    2. Locate the specific world folder for the server you want to customize. This is typically found within the server's directory structure (e.g., `./servers/your_server_name/worlds/your_world_name/`).
    3. Copy your chosen image file into this specific world folder.
    4. Rename the image file exactly to `world_icon.jpeg`.
    5. The icon should appear the next time the server list is loaded or refreshed in the web interface.

Default Behavior:

If a `world_icon.jpeg` file is not found within a specific world's directory, the default Bedrock Server Manager icon will be displayed for that world in the server list.

19.4 API Client

The `bsm-api-client` is an asynchronous Python library for interacting with the Bedrock Server Manager API.

This library allows developers to programmatically manage their Minecraft Bedrock servers. It is ideal for automating administrative tasks, building custom control panels, or integrating BSM into other systems and workflows.

- **Repository**: https://github.com/DMedina559/bsm-api-client

19.5 Home Assistant Integration

The Bedrock Server Manager integration for Home Assistant allows you to control and monitor your server directly from your Home Assistant dashboard.

This integration is built using the `bsm-api-client`.

- **Repository**: https://github.com/DMedina559/bsm-home-assistant-integration

CHAPTER

TWENTY

WEBSOCKET IMPLEMENTATION

This document provides an overview of the WebSocket implementation in the Bedrock Server Manager, covering backend (server-side) components.

The backend WebSocket implementation is built using FastAPI and is divided into two main components: a WebSocket router and a connection manager.

20.1 WebSocket Router

The WebSocket router is defined in `src/bedrock_server_manager/web/routers/websocket_router.py`. It is responsible for the following:

- **Endpoint**: Creates a WebSocket endpoint at `/ws`.
- **Authentication**: Authenticates the user on the first message payload. The client must send an authentication message within 5 seconds of connecting, containing their JWT access token (e.g., `{"action": "authenticate", "token": "eyJhb..."}`).
- **Message Handling**: Listens for incoming JSON messages from the client. After authentication, these messages are expected to have an `action` (`subscribe` or `unsubscribe`) and a `topic`.
- **Connection Management**: Hands off the connection and subscription management to the `ConnectionManager`.

20.2 Connection Manager

The `ConnectionManager` is a class defined in `src/bedrock_server_manager/web/websocket_manager.py`. It is the core of the backend WebSocket implementation and is responsible for the following:

- **Connection Tracking**: Keeps track of all active WebSocket connections.
- **Topic-Based Subscriptions**: Manages which clients are subscribed to which topics.
- **Message Broadcasting**: Provides methods for sending messages to a single client, all clients subscribed to a specific topic, or all clients for a specific user.

20.3 Wildcard Subscription

Clients can subscribe to the wildcard topic `*`. A client subscribed to `*` will receive **all** broadcast messages sent to any specific topic. This is useful for monitoring tools or dashboards that need a global view of system activity.

If a client is subscribed to both a specific topic (e.g., `event:server_start`) and the wildcard `*`, the system ensures the client only receives the message once.

20.4 Architecture

1. A client connects to the `/ws` endpoint (e.g., `ws://<host>:<port>/ws`).
2. The client immediately sends an authentication message: `{"action": "authenticate", "token": "<your_access_token>"}`.
3. The `websocket_router` authenticates the user using the provided token and passes the connection to the `ConnectionManager`.
4. The client sends a `subscribe` message to a topic.
5. The `websocket_router` calls the `ConnectionManager` to subscribe the client to the topic.
6. When an event occurs on the server (e.g., the server status changes), the `ConnectionManager` is used to broadcast a message to all clients subscribed to the relevant topic.

20.5 Available Topics

The following WebSocket topics are available for subscription:

20.5.1 Event Topics

Event topics broadcast a message when a specific event occurs on the server. The topic name is in the format `event:{event_name}`.

- `event:before_addon_import`
- `event:after_addon_import`
- `event:before_backup`
- `event:after_backup`
- `event:before_restore`
- `event:after_restore`
- `event:before_prune_backups`
- `event:after_prune_backups`
- `event:before_players_add`
- `event:after_players_add`
- `event:before_player_db_scan`
- `event:after_player_db_scan`
- `event:before_server_status_change`
- `event:after_server_status_change`
- `event:before_server_statuses_updated`
- `event:before_server_start`
- `event:after_server_start`
- `event:before_server_stop`
- `event:after_server_stop`
- `event:before_command_send`

- `event:after_command_send`
- `event:before_allowlist_change`
- `event:after_allowlist_change`
- `event:before_server_install`
- `event:after_server_install`
- `event:before_server_update`
- `event:after_server_update`
- `event:before_autostart_change`
- `event:after_autostart_change`
- `event:before_web_server_start`
- `event:after_web_server_start`
- `event:before_web_server_stop`
- `event:after_web_server_stop`
- `event:before_world_export`
- `event:after_world_export`
- `event:before_world_import`
- `event:after_world_import`
- `event:before_world_reset`
- `event:after_world_reset`

20.5.2 Resource Monitor Topics

Resource monitor topics broadcast resource usage information for a specific server. The topic name is in the format `resource-monitor:{server_name}`. Replace `{server_name}` with the name of the server you want to monitor.

20.5.3 Task Manager Topics

The Task Manager is responsible for handling background tasks and notifying the user of their progress. Unlike event topics or resource monitor topics, the client does not need to explicitly subscribe to task topics. Instead, the server automatically pushes updates to the user who initiated the task.

The message structure for task updates is as follows:

```
{
    "type": "task_update",
    "topic": "task:{task_id}",
    "data": {
        "status": "in_progress",
        "message": "Task is running.",
        "result": null,
        "username": "username"
    }
}
```

- `type`: Indicates the type of message. For task updates, this is always `task_update`.

- `topic`: The topic of the message. This follows the format `task:{task_id}`, where `{task_id}` is the unique identifier of the task.
- `data`: An object containing details about the task:
  - `status`: The current status of the task (`in_progress`, `success`, or `error`).
  - `message`: A human-readable message describing the status.
  - `result`: The result of the task, if any.
  - `username`: The username of the user who initiated the task.

CHAPTER

TWENTYONE

THEMING

The Bedrock Server Manager web UI now supports theming, allowing you to customize the look and feel of the application.

21.1 How it Works

Theming is done through CSS variables. The application defines a set of CSS variables that control the colors, fonts, and other style elements of the application. You can create your own custom themes by overriding these variables in a CSS file.

21.2 Creating a Custom Theme

To create a custom theme, you will need to create a CSS file in the `themes` directory in your application data directory. The name of the file will be the name of your theme. For example, if you create a file named `my-theme.css`, your theme will be named `my-theme`.

Your theme file should contain a `:root` block that defines the CSS variables that you want to override. For example, to change the text color to red, you would add the following to your theme file:

```
:root {
    --text-color: red;
}
```

21.3 Available Variables

The following is a list of the available CSS variables and what they control:

Variable	Description
`--background-image`	The background image of the application.
`--background-overlay-color`	The color of the overlay that is displayed on top of the background image.
`--text-color`	The default text color.
`--container-background-color`	The background color of the main content container.
`--border-color`	The default border color.
`--header-text-color`	The text color of the header.
`--monitor-output-background-color`	The background color of the monitor output.
`--monitor-output-text-color`	The text color of the monitor output.
`--monitor-output-border-color`	The border color of the monitor output.

continues on next page

Table 1 – continued from previous page

Variable	Description
`--splash-text-color`	The color of the splash text.
`--button-background-color`	The background color of buttons.
`--button-border-color`	The border color of buttons.
`--button-text-color`	The text color of buttons.
`--button-hover-background-color`	The background color of buttons when hovered.
`--button-hover-border-color`	The border color of buttons when hovered.
`--button-hover-text-color`	The text color of buttons when hovered.
`--button-active-background-color`	The background color of buttons when active.
`--button-active-border-color`	The border color of buttons when active.
`--primary-button-background-color`	The background color of primary buttons.
`--primary-button-border-color`	The border color of primary buttons.
`--primary-button-text-color`	The text color of primary buttons.
`--primary-button-hover-background-color`	The background color of primary buttons when hovered.
`--primary-button-hover-border-color`	The border color of primary buttons when hovered.
`--primary-button-active-background-color`	The background color of primary buttons when active.
`--primary-button-active-border-color`	The border color of primary buttons when active.
`--danger-button-background-color`	The background color of danger buttons.
`--danger-button-border-color`	The border color of danger buttons.
`--danger-button-text-color`	The text color of danger buttons.
`--danger-button-hover-background-color`	The background color of danger buttons when hovered.
`--danger-button-hover-border-color`	The border color of danger buttons when hovered.
`--danger-button-active-background-color`	The background color of danger buttons when active.
`--danger-button-active-border-color`	The border color of danger buttons when active.
`--form-input-background-color`	The background color of form inputs.
`--form-input-border-color`	The border color of form inputs.
`--form-input-text-color`	The text color of form inputs.
`--form-input-focus-border-color`	The border color of form inputs when focused.
`--form-label-text-color`	The text color of form labels.
`--validation-error-text-color`	The text color of validation errors.
`--form-input-invalid-border-color`	The border color of invalid form inputs.
`--form-input-invalid-background-color`	The background color of invalid form inputs.
`--table-header-background-color`	The background color of table headers.
`--table-header-text-color`	The text color of table headers.
`--table-border-color`	The border color of tables.
`--table-row-hover-background-color`	The background color of table rows when hovered.
`--tab-background-color`	The background color of tabs.
`--tab-border-color`	The border color of tabs.
`--tab-text-color`	The text color of tabs.
`--tab-hover-background-color`	The background color of tabs when hovered.
`--tab-hover-text-color`	The text color of tabs when hovered.
`--tab-active-background-color`	The background color of active tabs.
`--tab-active-text-color`	The text color of active tabs.
`--message-success-background-color`	The background color of success messages.
`--message-success-border-color`	The border color of success messages.
`--message-success-text-color`	The text color of success messages.
`--message-error-background-color`	The background color of error messages.
`--message-error-border-color`	The border color of error messages.
`--message-error-text-color`	The text color of error messages.
`--message-warning-background-color`	The background color of warning messages.
`--message-warning-border-color`	The border color of warning messages.

continues on next page

Table 1 – continued from previous page

Variable	Description
`--message-warning-text-color`	The text color of warning messages.
`--sidebar-background-color`	The background color of the sidebar.
`--sidebar-border-color`	The border color of the sidebar.
`--sidebar-link-text-color`	The text color of sidebar links.
`--sidebar-link-hover-background-color`	The background color of sidebar links when hovered.
`--sidebar-link-hover-text-color`	The text color of sidebar links when hovered.
`--sidebar-link-active-background-color`	The background color of active sidebar links.
`--sidebar-link-active-border-color`	The border color of active sidebar links.
`--server-card-background-color`	The background color of server cards.
`--server-card-border-color`	The border color of server cards.
`--server-card-hover-background-color`	The background color of server cards when hovered.
`--server-card-info-text-color`	The text color of server card info.
`--server-card-info-label-color`	The text color of server card labels.
`--toggle-switch-off-background-color`	The background color of toggle switches when off.
`--toggle-switch-off-border-color`	The border color of toggle switches when off.
`--toggle-switch-off-handle-background-color`	The background color of toggle switch handles when off.
`--toggle-switch-off-handle-border-color`	The border color of toggle switch handles when off.
`--toggle-switch-on-background-color`	The background color of toggle switches when on.
`--toggle-switch-on-border-color`	The border color of toggle switches when on.
`--toggle-switch-on-handle-background-color`	The background color of toggle switch handles when on.
`--toggle-switch-on-handle-border-color`	The border color of toggle switch handles when on.

21.4 Plugin Theming

Plugins can also use the theme engine to style their own custom pages. To do this, you will need to use the CSS variables in your plugin's CSS files. For example, to use the theme's text color, you would use the following CSS:

```
.my-plugin-text {
    color: var(--text-color);
}
```

21.5 Images

CHAPTER

TWENTYTWO

CHANGELOG

22.1 3.10.6

Released on 2026-09-05 - GitHub - PyPI

> **Note**
>
> Paving the way for Async: This release introduces support for async plugins and task loops! This is the first step toward the fully asynchronous refactor starting in the next major version. This will significantly improve performance and keep the main thread responsive without relying on heavy OS threads.

22.1.1 What's New:

Plugin Refactors:

The plugin architecture has been heavily refined to be more intuitive, safer, and feature-rich:

- **New Event System & `@app_event` Decorator**: Event registration has been vastly cleaned up. You can now use the new `@app_event` decorator for a much more streamlined way to register app/custom events, replacing the legacy methods (e.g `def before_server_stop(...)`) with the new decorator (e.g `@app_event("before_server_start")`)
- **Async Plugin Support**: Plugins now officially support `async` event methods, allowing you to perform non-blocking operations directly in your event hooks. This is the first step towards a full asynchronous refactor.
- **Cancellable Events (`event.cancel(...)`)**: Core operations can now be intercepted and cancelled by plugins. The new `CancellableEvent` object is injected into `before` hooks. Calling `event.cancel(reason)` will short-circuit the core operation and return a cancellation payload. This is useful for scenarios where you want to cancel a start if a backup or update fails etc, etc.
- **Periodic Background Tasks (`@task_loop`)**: Plugins can now safely run periodic tasks. Using the lightweight `@task_loop(interval_in_seconds)` decorator on a plugin method will automatically schedule and run it, with the `PluginManager` seamlessly handling startup and safe cancellation during unload.
- **Topological Dependency Loading**: The `PluginManager` now handles topological sorting of plugins. You can define `dependencies` and `optional_dependencies` in your plugin attributes, ensuring they load in the correct order.

- **Event Identity Keys & Re-entrancy Refactor**: A runtime `_event_registry` is now used for per-event identity keys, preventing infinite loops during event dispatches, removing the need for hardcoded app events.
- **Auto-Generated Event Documentation**: The `api_docs_generator` plugin has been extended to automatically generate markdown documentation for all available plugin events in the current app installation.

What's Changed

Features

- Feat/improved plugins by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/400

Style & Refactoring

- Refactor/app-start-up-init by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/396

Dependencies

- dep: update httpx2 requirement from <2.11,>=2.5.0 to >=2.5.0,<2.13 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/395
- dep: update mcstatus requirement from <14.1,>=12.0 to >=12.0,<14.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/393
- dep: update click requirement from <8.5,>=8.2.0 to >=8.2.0,<8.6 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/398
- dep: update isort requirement from <8.1,>=5.13.0 to >=5.13.0,<9.1 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/399

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.5...3.10.6

22.2 3.10.5

Released on 2026-08-20 - GitHub - PyPI

> **Important**
>
> **DATABASE UPGRADE RECOMMENDED**
>
> You should run the `database upgrade` command **before** starting the webserver to migrate log configuration into its new location
>
> Docker Users Read: https://bedrock-server-manager.readthedocs.io/en/latest/extras/docker.html#overriding-the-startup-command-e-g-for-migrations

> **Important**
>
> As of this release the default location for app data in new installs is `{config_dir}/data`

This release includes better support for reverse proxy setups by using the `x-ingress-path` header

22.2.1 What's New:

- New cli flags to override the `config_dir`, `data_dir`, `db_url`, `log_level`
  - Adds env var support for all 4 also
  - Example: `bedrock-server-manager --log-level debug web start`
- Started initial logging revamp by first moving it to the `{config_dir}/logs` location and moving log level out of the db

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.4...3.10.5

22.3 3.10.4

Released on 2026-08-08 - GitHub - PyPI

22.3.1 What's Changed

Dependencies

- dep: bump docker/login-action from 4.5.2 to 4.6.0 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/385
- dep: update alembic requirement from <1.19,>=1.16 to >=1.16,<1.20 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/387
- dep: update uvicorn requirement from <0.52,>=0.35.0 to >=0.35.0,<0.53 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/386

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.3...3.10.4

22.4 3.10.3

Released on 2026-07-31 - GitHub - PyPI

- Fixes #382

22.4.1 What's Changed

Dependencies

- dep: bump docker/login-action from 4 to 4.5.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/383
- dep: update fastapi requirement from <0.141,>=0.115.0 to >=0.115.0,<0.142 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/381

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.2...3.10.3

22.5 3.10.2

Released on 2026-07-26 - GitHub - PyPI

22.5.1 What's Changed

Dependencies

- dep: update platformdirs requirement from <4.11,>=4.3.0 to >=4.3.0,<4.12 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/376
- dep: update fastapi requirement from <0.140,>=0.115.0 to >=0.115.0,<0.141 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/378
- dep: update httpx2 requirement from <2.8,>=2.5.0 to >=2.5.0,<2.10 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/377

22.5.2 New Contributors

- @pre-commit-ci[bot] made their first contribution in https://github.com/DMedina559/bedrock-server-manager/pull/374

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.1...3.10.2

22.6 3.10.1

Released on 2026-07-20 - GitHub - PyPI

22.6.1 What's Changed

Features

- Feat/expanded properties validation by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/372

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.10.0...3.10.1

22.7 3.10.0

Released on 2026-07-16 - GitHub - PyPI

Important

DATABASE UPGRADE REQUIRED

You must run the `database upgrade` command **before** starting the webserver

Docker Users Read: https://bedrock-server-manager.readthedocs.io/en/latest/extras/docker.html#overriding-the-startup-command-e-g-for-migrations

Warning

Breaking Changes for Plugins and HTTP API Users

This release introduces the final set of major architectural changes aimed at improving modularity and performance:

- **Module Splitting:** Several core functionalities have been decoupled into their own dedicated modules. Logic pertaining to `allowlist`, `properties`, and `permissions` are now isolated, meaning imports and plugin references to these features will need to be updated.

- **Endpoint Consolidation:** Redundant HTTP endpoints—specifically those directly querying server configuration, status, and version—have been removed. Clients should migrate to using the unified server settings or the new `/api/server/{server_name}/summary` endpoint to retrieve these details.

22.7.1 New Features:

- Player Bans!
  - You can now ban players per server and give a provided reason
  - Uses the bedrock server `/kick` command
  - IMPORTANT: Player Bans uses the player monitoring settings, lower intervals result in more reliable ban enforcements
  - IMPORTANT: While I tried to make sure this system is robust, there were a few times i was able to joined a server for a couple seconds before the ban kicked me. This happened more often when `settings.monitoring.player_interval_sec` was set to higher intervals, i recommend settings this to 5 seconds, or not relying on this ban system entirely.
  - TIP: This system is only temporary until Mojang adds a dedicated ban system to bedrock.
- Online Players!
  - Now instead of just a number you also get a list of player names/xuid that are online per server
  - Used in the Player Ban system
- Database Backups!
  - Backup and restore the database through the cli `database backup`/`database restore` commands
  - Replaced the `database migrate` command
  - IMPORTANT: Stores data as plain `json`, **hashed** passwords and other data are viewable. Store backups securely!

What's Changed

Breaking Changes

- Refactor!/database refactor by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/333
- Refactor!/bedrock server manger class removal by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/341
- Refactor!/cleanup splitups removals by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/348
- Refactor!/minor method changes by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/353
- Fix!/swagger UI by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/356
- Refactor!/websocket auth by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/358
- Release!/3.10.0 by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/370

Features

- Feat/online players list by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/326
- Feat/player ban system by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/334
- Feat/database backup restore by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/362

Bug Fixes

- Fix/delete server db entries by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/329
- fix: windows system service by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/342
- Fix/auto add missing properties by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/350

Documentation

- Docs/add missing modules by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/354

Style & Refactoring

- refactor: use sys.executable instead of manual EXPATH by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/343
- Refactor/restructure fastapi deps by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/359
- Refactor/plugin events by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/360
- Refactor/v2 tests by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/367

Performance

- Perf/various improvements by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/336

Tests

- test: use `pytest-subprocess` by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/337

Dependencies

- dep: update pytest-subprocess requirement from <1.6.0,>=1.5.4 to >=1.5.4,<1.7.0 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/338
- dep: update platformdirs requirement from <4.10,>=4.3.0 to >=4.3.0,<4.11 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/335
- dep: update sphinx-github-changelog requirement from <2.1,>=1.7.0 to >=1.7.0,<2.4 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/332

- dep: update pytest-asyncio requirement from <1.4,>=1.1.0 to >=1.1.0,<1.5 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/331
- dep: update uvicorn requirement from <0.48,>=0.35.0 to >=0.35.0,<0.49 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/327
- dep: bump actions/checkout from 6 to 7 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/347
- dep: update pytest requirement from <9.1,>=8.4.0 to >=8.4.0,<9.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/346
- dep: update fastapi requirement from <0.137,>=0.115.0 to >=0.115.0,<0.138 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/345
- dep: update pywin32 requirement from <312,>=310 to >=310,<313 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/340
- dep: update uvicorn requirement from <0.49,>=0.35.0 to >=0.35.0,<0.50 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/339
- dep: update fastapi requirement from <0.138,>=0.115.0 to >=0.115.0,<0.139 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/352
- dep: update mypy requirement from <2.2,>=1.10.0 to >=1.10.0,<2.3 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/366
- dep: update uvicorn requirement from <0.50,>=0.35.0 to >=0.35.0,<0.52 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/365
- dep: update mcstatus requirement from <13.2,>=12.0 to >=12.0,<14.1 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/357
- dep: update fastapi requirement from <0.139,>=0.115.0 to >=0.115.0,<0.140 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/361
- dep: update httpx2 requirement from <2.6,>=2.5.0 to >=2.5.0,<2.8 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/369
- dep: update mypy requirement from <2.3,>=1.10.0 to >=1.10.0,<2.4 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/368

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.9.1...3.10.0

22.8 3.9.1

Released on 2026-05-06 - GitHub - PyPI

22.8.1 What's Changed

Style & Refactoring

- refactor: centralize list files function by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/312

Dependencies

- dep: update fastapi requirement from <0.136,>=0.115.0 to >=0.115.0,<0.137 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/310
- dep: update uvicorn requirement from <0.43,>=0.35.0 to >=0.35.0,<0.45 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/302

- dep: update uvicorn requirement from <0.45,>=0.35.0 to >=0.35.0,<0.47 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/315
- dep: update mcstatus requirement from <13.1,>=12.0 to >=12.0,<13.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/313

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.9.0...3.9.1

22.9 3.9.0

Released on 2026-04-15 - GitHub - PyPI

> **Warning**
>
> Breaking changes for Plugins and HTTP API users Please see: #297

22.9.1 What's Changed

Breaking Changes

- Refactor!/remove UI by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/282
- Refactor!/router changes by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/293

Features

- Feat/addon management by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/296
- Feat/queue actions instead of skip by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/300

Bug Fixes

- fix/mimetypes-content-type by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/291
- Fix/remote sessions by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/301

Documentation

- Docs/in depth plugin docs by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/287
- Docs/more classes by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/289

Style & Refactoring

- Refactor/prioritize cookies auth by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/298

Dependencies

- dep: update requests requirement from <2.33,>=2.32.3 to >=2.32.3,<2.34 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/285
- dep: update uvicorn[standard] requirement from <0.42,>=0.35.0 to >=0.35.0,<0.43 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/283
- dep: update black requirement from <26.2,>=25.1.0 to >=25.1.0,<26.4 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/278
- dep: bump docker/build-push-action from 6 to 7 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/277
- dep: bump docker/metadata-action from 5 to 6 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/276
- dep: update mcstatus requirement from <12.3,>=12.0 to >=12.0,<13.1 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/275
- dep: bump docker/login-action from 3 to 4 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/273
- dep: update mypy requirement from <1.20,>=1.10.0 to >=1.10.0,<1.21 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/288
- dep: update fastapi requirement from <0.130,>=0.115.0 to >=0.115.0,<0.136 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/269
- dep: bump actions/github-script from 8 to 9 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/304

Chore

- Chore/more splash text by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/294
- chore: change addon icon router to no use auth by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/299

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.8.0...3.9.0

22.10 3.8.0

Released on 2026-02-26 - GitHub - PyPI

> **Warning**
>
> Python 3.11+ required

> **Tip**
>
> Try out the new UI now!
>
> This release introduces a new React based UI that greatly enhances the user experience. The goal is to have the new UI completely replace the current legacy ui by `3.9.0`
>
> To navigate back to the legacy UI simply use the `Legacy UI (DEPRECATED)` link at the bottom of the webpage.

Note

This release also fixes a Server Permissions bug which caused no players to show up when the permissions.json is empty (excluded global players)

Tip

Its recommended to run the `migrate old-config` command to clean up some unused settings in the database

22.10.1 New UI Improvements:

- Added Global Players page which allows manually adding players to the global db, or trigger a scan
- Added Audit Log page which shows user actions, app logs, background tasks
- Improved Monitor page to show bedrock logs and quick action buttons
- Merged Worlds/Addons pages to use tabs, allows uploading content when Content Uploader plugin is enabled
- Merged Allowlist/Permissions pages into Access Control, permissions tab has been improved to allow manually adding Name and XXUID to server permissions
- Improved Properties page with search and custom properties entry
- New plugin page system, now plugin routers can respond a json ui for the V2 frontend to render (legacy plugin pages use iframe in new ui)

22.10.2 What's Changed

Breaking Changes

- Build!/bump min python ver by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/209
- Chore!/cleanup unused code by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/217
- Chore!/various cleanup by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/240
- Refacor!/migrate to legacy by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/257

Features

- Feat/wildcard subscriptions by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/210
- Feat/frontend revamp by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/247
- Feat/v2 content upload page by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/249
- Feat(v2 UI)/add missing delete button by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/250
- Feat/more json UI additions by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/254

- Feat/improve plugin meta by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/255
- Feat/even more json UI components by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/256

Bug Fixes

- Fix/scripting addons by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/230
- Fix/convert start server to background task by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/233
- Fix/websocket messages by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/239

Documentation

- Docs/docker lan exposer by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/226
- Docs/feature documentation by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/232

Style & Refactoring

- Style/linting improvements by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/214

Performance

- Perf/minor websocket improvements by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/259

Tests

- Test/add-coverage-report by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/227

Build & CI

- Workflow/improved build lint test by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/215

Dependencies

- dep: bump eslint-plugin-prettier from 5.5.4 to 5.5.5 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/208
- dep: bump prettier from 3.7.4 to 3.8.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/213
- dep: update mcstatus requirement from <12.1,>=12.0 to >=12.0,<12.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/212
- dep: update myst-parser requirement from <4.1,>=4.0.0 to >=4.0.0,<5.1 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/211
- dep: update platformdirs requirement from <4.4,>=4.3.0 to >=4.3.0,<4.6 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/220

- dep: update black requirement from <25.13,>=25.1.0 to >=25.1.0,<26.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/219
- dep: update bsm-api-client[cli] requirement from <1.4.0,>=1.2.1 to >=1.2.1,<1.5.0 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/218
- dep: bump prettier from 3.8.0 to 3.8.1 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/223
- dep: bump globals from 17.0.0 to 17.1.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/224
- dep: update pytest-cov requirement from <7.0,>=6.0.0 to >=6.0.0,<8.0 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/228
- dep: bump globals from 17.1.0 to 17.2.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/231
- dep: bump globals from 17.2.0 to 17.3.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/234
- dep: bump @babel/preset-env from 7.28.6 to 7.29.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/235
- dep: bump @babel/core from 7.28.6 to 7.29.0 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/236
- dep: bump esbuild from 0.27.2 to 0.27.3 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/237
- dep: update mcstatus requirement from <12.2,>=12.0 to >=12.0,<12.3 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/242
- dep: update platformdirs requirement from <4.6,>=4.3.0 to >=4.3.0,<4.8 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/243
- dep: update fastapi requirement from <0.129,>=0.115.0 to >=0.115.0,<0.130 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/244
- dep: update uvicorn[standard] requirement from <0.41,>=0.35.0 to >=0.35.0,<0.42 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/246
- dep: update platformdirs requirement from <4.8,>=4.3.0 to >=4.3.0,<4.10 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/245
- dep: update isort requirement from <7.1,>=5.13.0 to >=5.13.0,<8.1 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/251

Chore

- Chore/cleanup repository by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/216
- Chore/cleanup frontend js by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/241
- Chore/add splash txt to app context by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/248
- Chore(v2 frontend)/minor backend improvements by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/252
- chore: set player count to 0 by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/253

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.7.2...3.8.0

22.11 3.7.2

Released on 2026-01-14 - GitHub - PyPI

> **Important**
>
> This is the last release to to support Python 3.10

> **Note**
>
> This release significantly changes the changelog format

22.11.1 What's Changed

Features

- Feat/3.7.2 by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/207

Documentation

- Docs/various updates by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/205

Build & CI

- build(deps-dev): update black requirement from <25.12,>=25.1.0 to >=25.1.0,<25.13 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/183
- build(deps-dev): bump eslint from 9.39.1 to 9.39.2 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/184
- build(deps-dev): bump esbuild from 0.27.1 to 0.27.2 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/188
- build(deps): update uvicorn[standard] requirement from <0.39,>=0.35.0 to >=0.35.0,<0.41 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/190
- build(deps): update psutil requirement from <7.2,>=7.0.0 to >=7.0.0,<7.3 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/191
- build(deps): update fastapi requirement from <0.125,>=0.115.0 to >=0.115.0,<0.129 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/192
- Action/upgrade during build lint test by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/193
- Workflow/auto label by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/199
- build(deps): update alembic requirement from <1.18,>=1.16 to >=1.16,<1.19 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/195
- build(deps-dev): update sphinx requirement from <8.3,>=8.2.0 to >=8.2.0,<9.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/194
- build(deps): bump actions/github-script from 7 to 8 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/202

- build(deps-dev): update sphinx-rtd-theme requirement from <3.1,>=3.0.0 to >=3.0.0,<3.2 by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/201
- build(deps-dev): bump @babel/preset-env from 7.28.5 to 7.28.6 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/204
- build(deps-dev): bump @babel/core from 7.28.5 to 7.28.6 in /frontend by @dependabot[bot] in https://github.com/DMedina559/bedrock-server-manager/pull/203

Chore

- Templates/improve issue templates by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/197
- Chore/label-cleanup by @DMedina559 in https://github.com/DMedina559/bedrock-server-manager/pull/206

Full Changelog: https://github.com/DMedina559/bedrock-server-manager/compare/3.7.1...3.7.2

22.12 3.7.1

Released on 2025-12-11 - GitHub - PyPI

22.12.1 New Features:

1. Re added environment variable support for data_dir and db_url
   - `BSM_DATA_DIR`
   - `BSM_DB_URL`
2. Added support for `.mcaddon`, `.mcpack`, and `.mcworld` file installations that starts with a top level folder

22.12.2 Bug Fixes:

3. Fixed various WebSocket issues

22.13 3.7.0

Released on 2025-11-23 - GitHub - PyPI

22.13.1 New Features:

1. Added WebSocket support
   - Various polling elements from the frontend has been replaced with WebSocket connections for real-time updates

22.13.2 Backend Changes:

2. BREAKING CHANGE: Removed `instances.py` module
   - Plugins and other modules should now access the `AppContext` object via dependency injection in FastAPI
3. Fixed Custom Downloads installs
4. Various dependency updates

22.14 3.6.3

Released on 2025-09-10 - GitHub - PyPI

22.14.1 New Features:

1. Added `reset-password` command to reset a users password
2. Added `database upgrade` command to upgrade the database to the latest version
    - Removed `migrate database` command
3. Added `--port` to `web start` command to override the port from config
4. Docker image now available on GitHub Container Registry (GHCR) and Docker Hub
    - Image: `ghcr.io/dmedina559/bedrock-server-manager:latest`, `dmedina559/bedrock-server-manager:latest`
    - Docs: Docker Guide

22.14.2 Backend Changes:

5. Add `reload` to `AppContext` and some related classes
6. Refactored WebUI js to npm build system
7. Added `build.sh` and `build.bat` scripts to build frontend and python packages

22.15 3.6.2

Released on 2025-08-24 - GitHub - PyPI

22.15.1 Bug Fixes:

1. Fixed missing `AppContext` in System service creation
2. Fixed systemd environment file credentials not migrating in `old-config` command
3. Potential fix for `serverName` required error in web UI setup

22.16 3.6.1

Released on 2025-08-24 - GitHub - PyPI

22.16.1 New Features:

1. Add `mcstatus`
    - Used to check online players count
    - `player_count` is returned with `get_all_servers_data`
2. Added `TaskManager` class in web.tasks
    - Used for routers that used FastAPI's BackgroundTask
    - Durring shutdown all running background tasks will be brought to the main thread where they'll finish running or graceful shutdown

22.16.2 Backend Changes:

3. Added shutdown flag to `BedrockProcessManager`
    - Raises `RuntimeError` if trying to register a server durring shutdown
4. Refactored/Removed web.templates into `AppContext`
5. Added `get_app_context` and `get_templates` to web.dependencies
6. Added `self.player_count` attribute to `BedrockServer`

22.17 3.6.0

Released on 2025-08-23 - GitHub - PyPI

```
With version 3.6.0, the web server is now a required component and acts as the central␣
↪management point for all Bedrock servers. This change simplifies process management␣
↪and improves stability.
Key improvements include:
All server processes are now managed directly by the web server.
Configuration data has been migrated from JSON files to a database for better␣
↪performance and reliability.
The previous backend CLI has been removed. All actions are now handled via the web␣
↪interface.
These changes create a more robust and efficient platform for managing your servers.
```

22.17.1 Important Notices & Breaking Changes

```
Stop servers before updating!
```

- BREAKING CHANGE: Removed CLI Plugin support
- Requires the BSM app to always be running (A System Service for the web server is recommended `bedrock-server-manager web service configure`)
- Its recommended to run the `bedrock-server-manager migrate old-config` command after updating to 3.6.0 to ensure smooth migration
    - If you plan on using an external database, you should run the `bedrock-server-manager setup` command first before running the `bedrock-server-manager migrate old-config` command
    - After running this command you should also run the `bedrock-server-manager setup` command to ensure basic migration was successful

22.17.2 Core Architectural Changes

- Moved most of the CLI commands to the `bsm-api-client[cli]` package
- Refactored most of the app away from global singletons to a passed around `AppContext` object
- Added gracefull shutdown for BSM
    - Shuts down all running bedrock servers
    - Unloads all plugins
- Web server now starts without needing any setup

22.17.3 Database and Configuration Migration

- Database migrations

```
Using the default database has been observed to cause slow loadding times for the␣
↪web server on some systems, if you experience this issue, consider using an␣
↪external database such as MySQL or PostgreSQL.
```

  - Migrated BSM from a JSON based config to a SQLite database
  - Use default included `bedrock_server_manager.db` or use your own external database
  - Automatically migrates from the old JSON config to the new database
  - All json config files are now stored in the database
    - This includes the global config, server configs, and plugin configs, player configs
  - All environment variables are now stored in the database
    - Web server environment variables are now stored in the database and are auto migrated to the database
    - App Data Directory is now stored in a seperate json file in the users `%APPDATA%` directory (`~/.config/bedrock-server-manager` on linux, `~/AppData/local/bedrock-server-manager` on Windows)
      - Database URL is also stored in this file

- Added `migrate` command
  - `migrate databse` command to migrate the database to new versions as needed (for future updates)
  - `migrate old-config` command to migrate from the old JSON config (<V3.5) to the new database
    - This command will migrate the following:
      - `bedrock_server_manager.json` (global config)
      - `<server_name>_config.json` (server configs)
      - `players.json` (global player)
      - `plugins.json` (plugins)`
      - `BEDROCK_SERVER_MANAGER_DATA_DIR` (app data directory)
      - `BEDROCK_SERVER_MANAGER_USERNAME`, `BEDROCK_SERVER_MANAGER_PASSWORD`, `BEDROCK_SERVER_MANAGER_TOKEN` (web auth environment variables)
      - Bedrock Server System Services

- Added `setup` command to setup App config
  - Sets up the App Data Directory and database URL, Web Host and Port, system service

22.17.4 Server Management

- Revamped server start methods
  - Now uses a generic process open instead of complex platform specific named pipe methods
    - This vastly improves cross platform compatibility
  - System Services for individual servers will no longer work, is now handled by the `autostart_plugin` during BSM startup

- Added crash restarts

  - Bedrock servers will now restart if they crashed

22.17.5 Multi-User and Access Control

- Added Multi user support with Access Control
  - Admins can now create and manage users with different access levels
    - Current access levels:
      - Admin: Full access to full application
      - Moderator: Limited to basic server management not including updates, installs, or content management
      - User: Limited to view privileges, can only view basic data
  - Admins can create a registration link for users to register
    - Link is valid for 24 hours
  - Admins can enable/disable user accounts, and change their roles
- Themes are now user specific
  - Each user can now have their own theme

22.17.6 System Service Improvements

- Improved system service support
  - On Windows you can now enter account credentials to run the service as a specific user
  - Added `--system` flag for linux to create a systemd service for the web server on the system level

22.17.7 Plugin and Developer Updates

- If you're a plugin developer with FASTAPI plugins, you should update your plugins to use the new `get_current_user`, `get_admin_user`, `get_moderator_user` dependency to ensure proper access control
- Added the `server_lifecycle_manager` as a Plugin Method
  - A context manager useful for plugins that need to perform actions on servers the requires the server to be stopped
- The `get_*_instance` functions in `instances.py` have been deprecated (will be removed in 3.7.0)
  - Instead the AppContext object (created in the CLI app) is passed around to all functions
  - FastAPI plugins can access the `AppContext` object via `app.state.app_context`
- Added `app_context` to the `PluginAPI` class, plugins can now access the AppContext object to access the current app state with `self.api.app_context`
  - AppContext will be automatically passed to all plugin methods
- Plugins are now only loaded during the web server startup, no longer during the CLI app startup

22.17.8 Other Backend Changes

- Various backend changes such as:
  - Object based authentication
  - Changes routers to use the new `get_current_user`, `get_admin_user`, `get_moderator_user` dependencies

- Added `AppContext` object
  - This object holds the current app state such as:
    - Settings instance
    - Database instance
    - BedrockServerManager instance
    - BedrockServer instances
    - BedrockProcessManager instance
    - PluginManager instance
  - Most API modules have been refactored to use the new AppContext object

22.18 3.5.7

Released on 2025-07-24 - GitHub - PyPI

```
Experience the new CLI before the 3.6.0 update now:

`pip install --upgrade bedrock-server-manager[cli]`

Use the `bsm-api-client` command to interact with the web server

View all commands at: [BSM API Commands](https://bedrock-server-manager.readthedocs.io/
↪en/latest/cli/api_client_commands.html)
```

1. Added generic task manager for FastAPI background task
   - Background task now retun a task ID you can use to monitor various task such as server installs, backups and more.
   - Added `/api/tasks/status/{task_id}` for status tracking
   - If your FASTAPI plugin uses background task, consider migrating to the `bedrock_server_manager.web.tasks` functions for a consistent user experience
2. Fixed server version api endpoint
3. Improved CLI permissions menu experience
4. No longer restart server after properties change

22.19 3.5.6

Released on 2025-07-19 - GitHub - PyPI

1. Fixed allowlist remove command
2. Filter logs to only show BSM and Plugin logs
3. Minor backend changes
4. Added various tests for the CLI and Web API
5. BREAKING CHANGE: Task scheduler functionality has been removed from the CLI and Web API.

```
The current task scheduler functionality will be reintroduced in a future as␣
↪optional plugins.
```

22.20 3.5.5

Released on 2025-07-17 - GitHub - PyPI

1. Hot fix for server properties path

22.21 3.5.4

Released on 2025-07-17 - GitHub - PyPI

1. Fixed custom cli/web plugins not being loaded correctly.

22.22 3.5.3

Released on 2025-07-17 - GitHub - PyPI

1. Optimized class initialization

22.23 3.5.2

Released on 2025-07-15 - GitHub - PyPI

1. Fixed missing html for Content Upload Plugin.
2. Hardcode uvicorn workers to 1 to avoid potential conflict with file locking and theme loading issues.

22.24 3.5.1

Released on 2025-07-14 - GitHub - PyPI

1. Fixed a bug where running the web server behind a reverse proxy was causing issues with the web UI and API routes.

22.25 3.5.0

Released on 2025-07-14 - GitHub - PyPI

```
BREAKING CHANGE: WEB USERS
If you are using the web server, you must regenerate your password hash and auth tokens.
```

```
Documentation for the Bedrock Server Manager (BSM) has been completely revamped and is␣
↪now available at:

Main:   [https://bedrock-server-manager.readthedocs.io/](https://bedrock-server-manager.
↪readthedocs.io/)

Mirror: [https://dmedina559.github.io/bedrock-server-manager/](https://dmedina559.github.
↪io/bedrock-server-manager/)
```

1. **CLI/WEB Plugins**: Experimental support for custom CLI and Web plugins

```
When using these new plugin types that extend Bedrock Server Manager (BSM)␣
↪functionality, exercise caution during installation and execution. Always ensure␣
↪you understand the plugin's operations before running it.
```

```
Plugins that add web endpoints have the potential to expose sensitive data. If you
↪'re developing such a plugin, it's highly recommended to use  the `bedrock_server_
↪manager.web` `get_current_user` dependencies within your FastAPI routers. This␣
↪helps secure your endpoints with the same authentication system the main web app␣
↪uses.
```

   - Plugins can add custom CLI commands or entities to the CLI menu
   - Plugins can add custom web API endpoints/UI menus
   - New default plugin: Upload Content Plugin (disabled by default)
   - Support for package format plugin
     - In addition to the standalone py file plugins, support for plugins in a python package format has been added

2. **Windows Service**:

   Running Bedrock Server Manager components (Web UI, individual Bedrock Servers) as Windows Services allows them to run in the background, start automatically on boot, and be managed by the Windows Service Control Manager.

```
It is **highly recommended** to use a Python installation downloaded from [python.
↪org](https://www.python.org/downloads/windows/) rather than the version available␣
↪from the Microsoft Store. The Microsoft Store version of Python has been observed␣
↪to cause significant file locking issues when used with Windows Services,␣
↪preventing other Python scripts or applications (including new BSM CLI sessions)␣
↪from running correctly, even when using virtual environments. Using the python.
↪org installer typically avoids these specific locking problems.
```

   - General Considerations:
     - Administrator Privileges: Creating, configuring, starting, and stopping Windows services requires Administrator privileges. You can achieve this by:
       * Running your Command Prompt or PowerShell as an Administrator before executing bsm commands.
       * Using the sudo command on recent versions of Windows if you have it configured.
         · See Microsoft Sudo for Windows documentation.
     - Service Account: By default, services created by BSM might be configured to run as the "Local System" account. For Bedrock servers, which often need to access user-specific paths for server files, content, and backups (e.g., in your user profile or a directory your user owns), it is usually necessary to change the service's "Log On As" account to your local Windows user account. This can be done via the Services app (services.msc) after the service has been created by BSM. This step ensures the service has the correct permissions. Failure to do this is a common cause of services not starting or functioning correctly.

3. **Web server as a service**:

   - Create a system service (Systemd on Linux / Windows Service).

- On Linux, you'll need to manually create an environment file with your BSM environment variables and add the `EnvironmentFile=` line to your systemd file.
- The `host` will be read from the JSON config file.
- On Windows, uvicorn workers will default to 1 instead of the configured settings value.

4. **Web UI Themes**:
   - Added support for custom themes in the Web UI.
   - Themes can be placed in the `themes` directory.
   - The Web UI will automatically list them in the settings menu.
   - Various themes are available, including:
     - `dark`
     - `light`
     - `red`
     - `blue`
     - `yellow`
     - `green`
     - `black`
     - `gradient`
   - The default theme is `dark` theme (original OreUI insipred).
5. **Config JSON migrations**:
   - The global `script_config.json` has been renamed to `bedrock_server_manager.json`.
   - Global settings have been migrated to a new nested format. A backup will be created, and an auto-migration will be attempted.
   - Server config JSONs have been migrated to a new nested format. Auto-migration will be attempted, and any custom config options will be moved to a nested `custom` section.
   - Added `web_server_host` to config. If the web server is started without a host argument, it will read from the JSON file.
   - `BASE_DIR` has been migrated to `paths.servers`.
6. **Background Task**:
   - API routes for `update`, `start`, `stop`, and other long running endpoints have been converted to FastAPI's BackgroundTasks for better responsiveness.
   - As a result Web APIs now instantly return a success response instead of the actual result for the task. In future versions, these endpoints will be updated to handle this.
7. **New plugin APIs**:
   - APIs to read global and server configurations.
   - API to set custom global and server configurations.
8. **Custom Bedrock Server Zips**
   - Added support for custom Bedrock Server zips.
   - These can be placed in the `DOWNLOAD_DIR/custom` directory.

- The Web/CLI UI will automatically list them in the server creation, when target version is set to `CUSTOM`.

9. Fixed world icon API route path typo: `word` -> `world`.
10. Plugin Event: The `before_server_stop` event no longer uses the `mode` variable.
11. Added a settings menu to the Web UI.

```
```{note}
Not all settings (like web host/port) will be reloaded on the fly. These require a full␣
↪application restart to take effect.
```
```

12. BREAKING CHANGE: Migrated Flask to FastAPI for the Web API.
    - This allows for better performance and more modern features.
    - The Web UI has been updated to work with the new FastAPI routes.
    - Always up-to-date HTTP API docs are now available in the Web UI footer `HTTP API` link or at `http(s)/bs.host.url/docs`.
    - The OpenAPI json are also available at `http(s)/bs.host.url/api/openapi.json`.
    - Switched to `uvicorn` as the ASGI server for the Web API.
    - Switched to `bcrypt` for password hashing in the Web API.
      - This requires you to regenerate your password hash and auth tokens.
13. BREAKING CHANGE: `/api/login` has been changed to `/auth/token`
14. Noteworthy change:
    - `/api/plugins/reload` method changed to `PUT` instead of `POST`.

22.26 3.4.1

Released on 2025-06-25 - GitHub - PyPI

1. Fixed settings target version for new server installations.
2. For simplicity, documentation has been removed from the web server and moved to the repository's `/docs` folder.

22.27 3.4.0

Released on 2025-06-25 - GitHub - PyPI

```
STOP SERVERS BEFORE UPDATING
```

1. **BREAKING CHANGE**: Linux `systemd` service files have been changed.
    - You must reconfigure autostart (`autostart --reconfigure`) to update your `systemd` service files.
    - This is due to the service `type` being changed from `forking` to `simple`.
    - BSM now directly manages the `bedrock_server` process, and `screen` has been removed as a dependency. You can uninstall it with `sudo apt remove screen`.
    - A note to Linux users: I apologize for the repeated `systemd` reconfigurations, but this should be the last time!
2. **The biggest feature since the web server: PLUGINS!**

- Plugins are a new way to extend BSM's functionality.
- Install plugins by placing the `.py` file in the plugins folder (e.g., `~/bedrock-server-manager/plugins`).
- Enable/disable plugins in `plugins.json` located in the config directory (e.g., `~/bedrock-server-manager/.config`). Plugins are disabled by default.
- See the Plugin Documentation for details.
- See default plugins for examples.
- Added a `plugin` command to the CLI. See the updated CLI Commands.
- Added plugin management to the CLI and Web UI.
- Default plugins are included and can be enabled, disabled, or overridden.
- A custom event system allows plugins to communicate with each other and external sources.

3. **BREAKING CHANGE**: Removed `/restore/all` route. This is now part of the `/restore/action` route, where `all` is a type parameter in the JSON body.
4. **BREAKING CHANGE**: Removed `/world_name` route. Use the get properties route instead.
5. Changed `/backups/list/` to `/backup/list/`.
6. Optimized various HTML routes. Pages now render faster, with data being pulled dynamically via JavaScript. The servers table in the Web UI updates automatically without a page refresh.
7. Improved resource usage monitoring by refactoring it into a generic `ResourceMonitor` class.
8. Improved filesystem functions by using `threading.Lock` to prevent race conditions.

22.28 3.3.1

Released on 2025-06-17 - GitHub - PyPI

1. Fixed panorama display in the Web UI.
2. Fixed server process handling in the Web API.
3. Fixed task command execution in the Web UI.

22.29 3.3.0

Released on 2025-06-16 - GitHub - PyPI

```
**MAJOR BREAKING CHANGES - READ FULL CHANGELOG BEFORE UPDATING**
```

```
**STOP SERVERS BEFORE UPDATING**

Start/Stop methods have been revamped. If you update before stopping your servers, you␣
↪may need to manually terminate the running `bedrock_server` processes.
```

1. **Revamped all CLI modules**: Any scripts or scheduled tasks using the old CLI commands must be recreated.
   - See the updated `CLI_COMMANDS.md` for the new command structure.
   - The CLI has been modernized using the `click` and `questionary` modules for a better interactive experience.
2. **Changed allowlist remove player route** to `/api/server/{server_name}/allowlist/remove`.

3. **Revamped start server actions**:
    - **Windows**: Starting a server now creates a named pipe for sending commands. The `-m/--mode` flag controls behavior (`detached` for background, `direct` for foreground).
    - **Linux**: The `start` command now incorporates `systemd` logic. The `-m/--mode` flag controls behavior (`detached` tries `systemctl` first, `direct` uses `screen`). You must recreate your `systemd` service files.
4. Added `reset-world` action to delete a server's world folder (available in CLI and Web UI).
5. Added command sending for actions like shutdown messages and reloading configurations.
6. Application can now be run as a module with `python -m bedrock_server_manager <command>`.
7. Changed `list-backups` `backup-type` from 'config' to 'allowlist', 'properties', or 'permissions'.
8. CLI and File log levels now have separate configuration values (`CLI_LOG_LEVEL`, `FILE_LOG_LEVEL`).
9. Fixed `stop-web-server` bug where it could kill the wrong process.
10. Removed getting started guides and other extra content from the web server.
11. Refactored core functions and classes (`BedrockServerManager`, `BedrockServer`, `BedrockDownloader`) for better structure and capabilities.
12. Migrated task schedulers to their own classes to reduce redundant logic.
13. General code cleanup, file reorganization, and removal of redundant functions.

22.30 3.2.5

Released on 2025-06-13 - GitHub - PyPI

1. Fixed the server download function to use the updated Minecraft website URL.

22.31 3.2.4

Released on 2025-05-22 - GitHub - PyPI

1. Fixed the `restart-server` command in the CLI.

22.32 3.2.3

Released on 2025-05-12 - GitHub - PyPI

1. Added `/api/server/{server_name}/backups/list/{type}` route.
2. Added `/api/content/worlds` route.
3. Added `/api/content/addons` route.
4. Added `/api/players/get` route.
5. Added `/api/players/add` route.
6. Added `/api/server/{server_name}/permissions_data` route.
7. File/folder paths for HTTP restore and prune APIs must now be relative.
8. Routes like `/api/servers/{server_name}/` are now `/api/server/{server_name}/`.
9. Fixed passing the `host` argument to the web server.

22.33 3.2.2

Released on 2025-05-06 - GitHub - PyPI

1. Fixed wrong module being used for the "read server properties" route.

22.34 3.2.1

Released on 2025-05-06 - GitHub - PyPI

1. Added `/api/info` route.
2. Added `/api/server/{server_name}/read_properties` route.
3. The server is now stopped before exporting a world.

22.35 3.2.0

Released on 2025-04-23 - GitHub - PyPI

1. Added "export world" functionality to the Web UI. The export now goes to the content folder.
2. Added "remove player from allowlist" to the Web UI and CLI.
3. Added a command blacklist for sensitive commands like `stop`.
4. Added more splash text.

22.36 3.1.4

Released on 2025-04-17 - GitHub - PyPI

1. Revised HTTP docs included with the web server.
2. Added `/api/servers` endpoint to list all servers and their status.

22.37 3.1.3

Released on 2025-04-16 - GitHub - PyPI

1. Revised docs included with the web server.

22.38 3.1.2

Released on 2025-04-14 - GitHub - PyPI

1. Fixed various JavaScript, allowlist, backup, and session token issues in the Web UI.

22.39 3.1.1

Released on 2025-04-13 - GitHub - PyPI

1. Fixed missing js files and images for web server
2. Added missing generate-password command
3. Fixed manage-script-config command

22.40 3.1.0

Released on 2025-04-13 - GitHub - PyPI

1. Added Web Server
    1. Environment Variable: BEDROCK_SERVER_MANAGER_USERNAME
        1. Required. Plain text username for web UI and API login.
    2. Environment Variable: BEDROCK_SERVER_MANAGER_PASSWORD
        1. Required. Hashed password for web UI and API login. Use the generate-password utility.
    3. Environment Variable: BEDROCK_SERVER_MANAGER_SECRET
        1. Strongly Recommended (Effectively Required for Web UI). A long, random, secret string. If not set, a temporary key is generated, and web UI sessions will not persist across restarts.
    4. Environment Variable: BEDROCK_SERVER_MANAGER_TOKEN
        1. Strongly Recommended. A long, random, secret string (different from _SECRET). If not set, a temporary key is generated, and JWT tokens used for API authentication will become invalid across restarts.
        2. JWT tokens expire every 4 weeks by default
    5. Set port in script_config.json : WEB_PORT
        1. Defaults to 11325
    6. Its recommended to run this behind a reverse proxy of your choice (NGINX, CADDY, etc etc)
    7. Uses Waitress WSGI server
    8. Customizable panorama
        1. Save any jpeg as panorama.jpeg in ./.config
    9. Mobile friendly experience
2. Added generate-password command
    1. Used to generate a hash to set as the BEDROCK_SERVER_MANAGER_PASSWORD Environment Variable
3. Added start-web-server command
    1. Command Arguments:
        1. -d | –debug : runs the server in the flask debug server
            1. Not recommend to use in production
        2. -m | –mode : direct, detached . Sets which mode to run the server
            1. direct: Directly runs the web server in the foreground
            2. detached: Runs the web server in a separate background process
        3. -H | –host : Which host to listen to
            1. Defaults to 127.0.0.1
4. Added stop-web-server command
    1. Stops a detached web server process
5. Removed all related lingering configuration (linux only)
6. Refactored cli.py and handlers.py into cli/api modules

7. Added more detailed logging throughout code
8. Added more detailed docstrings/comments throughout code
9. Added html docs which covers the http apis, cli environment, and user guide
10. Removed redundant commands
11. Added WEB_PORT and TOKEN_EXPIRE_WEEKS to script_config.json
12. Added various documentation accessible in the web server
13. Added splash text to Main Menu in CLI

22.41 3.0.3

Released on 2025-03-22 - GitHub - PyPI

1. Fixed Linux resource usage monitor
2. Fixed Linux systemd enable/disable

22.42 3.0.2

Released on 2025-03-21 - GitHub - PyPI

1. Fixed EXPATH variable in Linux scheduler
2. Fixed Modify Windows Task

22.43 3.0.1

Released on 2025-03-20 - GitHub - PyPI

22.43.1 3.0.1

1. Added handlers module
    - Refactored cli to use handlers
2. Refactored settings module
    - Migrated settings to a class
3. Fixed logger variables
4. [WIP] Send command support for Windows
    - Requires pywin32 module to be installed
    - Requires seperate start method not currently available

22.44 3.0.0

Released on 2025-03-14 - GitHub - PyPI

22.44.1 3.0.0

1. BREAKING CHANGE: Completely refactored .py script to a pip package
    - Now installed/updated with `pip` command
2. Use `BEDROCK_SERVER_MANAGER_DATA_DIR` env variable for default data location
    - Defaults to `$HOME/bedrock-server-manager` if variable doesnt exist
    - Follow your platforms documentation for setting Enviroment Variables
3. Logging refactored to use standard python logging
    - Most functions now raise Exceptions instead of returning an error code
4. Removed windows-start/stop commands
5. Added new commands
6. The following variables were added to script_config.json
    - CONTENT_DIR
    - DOWNLOAD_DIR
    - BACKUP_DIR
    - LOG_DIR
    - LOGS_KEEP
    - LOG_LEVEL

CHAPTER

TWENTYTHREE

INTRODUCTION TO PLUGINS

This guide explains how to manage and create plugins to extend and customize the Bedrock Server Manager. Plugins can add new features, notifications, and automation to your server management workflow.

23.1 How Plugins Work

Plugins are small Python scripts that "hook into" the Bedrock Server Manager to add functionality. You don't need to know how to code to use them. You can simply download a plugin and activate it.

23.1.1 Finding Plugins

1. **Locate the `plugins` directory:** Find the application's data directory. Inside, there will be a `plugins` folder. If it doesn't exist, the application will create it on its first run.
2. **Install a plugin:** To install a new plugin, simply place its Python file (e.g., `some_plugin.py`) inside this `plugins` directory.

23.2 Managing Plugins with the Web Server

You can easily control which plugins are active from the Web Server.

Changes made are saved immediately and will be applied the next time the application starts or reload.

> **Note**
>
> If you're changing the status of a FASTAPI plugin you must fully restart the web server

---

23.3 Configuration Storage

Plugin statuses, versions, and descriptions are stored in the database.

- **Location:** The database is located in your application's configuration directory (typically `~/.bedrock-server-manager/.config/`).

- **Functionality:** It stores a list of all known plugins and whether they are enabled (`true`) or disabled (`false`), their version, and description, and author.
- **Discovery:** When the application starts, it scans the `plugins` directory. Any new `.py` files found will be automatically added as `disabled`. You can then enable them through the web ui.

23.3.1 Default Plugins

A few essential built-in plugins are included in BSM to provide core functionality. You can enable/disable them if you wish. The current list includes:

- `server_lifecycle_notifications` (enabled by default)
- `world_operation_notifications` (enabled by default)
- `auto_reload_config` (enabled by default)
- `autostart_plugin` (enabled by default)
- `update_before_start` (enabled by default)
- `backup_on_start` (disabled by default)
- `content_uploader_plugin` (disabled by default)
- `download_page_plugin` (disabled by default)

23.4 Creating Your Own Plugins

Check out the *Plugin Developer Guide* for detailed instructions on how to create your own plugins. This guide covers everything from using the `PluginBase` class to using event hooks, and the plugin API.

CHAPTER

TWENTYFOUR

DEVELOPING PLUGINS

This guide will walk you through creating your own plugins to extend and customize the Bedrock Server Manager. The plugin system is designed to be simple yet powerful, allowing you to hook into various application events and use the core application's functions safely.

This guide assumes you have a basic understanding of Python programming.

For a complete list of all available event, see the *Available Events*. For a complete list of all available APIs, see the *Available APIs*.

24.1 1. Getting Started: Your First Plugin

1. **Locate a `plugins` directory:**
   - **User Plugins:** Find the application's data directory (typically `~/.bedrock-server-manager/` or where `BSM_DATA_DIR` points). Inside, there will be a `plugins` folder. This is for your custom plugins.
   - **Default Plugins:** The application also ships with default plugins located within its installation source at `src/bedrock_server_manager/plugins/default/`. While you can look here for examples, you should place your custom plugins in the user plugins directory.
   - **Root `plugins/` folder (for development/examples):** The main repository also contains a `plugins/` folder in its root. This is primarily for development-time examples and testing of the plugin system itself. For user-created plugins meant for regular use, the user plugins directory is preferred.
2. **Choose your plugin structure:** Plugins can be single Python files or complete Python packages (directories). This will be detailed in the next section.
3. **Write the code:** Create your plugin file(s) and define a class that inherits from `PluginBase`.

Here is the most basic "Hello World" plugin:

```
# my_first_plugin.py
from bedrock_server_manager import PluginBase, app_event

class MyFirstPlugin(PluginBase):
    """
    This is an example description that will be saved in the database
    """
    version = "1.0.0"  # Mandatory version attribute

    @app_event("on_load")
```

(continues on next page)

(continued from previous page)

```
    async def plugin_loaded(self, **kwargs):
        """This event is called when the plugin is loaded by the manager."""
        self.logger.info("Hello from MyFirstPlugin!")

    @app_event("after_server_start")
    async def after_server_start(self, **kwargs):
        """This event is called after a server has started."""
        server_name = kwargs.get("server_name")
        result = kwargs.get("result", {})
        if result.get("status") == "success":
            self.logger.info(f"Server '{server_name}' has started successfully!")
```

4. **Run the application:** Start the Bedrock Server Manager.
5. **Enable your plugin:** Navigate to the Web UI to activate it. You should see your "Hello from MyFirstPlugin!" message in the logs on the next startup or plugins reload.

24.2 2. Plugin Structures: Single File vs. Package

Bedrock Server Manager supports two primary ways to structure your plugin:

24.2.1 2.1. Single-File Plugin

This is the simplest structure, suitable for smaller plugins.

- Create a Python file (e.g., `my_simple_plugin.py`) directly in one of the plugin search paths (e.g., your user `plugins` directory).
- The filename (without the `.py` extension, so `my_simple_plugin` in this case) becomes the internal name of your plugin.
- Your `PluginBase` subclass must be defined within this file.

This is the structure used in the "Hello World" example above.

24.2.2 2.2. Package-Based Plugin (Directory)

For more complex plugins that might include multiple Python modules, templates, static files, or other resources, structuring your plugin as a Python package (a directory) is recommended.

- Create a directory in one of the plugin search paths (e.g., `plugins/my_packaged_plugin/`).
- The name of this directory (`my_packaged_plugin`) becomes the internal name of your plugin.
- Inside this directory, you **must** have an `__init__.py` file.
- The main `PluginBase` subclass for your plugin should be defined (or imported and made available) in this `__init__.py` file.

Example Directory Structure for a Packaged Plugin:

```
plugins/
└── my_packaged_plugin/       # Plugin Name: my_packaged_plugin
    ├── __init__.py           # Main plugin file, contains MyPluginClass(PluginBase)
    └── internal_logic.py     # Optional: other Python modules for your plugin
```

24.3 2.3. Plugin Metadata Attributes

To help the Plugin Manager identify and display your plugin correctly, your `PluginBase` subclass should define several metadata attributes:

- **`version`** (str) - *Required*. Used for versioning and updates.
- **`name`** (str) - Optional. The display name of the plugin (defaults to the module's file name).
- **`description`** (str) - Optional. A short sentence describing what your plugin does.
- **`author`** (str) - Optional. The creator of the plugin.
- **`dependencies`** (List[str]) - Optional. A list of plugin names that **must** load before this one.
- **`optional_dependencies`** (List[str]) - Optional. Plugins to load first *if* they are present.

```
from bedrock_server_manager import PluginBase, app_event

class MyAwesomePlugin(PluginBase):
    name = "My Awesome Automation Plugin"
    version = "1.0.0"
    author = "Jane Doe"
    description = "Automatically performs server backups and chat translations."
    dependencies = ["core_backup_plugin"]

    @app_event("on_load")
    async def plugin_loaded(self, **kwargs):
        self.logger.info(f"{self.name} v{self.version} by {self.author} loaded!")
```

This package structure allows for better organization. Python's standard import mechanisms (e.g., `from . import internal_logic`) will work within your plugin package.

24.4 3. The `PluginBase` Class

Every plugin **must** inherit from `bedrock_server_manager.PluginBase` (typically imported as `from bedrock_server_manager import PluginBase`). When your plugin is initialized, you are provided with three essential attributes:

- `self.name` (str): The name of your plugin, derived from its filename.
- `self.logger` (logging.Logger): A pre-configured Python logger. **Always use this for logging.**
- `self.api` (AppAPI): Your gateway to interacting with the main application. It dynamically exposes core application APIs to plugins safely and with robust type hints available in most modern IDEs.

Important

Important Plugin Class Requirements:

- **`version` Attribute (Mandatory):** Your plugin class **must** define a class-level attribute named `version` as a string (e.g., `version = "1.0.0"`). Plugins without a valid `version` attribute will not be loaded.
- **Description (from Docstring):** The description for your plugin is automatically extracted from the main docstring of your plugin class.

24.5 3. Understanding Event Hooks

Event hooks are methods from `PluginBase` that you can override. The Plugin Manager calls these methods when the corresponding event occurs.

- **`before_*` events:** Called *before* an action is attempted.
- **`after_*` events:** Called *after* an action has been attempted. They are always passed a `result` dictionary that you can inspect to see if the action succeeded or failed.

24.5.1 Asynchronous Architecture & Event Hooks (Version 4.0)

In Version 4.0, Bedrock Server Manager operates on a fully asynchronous architecture. Event handlers and lifecycle methods are defined using `async def` and `await` calls.

> **Important**
>
> **Plugin Boundaries & Architecture Standard:**
>
> Plugins **must not** attempt to import or access BSM internal objects (such as `app_context`, `BedrockServer`, or direct database sessions) directly. All interactions with the core application, background tasks, settings, and server operations must go exclusively through `PluginBase` methods and `self.api` methods (e.g., `await self.api.start_server(...)`, `await self.api.run_task(...)`, `await self.get_plugin_setting(...)`).

Define event handlers using the `@app_event` decorator with `async def`:

```
import asyncio
from bedrock_server_manager import PluginBase, app_event

class MyAsyncPlugin(PluginBase):
    version = "4.0.0"

    @app_event("before_server_start")
    async def handle_before_start(self, **kwargs):
        """This hook is awaited by the core application asynchronously."""
        server_name = kwargs.get("server_name")
        self.logger.info(f"Preparing to start {server_name} in 3 seconds...")

        # Non-blocking async operations
        await asyncio.sleep(3)

        # Call core APIs using await
        await self.api.send_command(server_name, "say Server starting in 3 seconds!")
        self.logger.info("Done waiting. Proceeding with server start.")
```

24.6 4. Advanced Topics

The plugin system offers many advanced features for deep integration:

- ***Custom Events & Interception*:** Learn how to send inter-plugin messages, trigger actions externally, and intercept/cancel core application operations before they happen.
- ***Plugin Settings & Storage*:** Discover how to persistently save and load configurations specific to your plugin within the application's database.

- ***Custom FastAPI Endpoints*:** Extend the web server itself by registering your own web routes, APIs, and Native JSON UI pages.
- ***Background Task Loops*:** Use the `@task_loop` decorator to easily schedule asynchronous or synchronous repeating background jobs without blocking the main event loop.

24.7 5. Best Practices

Tip

- **Always use `self.logger`:** Do not use `print()`. The provided logger is integrated with the application's logging system.
- **Handle exceptions:** Wrap API calls in `try...except` blocks to handle potential failures gracefully.
- **Check the `result` dictionary:** After an `after_*` event, inspect the `result['status']` to confirm the outcome.
- **Avoid blocking operations:** Long-running tasks in your event handlers or FastAPI endpoints can freeze the application. Use the *Task Manager* to offload them to background threads.
- **Use the API for operations:** Do not directly manipulate server files or directories. Use the provided `self.api` functions to ensure thread-safety and consistency.

CHAPTER

TWENTYFIVE

AVAILABLE APIS

This was auto-generated by the `api_docs_generator` plugin on 2026-09-24 19:55:52 UTC. Application Version: 4.0.0

This list contains all functions available to plugins via the `self.api` object. For an updated list of available APIs, please download and run the api_docs_generator plugin.

25.1 add_players_manually_api

```
await self.api.add_players_manually_api(player_strings: List[str], app_context: bedrock_
↪server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Adds or updates player data in the database.

Parameters:

Name	Type	Default
player_strings	List[str]	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.2 add_server_ban_api

```
await self.api.add_server_ban_api(app_context: bedrock_server_manager.context.AppContext,
↪ server_name: str, player_name: str, xuid: str, reason: str | None = None)
```

- **Async (Requires await):** Yes **Description:** Adds a player to the server ban list.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED
server_name	str	REQUIRED
player_name	str	REQUIRED
xuid	str	REQUIRED
reason	`str	None`

25.3 add_to_allowlist

```
await self.api.add_to_allowlist(server_name: str, new_players_data: List[Dict[str, Any]],
↪ app_context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Adds a list of players to the server's allowlist.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
new_players_data	List[Dict[str, Any]]	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.4 backup_all

```
await self.api.backup_all(server_name: str, app_context: bedrock_server_manager.context.
↪AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Performs a full backup of the server's world and configuration files.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.5 backup_config_file

```
await self.api.backup_config_file(server_name: str, file_to_backup: str, app_context:␣
↪bedrock_server_manager.context.AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Creates a backup of a specific server configuration file.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
file_to_backup	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.6 backup_world

```
await self.api.backup_world(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Creates a backup of the server's world directory.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.7 disable_addon

```
await self.api.disable_addon(server_name: str, pack_uuid: str, pack_type: str, app_
↪context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Disables an active addon for a server's active world, preserving files.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
pack_uuid	str	REQUIRED
pack_type	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.8 enable_addon

```
await self.api.enable_addon(server_name: str, pack_uuid: str, pack_type: str, app_
↪context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Enables a disabled addon for a server's active world.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
pack_uuid	str	REQUIRED
pack_type	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.9 export_world

```
await self.api.export_world(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext, export_dir: str | None = None, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Exports the server's currently active world to a .mcworld archive.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
export_dir	`str	None`	None`
stop_start_server	bool	True

25.10 get_all_global_settings

```
await self.api.get_all_global_settings(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Reads the entire global application settings configuration.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.11 get_all_known_players_api

```
await self.api.get_all_known_players_api(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves all player data from the database.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.12 get_all_server_settings

```
await self.api.get_all_server_settings(server_name: str, app_context: bedrock_server_
↪manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Reads the entire JSON configuration for a specific server from its

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.13 get_all_servers_data

```
await self.api.get_all_servers_data(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves status and version for all detected servers.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.14 get_allowlist

```
await self.api.get_allowlist(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the current allowlist for a given server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.15 get_bedrock_process_info

```
await self.api.get_bedrock_process_info(server_name: str, app_context: bedrock_server_
↪manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves resource usage for a running Bedrock server process.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.16 get_global_setting

```
await self.api.get_global_setting(key: str, app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Reads a single value from the global application settings.

Parameters:

Name	Type	Default
key	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.17 get_permissions

```
await self.api.get_permissions(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the permissions configuration for a server, formatted with player names.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.18 get_plugin_statuses

```
await self.api.get_plugin_statuses(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the statuses and metadata of all discovered plugins.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.19 get_properties

```
await self.api.get_properties(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the current server properties for a given server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.20 get_server_bans_api

```
await self.api.get_server_bans_api(app_context: bedrock_server_manager.context.
↪AppContext, server_name: str)
```

- **Async (Requires await):** Yes **Description:** Retrieves all bans for a specific server.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED
server_name	str	REQUIRED

25.21 get_server_running_status

```
await self.api.get_server_running_status(server_name: str, app_context: bedrock_server_
↪manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Checks if the server process is currently running.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.22 get_server_setting

```
await self.api.get_server_setting(server_name: str, key: str, app_context: bedrock_
↪server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Reads any value from a server's specific JSON configuration file

Parameters:

Name	Type	Default
server_name	str	REQUIRED
key	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.23 get_server_summary

```
await self.api.get_server_summary(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the summary information for a specific server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.24 get_system_and_app_info

```
self.api.get_system_and_app_info(app_context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** No **Description:** Retrieves basic system and application information.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.25 get_world_name

```
await self.api.get_world_name(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Retrieves the configured world name (`level-name`) for a server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.26 import_addon

```
await self.api.import_addon(server_name: str, addon_file_path: str, app_context: bedrock_
↪server_manager.context.AppContext, stop_start_server: bool = True, restart_only_on_
↪success: bool = True)
```

- **Async (Requires await):** Yes **Description:** Installs an addon to a specified Bedrock server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
addon_file_path	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True
restart_only_on_success	bool	True

25.27 import_world

```
await self.api.import_world(server_name: str, selected_file_path: str, app_context:␣
↪bedrock_server_manager.context.AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Imports a world from a .mcworld file, replacing the active world.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
selected_file_path	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.28 install_new_server

```
await self.api.install_new_server(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext, target_version: str = 'LATEST', server_zip_path: str | None = None)
```

- **Async (Requires await):** Yes **Description:** Installs a new server instance.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
target_version	str	'LATEST'
server_zip_path	`str	None`

25.29 list_available_addons

```
await self.api.list_available_addons(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Lists available .mcaddon and .mcpack files from the content directory.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.30 list_available_worlds_api

```
await self.api.list_available_worlds_api(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Lists available .mcworld files from the content directory.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.31 list_backup_files

```
await self.api.list_backup_files(server_name: str, backup_type: str, app_context:␣
↪bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Lists available backup files for a given server and type.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
backup_type	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.32 list_installed_addons

```
await self.api.list_installed_addons(server_name: str, app_context: bedrock_server_
↪manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Lists all addons for a server's active world.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.33 prune_download_cache

```
await self.api.prune_download_cache(download_dir: str, keep_count: int | None = None,␣
↪app_context: bedrock_server_manager.context.AppContext | None = None)
```

- **Async (Requires await):** Yes **Description:** Prunes old downloaded server archives (.zip) in a directory.

Parameters:

Name	Type	Default
download_dir	str	REQUIRED
keep_count	`int	None`
app_context	`bedrock_server_manager.context.AppContext	None`

25.34 prune_old_backups

```
await self.api.prune_old_backups(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Prunes old backups for a server based on retention settings.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.35 remove_from_allowlist

```
await self.api.remove_from_allowlist(server_name: str, player_names: List[str], app_
↪context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Removes a list of players from the server's allowlist.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
player_names	List[str]	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.36 reorder_addons

```
await self.api.reorder_addons(server_name: str, uuids: list[str], pack_type: str, app_
↪context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Reorders the active addons for a server's active world.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
uuids	list[str]	REQUIRED
pack_type	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.37 restart_server

```
await self.api.restart_server(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext, send_message: bool = True)
```

- **Async (Requires await):** Yes **Description:** Restarts the specified Bedrock server by orchestrating stop and start.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
send_message	bool	True

25.38 restore_all

```
await self.api.restore_all(server_name: str, app_context: bedrock_server_manager.context.
↪AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Restores the server from the latest available backups.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.39 restore_config_file

```
await self.api.restore_config_file(server_name: str, backup_file_path: str, app_context:␣
↪bedrock_server_manager.context.AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Restores a specific config file from a backup.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
backup_file_path	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.40 restore_world

```
await self.api.restore_world(server_name: str, backup_file_path: str, app_context:␣
↪bedrock_server_manager.context.AppContext, stop_start_server: bool = True)
```

- **Async (Requires await):** Yes **Description:** Restores a server's world from a specific backup file.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
backup_file_path	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
stop_start_server	bool	True

25.41 run_task

```
await self.api.run_task(target_function: Callable, app_context: bedrock_server_manager.
↪context.AppContext, username: str | None = None, args: Any, kwargs: Any)
```

- **Async (Requires await):** Yes **Description:** Submits a function to be run in the background by the TaskManager.

Parameters:

Name	Type	Default
target_function	Callable	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
username	`str	None`
args	Any	REQUIRED
kwargs	Any	REQUIRED

25.42 scan_and_update_player_db_api

```
await self.api.scan_and_update_player_db_api(app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Scans all server logs to discover and save player data.

Parameters:

Name	Type	Default
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.43 send_command

```
await self.api.send_command(server_name: str, command: str, app_context: bedrock_server_
↪manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Sends a command to a running Bedrock server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
command	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.44 server_lifecycle_manager

```
self.api.server_lifecycle_manager(server_name: str, stop_before: bool, app_context:␣
↪bedrock_server_manager.context.AppContext, start_after: bool = True, restart_on_
↪success_only: bool = False)
```

- **Async (Requires await):** No **Description:** A context manager to safely stop and restart a server for an operation.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
stop_before	bool	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
start_after	bool	True
restart_on_success_only	bool	False

25.45 set_custom_global_setting

```
await self.api.set_custom_global_setting(key: str, value: Any, app_context: bedrock_
↪server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Writes a custom value to the global application settings.

Parameters:

Name	Type	Default
key	str	REQUIRED
value	Any	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.46 set_permissions

```
await self.api.set_permissions(server_name: str, xuid: str, player_name: str | None,␣
↪permission: str, app_context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Sets a player's permission level for a given server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
xuid	str	REQUIRED
player_name	`str	None`
permission	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.47 set_properties

```
await self.api.set_properties(server_name: str, properties_to_update: Dict[str, str],␣
↪app_context: bedrock_server_manager.context.AppContext, restart_after_modify: bool =␣
↪False)
```

- **Async (Requires await):** Yes **Description:** Sets one or multiple server properties for a given server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
properties_to_update	Dict[str, str]	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
restart_after_modify	bool	False

25.48 set_server_custom_value

```
await self.api.set_server_custom_value(server_name: str, key: str, value: Any, app_
↪context: bedrock_server_manager.context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Writes a key-value pair to the 'custom' section of a server's specific

Parameters:

Name	Type	Default
server_name	str	REQUIRED
key	str	REQUIRED
value	Any	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.49 start_server

```
await self.api.start_server(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext)
```

- **Async (Requires await):** Yes **Description:** Starts the specified Bedrock server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.50 stop_server

```
await self.api.stop_server(server_name: str, app_context: bedrock_server_manager.context.
↪AppContext)
```

- **Async (Requires await):** Yes **Description:** Stops the specified Bedrock server.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED

25.51 update_server

```
await self.api.update_server(server_name: str, app_context: bedrock_server_manager.
↪context.AppContext, send_message: bool = True)
```

- **Async (Requires await):** Yes **Description:** Updates an existing server instance.

Parameters:

Name	Type	Default
server_name	str	REQUIRED
app_context	bedrock_server_manager.context.AppContext	REQUIRED
send_message	bool	True

25.52 validate_property_value

```
self.api.validate_property_value(property_name: str, value: str)
```

- **Async (Requires await):** No **Description:** Validates a specific server property value before it gets applied.

Parameters:

Name	Type	Default
property_name	str	REQUIRED
value	str	REQUIRED

CHAPTER

TWENTYSIX

AVAILABLE EVENTS

This was auto-generated by the `api_docs_generator` *plugin on 2026-09-24 19:55:52 UTC. Application Version: 4.0.0*

This list contains all core application events available to plugins. You can listen to these events using the `@app_event("event_name")` decorator. If an event is **Cancellable**, you can access `kwargs['event']` (which is a `CancellableEvent` object) and call `event.cancel(reason)` to halt the core operation.

26.1 after_add_server_ban

Description: Triggered after 'add_server_ban_api'. Adds a player to the server ban list.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.1.1 Listener Signature:

```
@app_event("after_add_server_ban")
def on_after_add_server_ban(self, server_name, player_name, xuid, reason, result,␣
↪**kwargs):
    pass
```

26.1.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`player_name`	`str`
`xuid`	`str`
`reason`	`Optional[str]`
`result`	`Dict[str, Any]`
`_triggering_plugin`	`str` (core, or plugin name)

26.2 after_addon_disable

Description: Triggered after 'disable_addon'. Disables an active addon for a server's active world, preserving files.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.2.1 Listener Signature:

```
@app_event("after_addon_disable")
def on_after_addon_disable(self, server_name, pack_uuid, pack_type, result, **kwargs):
    pass
```

26.2.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.3 after_addon_enable

Description: Triggered after 'enable_addon'. Enables a disabled addon for a server's active world.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.3.1 Listener Signature:

```
@app_event("after_addon_enable")
def on_after_addon_enable(self, server_name, pack_uuid, pack_type, result, **kwargs):
    pass
```

26.3.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.4 after_addon_import

Description: Triggered after 'import_addon'. Installs an addon to a specified Bedrock server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, addon_file_path`

26.4.1 Listener Signature:

```
@app_event("after_addon_import")
def on_after_addon_import(self, server_name, addon_file_path, stop_start_server, restart_
↪only_on_success, result, **kwargs):
    pass
```

26.4.2 Available kwargs:

Key Name	Type
server_name	str
addon_file_path	str
stop_start_server	bool
restart_only_on_success	bool
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.5 after_addon_reorder

Description: Triggered after 'reorder_addons'. Reorders the active addons for a server's active world.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.5.1 Listener Signature:

```
@app_event("after_addon_reorder")
def on_after_addon_reorder(self, server_name, uuids, pack_type, result, **kwargs):
    pass
```

26.5.2 Available kwargs:

Key Name	Type
server_name	str
uuids	list[str]
pack_type	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.6 after_addon_subpack_update

Description: Triggered after 'update_subpack'. Updates the active subpack for an addon on a server's active world.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.6.1 Listener Signature:

```
@app_event("after_addon_subpack_update")
def on_after_addon_subpack_update(self, server_name, pack_uuid, pack_type, subpack_name,␣
↪result, **kwargs):
    pass
```

26.6.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
subpack_name	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.7 after_addon_uninstall

Description: Triggered after 'uninstall_addon'. Uninstalls an addon for a server's active world, deleting its files.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.7.1 Listener Signature:

```
@app_event("after_addon_uninstall")
def on_after_addon_uninstall(self, server_name, pack_uuid, pack_type, result, **kwargs):
    pass
```

26.7.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.8 after_allowlist_change

Description: Triggered after 'remove_from_allowlist'. Removes a list of players from the server's allowlist.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.8.1 Listener Signature:

```
@app_event("after_allowlist_change")
def on_after_allowlist_change(self, server_name, player_names, result, **kwargs):
    pass
```

26.8.2 Available kwargs:

Key Name	Type
server_name	str
player_names	List[str]
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.9 after_backup

Description: Triggered after 'backup_all'. Performs a full backup of the server's world and configuration files.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, backup_type`

26.9.1 Listener Signature:

```
@app_event("after_backup")
def on_after_backup(self, server_name, stop_start_server, result, **kwargs):
    pass
```

26.9.2 Available kwargs:

Key Name	Type
server_name	str
stop_start_server	bool
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.10 after_command_send

Description: Triggered after 'send_command'. Sends a command to a running Bedrock server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, command`

26.10.1 Listener Signature:

```
@app_event("after_command_send")
def on_after_command_send(self, server_name, command, result, **kwargs):
    pass
```

26.10.2 Available kwargs:

Key Name	Type
server_name	str
command	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.11 after_delete_server_data

Description: Triggered after 'delete_server_data'. Deletes all data associated with a Bedrock server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.11.1 Listener Signature:

```
@app_event("after_delete_server_data")
def on_after_delete_server_data(self, server_name, stop_if_running, result, **kwargs):
    pass
```

26.11.2 Available kwargs:

Key Name	Type
server_name	str
stop_if_running	bool
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.12 after_permission_change

Description: Triggered after 'set_permissions'. Sets a player's permission level for a given server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.12.1 Listener Signature:

```
@app_event("after_permission_change")
def on_after_permission_change(self, server_name, xuid, player_name, permission, result,␣
↪**kwargs):
    pass
```

26.12.2 Available kwargs:

Key Name	Type
server_name	str
xuid	str
player_name	Optional[str]
permission	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.13 after_player_db_scan

Description: Triggered after 'scan_and_update_player_db_api'. Scans all server logs to discover and save player data.

- **Cancellable:** No

26.13.1 Listener Signature:

```
@app_event("after_player_db_scan")
def on_after_player_db_scan(self, result, **kwargs):
    pass
```

26.13.2 Available kwargs:

Key Name	Type
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.14 after_players_add

Description: Triggered after 'add_players_manually_api'. Adds or updates player data in the database.

- **Cancellable:** No

26.14.1 Listener Signature:

```
@app_event("after_players_add")
def on_after_players_add(self, player_strings, result, **kwargs):
    pass
```

26.14.2 Available kwargs:

Key Name	Type
player_strings	List[str]
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.15 after_properties_change

Description: Triggered after 'set_properties'. Sets one or multiple server properties for a given server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** server_name

26.15.1 Listener Signature:

```
@app_event("after_properties_change")
def on_after_properties_change(self, server_name, properties_to_update, restart_after_
↪modify, result, **kwargs):
    pass
```

26.15.2 Available kwargs:

Key Name	Type
server_name	str
properties_to_update	Dict[str, str]
restart_after_modify	bool
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.16 after_prune_backups

Description: Triggered after 'prune_old_backups'. Prunes old backups for a server based on retention settings.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.16.1 Listener Signature:

```
@app_event("after_prune_backups")
def on_after_prune_backups(self, server_name, result, **kwargs):
    pass
```

26.16.2 Available kwargs:

Key Name	Type
server_name	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.17 after_prune_download_cache

Description: Triggered after 'prune_download_cache'. Prunes old downloaded server archives (.zip) in a directory.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `download_dir, keep_count`

26.17.1 Listener Signature:

```
@app_event("after_prune_download_cache")
def on_after_prune_download_cache(self, download_dir, keep_count, result, **kwargs):
    pass
```

26.17.2 Available kwargs:

Key Name	Type
download_dir	str
keep_count	Optional[int]
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.18 after_remove_server_ban

Description: Triggered after 'remove_server_ban_api'. Removes a player from the server ban list.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.18.1 Listener Signature:

```
@app_event("after_remove_server_ban")
def on_after_remove_server_ban(self, server_name, xuid, result, **kwargs):
    pass
```

26.18.2 Available kwargs:

Key Name	Type
server_name	str
xuid	str
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.19 after_restore

Description: Triggered after 'restore_config_file'. Restores a specific config file from a backup.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, restore_type`

26.19.1 Listener Signature:

```
@app_event("after_restore")
def on_after_restore(self, server_name, backup_file_path, stop_start_server, result,␣
↪**kwargs):
    pass
```

26.19.2 Available kwargs:

Key Name	Type
server_name	str
backup_file_path	str
stop_start_server	bool
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.20 after_server_install

Description: Triggered after 'install_new_server'. Installs a new server instance.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, target_version`

26.20.1 Listener Signature:

```
@app_event("after_server_install")
def on_after_server_install(self, server_name, target_version, server_zip_path, result,␣
↪**kwargs):
    pass
```

26.20.2 Available kwargs:

Key Name	Type
server_name	str
target_version	str
server_zip_path	Optional[str]
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.21 after_server_players_change

Description: Triggered after 'update_server_player_stats_api'. Internal API to trigger player stat updates for websockets/plugins.

- **Cancellable:** No

26.21.1 Listener Signature:

```
@app_event("after_server_players_change")
def on_after_server_players_change(self, server_name, player_count, players, result,␣
↪**kwargs):
    pass
```

26.21.2 Available kwargs:

Key Name	Type
server_name	str
player_count	int
players	list
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.22 after_server_start

Description: Triggered after 'start_server'. Starts the specified Bedrock server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.22.1 Listener Signature:

```
@app_event("after_server_start")
def on_after_server_start(self, server_name, result, **kwargs):
    pass
```

26.22.2 Available kwargs:

Key Name	Type
server_name	str
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.23 after_server_status_change

Description: Triggered after 'set_server_status_api'. Internal API to set server status and trigger events.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, status`

26.23.1 Listener Signature:

```
@app_event("after_server_status_change")
def on_after_server_status_change(self, server_name, status, result, **kwargs):
    pass
```

26.23.2 Available kwargs:

Key Name	Type
server_name	str
status	str
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.24 after_server_stop

Description: Triggered after 'stop_server'. Stops the specified Bedrock server.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.24.1 Listener Signature:

```
@app_event("after_server_stop")
def on_after_server_stop(self, server_name, result, **kwargs):
    pass
```

26.24.2 Available kwargs:

Key Name	Type
server_name	str
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.25 after_server_update

Description: Triggered after 'update_server'. Updates an existing server instance.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, target_version`

26.25.1 Listener Signature:

```
@app_event("after_server_update")
def on_after_server_update(self, server_name, send_message, result, **kwargs):
    pass
```

26.25.2 Available kwargs:

Key Name	Type
server_name	str
send_message	bool
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.26 after_set_plugin_status

Description: Triggered after 'set_plugin_status'. Sets the enabled/disabled status for a specific plugin.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `plugin_name, new_status`

26.26.1 Listener Signature:

```
@app_event("after_set_plugin_status")
def on_after_set_plugin_status(self, plugin_name, enabled, result, **kwargs):
    pass
```

26.26.2 Available kwargs:

Key Name	Type
plugin_name	str
enabled	bool
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.27 after_set_server_setting

Description: Triggered after 'set_server_setting'. Writes any value to a server's specific JSON configuration file

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, key`

26.27.1 Listener Signature:

```
@app_event("after_set_server_setting")
def on_after_set_server_setting(self, server_name, key, value, result, **kwargs):
    pass
```

26.27.2 Available kwargs:

Key Name	Type
server_name	str
key	str
value	Any
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.28 after_setting_update

Description: Triggered after 'set_global_setting'. Writes a value to the global application settings.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `key`

26.28.1 Listener Signature:

```
@app_event("after_setting_update")
def on_after_setting_update(self, key, value, result, **kwargs):
    pass
```

26.28.2 Available kwargs:

Key Name	Type
key	str
value	Any
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.29 after_world_export

Description: Triggered after 'export_world'. Exports the server's currently active world to a .mcworld archive.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, export_dir`

26.29.1 Listener Signature:

```
@app_event("after_world_export")
def on_after_world_export(self, server_name, export_dir, stop_start_server, result,␣
↪**kwargs):
    pass
```

26.29.2 Available kwargs:

Key Name	Type
server_name	str
export_dir	Optional[str]
stop_start_server	bool
result	Dict[str, Any]
_triggering_plugin	str (core, or plugin name)

26.30 after_world_import

Description: Triggered after 'import_world'. Imports a world from a .mcworld file, replacing the active world.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name, file_path`

26.30.1 Listener Signature:

```
@app_event("after_world_import")
def on_after_world_import(self, server_name, selected_file_path, stop_start_server,␣
↪result, **kwargs):
    pass
```

26.30.2 Available kwargs:

Key Name	Type
server_name	str
selected_file_path	str
stop_start_server	bool
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.31 after_world_reset

Description: Triggered after 'reset_world'. Resets the server's world by deleting the active world directory.

- **Cancellable:** No
- **Identity Keys (for re-entrancy):** `server_name`

26.31.1 Listener Signature:

```
@app_event("after_world_reset")
def on_after_world_reset(self, server_name, result, **kwargs):
    pass
```

26.31.2 Available kwargs:

Key Name	Type
server_name	str
result	Dict[str, str]
_triggering_plugin	str (core, or plugin name)

26.32 before_add_server_ban

Description: Triggered before 'add_server_ban_api'. Adds a player to the server ban list.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.32.1 Listener Signature:

```
@app_event("before_add_server_ban")
def on_before_add_server_ban(self, server_name, player_name, xuid, reason, event,␣
↪**kwargs):
    pass
```

26.32.2 Available kwargs:

Key Name	Type
server_name	str
player_name	str
xuid	str
reason	Optional[str]
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.33 before_addon_disable

Description: Triggered before 'disable_addon'. Disables an active addon for a server's active world, preserving files.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.33.1 Listener Signature:

```
@app_event("before_addon_disable")
def on_before_addon_disable(self, server_name, pack_uuid, pack_type, event, **kwargs):
    pass
```

26.33.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.34 before_addon_enable

Description: Triggered before 'enable_addon'. Enables a disabled addon for a server's active world.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.34.1 Listener Signature:

```
@app_event("before_addon_enable")
def on_before_addon_enable(self, server_name, pack_uuid, pack_type, event, **kwargs):
    pass
```

26.34.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.35 before_addon_import

Description: Triggered before 'import_addon'. Installs an addon to a specified Bedrock server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, addon_file_path`

26.35.1 Listener Signature:

```
@app_event("before_addon_import")
def on_before_addon_import(self, server_name, addon_file_path, stop_start_server,␣
↪restart_only_on_success, event, **kwargs):
    pass
```

26.35.2 Available kwargs:

Key Name	Type
server_name	str
addon_file_path	str
stop_start_server	bool
restart_only_on_success	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.36 before_addon_reorder

Description: Triggered before 'reorder_addons'. Reorders the active addons for a server's active world.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.36.1 Listener Signature:

```
@app_event("before_addon_reorder")
def on_before_addon_reorder(self, server_name, uuids, pack_type, event, **kwargs):
    pass
```

26.36.2 Available kwargs:

Key Name	Type
server_name	str
uuids	list[str]
pack_type	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.37 before_addon_subpack_update

Description: Triggered before 'update_subpack'. Updates the active subpack for an addon on a server's active world.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.37.1 Listener Signature:

```
@app_event("before_addon_subpack_update")
def on_before_addon_subpack_update(self, server_name, pack_uuid, pack_type, subpack_name,
↪ event, **kwargs):
    pass
```

26.37.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
subpack_name	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.38 before_addon_uninstall

Description: Triggered before 'uninstall_addon'. Uninstalls an addon for a server's active world, deleting its files.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, pack_uuid`

26.38.1 Listener Signature:

```
@app_event("before_addon_uninstall")
def on_before_addon_uninstall(self, server_name, pack_uuid, pack_type, event, **kwargs):
    pass
```

26.38.2 Available kwargs:

Key Name	Type
server_name	str
pack_uuid	str
pack_type	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.39 before_allowlist_change

Description: Triggered before 'remove_from_allowlist'. Removes a list of players from the server's allowlist.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.39.1 Listener Signature:

```
@app_event("before_allowlist_change")
def on_before_allowlist_change(self, server_name, player_names, event, **kwargs):
    pass
```

26.39.2 Available kwargs:

Key Name	Type
server_name	str
player_names	List[str]
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.40 before_backup

Description: Triggered before 'backup_all'. Performs a full backup of the server's world and configuration files.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, backup_type`

26.40.1 Listener Signature:

```
@app_event("before_backup")
def on_before_backup(self, server_name, stop_start_server, event, **kwargs):
    pass
```

26.40.2 Available kwargs:

Key Name	Type
server_name	str
stop_start_server	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.41 before_command_send

Description: Triggered before 'send_command'. Sends a command to a running Bedrock server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, command`

26.41.1 Listener Signature:

```
@app_event("before_command_send")
def on_before_command_send(self, server_name, command, event, **kwargs):
    pass
```

26.41.2 Available kwargs:

Key Name	Type
server_name	str
command	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.42 before_delete_server_data

Description: Triggered before 'delete_server_data'. Deletes all data associated with a Bedrock server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.42.1 Listener Signature:

```
@app_event("before_delete_server_data")
def on_before_delete_server_data(self, server_name, stop_if_running, event, **kwargs):
    pass
```

26.42.2 Available kwargs:

Key Name	Type
server_name	str
stop_if_running	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.43 before_permission_change

Description: Triggered before 'set_permissions'. Sets a player's permission level for a given server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.43.1 Listener Signature:

```
@app_event("before_permission_change")
def on_before_permission_change(self, server_name, xuid, player_name, permission, event,␣
↪**kwargs):
    pass
```

26.43.2 Available kwargs:

Key Name	Type
server_name	str
xuid	str
player_name	Optional[str]
permission	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.44 before_player_db_scan

Description: Triggered before 'scan_and_update_player_db_api'. Scans all server logs to discover and save player data.

- **Cancellable:** Yes (`event.cancel()`)

26.44.1 Listener Signature:

```
@app_event("before_player_db_scan")
def on_before_player_db_scan(self, event, **kwargs):
    pass
```

26.44.2 Available kwargs:

Key Name	Type
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.45 before_players_add

Description: Triggered before 'add_players_manually_api'. Adds or updates player data in the database.

- **Cancellable:** Yes (`event.cancel()`)

26.45.1 Listener Signature:

```
@app_event("before_players_add")
def on_before_players_add(self, player_strings, event, **kwargs):
    pass
```

26.45.2 Available kwargs:

Key Name	Type
`player_strings`	`List[str]`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.46 before_properties_change

Description: Triggered before 'set_properties'. Sets one or multiple server properties for a given server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.46.1 Listener Signature:

```
@app_event("before_properties_change")
def on_before_properties_change(self, server_name, properties_to_update, restart_after_
↪modify, event, **kwargs):
    pass
```

26.46.2 Available kwargs:

Key Name	Type
server_name	str
properties_to_update	Dict[str, str]
restart_after_modify	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.47 before_prune_backups

Description: Triggered before 'prune_old_backups'. Prunes old backups for a server based on retention settings.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.47.1 Listener Signature:

```
@app_event("before_prune_backups")
def on_before_prune_backups(self, server_name, event, **kwargs):
    pass
```

26.47.2 Available kwargs:

Key Name	Type
server_name	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.48 before_prune_download_cache

Description: Triggered before 'prune_download_cache'. Prunes old downloaded server archives (.zip) in a directory.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `download_dir, keep_count`

26.48.1 Listener Signature:

```
@app_event("before_prune_download_cache")
def on_before_prune_download_cache(self, download_dir, keep_count, event, **kwargs):
    pass
```

26.48.2 Available kwargs:

Key Name	Type
download_dir	str
keep_count	Optional[int]
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.49 before_remove_server_ban

Description: Triggered before 'remove_server_ban_api'. Removes a player from the server ban list.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, xuid`

26.49.1 Listener Signature:

```
@app_event("before_remove_server_ban")
def on_before_remove_server_ban(self, server_name, xuid, event, **kwargs):
    pass
```

26.49.2 Available kwargs:

Key Name	Type
server_name	str
xuid	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.50 before_restore

Description: Triggered before 'restore_config_file'. Restores a specific config file from a backup.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, restore_type`

26.50.1 Listener Signature:

```
@app_event("before_restore")
def on_before_restore(self, server_name, backup_file_path, stop_start_server, event,␣
↪**kwargs):
    pass
```

26.50.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`backup_file_path`	`str`
`stop_start_server`	`bool`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.51 `before_server_install`

Description: Triggered before 'install_new_server'. Installs a new server instance.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, target_version`

26.51.1 Listener Signature:

```
@app_event("before_server_install")
def on_before_server_install(self, server_name, target_version, server_zip_path, event,␣
↪**kwargs):
    pass
```

26.51.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`target_version`	`str`
`server_zip_path`	`Optional[str]`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.52 `before_server_players_change`

Description: Triggered before 'update_server_player_stats_api'. Internal API to trigger player stat updates for websockets/plugins.

- **Cancellable:** Yes (`event.cancel()`)

26.52.1 Listener Signature:

```
@app_event("before_server_players_change")
def on_before_server_players_change(self, server_name, player_count, players, event,␣
↪**kwargs):
    pass
```

26.52.2 Available kwargs:

Key Name	Type
server_name	str
player_count	int
players	list
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.53 before_server_start

Description: Triggered before 'start_server'. Starts the specified Bedrock server.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.53.1 Listener Signature:

```
@app_event("before_server_start")
def on_before_server_start(self, server_name, event, **kwargs):
    pass
```

26.53.2 Available kwargs:

Key Name	Type
server_name	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.54 before_server_status_change

Description: Triggered before 'set_server_status_api'. Internal API to set server status and trigger events.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, status`

26.54.1 Listener Signature:

```
@app_event("before_server_status_change")
def on_before_server_status_change(self, server_name, status, event, **kwargs):
    pass
```

26.54.2 Available kwargs:

Key Name	Type
server_name	str
status	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.55 before_server_stop

Description: Triggered before 'stop_server'. Stops the specified Bedrock server.

- **Cancellable:** Yes (event.cancel())
- **Identity Keys (for re-entrancy):** server_name

26.55.1 Listener Signature:

```
@app_event("before_server_stop")
def on_before_server_stop(self, server_name, event, **kwargs):
    pass
```

26.55.2 Available kwargs:

Key Name	Type
server_name	str
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.56 before_server_update

Description: Triggered before 'update_server'. Updates an existing server instance.

- **Cancellable:** Yes (event.cancel())
- **Identity Keys (for re-entrancy):** server_name, target_version

26.56.1 Listener Signature:

```
@app_event("before_server_update")
def on_before_server_update(self, server_name, send_message, event, **kwargs):
    pass
```

26.56.2 Available kwargs:

Key Name	Type
server_name	str
send_message	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.57 before_set_plugin_status

Description: Triggered before 'set_plugin_status'. Sets the enabled/disabled status for a specific plugin.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `plugin_name, new_status`

26.57.1 Listener Signature:

```
@app_event("before_set_plugin_status")
def on_before_set_plugin_status(self, plugin_name, enabled, event, **kwargs):
    pass
```

26.57.2 Available kwargs:

Key Name	Type
plugin_name	str
enabled	bool
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.58 before_set_server_setting

Description: Triggered before 'set_server_setting'. Writes any value to a server's specific JSON configuration file

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, key`

26.58.1 Listener Signature:

```
@app_event("before_set_server_setting")
def on_before_set_server_setting(self, server_name, key, value, event, **kwargs):
    pass
```

26.58.2 Available kwargs:

Key Name	Type
server_name	str
key	str
value	Any
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.59 before_setting_update

Description: Triggered before 'set_global_setting'. Writes a value to the global application settings.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `key`

26.59.1 Listener Signature:

```
@app_event("before_setting_update")
def on_before_setting_update(self, key, value, event, **kwargs):
    pass
```

26.59.2 Available kwargs:

Key Name	Type
key	str
value	Any
event	CancellableEvent
_triggering_plugin	str (core, or plugin name)

26.60 before_world_export

Description: Triggered before 'export_world'. Exports the server's currently active world to a .mcworld archive.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, export_dir`

26.60.1 Listener Signature:

```
@app_event("before_world_export")
def on_before_world_export(self, server_name, export_dir, stop_start_server, event,␣
↪**kwargs):
    pass
```

26.60.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`export_dir`	`Optional[str]`
`stop_start_server`	`bool`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.61 `before_world_import`

Description: Triggered before 'import_world'. Imports a world from a .mcworld file, replacing the active world.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name, file_path`

26.61.1 Listener Signature:

```
@app_event("before_world_import")
def on_before_world_import(self, server_name, selected_file_path, stop_start_server,␣
↪event, **kwargs):
    pass
```

26.61.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`selected_file_path`	`str`
`stop_start_server`	`bool`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

26.62 `before_world_reset`

Description: Triggered before 'reset_world'. Resets the server's world by deleting the active world directory.

- **Cancellable:** Yes (`event.cancel()`)
- **Identity Keys (for re-entrancy):** `server_name`

26.62.1 Listener Signature:

```
@app_event("before_world_reset")
def on_before_world_reset(self, server_name, event, **kwargs):
    pass
```

26.62.2 Available `kwargs`:

Key Name	Type
`server_name`	`str`
`event`	`CancellableEvent`
`_triggering_plugin`	`str` (core, or plugin name)

CHAPTER

TWENTYSEVEN

PLUGIN BASE

Defines the abstract base class (ABC) for all plugins.

This module provides the *PluginBase* class, which serves as the foundational template for all plugins within the Bedrock Server Manager ecosystem. Plugins must inherit from this class to be recognized and loaded by the *PluginManager*.

By using the `@app_event("event_name")` decorator on their methods, plugins can subscribe to and react to specific events triggered by the core application or other parts of the server manager.

class bedrock_server_manager.plugins.plugin_base.**PluginBase**(*plugin_name: str*, *api: AppAPI*, *logger: Logger*)

Bases: `ABC`

The abstract base class (ABC) from which all plugins must inherit.

Plugins should subclass *PluginBase* and **must** define a class attribute named `version` (e.g., `version = "1.0.0"`). This version string is used by the *PluginManager* for metadata and potential compatibility checks.

Instances of concrete plugin subclasses are provided with the following attributes by the *PluginManager* during initialization:

name

The plugin's name, typically derived from its Python module filename (e.g., "my_plugin" for `my_plugin.py`).

Type
str

api

An instance of the API bridge, providing safe access to core application functions.

Type
`AppAPI`

logger

A pre-configured logger instance, specific to this plugin. Log messages will automatically include the plugin's name.

Type
`logging.Logger`

version

The plugin's own version string, copied from its class attribute.

Type
str

Plugins implement their functionality by using the `@app_event("event_name")` decorator on their methods to listen for specific application events. These methods are called by the *PluginManager* when corresponding application events occur.

author: str = 'N/A'

description: str = ''

dependencies: List[str] = []

optional_dependencies: List[str] = []

name: str = 'N/A'

async get_plugin_setting(*key: str*, *default: Any = None*) → Any

Retrieves a setting specific to this plugin.

Parameters

- **key** (*str*) – The setting key.
- **default** (*Any*) – The default value to return if the setting is not found.

Returns
The setting value or the default.

Return type
Any

async set_plugin_setting(*key: str*, *value: Any*) → Dict[str, Any]

Saves a setting specific to this plugin.

Parameters

- **key** (*str*) – The setting key.
- **value** (*Any*) – The value to save.

Returns
The result of the save operation.

Return type
Dict[str, Any]

get_fastapi_routers() → List[Any]

Called by the PluginManager after the plugin is loaded to retrieve any custom FastAPI routers (fastapi.APIRouter instances) the plugin wishes to register with the main web application.

Plugins should override this method to return a list of APIRouter objects.

Returns
A list of fastapi.APIRouter objects. Defaults to an empty list.

Return type
List[Any]

get_static_mounts() → List[tuple[str, Path, str]]

Called by the PluginManager after the plugin is loaded to retrieve configurations for mounting static file directories for this plugin.

Each configuration should be a tuple: *(mount_path, directory_path, name)*, suitable for *FastAPI.mount(mount_path, StaticFiles(directory=directory_path), name=name)*.

- ***mount_path* (str): The URL path prefix for these static files (e.g., "/static/myplugin").**
  This should be unique among plugins.
- ***directory_path* (Path): A *pathlib.Path* object pointing to the directory**
  containing the static files for this plugin.
- *name* (str): A unique name for this static mount (e.g., "myplugin_static").

Example

from pathlib import Path # Assuming static files are in a 'static' subdir relative to the plugin file static_dir = Path(__file__).parent / "static" return [("/static/myplugin", static_dir, "myplugin_static")]

Returns

A list of tuples, each for a static directory mount.
Defaults to an empty list.

Return type
List[tuple[str, Path, str]]

CHAPTER

TWENTYEIGHT

PLUGIN MANAGER

Manages plugin discovery, loading, configuration, lifecycle, and event dispatch.

This module is central to the plugin architecture of the Bedrock Server Manager. The *PluginManager* class handles all aspects of plugin interaction.

class bedrock_server_manager.plugins.plugin_manager.**PluginManager**(*state: Any*, *storage: Any*, *settings: Any*, *app_context:* AppContext | *None* = *None*)

Bases: `object`

Manages the discovery, loading, configuration, and lifecycle of all plugins.

async shutdown() → None

Gracefully shuts down the PluginManager, unloading all plugins and cleaning up resources.

get_native_ui_routes() → List[Dict[str, str]]

Returns a list of routes that are tagged for native UI rendering.

register_app_event_listener(*event_name: str*, *callback: Callable[[...], Any]*, *listening_plugin_name: str*) → None

Registers a callback function to listen for a specific application event.

async dispatch_event(*target_plugin:* PluginBase, *event_name: str*, **args: Any*, ***kwargs: Any*) → None

Asynchronously dispatches an event to a specific plugin instance.

Checks both internal slug (api._plugin_name) and display name (target_plugin.name) to ensure listeners are reliably found. Synchronous callbacks are executed off the event loop via asyncio.to_thread. Internal context references (e.g. app_context) are filtered out to prevent context leakage.

async trigger_event(*event_name: str*, **args: Any*, ***kwargs: Any*) → None

Asynchronously triggers an event on all loaded plugins.

async trigger_guarded_event(*event: str*, **args: Any*, ***kwargs: Any*) → None

Triggers an event only if the guard variable is not set in the environment.

async start_plugin_tasks() → None

Starts background tasks for all plugins that have methods decorated with @task_loop.

Includes idempotency checks to prevent duplicate task loops if called multiple times.

async load_plugins() → None

Discovers, loads, initializes, and starts tasks for all enabled plugins.

async unload_plugins() → None

Unloads all currently loaded plugins, cleans up background tasks, and purges imported modules.

async reload() → None

Gracefully reloads all plugins and restarts background tasks.

get_plugin_status(*plugin_name: str*) → str

Returns the runtime status of a given plugin ('LOADED', 'DISABLED', 'ERROR', or 'UNLOADED').

async unload_plugin_by_name(*plugin_name: str*) → bool

Unloads a single plugin by name, stopping its tasks and unregistering its listeners.

async load_plugin_by_name(*plugin_name: str*) → bool

Loads and initializes a single plugin by name if discoverable.

async reload_plugin(*plugin_name: str*) → bool

Reloads a single plugin by name.

async enable_plugin(*plugin_name: str*, *load_immediately: bool = True*) → bool

Enables a plugin in config and optionally loads it immediately.

async disable_plugin(*plugin_name: str*, *unload_immediately: bool = True*) → bool

Disables a plugin in config and optionally unloads it immediately.

CHAPTER

TWENTYNINE

CUSTOM EVENTS & ADVANCED HOOKS

29.1 Custom Plugin Events (Inter-Plugin Communication)

Plugins can define, send, and listen to their own custom events for complex interactions.

- **Sending Events:** Use `await self.api.send_event("myplugin:custom_action", arg1, kwarg1="value")`.
- **Listening for Events:** Decorate an async method with `@app_event("some:event")`.
- **Callback Arguments:** Your callback function will receive any `*args` and `**kwargs` from the sender.

29.1.1 Example: "I'm Home" Automation (Triggered via HTTP API)

An external system can trigger a plugin to start a server by sending a POST request to `/api/plugins/trigger_event` with a JSON body. The corresponding plugin would listen for this event:

```
# home_automation_starter_plugin.py
from bedrock_server_manager import PluginBase, app_event

TARGET_SERVER_NAME = "main_survival"

class HomeAutomationStarterPlugin(PluginBase):
    version = "4.0.0"

    @app_event("on_load")
    async def plugin_loaded(self, **kwargs):
        self.logger.info(f"Listening for 'automation:user_arrived_home' to start '
↪{TARGET_SERVER_NAME}'.")

    @app_event("automation:user_arrived_home")
    async def handle_user_arrival(self, **kwargs):
        user_id = kwargs.get('user_id', 'UnknownUser')
        self.logger.info(f"Received arrival event for user '{user_id}'.")

        status = await self.api.get_server_running_status(server_name=TARGET_SERVER_NAME)
        if status.get("is_running"):
            self.logger.info(f"Server '{TARGET_SERVER_NAME}' is already running.")
            return

        await self.api.start_server(server_name=TARGET_SERVER_NAME)
```

29.1.2 Advanced Event Hooks (Cancellation & Interception)

You can use `before_*` hooks to intercept actions. If an event is marked as **Cancellable**, the event payload will include a `CancellableEvent` object named `event` (or accessible via `kwargs['event']`). You can call `event.cancel(reason)` to completely halt the core application operation!

```
from bedrock_server_manager import app_event, PluginBase

class BackupBeforeStartPlugin(PluginBase):
    version = "4.0.0"

    @app_event("before_server_start")
    async def handle_before_server_start(self, **kwargs):
        """Runs automatically before a server is started."""
        server_name = kwargs.get("server_name")
        event = kwargs.get("event")
        self.logger.info(f"Intercepted start request for {server_name}. Running quick␣
↪backup...")

        result = await self.api.backup_world(server_name=server_name)

        if result.get("status") == "success":
            self.logger.info("Backup completed. Allowing server to start.")
        else:
            self.logger.error("Backup failed! Halting server start.")
            if event:
                event.cancel("Pre-start backup failed, aborting start for safety.")
```

29.1.3 Catch-All Event Listeners (Wildcard)

You can listen to *every* event dispatched by the Plugin Manager by using the wildcard * symbol in your `@app_event` decorator. This is useful for auditing, debugging, or logging tools.

```
from bedrock_server_manager import app_event, PluginBase

class AuditPlugin(PluginBase):
    version = "4.0.0"

    @app_event("*")
    async def on_any_event(self, event_name: str, **kwargs):
        # Note: The wildcard listener receives the original 'event_name' as its first␣
↪positional argument.
        self.logger.info(f"System fired event: {event_name} with args: {kwargs}")
```

CHAPTER

THIRTY

PLUGIN SETTINGS AND STORAGE

30.1 Managing Configuration

Plugins often need to store configuration data persistently.

30.1.1 1. Isolated Plugin Settings

Bedrock Server Manager provides built-in methods on `PluginBase` to safely read and write your plugin's configuration to the database without conflicting with other plugins or core settings. This data is stored globally for your plugin.

- **Saving Data:** `await self.set_plugin_setting(key="my_setting", value="my_value")`
- **Loading Data:** `await self.get_plugin_setting(key="my_setting", default="default_value")`

```
from bedrock_server_manager import app_event, PluginBase

class MyConfigurablePlugin(PluginBase):
    version = "4.0.0"

    @app_event("on_load")
    async def plugin_loaded(self, **kwargs):
        # Load existing settings or set defaults asynchronously
        self.plugin_config = await self.get_plugin_setting("config", default={"enable_
↪feature_x": True})

        if "api_key" not in self.plugin_config:
            self.logger.info("Initializing defaults.")
            self.plugin_config["api_key"] = "YOUR_KEY_HERE"

            # Save the updated configuration
            await self.set_plugin_setting("config", self.plugin_config)

        self.logger.info(f"Loaded config: {self.plugin_config}")
```

30.1.2 2. Custom Global Application Settings

If you need to store global settings outside of your plugin's isolated namespace, you can use the custom global setting API. These are stored in the database under the `custom.` namespace.

- **Saving Data:** `await self.api.set_custom_global_setting(key="my_global_key", value="my_value")`
- **Loading Data:** `await self.api.get_global_setting(key="custom.my_global_key")`

30.1.3 3. Server-Specific Custom Settings

You can also store custom settings that apply only to a specific Minecraft server.

- **Saving Server Data:** `await self.api.set_server_custom_value(server_name="survival_world", key="some_key", value="some_value")`
- **Loading Server Data:** `await self.api.get_server_setting(server_name="survival_world", key="custom.some_key")`

CHAPTER

THIRTYONE

CUSTOM FASTAPI ENDPOINTS

31.1 Extending Web Functionality

Plugins can significantly extend Bedrock Server Manager by adding their own custom FastAPI web endpoints. This allows for deep integration and tailored functionality.

To enable this, your plugin class (derived from `PluginBase`) needs to override one or both of the following methods:

- **`get_fastapi_routers(self) -> List[fastapi.APIRouter]`**: This method should return a list of FastAPI `APIRouter` instances that your plugin wants to add to the main web application.

The Plugin Manager will call these methods on your plugin instance after it's loaded. The collected commands and routers are then integrated into the main application.

31.1.1 Adding Custom FastAPI Endpoints (Web APIs and Pages)

To add web endpoints, define your FastAPI `APIRouter` instances and return them in a list from `get_fastapi_routers()`. These routers will be included in the main FastAPI application.

Example:

```
# my_web_api_plugin.py
from bedrock_server_manager import PluginBase
from fastapi import APIRouter, Depends
from fastapi.responses import JSONResponse

# Attempt to import authentication dependency; provide a fallback for isolated testing/
↪robustness
# There are three access roles admin, moderator, and user.

# - get_current_user: User, read only access APIs
# - get_moderator_user: Moderator, basic server management APIs, not including installs,␣
↪updates, or content management
# - get_admin_user: Admin, full access to all APIs

try:
    from bedrock_server_manager.web import get_current_user
    HAS_AUTH_DEP = True
except ImportError:
    HAS_AUTH_DEP = False
    async def get_current_active_user(): return {"username": "anonymous_plugin_user"} #␣
↪Dummy
```

(continues on next page)

(continued from previous page)

```
# Create an APIRouter instance
plugin_web_router = APIRouter(
    prefix="/my_web_plugin",  # URL prefix for all routes in this router
    tags=["My Web Plugin"],   # Tag for OpenAPI documentation (e.g., /docs)
    dependencies=[Depends(get_current_user)] if HAS_AUTH_DEP else [] # Secure all routes
)

@plugin_web_router.get("/info")
async def get_plugin_web_info():
    """Returns some information via the plugin's web API."""
    return {"plugin_name": "My Web API Plugin", "message": "API is active!"}

@plugin_web_router.post("/submit_data")
async def submit_data_to_plugin(data: dict):
    """A sample POST endpoint for the plugin."""
    # In a real plugin, you might use self.api here if you had access to it from the␣
↪router
    # or if the router was created within the plugin instance method that has `self`.
    # This example keeps the router definition self-contained for clarity.
    return {"status": "success", "received_data": data, "plugin_response": "Data␣
↪processed by My Web API Plugin."}

@plugin_web_router.get(
    "/ui",
    response_class=JSONResponse,
    name="My Plugin UI",
    tags=["plugin-json-ui"]  # <--- This tag enables the Native UI renderer
)
async def get_plugin_ui():
    """Serves a custom JSON UI page from the plugin."""
    return JSONResponse(content={
        "type": "Container",
        "children": [
            {
                "type": "Text",
                "props": {"content": "Hello from My Web Plugin's Custom JSON UI Page!",
↪"variant": "h1"}
            }
        ]
    })

from bedrock_server_manager import app_event

class MyWebAPIPlugin(PluginBase):
    version = "4.0.0" # Mandatory

    @app_event("on_load")
    async def plugin_loaded(self, **kwargs):
        self.logger.info(f"{self.name} v{self.version} loaded.")
        if not HAS_AUTH_DEP:
            self.logger.warning("Auth dependency 'get_current_user' not found. Plugin␣
↪API endpoints might be unsecured.")
```

(continues on next page)

(continued from previous page)

```
    def get_fastapi_routers(self):
        self.logger.info(f"Providing FastAPI router for '/my_web_plugin'.")
        return [plugin_web_router] # Return a list containing your router(s)
```

After enabling `my_web_api_plugin.py` and restarting the Bedrock Server Manager web server, you could access:

- `GET /my_web_plugin/info` (API endpoint)
- `POST /my_web_plugin/submit_data` (API endpoint, with a JSON body)
- `GET /my_web_plugin/ui` (Native JSON UI Page - visible in the Web Sidebar under Plugins)

These endpoints will also be listed in the OpenAPI documentation (e.g., at `/api/openapi.json` or `/docs`).

Native JSON UI

Bedrock Server Manager allows plugins to define native UI pages using a simple JSON schema. This eliminates the need for plugin developers to write frontend code (React, HTML, CSS) while still providing a rich, interactive user interface that matches the application's look and feel.

Instead of serving HTML or Jinja2 templates, your plugin defines a FastAPI route that returns a JSON response. This route is tagged with `plugin-json-ui`. The frontend detects this tag and renders the JSON using a dynamic component renderer.

For more information and available components, refer to the *Native JSON UI* documentation.

Tip

Tips for Plugin Web Endpoints:

- **Unique Prefixes & Mount Names:** Essential for routers and static mounts to avoid conflicts.
- **Authentication:** Apply as needed to your plugin's routers or individual routes.
- **Native JSON UI:** Tag your JSON UI routers with `plugin-ui` to have it added to the Web UI.

CHAPTER

THIRTYTWO

NATIVE JSON UI FOR PLUGINS

Bedrock Server Manager allows plugins to define native UI pages using a simple JSON schema. This eliminates the need for plugin developers to write frontend code (React, HTML, CSS) while still providing a rich, interactive user interface that matches the application's look and feel.

32.1 How it Works

Instead of serving HTML or Jinja2 templates, your plugin defines a FastAPI route that returns a JSON response. This route is tagged with `plugin-json-ui`. The frontend detects this tag and renders the JSON using a dynamic component renderer.

32.1.1 Global Server Context

The Web UI features a global server selector in the sidebar. When a user navigates to your plugin's Native UI page, the frontend automatically appends the currently selected server name to the URL as a query parameter (e.g., `?server=survival_server`).

Your FastAPI route should accept this `server` query parameter to dynamically fetch and display data for the chosen server. If the user changes the server in the dropdown, your UI route will be re-fetched with the new `server` parameter.

Example:

```
@self.router.get("/my_plugin/ui", response_class=JSONResponse, tags=["plugin-json-ui"])
async def get_ui(server: str = "default_server"): # Accept the server query param
    # Fetch data specific to the selected server using the Core API
    status = await self.api.get_server_running_status(server)
    is_running = status.get("is_running", False)

    return JSONResponse(content={
        "type": "Card",
        "props": {"title": f"Status for {server}"},
        "children": [
            {
                "type": "Badge",
                "props": {
                    "content": "Online" if is_running else "Offline",
                    "variant": "success" if is_running else "danger"
                }
            }
        ]
    })
```

32.1.2 Basic Example

Here is a minimal example of a plugin that adds a native UI page:

```
from fastapi import APIRouter, Request
from fastapi.responses import JSONResponse
from bedrock_server_manager import PluginBase

class MyPlugin(PluginBase):
    def on_load(self, **kwargs):
        self.router = APIRouter(tags=["My Plugin"])

        @self.router.get(
            "/my_plugin/ui",
            response_class=JSONResponse,
            name="My Plugin UI",
            tags=["plugin-json-ui"]  # <--- This tag enables the Native UI renderer
        )
        async def get_ui(request: Request):
            return JSONResponse(content={
                "type": "Container",
                "children": [
                    {
                        "type": "Text",
                        "props": {"content": "Hello from Native UI!", "variant": "h1"}
                    },
                    {
                        "type": "Card",
                        "props": {"title": "My Card"},
                        "children": [
                            {"type": "Text", "props": {"content": "This is content␣
↪inside a card."}}
                        ]
                    }
                ]
            })

    def get_fastapi_routers(self, **kwargs):
        return [self.router]
```

32.2 Available Components

The JSON schema supports a wide range of components. Each component is an object with `type`, `props`, and optional `children`.

32.2.1 Layout

- **Container**: Main wrapper.
- **Card**: Boxed container with an optional `title`.
- **Row**: Horizontal layout.
- **Column**: Vertical layout.
- **Divider**: Horizontal line separator.

32.2.2 Typography

- **Text**: Renders text.
  - `content`: The text to display.
  - `variant`: `h1`, `h2`, `h3`, or `body`.

32.2.3 Basic Inputs

- **Input**: Text input.
- **Select**: Dropdown menu.
- **Switch**: Toggle switch.
- **Checkbox**: Checkbox input.
- **Button**: Clickable button.
  - `label`: Text on the button.
  - `icon`: Name of the Lucide icon (e.g., "Save", "Play").
  - `variant`: `primary`, `secondary`, `danger`.
  - `onClickAction`: Action definition (see Actions below).
- **FileUpload**: File selection input.

32.2.4 Display & Media

- **Badge**: Status label.
  - `content`: Text.
  - `variant`: `primary`, `secondary`, `success`, `danger`, `warning`.
- **CodeBlock**: Displays code or logs with a copy button.
  - `content`: The code/text.
  - `title`: Optional header.
- **Image**: Displays an image.
- **Link**: External or internal link.
- **iframe**: Embeds external content.

32.2.5 Interactive

- **Accordion**: Collapsible section.
- **Modal**: Dialog popup.
  - `id`: Unique ID to control visibility.
  - `title`: Header text.
  - `children`: Content inside the modal.

32.2.6 Data & Monitoring

- **Table**: Displays tabular data.
- **StatCard**: Dashboard metric card.
  - `label`: Title of the metric.
  - `value`: The value to display.
  - `icon`: Icon name.
  - `trend`: `up`, `down`, or `neutral`.
- **StatusIndicator**: Dot with status text.
  - `status`: `running` (green), `stopped` (red), `warning` (orange).
- **LogViewer**: Scrolling log display.
  - `lines`: Array of log strings.
  - `height`: Height in pixels (default 200).
- **Chart**: Data visualization (Area, Line, Bar).
  - `type`: `area`, `line`, `bar`.
  - `data`: Array of data objects.
  - `xAxis`: Key for the X-axis (e.g., "time").
  - `series`: Array of objects defining lines/bars (`dataKey`, `color`, `name`).

32.3 Actions

Components like `Button` can trigger actions via the `onClickAction` prop.

32.3.1 API Call

Sends a POST request to a backend endpoint.

```
{
    "type": "api_call",
    "endpoint": "/api/my_plugin/submit",
    "includeFormState": true,  // Sends current input values
    "refresh": true,           // Refreshes the UI after success
    "closeModal": true         // Closes any open modal
}
```

32.3.2 Navigate

Navigates to another URL or updates query parameters.

```
{
    "type": "navigate",
    "url": "/another/page"
}
```

32.3.3 Modal Control

Opens or closes a modal by ID.

```
{
    "type": "open_modal",
    "modalId": "my_modal_id"
}
```

```
{
    "type": "close_modal"
}
```

32.4 Real-time Updates (WebSockets)

You can subscribe to WebSocket topics to update components like Charts, StatCards, and LogViewers in real-time.

1. **Define Subscriptions**: Add `websocketSubscriptions` to the root of your JSON response.

   ```
   {
       "websocketSubscriptions": ["my_plugin:stats", "server_log:{server}"],
       "type": "Container"
   }
   ```

   - `{server}` is automatically replaced with the currently selected server name.

2. **Bind Components**: Pass `socketTopic` to components.
   - **Chart**: Appends new data points from the socket message to the chart history.
   - **LogViewer**: Appends new log lines.
   - **StatCard**: Updates the value. Use `dataKey` to specify which field in the socket message to display (e.g., `process_info.cpu_percent`).

   ```
   {
       "type": "StatCard",
       "props": {
           "label": "CPU",
           "socketTopic": "my_plugin:stats",
           "dataKey": "cpu_usage"
       }
   }
   ```

32.5 Icons

The system uses Lucide React icons. You can use any valid Lucide icon name in props like `icon`. Common icons include: `Activity`, `AlertCircle`, `CheckCircle2`, `Info`, `Terminal`, `Save`, `Trash2`, `Plus`, `X`, `Upload`, `Download`, `Play`, `Square`, `RotateCcw`.

32.6 Comprehensive Examples

32.6.1 Form and API Submission

This example creates a simple form inside a Card and a Save button that submits the form data to an API endpoint. The `includeFormState: true` prop ensures all input values on the page are sent in the POST request body.

```
{
  "type": "Container",
  "children": [
    {
      "type": "Card",
      "props": {
        "title": "Plugin Settings"
      },
      "children": [
        {
          "type": "Input",
          "props": {
            "name": "api_key",
            "label": "External API Key",
            "type": "password",
            "placeholder": "Enter your key..."
          }
        },
        {
          "type": "Switch",
          "props": {
            "name": "enable_feature_x",
            "label": "Enable Feature X",
            "defaultChecked": true
          }
        },
        {
          "type": "Button",
          "props": {
            "label": "Save Configuration",
            "icon": "Save",
            "variant": "primary",
            "onClickAction": {
              "type": "api_call",
              "endpoint": "/api/plugins/my_plugin/save_settings",
              "method": "POST",
              "includeFormState": true,
              "refresh": true
            }
          }
        }
      ]
    }
  ]
}
```

32.6.2 Table View

Displaying data in a table with customizable columns and rows.

```
{
  "type": "Card",
  "props": {
    "title": "Recent Actions"
  },
  "children": [
    {
      "type": "Table",
      "props": {
        "columns": [
          { "header": "User", "accessor": "user" },
          { "header": "Action", "accessor": "action" },
          { "header": "Timestamp", "accessor": "timestamp" },
          { "header": "Status", "accessor": "status" }
        ],
        "data": [
          { "user": "admin", "action": "Server Start", "timestamp": "2023-10-27 10:00",
↪"status": "Success" },
          { "user": "mod_1", "action": "Config Update", "timestamp": "2023-10-27 11:30",
↪"status": "Success" },
          { "user": "admin", "action": "Backup Run", "timestamp": "2023-10-27 12:00",
↪"status": "Failed" }
        ]
      }
    }
  ]
}
```

32.6.3 Real-Time Monitoring Dashboard

A combination of layout components (`Row`, `Column`) and real-time components (`StatCard`, `LogViewer`) bound to WebSocket topics. Notice the root `websocketSubscriptions` array.

```
{
  "websocketSubscriptions": [
    "server_process_stats:{server}",
    "server_log:{server}"
  ],
  "type": "Container",
  "children": [
    {
      "type": "Text",
      "props": {
        "content": "Live Server Telemetry",
        "variant": "h2"
      }
    },
    {
      "type": "Row",
      "children": [
```

(continues on next page)

(continued from previous page)

```
        {
          "type": "Column",
          "children": [
            {
              "type": "StatCard",
              "props": {
                "label": "CPU Usage",
                "value": "0%",
                "icon": "Cpu",
                "socketTopic": "server_process_stats:{server}",
                "dataKey": "cpu_percent_str",
                "trend": "neutral"
              }
            }
          ]
        },
        {
          "type": "Column",
          "children": [
            {
              "type": "StatCard",
              "props": {
                "label": "Memory",
                "value": "0 MB",
                "icon": "MemoryStick",
                "socketTopic": "server_process_stats:{server}",
                "dataKey": "memory_str"
              }
            }
          ]
        }
      ]
    },
    {
      "type": "Card",
      "props": {
        "title": "Live Console"
      },
      "children": [
        {
          "type": "LogViewer",
          "props": {
            "socketTopic": "server_log:{server}",
            "lines": ["--- Waiting for logs ---"],
            "height": 300
          }
        }
      ]
    }
  ]
}
```

CHAPTER

THIRTYTHREE

USING THE TASK MANAGER

When developing plugins for Bedrock Server Manager, you may encounter situations where a task takes a long time to complete (e.g., downloading large files, processing backups, making complex external API calls).

Running these tasks directly within an event hook or a FastAPI endpoint blocks the main application thread or event loop. This can cause the web interface to become unresponsive and potentially lead to WebSocket disconnections.

To solve this, Bedrock Server Manager provides a `TaskManager` to offload long-running operations to background threads.

33.1 Submitting Background Tasks via `self.api.run_task`

In Version 4.0, plugins interact with background task scheduling exclusively through the safe `self.api.run_task` method. Plugins **must not** attempt to access `app_context` or `TaskManager` objects directly.

The `await self.api.run_task(...)` method accepts the target function to execute along with any required positional or keyword arguments, submitting the function to be executed in the background and returning a unique `task_id`.

33.1.1 Example: A Long-Running Task

Here is an example of a plugin that provides a web endpoint to trigger a long-running background task.

```
import time
import logging
from fastapi import APIRouter, Depends
from bedrock_server_manager import PluginBase

# It is recommended to import the application context dependency
# to cleanly access the task manager from decoupled FastAPI routers.
from bedrock_server_manager.web.dependencies import get_app_context
from bedrock_server_manager.web.auth_utils import get_current_user
from bedrock_server_manager.context import AppContext
from bedrock_server_manager.web.schemas import User

logger = logging.getLogger(__name__)

plugin_web_router = APIRouter(
    prefix="/my_task_plugin",
    tags=["My Task Plugin"]
)

def my_long_running_function(seconds: int, name: str):
```

(continues on next page)

(continued from previous page)

```
    """This function runs in the background."""
    logger.info(f"Starting long task for {name}...")
    # Simulate a long-running process
    time.sleep(seconds)
    logger.info(f"Finished long task for {name}!")
    return f"Processed {name} successfully in {seconds} seconds."

class MyTaskPlugin(PluginBase):
    version = "4.0.0"

    @app_event("on_load")
    async def plugin_loaded(self, **kwargs):
        self.logger.info("MyTaskPlugin loaded.")

    def get_fastapi_routers(self):
        router = APIRouter(prefix="/my_task_plugin", tags=["My Task Plugin"])

        @router.post("/start_task")
        async def trigger_task(
            seconds: int = 5,
            name: str = "example",
            current_user: Dict[str, Any] = Depends(get_admin_user)
        ):
            # Submit the function to run in the background via self.api.run_task.
            task_id = await self.api.run_task(
                my_long_running_function,
                username=current_user.get("username"),
                seconds=seconds,
                name=name
            )

            return {"status": "success", "task_id": task_id, "message": "Task started in␣
↪the background."}

        return [router]
```

33.2 Tracking Task Status and UI Updates

When you submit a task using `run_task`, the `TaskManager` automatically tracks its status (`in_progress`, `success`, `error`).

The task manager returns a dictionary containing the task details:

```
{
    "status": "in_progress", // Or "success", "error"
    "message": "Task is running.", // Or the error/success message
    "result": null, // Will contain the return value of your function upon success
    "username": "admin"
}
```

33.2.1 WebSocket Notifications

If you provide the `username` argument when calling `run_task`, the `TaskManager` will automatically send WebSocket notifications to that specific user whenever the task status updates (e.g., when it completes or fails).

The frontend can listen for these notifications on the `task:{task_id}` topic to update the UI without needing to poll the `task_status` endpoint repeatedly.

33.2.2 Handling Exceptions

If your background function raises an exception, the task manager will catch it, log the error using the main application logger, and update the task's status to `error`. The exception message will be stored in the task's `message` field.

33.3 Using `@task_loop` for Periodic Tasks

If you need a task to run continuously in the background on a fixed interval (e.g., polling an external API, performing cleanups), you don't need to manually interact with the `TaskManager`.

Instead, use the `@task_loop(interval)` decorator provided by the plugin system. The `PluginManager` will automatically schedule these methods as background tasks when your plugin is loaded and cancel them when it is unloaded.

Both synchronous and asynchronous methods are supported.

```
import asyncio
import time
from bedrock_server_manager import PluginBase
from bedrock_server_manager.plugins.task_loop import task_loop

class MyPollingPlugin(PluginBase):
    version = "1.0.0"

    # Async example
    @task_loop(interval=300) # Runs every 5 minutes (300 seconds)
    async def poll_remote_service_async(self):
        self.logger.info("Polling remote service asynchronously...")
        # Simulate an I/O bound request safely without blocking the event loop
        await asyncio.sleep(2)
        self.logger.info("Async polling complete.")

    # Sync example
    @task_loop(interval=60) # Runs every 1 minute
    def clean_up_local_files_sync(self):
        self.logger.info("Cleaning up files synchronously...")
        # The Plugin Manager automatically runs this in a thread pool
        # so time.sleep won't freeze the main app loop.
        time.sleep(1)
        self.logger.info("Sync cleanup complete.")
```

CHAPTER

THIRTYFOUR

API DOCUMENTATION

The functions listed in this part represent the primary, high-level interface for interacting with the Bedrock Server Manager. These APIs are used by all official components, including the Command-Line Interface (CLI), the Web, and the Plugin system.

These APIs provide a safe, consistent, and stable way to manage servers and the application itself.

> **Note**
>
> **For Plugin Developers:** The `self.api` object in your plugin exposes these APIs for your usage. However, to ensure consistency and safety of user data, only a specific **subset** of these APIs are endorsed for plugin use. Please refer to the *Available APIs* sections of the Plugins Docs for the full list of functions that are guaranteed to be stable and supported for plugins.

34.1 Server API Documentation

Provides API functions for managing Bedrock server instances.

This module serves as a key interface layer for server-specific operations within the Bedrock Server Manager. It leverages the *BedrockServer* core class to perform a variety of actions such as server lifecycle management (starting, stopping, restarting), configuration (getting/setting server-specific properties), and command execution.

The functions within this module are designed to return structured dictionary responses, making them suitable for consumption by web API routes, command-line interface (CLI) commands, or other parts of the application. This module also integrates with the plugin system by exposing many of its functions as callable APIs for plugins (via `api_method()`) and by triggering various plugin events during server operations.

async bedrock_server_manager.api.server.**get_server_setting**(*server_name: str*, *key: str*, *app_context:* AppContext) → Dict[str, Any]

Reads any value from a server's specific JSON configuration file (e.g., `<server_name>_config.json`) using dot-notation for keys.

Parameters

- **server_name** (*str*) – The name of the server.
- **key** (*str*) – The dot-notation key to read from the server's JSON configuration (e.g., "server_info.status", "settings.autoupdate", "custom.my_value").

Returns

A dictionary containing the operation result. On success: `{"status": "success", "value": <retrieved_value>}` On error: `{"status": "error", "message": "<error_message>"}` The `<retrieved_value>` will be `None` if the key is not found.

Return type
Dict[str, Any]

Raises

- *InvalidServerNameError* – If *server_name* is empty.
- *MissingArgumentError* – If *key* is empty.

async bedrock_server_manager.api.server.**set_server_setting**(*server_name: str*, *key: str*, *value: Any*, *app_context:* AppContext) → Dict[str, Any]

Writes any value to a server's specific JSON configuration file (e.g., `<server_name>_config.json`) using dot-notation for keys. Intermediate dictionaries will be created if they don't exist along the key path.

Parameters

- **server_name** (*str*) – The name of the server.
- **key** (*str*) – The dot-notation key to write to in the server's JSON configuration (e.g., "server_info.status", "custom.new_setting").
- **value** (*Any*) – The new value to write. Must be JSON serializable.

Returns
A dictionary containing the operation result. On success: `{"status": "success", "message": "Setting '<key>' updated..."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, Any]

Raises

- *InvalidServerNameError* – If *server_name* is empty.
- *MissingArgumentError* – If *key* is empty.
- *ConfigParseError* – If *value* is not JSON serializable or if an intermediate part of the *key* path conflicts with an existing non-dictionary item.

async bedrock_server_manager.api.server.**set_server_custom_value**(*server_name: str*, *key: str*, *value: Any*, *app_context:* AppContext) → Dict[str, Any]

Writes a key-value pair to the 'custom' section of a server's specific JSON configuration file (e.g., `<server_name>_config.json`). This is a sandboxed way for plugins or users to store arbitrary data associated with a server. The key will be stored as `custom.<key>`.

Parameters

- **server_name** (*str*) – The name of the server.
- **key** (*str*) – The key (string) for the custom value within the 'custom' section. Cannot be empty.
- **value** (*Any*) – The value to write. Must be JSON serializable.

Returns
A dictionary containing the operation result. On success: `{"status": "success", "message": "Custom value '<key>' updated..."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, Any]

Raises

- ***InvalidServerNameError*** – If *server_name* is empty.
- ***MissingArgumentError*** – If *key* is empty.
- ***ConfigParseError*** – If *value* is not JSON serializable.

async bedrock_server_manager.api.server.**get_all_server_settings**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Reads the entire JSON configuration for a specific server from its dedicated configuration file (e.g., `<server_name>_config.json`). If the file doesn't exist, it will be created with default values. Handles schema migration if an older config format is detected.

Parameters

server_name (*str*) – The name of the server.

Returns

A dictionary containing the operation result. On success: `{"status": "success", **<all_settings_dict>}` On error: `{"status": "error", "message": "<error_message>"}`

Return type

Dict[str, Any]

Raises

- ***InvalidServerNameError*** – If *server_name* is empty.
- ***FileOperationError*** – If creating/reading the config directory/file fails.

async bedrock_server_manager.api.server.**get_server_summary**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves the summary information for a specific server.

This endpoint gets the server summary using the lightweight get_summary_info method, providing a snapshot of the server's basic details like status and player count.

Parameters

- **server_name** (*str*) – The name of the server to get the summary for.
- **app_context** (AppContext) – The application context.

Returns

On success, `{"status": "success", "summary": {...}}`. On error, `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, Any]

async bedrock_server_manager.api.server.**start_server**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Starts the specified Bedrock server.

async bedrock_server_manager.api.server.**stop_server**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Stops the specified Bedrock server.

Triggers the `before_server_stop` and `after_server_stop` plugin events. The method used for stopping `stop()`, which involves gracefully shutdown, with a forceful fallback.

Parameters
server_name (*str*) – The name of the server to stop.

Returns

A dictionary containing the operation result.

On success: `{"status": "success", "message": "Server... stopped successfully."}` or
`{"status": "success", "message": "Server... stop initiated via <service_manager>."}`

On error (e.g., already stopped): `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***ServerStopError*** – If the server fails to stop after all attempts.
- ***BSMError*** – For other application-specific errors during shutdown.

async bedrock_server_manager.api.server.**restart_server**(*server_name: str*, *app_context:* AppContext, *send_message: bool = True*) → Dict[str, Any]

Restarts the specified Bedrock server by orchestrating stop and start.

This function internally calls *stop_server()* and then *start_server()*.

- If the server is already stopped, this function will attempt to start it.
- If running, it will attempt to stop it (optionally sending a restart message to the server if `send_message=True`), wait briefly for the stop to complete, and then start it again.

Parameters

- **server_name** (*str*) – The name of the server to restart.
- **send_message** (*bool, optional*) – If `True`, attempts to send a "say Restarting server..." message to the server console via `send_command()` before stopping. Defaults to `True`.

Returns
A dictionary with the operation status and a message, reflecting the outcome of the start/stop operations. On success: `{"status": "success", "message": "Server... restarted successfully."}` On error: `{"status": "error", "message": "Restart failed: <reason>"}`

Return type
Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***ServerStartError*** – If the start phase fails (from *start_server()*).
- ***ServerStopError*** – If the stop phase fails (from *stop_server()*).
- ***BSMError*** – For other application-specific errors.

async bedrock_server_manager.api.server.**send_command**(*server_name: str*, *command: str*, *app_context:* AppContext) → Dict[str, str]

Sends a command to a running Bedrock server.

The command is checked against a blacklist (defined by `API_COMMAND_BLACKLIST`) before being sent via `send_command()`. Triggers `before_command_send` and `after_command_send` plugin events.

Parameters

- **server_name** (*str*) – The name of the server to send the command to.
- **command** (*str*) – The command string to send (e.g., "list", "say Hello"). Cannot be empty.

Returns

On successful command submission, returns a dictionary: `{"status": "success", "message": "Command '<command>' sent successfully."}`. If an error occurs, an exception is raised instead of returning an error dictionary.

Return type

Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***MissingArgumentError*** – If *command* is empty.
- ***BlockedCommandError*** – If the command is in the API blacklist.
- ***ServerNotRunningError*** – If the target server is not running.
- ***SendCommandError*** – For underlying issues during command transmission (e.g., pipe errors).
- ***ServerError*** – For other unexpected errors during the operation.

async bedrock_server_manager.api.server.**delete_server_data**(*server_name: str*, *app_context:* AppContext, *stop_if_running: bool = True*) → Dict[str, str]

Deletes all data associated with a Bedrock server.

Danger

This is a **HIGHLY DESTRUCTIVE** and irreversible operation.

It calls `delete_all_data()`, which removes: - The server's main installation directory. - The server's JSON configuration subdirectory. - The server's entire backup directory. - The server's PID file.

Triggers `before_delete_server_data` and `after_delete_server_data` plugin events.

Parameters

- **server_name** (*str*) – The name of the server to delete.
- **stop_if_running** (*bool, optional*) – If `True` (default), the server will be stopped using *stop_server()* before its data is deleted. If `False` and the server is running, the operation will likely fail due to file locks or other conflicts.

Returns

A dictionary with the operation status and a message. On success: `{"status": "success", "message": "All data for server... deleted successfully."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***ServerStopError*** – If *stop_if_running* is `True` and the server fails to stop.
- ***FileOperationError*** – If deleting one or more essential directories or files fails.
- ***BSMError*** – For other application-specific errors.

bedrock_server_manager.api.server.**server_lifecycle_manager**(*server_name: str*, *stop_before: bool*, *app_context:* AppContext, *start_after: bool = True*, *restart_on_success_only: bool = False*)

A context manager to safely stop and restart a server for an operation.

async bedrock_server_manager.api.server.**set_server_status_api**(*server_name: str*, *status: str*, *app_context:* AppContext) → Dict[str, Any]

Internal API to set server status and trigger events.

async bedrock_server_manager.api.server.**update_server_player_stats_api**(*server_name: str*, *player_count: int*, *players: list*, *app_context:* AppContext) → Dict[str, Any]

Internal API to trigger player stat updates for websockets/plugins.

34.2 Allowlist API Documentation

async bedrock_server_manager.api.allowlist.**add_to_allowlist**(*server_name: str*, *new_players_data: List[Dict[str, Any]]*, *app_context:* AppContext) → Dict[str, Any]

Adds a list of players to the server's allowlist.

Parameters

- **server_name** (*str*) – The name of the server.
- **new_players_data** (*List[Dict[str, Any]]*) – List of player dictionaries (with 'name' and 'xuid').
- **app_context** (AppContext) – The application context.

Returns
Status dictionary containing success or error message.

Return type
Dict[str, Any]

async bedrock_server_manager.api.allowlist.**get_allowlist**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves the current allowlist for a given server.

Parameters

- **server_name** (*str*) – The name of the server.
- **app_context** (AppContext) – The application context.

Returns
A dictionary containing the list of players on the allowlist.

Return type
Dict[str, Any]

async bedrock_server_manager.api.allowlist.**remove_from_allowlist**(*server_name: str*, *player_names: List[str]*, *app_context:* AppContext) → Dict[str, Any]

Removes a list of players from the server's allowlist.

Parameters

- **server_name** (*str*) – The name of the server.
- **player_names** (*List[str]*) – List of player names to remove.
- **app_context** (AppContext) – The application context.

Returns
Status dictionary containing success or error message, and details of removed vs not_found players.

Return type
Dict[str, Any]

34.3 Ban API Documentation

async bedrock_server_manager.api.ban.**add_server_ban_api**(*app_context:* AppContext, *server_name: str*, *player_name: str*, *xuid: str*, *reason: str | None = None*) → Dict[str, Any]

Adds a player to the server ban list.

async bedrock_server_manager.api.ban.**remove_server_ban_api**(*app_context:* AppContext, *server_name: str*, *xuid: str*) → Dict[str, Any]

Removes a player from the server ban list.

async bedrock_server_manager.api.ban.**get_server_bans_api**(*app_context:* AppContext, *server_name: str*) → Dict[str, Any]

Retrieves all bans for a specific server.

34.4 Install API Documentation

async bedrock_server_manager.api.install.**install_new_server**(*server_name: str*, *app_context:* AppContext, *target_version: str = 'LATEST'*, *server_zip_path: str | None = None*) → Dict[str, Any]

Installs a new server instance.

Parameters

- **server_name** (*str*) – The name for the new server.

- **app_context** (AppContext) – The application context.
- **target_version** (*str, optional*) – The target version to install (default: "LATEST").
- **server_zip_path** (*Optional[str], optional*) – Path to a local zip file for installation.

Returns
A dictionary containing the installation status and message.

Return type
Dict[str, Any]

async bedrock_server_manager.api.install.**update_server**(*server_name: str*, *app_context:* AppContext, *send_message: bool = True*) → Dict[str, Any]

Updates an existing server instance.

Parameters

- **server_name** (*str*) – The name of the server to update.
- **app_context** (AppContext) – The application context.
- **send_message** (*bool, optional*) – Whether to send a message to players before updating.

Returns
A dictionary containing the update status, a boolean flag *updated*, and a message.

Return type
Dict[str, Any]

34.5 Permissions API Documentation

async bedrock_server_manager.api.permissions.**set_permissions**(*server_name: str*, *xuid: str*, *player_name: str | None*, *permission: str*, *app_context:* AppContext) → Dict[str, str]

Sets a player's permission level for a given server.

Parameters

- **server_name** (*str*) – The name of the server.
- **xuid** (*str*) – The XUID of the player.
- **player_name** (*Optional[str]*) – The name of the player.
- **permission** (*str*) – The permission level to grant.
- **app_context** (AppContext) – The application context.

Returns
A dictionary with the status and result message.

Return type
Dict[str, str]

async bedrock_server_manager.api.permissions.**get_permissions**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves the permissions configuration for a server, formatted with player names.

Parameters

- **server_name** (*str*) – The name of the server.
- **app_context** (AppContext) – The application context.

Returns
A dictionary containing a list of permission objects.

Return type
Dict[str, Any]

34.6 Properties API Documentation

async bedrock_server_manager.api.properties.**get_properties**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves the current server properties for a given server.

Parameters

- **server_name** (*str*) – The name of the server.
- **app_context** (AppContext) – The application context.

Returns
A dictionary containing the parsed server properties.

Return type
Dict[str, Any]

bedrock_server_manager.api.properties.**validate_property_value**(*property_name: str*, *value: str*) → Dict[str, str]

Validates a specific server property value before it gets applied.

Parameters

- **property_name** (*str*) – The name of the property to validate.
- **value** (*str*) – The proposed value for the property.

Returns
A dictionary indicating success or validation error message.

Return type
Dict[str, str]

async bedrock_server_manager.api.properties.**set_properties**(*server_name: str*, *properties_to_update: Dict[str, str]*, *app_context:* AppContext, *restart_after_modify: bool = False*) → Dict[str, str]

Sets one or multiple server properties for a given server.

Parameters

- **server_name** (*str*) – The name of the server.
- **properties_to_update** (*Dict[str, str]*) – A dictionary of property keys and new values.
- **app_context** (AppContext) – The application context.
- **restart_after_modify** (*bool, optional*) – Whether to restart the server if running. Defaults to False.

Returns
A dictionary with the status and result message.

Return type
Dict[str, str]

34.7 Settings API Documentation

Provides an API for interacting with global application settings.

This module offers functions to read, write, and reload application-wide configuration values. These settings are managed by the *Settings* class and are typically stored in the main `bedrock_server_manager.json` configuration file.

The functions provided here allow other parts of the application, including plugins (via methods exposed by `api_method()`), to programmatically access and modify these global settings.

async bedrock_server_manager.api.settings.**get_global_setting**(*key: str*, *app_context:* AppContext) → Dict[str, Any]

Reads a single value from the global application settings.

This function uses `get()` to retrieve the value associated with the given *key*.

Parameters
key (*str*) – The dot-notation key for the setting (e.g., "paths.backups", "web.port").

Returns
A dictionary with the operation result. On success: `{"status": "success", "value": <retrieved_value>}`. The `<retrieved_value>` will be `None` if the key does not exist in the settings. On error (unexpected): `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises
MissingArgumentError – If *key* is empty.

async bedrock_server_manager.api.settings.**get_all_global_settings**(*app_context:* AppContext) → Dict[str, Any]

Reads the entire global application settings configuration.

Returns a copy of all currently loaded settings from the *Settings* instance.

Returns
A dictionary with the operation result. On success: `{"status": "success", **<all_settings_dict>}`. On error (unexpected): `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

async bedrock_server_manager.api.settings.**set_global_setting**(*key: str*, *value: Any*, *app_context:* AppContext) → Dict[str, Any]

Writes a value to the global application settings.

This function uses `set()` to update the value associated with the given *key* and persists the changes to the main application configuration file (e.g., `bedrock_server_manager.json`).

Parameters

- **key** (*str*) – The dot-notation key for the setting to update (e.g., "web.port", "paths.backups").
- **value** (*Any*) – The new value to set. This value must be JSON-serializable.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Global setting '<key>' updated successfully."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, Any]

Raises

- ***MissingArgumentError*** – If *key* is empty.
- ***BSMError*** – Can be raised by `set()` for issues like write errors (e.g., `ConfigWriteError`) or if the value is not JSON serializable.

async bedrock_server_manager.api.settings.**set_custom_global_setting**(*key: str*, *value: Any*, *app_context:* AppContext) → Dict[str, Any]

Writes a custom value to the global application settings.

This function uses `set()` to update the value associated with the given *key* and persists the changes to the main application configuration file (e.g., `bedrock_server_manager.json`).

Parameters

- **key** (*str*) – The key for the setting to update (e.g., "custom_dir", "somekey"). This will be prefixed with "custom.".
- **value** (*Any*) – The new value to set. This value must be JSON-serializable.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Global setting '<key>' updated successfully."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, Any]

Raises

- ***MissingArgumentError*** – If *key* is empty.
- ***BSMError*** – Can be raised by `set()` for issues like write errors (e.g., `ConfigWriteError`) or if the value is not JSON serializable.

async bedrock_server_manager.api.settings.**reload_global_settings**(*app_context:* AppContext) → Dict[str, str]

Forces a reload of settings and logging config from the file.

This is useful if `bedrock_server_manager.json` has been edited manually and the application needs to pick up the changes without restarting. It calls `reload()` to re-read the configuration file, and then calls `setup_logging()` to re-apply the logging configuration based on the (potentially) new settings.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Global settings and logging configuration have been reloaded."}` On error: `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, str]

Raises
BSMError – Can be raised by `reload()` (e.g., `ConfigLoadError`) or by `setup_logging()` if critical logging settings are missing or invalid after reload.

34.8 Backup & Restore API Documentation

Provides API functions for server backup, restore, and pruning operations.

This module offers a high-level interface for managing the backup and restoration of Bedrock server data. It orchestrates calls to methods of the *BedrockServer* class, primarily those provided by the `ServerBackupMixin`.

Key functionalities include:

- Listing available backup files (*list_backup_files()*).
- Backing up individual components like the server world (*backup_world()*) or specific configuration files (*backup_config_file()*).
- Performing a comprehensive backup of all standard server data (*backup_all()*).
- Restoring all server data from the latest available backups (*restore_all()*).
- Restoring the server world from a specific `.mcworld` file (*restore_world()*).
- Restoring a specific configuration file from its backup (*restore_config_file()*).
- Pruning old backups based on retention policies (*prune_old_backups()*).

Operations involving file modifications are thread-safe using a unified lock (`_backup_restore_lock`). For actions requiring the server to be offline, this module utilizes the *server_lifecycle_manager()* to safely stop and restart the server. All functions are exposed to the plugin system.

async bedrock_server_manager.api.backup_restore.**list_backup_files**(*server_name: str*, *backup_type: str*, *app_context:* AppContext) → Dict[str, Any]

Lists available backup files for a given server and type.

This is a read-only operation and does not require a lock. It calls `list_backups()`.

Parameters

- **server_name** (*str*) – The name of the server.
- **backup_type** (*str*) – The type of backups to list. Valid options are “world”, “properties”, “allowlist”, “permissions”, or “all” (case-insensitive).

Returns
A dictionary with the operation result. On success: `{"status": "success", "backups": BackupData}`. If *backup_type* is specific (e.g., “world”), *BackupData* is `List[str]` of backup file paths. If *backup_type* is “all”, *BackupData* is `Dict[str, List[str]]` categorizing backups (e.g., `{"world_backups": [...], "properties_backups": [...]}`). An empty list/dict is returned if no backups are found or backup dir is missing. On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises

- *InvalidServerNameError* – If the server name is empty.
- *MissingArgumentError* – If *backup_type* is empty.
- *UserInputError* – If *backup_type* is invalid.
- *ConfigurationError* – If the server's backup directory is not configured.
- *FileOperationError* – For OS errors during file listing.

async bedrock_server_manager.api.backup_restore.**backup_world**(*server_name: str*, *app_context:* AppContext) → Dict[str, str]

Creates a backup of the server's world directory.

This operation is thread-safe and guarded by a lock. It calls the internal `_backup_world_data_internal` method of the *BedrockServer* instance, which handles determining the active world, exporting it to a `.mcworld` file (performing a live backup using save hold if the server is running), and pruning old world backups. Triggers `before_backup` and `after_backup` plugin events (with type "world").

Parameters

server_name (*str*) – The name of the server whose world is to be backed up.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "World backup '<filename>' created..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, str]

Raises

- *MissingArgumentError* – If *server_name* is empty.
- *BSMError* – Propagates errors from underlying operations, including: *ConfigurationError* (backup path not set), *AppFileNotFoundError* (world dir missing), or *BackupRestoreError* (export/pruning issues).

async bedrock_server_manager.api.backup_restore.**backup_config_file**(*server_name: str*, *file_to_backup: str*, *app_context:* AppContext) → Dict[str, str]

Creates a backup of a specific server configuration file.

This operation is thread-safe and guarded by a lock. It calls the internal `_backup_config_file_internal` method of the *BedrockServer* instance. This core method copies the specified file (e.g., `server.properties`) from the server's installation directory to a timestamped backup in the server's backup directory, then prunes older backups of that file type. Triggers `before_backup` and `after_backup` plugin events (with type "config_file").

Parameters

- **server_name** (*str*) – The name of the server.
- **file_to_backup** (*str*) – The name of the configuration file to back up (e.g., "server.properties", "allowlist.json"). This file is expected to be in the root of the server's installation directory.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "Config file '<name>' backed up as '<backup_name>'..."}` If original file not

found: {"status": "error", "message": "Config file backup failed: File ... not found."} (or similar from BSMError) On other error: {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, str]

Raises

- ***MissingArgumentError*** – If *server_name* or *file_to_backup* is empty.
- ***BSMError*** – Propagates errors from underlying operations, including *ConfigurationError* (backup path not set) or *FileOperationError* (file copy/pruning issues).

async bedrock_server_manager.api.backup_restore.**backup_all**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Performs a full backup of the server's world and configuration files.

This operation is thread-safe and guarded by a lock. It calls `backup_all_data()`. Triggers `before_backup` and `after_backup` plugin events (with type "all").

Parameters
server_name (*str*) – The name of the server to back up.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: {"status": "success", "message": "Full backup completed...", "details": BackupResultsDict} where `BackupResultsDict` maps component names (e.g., "world", "allowlist.json") to the path of their backup file, or `None` if a component's backup failed. On error (e.g., critical world backup failure): {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, Any]

Raises

- ***MissingArgumentError*** – If *server_name* is empty.
- ***BSMError*** – Propagates errors from underlying operations, including: *ConfigurationError* (backup path not set) or *BackupRestoreError* (if critical world backup fails).

async bedrock_server_manager.api.backup_restore.**restore_all**(*server_name: str*, *app_context:* AppContext, *stop_start_server: bool = True*) → Dict[str, Any]

Restores the server from the latest available backups.

This operation is thread-safe and guarded by a lock. It calls `restore_all_data_from_latest()`. If *stop_start_server* is `True`, the *server_lifecycle_manager()* is used to manage the server's state, restarting it only if the restore operation (all components) is successful.

> **Warning**
>
> This operation **OVERWRITES** current world data and configuration files in the server's installation directory with content from the latest backups.

Triggers `before_restore` and `after_restore` plugin events (with type "all").

Parameters

- **server_name** (*str*) – The name of the server to restore.
- **stop_start_server** (*bool, optional*) – If `True`, the server will be stopped before restoring and restarted afterwards only if the entire restore operation succeeds. Defaults to `True`.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "Restore_all completed...", "details": RestoreResultsDict}` where `RestoreResultsDict` maps component names to their restored paths or `None` on failure/skip. If no backups found: `{"status": "success", "message": "No backups found..."}` On error: `{"status": "error", "message": "<error_message>"}` (e.g., if a component failed to restore).

Return type

Dict[str, Any]

Raises

- ***MissingArgumentError*** – If *server_name* is empty.
- ***BSMError*** – Propagates errors from underlying operations, including: *ConfigurationError* (backup path not set), *BackupRestoreError* (if any component fails to restore), or errors from server stop/start.

`async` bedrock_server_manager.api.backup_restore.**restore_world**(*server_name: str*, *backup_file_path: str*, *app_context:* AppContext, *stop_start_server: bool = True*) → Dict[str, str]

Restores a server's world from a specific backup file.

This operation is thread-safe and guarded by a lock. If *stop_start_server* is `True`, it uses the *server_lifecycle_manager()* to manage the server's state, restarting it only if the restore is successful. The core world import is performed by `import_world()`.

> **Warning**
>
> This is a **DESTRUCTIVE** operation. The existing active world directory will be deleted before the new world is imported from the backup.

Triggers `before_restore` and `after_restore` plugin events (with type "world").

Parameters

- **server_name** (*str*) – The name of the server.
- **backup_file_path** (*str*) – The absolute path to the `.mcworld` backup file to be restored.
- **stop_start_server** (*bool, optional*) – If `True`, the server will be stopped before restoring and restarted afterwards only if the restore is successful. Defaults to `True`.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "World restore from '<filename>' completed..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

- ***MissingArgumentError*** – If *server_name* or *backup_file_path* is empty.
- ***AppFileNotFoundError*** – If *backup_file_path* does not exist.
- ***BSMError*** – Propagates errors from underlying operations like *BackupRestoreError*, *ExtractError*, or errors from server stop/start.

async bedrock_server_manager.api.backup_restore.**restore_config_file**(*server_name: str*, *backup_file_path: str*, *app_context:* AppContext, *stop_start_server: bool = True*) → Dict[str, str]

Restores a specific config file from a backup.

This operation is thread-safe and guarded by a lock. If *stop_start_server* is `True`, it uses the *server_lifecycle_manager()* to manage the server's state, restarting it only if the restore is successful. The core config file restoration is performed by the internal `_restore_config_file_internal` method of the *BedrockServer* instance.

> **Warning**
>
> This operation **OVERWRITES** the current version of the configuration file in the server's installation directory with the content from the backup.

Triggers `before_restore` and `after_restore` plugin events (with type "config_file").

Parameters

- **server_name** (*str*) – The name of the server.
- **backup_file_path** (*str*) – The absolute path to the configuration backup file (e.g., `.../server_backup_YYYYMMDD_HHMMSS.properties`).
- **stop_start_server** (*bool, optional*) – If `True`, the server will be stopped before restoring and restarted afterwards only if the restore is successful. Defaults to `True`.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "Config file '<original_name>' restored from '<backup_name>'..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

- ***MissingArgumentError*** – If *server_name* or *backup_file_path* is empty.
- ***AppFileNotFoundError*** – If *backup_file_path* does not exist.
- ***UserInputError*** – If the backup filename format is unrecognized.
- ***BSMError*** – Propagates errors from underlying operations like *FileOperationError* or errors from server stop/start.

async bedrock_server_manager.api.backup_restore.**prune_old_backups**(*server_name: str*, *app_context:* AppContext) → Dict[str, str]

Prunes old backups for a server based on retention settings.

This operation is thread-safe and guarded by a lock. It iteratively calls `prune_server_backups()` for the server's world (`.mcworld` files) and standard configuration files (`server.properties`, `allowlist.json`, `permissions.json`). The number of backups to keep is determined by the `retention.backups` application setting. Triggers `before_prune_backups` and `after_prune_backups` plugin events.

Parameters
server_name (*str*) – The name of the server whose backups are to be pruned.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On full success: `{"status": "success", "message": "Backup pruning completed..."}` If some components fail pruning: `{"status": "error", "message": "Pruning completed with errors: <details>"}` If backup directory not found: `{"status": "success", "message": "No backup directory found..."}` On other setup error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises
- ***MissingArgumentError*** – If *server_name* is empty.
- ***BSMError*** – Propagates errors from underlying operations, particularly *ConfigurationError* if backup path is not set, or *UserInputError* if retention settings are invalid. Individual *FileOperationError* for components are typically aggregated into the error message.

34.9 World API Documentation

async bedrock_server_manager.api.world.**get_world_name**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves the configured world name (*level-name*) for a server.

This function reads the *server.properties* file to get the name of the directory where the world data is stored, by calling `get_world_name()`.

Parameters
server_name (*str*) – The name of the server to query.

Returns
A dictionary with the operation result. On success: `{"status": "success", "world_name": "<world_name_str>"}`. On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises
- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***BSMError*** – Can be raised by *BedrockServer* instantiation or by the underlying *get_world_name* method (e.g., *AppFileNotFoundError* if `server.properties` is missing, or *ConfigParseError* if `level-name` is missing or malformed).

async bedrock_server_manager.api.world.**export_world**(*server_name: str*, *app_context:* AppContext, *export_dir: str | None = None*) → Dict[str, Any]

Exports the server's currently active world to a .mcworld archive.

This operation is thread-safe due to `_world_lock`. The core world export is performed by `export_world()`, which performs a live backup using save hold if the server is running. Triggers `before_world_export` and `after_world_export` plugin events.

Parameters

- **server_name** (*str*) – The name of the server whose world is to be exported.
- **export_dir** (*Optional[str], optional*) – The directory to save the exported `.mcworld` file. If `None`, it defaults to a "worlds" subdirectory within the application's global content directory (defined by `paths.content` setting). Defaults to `None`.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "export_file": "<path_to_mcworld>", "message": "World '<name>' exported..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, Any]

Raises

- ***InvalidServerNameError*** – If *server_name* is empty.
- ***FileOperationError*** – If the content directory setting (`paths.content`) is missing when *export_dir* is `None`, or for other file I/O errors during export.
- ***BSMError*** – Propagates errors from underlying operations, including *AppFileNotFoundError* if world directory is missing, or *BackupRestoreError* from export.

async bedrock_server_manager.api.world.**import_world**(*server_name: str*, *selected_file_path: str*, *app_context:* AppContext, *stop_start_server: bool = True*) → Dict[str, str]

Imports a world from a .mcworld file, replacing the active world.

This is a destructive operation that replaces the current world. It is thread-safe and manages the server lifecycle to ensure data integrity.

> **Warning**
>
> This is a **DESTRUCTIVE** operation. The existing active world directory will be deleted before the new world is imported.

If *stop_start_server* is `True`, this function uses the *server_lifecycle_manager()* to ensure the server is stopped during the import. The core world import is performed by `import_world()`. Triggers `before_world_import` and `after_world_import` plugin events.

Parameters

- **server_name** (*str*) – The name of the server to import the world into.
- **selected_file_path** (*str*) – The absolute path to the `.mcworld` file to import.

- **stop_start_server** (*bool, optional*) – If `True`, the server will be stopped before the import and restarted afterwards. Defaults to `True`.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "World '<name>' imported..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is empty.
- ***MissingArgumentError*** – If *selected_file_path* is empty.
- **FileNotFoundError** – If *selected_file_path* does not exist (from implementation).
- ***BSMError*** – Propagates errors from underlying operations, including *BackupRestoreError* from import, *ExtractError*, or errors from server stop/start.

async bedrock_server_manager.api.world.**reset_world**(*server_name: str*, *app_context:* AppContext) → Dict[str, str]

Resets the server's world by deleting the active world directory.

This is a destructive action. Upon next start, the server will generate a new world based on its *server.properties* configuration. This function is thread-safe and manages the server lifecycle.

> **Warning**
>
> This is a **DESTRUCTIVE** operation. The existing active world directory will be permanently removed.

This function uses the *server_lifecycle_manager()* to ensure the server is stopped before deleting the world and restarted afterwards (which will trigger new world generation). The active world directory is deleted using `delete_world()`. Triggers `before_world_reset` and `after_world_reset` plugin events.

Parameters
server_name (*str*) – The name of the server whose world is to be reset.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "World '<name>' reset successfully."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

- ***InvalidServerNameError*** – If *server_name* is empty.
- ***BSMError*** – Propagates errors from underlying operations, including *FileOperationError* from deletion, errors determining the world name, or errors from server stop/start.

34.10 Addon API Documentation

async bedrock_server_manager.api.addon.**list_available_addons**(*app_context:* AppContext) → Dict[str, Any]

Lists available .mcaddon and .mcpack files from the content directory.

Scans the addons sub-folder within the application's global content directory.

Returns
A dictionary with the operation result. On success: `{"status": "success", "files": List[str]}` where *files* is a list of absolute paths to `.mcaddon` or `.mcpack` files. On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises
FileError – If the content directory is not configured or accessible.

async bedrock_server_manager.api.addon.**import_addon**(*server_name: str*, *addon_file_path: str*, *app_context:* AppContext, *stop_start_server: bool = True*, *restart_only_on_success: bool = True*) → Dict[str, str]

Installs an addon to a specified Bedrock server.

This function handles the import and installation of an addon file (.mcaddon or .mcpack) into the server's addon directories. It is thread-safe, using a lock to prevent concurrent addon operations which could lead to corrupted files. It calls `process_addon_file()` for the core processing logic.

The function can optionally manage the server's lifecycle by stopping it before the installation and restarting it after, using the *server_lifecycle_manager()*. Triggers `before_addon_import` and `after_addon_import` plugin events.

Parameters

- **server_name** (*str*) – The name of the server to install the addon on.
- **addon_file_path** (*str*) – The absolute path to the addon file (`.mcaddon` or `.mcpack`).
- **stop_start_server** (*bool, optional*) – If `True`, the server will be stopped before installation and started afterward. Defaults to `True`.
- **restart_only_on_success** (*bool, optional*) – If `True` and *stop_start_server* is `True`, the server will only be restarted if the addon installation succeeds. Defaults to `True`.

Returns
A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). On success: `{"status": "success", "message": "Addon '<filename>' installed..."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

- ***MissingArgumentError*** – If *server_name* or *addon_file_path* is not provided.
- ***AppFileNotFoundError*** – If the file at *addon_file_path* does not exist.
- ***InvalidServerNameError*** – If the server name is not valid (raised from BedrockServer).

- ***BSMError*** – Propagates errors from underlying operations, including *UserInputError* (unsupported addon type), *ExtractError*, *FileOperationError*, or errors from server stop/start.

async bedrock_server_manager.api.addon.**list_installed_addons**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Lists all addons for a server's active world.

Parameters

- **server_name** (*str*) – The name of the server.
- **app_context** (AppContext) – The application context.

Returns
A dictionary containing the addon lists.

Return type
Dict[str, Any]

async bedrock_server_manager.api.addon.**enable_addon**(*server_name: str*, *pack_uuid: str*, *pack_type: str*, *app_context:* AppContext) → Dict[str, str]

Enables a disabled addon for a server's active world.

Parameters

- **server_name** (*str*) – The name of the server.
- **pack_uuid** (*str*) – The UUID of the pack to enable.
- **pack_type** (*str*) – The type of the pack.
- **app_context** (AppContext) – The application context.

Returns
Status of the operation.

Return type
Dict[str, str]

async bedrock_server_manager.api.addon.**disable_addon**(*server_name: str*, *pack_uuid: str*, *pack_type: str*, *app_context:* AppContext) → Dict[str, str]

Disables an active addon for a server's active world, preserving files.

Parameters

- **server_name** (*str*) – The name of the server.
- **pack_uuid** (*str*) – The UUID of the pack to disable.
- **pack_type** (*str*) – The type of the pack.
- **app_context** (AppContext) – The application context.

Returns
Status of the operation.

Return type
Dict[str, str]

async bedrock_server_manager.api.addon.**update_subpack**(*server_name: str*, *pack_uuid: str*, *pack_type: str*, *subpack_name: str*, *app_context:* AppContext) → Dict[str, str]

Updates the active subpack for an addon on a server's active world.

Parameters

- **server_name** (*str*) – The name of the server.
- **pack_uuid** (*str*) – The UUID of the pack.
- **pack_type** (*str*) – The type of the pack.
- **subpack_name** (*str*) – The folder name of the target subpack.
- **app_context** (AppContext) – The application context.

Returns
Status of the operation.

Return type
Dict[str, str]

async bedrock_server_manager.api.addon.**uninstall_addon**(*server_name: str*, *pack_uuid: str*, *pack_type: str*, *app_context:* AppContext) → Dict[str, str]

Uninstalls an addon for a server's active world, deleting its files.

Parameters

- **server_name** (*str*) – The name of the server.
- **pack_uuid** (*str*) – The UUID of the pack to uninstall.
- **pack_type** (*str*) – The type of the pack.
- **app_context** (AppContext) – The application context.

Returns
Status of the operation.

Return type
Dict[str, str]

async bedrock_server_manager.api.addon.**reorder_addons**(*server_name: str*, *uuids: list[str]*, *pack_type: str*, *app_context:* AppContext) → Dict[str, str]

Reorders the active addons for a server's active world.

Parameters

- **server_name** (*str*) – The name of the server.
- **uuids** (*list[str]*) – The exact list of active UUIDs in their new order.
- **pack_type** (*str*) – The type of the pack.
- **app_context** (AppContext) – The application context.

Returns
Status of the operation.

Return type
Dict[str, str]

34.11 Plugins API Documentation

Provides API functions for interacting with the application's plugin system.

This module serves as an interface to the global plugin manager instance, which is an object of *PluginManager*. It allows for retrieving plugin statuses, enabling or disabling plugins, reloading the plugin system, and triggering custom plugin events from external sources.

Key functionalities include: - Getting statuses and metadata of all discovered plugins (*get_plugin_statuses()*). - Setting the enabled/disabled state of a specific plugin (*set_plugin_status()*). - Reloading all plugins (*reload_plugins()*). - Triggering custom plugin events externally (*trigger_external_app_event_api()*).

These functions facilitate management and interaction with plugins, primarily for use by administrative interfaces like a web UI or CLI.

async bedrock_server_manager.api.plugins.**get_plugin_statuses**(*app_context:* AppContext) → Dict[str, Any]

Retrieves the statuses and metadata of all discovered plugins.

This function first ensures the plugin configuration is synchronized with the plugin files on disk by calling an internal method of the *PluginManager*. It then returns a copy of the current plugin configuration.

Parameters
plugin_manager (PluginManager) – The plugin manager instance.

Returns
A dictionary with the operation result. On success: `{"status": "success", "plugins": PluginConfigDict}` where `PluginConfigDict` is a dictionary mapping plugin names (str) to their configuration (another dict containing keys like "enabled" (bool), "description" (str), "version" (str)). On error (unexpected): `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

async bedrock_server_manager.api.plugins.**set_plugin_status**(*plugin_name: str*, *enabled: bool*, *app_context:* AppContext) → Dict[str, Any]

Sets the enabled/disabled status for a specific plugin.

This function updates the `enabled` field for the given *plugin_name* in the plugin configuration, which is managed by the *PluginManager*. The configuration is synchronized with disk before modification, and the changes are saved back to the database.

> **Note**
> For the change in enabled status to take full effect (i.e., for the plugin to be loaded or unloaded), *reload_plugins()* typically needs to be called afterwards.

Parameters

- **plugin_manager** (PluginManager) – The plugin manager instance.
- **plugin_name** (*str*) – The name of the plugin to configure.
- **enabled** (*bool*) – `True` to enable the plugin, `False` to disable it.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message":`

"Plugin '<name>' has been <enabled/disabled>..."} On error: {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, Any]

Raises
UserInputError – If *plugin_name* is empty or not found in the configuration.

async bedrock_server_manager.api.plugins.**reload_single_plugin**(*plugin_name: str*, *app_context:* AppContext) → Dict[str, Any]

Reloads a single plugin by name.

async bedrock_server_manager.api.plugins.**reload_plugins**(*app_context:* AppContext) → Dict[str, Any]

Triggers the plugin manager to unload all active plugins and then reload all plugins based on the current configuration.

This function calls the *reload* method of the *PluginManager* instance.

Parameters
plugin_manager (PluginManager) – The plugin manager instance.

Returns
A dictionary with the operation result. On success: {"status": "success", "message": "Plugins have been reloaded successfully."} On error (unexpected): {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, Any]

async bedrock_server_manager.api.plugins.**trigger_external_app_event_api**(*event_name: str*, *app_context:* AppContext, *payload: Dict[str, Any] | None = None*) → Dict[str, Any]

Allows an external source (like a web route or CLI) to trigger a custom plugin event.

This function calls the *trigger_event* method of the *PluginManager* instance. The *triggering_plugin_name* argument for the core method is set to "external_api_trigger" to identify the source of this event.

Parameters

- **plugin_manager** (PluginManager) – The plugin manager instance.
- **event_name** (*str*) – The name of the custom event to trigger. Must follow the 'namespace:event_name' format for custom events.
- **payload** (*Optional[Dict[str, Any]], optional*) – A dictionary of data to pass as keyword arguments to the event listeners' callback functions. Defaults to None (an empty dictionary will be passed).

Returns
A dictionary with the operation result. On success: {"status": "success", "message": "Event '<event_name>' triggered."} On error (e.g., invalid event name format, unexpected error during dispatch): {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, Any]

Raises

UserInputError – If *event_name* is empty or does not follow the 'namespace:event_name' format required by the plugin manager.

34.12 Application API Documentation

Provides API functions for application-wide information and actions.

This module offers endpoints to retrieve general details about the Bedrock Server Manager application itself and to perform operations that span across multiple server instances.

Key functionalities include:

- Retrieving application metadata (name, version, OS, key directories) via `get_application_info_api()`.
- Listing globally available content like world templates (*list_available_worlds_api()*) and addons (`list_available_addons_api()`).
- Aggregating status and version information for all detected server instances using *get_all_servers_data()*.

These functions are exposed to the plugin system via `api_method()` and are intended for use by UIs, CLIs, or other high-level components.

async bedrock_server_manager.api.application.**list_available_worlds_api**(*app_context:* AppContext) → Dict[str, Any]

Lists available .mcworld files from the content directory.

Scans the `worlds` sub-folder within the application's global content directory.

Returns

A dictionary with the operation result. On success: `{"status": "success", "files": List[str]}` where *files* is a list of absolute paths to `.mcworld` files. On error: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, Any]

Raises

FileError – If the content directory is not configured or accessible.

async bedrock_server_manager.api.application.**get_all_servers_data**(*app_context:* AppContext) → Dict[str, Any]

Retrieves status and version for all detected servers.

This function acts as an API orchestrator to gather data from all individual server instances. It can handle partial failures, where data for some servers is retrieved successfully while others fail (errors for individual servers are included in the message). The status of each server is also reconciled with its live state during this call.

Returns

A dictionary with the operation result.

On full success (all servers processed without error):
`{"status": "success", "servers": List[ServerDataDict]}`

On partial success (some individual server errors occurred during scan):
`{"status": "success", "servers": List[ServerDataDict], "message":`

"Completed with errors: <details>"} The servers list contains data for successfully processed servers.

On total failure (e.g., cannot access base server directory):
{"status": "error", "message": "<error_message>"}

Each ServerDataDict contains keys like "name", "status", "version".

Return type
Dict[str, Any]

Raises
BSMError – If there's a fundamental issue accessing the base server directory (e.g., *AppFileNotFoundError*).

bedrock_server_manager.api.application.**get_system_and_app_info**(*app_context:* AppContext) → Dict[str, Any]

Retrieves basic system and application information.

Uses global settings to get OS type and app version.

Returns
On success: {"status": "success", "os_type": "...", "app_version": "...", "splash_text": "..."}. On error: {"status": "error", "message": "An unexpected error occurred."}

Return type
Dict[str, Any]

async bedrock_server_manager.api.application.**run_task**(*target_function: Callable*, *app_context:* AppContext, *username: str | None = None*, **args: Any*, ***kwargs: Any*) → str

Submits a function to be run in the background by the TaskManager.

Parameters

- **target_function** (*Callable*) – The function to execute.
- **app_context** (AppContext) – The application context.
- **username** (*Optional[str], optional*) – The user associated with the task for WebSocket notifications.
- ***args** (*Any*) – Positional arguments for the target function.
- ****kwargs** (*Any*) – Keyword arguments for the target function.

Returns
The ID of the created task.

Return type
str

async bedrock_server_manager.api.application.**update_server_statuses**(*app_context:* AppContext) → Dict[str, Any]

Reconciles the status in config files with the runtime state for all servers.

This function calls the core function to get servers data. During that call, for each discovered server, its get_status() method is invoked. This method determines the actual running state of the server process and updates the status field in the server's JSON configuration file (e.g., <server_name>_config.json) if there's a discrepancy between the stored status and the live status.

Returns
A dictionary summarizing the operation. On success (even with individual server errors during discovery): `{"status": "success", "message": "Status check completed for <n> servers."}` or `{"status": "error", "message": "Completed with errors: <details>", "updated_servers_count": <n>}` (The "error" status here primarily reflects issues during the overall scan, like directory access problems, rather than individual server status update failures, which are logged and included in the message if *discovery_errors* occur.)

Return type
Dict[str, Any]

34.13 Player API Documentation

Provides API functions for managing the central player database.

This module offers an interface to interact with the application's central player database, typically stored in the database. It leverages the application context to perform operations such as:

- Manually adding or updating player entries (gamertag and XUID) via *add_players_manually_api()*.
- Retrieving all known player entries from the database using *get_all_known_players_api()*.
- Discovering players by scanning server logs and updating the database via *scan_and_update_player_db_api()*.

These functions are exposed to the plugin system and provide a structured way to manage player data globally across all server instances.

async bedrock_server_manager.api.player.**add_players_manually_api**(*player_strings: List[str]*, *app_context:* AppContext) → Dict[str, Any]

Adds or updates player data in the database.

This function takes a list of strings, each containing a player's gamertag and XUID, parses them, and saves the data to the player database. Triggers `before_players_add` and `after_players_add` plugin events.

Parameters
player_strings (*List[str]*) – A list of strings. Each string should represent a single player in the format "gamertag:xuid" (e.g., `"PlayerOne:1234567890123456"`). Example list: `["PlayerOne:123...", "PlayerTwo:654..."]`.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "<n> player entries processed...", "count": <n>}` On error (parsing or saving): `{"status": "error", "message": "<error_message>"}`

Return type
Dict[str, Any]

Raises
- ***UserInputError*** – If any player string in *player_strings* is malformed (propagated from `parse_player_cli_argument`).
- ***BSMError*** – If saving to the database fails.

async bedrock_server_manager.api.player.**get_all_known_players_api**(*app_context:* AppContext) → Dict[str, Any]

Retrieves all player data from the database.

Returns
A dictionary with the operation result. On success: `{"status": "success", "players": List[PlayerDict]}` where each `PlayerDict` typically contains "name" and "xuid". Returns an empty list for *players* if the database is empty. On unexpected error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

async bedrock_server_manager.api.player.**scan_and_update_player_db_api**(*app_context:* AppContext) → Dict[str, Any]

Scans all server logs to discover and save player data.

This function iterates through the log files of all managed servers, extracts player connection information (gamertag and XUID), and updates the central player database with any new findings. Triggers `before_player_db_scan` and `after_player_db_scan` plugin events.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "<summary_message>", "details": ScanResultDict}` where `ScanResultDict` contains keys like: `"total_entries_in_logs"` (int), `"unique_players_submitted_for_saving"` (int), `"actually_saved_or_updated_in_db"` (int), `"scan_errors"` (List[Dict[str, str]]). On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises
BSMError – Can be raised by the underlying manager method, e.g., *AppFileNotFoundError* if the main server base directory is misconfigured, or *FileOperationError* if the final save to the database fails. Individual server scan errors are reported within the "details" part of a successful response.

34.14 System API Documentation

Provides API functions for system-level server interactions and information.

This module serves as an interface for querying system-related information about server processes and for managing their integration with the host operating system's service management capabilities. It primarily orchestrates calls to the *BedrockServer* class.

Key functionalities include:

- Querying server process resource usage (e.g., PID, CPU, memory) via *get_bedrock_process_info()*.
- Managing OS-level services (systemd on Linux, Windows Services on Windows) for servers, including creation (`create_server_service()`), enabling (`enable_server_service()`), and disabling (`disable_server_service()`) auto-start.
- Configuring server-specific settings like autoupdate behavior via `set_autoupdate()`.

These functions are designed for use by higher-level application components, such as the web UI or CLI, to provide system-level control and monitoring.

async bedrock_server_manager.api.system.**get_server_running_status**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Checks if the server process is currently running.

This function queries the operating system to determine if the Bedrock server process associated with the given server name is active by calling `is_running()`.

Parameters
server_name (*str*) – The name of the server to check.

Returns
A dictionary with the operation result. On success: `{"status": "success", "is_running": bool}` (`is_running` is `True` if the process is active, `False` otherwise). On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***BSMError*** – Can be raised by *BedrockServer* instantiation or if *is_running* encounters a critical issue (e.g., misconfiguration).

async bedrock_server_manager.api.system.**get_bedrock_process_info**(*server_name: str*, *app_context:* AppContext) → Dict[str, Any]

Retrieves resource usage for a running Bedrock server process.

This function queries the system for the server's process by calling `get_process_info()` and returns details like PID, CPU usage, memory consumption, and uptime.

Parameters
server_name (*str*) – The name of the server to query.

Returns
A dictionary with the operation status and process information. On success with a running process: `{"status": "success", "process_info": {"pid": int, "cpu_percent": float, "memory_mb": float, "uptime": str}}`. If the process is not found or inaccessible: `{"status": "success", "process_info": None, "message": "Server process '<name>' not found..."}`. On error during retrieval: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, Any]

Raises

- ***InvalidServerNameError*** – If *server_name* is not provided.
- ***BSMError*** – Can be raised by *BedrockServer* instantiation if core application settings are misconfigured, or by `get_process_info` if `psutil` is unavailable or encounters issues.

34.15 Web API Documentation

Provides API functions for managing the application's own web user interface.

This module contains the logic for controlling the lifecycle and querying the status of the built-in web UI, which is powered by FastAPI. It handles:

- Starting the web server in 'direct' (blocking) or 'detached' (background) modes (*start_web_server_api()*).
- Stopping the detached web server process (*stop_web_server_api()*).

- Checking the runtime status of the web server (*get_web_server_status_api()*).
- Managing the system service for the Web UI (create, enable, disable, remove, get status) via functions like *create_web_ui_service()* and *get_web_ui_service_status()*.

These functions are intended for programmatic control of the application's web server, often used by CLI commands or service management scripts.

bedrock_server_manager.api.web.**start_web_server_api**(*app_context:* AppContext, *host: str | None = None*, *port: int | None = None*, *debug: bool = False*, *mode: str = 'direct'*) → Dict[str, Any]

Starts the application's web server.

This function can start the web server in two modes:

- 'direct': A blocking call that runs the server in the current process Useful for development or when managed by an external process manager.
- 'detached': Launches the server as a new background process and creates a PID file to track it. Requires the *psutil* library. Uses various methods from *process* for process management.

Triggers `before_web_server_start` and `after_web_server_start` plugin events.

Parameters

- **host** (*Optional[str], optional*) – The host address to bind the web server to. If `None`, defaults to the application's configured setting (typically "127.0.0.1"). Defaults to `None`.
- **debug** (*bool, optional*) – If `True`, starts the web server (Uvicorn) in debug mode (e.g., with auto-reload). Defaults to `False`.
- **mode** (*str, optional*) – The start mode, either 'direct' or 'detached'. Defaults to 'direct'.

Returns

A dictionary with the operation result. If 'direct' mode: `{"status": "success", "message": "Web server (direct mode) shut down."}` If 'detached' mode (success): `{"status": "success", "pid": <pid>, "message": "Web server started (PID: <pid>)."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, Any]

Raises

- ***UserInputError*** – If the provided *mode* is invalid.
- ***SystemError*** – If *detached* mode is used but *psutil* is not installed.
- ***ServerProcessError*** – If the web server is already running in detached mode.
- ***BSMError*** – For other application-specific errors during startup.

bedrock_server_manager.api.web.**stop_web_server_api**(*app_context:* AppContext) → Dict[str, str]

Stops the detached web server process.

This function reads the PID from the web server's PID file, verifies that the process is the correct one using expected executable and arguments, and then terminates it. Uses utilities from *process*. Requires the *psutil* library. Triggers `before_web_server_stop` and `after_web_server_stop` plugin events.

Returns

A dictionary with the operation result. If successfully stopped: `{"status": "success", "message": "Web server (PID: <pid>) stopped."}` If not running (no PID file or stale

PID): {"status": "success", "message": "Web server not running..."} On error (e.g., PID file error, process mismatch, termination error): {"status": "error", "message": "<error_message>"}.

Return type
Dict[str, str]

Raises

- ***SystemError*** – If *psutil* is not installed.
- ***BSMError*** – Propagates errors from underlying operations, including: *FileOperationError* (PID file issues), *ServerProcessError* (process verification failure), *ServerStopError* (termination failure), *ConfigurationError* (if web UI paths not configured in BSM).

bedrock_server_manager.api.web.**get_web_server_status_api**(*app_context:* AppContext) → Dict[str, Any]

Checks the status of the web server process.

This function verifies the web server's status by checking for a valid PID file and then inspecting the process itself (using utilities from *process*) to ensure it is running and is the correct executable. Requires the *psutil* library.

Returns
A dictionary with the web server's status. The "status" field can be one of "RUNNING", "STOPPED", "MISMATCHED_PROCESS", or "ERROR". If running, "pid" (int) will be present. A "message" (str) field provides details. Example: {"status": "RUNNING", "pid": 1234, "message": "Web server running..."}

Return type
Dict[str, Any]

Raises
BSMError – Can be raised by underlying operations if critical errors occur (e.g., *ConfigurationError* if web UI paths are not set up), though many operational errors are returned in the status dictionary.

bedrock_server_manager.api.web.**create_web_ui_service**(*app_context:* AppContext, *autostart: bool = False*, *system: bool = False*, *username: str | None = None*, *password: str | None = None*) → Dict[str, str]

Creates (or updates) a system service for the Web UI.

On Linux, this creates a systemd user service. On Windows, this creates a Windows Service (typically requires Administrator privileges). This function calls `create_web_service_file()`, and then either `enable_web_service()` or `disable_web_service()` based on the *autostart* flag. Triggers `before_web_service_change` and `after_web_service_change` plugin events.

Parameters

- **autostart** (*bool, optional*) – If `True`, the service will be enabled to start automatically on system boot/login. If `False`, it will be created but left disabled. Defaults to `False`.
- **system** (*bool, optional*) – If `True`, creates a system-wide service on Linux. This option is ignored on other operating systems. Defaults to `False`.
- **username** (*Optional[str], optional*) – The username to run the service as on Windows. This option is ignored on other operating systems. Defaults to `None`.
- **password** (*Optional[str], optional*) – The password for the user on Windows. This option is ignored on other operating systems. Defaults to `None`.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Web UI system service created and <enabled/disabled> successfully."}` On error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises
BSMError – Propagates errors from the underlying service management calls, such as *SystemError*, *PermissionsError*, *CommandNotFoundError*, or *FileOperationError*.

bedrock_server_manager.api.web.**enable_web_ui_service**(*app_context:* AppContext, *system: bool = False*) → Dict[str, str]

Enables the Web UI system service for autostart.

On Linux, this enables the systemd user service. On Windows, this sets the Windows Service start type to 'Automatic' (typically requires Administrator privileges). It calls `enable_web_service()`. Triggers `before_web_service_change` and `after_web_service_change` plugin events.

Parameters
system (*bool, optional*) – If `True`, enables a system-wide service on Linux. This option is ignored on other operating systems. Defaults to `False`.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Web UI service enabled successfully."}` If service management tools are unavailable: `{"status": "error", "message": "System service management tool ... not found."}` On other error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises
BSMError – Propagates errors from the underlying `enable_web_service()` call (e.g., *SystemError*, *PermissionsError*).

bedrock_server_manager.api.web.**disable_web_ui_service**(*app_context:* AppContext, *system: bool = False*) → Dict[str, str]

Disables the Web UI system service from autostarting.

On Linux, this disables the systemd user service. On Windows, this sets the Windows Service start type to 'Disabled' or 'Manual' (typically requires Administrator privileges). It calls `disable_web_service()`. Triggers `before_web_service_change` and `after_web_service_change` plugin events.

Parameters
system (*bool, optional*) – If `True`, disables a system-wide service on Linux. This option is ignored on other operating systems. Defaults to `False`.

Returns
A dictionary with the operation result. On success: `{"status": "success", "message": "Web UI service disabled successfully."}` If service management tools are unavailable: `{"status": "error", "message": "System service management tool ... not found."}` On other error: `{"status": "error", "message": "<error_message>"}`.

Return type
Dict[str, str]

Raises

BSMError – Propagates errors from the underlying `disable_web_service()` call (e.g., *SystemError*, *PermissionsError*).

bedrock_server_manager.api.web.**remove_web_ui_service**(*app_context:* AppContext, *system: bool = False*) → Dict[str, str]

Removes the Web UI system service.

The service should ideally be stopped and disabled before removal. This function calls `remove_web_service_file()`.

> **Warning**
>
> This is a **DESTRUCTIVE** operation that removes the service definition from the system.

Triggers `before_web_service_change` and `after_web_service_change` plugin events.

Parameters

system (*bool, optional*) – If `True`, removes a system-wide service on Linux. This option is ignored on other operating systems. Defaults to `False`.

Returns

A dictionary with the operation result. On success (service removed or was not found): `{"status": "success", "message": "Web UI service removed successfully."}` If service management tools are unavailable: `{"status": "error", "message": "System service management tool ... not found."}` On other error: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, str]

Raises

BSMError – Propagates errors from the underlying `remove_web_service_file()` call (e.g., *SystemError*, *PermissionsError*, *FileOperationError*).

bedrock_server_manager.api.web.**get_web_ui_service_status**(*app_context:* AppContext, *system: bool = False*) → Dict[str, Any]

Gets the current status of the Web UI system service.

This function calls several service-related functions: `check_web_service_exists()`, `is_web_service_active()`, and `is_web_service_enabled()`.

Parameters

system (*bool, optional*) – If `True`, checks a system-wide service on Linux. This option is ignored on other operating systems. Defaults to `False`.

Returns

A dictionary with the operation result. On success: `{"status": "success", "service_exists": bool, "is_active": bool, "is_enabled": bool, "message": Optional[str]}`. - `service_exists`: `True` if the service definition is found on the system. - `is_active`: `True` if the service is currently running. - `is_enabled`: `True` if the service is set to start automatically. A "message" field may be present if service management tools are unavailable. On error during checks: `{"status": "error", "message": "<error_message>"}`.

Return type

Dict[str, Any]

34.16 Misc API Documentation

Provides API functions for miscellaneous or global operations.

This module contains functions that are not tied to a specific server instance, such as managing the global download cache for server executables. Operations are designed to be thread-safe.

async bedrock_server_manager.api.misc.**prune_download_cache**(*download_dir: str*, *keep_count: int | None = None*, *app_context: AppContext | None = None*) → Dict[str, str]

Prunes old downloaded server archives (.zip) in a directory.

This function removes older `bedrock-server-*.zip` archive files from the specified *download_dir*, keeping a specified number of the most recent files. It delegates the actual pruning logic to *prune_old_downloads()*.

The operation uses a non-blocking lock (`_misc_lock`) to ensure thread safety; if another pruning operation is in progress, this call will be skipped, returning a "skipped" status. Triggers `before_prune_download_cache` and `after_prune_download_cache` plugin events.

Parameters

- **download_dir** (*str*) – The path to the directory containing the downloaded server archives.
- **keep_count** (*Optional[int], optional*) – The number of recent archives to keep. If `None`, the value from the global application setting `retention.downloads` (defaulting to 3 if not set) is used. Defaults to `None`.

Returns

A dictionary with the operation result. Possible statuses: "success", "error", or "skipped" (if lock not acquired). Example: `{"status": "success", "message": "Download cache pruned..."}`

Return type

Dict[str, str]

Raises

- ***MissingArgumentError*** – If *download_dir* is not provided.
- ***UserInputError*** – If *keep_count* (or the `retention.downloads` setting) is not a valid non-negative integer.
- ***BSMError*** – Can be raised by the underlying prune operation for issues like *AppFileNotFoundError* (if *download_dir* is invalid after initial checks) or *FileOperationError*.

CHAPTER

THIRTYFIVE

CORE CLASS DOCUMENTATION

These are the backend core classes used in the Bedrock Server Manager, used by the higher level API functions *API Documentation*. Each class is documented with its methods and properties.

35.1 App Context

class bedrock_server_manager.context.**AppContext**(*config_dir: str | None = None*, *data_dir: str | None = None*, *db_url: str | None = None*, *log_level: str | None = None*, *logger: Logger | None = None*)

Bases: `object`

A context object that holds application-wide instances and caches.

__init__(*config_dir: str | None = None*, *data_dir: str | None = None*, *db_url: str | None = None*, *log_level: str | None = None*, *logger: Logger | None = None*)

Initializes the AppContext.

async load()

Loads the application context by initializing the settings, AppState, and Storage.

async reload()

Reloads the application context by reloading settings and all components.

async flush()

Flushes any unpersisted application state to storage.

async shutdown()

Shuts down application context components and flushes pending state to storage.

property pre_app_config: Dict[str, Any]

Lazily loads and caches the pre-application configuration dictionary from bedrock_server_manager.json (resolving CLI and Env overrides).

get_pre_app_config(*key: str*, *default: Any = None*) → Any

Retrieves a single value from the cached pre-application configuration. Supports dot notation for nested keys (e.g., 'web.cors_origins').

Parameters

- **key** (*str*) – The key of the value to retrieve.
- **default** (*Any, optional*) – The default value to return if the key is not found.

Returns

The configuration value or the default.

Return type
Any

property config_dir: str

The absolute path to the application's configuration directory.

Type
str

property data_dir: str

The absolute path to the application's data directory.

Type
str

property db_url: str

The application's configured database URL.

Type
str

property log_level: str

The application's configured log level.

Type
str

property log_dir: str

The absolute path to the application's logs directory.

Type
str

property needs_setup: bool

Indicates whether the application requires initial setup.

Evaluates to True if no user with the role 'admin' exists in the database. The result is cached internally after the first check that returns False.

Type
bool

property api: AppAPI

Lazily loads and returns the API instance.

property db: *Database*

Lazily loads and returns the Database instance.

property state: AppState

Returns the AppState instance.

property storage: Storage

Returns the Storage instance.

property settings: *Settings*

Returns the Settings instance.

property plugin_manager: *PluginManager*

Lazily loads and returns the PluginManager instance.

property task_manager: *TaskManager*

Lazily loads and returns the TaskManager instance.

property connection_manager: *ConnectionManager*

Lazily loads and returns the ConnectionManager instance.

property resource_monitor: *ResourceMonitor*

Lazily loads and returns the ResourceMonitor instance.

property log_streamer: LogStreamer

Lazily loads and returns the LogStreamer instance.

property settings_service

Returns the SettingsService instance.

property server_service

Returns the ServerService instance.

property plugin_service

Returns the PluginService instance.

property user_service

Returns the UserService instance.

property bedrock_process_manager: *BedrockProcessManager*

Lazily loads and returns the BedrockProcessManager instance.

get_server(*server_name: str*) → *BedrockServer*

Retrieve or create a BedrockServer instance.

async remove_server(*server_name: str*)

Stops a server, removes it from the process manager, and discards it from the context cache.

35.2 Task Manager

class bedrock_server_manager.web.tasks.**TaskManager**(*connection_manager: Any*, *max_workers: int | None = None*)

Bases: object

Manages background tasks using asyncio.

__init__(*connection_manager: Any*, *max_workers: int | None = None*)

Initializes the TaskManager with explicit dependencies.

async run_task(*target_function: Callable*, *username: str | None = None*, **args: Any*, ***kwargs: Any*) → str

Submits a function to be run in the background. Supports both asynchronous functions (scheduled directly) and synchronous functions (run in an executor via asyncio.to_thread).

Parameters

- **target_function** – The function to execute.
- **username** – The user associated with the task for WebSocket notifications.
- ***args** – Positional arguments for the target function.
- ****kwargs** – Keyword arguments for the target function.

Returns
The ID of the created task.

async cancel_task(*task_id: str*) → bool

Attempts to cancel a running task.

Parameters
task_id (*str*) – The ID of the task to cancel.

Returns
True if the task was found and cancellation was requested, False otherwise.

Return type
bool

async get_task(*task_id: str*) → Dict[str, Any] | None

Retrieves the status of a task.

async get_all_tasks() → Dict[str, Dict[str, Any]]

Retrieves all tasks.

async shutdown()

Waits for all background tasks to complete asynchronously.

35.3 Connection Manager

class bedrock_server_manager.web.websocket_manager.**ConnectionManager**

Bases: object

Manages WebSocket connections and topic-based subscriptions.

__init__() → None

async connect(*websocket: WebSocket*, *user: UserResponse*) → str

Accepts a new WebSocket connection, tracks it, and returns the client ID.

async disconnect(*client_id: str*)

Removes a client's connection and all their subscriptions.

async subscribe(*client_id: str*, *topic: str*)

Subscribes a client to a given topic.

async unsubscribe(*client_id: str*, *topic: str*)

Unsubscribes a client from a given topic.

async shutdown()

Gracefully disconnects all active WebSocket connections.

async send_to_client(*data: Any*, *client_id: str*)

Sends a JSON message to a single client.

async broadcast_to_topic(*topic: str*, *data: Any*)

Broadcasts a JSON message to all clients subscribed to a topic.

async send_to_user(*username: str*, *data: Any*)

Sends a JSON message to all active connections for a specific user.

35.4 Database

class bedrock_server_manager.db.database.**Database**(*db_url: str*)

Bases: object

Manages asynchronous database connections and sessions.

db_url

The database connection URL.

Type
str

engine

The SQLAlchemy async engine.

Type
AsyncEngine

SessionLocal

The async session factory.

Type
async_sessionmaker

_tables_created

Flag indicating if tables have been created.

Type
bool

__init__(*db_url: str*)

Initializes the Database instance.

Parameters
db_url (*str*) – The database connection URL.

initialize() → None

Initializes the async database engine and session factory. This method is idempotent.

session_manager() → AsyncGenerator[AsyncSession, None]

Async context manager for database sessions.

Yields
AsyncSession – An async database session.

close() → None

Closes the database connection engine synchronously if possible.

async shutdown() → None

Gracefully closes the database connection engine asynchronously.

35.5 Settings Core Documentation

class bedrock_server_manager.**Settings**(*config_dir: str*, *data_dir: str*, *app_context: Any | None = None*)

Bases: object

Manages loading, accessing, and saving application settings.

This class acts as a single source of truth for all configuration data. It handles:

- Determining appropriate application data and configuration directories based on the environment (respecting `BSM_DATA_DIR`).
- Loading settings from a database.
- Providing sensible default values for missing settings.
- Saving changes back to the database.
- Ensuring critical directories (e.g., for servers, backups, logs) exist.

Settings are stored in a key-value format in the database and can be accessed programmatically using dot-notation via the *get()* and *set()* methods (e.g., `settings.get('paths.servers')`).

A global instance of this class, named *settings*, is typically used throughout the application.

Attributes:

__init__(*config_dir: str*, *data_dir: str*, *app_context: Any | None = None*)

Initializes the Settings object.

property state: AppState

property storage: Storage

property default_config: dict

Provides the default configuration values for the application.

These defaults are used when a configuration file is not found or when a specific setting is missing from an existing configuration file. Paths are constructed dynamically based on the determined application data directory (see `_determine_app_data_dir()`).

The structure of the default configuration is as follows:

```
{
    "paths": {
        "servers": "<app_data_dir>/servers",
        "content": "<app_data_dir>/content",
        "downloads": "<app_data_dir>/.downloads",
        "backups": "<app_data_dir>/backups",
        "plugins": "<app_data_dir>/plugins",
    },
    "retention": {
        "backups": 3,
        "downloads": 3,
    },
    "web": {
        "host": "127.0.0.1",
        "jwt_secret_key": "randomly_generated_key",
        "port": 11325,
        "token_expires_weeks": 4,
    },
    "monitoring": {
        "max_retiries": 3,
        "process_interval_sec": 10,
        "player_interval_sec": 10,
    },
    "custom": {}
}
```

Returns
A dictionary of default settings with a nested structure.

Return type
dict

`get`(*key: str*, *default: Any = None*) → Any

Retrieves a setting value using dot-notation for nested access.

`async load`() → None

Loads settings from the database asynchronously.

`async set`(*key: str*, *value: Any*) → None

Sets a configuration value using dot-notation asynchronously via SettingsService.

`async reload`()

Reloads the settings from the database asynchronously.

35.6 Bedrock Server Core Documentation

`class` bedrock_server_manager.`BedrockServer`(*server_name: str*, **args: Any*, *settings: Any | None = None*, *app_context:* AppContext *| None = None*, *state: Any | None = None*, *storage: Any | None = None*, ***kwargs: Any*)

Bases: `ServerStateMixin`, `ServerProcessMixin`, `ServerInstallationMixin`, `ServerWorldMixin`, `ServerAddonMixin`, `ServerBackupMixin`, `ServerPlayerMixin`, `ServerAllowlistMixin`, `ServerPermissionsMixin`, `ServerPropertiesMixin`, `ServerInstallUpdateMixin`, `BedrockServerBaseMixin`

Represents and manages a single Minecraft Bedrock Server instance.

This class is the primary interface for all server-specific operations within the Bedrock Server Manager. It consolidates a wide range of functionalities by inheriting from the various specialized mixin classes listed above. Each instance of *BedrockServer* is tied to a unique server name and provides methods to control, configure, and maintain that server.

The order of mixin inheritance is significant for Python's Method Resolution Order (MRO), ensuring that methods are overridden and extended correctly. The `BedrockServerBaseMixin` provides core attributes and foundational initialization logic.

`server_name`

The unique name identifying this server instance.

Type
str

`settings`

The application's global settings object.

Type
Settings

`base_dir`

The base directory where all server installation directories reside (from settings: `paths.servers`).

Type
str

`server_dir`
: The full path to this specific server's installation directory (e.g., `<base_dir>/<server_name>`).

  Type
  : str

`app_config_dir`
: Path to the application's global configuration directory (from settings: `_config_dir`).

  Type
  : str

`os_type`
: The current operating system, e.g., "Linux", "Windows".

  Type
  : str

`logger`
: A logger instance specific to this server instance.

  Type
  : `logging.Logger`

> **Note**
>
> Many other attributes related to specific functionalities (e.g., server state, process information, world data paths) are available from the inherited mixin classes. Refer to the documentation of individual mixins for more details.

Key Methods (Illustrative list, grouped by typical functionality):

Installation & Validation (from `ServerInstallationMixin`):

- `is_installed()`
- `validate_installation()`
- `set_filesystem_permissions()`
- `delete_all_data()`

State Management (from `ServerStateMixin`):

- `get_status()`
- `get_version()`
- `set_version()`
- `get_world_name()`
- `get_custom_config_value()`
- `set_custom_config_value()`

Process Management (from `ServerProcessMixin`):

- `is_running()`
- `get_process_info()`
- `start()`

- `stop()`
- `send_command()`

World Management (from `ServerWorldMixin`):

- `export_world()`
- `import_world()`
- `delete_world()`

Addon Management (from `ServerAddonMixin`):

- `process_addon_file()`
- `list_installed_addons()`
- `export_addon()`
- `remove_addon()`

Backup & Restore (from `ServerBackupMixin`):

- `backup_all_data()`
- `restore_all_data_from_latest()`
- `prune_server_backups()`
- `list_backups()`

Player Log Scanning (from `ServerPlayerMixin`):

- `scan_log_for_players()`

Config File Management (from `ServerAllowlistMixin`, `ServerPermissionsMixin`, and `ServerPropertiesMixin`):

- `get_allowlist()`
- `add_to_allowlist()`
- `set_player_permission()`
- `get_server_properties()`
- `set_server_property()`

Installation & Updates (from `ServerInstallUpdateMixin`):

- `is_update_needed()`
- `install_or_update()`

> **Note**
>
> This is not an exhaustive list of all available methods. Many more specialized methods are accessible from the respective mixin classes. Please consult the documentation for each individual mixin for a comprehensive list of its capabilities.

`__init__`(*server_name: str*, **args: Any*, *settings: Any | None = None*, *app_context:* AppContext *| None = None*, *state: Any | None = None*, *storage: Any | None = None*, ***kwargs: Any*) → None

Initializes a BedrockServer instance.

This constructor is responsible for setting up a *BedrockServer* object, which represents a specific Minecraft Bedrock server. It calls `super().__init__(...)`, triggering the initialization methods of all inherited mixin classes according to Python's Method Resolution Order (MRO). This process starts with the first mixin in the inheritance list (currently `ServerStateMixin`) and culminates with `BedrockServerBaseMixin`, which establishes fundamental server attributes.

Parameters

- **server_name** (*str*) – The unique name for this server instance.
- ***args** – Variable length argument list.
- **settings** – Settings object instance.
- **app_context** (*AppContext*) – Optional instance of the application context.
- ****kwargs** – Arbitrary keyword arguments.

async get_summary_info() → Dict[str, Any]

Returns a generic summary of the server's current status and state asynchronously.

async add_to_allowlist(*players_to_add: List[Dict[str, Any]]*) → int

Adds players to the allowlist asynchronously.

property allowlist_json_path: str

The absolute path to this server's `allowlist.json` file.

Type
str

async backup_all_data() → Dict[str, str | None]

Performs a full backup of the server's active world and standard configuration files.

This method orchestrates the backup of the following components:

- The active world: Determined by `await self.get_world_name()` (from `ServerStateMixin`), then backed up to a `.mcworld` file via `_backup_world_data_internal()`.
- `allowlist.json`: Backed up via `_backup_config_file_internal()`.
- `permissions.json`: Backed up via `_backup_config_file_internal()`.
- `server.properties`: Backed up via `_backup_config_file_internal()`.

Each component is backed up individually. The server's specific backup directory (derived from *server_backup_directory*) is created if it doesn't already exist.

If the critical world backup fails, a *BackupRestoreError* is raised *after* attempting to back up all configuration files. Failures in backing up individual configuration files are logged as errors, and their corresponding entry in the returned dictionary will be `None`, but they do not stop the backup of other components.

Returns
A dictionary mapping component names (e.g., "world", "allowlist.json") to the absolute path of their backup file. If a component's backup failed or was skipped (e.g., the original file was not found), its value will be `None`.

Return type
Dict[str, Optional[str]]

Raises

- ***ConfigurationError*** – If the server's backup directory path (*server_backup_directory*) is not configured in settings.

- *FileOperationError* – If creation of the main server backup directory (under the global backup path) fails.
- *BackupRestoreError* – If the critical world backup operation fails. Other underlying errors from helper methods (like *AppFileNotFoundError* from `_backup_world_data_internal()` if the world directory is missing) can also propagate.
- **AttributeError** – If required methods from other mixins (e.g., `get_world_name()` from `ServerStateMixin` or `export_world()` from `ServerWorldMixin`) are not available.

property bedrock_executable_name: str

The platform-specific name of the Bedrock server executable. Returns "bedrock_server.exe" on Windows, "bedrock_server" otherwise.

Type
str

property bedrock_executable_path: str

The full, absolute path to this server's Bedrock executable. Constructed by joining *self.server_dir* and *self.bedrock_executable_name*.

Type
str

async delete_all_data() → None

Deletes **ALL** data associated with this Bedrock server instance asynchronously.

async delete_server_files(*item_description_prefix: str = 'server installation files for'*) → bool

Deletes the server's entire installation directory asynchronously.

async delete_world() → bool

Deletes the server's currently active world directory asynchronously.

> **Warning**
>
> This is a **DESTRUCTIVE** operation. The active world's data will be permanently removed. The server will typically generate a new world with the same name on its next startup if the `level-name` in `server.properties` is not changed.

This method determines the active world's directory path using `_get_active_world_directory_path()` and then uses the robust deletion utility *delete_path_robustly()* to remove it.

Returns
`True` if the active world directory was successfully deleted or if it did not exist initially.

Return type
bool

Raises

- *FileOperationError* – If determining the world path fails (e.g., due to issues with `server.properties` or if the path is not a directory), or if the deletion itself fails critically (though *delete_path_robustly* attempts to handle many common issues).
- *AppFileNotFoundError* – If `server.properties` is missing (propagated from `_get_active_world_directory_path()` via *get_world_name*).

- *ConfigParseError* – If `level-name` is missing from `server.properties` (propagated).
- **AttributeError** – If `get_world_name()` method (from StateMixin) is not available.

async disable_addon(*pack_uuid: str*, *pack_type: str*, *world_name: str | None = None*) → None

Disables an addon by removing it from the world's activation list asynchronously, preserving files.

Parameters

- **pack_uuid** (*str*) – The UUID of the pack to disable.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **world_name** (*Optional[str]*) – The name of the world.

async enable_addon(*pack_uuid: str*, *pack_type: str*, *world_name: str | None = None*) → None

Enables a physically installed addon in a world asynchronously.

Parameters

- **pack_uuid** (*str*) – The UUID of the pack to enable.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **world_name** (*Optional[str]*) – The name of the world.

async export_addon(*pack_uuid: str*, *pack_type: str*, *export_dir: str*, *world_name: str | None = None*) → str

Exports a specific installed addon from a world into a `.mcpack` file asynchronously.

This method locates an installed behavior or resource pack within the specified world by its UUID, then archives its contents into a new `.mcpack` file. The exported file is named using the pack's name and version (e.g., `MyPack_1.0.0.mcpack`) and saved in the `export_dir`.

Parameters

- **pack_uuid** (*str*) – The UUID of the pack to export.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **export_dir** (*str*) – The absolute path to the directory where the `.mcpack` file will be saved. This directory will be created if it does not already exist.
- **world_name** (*Optional[str]*) – The name of the world from which to export the addon. If `None` (default), uses the server's currently active world name.

Returns

The absolute path to the created `.mcpack` file.

Return type

str

Raises

- *MissingArgumentError* – If `pack_uuid`, `pack_type`, or `export_dir` are empty or not provided.
- *UserInputError* – If `pack_type` is not `"behavior"` or `"resource"`.
- *AppFileNotFoundError* – If the specified pack (by UUID and type) cannot be found in the physical `behavior_packs` or `resource_packs` folder of the world, or if the world directory itself is missing.
- *FileOperationError* – If any OS-level error occurs during directory creation, file scanning, or `.mcpack` archive creation (e.g., permission issues, disk full).

- **AttributeError** – If methods like `get_world_name()` are unavailable when `world_name` is `None`.

async export_world(*world_dir_name: str*, *target_mcworld_file_path: str*) → None

Exports a specified world directory into a `.mcworld` archive file asynchronously.

This method takes the name of a world directory (located within the server's "worlds" folder), archives its entire contents into a ZIP file, and then renames this ZIP file to have a `.mcworld` extension, saving it to *target_mcworld_file_path*.

If the server is running and the specified world is currently active, a live backup using *save hold* / *save query* / *save resume* is performed.

The parent directory for *target_mcworld_file_path* will be created if it does not exist. If *target_mcworld_file_path* itself already exists, it will be overwritten. A temporary `.zip` file is created during the process and is cleaned up.

Parameters

- **world_dir_name** (*str*) – The name of the world directory to export, relative to the server's "worlds" folder (e.g., "MyFavoriteWorld").
- **target_mcworld_file_path** (*str*) – The absolute path where the resulting `.mcworld` archive file should be saved.

Raises

- ***MissingArgumentError*** – If *world_dir_name* or *target_mcworld_file_path* are empty or not strings.
- ***AppFileNotFoundError*** – If the source world directory (`<server_dir>/worlds/<world_dir_name>`) does not exist or is not a directory.
- ***FileOperationError*** – If creating the parent directory for the *target_mcworld_file_path* fails due to an `OSError`.
- ***BackupRestoreError*** – If creating the ZIP archive (via `shutil.make_archive`) or renaming it to `.mcworld` fails, or for other unexpected errors during the export process. This can wrap underlying `OSError` or other exceptions.

async extract_mcworld(*mcworld_file_path: str*, *target_world_dir_name: str*) → str

Extracts a `.mcworld` archive file into a specified world directory name asynchronously.

The extraction target is a subdirectory named *target_world_dir_name* within the server's main "worlds" folder (see `_worlds_base_dir_in_server`).

> **Warning**
>
> If the *target_world_dir_name* directory already exists, **it will be deleted** before extraction to ensure a clean import.

Parameters

- **mcworld_file_path** (*str*) – The absolute path to the `.mcworld` file to be extracted.
- **target_world_dir_name** (*str*) – The desired name for the world directory that will be created inside the server's "worlds" folder to contain the extracted content.

Returns

The absolute path to the directory where the world was extracted (e.g., `<server_dir>/worlds/<target_world_dir_name>`).

Return type
str

Raises

- *MissingArgumentError* – If *mcworld_file_path* or *target_world_dir_name* are empty or not strings.
- *AppFileNotFoundError* – If the source *mcworld_file_path* does not exist or is not a file.
- *FileOperationError* – If creating the target directory structure or clearing a pre-existing target directory fails (e.g., due to permissions or other `OSError`).
- *ExtractError* – If the `.mcworld` file is not a valid ZIP archive or if an error occurs during the extraction process itself.

`async get_allowlist()` → List[Dict[str, Any]]

Reads the *allowlist.json* file asynchronously and returns its contents.

`async get_autostart()` → bool

Retrieves the 'autostart' setting from the server's config asynchronously.

Accesses `settings.autostart` via `_manage_json_config()`.

Returns
The autostart status (`True` or `False`). Defaults to `False` if the setting is not found or an error occurs during retrieval.

Return type
bool

`async get_autoupdate()` → bool

Retrieves the 'autoupdate' setting from the server's config asynchronously.

Accesses `settings.autoupdate` via `_manage_json_config()`.

Returns
The autoupdate status (`True` or `False`). Defaults to `False` if the setting is not found or an error occurs during retrieval.

Return type
bool

`async get_custom_config_value(`*key: str*`)` → Any | None

Retrieves a custom value from the 'custom' section of the server's config asynchronously.

Accesses `custom.<key>` via `_manage_json_config()`.

Parameters
key (*str*) – The key of the custom value to retrieve.

Returns
The retrieved custom value, or `None` if the key is not found or an error occurs.

Return type
Optional[Any]

Raises
UserInputError – If *key* is not a non-empty string.

get_file_lock(*filepath: str*) → ReentrantAsyncLock

Retrieves or creates a ReentrantAsyncLock for the specified filepath.

This ensures that asynchronous operations (like atomic JSON writes) do not concurrently collide when targeting the same configuration file, while allowing re-entrant locks within the same task.

Parameters
filepath (*str*) – The absolute path to the file.

Returns
The re-entrant lock associated with the given file.

Return type
ReentrantAsyncLock

async get_formatted_permissions(*storage: Any*) → List[Dict[str, Any]]

Retrieves permissions and maps XUIDs to known player names asynchronously.

get_pid_file_path() → str

Gets the full, absolute path to this server's primary PID file.

This file is conventionally used to store the process ID of the running Bedrock server instance, especially when it's managed as a background process or service. The path is constructed using this server's *server_config_dir* and the filename generated by `_get_server_pid_filename_default()`.

Returns
The absolute path to the PID file.

Return type
str

async get_process_info() → Dict[str, Any] | None

Gets resource usage information (PID, CPU, Memory, Uptime) for the running server process asynchronously.

This method first uses *get_verified_bedrock_process()* to locate and verify the Bedrock server process associated with this server instance. If a valid process is found, it then uses the `_resource_monitor` (an instance of *ResourceMonitor* from the base mixin) to calculate its current resource statistics.

Returns
A dictionary containing process information if the server is running, verified, and `psutil` is available. The dictionary has keys: "pid", "cpu_percent", "memory_mb", "uptime". Returns `None` if the server is not running, cannot be verified, `psutil` is unavailable, or if an error occurs during statistics retrieval. Example: `{"pid": 1234, "cpu_percent": 15.2, "memory_mb": 256.5, "uptime": "0:10:30"}`

Return type
Optional[Dict[str, Any]]

async get_server_properties() → Dict[str, str]

Reads the *server.properties* file asynchronously and returns its contents.

async get_server_property(*property_key: str*, *default: Any | None = None*) → Any | None

Reads a specific property asynchronously.

async get_status() → str

Determines and returns the current reconciled operational status of the server asynchronously.

This method attempts to determine if the server process is actually running (by calling `self.is_running()`, which is expected to be provided by another mixin like `ProcessMixin`). It

then compares this live status with the status stored in the server's configuration (retrieved via *get_status_from_config()*).

If a discrepancy is found (e.g., process is running but config says "STOPPED", or vice-versa when config said "RUNNING"), it updates the stored status in the config to reflect the actual state.

Returns
The reconciled operational status of the server as a string (e.g., "RUNNING", "STOPPED"). If `self.is_running()` is not available or fails, it falls back to returning the last known status from config.

Return type
str

async get_status_from_config() → str

Retrieves the stored 'status' from the server's config asynchronously.

Accesses `server_info.status` via `_manage_json_config()`. This reflects the last known status written to the config, not necessarily the live process status. For live status, use *get_status()*.

Returns
The stored status string (e.g., "RUNNING", "STOPPED"), or "UNKNOWN" if not set or on error.

Return type
str

async get_target_version() → str

Retrieves the 'target_version' from the server's config asynchronously.

Accesses `settings.target_version` via `_manage_json_config()`. This indicates the version the server aims to be on, often "LATEST" or a specific version string.

Returns
The target version string, or "LATEST" if not set or on error.

Return type
str

async get_version() → str

Retrieves the 'installed_version' from the server's config asynchronously.

Accesses `server_info.installed_version` via `_manage_json_config()`.

Returns
The installed version string, or "UNKNOWN" if not set or on error.

Return type
str

async get_world_icon_filesystem_path() → str | None

Optional[str]: The absolute filesystem path to the world icon for the active world.

This is constructed by joining the active world's directory path (from `_get_active_world_directory_path()`) with the standard icon filename (*world_icon_filename*).

Returns `None` if the active world directory path cannot be determined (e.g., if `get_world_name()` fails or is unavailable).

async get_world_name() → str

Reads the `level-name` property from the server's `server.properties` file asynchronously.

Returns
The name of the world as specified in `server.properties`.

Return type
str

Raises

- ***AppFileNotFoundError*** – If the `server.properties` file does not exist at the expected path (*server_properties_path*).
- ***ConfigParseError*** – If the file cannot be read (e.g., due to permissions) or if the `level-name` key is missing, malformed, or has an empty value.

async has_world_icon() → bool

Checks if the standard world icon file (`world_icon.jpeg`) exists for the active world asynchronously.

This method uses `world_icon_filesystem_path` to determine the expected location of the icon and checks if a file exists at that path.

Returns
`True` if the world icon file exists and is a regular file, `False` otherwise (e.g., path cannot be determined, file does not exist, or is not a file).

Return type
bool

async import_world(*mcworld_backup_file_path: str*) → str

Imports a `.mcworld` file asynchronously, replacing the server's currently active world.

> **Warning**
>
> This is a **DESTRUCTIVE** operation. The existing active world directory will be deleted before the new world is imported.

This method first determines the name of the server's active world by calling `await self.get_world_name()` (expected from `ServerStateMixin`). It then uses *extract_mcworld()* to extract the contents of the provided *mcworld_backup_file_path* into a directory with that active world name, effectively replacing it.

Parameters
mcworld_backup_file_path (*str*) – The absolute path to the source `.mcworld` file that contains the world data to import.

Returns
The name of the world directory (which is the active world name) that the `.mcworld` file was imported into.

Return type
str

Raises

- ***MissingArgumentError*** – If *mcworld_backup_file_path* is empty or not a string.
- ***AppFileNotFoundError*** – If the source *mcworld_backup_file_path* does not exist.
- ***BackupRestoreError*** – If any part of the import process fails, including failure to determine the active world name, or errors during extraction (which can wrap *ExtractError*, *FileOperationError*, etc.).

- **AttributeError** – If `get_world_name()` is missing.

async install_or_update(*target_version_specification: str*, *force_reinstall: bool = False*, *server_zip_path: str | None = None*) → None

Installs or updates the Bedrock server to a specified version or dynamic target asynchronously.

This is the primary method for managing server software versions. It orchestrates the entire workflow:

1. Checks if an update is needed using *is_update_needed()* (unless *force_reinstall* is `True` or the server isn't installed).
2. If the server is running, stops it using `self.stop()` (expected from `ServerProcessMixin`).
3. Updates the server's persisted status to "INSTALLING" or "UPDATING" (via `await self.set_status_in_config()` from `ServerStateMixin`).
4. If it's a new installation, sets the target version in the config.
5. Initializes a *BedrockDownloader* for the *target_version_specification*.
6. Calls *BedrockDownloader.prepare_download_assets()* to download/verify files.
7. Calls the internal helper `_perform_server_files_setup()` to extract the archive and set permissions. This helper, in turn, relies on `self.set_filesystem_permissions()` (expected from another mixin).
8. Updates the server's persisted installed version (via `await self.set_version()` from `ServerStateMixin`) and final status ("INSTALLED" or "UPDATED").
9. Cleans up the downloaded ZIP archive.

Parameters

- **target_version_specification** (*str*) – The target version to install or update to. Can be a specific version string (e.g., "1.20.10.01"), "LATEST" (for the latest stable release), or "PREVIEW" (for the latest preview release).
- **force_reinstall** (*bool, optional*) – If `True`, the server software will be reinstalled/extracted even if *is_update_needed()* reports that the current version matches the target. Defaults to `False`.
- **server_zip_path** (*str, optional*) – A local path to a pre-downloaded Bedrock server zip file.

Raises

- ***MissingArgumentError*** – If *target_version_specification* is empty.
- ***ServerStopError*** – If the server is running and fails to stop.
- ***DownloadError*** – If the server software download fails.
- ***ExtractError*** – If the downloaded server archive cannot be extracted.
- ***PermissionsError*** – If filesystem permissions cannot be set after extraction.
- ***FileOperationError*** – For other unexpected file I/O errors during the process.
- **AttributeError** – If essential methods from other mixins (like *is_installed*, *stop*, *set_status_in_config*, *set_version*, *set_filesystem_permissions*) are not available on the instance.
- ***BSMError*** – For other known application-specific errors during the process.

async is_installed() → bool

Checks if the server installation is valid asynchronously, without raising exceptions.

async is_running() → bool

Checks if the Bedrock server process is currently running and verified asynchronously.

async is_update_needed(*target_version_specification: str*) → bool

Checks if the server's installed version requires an update to meet the target asynchronously.

This method compares the server's currently installed version (obtained via `await self.get_version()`, expected from `ServerStateMixin`) against the *target_version_specification*. The target can be:

- A specific version string (e.g., "1.20.10.01").
- "LATEST" (for the latest stable release).
- "PREVIEW" (for the latest preview release).

If the target is "LATEST" or "PREVIEW", this method uses *BedrockDownloader* to fetch the actual latest version number corresponding to that specification for comparison. If the target is a specific version, it's compared directly.

Parameters
target_version_specification (*str*) – The target version to check against (e.g., "1.20.10.01", "LATEST", "PREVIEW").

Returns
`True` if an update is determined to be needed, `False` otherwise. Returns `True` as a fail-safe if the current version is "UNKNOWN" or if there are errors fetching remote version information for "LATEST"/"PREVIEW".

Return type
bool

Raises
- ***MissingArgumentError*** – If *target_version_specification* is empty or not a string.
- **AttributeError** – If `await self.get_version()` method is not available.

async list_backups(*backup_type: str*) → List[str] | Dict[str, List[str]]

Retrieves a list of available backup files for this server, sorted newest first.

This method scans the server's specific backup directory (obtained via *server_backup_directory*) for backup files matching the specified `backup_type`. Backups are sorted by their modification time, with the most recent backup appearing first.

Valid `backup_type` options (case-insensitive):

- `"world"`: Lists `*.mcworld` files (world backups).
- `"properties"`: Lists `server_backup_*.properties` files.
- `"allowlist"`: Lists `allowlist_backup_*.json` files.
- `"permissions"`: Lists `permissions_backup_*.json` files.
- `"all"`: Returns a dictionary categorizing all found backup types.

Parameters
backup_type (*str*) – The type of backups to list. Must be one of "world", "properties", "allowlist", "permissions", or "all".

Returns

- If `backup_type` is specific (e.g., "world"), returns a list of absolute backup file paths, sorted by modification time (newest first). An empty list is returned if no matching backups are found or if the server's backup directory doesn't exist.
- If `backup_type` is "all", returns a dictionary where keys are backup categories (e.g., "world_backups", "properties_backups") and values are the corresponding sorted lists of file paths. An empty dictionary is returned if the backup directory doesn't exist or no backups of any type are found.

Return type
Union[List[str], Dict[str, List[str]]]

Raises

- ***MissingArgumentError*** – If `backup_type` is empty or not a string.
- ***UserInputError*** – If `backup_type` is not one of the valid options.
- ***ConfigurationError*** – If the server's backup directory is not configured (i.e., *server_backup_directory* is `None`).
- ***FileOperationError*** – If an `OSError` occurs during filesystem listing (e.g., permission issues).

async list_installed_addons(*world_name: str | None = None*) → Dict[str, List[Dict[str, Any]]]

Lists all behavior and resource packs for a specified world asynchronously.

This method provides a detailed inventory of addons by:

1. Scanning the physical pack folders (`behavior_packs` and `resource_packs`) within the specified world's directory to find all installed packs by reading their `manifest.json` files.
2. Reading the world's activation JSON files (`world_behavior_packs.json` and `world_resource_packs.json`) to determine which packs are active.
3. Comparing these two sets of information to determine the status of each pack.

The status can be:

- `ACTIVE`: The pack is physically present and listed in the activation file.
- `INACTIVE`: The pack is physically present but not listed in the activation file.
- `ORPHANED`: The pack is listed in the activation file but not physically present.

Parameters
world_name (*Optional[str]*) – The name of the world to inspect. If `None` (default), uses the server's currently active world name obtained via `get_world_name()`.

Returns

A dictionary with two keys: `"behavior_packs"` and `"resource_packs"`. Each key maps to a list of dictionaries, where each dictionary represents an addon with the following string keys:

- `"name"` (str): The display name of the pack from its manifest.
- `"uuid"` (str): The UUID of the pack from its manifest.
- `"version"` (List[int]): The version of the pack (e.g., `[1, 0, 0]`).
- `"status"` (str): The activation status: 'ACTIVE', 'INACTIVE', or 'ORPHANED'.

The lists of pack dictionaries are sorted by pack name.

Return type
Dict[str, List[Dict[str, Any]]]

Raises

- *AppFileNotFoundError* – If the directory for the specified `world_name` does not exist.
- **AttributeError** – If `get_world_name()` is not available when `world_name` is `None`.

property permissions_json_path: str

The absolute path to this server's `permissions.json` file.

Type
str

async process_addon_file(*addon_file_path: str*) → None

Processes an addon file asynchronously.

This method acts as a high-level dispatcher for asynchronous addon processing. It inspects the file extension of the provided `addon_file_path` to determine if it's an `.mcaddon` or `.mcpack` file. It then delegates the actual processing to the corresponding internal helper methods: `_process_mcaddon_archive()` for `.mcaddon` files or `_process_mcpack_archive()` for `.mcpack` files.

Parameters
addon_file_path (*str*) – The absolute path to the addon file (`.mcaddon` or `.mcpack`) to be processed.

Raises

- *MissingArgumentError* – If `addon_file_path` is empty or not provided.
- *AppFileNotFoundError* – If the file specified by `addon_file_path` does not exist or is not a file.
- *UserInputError* – If the file extension is not `.mcaddon` or `.mcpack` (case-insensitive).

async prune_server_backups(*component_prefix: str*, *file_extension: str*) → None

Removes the oldest backups for a specific component to adhere to retention policies.

This method targets backup files within this server's specific backup directory

(see *server_backup_directory*) that match a given `component_prefix` (e.g., `MyActiveWorld_backup_`, `server_backup_`) and `file_extension` (e.g., `mcworld`, `properties`, `json`).

It retrieves the number of backups to keep from the application settings (key: `retention.backups`, defaulting to 3 if not set or invalid). If more backups than this configured number are found (sorted by modification time), the oldest ones are deleted until the retention count is met.

Parameters

- **component_prefix** (*str*) – The prefix part of the backup filenames to target (e.g., `MyActiveWorld_backup_` for world backups, `server_backup_` for server.properties backups). Should not be empty.
- **file_extension** (*str*) – The extension of the backup files, without the leading dot (e.g., "mcworld", "json", "properties"). Should not be empty.

Raises

- *ConfigurationError* – If the server's backup directory path (*server_backup_directory*) is not configured in settings.

- *MissingArgumentError* – If `component_prefix` or `file_extension` are empty or not strings.
- *UserInputError* – If the `retention.backups` setting value from application settings is invalid (e.g., not a non-negative integer).
- *FileOperationError* – If an `OSError` occurs during file listing or deletion (e.g., permission issues), or if not all required old backups could be deleted successfully.

async remove_addon(*pack_uuid: str*, *pack_type: str*, *world_name: str | None = None*) → None

Removes a specific addon from a world asynchronously.

> **Warning**
>
> This is a destructive operation. It permanently deletes the addon's files from the world's `behavior_packs` or `resource_packs` directory and deactivates the addon by removing its entry from the world's corresponding activation JSON file (e.g., `world_behavior_packs.json`).

If the addon's files are not found, it will still attempt to remove its entry from the activation JSON file.

Parameters

- **pack_uuid** (*str*) – The UUID of the pack to remove.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **world_name** (*Optional[str]*) – The name of the world from which to remove the addon. If `None` (default), uses the server's currently active world name.

Raises

- *MissingArgumentError* – If `pack_uuid` or `pack_type` are empty or not provided.
- *UserInputError* – If `pack_type` is not `"behavior"` or `"resource"`.
- *FileOperationError* – If an OS-level error occurs during file/directory deletion or when updating the world's activation JSON file.
- **AttributeError** – If methods like `get_world_name()` are unavailable when `world_name` is `None`.

async remove_from_allowlist(*player_name_to_remove: str*) → bool

Removes a player from the allowlist asynchronously.

async reorder_addons(*uuids: List[str]*, *pack_type: str*, *world_name: str | None = None*) → None

Reorders the active addons based on a provided list of UUIDs asynchronously.

This method strictly verifies that the provided list of UUIDs contains the exact same set of active UUIDs.

Parameters

- **uuids** (*List[str]*) – The exact active UUIDs in their new order.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **world_name** (*Optional[str]*) – The name of the world.

async restore_all_data_from_latest() → Dict[str, str | None]

Restores the server's active world and standard configuration files from their latest backups.

This method attempts to restore the following components by finding their most recent backup file (sorted by modification time) in the server's specific backup directory (see *server_backup_directory*):

- The active world: Restored using `import_world()` after finding the latest `.mcworld` backup matching the active world's name.
- `server.properties`: Restored via `_restore_config_file_internal()`.
- `allowlist.json`: Restored via `_restore_config_file_internal()`.
- `permissions.json`: Restored via `_restore_config_file_internal()`.

Each component is restored individually. If a backup for a specific component is not found, or if the restore operation for that component fails, the issue is logged, and the process continues with other components. A *BackupRestoreError* is raised at the end if any component failed to restore, summarizing all failures.

The server's main installation directory (`server_dir`) is created if it doesn't exist before attempting to restore files into it.

> **Warning**
>
> This operation **overwrites** current world data and configuration files in the server's installation directory with content from the backups. Ensure this is the desired action before proceeding.

Returns
A dictionary mapping component names (e.g., "world", "server.properties") to the absolute path where they were restored in the server's installation directory. If a component's restore was skipped (e.g., no backup found) or failed, its value in the dictionary will be `None`. Returns an empty dictionary if the server's backup directory itself is not found or is inaccessible.

Return type
Dict[str, Optional[str]]

Raises

- *ConfigurationError* – If the server's backup directory path (*server_backup_directory*) is not configured in settings.
- *FileOperationError* – If creation of the server's main installation directory (`self.server_dir`) fails.
- *BackupRestoreError* – If one or more components (world or configuration files) fail to restore. The error message will summarize all failures.
- **AttributeError** – If required methods from other mixins (e.g., `get_world_name()` or `import_world()`) are not available on the server instance.

async scan_log_for_players(*incremental: bool = False*) → List[Dict[str, str]]

Scans the server's log file for player connection entries to extract gamertags and XUIDs asynchronously.

This method reads the server's primary output log file (obtained via `server_log_path`) to find player connections. It collects unique players based on their XUID to avoid duplicates.

Parameters
incremental (*bool*) – If True, starts reading from the last recorded position instead of the beginning. Useful for periodic polling to save memory.

Returns

A list of unique player data dictionaries found in the log. Each dictionary has two keys:

- "name" (str): The player's gamertag.
- "xuid" (str): The player's Xbox User ID (XUID).

Returns an empty list if the log file doesn't exist, is empty, or if no player connection entries are found.

Return type
List[Dict[str, str]]

Raises
FileOperationError – If an OS-level error occurs while trying to read the log file (e.g., permission issues).

async send_command(*command: str*) → None

Sends a command string to the running Bedrock server process asynchronously.

property server_backup_directory: str | None

The absolute path to this server's specific backup directory.

This path is constructed by joining the global backup directory path (from `settings.get("paths.backups")`) with the server's name (`server_name`). The directory itself is created by backup methods if it doesn't exist.

Returns
The absolute path to the backup directory if `paths.backups` is configured in settings, otherwise `None` (and a warning is logged).

Type
Optional[str]

property server_config_dir: str

The absolute path to this server instance's dedicated configuration subdirectory. This is usually `<app_config_dir>/<server_name>/`, and is intended to store server-specific configuration files, PID files, etc. The directory itself is not created by this property.

Type
str

property server_log_path: str

The expected absolute path to the server's main output log file. Typically `<server_dir>/server_output.txt`.

Type
str

property server_properties_path: str

The absolute path to this server's `server.properties` file.

Type
str

async set_autostart(*value: bool*) → None

Sets the 'autostart' setting in the server's config asynchronously.

Updates `settings.autostart` via `_manage_json_config()`.

Parameters
value (*bool*) – The boolean value to set for autostart.

Raises
UserInputError – If *value* is not a boolean.

async set_autoupdate(*value: bool*) → None

Sets the 'autoupdate' setting in the server's config asynchronously.

Updates `settings.autoupdate` via `_manage_json_config()`.

Parameters
value (*bool*) – The boolean value to set for autoupdate.

Raises
UserInputError – If *value* is not a boolean.

async set_custom_config_value(*key: str*, *value: Any*) → None

Sets a custom key-value pair in the 'custom' section of the server's config asynchronously.

Updates `custom.<key>` via `_manage_json_config()`.

Parameters

- **key** (*str*) – The key for the custom value.
- **value** (*Any*) – The value to set. Must be JSON serializable.

Raises

- *UserInputError* – If *key* is not a non-empty string.
- *ConfigParseError* – If *value* is not JSON serializable (from underlying save).

async set_filesystem_permissions() → None

Sets appropriate filesystem permissions for the server's installation directory asynchronously.

async set_player_permission(*xuid: str*, *permission_level: str*, *player_name: str | None = None*) → None

Sets the permission level for a player asynchronously.

async set_server_property(*property_key: str*, *property_value: Any*) → None

Updates a specific property in *server.properties* asynchronously.

async set_status_in_config(*status_string: str*) → None

Sets the 'status' in the server's config asynchronously.

Updates `server_info.status` via `_manage_json_config()`. This is used to persist the server's state.

Parameters
status_string (*str*) – The status string to set (e.g., "RUNNING", "STOPPED").

Raises
UserInputError – If *status_string* is not a string.

async set_target_version(*version_string: str*) → None

Sets the 'target_version' in the server's config asynchronously.

Updates `settings.target_version` via `_manage_json_config()`.

Parameters
version_string (*str*) – The target version string to set (e.g., "LATEST", "1.20.30.02").

Raises
UserInputError – If *version_string* is not a string.

async set_version(*version_string: str*) → None

Sets the 'installed_version' in the server's config asynchronously.

Updates `server_info.installed_version` via `_manage_json_config()`.

Parameters
version_string (*str*) – The version string to set (e.g., "1.20.30.02").

Raises
UserInputError – If *version_string* is not a string.

async start() → None

Starts the Bedrock server process asynchronously.

async stop() → None

Stops the Bedrock server process gracefully, with a forceful fallback asynchronously.

async update_online_players() → List[Dict[str, str]]

Incrementally parses the server log to update the list of currently online players asynchronously.

Reads new lines from the log file starting from the last known cursor position (*self._log_file_cursor*), updates the *self.players* attribute, and saves the new cursor position.

Returns
The updated list of dictionaries for each currently online player, containing their "name" and "xuid".

Return type
List[Dict[str, str]]

async update_subpack(*pack_uuid: str*, *pack_type: str*, *subpack_name: str*, *world_name: str | None = None*) → None

Updates the active subpack for an already enabled addon asynchronously.

Parameters

- **pack_uuid** (*str*) – The UUID of the active pack.
- **pack_type** (*str*) – The type of pack; must be either `"behavior"` or `"resource"`.
- **subpack_name** (*str*) – The new subpack folder name to set.
- **world_name** (*Optional[str]*) – The name of the world.

async validate_installation() → bool

Validates that the server installation directory and executable exist asynchronously.

property world_icon_filename: str

The standard filename for a world's icon image (`world_icon.jpeg`).

Type
str

35.7 Bedrock Downloader Core Documentation

class bedrock_server_manager.**BedrockDownloader**(*settings_obj: Settings*, *server_dir: str*, *target_version: str = 'LATEST'*, *server_zip_path: str | None = None*)

Bases: `object`

Manages the download, extraction, and setup of a Bedrock Server instance.

This class orchestrates the process of obtaining and preparing Minecraft Bedrock server files for a specific server instance. It handles:

- **Version Targeting**: Allows specifying "LATEST" (stable), "PREVIEW", or a concrete version number (e.g., "1.20.10.01", "1.20.10.01-PREVIEW").

- **URL Resolution**: Primarily uses the official Minecraft download API to find the correct download URL for the target version and operating system (Linux/Windows).
- **Downloading**: Downloads the server ZIP archive from the resolved URL, saving it to a version-specific (stable/preview) subdirectory within the configured global downloads path. It skips downloading if the file already exists.
- **Extraction**: Extracts the contents of the downloaded ZIP archive into the specified server directory. It supports "fresh install" and "update" modes, where the latter preserves user data like worlds, properties, and allowlists.
- **Cache Pruning**: After a download, it can trigger pruning of older ZIP files within its specific download subdirectory (stable or preview) based on retention settings.

An instance of this class is typically created when a new server needs to be installed or an existing one updated.

`settings`

The application settings object.

Type
Settings

`server_dir`

The absolute path to the target server directory.

Type
str

`input_target_version`

The user-provided version string.

Type
str

`os_name`

The name of the current operating system (e.g., "Linux", "Windows").

Type
str

`base_download_dir`

The root directory for all downloads.

Type
Optional[str]

`resolved_download_url`

The final URL used for downloading.

Type
Optional[str]

`actual_version`

The specific version number resolved (e.g., "1.20.10.01").

Type
Optional[str]

`zip_file_path`

Full path to the downloaded server ZIP file.

Type
Optional[str]

specific_download_dir

Path to the subdirectory within *base_download_dir* used for this instance's downloads (e.g., "…/downloads/stable").

Type
Optional[str]

__init__(*settings_obj: Settings*, *server_dir: str*, *target_version: str = 'LATEST'*, *server_zip_path: str | None = None*)

Initializes the BedrockDownloader.

Parameters

- **settings_obj** (Settings) – The application's `Settings` object, providing access to configuration like download paths.
- **server_dir** (*str*) – The target directory where the server files will be installed or updated. This path will be converted to an absolute path.
- **target_version** (*str, optional*) – The version identifier for the server to download. Defaults to "LATEST". Examples: "LATEST", "PREVIEW", "CUSTOM", "1.20.10.01", "1.20.10.01-PREVIEW".
- **server_zip_path** (*str, optional*) – Absolute path to a custom server ZIP file. Required if *target_version* is "CUSTOM".

Raises

- ***MissingArgumentError*** – If *settings_obj*, *server_dir*, or *target_version* are not provided or are empty.
- ***ConfigurationError*** – If the *paths.downloads* setting is missing or empty in the provided *settings_obj*.

PRESERVED_ITEMS_ON_UPDATE: Set[str] = {'allowlist.json', 'permissions.json', 'server.properties', 'worlds/'}

server_zip_path: str | None

get_actual_version() → str | None

Returns the resolved actual version string of the server.

This value is populated after *get_version_for_target_spec()* or *prepare_download_assets()* has been successfully called.

Returns
The actual version string (e.g., "1.20.10.01"), or `None` if the version has not been resolved yet.

Return type
Optional[str]

get_zip_file_path() → str | None

Returns the absolute path to the downloaded (or identified) server ZIP file.

This value is populated after *prepare_download_assets()* has successfully identified or downloaded the ZIP file.

Returns
The full path to the server ZIP file, or `None` if not yet determined.

Return type
Optional[str]

`get_specific_download_dir()` → str | None

Returns the specific download directory used for this instance's downloads.

This is typically a subdirectory like 'stable' or 'preview' within the main download directory. It's populated by *prepare_download_assets()*.

Returns
The absolute path to the instance's specific download directory (e.g., `/path/to/downloads/stable`), or `None` if not yet determined.

Return type
Optional[str]

`get_resolved_download_url()` → str | None

Returns the fully resolved download URL for the server archive.

This value is populated after *get_version_for_target_spec()* or *prepare_download_assets()* (which calls the former) has successfully resolved the URL.

Returns
The complete download URL, or `None` if not yet resolved.

Return type
Optional[str]

`async extract_server_files`(*is_update: bool*)

Extracts server files asynchronously.

`async full_server_setup`(*is_update: bool*) → str

Performs the complete server setup asynchronously.

`async get_version_for_target_spec()` → str

Resolves the target version asynchronously.

`async prepare_download_assets()` → Tuple[str, str, str]

Prepares download assets asynchronously.

35.8 Bedrock Process Manager Core Documentation

`class` bedrock_server_manager.`BedrockProcessManager`(*settings: Any*, *storage: Any*, *server_provider: Any | None = None*, *api: Any | None = None*)

Bases: `object`

Manages Bedrock server processes, including monitoring and restarting.

The `BedrockProcessManager` runs a background monitoring thread that checks the status of registered servers at regular intervals. If a server is found to be stopped without being intentionally stopped (crashed), it attempts to restart it. It also periodically queries server status to update player counts and scan logs for player activity.

`servers`

A dictionary mapping server names to *BedrockServer* instances currently being managed.

Type
Dict[str, *BedrockServer*]

logger

Logger for the process manager.

Type
logging.Logger

app_context

The application context.

Type
AppContext

settings

The application settings.

Type
Settings

__init__(*settings: Any*, *storage: Any*, *server_provider: Any | None = None*, *api: Any | None = None*)

Initializes the BedrockProcessManager with explicit dependencies.

async start()

Call this from the main thread to start monitoring.

async add_server(*server:* BedrockServer)

Adds a server to be managed by the process manager.

Parameters
server (BedrockServer) – The server instance to monitor.

async remove_server(*server_name: str*)

Removes a server from the process manager.

This stops the manager from monitoring the server, but does not stop the server process itself.

Parameters
server_name (*str*) – The name of the server to remove.

async shutdown()

Shuts down all managed servers concurrently and stops the monitoring task.

This method: 1. Sets the shutdown event for the monitoring task and cancels it immediately. 2. Spawns tasks to stop all currently running servers concurrently with exception handling.

async write_error_status(*server_name: str*)

Writes 'ERROR' to server config status asynchronously.

Parameters
server_name (*str*) – The name of the server.

Raises
FileOperationError – If writing to the config file fails.

35.9 Resource Monitor Core Documentation

class bedrock_server_manager.core.system.base.**ResourceMonitor**(**args: Any*, ***kwargs: Any*)

Bases: object

A singleton class for monitoring process resource usage (CPU, memory, uptime).

This class provides a way to get resource statistics for a given `psutil.Process` object. It is implemented as a thread-safe singleton to ensure that the internal state required for calculating CPU percentage (which relies on comparing CPU times between calls) is maintained consistently across the application for each monitored process.

The monitor stores the last CPU times and timestamp on a per-PID basis to correctly calculate CPU utilization for individual processes.

Requires the `psutil` library. If `psutil` is not available (indicated by `PSUTIL_AVAILABLE` being `False`), methods like *get_stats()* will typically log a warning and return `None` or raise an error.

_instance

The single instance of this class.

Type
Optional[*ResourceMonitor*]

_lock

A lock to ensure thread-safe singleton creation and initialization.

Type
threading.Lock

_last_readings

A dictionary storing the last CPU times (`psutil.cpu_times` result) and timestamp for each monitored PID. Keyed by PID.

Type
Dict[int, Tuple[Any, float]]

_initialized

A flag to ensure `__init__` logic runs only once.

Type
bool

__init__() → None

Initializes the resource monitor's state.

This constructor is called only once for the singleton instance. It sets up the *_processes* dictionary to cache psutil.Process objects and an *_initialized* flag.

get_stats(*process: Process*) → Dict[str, Any] | None

Calculates and returns resource usage statistics for the given process.

If `psutil` is not available, this method logs a warning and returns `None`.

The CPU percentage is calculated using psutil's built-in cpu_percent.

Parameters
process (*psutil.Process*) – An instance of `psutil.Process` representing the process to monitor.

Returns

A dictionary containing the statistics if successful, or `None` if `psutil` is unavailable or the process is inaccessible (e.g., `psutil.NoSuchProcess`, `psutil.AccessDenied`). The dictionary structure is:

```
{
    "pid": int,           # Process ID
    "cpu_percent": float, # CPU usage percentage (e.g., 12.3)
    "memory_mb": float,   # Resident Set Size (RSS) memory in megabytes
    "uptime": str         # Process uptime formatted as "HH:MM:SS"
}
```

Return type
Optional[Dict[str, Any]]

Raises

- **TypeError** – If the input *process* is not a `psutil.Process` instance.
- ***SystemError*** – If an unexpected error occurs while fetching stats using `psutil` (other than `NoSuchProcess` or `AccessDenied`).

CHAPTER

THIRTYSIX

CORE DOCUMENTATION

> **Warning**
>
> The modules documented below are the internal engine of the Bedrock Server Manager. They are **not** part of the public API, are subject to change without notice, and **must not** be used directly by plugins. Functionality may be changed at any time.

This documentation is provided mostly for contributors working on the core application, or curious into its underlying functions not listed else where.

36.1 Error Class Documentation

Custom exception hierarchy for the bedrock_server_manager package.

Key Principles:

- A single base exception, *BSMError*, for all application errors.
- A clear hierarchy with logical categories (e.g., FileError, ServerError).
- Inheritance from standard Python exceptions (e.g., ValueError, FileNotFoundError) to allow for flexible and standard exception handling.

All custom exceptions inherit from *BSMError*.

exception bedrock_server_manager.error.**BSMError**

Bases: Exception

Base class for all custom exceptions in this project.

exception bedrock_server_manager.error.**FileError**

Bases: *BSMError*

Base for errors related to file or directory operations.

exception bedrock_server_manager.error.**ServerError**

Bases: *BSMError*

Base for errors related to managing the Bedrock server process.

exception bedrock_server_manager.error.**ConfigurationError**

Bases: *BSMError*

Base for errors related to configuration files or values.

exception bedrock_server_manager.error.**StorageError**

Bases: *BSMError*

Base for errors related to database persistence or storage transactions.

exception bedrock_server_manager.error.**SystemError**

Bases: *BSMError*

Base for errors related to interactions with the host operating system.

exception bedrock_server_manager.error.**NetworkError**

Bases: *BSMError*

Base for errors related to network connectivity.

exception bedrock_server_manager.error.**UserInputError**

Bases: *BSMError*, ValueError

Base for errors caused by invalid user input. Inherits from *ValueError* for broader compatibility.

exception bedrock_server_manager.error.**AppFileNotFoundError**(*path: str*, *description: str = 'Required file or directory'*)

Bases: *FileError*, FileNotFoundError

Raised when an essential application file or directory is not found. Inherits from *FileNotFoundError* for standard exception handling.

exception bedrock_server_manager.error.**FileOperationError**

Bases: *FileError*

Raised for general failures during file operations like copy, move, or delete.

exception bedrock_server_manager.error.**DownloadError**

Bases: *FileError*, OSError

Raised when downloading a file fails. Inherits from IOError.

exception bedrock_server_manager.error.**ExtractError**

Bases: *FileError*

Raised when extracting an archive (zip, etc.) fails.

exception bedrock_server_manager.error.**BackupRestoreError**

Bases: *FileOperationError*

Raised when a backup or restore operation fails.

exception bedrock_server_manager.error.**ServerProcessError**

Bases: *ServerError*

Base for errors in starting, stopping, or checking the server process.

exception bedrock_server_manager.error.**ServerStartError**

Bases: *ServerProcessError*

Raised when the server process fails to start.

exception bedrock_server_manager.error.**ServerStopError**

Bases: *ServerProcessError*

Raised when the server process fails to stop.

exception bedrock_server_manager.error.**ServerNotRunningError**

Bases: *ServerProcessError*

Raised when an operation requires the server to be running, but it's not.

exception bedrock_server_manager.error.**SendCommandError**

Bases: *ServerError*

Raised when sending a command to the server console fails.

exception bedrock_server_manager.error.**ConfigParseError**

Bases: *ConfigurationError*, ValueError

Raised when a configuration file is malformed or contains invalid values. Inherits from *ValueError* for standard exception handling.

exception bedrock_server_manager.error.**BlockedCommandError**

Bases: *ConfigParseError*

Raised when an attempt is made to send a command blocked by configuration.

exception bedrock_server_manager.error.**PermissionsError**

Bases: *SystemError*, PermissionError

Raised for OS-level file or directory permission errors. Inherits from *PermissionError* for standard exception handling.

exception bedrock_server_manager.error.**CommandNotFoundError**(*command_name: str*, *message='System command not found'*)

Bases: *SystemError*

Raised when a required system command (e.g., 'systemctl', 'unzip') is not found.

exception bedrock_server_manager.error.**InternetConnectivityError**

Bases: *NetworkError*

Raised when an operation requires internet access, but it's unavailable.

exception bedrock_server_manager.error.**MissingArgumentError**

Bases: *UserInputError*

Raised when a required function or command-line argument is missing.

exception bedrock_server_manager.error.**InvalidServerNameError**

Bases: *UserInputError*

Raised when a provided server name is invalid or contains illegal characters.

36.2 System Base Core Documentation

Provides base system utilities and foundational cross-platform functionalities.

This module contains a collection of essential system-level utilities that are generally platform-agnostic or provide a common interface for operations that might have platform-specific nuances handled elsewhere. It serves as a core part of the system interaction layer.

Key functionalities include:

- **Internet Connectivity:**
  - *check_internet_connectivity()*: Verifies basic internet access.

- **Filesystem Operations:**
  - *set_server_folder_permissions()*: Sets appropriate permissions for server directories, with platform-aware logic.
  - *delete_path_robustly()*: A utility for robustly deleting files or directories, handling potential read-only issues.
- **Process Information:**
  - *is_server_running()*: Checks if a Bedrock server process is active and verified, relying on *process*.
- **Resource Monitoring:**
  - *ResourceMonitor*: A singleton class for monitoring CPU and memory usage of processes, requiring the `psutil` library.

Constants:

- `PSUTIL_AVAILABLE`: Boolean indicating if `psutil` was imported, critical for *ResourceMonitor*.

Internal Helpers:

- `_handle_remove_readonly_onerror()`: An error handler for `shutil.rmtree`.

bedrock_server_manager.core.system.base.**can_manage_services**() → bool

Indicates if a system service manager (`systemctl` or `sc.exe`) is available.

If `True`, features related to managing system services (for the Web UI or game servers) can be expected to work.

async bedrock_server_manager.core.system.base.**check_internet_connectivity**(*url: str = 'http://clients3.google.com/generate_2*, *timeout: int = 3*) → None

Asynchronously checks for basic internet connectivity by attempting an HTTP GET request.

This function tries to establish an HTTP connection to a specified *url* with a given *timeout*. Success indicates likely internet access. Failure (timeout or other `ClientError`) raises an *InternetConnectivityError*.

Parameters

- **url** (*str, optional*) – The URL to connect to. Defaults to "http://clients3.google.com/generate_204".
- **timeout** (*int, optional*) – The connection timeout in seconds. Defaults to 3.

Raises

InternetConnectivityError – If the HTTP connection fails due to a timeout, network error, or any other unexpected exception during the check.

async bedrock_server_manager.core.system.base.**delete_path_robustly**(*path_to_delete: str*, *item_description: str*) → bool

Asynchronously deletes a file or directory robustly, attempting to handle read-only attributes.

This function attempts to delete the specified *path_to_delete*.

- If it's a directory, it uses *shutil.rmtree* (via *asyncio.to_thread*) with a custom error handler (`_handle_remove_readonly_onerror()`) that tries to make read-only files writable before retrying deletion.
- If it's a file, it first checks if it's writable. If not, it attempts to make it writable (`stat.S_IWRITE | stat.S_IWUSR`) before calling *aiofiles.os.remove*.

- If the path does not exist, it logs this and returns `True`.
- If the path is neither a file nor a directory, it logs a warning and returns `False`.

Deletion failures are logged, and the function returns `False` in such cases.

Parameters

- **path_to_delete** (*str*) – The absolute path to the file or directory to delete.
- **item_description** (*str*) – A human-readable description of the item being deleted, used for logging messages (e.g., "temporary file", "old backup directory").

Returns
`True` if the deletion was successful or if the path did not exist initially. `False` if an error occurred during deletion or if the path was neither a file nor a directory.

Return type
bool

Raises
MissingArgumentError – If *path_to_delete* or *item_description* are empty or not strings.

async bedrock_server_manager.core.system.base.**find_files**(*directory: str*, *pattern: str = '*'*, *sort_by: str = 'name'*, *reverse: bool = False*, *include_metadata: bool = False*) → List[str] | List[Dict[str, Any]]

Asynchronously finds files in a directory matching a pattern and returns paths or metadata.

Parameters

- **directory** (*str*) – The root directory to search in.
- **pattern** (*str*) – Glob pattern for matching files. Defaults to "*".
- **sort_by** (*str*) – Sorting criterion: "name", "mtime", "size". Defaults to "name".
- **reverse** (*bool*) – Whether to sort in reverse order. Defaults to False.
- **include_metadata** (*bool*) – If True, returns a list of dictionaries with metadata (path, name, size, mtime). If False, returns a list of string paths.

Returns
A list of paths or metadata dictionaries.

Return type
Union[List[str], List[Dict[str, Any]]]

async bedrock_server_manager.core.system.base.**is_server_running**(*server_name: str*, *server_dir: str*, *config_dir: str*) → bool

Asynchronously checks if a specific Bedrock server process is running and verified.

This acts as a high-level convenience wrapper around *get_verified_bedrock_process()*.

async bedrock_server_manager.core.system.base.**set_server_folder_permissions**(*server_dir: str*) → None

Asynchronously sets appropriate permissions for a Bedrock server installation directory.

This function adjusts permissions recursively for the specified *server_dir* based on the operating system:

- **On Linux**:
  - Ownership of all files and directories is set to the current effective user (UID) and group (GID) using `os.chown`.

  - Directories are set to `0o775` (rwxrwxr-x).
  - The main server executable (assumed to be named "bedrock_server") is set to `0o775` (rwxrwxr-x).
  - Other files are set to `0o664` (rw-rw-r–).
- **On Windows**:
  - Ensures that the "write" permission (`stat.S_IWRITE` or `stat.S_IWUSR`) is set for all files and directories. It preserves other existing permissions by ORing with the current mode.
- **Other OS**:
  - Logs a warning that permission setting is not implemented.

Parameters
server_dir (*str*) – The absolute path to the server's installation directory.

Raises
- ***MissingArgumentError*** – If *server_dir* is empty or not a string.
- ***AppFileNotFoundError*** – If *server_dir* does not exist or is not a directory.
- ***PermissionsError*** – If any `OSError` occurs during *os.chown* or *os.chmod* operations (e.g., due to insufficient privileges to change ownership or permissions), or for other unexpected errors.

36.3 System Process Core Documentation

Provides generic, cross-platform process management utilities.

This module abstracts common tasks related to system process interaction, aiming to provide a consistent interface regardless of the underlying OS (though some behaviors might be platform-specific, as noted in function docstrings). It is crucial for managing Bedrock server processes, launcher processes, and potentially other child processes.

The module heavily relies on the `psutil` library for many of its capabilities. Its availability is checked by the `PSUTIL_AVAILABLE` flag, and functions requiring `psutil` will typically raise a `SystemError` if it's not installed.

Key Functionality Groups:

- **PID File Management:**
  - *get_pid_file_path()*,
  - *get_bedrock_server_pid_file_path()*,
  - *get_bedrock_launcher_pid_file_path()*,
  - *read_pid_from_file()*,
  - *write_pid_to_file()*,
  - *remove_pid_file_if_exists()*.
- **Process Status & Verification:**
  - *is_process_running()*,
  - *verify_process_identity()*,
  - *get_verified_bedrock_process()*.
- **Process Creation & Termination:**
  - *launch_detached_process()* (using *GuardedProcess*),

  - *terminate_process_by_pid()*.

- **Guarded Subprocessing:**
  - *GuardedProcess*: A wrapper around `subprocess` to inject a recursion guard environment variable, preventing re-initialization loops when the application calls itself.

Constants:

- `PSUTIL_AVAILABLE`: Boolean indicating if `psutil` was imported.

class bedrock_server_manager.core.system.process.**GuardedProcess**(*command: Sequence[str | PathLike]*)

Wraps `subprocess` calls to inject a recursion guard environment variable.

When the application needs to launch a new instance of itself as a subprocess (e.g., to start a server in a detached background process via *launch_detached_process()*), this class is used to manage the subprocess creation.

Its primary purpose is to set a specific environment variable, defined by `GUARD_VARIABLE` (e.g., `BSM_GUARDED_EXECUTION="1"`), in the environment of the child process. The application's main entry point or initialization logic can then check for the presence of this variable. If detected, it can skip certain initialization steps (like reloading plugins, re-parsing arguments for the main CLI) that are unnecessary or problematic for a guarded, secondary instance. This helps prevent issues like duplicate plugin loading or unintended recursive behavior.

The class provides *run()* and *popen()* methods that mirror the standard `subprocess.run` and `subprocess.Popen` functions but ensure the guarded environment is passed to the child process.

command

The command and its arguments.

Type
List[Union[str, os.PathLike]]

guard_env

A copy of the current environment with the guard variable added.

Type
Dict[str, str]

async create_subprocess_exec(***kwargs: Any*) → Process

Wraps `asyncio.create_subprocess_exec`, injecting the guarded environment.

Parameters
****kwargs** (*Any*) – Keyword arguments to pass directly. If `env` is provided in `kwargs`, it will be overwritten.

Returns
The asynchronous process object.

Return type
asyncio.subprocess.Process

async create_subprocess_shell(***kwargs: Any*) → Process

Wraps `asyncio.create_subprocess_shell`, injecting the guarded environment.

Parameters
****kwargs** (*Any*) – Keyword arguments to pass directly. If `env` is provided in `kwargs`, it will be overwritten.

Returns
The asynchronous process object.

Return type
asyncio.subprocess.Process

popen(***kwargs: Any*) → Popen

Wraps `subprocess.Popen`, injecting the guarded environment.

This method calls `subprocess.Popen` with the stored `self.command` and passes along any additional `**kwargs`. The crucial difference is that it automatically sets the `env` keyword argument for `subprocess.Popen` to `self.guard_env`.

Parameters
****kwargs** (*Any*) – Keyword arguments to pass directly to `subprocess.Popen`. If `env` is provided in `kwargs`, it will be overwritten.

Returns
The `subprocess.Popen` object for the new process.

Return type
subprocess.Popen

run(***kwargs: Any*) → CompletedProcess

Wraps `subprocess.run`, injecting the guarded environment.

This method calls `subprocess.run` with the stored `self.command` and passes along any additional `**kwargs`. The crucial difference is that it automatically sets the `env` keyword argument for `subprocess.run` to `self.guard_env`.

Parameters
****kwargs** (*Any*) – Keyword arguments to pass directly to `subprocess.run`. If `env` is provided in `kwargs`, it will be overwritten.

Returns
The result of the `subprocess.run` call.

Return type
subprocess.CompletedProcess

async bedrock_server_manager.core.system.process.**get_bedrock_launcher_pid_file_path**(*server_name: str*, *config_dir: str*) → str

Asynchronously constructs the path for a Bedrock server's LAUNCHER process PID file.

This PID file is intended for the "launcher" or "wrapper" process that manages the actual Bedrock server (e.g., a process started by this application to run the server in the background or foreground with IPC). The PID file is typically named `bedrock_<server_name>_launcher.pid` and is placed directly in the provided *config_dir* (unlike the server's own PID file which might be in a subdirectory).

If *config_dir* does not exist, this function will attempt to create it asynchronously.

Parameters

- **server_name** (*str*) – The unique name of the server instance this launcher is associated with.
- **config_dir** (*str*) – The main configuration directory for the application. If it doesn't exist, an attempt will be made to create it.

Returns
The absolute path to where the launcher's PID file should be located.

Return type
str

Raises

- *MissingArgumentError* – If *server_name* or *config_dir* are empty or not strings.
- *AppFileNotFoundError* – If *config_dir* does not exist and cannot be created.

async bedrock_server_manager.core.system.process.**get_bedrock_server_pid_file_path**(*server_name: str*, *config_dir: str*) → str

Asynchronously constructs the standardized path to a Bedrock server's main process PID file.

This function generates a path for a PID file specific to a Bedrock server instance. The PID file is typically located in a subdirectory named after the *server_name* within the main *config_dir*. The filename itself is `bedrock_<server_name>.pid`.

Parameters

- **server_name** (*str*) – The unique name of the server instance.
- **config_dir** (*str*) – The main configuration directory for the application. This directory must exist.

Returns
The absolute path to where the server's PID file should be located.

Return type
str

Raises

- *MissingArgumentError* – If *server_name* or *config_dir* are empty or not strings.
- *AppFileNotFoundError* – If the *config_dir* does not exist, or if the derived server-specific config subdirectory (e.g., <*config_dir*>/<*server_name*>) does not exist.

async bedrock_server_manager.core.system.process.**get_pid_file_path**(*config_dir: str*, *pid_filename: str*) → str

Asynchronously constructs the full, absolute path for a generic PID file.

This utility function joins the provided configuration directory and PID filename to produce a standardized, absolute path for a PID file, verifying directory existence asynchronously.

Parameters

- **config_dir** (*str*) – The absolute path to the application's (or a specific component's) configuration directory where the PID file should reside.
- **pid_filename** (*str*) – The base name of the PID file (e.g., "web_server.pid", "my_process.pid").

Returns
The absolute path to where the PID file should be stored.

Return type
str

Raises

- *AppFileNotFoundError* – If *config_dir* is not provided, is not a string, or is not an existing directory.
- *MissingArgumentError* – If *pid_filename* is not provided or is an empty string.

async bedrock_server_manager.core.system.process.**get_verified_bedrock_process**(*server_name: str*, *server_dir: str*, *config_dir: str*) → Process | None

Asynchronously finds, verifies, and returns the Bedrock server process using its PID file.

This function utilizes the async variants of the core system checks to verify if a Bedrock server process is currently running and matches expectations.

async bedrock_server_manager.core.system.process.**is_process_running**(*pid: int*) → bool

Asynchronously checks if a process with the given PID is currently running.

This function relies on `psutil.pid_exists()` to determine if a process with the specified *pid* is active on the system, offloaded to a thread.

Parameters
pid (*int*) – The process ID to check.

Returns
`True` if a process with the given *pid* exists and is running, `False` otherwise.

Return type
bool

Raises

- *SystemError* – If the `psutil` library is not available.
- *MissingArgumentError* – If *pid* is not an integer.

async bedrock_server_manager.core.system.process.**launch_detached_process**(*command: List[str]*, *launcher_pid_file_path: str*) → int

Asynchronously launches a command as a detached background process and records its PID.

This function uses the *GuardedProcess* wrapper to execute the given *command* via `asyncio.create_subprocess_exec`. Standard input, output, and error streams of the new process are redirected to `subprocess.DEVNULL`.

The PID of the newly launched detached process is written to the file specified by *launcher_pid_file_path* using *write_pid_to_file()*.

Parameters

- **command** (*List[str]*) – The command and its arguments.
- **launcher_pid_file_path** (*str*) – The absolute path to the PID file.

Returns
The Process ID (PID) of the newly launched detached process.

Return type
int

async bedrock_server_manager.core.system.process.**read_pid_from_file**(*pid_file_path: str*) → int | None

Asynchronously reads a Process ID (PID) from the specified file.

This function attempts to open and read the contents of the file at *pid_file_path*. It expects the file to contain a single integer representing a PID.

Parameters
pid_file_path (*str*) – The absolute path to the PID file.

Returns
The PID as an integer if the file exists, is readable, and contains a valid integer. Returns `None` otherwise (e.g., file not found, permission denied, invalid content).

Return type
Optional[int]

Raises

- ***MissingArgumentError*** – If *pid_file_path* is not provided or is empty.
- ***FileOperationError*** – If there's an error reading or parsing the file content that is not handled (though most are caught and logged).

async bedrock_server_manager.core.system.process.**remove_pid_file_if_exists**(*pid_file_path: str*) → bool

Asynchronously removes the specified PID file if it exists, logging outcomes.

This function checks for the existence of a file at *pid_file_path*. If it exists, an attempt is made to delete it. Deletion failures due to `OSError` (e.g., permission issues) are logged as warnings but do not propagate the exception.

Parameters
pid_file_path (*str*) – The absolute path to the PID file to remove.

Returns
`True` if the file was successfully removed or if it did not exist initially. `False` if an `OSError` occurred during an attempted removal.

Return type
bool

Raises
MissingArgumentError – If *pid_file_path* is not provided or is empty.

async bedrock_server_manager.core.system.process.**terminate_process_by_pid**(*pid: int*, *terminate_timeout: int = 5*, *kill_timeout: int = 2*)

Attempts to gracefully terminate, then forcefully kill, a process by PID.

This function implements a two-stage termination strategy:

1. **Graceful Termination (SIGTERM)**: It first sends a SIGTERM signal (`process.terminate()`) to the process with the given *pid*. It then waits for *terminate_timeout* seconds for the process to exit cleanly.
2. **Forceful Kill (SIGKILL)**: If the process does not terminate within the *terminate_timeout*, a SIGKILL signal (`process.kill()`) is sent to forcibly stop it. It then waits for *kill_timeout* seconds.

This approach gives the target process a chance to shut down gracefully (e.g., save data, release resources) before resorting to a forceful kill.

Parameters

- **pid** (*int*) – The Process ID of the process to terminate.
- **terminate_timeout** (*int, optional*) – Seconds to wait for graceful termination after SIGTERM. Defaults to 5.
- **kill_timeout** (*int, optional*) – Seconds to wait for the process to die after SIGKILL. Defaults to 2.

Raises

- *SystemError* – If the `psutil` library is not available.
- *MissingArgumentError* – If *pid* is not an integer.
- *PermissionsError* – If `psutil` is denied access when trying to terminate the process.
- *ServerStopError* – For other `psutil` errors (e.g., process already exited before kill, unexpected errors) or if timeouts are invalid.

async bedrock_server_manager.core.system.process.**verify_process_identity**(*pid: int*, *expected_executable_path: str | None = None*, *expected_cwd: str | None = None*, *expected_command_args: str | List[str] | None = None*)

Verifies if a running process matches an expected signature.

This function checks a process, identified by its *pid*, against one or more provided criteria: its executable path, its current working directory (CWD), and/or specific command-line arguments. This is useful for confirming that a PID read from a file indeed corresponds to the expected application or server process, and not some other unrelated process that happens to have the same PID if the original process died and its PID was recycled.

It uses `psutil.Process(pid).oneshot()` for efficient information retrieval. Path comparisons (for executable and CWD) are case-insensitive and normalized.

Parameters

- **pid** (*int*) – The Process ID of the process to verify.
- **expected_executable_path** (*Optional[str], optional*) – The expected absolute path of the main executable. Defaults to `None`.
- **expected_cwd** (*Optional[str], optional*) – The expected absolute current working directory of the process. Defaults to `None`.
- **expected_command_args** (*Optional[Union[str, List[str]]], optional*) – A specific string or a list of strings that are expected to be present in the process's command-line arguments. Defaults to `None`.

Raises

- *SystemError* – If the `psutil` library is not available or if `psutil` encounters an internal error retrieving process information.
- *ServerProcessError* – If the process with the given *pid* does not exist, or if it exists but does not match one or more of the provided expected criteria (mismatch details are included in the error message).
- *PermissionsError* – If `psutil` is denied access when trying to get information for the specified *pid*.

- *MissingArgumentError* – If *pid* is not an integer or if none of the optional verification criteria (*expected_executable_path*, *expected_cwd*, *expected_command_args*) are provided.

async bedrock_server_manager.core.system.process.**write_pid_to_file**(*pid_file_path: str*, *pid: int*)

Asynchronously writes a Process ID (PID) to the specified file.

This function ensures the directory containing the *pid_file_path* exists (creating it if necessary) and then writes the *pid* to the file as a string.

Parameters

- **pid_file_path** (*str*) – The absolute path to the file where the PID should be written.
- **pid** (*int*) – The Process ID to write.

Raises

- *MissingArgumentError* – If *pid_file_path* is empty or not a string, or if *pid* is not an integer.
- *FileOperationError* – If there is an error creating the directory or writing to the file (e.g., permission issues).

36.4 Miscellaneous Core Documentation

async bedrock_server_manager.core.downloader.**prune_old_downloads**(*download_dir: str*, *download_keep: int*)

Asynchronously removes the oldest downloaded server ZIP files from a directory.

This function keeps a specified number of the most recent downloads and deletes the rest to manage disk space using native async file operations.

Parameters

- **download_dir** – The directory containing the downloaded `bedrock-server-*.zip` files.
- **download_keep** – The number of most recent ZIP files to retain.

Raises

- *MissingArgumentError* – If *download_dir* is not provided.
- *UserInputError* – If *download_keep* is not a non-negative integer.
- *AppFileNotFoundError* – If *download_dir* does not exist.
- *FileOperationError* – If there's an error accessing or deleting files.

36.5 Platform Dependent Core Documentation

36.5.1 System Linux Core Documentation

Provides Linux-specific implementations for system service management.

This module is tailored for Linux environments and focuses on two main areas:

1. **Systemd User Service Management**: Functions for creating, enabling, disabling, and checking the existence of systemd user services. These are typically used to manage Bedrock servers as background daemons that can start on user login. Operations usually involve interaction with `systemctl --user`.

Key Functionality Groups:

- **Systemd User Service Utilities (Linux-specific):**

- get_systemd_user_service_file_path()
- *check_service_exists()*
- *create_systemd_service_file()*
- *enable_systemd_service()*
- *disable_systemd_service()*

Global State:

- `_foreground_server_shutdown_event`: A threading event for managing the lifecycle of foreground server processes.

> **Note**
>
> Most functions in this module will perform a platform check and may return early or behave differently if not run on a Linux system.

bedrock_server_manager.core.system.linux.**check_service_exists**(*service_name_full: str*, *system: bool = False*) → bool

Checks if a systemd user service file exists on Linux.

This function determines if a service is defined by checking for the presence of its service unit file in the standard systemd user directory (obtained via *get_systemd_service_file_path()*).

Parameters

- **service_name_full** (*str*) – The full name of the service unit to check (e.g., "my-app.service" or "my-app").
- **system** (*bool, optional*) – If `True`, checks for a system-wide service. Defaults to `False`.

Returns

`True` if the service file exists, `False` otherwise. Returns `False` if the current operating system is not Linux.

Return type

bool

Raises

MissingArgumentError – If *service_name_full* is empty or not a string.

bedrock_server_manager.core.system.linux.**create_systemd_service_file**(*service_name_full: str*, *description: str*, *working_directory: str*, *exec_start_command: str*, *exec_stop_command: str | None = None*, *exec_start_pre_command: str | None = None*, *service_type: str = 'forking'*, *restart_policy: str = 'on-failure'*, *restart_sec: int = 10*, *after_targets: str = 'network.target'*, *system: bool = False*) → None

Creates or updates a systemd user service file on Linux and reloads the daemon.

This function generates a systemd service unit file with the specified parameters and places it in the user's systemd directory (typically `~/.config/systemd/user/`). After writing the file, it executes `systemctl --user daemon-reload` to ensure systemd recognizes any changes.

If the function is called on a non-Linux system, it logs a warning and returns.

Parameters

- **service_name_full** (*str*) – The full name for the service unit file (e.g., "my-app.service" or "my-app", ".service" suffix is optional).
- **description** (*str*) – A human-readable description for the service. Used for the `Description=` field in the unit file.
- **working_directory** (*str*) – The absolute path to the working directory for the service process. Used for `WorkingDirectory=`.
- **exec_start_command** (*str*) – The command (with arguments) to execute when the service starts. Used for `ExecStart=`.
- **exec_stop_command** (*Optional[str], optional*) – The command to execute when the service stops. Used for `ExecStop=`. Defaults to `None`.
- **exec_start_pre_command** (*Optional[str], optional*) – A command to execute before the main `ExecStart` command. Used for `ExecStartPre=`. Defaults to `None`.
- **service_type** (*str, optional*) – The systemd service type (e.g., "simple", "forking", "oneshot"). Used for `Type=`. Defaults to "forking".
- **restart_policy** (*str, optional*) – The systemd `Restart=` policy (e.g., "no", "on-success", "on-failure", "always"). Defaults to "on-failure".
- **restart_sec** (*int, optional*) – Time in seconds to wait before restarting the service if *restart_policy* is active. Used for `RestartSec=`. Defaults to 10.
- **after_targets** (*str, optional*) – Specifies other systemd units that this service should start after. Used for `After=`. Defaults to "network.target".
- **system** (*bool, optional*) – If `True`, creates a system-wide service. Defaults to `False`.

Raises

- *MissingArgumentError* – If any of *service_name_full*, *description*, *working_directory*, or *exec_start_command* are empty or not strings.
- *AppFileNotFoundError* – If the specified *working_directory* does not exist or is not a directory.
- *FileOperationError* – If creating the systemd user directory or writing the service file fails (e.g., due to permissions).
- *CommandNotFoundError* – If the `systemctl` command is not found in the system's PATH.
- *SystemError* – If `systemctl --user daemon-reload` fails.

bedrock_server_manager.core.system.linux.**disable_systemd_service**(*service_name_full: str*, *system: bool = False*) → None

Disables a systemd user service on Linux from starting on user login.

This function uses `systemctl --user disable <service_name>` to disable the specified service. It first checks if the service file exists and if the service is already disabled (or not enabled) to avoid errors or redundant operations. Static or masked services cannot be disabled this way and will be logged accordingly.

If the function is called on a non-Linux system, it returns early.

Parameters

- **service_name_full** (*str*) – The full name of the service unit to disable (e.g., "my-app.service" or "my-app").
- **system** (*bool, optional*) – If `True`, disables a system-wide service. Defaults to `False`.

Raises

- ***MissingArgumentError*** – If *service_name_full* is empty or not a string.
- ***CommandNotFoundError*** – If the `systemctl` command is not found.
- ***SystemError*** – If the `systemctl disable` command fails for reasons other than the service being static or masked.

bedrock_server_manager.core.system.linux.**enable_systemd_service**(*service_name_full: str*, *system: bool = False*) → None

Enables a systemd user service on Linux to start on user login.

This function uses `systemctl --user enable <service_name>` to enable the specified service. It first checks if the service file exists and if the service is already enabled to avoid redundant operations.

If the function is called on a non-Linux system, it returns early.

Parameters

- **service_name_full** (*str*) – The full name of the service unit to enable (e.g., "my-app.service" or "my-app").
- **system** (*bool, optional*) – If `True`, enables a system-wide service. Defaults to `False`.

Raises

- ***MissingArgumentError*** – If *service_name_full* is empty or not a string.
- ***CommandNotFoundError*** – If the `systemctl` command is not found.
- ***SystemError*** – If the service unit file (checked by *check_service_exists()*) does not exist, or if the `systemctl enable` command fails.

bedrock_server_manager.core.system.linux.**get_systemd_service_file_path**(*service_name_full: str*, *system: bool = False*) → str

Generates the standard path for a systemd service file on Linux.

Systemd user service files are typically located in the user's `~/.config/systemd/user/` directory. This function constructs that path. If the provided *service_name_full* does not end with ".service", the suffix is automatically appended.

Parameters

- **service_name_full** (*str*) – The full name of the service unit. It can be provided with or without the `.service` suffix (e.g., "my-app.service" or "my-app").
- **system** (*bool, optional*) – If `True`, returns the path for a system-wide service. Defaults to `False`.

Returns

The absolute path to where the systemd user service file should be located (e.g., "/home/user/.config/systemd/user/my-app.service").

Return type
str

Raises
MissingArgumentError – If *service_name_full* is empty or not a string.

36.5.2 System Windows Core Documentation

Provides Windows-specific implementations for system service management.

This module offers functionalities tailored for the Windows operating system, primarily focused on managing Windows Services. It leverages the `pywin32` package for many of its operations, and its availability is checked by the `PYWIN32_AVAILABLE` flag. Some optional `pywin32` modules for cleanup are checked via `PYWIN32_HAS_OPTIONAL_MODULES`.

Key functionalities include:

Windows Service Management (Requires Administrator Privileges):

- Checking if a service exists (*check_service_exists()*).
- Creating or updating a Windows Service to run the Bedrock server (*create_windows_service()*).
- Enabling (*enable_windows_service()*) or disabling (*disable_windows_service()*) a service.
- Deleting a service, including cleanup of associated registry entries like performance counters and event log sources (*delete_windows_service()*).

Note

Functions interacting with the Windows Service Control Manager (SCM) typically require Administrator privileges to execute successfully. The module attempts to handle *PermissionsError* where appropriate.

bedrock_server_manager.core.system.windows.**check_service_exists**(*service_name: str*) → bool

Checks if a Windows service with the given name exists.

Requires Administrator privileges for a definitive check, otherwise, it might return `False` due to access denied errors.

Parameters
service_name (*str*) – The short name of the service to check (e.g., "MyBedrockServer").

Returns
`True` if the service exists, `False` otherwise. Returns `False` if `pywin32` is not installed or if access is denied (which is logged as a warning).

Return type
bool

Raises
- *MissingArgumentError* – If *service_name* is empty.
- *SystemError* – For unexpected Service Control Manager (SCM) errors other than "service does not exist" or "access denied".

bedrock_server_manager.core.system.windows.**create_windows_service**(*service_name: str*, *display_name: str*, *description: str*, *command: str*, *username: str | None = None*, *password: str | None = None*) → None

Creates a new Windows service or updates an existing one.

This function interacts with the Windows Service Control Manager (SCM) to either create a new service or modify the configuration of an existing service (specifically its display name, description, and executable command). The service is configured for automatic start (`SERVICE_AUTO_START`) and runs as `LocalSystem` by default upon creation.

Requires Administrator privileges.

Parameters

- **service_name** (*str*) – The short, unique name for the service (e.g., "MyBedrockServerSvc").
- **display_name** (*str*) – The user-friendly name shown in the Services console (e.g., "My Bedrock Server").
- **description** (*str*) – A detailed description of the service.
- **command** (*str*) – The full command line to execute for the service, including the path to the executable and any arguments. Example: `"C:\path\to\python.exe C:\path\to\script.py --run-as-service"`
- **username** (*Optional[str], optional*) – The username to run the service as. If `None`, the service runs as `LocalSystem`. Defaults to `None`.
- **password** (*Optional[str], optional*) – The password for the user. Required if *username* is provided. Defaults to `None`.

Raises

- ***SystemError*** – If `pywin32` is not available, or for other unexpected SCM errors during service creation/update.
- ***MissingArgumentError*** – If any of the required string arguments (*service_name*, *display_name*, *description*, *command*) are empty.
- ***PermissionsError*** – If the operation fails due to insufficient privileges (typically requires Administrator rights).

bedrock_server_manager.core.system.windows.**delete_windows_service**(*service_name: str*) → None

Deletes a Windows service and performs associated cleanup.

This function interacts with the Windows Service Control Manager (SCM) to delete the specified service. It also attempts to perform cleanup operations such as unloading performance counters and removing event log sources from the registry, provided that optional `pywin32` modules (like `perfmon` and `win32evtlogutil`) are available (checked by `PYWIN32_HAS_OPTIONAL_MODULES`).

The service should ideally be stopped before deletion, though this function does not explicitly stop it. If the service does not exist or is already marked for deletion, warnings are logged, and the function may proceed with cleanup or return gracefully.

Requires Administrator privileges.

Parameters

service_name (*str*) – The short name of the service to delete.

Raises

- ***SystemError*** – If `pywin32` is not available or for critical SCM errors during deletion (e.g., service cannot be deleted for reasons other than access denied, not existing, or already marked for deletion).

- *MissingArgumentError* – If *service_name* is empty.
- *PermissionsError* – If the operation fails due to insufficient privileges.

bedrock_server_manager.core.system.windows.**disable_windows_service**(*service_name: str*) → None

Disables a Windows service by setting its start type to 'Disabled'.

This function interacts with the Windows Service Control Manager (SCM) to change the start type of the specified service to SERVICE_DISABLED. If the service does not exist, a warning is logged, and the function returns gracefully.

Requires Administrator privileges.

Parameters
service_name (*str*) – The short name of the service to disable.

Raises

- *SystemError* – If pywin32 is not available or for unexpected SCM errors (other than service not existing).
- *MissingArgumentError* – If *service_name* is empty.
- *PermissionsError* – If the operation fails due to insufficient privileges.

bedrock_server_manager.core.system.windows.**enable_windows_service**(*service_name: str*) → None

Enables a Windows service by setting its start type to 'Automatic'.

This function interacts with the Windows Service Control Manager (SCM) to change the start type of the specified service to SERVICE_AUTO_START.

Requires Administrator privileges.

Parameters
service_name (*str*) – The short name of the service to enable.

Raises

- *SystemError* – If pywin32 is not available, if the service does not exist, or for other unexpected SCM errors.
- *MissingArgumentError* – If *service_name* is empty.
- *PermissionsError* – If the operation fails due to insufficient privileges.

Source Code: Github

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

T

U

V

W

Z