minecraft-launcher-lib

Release 8.0

JakobDev

Oct 30, 2025

CONTENTS

minecraft-launcher-lib is a easy to use Python library for creating your own Minecraft Launcher.

CHAPTER

ONE

INTRODUCTION

1.1 What's this?

minecraft-launcher-lib is (as the Name might suggest) a Python library for creating a custom Minecraft launcher. It allows you to easily install and launch Minecraft without needing to know technical details. It also included functions for some optional things you may want to have e.g. Installing modloaders like Forge/Fabric/Quilt or installing Modpacks. Many different things are included, so you can write a Launcher that fit's exactly your needs.

- You want a simple script that just launches the latest Minecraft version? No problem!
- You want to play a Modpack with your friends, but they have problems installing it? Just create a custom launcher using minecraft-launcher-lib that installs Minecraft together with the Modpack and automatically connects to your server.
- You want to create a branded Launcher for your Modpack, to make sure it everyone can install it? With minecraft-launcher-lib you can do this easily!

1.2 Goals

These are the main goals of minecraft-launcher-lib:

- minecraft-launcher-lib can launch any version out there. If the official Launcher can launch it, minecraft-launcher-lib can launch it too!
- minecraft-launcher-lib installs and launches the Game, but never patches it.
- minecraft-launcher-lib works on every Operating System and architecture that is supported by Minecraft.
- minecraft-launcher-lib is written in pure. It only depends on requests for the Network. No big dependency tree is used.
- minecraft-launcher-lib should stay backwards compatible. This is not always possible because of changes on the side of Mojang/Microsoft e.g. the switch from Mojang to Microsoft accounts, but if minecraft-launcher-lib is not forced, it should never break backwards compatibility. A program that is written using an older version should also work on the latest version without changes. To ensure this, minecraft-launcher-lib has a test coverage of over 95%.
- minecraft-launcher-lib is fully static typed. It helps you develop with an IDE and you can use type checkers like mypy.

1.3 Getting Started

This documentation contains a tutorial. You should start with the *Getting Started* tutorial. You can also take a look at the *Modules* documentation which contains the full public API. There are also *Examples* which shows how to use

minecraft-launcher-lib in real code. If you want to see full programs which are using minecraft-launcher-lib, you can visit the *Showcase*.

CHAPTER

TWO

INSTALLATION

Here are different methods to install minecraft-launcher-lib.

- **pip**

  Like almost every other Python library, minecraft-launcher-lib can be installed directly from PyPI. That is the preferred way to install it for the most people.

  ```
  pip install -U minecraft-launcher-lib
  ```

- **AUR**

  If you use Arch Linux or a Arch based Distro like Manjaro, you may want to install it from the Arch User Repository.

- **From Source**

  You can also install minecraft-launcher-lib directly from source. That will give you the latest changes that are not in a release yet. Please do this only if you have a good reason.

  ```
  pip install -U git+https://codeberg.org/JakobDev/minecraft-launcher-lib.git
  ```

CHAPTER

THREE

TUTORIAL

3.1 Getting Started

This first chapter of the documentation shows how to install and run Minecraft. The login with Microsoft is skipped here.

3.1.1 Minecraft Directory

To get started with minecraft-launcher-lib, you need a Minecraft Directory first. You can use a new directory or the default Directory of Minecraft. You can get the default Directory with *get_minecraft_directory()*.

```
# Get the Minecraft Directory of your System
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
# Or use your own
minecraft_directory = "path/to/your/minecraft/directory"
```

3.1.2 Install Minecraft

Before you can launch launch Minecraft, you need to install it. This can be done by using install_minecraft_version. Let's say we want to install Version 1.17 in our Minecraft Directory.

```
minecraft_launcher_lib.install.install_minecraft_version("1.17", minecraft_directory)
```

To get the information how to install Minecraft, minecraft-launcher-lib looks first for a JSON file in your Minecraft Directory. In the case of 1.17 it's `minecraft_directory/versions/1.17/1.17.json`. This allows installing moded versions that are not official from Mojang. If the JSON file not does exists minecraft-launcher-lib tries to download it from the Mojang Servers. install_minecraft_version ensures that the Minecraft installation is correct, so you need to call it every time before you launch Minecraft, even if you had the version already installed.

3.1.3 Get Minecraft Versions

If you don't want to start a single version like 1.17 every time you need a list of all Minecraft version. To get that list use `minecraft_launcher_lib.utils.get_available_versions(minecraft_directory)`. It returns this list:

```
[
    {
        "id": "some_id",
        "type": "release"
    },
    {
        "id": "some_other_id",
```

(continues on next page)

(continued from previous page)

```
        "type": "snapshot"
    }
]
```

The id is the Minecraft version that can be used as argument for install_minecraft_version and other versions. The type says what type the version is. Possible values are currently: `release`, `snapshot`, `beta`, `alpha`. Moded Versions can also use a custom value, so don't rely on this list.

To get the latest version, use *get_latest_version()*.

```
latest_release = minecraft_launcher_lib.utils.get_latest_version()["release"]
latest_snapshot = minecraft_launcher_lib.utils.get_latest_version()["snapshot"]
```

3.1.4 Launch Minecraft

Since you know how to install Minecraft, it's now time to start it. First we need a dict with all options. The minimal options dict is this:

```
{
    "username": "The Username",
    "uuid": "The UUID",
    "token": "The acces token"
}
```

The Username and UUID belongs to a Account. Since Name and UUID are public, the Token is used to log in. The token is generated every time when a User logs in with his Microsoft Account. Minecraft can be launched with a not existing user and a wrong token. This can be used for test cases. minecraft-launcher-lib allows creating a dict with a test user.

```
options = minecraft_launcher_lib.utils.generate_test_options()
```

We use the test options here to keep it simple. The login with Microsoft comes latter. Keep in mind that publishing a Launcher which allows User who haven't bought Minecraft to play is illegal, so use this only for test cases in development. You can add more options to the dict like the resolution, but this is not needed to launch.

Now we have the options, we need to get the Minecraft command. In this case for Version 1.17.

```
minecraft_command = minecraft_launcher_lib.command.get_minecraft_command("1.17",␣
↪minecraft_directory, options)
```

The command that your get is a list of strings that can be used to run Minecraft e.g. with the subprocess module.

3.2 Microsoft Login

Login with a Microsoft Account requires a Web browser and a Azure Application.

3.2.1 Create Azure Application

To login with Microsoft you need to create a Azure Application first. Follow this tutorial to create one. You need the Client ID, the Secret and the redirect URL of your new Application.

3.2.2 Apply for Permission

As stated here, new created Azure Apps need to apply for Permission using this Form before they can use the Minecraft API. Apps that have been created before this change keeps working without a chance. *complete_login()* will raise a *AzureAppNotPermitted* Exception if your App don't have the Permission to use the Minecraft API. If you get any other Exception, that probably means something else with your Azure App is not right.

3.2.3 Let the User log in

The login happens in a Web browser. This can be the normal Browser of the System or a Browser Widget embed in your Program. To get the url that is used for the login use minecraft_launcher_lib.microsoft_account.get_login_url(client_id: str, redirect_uri: str). Open the URL and test if you can login. After you've logged in you will be redirected to `https://<your redirect URL>?code=codegoeshere&state=<optional`. `codegoeshere` is the code that you need. You can use `minecraft_launcher_lib.microsoft_account.get_auth_code_from_url(url: str)` to get the code from the url. You can also use `minecraft_launcher_lib.microsoft_account.url_contains_auth_code(url: str)` to check if the given URL has a code.

3.2.4 Secure option

The `minecraft_launcher_lib.microsoft_account.get_secure_login_data(client_id: str, redirect_uri: str, state: str = _generate_state())` generates the login data for a secure login with pkce and state to prevent Cross-Site Request Forgery attacks and authorization code injection attacks. This is the recommended way to login. You can parse the auth code and verify the state with `minecraft_launcher_lib.microsoft_account.parse_auth_code_url(url: str, state: str)`

3.2.5 Do the Login

Use `minecraft_launcher_lib.microsoft_account.complete_login(client_id: str, redirect_uri: str, auth_code: str, code_verifier: Optional[str])` to login to Minecraft. The auth code is the code from URL you've got in the previous step. The code verifier is the code verifier you've got if you used the secure login method. You get this result:

```
{
    "id" : "The uuid",
    "name" : "The username",
    "access_token": "The access token",
    "refresh_token": "The refresh token",
    "skins" : [{
        "id" : "6a6e65e5-76dd-4c3c-a625-162924514568",
        "state" : "ACTIVE",
        "url" : "http://textures.minecraft.net/texture/
↪1a4af718455d4aab528e7a61f86fa25e6a369d1768dcb13f7df319a713eb810b",
        "variant" : "CLASSIC",
        "alias" : "STEVE"
    } ],
    "capes" : []
}
```

As you can see it contains everything you need for the options dict of *get_minecraft_command()*.

3.2.6 Refresh

To refresh just use `minecraft_launcher_lib.microsoft_account.complete_refresh(client_id: str, refresh_token: str)`. The refresh token is from the function above. If the refresh fails, it will throw a InvalidRefreshToken exception. In this case you need the user to login again.

3.3 Get Installation Progress

Installing a new Minecraft version can, depending on the internet connection, take some time. It would be nice to show the user the progress e.g. in a Progressbar.

To tell your program the current progress, minecraft-launcher-lib uses callbacks. Callbacks are just normal functions that you write and that are called by minecraft-launcher-lib. Here is a example:

```
import minecraft_launcher_lib


current_max = 0


def set_status(status: str):
    print(status)


def set_progress(progress: int):
    if current_max != 0:
        print(f"{progress}/{current_max}")


def set_max(new_max: int):
    global current_max
    current_max = new_max


minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()

callback = {
    "setStatus": set_status,
    "setProgress": set_progress,
    "setMax": set_max
}

minecraft_launcher_lib.install.install_minecraft_version("1.17", minecraft_directory,␣
↪callback=callback)
```

As you can see callback is a dict with functions. The functions are defined by you. You can write in these functions whatever you want. In the example above it prints the current status to the commandline.

3.4 More Launch Options

minecraft-launcher-lib offers various options for launching Minecraft. This page shows the most important ones. For a full list check out the documentation of the *command* module.

3.4.1 JVM Arguments

JVM Arguments are a list of strings. Each argument is a entry in the list. Here is a example:

```
# Right
options["jvmArguments"] = ["-Xmx2G", "-Xms2G"]

# Wrong
options["jvmArguments"] = ["-Xmx2G -Xms2G"]

# Wrong
options["jvmArguments"] = "-Xmx2G -Xms2G"
```

Make sure every argument starts with a `-`, otherwise Minecraft will not start with a `Could not find or load main class` error.

3.4.2 Java Executable

The Java Executable is the path the Java which is used to run Minecraft. If the client.json contains a Java Runtime, it minecraft-launcher-lib will download and use these version. Otherwise it will just use the java command. minecraft-launcher-lib allows to overwrite this. This can be useful, if you want to start a older version which needs a older Java and does not contain a runtime in the `client.json`.

There are 2 options to overwrite the Java Executable: `executablePath` and `defaultExecutablePath`. The difference is, that `executablePath` is always used. `defaultExecutablePath` is only used, when the `client.json` has set no Java Runtime. If the client.json contains a Runtime, the Runtime will be preferred over the defaultExecutablePath.

```
options["executablePath"] = "path/to/java"
options["defaultExecutablePath"] = "path/to/java"
```

3.4.3 Custom Resolution

minecraft-launcher-lib allows starting Minecraft with a custom resolution. The first thing you have to do is enable the custom resolution. After that you can set it:

```
# Enable custom resolution
options["customResolution"] = True
# Set custom resolution
options["resolutionWidth"] = "600"
options["resolutionHeight"] = "500"
```

Make sure you use strings and not int for the resolution.

3.4.4 Game Directory

The Game Directory is the directory where Minecraft saves all his stuff like Worlds, Resourcepacks, Options etc. By default your Minecraft Directory is used as Game Directory.

```
options["gameDirectory"] = "path/to/your/game/directory"
```

If the directory does not exists, Minecraft will create it.

3.4.5 Use Demo Mode

Minecraft has a build-in Demo mode, which is used, if somebody who does not bought Minecraft launches the Game through the official launcher. minecraft-launcher-lib allows you to enable the Demo mode. You need at least Minecraft version 1.3.1 to use the Demo mode.

```
options["demo"] = True
```

3.5 Install a mod loader

minecraft-launcher-lib allows you to install a mod loader for Minecraft using the *mod_loader* module. Currently the following loader are supported:

- Forge
- NeoForge
- Fabric
- Quilt

To install a mod loader, minecraft-launcher-lib provides the mod loader module, which has a unified way of installing.

Fist you need to get the loader by it's id. Valid ID's are `forge`, `neoforge`, `fabric` and `quilt`.

In this exmaple, we want to install fabric. So let's get fabric.

```
fabric = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
```

Now we can install fabric for version 1.21.

```
fabric.install("1.21", minecraft_directory)
```

You can also take a look at the *complete example*.

3.6 Custom Types

You will see types like e.g MinecraftOptions or MinecraftVersionInfo. They are defined in *types* and *microsoft_types*. They are all normal Dicts. Let's take a look at MinecraftVersionInfo:

```
class MinecraftVersionInfo(TypedDict):
    id: str
    type: str
    releaseTime: datetime.datetime
```

It means the following: This function returns a Dict with these keys:

- id: A string
- type: A string
- releaseTime: A datetime.datetime

This type definition is just there to help your IDE. The function itself returns just a normal Dict.

```
version_list = minecraft_launcher_lib.utils.get_version_list()
print(version_list[0]["id"])
```

(continues on next page)

(continued from previous page)

```
print(type(version_list[0]))
# <class 'dict'>
```

As said above, it is there to help your IDE. When the function definition just say, that it returns a Dict, your IDE will not know what the Dict contains. But when using a TypedDict, your IDE, will exactly know what the Dict contains and can offer your better autocompletion and a better type checking.

You can even use it when you are calling a function:

```
import minecraft_launcher_lib

options: minecraft_launcher_lib.types.MinecraftOptions = {}
options["username"] = "Test123"
```

When using a IDE, you will see that it will start autocompleting the keys of the Dict while writing.

For more information about TypedDict see PEP 589.

3.7 Getting Help

Here are some sources if you need more information.

- This documentation contains under modules a list of modules with every function you can use. You should check it out. You can test the functions in a interactive Python Shell.
- Check out the Examples on Codeberg.
- You can check out the source of the programs in the *Showcase*.
- If none of them above help, feel free to open a Issue.

CHAPTER

FOUR

MODULES

4.1 command

command contains the function for creating the minecraft command

get_minecraft_command(*version: str*, *minecraft_directory: str | PathLike*, *options:* MinecraftOptions) → list[str]

Returns the command to run Minecraft as a list. The returned command can be executed using Python's subprocess module.

You must provide a dictionary containing launch options. Some options are required and others are optional. The following options are available:

```
options = {
    # This is needed
    "username": "", # The Username,
    "uuid": "", # uuid of the user,
    "token": "", # the accessToken,
    # This is optional
    "executablePath": "java", # The path to the java executable
    "defaultExecutablePath": "java", # The path to the java executable if the␣
↪client.json has none
    "jvmArguments": [], # The jvmArguments
    "launcherName": "minecraft-launcher-lib", # The name of your launcher
    "launcherVersion": "1.0", # The version of your launcher
    "gameDirectory": "/home/user/.minecraft", # The gameDirectory (default is the␣
↪path given in arguments)
    "demo": False, # Run Minecraft in demo mode
    "customResolution": False, # Enable custom resolution
    "resolutionWidth": "854", # The resolution width
    "resolutionHeight": "480", # The resolution height
    "server": "example.com", # The IP of a server where Minecraft connect to after␣
↪start
    "port": "123", # The port of a server where Minecraft connect to after start
    "nativesDirectory": "minecraft_directory/versions/version/natives", # The␣
↪natives directory
    "enableLoggingConfig": False, # Enable use of the log4j configuration file
    "disableMultiplayer": False, # Disables the multiplayer
    "disableChat": False, # Disables the chat
    "quickPlayPath": None, # The Quick Play Path
    "quickPlaySingleplayer": None, # The Quick Play Singleplayer
    "quickPlayMultiplayer": None, # The Quick Play Multiplayer
```

(continues on next page)

(continued from previous page)

```
    "quickPlayRealms": None, # The Quick Play Realms
}
```

Use the *microsoft_account* module to obtain account-related information. For details about available options, see the *More Launch Options* tutorial.

To test this function without logging in, use *generate_test_options()* to create sample launch options. Only use those test options for development and testing until you implement proper account handling

Example:

```
options = minecraft_launcher_lib.utils.generate_test_options()
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
command = minecraft_launcher_lib.command.get_minecraft_command("1.21", minecraft_
↪directory, options)
subprocess.run(command, pwd=minecraft_directory)
```

Parameters

- **version** (*str*) – The Minecraft version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **options** (MinecraftOptions) – Some Options

Raises

VersionNotFound – The Minecraft version was not found

Returns

The command

Return type

list[str]

View the source code of this module

4.2 install

install allows you to install minecraft.

install_minecraft_version(*version: str*, *minecraft_directory: str | PathLike*, *callback:* CallbackDict | *None* = *None*) → None

Installs a Minecraft version to the specified path. It also verifies and repairs an existing installation, so call it before launching. Only missing or corrupted files will be downloaded.

You can pass a dict with functions as `callback` parameter to monitor the progress:

```
callback = {
    "setStatus": some_function, # This function is called to set a text
    "setProgress" some_function, # This function is called to set the progress.
    "setMax": some_function, # This function is called to set to max progress.
}
```

For more details, check the *corresponding tutorial*.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.install.install_minecraft_version("1.21", minecraft_
↪directory)
```

Parameters

- **version** (*str*) – The Minecraft version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **callback** (CallbackDict | *None*) – Some functions that are called to monitor the progress

Raises

- ***VersionNotFound*** – The Minecraft version was not found
- ***FileOutsideMinecraftDirectory*** – A File should be placed outside the given Minecraft directory

Return type
None

View the source code of this module

4.3 natives

natives contains a function for extracting natives libraries to a specific folder

extract_natives(*version: str*, *minecraft_directory: str | PathLike*, *extract_path: str | PathLike*) → None

Extract all native libraries from a version into the given directory. The directory will be created, if it does not exist.

The natives are all extracted while installing. So you don't need to use this function in most cases.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.natives.extract_natives("1.21", minecraft_directory, "/path/
↪to/extract")
```

Parameters

- **version** (*str*) – The Minecraft version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **extract_path** (*str | PathLike*) – The directory to which the native libraries will be extracted

Raises

- ***VersionNotFound*** – The Minecraft version was not found
- ***FileOutsideMinecraftDirectory*** – A File should be placed outside the given Minecraft directory

Return type
None

View the source code of this module

4.4 microsoft_account

microsoft_account contains functions for login with a Microsoft Account. Before using this module you need to create a Azure application. Many thanks to wiki.vg for it's documentation of the login process. You may want to read the *Microsoft Login* tutorial before using this module. For a list of all types see *microsoft_types*.

get_login_url(*client_id: str*, *redirect_uri: str*) → str

Generate a login url.For a more secure alternative, use *get_secure_login_data()*

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **redirect_uri** (*str*) – The Redirect URI of your Azure App

Returns
The url to the website on which the user logs in

Return type
str

generate_state() → str

Generates a random state

Return type
str

get_secure_login_data(*client_id: str*, *redirect_uri: str*, *state: str | None = None*) → tuple[str, str, str]

Generates the login data for a secure login with pkce and state.Prevents Cross-Site Request Forgery attacks and authorization code injection attacks.

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **redirect_uri** (*str*) – The Redirect URI of your Azure App
- **state** (*str | None*) – You can use a existing state. If not set, a state will be generated using *generate_state()*.

Return type
tuple[str, str, str]

url_contains_auth_code(*url: str*) → bool

Checks if the given url contains a authorization code

Parameters
url (*str*) – The URL to check

Return type
bool

get_auth_code_from_url(*url: str*) → str | None

Get the authorization code from the url.If you want to check the state, use *parse_auth_code_url()*, which throws errors instead of returning an optional value.

Parameters
url (*str*) – The URL to parse

Returns
The auth code or None if the the code is nonexistent

Return type
str | None

parse_auth_code_url(*url: str*, *state: str | None*) → str

Parse the authorization code url and checks the state.

Parameters

- **url** (*str*) – The URL to parse
- **state** (*str | None*) – If set, the function raises a AssertionError, if the state do no match the state in the URL

Returns
The auth code

Return type
str

get_authorization_token(*client_id: str*, *client_secret: str | None*, *redirect_uri: str*, *auth_code: str*, *code_verifier: str | None = None*) → *AuthorizationTokenResponse*

Get the authorization token. This function is called during *complete_login()*, so you need to use this function ony if *complete_login()* doesn't work for you.

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **client_secret** (*str | None*) – The Client Secret of your Azure App. This is deprecated and should not been used anymore.
- **redirect_uri** (*str*) – The Redirect URI of your Azure App
- **auth_code** (*str*) – The Code you get from *parse_auth_code_url()*
- **code_verifier** (*str | None*) – The 3rd entry in the Tuple you get from *get_secure_login_data()*

Return type
AuthorizationTokenResponse

refresh_authorization_token(*client_id: str*, *client_secret: str | None*, *redirect_uri: str | None*, *refresh_token: str*) → *AuthorizationTokenResponse*

Refresh the authorization token. This function is called during *complete_refresh()*, so you need to use this function ony if *complete_refresh()* doesn't work for you.

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **client_secret** (*str | None*) – The Client Secret of your Azure App. This is deprecated and should not been used anymore.
- **redirect_uri** (*str | None*) – The Redirect URI of Azure App. This Parameter only exists for backwards compatibility and is not used anymore.
- **refresh_token** (*str*) – Your refresh token

Return type
AuthorizationTokenResponse

authenticate_with_xbl(*access_token: str*) → *XBLResponse*

Authenticate with Xbox Live. This function is called during *complete_login()*, so you need to use this function ony if *complete_login()* doesn't work for you.

Parameters
access_token (*str*) – The Token you get from *get_authorization_token()*

Return type
XBLResponse

authenticate_with_xsts(*xbl_token: str*) → *XSTSResponse*

Authenticate with XSTS. This function is called during *complete_login()*, so you need to use this function ony if *complete_login()* doesn't work for you.

Parameters
xbl_token (*str*) – The Token you get from *authenticate_with_xbl()*

Return type
XSTSResponse

authenticate_with_minecraft(*userhash: str*, *xsts_token: str*) → *MinecraftAuthenticateResponse*

Authenticate with Minecraft. This function is called during *complete_login()*, so you need to use this function ony if *complete_login()* doesn't work for you.

Parameters

- **userhash** (*str*) – The Hash you get from *authenticate_with_xbl()*
- **xsts_token** (*str*) – The Token you get from *authenticate_with_xsts()*

Return type
MinecraftAuthenticateResponse

get_store_information(*access_token: str*) → *MinecraftStoreResponse*

Get the store information.

Parameters
access_token (*str*) – The Token you get from *authenticate_with_minecraft()*

Return type
MinecraftStoreResponse

get_profile(*access_token: str*) → *MinecraftProfileResponse*

Get the profile. This function is called during *complete_login()*, so you need to use this function ony if *complete_login()* doesn't work for you.

Parameters
access_token (*str*) – The Token you get from *authenticate_with_minecraft()*

Return type
MinecraftProfileResponse

complete_login(*client_id: str*, *client_secret: str | None*, *redirect_uri: str*, *auth_code: str*, *code_verifier: str | None = None*) → *CompleteLoginResponse*

Do the complete login process.

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **client_secret** (*str | None*) – The Client Secret of your Azure App. This is deprecated and should not been used anymore.

- **redirect_uri** (*str*) – The Redirect URI of your Azure App
- **auth_code** (*str*) – The Code you get from *parse_auth_code_url()*
- **code_verifier** (*str | None*) – The 3rd entry in the Tuple you get from *get_secure_login_data()*

Raises

- ***AzureAppNotPermitted*** – Your Azure App don't have the Permission to use the Minecraft API
- ***AccountNotOwnMinecraft*** – The Account does not own Minecraft

Return type
CompleteLoginResponse

It returns the following:

```
{
    "id" : "The uuid",
    "name" : "The username",
    "access_token": "The acces token",
    "refresh_token": "The refresh token",
    "skins" : [{
        "id" : "6a6e65e5-76dd-4c3c-a625-162924514568",
        "state" : "ACTIVE",
        "url" : "http://textures.minecraft.net/texture/
↪1a4af718455d4aab528e7a61f86fa25e6a369d1768dcb13f7df319a713eb810b",
        "variant" : "CLASSIC",
        "alias" : "STEVE"
    } ],
    "capes" : []
}
```

complete_refresh(*client_id: str*, *client_secret: str | None*, *redirect_uri: str | None*, *refresh_token: str*) → *CompleteLoginResponse*

Do the complete login process with a refresh token. It returns the same as *complete_login()*.

Parameters

- **client_id** (*str*) – The Client ID of your Azure App
- **client_secret** (*str | None*) – The Client Secret of your Azure App. This is deprecated and should not been used anymore.
- **redirect_uri** (*str | None*) – The Redirect URI of Azure App. This Parameter only exists for backwards compatibility and is not used anymore.
- **refresh_token** (*str*) – Your refresh token

Raises

- ***InvalidRefreshToken*** – Your refresh token is not valid
- ***AccountNotOwnMinecraft*** – The Account does not own Minecraft

Return type
CompleteLoginResponse

View the source code of this module

4.5 utils

utils contains a few functions for helping you that doesn't fit in any other category

get_minecraft_directory() → str

Returns the default path to the .minecraft directory

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
print(f"The default minecraft directory is {minecraft_directory}")
```

Return type
str

get_latest_version() → *LatestMinecraftVersions*

Returns the latest version of Minecraft

Example:

```
latest_version = minecraft_launcher_lib.utils.get_latest_version()
print("Latest Release " + latest_version["release"])
print("Latest Snapshot " + latest_version["snapshot"])
```

Return type
LatestMinecraftVersions

get_version_list() → list[*MinecraftVersionInfo*]

Returns all versions that Mojang offers to download

Example:

```
for version in minecraft_launcher_lib.utils.get_version_list():
    print(version["id"])
```

Return type
list[MinecraftVersionInfo]

get_installed_versions(*minecraft_directory: str | PathLike*) → list[*MinecraftVersionInfo*]

Returns all installed versions

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
for version in minecraft_launcher_lib.utils.get_installed_versions(minecraft_
↪directory):
    print(version["id"])
```

Parameters
minecraft_directory (*str | PathLike*) – The path to your Minecraft directory

Return type
list[MinecraftVersionInfo]

get_available_versions(*minecraft_directory: str | PathLike*) → list[*MinecraftVersionInfo*]

Returns all installed versions and all versions that Mojang offers to download

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
for version in minecraft_launcher_lib.utils.get_available_versions(minecraft_
↪directory):
    print(version["id"])
```

Parameters
minecraft_directory (*str | PathLike*) – The path to your Minecraft directory

Return type
list[MinecraftVersionInfo]

get_java_executable() → str

Tries the find out the path to the default java executable. Returns `java`, if no path was found.

Example:

```
print("The path to Java is " + minecraft_launcher_lib.utils.get_java_executable())
```

Return type
str

get_library_version() → str

Returns the version of minecraft-launcher-lib

Example:

```
print(f"You are using version {minecraft_launcher_lib.utils.get_library_version()}␣
↪of minecraft-launcher-lib")
```

Return type
str

generate_test_options() → *MinecraftOptions*

Generates test options to launch minecraft. This includes a random name and a random uuid.

Note

This function is just for debugging and testing, if Minecraft works. The behavior of this function may change in the future. Do not use it in production.

Example:

```
version = "1.0"
options = minecraft_launcher_lib.utils.generate_test_options()
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
command = minecraft_launcher_lib.command.get_minecraft_command(version, minecraft_
↪directory, options)
subprocess.run(command)
```

Return type
MinecraftOptions

is_version_valid(*version: str*, *minecraft_directory: str | PathLike*) → bool

Checks if the given version exists. This checks if the given version is installed or offered to download by Mojang. Basically you can use this tho check, if the given version can be used with *install_minecraft_version()*.

Example:

```
version = "1.0"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
if minecraft_launcher_lib.utils.is_version_valid(version, minecraft_directory):
    print(f"{version} is a valid version")
else:
    print(f"{version} is not a valid version")
```

Parameters

- **version** (*str*) – A Minecraft version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory

Return type
bool

is_vanilla_version(*version: str*) → bool

Checks if the given version is a vanilla version

Example:

```
version = "1.0"
if minecraft_launcher_lib.utils.is_vanilla_version(version):
    print(f"{version} is a vanilla version")
else:
    print(f"{version} is not a vanilla version")
```

Parameters
version (*str*) – A Minecraft version

Return type
bool

is_platform_supported() → bool

Checks if the current platform is supported

Example:

```
if not minecraft_launcher_lib.utils.is_platform_supported():
    print("Your platform is not supported", file=sys.stderr)
    sys.exit(1)
```

Return type
bool

is_minecraft_installed(*minecraft_directory: str | PathLike*) → bool

Checks, if there is already a existing Minecraft Installation in the given Directory

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
if minecraft_launcher_lib.utils.is_minecraft_installed(minecraft_directory):
    print("Minecraft is installed")
else:
    print("Minecraft is not installed")
```

Parameters
minecraft_directory (*str | PathLike*) – The path to your Minecraft directory

Returns
Is a Installation is found

Return type
bool

View the source code of this module

4.6 news

news includes functions to retrieve news about Minecraft using the official API from Mojang

Warning

The format of the data returned by this API may change at any time

get_minecraft_news() → *MinecraftNews*

Returns general news about Minecraft. It uses https://launchercontent.mojang.com/news.json as source.

Example:

```
news = minecraft_launcher_lib.news.get_minecraft_news()
for entry in news["entries"]:
    print(entry["title"] + ":")
    print(entry["text"])
    print()
```

Returns
The news

Return type
MinecraftNews

get_java_patch_notes() → *JavaPatchNotes*

Returns the patch notes for Minecraft Java Edition. It uses https://launchercontent.mojang.com/javaPatchNotes.json as source.

Example:

```
news = minecraft_launcher_lib.news.get_java_patch_notes()
for entry in news["entries"]:
    print(entry["title"] + ":")
    print(entry["body"])
    print()
```

Returns
The patch notes

Return type
JavaPatchNotes

View the source code of this module

4.7 java_utils

java_utils contains some functions to help with Java

get_java_information(*path: str | PathLike*) → *JavaInformation*

Returns Some Information about the given Java Installation

> **Note**
>
> This Function executes the Java executable to determine details such as the version. This might be a security risk.

Example:

```
java_path = "<path>"
information = minecraft_launcher_lib.java_utils.get_java_information(java_path)
print("Name: " + information["name"])
print("Version: " + information["version"])
print("Java path: " + information["java_path"])
```

Parameters
path (*str | PathLike*) – The Path to the Installation. It must be the Directory. If your Java executable is e.g. /usr/lib/jvm/java-19-openjdk-amd64/bin/java this Parameter must be /usr/lib/jvm/java-19-openjdk-amd64.

Returns
A dict with Information about the given java installation

Raises
ValueError – Wrong path

Return type
JavaInformation

find_system_java_versions(*additional_directories: list[str | PathLike] | None = None*) → list[str]

Try to find all Java Versions installed on the System. You can use this to e.g. let the User choose between different Java Versions in a Dropdown. macOS is not supported yet.

Example:

```
for version in minecraft_launcher_lib.java_utils.find_system_java_versions():
    print(version)
```

Parameters
additional_directories (*list[str | PathLike] | None*) – A List of additional Directories to search for Java in custom locations

Returns
A List with all Directories of Java Installations

Return type
list[str]

find_system_java_versions_information(*additional_directories: list[str | PathLike] | None = None*) → list[*JavaInformation*]

Same as `find_system_java_version()`, but uses *get_java_information()* to get some Information about the Installation instead of just proving a Path. macOS is not supported yet

Note

This Function executes the Java executable to determine details such as the version. This might be a security risk.

Example:

```
for version_information in minecraft_launcher_lib.java_utils.find_system_java_
↪versions_information():
    print("Path: " + version_information["path"])
    print("Name: " + version_information["name"])
    print("Version: " + version_information["version"])
    print("Java path: " + version_information["java_path"])
    print()
```

Parameters
additional_directories (*list[str | PathLike] | None*) – A List of additional Directories to search for Java in custom locations

Returns
A List with Information of Java Installations

Return type
list[JavaInformation]

View the source code of this module

4.8 mod_loader

This module offers a standardized way to access the functions of various mod loaders. You should take a look at the *tutorial* before using this module.

Added in version 8.0.

get_mod_loader(*mod_loader_id: str*, */ (Positional-only parameter separator (PEP 570))*) → *ModLoader*

Returns the mod loader with the given ID

Example:

```
fabric = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
```

Parameters
mod_loader_id (*str*) – the mod loader id

Raises
ValueError – An invalid ID was passed

Returns
The mod loader

Return type
ModLoader

list_mod_loader() → list[str]

Returns a list of all available mod loader ids

Example:

```
id_list = minecraft_launcher_lib.mod_loader.list_mod_loader()
for current_id in id_list:
    print(current_id)
```

Returns
A list of all ids

Return type
list[str]

class ModLoader(*base: ModLoaderBase*)

Bases: `object`

This class offers a standardized way to access the functions of various mod loaders. You can obtain an instance of this class by calling *get_mod_loader()*. You should not create a new instance of this class on your own.

Parameters
base (*ModLoaderBase*)

get_id() → str

Returns the ID of the mod loader

Example:

```
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
print(mod_loader.get_id())
```

Return type
str

get_name() → str

Returns the name of the mod loader

Example:

```
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
print(mod_loader.get_name())
```

Return type
str

get_minecraft_versions(*stable_only: bool*) → list[str]

Returns a list of all Minecraft versions that the mod loader supports.

Example:

```
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
for version in mod_loader.get_minecraft_versions(True):
    print(version)
```

Parameters
stable_only (*bool*) – Only return stable versions and not snapshots. This parameter is not supported by every mod loader.

Return type
list[str]

is_minecraft_version_supported(*minecraft_version: str*) → bool

Checks if the given minecraft version is supported by the mod loader.

Example:

```
version = "1.21"
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
if mod_loader.is_minecraft_version_supported(version):
    print(f"{version} is supported")
else:
    print(f"{version} is not supported")
```

Parameters
minecraft_version (*str*) – A Minecraft version

Return type
bool

get_loader_versions(*minecraft_version: str*, *stable_only: bool*) → list[str]

Returns all loader versions from newest to oldest that exists for the given Minecraft version

Example:

```
version = "1.21"
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
loader_versions = mod_loader.get_loader_versions(version, True)
for current_version in loader_versions:
    print(current_version)
```

Parameters

- **minecraft_version** (*str*) – A Minecraft version.
- **stable_only** (*bool*) – Only return stable loader versions. This parameter is not supported by every mod loader.

Returns
A list of all loader versions

Return type
list[str]

get_latest_loader_version(*minecraft_version: str*) → str

Returns the latest loader version for the given Minecraft version.

Example:

```
version = "1.21"
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
latest_loader_version = mod_loader.get_latest_loader_version(version)
print(latest_loader_version)
```

Parameters

- **minecraft_version** (*str*) – A Minecraft version.
- **stable_only** – Only return stable loader versions. This parameter is not supported by every mod loader.

Raises
UnsupportedVersion – The given Minecraft version is not supported by the mod loader.

Returns
The latest loader version

Return type
str

get_installed_version(*minecraft_version: str*, *loader_version: str*) → str

This returns the version under which the mod loader will be installed. It can be used with e.g *get_minecraft_command()*.

Example:

```
vanilla_version = "1.21"
loader_version = "0.16.14"
mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader("fabric")
installed_version = mod_loader.get_installed_version(vanilla_version, loader_
↪version)
print(installed_version)
```

Parameters

- **minecraft_version** (*str*) – The vanilla version
- **loader_version** (*str*) – The loader version

Returns
The installed version

Return type
str

install(*minecraft_version: str*, *minecraft_directory: str | PathLike*, ** (Keyword-only parameters separator (PEP 3102))*, *loader_version: str | None = None*, *callback:* CallbackDict *| None = None*, *java: str | PathLike | None = None*) → str

Installs the mod loader for the given vanilla version.

Example:

```
vanilla_version = "1.21"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
mod_loader = mod_loader = minecraft_launcher_lib.mod_loader.get_mod_loader(
↪"fabric")
mod_loader.install(vanilla_version, minecraft_directory)
```

Parameters

- **minecraft_version** (*str*) – The vanilla Minecraft version for which to install the mod loader. If not installed, minecraft-launcher-lib will also install the vanilla version.
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **loader_version** (*str | None*) – The version of the mod loader as returned by *get_loader_versions()*.If not set the latest one will be used.
- **callback** (CallbackDict *| None*) – See *Get Installation Progress*
- **java** (*str | PathLike | None*) – If set use this Java executable to execute programs during the installation.

Raises

- ***VersionNotFound*** – An invalid minecraft version was passed.
- ***UnsupportedVersion*** – The given Minecraft version is not supported by the mod loader.

Raises
CalledProcessError: The execution of a program failed.

Returns
The same as *get_installed_version()*.

Return type
str

View the source code of this module

4.9 forge

Warning

This module is deprecated and has been replaced by *mod_loader*. It will no longer receive updates or bug fixes and may be removed in a future release.

Note

Before using this module, please read this comment from the forge developers:

```
Please do not automate the download and installation of Forge.
Our efforts are supported by ads from the download page.
If you MUST automate this, please consider supporting the project through https://www.
↪patreon.com/LexManos/
```

It's your choice, if you want to respect that and support forge.

forge contains functions for dealing with the Forge modloader

install_forge_version(*versionid: str*, *path: str | PathLike*, *callback: CallbackDict | None = None*, *java: str | PathLike | None = None*) → None

Installs the given Forge version

Parameters

- **versionid** (*str*) – A Forge Version. You can get a List of Forge versions using *list_forge_versions()*
- **path** (*str | PathLike*) – The path to your Minecraft directory
- **callback** (*CallbackDict | None*) – The same dict as for *install_minecraft_version()*
- **java** (*str | PathLike | None*) – A Path to a custom Java executable

Return type
None

Raises a *VersionNotFound* exception when the given forge version is not found

run_forge_installer(*version: str*, *java: str | PathLike | None = None*) → None

Run the forge installer of the given forge version

Parameters

- **version** (*str*) – A Forge Version. You can get a List of Forge versions using *list_forge_versions()*
- **java** (*str | PathLike | None*) – A Path to a custom Java executable

Return type
None

list_forge_versions() → list[str]

Returns a list of all forge versions

Return type
list[str]

find_forge_version(*vanilla_version: str*) → str | None

Find the latest forge version that is compatible to the given vanilla version

Parameters
vanilla_version (*str*) – A vanilla Minecraft version

Return type
str | None

is_forge_version_valid(*forge_version: str*) → bool

Checks if a forge version is valid

Parameters
forge_version (*str*) – A Forge Version

Return type
bool

supports_automatic_install(*forge_version: str*) → bool

Checks if install_forge_version() supports the given forge version

Parameters
forge_version (*str*) – A Forge Version

Return type
bool

forge_to_installed_version(*forge_version: str*) → str

Returns the Version under which Forge will be installed from the given Forge version.

Parameters
forge_version (*str*) – A Forge Version

Return type
str

Raises a ValueError if the Version is invalid.

View the source code of this module

4.10 fabric

Warning

This module is deprecated and has been replaced by *mod_loader*. It will no longer receive updates or bug fixes and may be removed in a future release.

fabric contains functions for dealing with the Fabric modloader.

get_all_minecraft_versions() → list[*FabricMinecraftVersion*]

Returns all available Minecraft Versions for Fabric

Example:

```
for version in minecraft_launcher_lib.fabric.get_all_minecraft_versions():
    print(version["version"])
```

Return type
list[FabricMinecraftVersion]

get_stable_minecraft_versions() → list[str]

Returns a list which only contains the stable Minecraft versions that supports Fabric

Example:

```
for version in minecraft_launcher_lib.fabric.get_stable_minecraft_versions():
    print(version)
```

Return type
list[str]

get_latest_minecraft_version() → str

Returns the latest unstable Minecraft versions that supports Fabric. This could be a snapshot.

Example:

```
print("Latest Minecraft version: " + minecraft_launcher_lib.fabric.get_latest_
↪minecraft_version())
```

Return type
str

get_latest_stable_minecraft_version() → str

Returns the latest stable Minecraft version that supports Fabric

Example:

```
print("Latest stable Minecraft version: " + minecraft_launcher_lib.fabric.get_
↪latest_stable_minecraft_version())
```

Return type
str

is_minecraft_version_supported(*version: str*) → bool

Checks if a Minecraft version supported by Fabric

Example:

```
version = "1.20"
if minecraft_launcher_lib.fabric.is_minecraft_version_supported(version):
    print(f"{version} is supported by fabric")
else:
    print(f"{version} is not supported by fabric")
```

Parameters
version (*str*) – A vanilla version

Return type
bool

get_all_loader_versions() → list[*FabricLoader*]

Returns all loader versions

Example:

```
for version in minecraft_launcher_lib.fabric.get_all_loader_versions():
    print(version["version"])
```

Return type
list[FabricLoader]

get_latest_loader_version() → str
Get the latest loader version

Example:

```
print("Latest loader version: " + minecraft_launcher_lib.fabric.get_latest_loader_
↪version())
```

Return type
str

get_latest_installer_version() → str
Returns the latest installer version

Example:

```
print("Latest installer version: " + minecraft_launcher_lib.fabric.get_latest_
↪installer_version())
```

Return type
str

install_fabric(*minecraft_version: str*, *minecraft_directory: str | PathLike*, *loader_version: str | None = None*, *callback:* CallbackDict *| None = None*, *java: str | PathLike | None = None*) → None
Installs the Fabric modloader.

Example:

```
minecraft_version = "1.20"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.fabric.install_fabric(minecraft_version, minecraft_directory)
```

Parameters

- **minecraft_version** (*str*) – A vanilla version that is supported by Fabric
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **loader_version** (*str | None*) – The fabric loader version. If not given it will use the latest
- **callback** (CallbackDict | *None*) – The same dict as for *install_minecraft_version()*
- **java** (*str | PathLike | None*) – A Path to a custom Java executable

Raises

- ***VersionNotFound*** – The given Minecraft does not exists
- ***UnsupportedVersion*** – The given Minecraft version is not supported by Fabric

Return type
None

View the source code of this module

4.11 quilt

Warning

This module is deprecated and has been replaced by *mod_loader*. It will no longer receive updates or bug fixes and may be removed in a future release.

quilt contains functions for dealing with the Quilt modloader.

You may have noticed, that the Functions are the same as in the *fabric* module. That's because Quilt is a Fork of Fabric. This module behaves exactly the same as the fabric module.

get_all_minecraft_versions() → list[*QuiltMinecraftVersion*]

Returns all available Minecraft Versions for Quilt

Example:

```
for version in minecraft_launcher_lib.quilt.get_all_minecraft_versions():
    print(version["version"])
```

Return type
list[QuiltMinecraftVersion]

get_stable_minecraft_versions() → list[str]

Returns a list which only contains the stable Minecraft versions that supports Quilt

Example:

```
for version in minecraft_launcher_lib.quilt.get_stable_minecraft_versions():
    print(version)
```

Return type
list[str]

get_latest_minecraft_version() → str

Returns the latest unstable Minecraft versions that supports Quilt. This could be a snapshot.

Example:

```
print("Latest Minecraft version: " + minecraft_launcher_lib.quilt.get_latest_
↪minecraft_version())
```

Return type
str

get_latest_stable_minecraft_version() → str

Returns the latest stable Minecraft version that supports Quilt

Example:

```
print("Latest stable Minecraft version: " + minecraft_launcher_lib.quilt.get_latest_
↪stable_minecraft_version())
```

Return type
str

is_minecraft_version_supported(*version: str*) → bool

Checks if a Minecraft version supported by Quilt

Example:

```
version = "1.20"
if minecraft_launcher_lib.quilt.is_minecraft_version_supported(version):
    print(f"{version} is supported by quilt")
else:
    print(f"{version} is not supported by quilt")
```

Parameters
version (*str*) – A vanilla version

Return type
bool

get_all_loader_versions() → list[*QuiltLoader*]

Returns all loader versions

Example:

```
for version in minecraft_launcher_lib.quilt.get_all_loader_versions():
    print(version["version"])
```

Return type
list[QuiltLoader]

get_latest_loader_version() → str

Get the latest loader version

Example:

```
print("Latest loader version: " + minecraft_launcher_lib.quilt.get_latest_loader_
↪version())
```

Return type
str

get_latest_installer_version() → str

Returns the latest installer version

Example:

```
print("Latest installer version: " + minecraft_launcher_lib.quilt.get_latest_
↪installer_version())
```

Return type
str

install_quilt(*minecraft_version: str*, *minecraft_directory: str | PathLike*, *loader_version: str | None = None*, *callback: CallbackDict | None = None*, *java: str | PathLike | None = None*) → None

Installs the Quilt modloader.

Example:

```
minecraft_version = "1.20"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.quilt.install_quilt(minecraft_version, minecraft_directory)
```

Parameters

- **minecraft_version** (*str*) – A vanilla version that is supported by Quilt
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **loader_version** (*str | None*) – The Quilt loader version. If not given it will use the latest
- **callback** (*CallbackDict | None*) – The same dict as for *install_minecraft_version()*
- **java** (*str | PathLike | None*) – A Path to a custom Java executable

Raises

- ***VersionNotFound*** – The given Minecraft does not exists
- ***UnsupportedVersion*** – The given Minecraft version is not supported by Quilt

Return type
None

View the source code of this module

4.12 runtime

runtime allows to install the java runtime. This module is used by *install_minecraft_version()*, so you don't need to use it in your code most of the time.

get_jvm_runtimes() → list[str]

Returns a list of all jvm runtimes

Example:

```
for runtime in minecraft_launcher_lib.runtime.get_jvm_runtimes():
    print(runtime)
```

Return type
list[str]

get_installed_jvm_runtimes(*minecraft_directory: str | PathLike*) → list[str]

Returns a list of all installed jvm runtimes

Example:

```
for runtime in minecraft_launcher_lib.runtime.get_installed_jvm_runtimes():
    print(runtime)
```

Parameters
minecraft_directory (*str | PathLike*) – The path to your Minecraft directory

Return type
list[str]

install_jvm_runtime(*jvm_version: str*, *minecraft_directory: str | PathLike*, *callback:* CallbackDict | *None* = *None*, *max_workers: int | None = None*) → None

Installs the given jvm runtime. callback is the same dict as in the install module.

Example:

```
runtime_version = "java-runtime-gamma"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.runtime.install_jvm_runtime(runtime_version, minecraft_
↪directory)
```

Parameters

- **jvm_version** (*str*) – The Name of the JVM version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **callback** (CallbackDict | *None*) – the same dict as for *install_minecraft_version()*
- **max_workers** (*int | None*) – number of workers for asynchronous downloads. If None, max_workers will be set automatically.

Raises

- ***VersionNotFound*** – The given JVM Version was not found
- ***FileOutsideMinecraftDirectory*** – A File should be placed outside the given Minecraft directory

Return type
None

get_executable_path(*jvm_version: str*, *minecraft_directory: str | PathLike*) → str | None

Returns the path to the executable. Returns None if none is found.

Example:

```
runtime_version = "java-runtime-gamma"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
executable_path = minecraft_launcher_lib.runtime.get_executable_path(runtime_
↪version, minecraft_directory)
if executable_path is not None:
    print(f"Executable path: {executable_path}")
else:
    print("The executable path was not found")
```

Parameters

- **jvm_version** (*str*) – The Name of the JVM version
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory

Return type
str | None

get_jvm_runtime_information(*jvm_version: str*) → *JvmRuntimeInformation*

Returns some Information about a JVM Version

Example:

```
runtime_version = "java-runtime-gamma"
information = minecraft_launcher_lib.runtime.get_jvm_runtime_information(runtime_
↪version)
print("Java version: " + information["name"])
print("Release date: " + information["released"].isoformat())
```

Parameters
jvm_version (*str*) – A JVM Version

Raises
- ***VersionNotFound*** – The given JVM Version was not found
- ***VersionNotFound*** – The given JVM Version is not available on this Platform

Returns
A Dict with Information

Return type
JvmRuntimeInformation

get_version_runtime_information(*version: str*, *minecraft_directory: str | PathLike*) → *VersionRuntimeInformation* | None

Returns information about the runtime used by a version

Example:

```
minecraft_version = "1.20"
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
information = minecraft_launcher_lib.runtime.get_version_runtime_
↪information(minecraft_version, minecraft_directory)
print("Name: " + information["name"])
print("Java version: " + str(information["javaMajorVersion"]))
```

Parameters
- **minecraft_directory** (*str | PathLike*) – The path to your Minecraft directory
- **version** (*str*)

Raises
VersionNotFound – The Minecraft version was not found

Returns
A Dict with Information. None if the version has no runtime information.

Return type
VersionRuntimeInformation | None

View the source code of this module

4.13 vanilla_launcher

vanilla_launcher contains some functions for interacting with the Vanilla Minecraft Launcher

load_vanilla_launcher_profiles(*minecraft_directory: str | PathLike*) → list[*VanillaLauncherProfile*]

Loads the profiles of the Vanilla Launcher from the given Minecraft directory

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
profiles = minecraft_launcher_lib.vanilla_launcher.load_vanilla_launcher_
↪profiles(minecraft_directory)

for profile in profiles:
    print(profile["name"])
```

Parameters
minecraft_directory (*str | PathLike*) – The Minecraft directory

Returns
A List with the Profiles

Return type
list[VanillaLauncherProfile]

vanilla_launcher_profile_to_minecraft_options(*vanilla_profile:* VanillaLauncherProfile) → *MinecraftOptions*

Converts a VanillaLauncherProfile into a Options dict, that can be used by *get_minecraft_command()*. You still need to add the Login Data to the Options before you can use it.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
profiles = minecraft_launcher_lib.vanilla_launcher.load_vanilla_launcher_
↪profiles(minecraft_directory)

test_options = minecraft_launcher_lib.utils.generate_test_options()
profile_options = minecraft_launcher_lib.vanilla_launcher.vanilla_launcher_profile_
↪to_minecraft_options(profiles[0])
options = test_options | profile_options

minecraft_version = minecraft_launcher_lib.vanilla_launcher.get_vanilla_launcher_
↪profile_version(profiles[0])
command = minecraft_launcher_lib.command.get_minecraft_command(minecraft_version,␣
↪minecraft_directory, options)
subprocess.run(command, cwd=minecraft_directory)
```

Parameters
vanilla_profile (VanillaLauncherProfile) – The profile as returned by *load_vanilla_launcher_profiles()*

Raises
InvalidVanillaLauncherProfile – The given Profile is invalid

Returns
The Options Dict

Return type
MinecraftOptions

get_vanilla_launcher_profile_version(*vanilla_profile:* VanillaLauncherProfile) → str

Returns the Minecraft version of the VanillaProfile. Handles `latest-release` and `latest-snapshot`.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
profiles = minecraft_launcher_lib.vanilla_launcher.load_vanilla_launcher_
↪profiles(minecraft_directory)

for profile in profiles:
    print(minecraft_launcher_lib.vanilla_launcher.get_vanilla_launcher_profile_
↪version(profile))
```

Parameters
vanilla_profile (VanillaLauncherProfile) – The Profile

Raises
InvalidVanillaLauncherProfile – The given Profile is invalid

Returns
The Minecraft version

Return type
str

add_vanilla_launcher_profile(*minecraft_directory: str | PathLike*, *vanilla_profile:* VanillaLauncherProfile) → None

Adds a new Profile to the Vanilla Launcher

Example:

```
profile: minecraft_launcher_lib.types.VanillaLauncherProfile = {
    "name": "test",
    "versionType": "latest-release",
}

minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.vanilla_launcher.add_vanilla_launcher_profile(minecraft_
↪directory, profile)
```

Parameters

- **minecraft_directory** (*str | PathLike*) – The Minecraft directory
- **vanilla_profile** (VanillaLauncherProfile) – The new Profile

Raises
InvalidVanillaLauncherProfile – The given Profile is invalid

Return type
None

do_vanilla_launcher_profiles_exists(*minecraft_directory: str | PathLike*) → bool

Checks if profiles from the vanilla launcher can be found

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
if minecraft_launcher_lib.vanilla_launcher.do_vanilla_launcher_profiles_
↪exists(minecraft_directory):
    print("Profiles exists")
else:
    print("Profiles do not exists")
```

Parameters
minecraft_directory (*str | PathLike*) – The Minecraft directory

Returns
If profiles exists

Return type
bool

create_empty_vanilla_launcher_profiles_file(*minecraft_directory: str | PathLike*) → None

Creates a `launcher_profiles.json` file with the correct structure inside the Minecraft directory. This function is useful for 3rd party software (e.g. some mod installers) that requires this file to be present. A existing file will be overwritten, so sue this function with caution.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.vanilla_launcher.create_empty_vanilla_launcher_profiles_
↪file(minecraft_directory)
```

Parameters
minecraft_directory (*str | PathLike*) – The Minecraft directory

Return type
None

Added in version 8.0.

ensure_vanilla_launcher_profiles_exists(*minecraft_directory: str | PathLike*) → None

Calls *create_empty_vanilla_launcher_profiles_file()* if *do_vanilla_launcher_profiles_exists()* returns `False`. You should call this function before you use a 3rd party software that requires `launcher_profiles.json` to be present.

Example:

```
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
minecraft_launcher_lib.vanilla_launcher.ensure_vanilla_launcher_profiles_
↪exists(minecraft_directory)
```

Parameters
minecraft_directory (*str | PathLike*) – The Minecraft directory

Return type
None

Added in version 8.0.

View the source code of this module

4.14 mrpack

mrpack allows you to install Modpacks from the Mrpack Format. You should also take a look at the *complete example*.

get_mrpack_information(*path: str | PathLike*) → *MrpackInformation*

Gets some Information from a .mrpack file

Example:

```
path = "/path/to/mrpack"
information = minecraft_launcher_lib.mrpack.get_mrpack_information(path)
print("Name: " + information["name"])
print("Summary: " + information["summary"])
```

Parameters
path (*Union[str, os.PathLike]*) – The Path the the .mrpack file

Returns
The Information about the given Mrpack

Return type
MrpackInformation

install_mrpack(*path: str | PathLike*, *minecraft_directory: str | PathLike*, *modpack_directory: str | PathLike | None = None*, *callback: CallbackDict | None = None*, *mrpack_install_options: MrpackInstallOptions | None = None*) → None

Installs a .mrpack file

`mrpack_install_options` is a dict. All Options are Optional.

```
mrpack_install_options = {
    "optionalFiles": [], # List with all Optional files
    "skipDependenciesInstall": False # If you want to skip the Dependencies install.
↪ Only used for testing purposes.
}
```

Example:

```
path = "/path/to/mrpack"
minecraft_directory = minecraft_directory.utils.get_minecraft_directory()
minecraft_launcher_lib.mrpack.install_mrpack(path, minecraft_directory)
```

Parameters

- **path** (*str | PathLike*) – The Path the the .mrpack file
- **minecraft_directory** (*str | PathLike*) – he path to your Minecraft directory
- **modpack_directory** (*str | PathLike | None*) – If you want to install the Pack in another Directory than your Minecraft directory, set it here.

- **callback** (CallbackDict | *None*) – The same dict as for *install_minecraft_version()*
- **mrpack_install_options** (MrpackInstallOptions | *None*) – Some Options to install the Pack (see below)

Raises
FileOutsideMinecraftDirectory – A File should be placed outside the given Minecraft directory

Return type
None

get_mrpack_launch_version(*path: str | PathLike*) → str

Returns that Version that needs to be used with *get_minecraft_command()*.

Example:

```
path = "/path/to/mrpack"
print("Launch version: " + minecraft_launcher_lib.mrpack.get_mrpack_launch_
↪version(path))
```

Parameters
path (*str | PathLike*) – The Path the the .mrpack file

Returns
The version

Return type
str

View the source code of this module

4.15 exceptions

exceptions contains all custom exceptions that can be raised by minecraft_launcher_lib

exception VersionNotFound(*version: str*)

Bases: ValueError

The given version does not exists

Parameters
version (*str*)

Return type
None

version: str

The version that caused the exception

msg: str

A message to display

exception UnsupportedVersion(*version: str*)

Bases: ValueError

This Exception is raised when you try to run *install_fabric()* or *install_quilt()* with a unsupported version

Parameters
version (*str*)

Return type
None

version: str
The version that caused the exception

msg: str
A message to display

exception ExternalProgramError(*command: list[str]*, *stdout: bytes*, *stderr: bytes*)
Bases: Exception

This Exception is raised when a external program failed

Deprecated since version 8.0.

Parameters
- **command** (*list[str]*)
- **stdout** (*bytes*)
- **stderr** (*bytes*)

Return type
None

command: list[str]
The command that caused the error

stdout: bytes
The stdout of the command

stderr: bytes
The stderr of the command

exception InvalidRefreshToken
Bases: ValueError

Raised when *complete_refresh()* is called with a invalid refresh token

exception InvalidVanillaLauncherProfile(*profile:* VanillaLauncherProfile)
Bases: ValueError

Raised when a function from the *vanilla_launcher* module is called with a invalid vanilla profile

Parameters
profile (VanillaLauncherProfile)

Return type
None

profile: *VanillaLauncherProfile*
The invalid profile

exception SecurityError(*code: str*, *message: str*)
Bases: Exception

Raised when something security related happens

Parameters

- **code** (*str*)
- **message** (*str*)

Return type
None

code: str
A Code to specify the Error

message: str
A Message to display

exception FileOutsideMinecraftDirectory(*path: str*, *minecraft_directory: str*)
Bases: *SecurityError*

Raised when a File should be placed outside the given Minecraft directory

Parameters

- **path** (*str*)
- **minecraft_directory** (*str*)

Return type
None

path: str
The Path of the File

minecraft_directory: str
The Minecraft directory of the File

exception InvalidChecksum(*url: str*, *path: str*, *expected_checksum: str*, *actual_checksum: str*)
Bases: *SecurityError*

Raised when a File did not match the Checksum

Parameters

- **url** (*str*)
- **path** (*str*)
- **expected_checksum** (*str*)
- **actual_checksum** (*str*)

Return type
None

url: str
The URL to the File with the wrong Checksum

path: str
The Path to the File with the wrong Checksum

expected_checksum: str
The expected Checksum

actual_checksum: str
The actual Checksum

exception AzureAppNotPermitted

Bases: Exception

Raised when you try to use a Azure App, that don't have the Permission to use the Minecraft API. Take a look at the For more information about the options take a look at the *Microsoft Login* tutorial to learn how to fix this.

Return type
None

exception PlatformNotSupported

Bases: Exception

Raised, when the current Platform is not supported by a feature

Return type
None

exception AccountNotOwnMinecraft

Bases: Exception

Raised by *complete_login()* and *complete_login()* when the Account does not own Minecraft

Return type
None

View the source code of this module

4.16 types

This module contains all Types for minecraft-launcher-lib. It may help your IDE. You don't need to use this module directly in your code. If you are not interested in static typing just ignore it. For more information about TypedDict see PEP 589.

class MinecraftOptions

Bases: TypedDict

username: str

uuid: str

token: str

executablePath: str

defaultExecutablePath: str

jvmArguments: list[str]

launcherName: str

launcherVersion: str

gameDirectory: str

demo: bool

customResolution: bool

resolutionWidth: str

resolutionHeight: str

server: str

port: str

nativesDirectory: str

enableLoggingConfig: bool

disableMultiplayer: bool

disableChat: bool

quickPlayPath: str | None

quickPlaySingleplayer: str | None

quickPlayMultiplayer: str | None

quickPlayRealms: str | None

class CallbackDict

Bases: TypedDict

setStatus: Callable[[str], None]

setProgress: Callable[[int], None]

setMax: Callable[[int], None]

class LatestMinecraftVersions

Bases: TypedDict

release: str

snapshot: str

class MinecraftVersionInfo

Bases: TypedDict

id: str

type: str

releaseTime: datetime

complianceLevel: int

class FabricMinecraftVersion

Bases: TypedDict

version: str

stable: bool

class FabricLoader

Bases: TypedDict

separator: str

build: int

maven: str

version: str

stable: bool

class QuiltMinecraftVersion

Bases: TypedDict

version: str

stable: bool

class QuiltLoader

Bases: TypedDict

separator: str

build: int

maven: str

version: str

class JavaInformation

Bases: TypedDict

path: str

name: str

version: str

java_path: str

javaw_path: str | None

is_64bit: bool

openjdk: bool

class VanillaLauncherProfileResolution

Bases: TypedDict

height: int

width: int

class VanillaLauncherProfile

Bases: TypedDict

name: str

version: str | None

versionType: Literal['latest-release', 'latest-snapshot', 'custom']

gameDirectory: str | None

javaExecutable: str | None

javaArguments: list[str] | None

customResolution: *VanillaLauncherProfileResolution* | None

class MrpackInformation

Bases: TypedDict

name: str

summary: str

versionId: str

formatVersion: int

minecraftVersion: str

optionalFiles: list[str]

class MrpackInstallOptions

Bases: TypedDict

optionalFiles: list[str]

skipDependenciesInstall: bool

class JvmRuntimeInformation

Bases: TypedDict

name: str

released: datetime

class VersionRuntimeInformation

Bases: TypedDict

name: str

javaMajorVersion: int

class MinecraftNews

Bases: TypedDict

version: Literal[1]

entries: list[_NewsEntry]

class JavaPatchNotes

Bases: TypedDict

version: Literal[1]

entries: list[_JavaPatchNoteEntry]

View the source code of this module

4.17 microsoft_types

This module contains all Types for the *microsoft_account* module. It has it's own module because of the many types needed that are not used somewhere else.

class AuthorizationTokenResponse

Bases: TypedDict

access_token: str

token_type: Literal['Bearer']

expires_in: int

scope: str

refresh_token: str

class XBLResponse

Bases: TypedDict

IssueInstant: str

NotAfter: str

Token: str

DisplayClaims: _DisplayClaims

class XSTSResponse

Bases: TypedDict

IssueInstant: str

NotAfter: str

Token: str

DisplayClaimns: _DisplayClaims

class MinecraftStoreResponse

Bases: TypedDict

signature: str

keyId: str

class MinecraftAuthenticateResponse

Bases: TypedDict

username: str

roles: list[Any]

access_token: str

token_type: str

expires_in: int

class MinecraftProfileResponse

Bases: TypedDict

id: str

name: str

skins: list[_MinecraftProfileSkin]

capes: list[_MinecraftProfileCape]

error: str

errorMessage: str

class CompleteLoginResponse

Bases: *MinecraftProfileResponse*

access_token: str

refresh_token: str

id: str

name: str

skins: list[_MinecraftProfileSkin]

capes: list[_MinecraftProfileCape]

error: str

errorMessage: str

View the source code of this module

CHAPTER

FIVE

FAQ

Q: Which Minecraft versions are supported?
A: minecraft-launcher-lib supports all Minecraft versions that the official Launcher from Mojang supports. It provides automatic installation for Vanilla, Forge and Fabric. Other versions can be launched after they got installed in other ways.

Q: Which Python versions are supported?
A: minecraft-launcher-lib supports at the moment Python 3.8 and above. PyPy is also official supported.

Q: Which Operating Systems are supported?
A: minecraft-launcher-lib official supports Windows, macOS and Linux, which are the Operating Systems that also supported by Mojang. It might work on other OS, but there is no guaranty.

Q: Can I use minecraft-launcher-lib in my project?
A: minecraft-launcher-lib is licensed under BSD 2-Clause what means it is OpenSource and it can be used in any of your projects. For more information check out the license.

Q: How can I make a cracked launcher?
A: Just buy Minecraft. It's cheaper than a AAA title and brings years of fun.

Q: Is the API stable?
A: All functions that are documented here are stable.

Q: Minecraft does not start
A: Please visit *Troubleshooting*.

Q: Minecraft is creating a logs folder inside my project directory
A: Minecraft is using the working directory for it's logs. You should run Minecraft with a other working directory.

Q: I get a `AzureAppNotPermitted` Exception
A: Please take a look at the *Microsoft Login* tutorial

Q: Does minecraft-launcher-lib supports the Bedrock Edition?
A: Only the Java Edition is supported and there are no plans to support Bedrock

CHAPTER

SIX

EXAMPLES

6.1 TkinterLauncher

```
#!/usr/bin/env python3
# This example shows how to write a basic launcher with Tkinter.
from tkinter import Tk, Label, Entry, Button, mainloop
from tkinter.ttk import Combobox
import minecraft_launcher_lib
import subprocess
import sys


def main():
    def launch():
        window.withdraw()

        minecraft_launcher_lib.install.install_minecraft_version(version_select.get(),␣
↪minecraft_directory)

        login_data = minecraft_launcher_lib.account.login_user(username_input.get(),␣
↪password_input.get())

        options = {
            "username": login_data["selectedProfile"]["name"],
            "uuid": login_data["selectedProfile"]["id"],
            "token": login_data["accessToken"]
        }
        minecraft_command = minecraft_launcher_lib.command.get_minecraft_command(version_
↪select.get(), minecraft_directory, options)

        subprocess.run(minecraft_command)

        sys.exit(0)

    window = Tk()
    window.title("Minecraft Launcher")

    Label(window, text="Username:").grid(row=0, column=0)
    username_input = Entry(window)
    username_input.grid(row=0, column=1)
    Label(window, text="Password:").grid(row=1, column=0)
```

(continues on next page)

(continued from previous page)

```
    password_input = Entry(window)
    password_input.grid(row=1, column=1)

    minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
    versions = minecraft_launcher_lib.utils.get_available_versions(minecraft_directory)
    version_list = []

    for i in versions:
        version_list.append(i["id"])

    Label(window, text="Version:").grid(row=2, column=0)
    version_select = Combobox(window, values=version_list)
    version_select.grid(row=2, column=1)
    version_select.current(0)

    Button(window, text="Launch", command=launch).grid(row=4, column=1)

    mainloop()


if __name__ == "__main__":
    main()
```

View this example on Codeberg

6.2 PyQtLoginWindow

```
#!/usr/bin/env python3
# This example shows how to use the PyQt WebEngine for the Login
# It needs the PyQt6 and PyQt6-WebEngine packages
from PyQt6.QtWidgets import QApplication, QMessageBox
from PyQt6.QtWebEngineWidgets import QWebEngineView
from PyQt6.QtWebEngineCore import QWebEngineProfile
from PyQt6.QtCore import QUrl, QLocale
import minecraft_launcher_lib
import json
import sys
import os

CLIENT_ID = "YOUR CLIENT ID"
REDIRECT_URL = "YOUR REDIRECT URL"


class LoginWindow(QWebEngineView):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Login Window Example")

        # Set the path where the refresh token is saved
        self.refresh_token_file = os.path.join(os.path.dirname(os.path.realpath(__file__
```

(continues on next page)

(continued from previous page)

```
↪)), "refresh_token.json")

        # Login with refresh token, if it exists
        if os.path.isfile(self.refresh_token_file):
            with open(self.refresh_token_file, "r", encoding="utf-8") as f:
                refresh_token = json.load(f)
                # Do the login with refresh token
                try:
                    account_informaton = minecraft_launcher_lib.microsoft_account.
↪complete_refresh(CLIENT_ID, None, REDIRECT_URL, refresh_token)
                    self.show_account_information(account_informaton)
                # Show the window if the refresh token is invalid
                except minecraft_launcher_lib.exceptions.InvalidRefreshToken:
                    pass

        # Open the login url
        login_url, self.state, self.code_verifier = minecraft_launcher_lib.microsoft_
↪account.get_secure_login_data(CLIENT_ID, REDIRECT_URL)
        self.load(QUrl(login_url))

        # Connects a function that is called when the url changed
        self.urlChanged.connect(self.new_url)

        self.show()

    def new_url(self, url: QUrl):
        try:
            # Get the code from the url
            auth_code = minecraft_launcher_lib.microsoft_account.parse_auth_code_url(url.
↪toString(), self.state)
            # Do the login
            account_information = minecraft_launcher_lib.microsoft_account.complete_
↪login(CLIENT_ID, None, REDIRECT_URL, auth_code, self.code_verifier)
            # Show the login information
            self.show_account_information(account_information)
        except AssertionError:
            print("States do not match!")
        except KeyError:
            print("Url not valid")

    def show_account_information(self, information_dict):
        information_string = f'Username: {information_dict["name"]}<br>'
        information_string += f'UUID: {information_dict["id"]}<br>'
        information_string += f'Token: {information_dict["access_token"]}<br>'

        # Save the refresh token in a file
        with open(self.refresh_token_file, "w", encoding="utf-8") as f:
            json.dump(information_dict["refresh_token"], f, ensure_ascii=False, indent=4)

        message_box = QMessageBox()
        message_box.setWindowTitle("Account information")
        message_box.setText(information_string)
```

(continues on next page)

(continued from previous page)

```
        message_box.setStandardButtons(QMessageBox.StandardButton.Ok)
        message_box.exec()

        # Exit the program
        sys.exit(0)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    # This line sets the language of the webpage to the system language
    QWebEngineProfile.defaultProfile().setHttpAcceptLanguage(QLocale.system().name().
↪split("_")[0])
    w = LoginWindow()
    sys.exit(app.exec())
```

View this example on Codeberg

6.3 ModLoader

```
#!/usr/bin/env python3
# This example shows how use the mod loader module
import minecraft_launcher_lib
import subprocess


def choose(options: list[str]) -> int:
    for pos, text in enumerate(options):
        print(f"{pos + 1}: {text}")

    while True:
        try:
            answer = int(input(f"Select[1-{len(options)}]:")) - 1
            if answer >= 0 and answer < len(options):
                return answer
        except ValueError:
            pass


def ask_yes_no(text: str) -> bool:
    while True:
        answer = input(f"{text}[y/n]:").strip().upper()

        if answer == "Y":
            return True
        elif answer == "N":
            return False
        else:
            print("Invalid answer. Use y or n.")


def main() -> None:
```

(continues on next page)

(continued from previous page)

```
    id_list = minecraft_launcher_lib.mod_loader.list_mod_loader()

    name_list = []
    for current_id in id_list:
        name_list.append(minecraft_launcher_lib.mod_loader.get_mod_loader(current_id).
↪get_name())

    print("Please select a mod loader:")

    loader = minecraft_launcher_lib.mod_loader.get_mod_loader(id_list[choose(name_list)])
    version_list = loader.get_minecraft_versions(True)

    print()
    print("Please select the Minecraft version for which you want to install the mod␣
↪loader.")
    vanilla_version = version_list[choose(version_list)]

    print()
    minecraft_directory = input("Enter the path to your Minecraft directory (leave blank␣
↪for default):").strip()
    if minecraft_directory == "":
        minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()

    installed_version = loader.install(vanilla_version, minecraft_directory, callback={
↪"setStatus": print})

    print("Finished")

    if not ask_yes_no("Do you want to launch Minecraft?"):
        return

    command = minecraft_launcher_lib.command.get_minecraft_command(installed_version,␣
↪minecraft_directory, minecraft_launcher_lib.utils.generate_test_options())
    subprocess.run(command, cwd=minecraft_directory)


if __name__ == "__main__":
    main()
```

View this example on Codeberg

6.4 PyQtInstallation

```
#!/usr/bin/env python3
# This example shows how to install Minecraft with a ProgressBar in PyQt
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QComboBox, QLineEdit,␣
↪QPushButton, QProgressBar, QFileDialog, QFormLayout, QHBoxLayout, QVBoxLayout
from PyQt6.QtCore import QThread, pyqtSignal
import minecraft_launcher_lib
import sys
```

(continues on next page)

(continued from previous page)

```
class InstallThread(QThread):
    progress_max = pyqtSignal("int")
    progress = pyqtSignal("int")
    text = pyqtSignal("QString")

    def __init__(self) -> None:
        QThread.__init__(self)
        self._callback_dict = {
            "setStatus": lambda text: self.text.emit(text),
            "setMax": lambda max_progress: self.progress_max.emit(max_progress),
            "setProgress": lambda progress: self.progress.emit(progress),
        }

    def set_data(self, version: str, directory) -> None:
        self._version = version
        self._directory = directory

    def run(self) -> None:
        minecraft_launcher_lib.install.install_minecraft_version(self._version, self._
↪directory, callback=self._callback_dict)


class Window(QWidget):
    def __init__(self) -> None:
        super().__init__()

        self._install_thread = InstallThread()

        self._version_combo_box = QComboBox()
        self._path_edit = QLineEdit()
        self._path_browse_button = QPushButton("Browse")
        self._progress_bar = QProgressBar()
        self._install_single_thread_button = QPushButton("Install Single Thread")
        self._install_multi_thread_button = QPushButton("Install Multi Thread")

        for i in minecraft_launcher_lib.utils.get_version_list():
            self._version_combo_box.addItem(i["id"])

        self._path_edit.setText(minecraft_launcher_lib.utils.get_minecraft_directory())

        self._progress_bar.setTextVisible(True)

        self._install_thread.progress_max.connect(lambda maximum: self._progress_bar.
↪setMaximum(maximum))
        self._install_thread.progress.connect(lambda value: self._progress_bar.
↪setValue(value))
        self._install_thread.text.connect(lambda text: self._progress_bar.
↪setFormat(text))
        self._install_thread.finished.connect(self._install_thread_finished)

        self._path_browse_button.clicked.connect(self._path_browse_button_clicked)
```

(continues on next page)

(continued from previous page)

```
        self._install_single_thread_button.clicked.connect(self._install_minecraft_
↪single_thread)
        self._install_multi_thread_button.clicked.connect(self._install_minecraft_multi_
↪thread)

        path_layout = QHBoxLayout()
        path_layout.addWidget(self._path_edit)
        path_layout.addWidget(self._path_browse_button)

        form_layout = QFormLayout()
        form_layout.addRow(QLabel("Version:"), self._version_combo_box)
        form_layout.addRow(QLabel("Path:"), path_layout)

        button_layout = QHBoxLayout()
        button_layout.addWidget(self._install_single_thread_button)
        button_layout.addWidget(self._install_multi_thread_button)

        main_layout = QVBoxLayout()
        main_layout.addLayout(form_layout)
        main_layout.addWidget(self._progress_bar)
        main_layout.addLayout(button_layout)

        self.setLayout(main_layout)
        self.setWindowTitle("PyQtInstallation")

    def _install_thread_finished(self) -> None:
        # This function is called after the Multi Thread Installation has been finished
        self._install_single_thread_button.setEnabled(True)
        self._install_multi_thread_button.setEnabled(True)

    def _path_browse_button_clicked(self) -> None:
        path = QFileDialog.getExistingDirectory(self, directory=self._path_edit.text())
        if path != "":
            self._path_edit.setText(path)

    def _install_minecraft_single_thread(self) -> None:
        # This function installs Minecraft in the same Thread as the GUI
        # This is much simpler than using a other Thread, but the GUI will freeze until␣
↪the function is completed
        callback = {
            "setStatus": lambda text: self._progress_bar.setFormat(text),
            "setProgress": lambda value: self._progress_bar.setValue(value),
            "setMax": lambda maximum: self._progress_bar.setMaximum(maximum)
        }

        minecraft_launcher_lib.install.install_minecraft_version(self._version_combo_box.
↪currentText(), self._path_edit.text(), callback=callback)

    def _install_minecraft_multi_thread(self) -> None:
        # This functions installs Minecraft on a other Thread than the GUI
        # This is more complex than using the same Thread, but the GUI will not freeze
        self._install_single_thread_button.setEnabled(False)
```

(continues on next page)

(continued from previous page)

```
        self._install_multi_thread_button.setEnabled(False)

        self._install_thread.set_data(self._version_combo_box.currentText(), self._path_
↪edit.text())
        self._install_thread.start()


def main():
    app = QApplication(sys.argv)

    w = Window()
    w.show()

    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```

View this example on Codeberg

6.5 InstallationProgress

```
#!/usr/bin/env python3
# This example shows how to show the progress of installation to the user.
import minecraft_launcher_lib


# Taken from https://stackoverflow.com/questions/3173320/text-progress-bar-in-the-console
def printProgressBar(iteration, total, prefix='', suffix='', decimals=1, length=100,␣
↪fill='', printEnd="\r"):
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration / float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print('\r%s |%s| %s%% %s' % (prefix, bar, percent, suffix), end=printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()
```

(continues on next page)

(continued from previous page)

```
def maximum(max_value, value):
    max_value[0] = value


def main():
    # lambda doesn't allow setting vars, so we need this little hack
    max_value = [0]

    callback = {
        "setStatus": lambda text: print(text),
        "setProgress": lambda value: printProgressBar(value, max_value[0]),
        "setMax": lambda value: maximum(max_value, value)
    }

    version = minecraft_launcher_lib.utils.get_latest_version()["release"]
    directory = minecraft_launcher_lib.utils.get_minecraft_directory()

    minecraft_launcher_lib.install.install_minecraft_version(version, directory,␣
↪callback=callback)


if __name__ == "__main__":
    main()
```

View this example on Codeberg

6.6 SimpleLaunch

```
#!/usr/bin/env python3
# This example shows how to simple install and launch the latest version of Minecraft.
import minecraft_launcher_lib
import subprocess
import sys

# Set the data for your Azure Application here. For more information look at the␣
↪documentation.
CLIENT_ID = "YOUR CLIENT ID"
REDIRECT_URL = "YOUR REDIRECT URL"

# Get latest version
latest_version = minecraft_launcher_lib.utils.get_latest_version()["release"]

# Get Minecraft directory
minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()

# Make sure, the latest version of Minecraft is installed
minecraft_launcher_lib.install.install_minecraft_version(latest_version, minecraft_
↪directory)

# Login
```

(continues on next page)

(continued from previous page)

```
login_url, state, code_verifier = minecraft_launcher_lib.microsoft_account.get_secure_
↪login_data(CLIENT_ID, REDIRECT_URL)
print(f"Please open {login_url} in your browser and copy the url you are redirected into␣
↪the prompt below.")
code_url = input()

# Get the code from the url
try:
    auth_code = minecraft_launcher_lib.microsoft_account.parse_auth_code_url(code_url,␣
↪state)
except AssertionError:
    print("States do not match!")
    sys.exit(1)
except KeyError:
    print("Url not valid")
    sys.exit(1)

# Get the login data
login_data = minecraft_launcher_lib.microsoft_account.complete_login(CLIENT_ID, None,␣
↪REDIRECT_URL, auth_code, code_verifier)

# Get Minecraft command
options = {
    "username": login_data["name"],
    "uuid": login_data["id"],
    "token": login_data["access_token"]
}
minecraft_command = minecraft_launcher_lib.command.get_minecraft_command(latest_version,␣
↪minecraft_directory, options)

# Start Minecraft
subprocess.run(minecraft_command, cwd=minecraft_directory)
```

View this example on Codeberg

6.7 PyQtLauncherWithOutput

```
#!/usr/bin/env python3
# This example shows a simple launcher with PyQt that shows the output of Minecraft in a␣
↪text filed
from PyQt6.QtWidgets import QApplication, QWidget, QPlainTextEdit, QLabel, QLineEdit,␣
↪QPushButton, QComboBox, QMessageBox, QHBoxLayout, QVBoxLayout
from PyQt6.QtCore import QProcess
import minecraft_launcher_lib
import sys


class GameOutputWidget(QPlainTextEdit):
    def __init__(self, launch_button):
        super().__init__()
        self.setReadOnly(True)
```

(continues on next page)

(continued from previous page)

```
        self.launch_button = launch_button
        # Disable line wrap
        self.setLineWrapMode(QPlainTextEdit.LineWrapMode.NoWrap)

    def display_output(self):
        # This function displays the output of Minecraft in the text field
        cursor = self.textCursor()
        cursor.movePosition(cursor.MoveOperation.End)
        cursor.insertText(bytes(self.process.readAll()).decode())
        cursor.movePosition(cursor.MoveOperation.End)
        self.ensureCursorVisible()

    def execute_command(self, command):
        # QProcess.start takes as first argument the program and as second the list of␣
↪arguments
        # So we need the filter the program from the command
        arguments = command[1:]
        # Deactivate the launch button
        self.launch_button.setEnabled(False)
        # Clear the text  field
        self.setPlainText("")
        self.process = QProcess(self)
        # Activate the launch button when Minecraft is closed
        self.process.finished.connect(lambda: self.launch_button.setEnabled(True))
        # Connect the function to display the output
        self.process.readyRead.connect(self.dataReady)
        # Start Minecraft
        self.process.start("java", arguments)


class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.username_edit = QLineEdit()
        self.password_edit = QLineEdit()
        self.version_select = QComboBox()
        launch_button = QPushButton("Launch")
        self.output_widget = GameOutputWidget(launch_button)

        # Set the password field to display *
        self.password_edit.setEchoMode(QLineEdit.EchoMode.Password)

        launch_button.clicked.connect(self.launch_minecraft)

        # Add all versions to the Version ComboBox
        self.minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()
        for i in minecraft_launcher_lib.utils.get_available_versions(self.minecraft_
↪directory):
            # Only add release versions
            if i["type"] == "release":
                self.version_select.addItem(i["id"])
```

(continues on next page)

(continued from previous page)

```
        # Create the layouts
        bottom_layout = QHBoxLayout()
        bottom_layout.addWidget(QLabel("Username:"))
        bottom_layout.addWidget(self.username_edit)
        bottom_layout.addWidget(QLabel("Password:"))
        bottom_layout.addWidget(self.password_edit)
        bottom_layout.addWidget(QLabel("Version:"))
        bottom_layout.addWidget(self.version_select)
        bottom_layout.addWidget(launch_button)

        main_layout = QVBoxLayout()
        main_layout.addWidget(self.output_widget)
        main_layout.addLayout(bottom_layout)

        self.setLayout(main_layout)
        self.setGeometry(0, 0, 800, 600)
        self.setWindowTitle("PyQt Launcher with Output")

    def launch_minecraft(self):
        # Get the selected version
        version = self.version_select.currentText()

        # Make sure the version is installed
        minecraft_launcher_lib.install.install_minecraft_version(version, self.minecraft_
↪directory)

        # Login
        login_data = minecraft_launcher_lib.account.login_user(self.username_edit.text(),
↪ self.password_edit.text())

        # Check if the login is correct
        if "errorMessage" in login_data:
            message_box = QMessageBox()
            message_box.setWindowTitle("Invalid credentials")
            message_box.setText("Invalid username or password")
            message_box.setStandardButtons(QMessageBox.StandardButtons.Ok)
            message_box.exec()
            return

        options = {
            "username": login_data["selectedProfile"]["name"],
            "uuid": login_data["selectedProfile"]["id"],
            "token": login_data["accessToken"]
        }

        # Get the command
        command = minecraft_launcher_lib.command.get_minecraft_command(version, self.
↪minecraft_directory, options)

        # Call the function from the
        self.output_widget.execute_command(command)
```

(continues on next page)

(continued from previous page)

```
def main():
    app = QApplication(sys.argv)
    w = MainWindow()
    w.show()
    sys.exit(app.exec())


if __name__ == "__main__":
    main()
```

View this example on Codeberg

6.8 Mrpack

```
#!/usr/bin/env python3
# This example shows how use the mrpack module
import minecraft_launcher_lib
import subprocess
import sys
import os


def ask_yes_no(text: str) -> bool:
    while True:
        answer = input(f"{text} [Y/N]: ").strip().upper()

        if answer == "Y":
            return True
        elif answer == "N":
            return False
        else:
            print("Invalid answer. Use Y or N.")


def main() -> None:
    mrpack_path = input("Please enter the Path to your .mrpack File: ")

    if not os.path.isfile(mrpack_path):
        print(f"{mrpack_path} was not found", file=sys.stderr)
        sys.exit(1)

    try:
        mrpack_information = minecraft_launcher_lib.mrpack.get_mrpack_information(mrpack_
↪path)
    except Exception:
        print(f"{mrpack_path} is not a valid .mrpack File")
        sys.exit(1)

    # Print some Information
    print("You have selected the following Pack:")
```

(continues on next page)

(continued from previous page)

```
    print("Name: " + mrpack_information["name"])
    print("Summary: " + mrpack_information["summary"])
    print("Minecraft version: " + mrpack_information["minecraftVersion"])

    if not ask_yes_no("Do you want to install this Pack?"):
        return

    # Ask the User for the Directories
    minecraft_directory = input("Please enter the Path to your Minecraft directory␣
↪(leave empty for default): ")

    if minecraft_directory == "":
        minecraft_directory = minecraft_launcher_lib.utils.get_minecraft_directory()

    modpack_directory = input("Please enter the Path to the Directory you want to␣
↪install the Modpack (leave empty for your Minecraft directory): ")

    if modpack_directory == "":
        modpack_directory = minecraft_directory

    # Adds the Optional Files
    mrpack_install_options: minecraft_launcher_lib.types.MrpackInstallOptions = {
↪"optionalFiles": []}
    for i in mrpack_information["optionalFiles"]:
        if ask_yes_no(f"The Pack includes the Optional File {i}. Do you want to install␣
↪it?"):
            mrpack_install_options["optionalFiles"].append(i)

    # Install
    print("Installing")
    minecraft_launcher_lib.mrpack.install_mrpack(mrpack_path, minecraft_directory,␣
↪modpack_directory=modpack_directory, mrpack_install_options=mrpack_install_options,␣
↪callback={"setStatus": print})
    print("Finished")

    if not ask_yes_no("Do you want to start Minecraft?"):
        return

    # We skip the Login in this Example
    options = minecraft_launcher_lib.utils.generate_test_options()
    options["gameDirectory"] = modpack_directory
    command = minecraft_launcher_lib.command.get_minecraft_command(minecraft_launcher_
↪lib.mrpack.get_mrpack_launch_version(mrpack_path), minecraft_directory, options)
    subprocess.run(command)


if __name__ == "__main__":
    main()
```

View this example on Codeberg

CHAPTER

SEVEN

TROUBLESHOOTING

Here is a quick checklist for you, if Minecraft doesn't starts.

- Start the Version in the official Launcher

  At first you should try to start the Minecraft version in official Launcher. If it didn't start there, the problem is not in minecraft-launcher-lib or your code.

- Call install_minecraft_version()

  Before you start Minecraft, you should first call *`install_minecraft_version()`*. This function ensures that everything is right and installs parts if needed. It should even be called, if you installed a version with a installer e.g. Optifine.

- Use the subprocess module

  There are many ways to call a shell command in Python. Maybe your GUI Tooolkit brings it's own function e.g. QProcess from Qt. But nor all of them supports a list of strings. To make sure this is not the problem try to start the command with `subproces.run(command)`. Do not use `os.system()`.

- Check the Java version

  While newer versions of Minecraft has a Java runtime version in it's client.json, older versions don't have it. Some older versions requires a older Java version. Make sure to launch this old versions with the right Java version. Check out the *command* module documentation to learn how to set the Java version.

- Check your JVM Arguments

  If you use custom JVM Arguments, make sure all of them start with `-` and not a other char e.g. a whitespace.

If all of the steps above failed, please fill a Bug Report.

CHAPTER

EIGHT

GLOSSARY

This page contains commonly used terms in the documentation along with their explanations.

- **Minecraft Directory**
  This is the directory where Minecraft is installed. It typically contains the `assets`, `libraries`, `runtime`, and `versions` directories.
- **Game Directory**
  This is the directory where Minecraft saves its data (e.g., worlds). It usually contains the `saves` and `resourcepacks` directories, as well as an `options.txt` file. By default, the Game Directory is the same as the Minecraft Directory.
- **Version**
  This refers to a specific version of Minecraft. It is installed in `<minecraft_directory>/versions`.
- **client.json**
  This is a JSON file that stores information about a specific Minecraft version. It is located in `<minecraft_directory>/versions/<version>/client.json`. For a detailed explanation, refer to the Minecraft Wiki .
- **Runtime**
  This refers to the Java Runtime used by Minecraft. The runtimes that come with Minecraft are installed in `<minecraft_directory>/runtimes`.

CHAPTER

NINE

CONTRIBUTE

Many thanks for contributing to minecraft-launcher-lib! You can do the following thinks:

- You can test minecraft-launcher-lib with all versions and write a Bug report if something did not work
- You can help developing minecraft-launcher-lib. Check out the *Develop* section of the documentation.
- You can can improve the documentation. See *Build and edit documentation*.
- You can also update the Examples and add new one.

CHAPTER

TEN

DEVELOP

10.1 Testing changes

While there are *automatic tests* for the utils functions, the main part (installing launching and logging in into Microsoft) must be tested by yourself.

Open a command line in the root directory of minecraft-launcher-lib and open the Python Interpreter. In the Interpreter run:

```
>>> import minecraft_launcher_lib
>>> print(minecraft_launcher_lib.__file__)
```

It should print the path to your current directory. After you've confirmed, that you are using the version you are currently working on, you can start testing.

Just run the functions you have changed and see, if everything worked correctly. Please test installation and launching in a clean directory and not your normal .minecraft directory.

10.2 Codestyle

minecraft-launcher-lib uses PEP8 as it's codestyle with the following additional rules:

- Lines longer than 80 chars are allowed. We are not in the 90s anymore. Any modern screen can display way more than 80 chars per line without scrolling.
- All functions must have type annotations
- All functions must have docstrings

10.2.1 Check the Codestyle

minecraft-launcher-lib uses flake8 along with the flake8-annotation, the flake8-docstring-checker plugin, and the flake8-assert-finder plugin to do a automatic style check. To get started, install it:

```
pip install flake8 flake8-annotation flake8-docstring-checker flake8-assert-finder
```

To run it, open a command line in the root directory of minecraft-launcher-lib and run:

```
flake8
```

If it prints nothing, everything is OK. If it prints something, you should fix it.

10.3 Automatic tests

minecraft-launcher-lib uses Pytest to run some automatic tests.

10.3.1 Using Pytest

To get started, install all test dependencies

```
pip install -r requirements-test.txt
```

To run the tests, open a command line in the root directory of minecraft-launcher-lib and execute:

```
pytest
```

If a test fails, you should fix the bug.

10.3.2 Test Coverage

To see a detailed test coverage report (which lines are executed during the tests), open `htmlcov/index.html` with your browser.

10.4 Static typing

minecraft-launcher-lib uses mypy to enforce static typing.

10.4.1 Using mypy

To get started, install mypy together with requests and the types for requests:

```
pip install requests types-requests mypy
```

To run mypy open a command line in the root directory of minecraft-launcher-lib and execute:

```
mypy minecraft_launcher_lib
```

If mypy shows a error,or, you should fix it.

10.5 Build and edit documentation

minecraft-launcher-lib uses Sphinx for it's documentation. The documentation is hosted on Read the Docs. You can find the source files for the documentation in doc folder.

10.5.1 Plugins

minecraft-launcher-lib uses the following Sphinx plugins:

- sphinx-reredirects: The module documentation has been moved in the modules directory. This Plugin creates redirects, so older links will still work.
- sphinx-notfound-page: Used for setting the custom 404 page.
- sphinx-rtd-dark-mode: Provides the dark theme, that you can turn on by clicking on the button in the bottom right corner.
- sphinx-copybutton: Provides the copy button on the code blocks

- sphinxext-opengraph: Generates Open Graph metadata

These plugins are all completely optional. You can build the documentation without having these plugins installed. The features, that the plugins provide are missing in that case.

10.5.2 Building

First you need to install Sphinx and the Read the Docs theme:

```
pip install sphinx sphinx-rtd-theme
```

You can also install the plugins, if you want to e.g. test the dark mode.

To build the documentation open a command line in the doc folder and run:

```
# Unix based Systems
make html
# Windows
.\make.bat html
```

Now you can view the documentation by opening _build/html/index.html in your favorite browser.

10.5.3 Examples

The examples are stored in the examples folder. They are added to the documentation during build. Please do not edit anything inside doc/examples. All files in this folder are overwritten during the build.

10.6 Making a Pull Request

minecraft-launcher-lib uses Codeberg as development platform. If you want to contribute some changes, a need to make a Pull Request (PR) against the Codeberg repo of minecraft-launcher-lib. A merge request is the same as a Pull Request on GitHub, which I think you should familiar with. It's works exactly the same way.

Before making a MR, you should follow this checklist:

- minecraft-launcher-lib currently targets version 3.8 of Python and the latest version of PyPy. Make sure, you don't use features that were added in a newer Python version.
- Your code should work on Linux, Mac and Windows
- Please make sure, you follow the *Codestyle*
- If possible, you should not break existing code that uses minecraft-launcher-lib
- If you can write a test, for your changes, you should do it
- You should update the *documentation* with your changes
- Please don't add extra dependencies if not absolutely needed

After you've created a PR, flake8 and Pytest with all supported versions (including PyPy) will run. If one of those fails, Codeberg will show it on the Website and you will get a mail.

Read the Docs is configured to build the documentation for each MR. Unfortunately, it will not provide a link or any hint, if the build was successful. You will have to visit this site and search for your MR.

I'm looking forward to see your contribution and thanks in advance!

CHAPTER

ELEVEN

SHOWCASE

Here is a List of programs that uses minecraft-launcher-lib. If you want to add yours, feel free to make a PR on Codeberg.

- jdMinecraftLauncher
- minecraft-launcher-cmd

CHAPTER

TWELVE

MIGRATION GUIDE

minecraft-launcher-lib strives to maintain a backwards-compatible API. However, because it is built around Minecraft and its ecosystem, breaking changes are occasionally necessary to accommodate upstream changes.

In most cases, old functions are deprecated rather than removed, so upgrading to the latest release and performing migrations later is possible.

Always use the latest version of minecraft-launcher-lib to ensure compatibility with newer Minecraft versions and to receive bug fixes.

The *changelog* lists all changes; this page summarizes the actions developers must take when upgrading their code. If an API change is missing from this page, treat it as a bug and report it.

12.1 8.0

- The `forge`, `fabric` and `quilt` modules have been replaced by the new `mod_loader` module. The old modules still exist but will no longer receive bug fixes and may be removed in a future release.

CHAPTER

THIRTEEN

CHANGELOG

13.1 8.0

- Added `mod_loader` module (#177)
- Deprecated `forge` module (#177)
- Deprecated `fabric` module (#177)
- Deprecated `quilt` module (#177)
- Hide cmd on Windows (izharus in #148)

13.2 7.1

- Fixed tempfile handling on Windows (#144)

13.3 7.0

- The minimum Python version was increased from 3.8 to 3.10
- The installation is now multithreaded (izharus in #139)
- Duplicated Assets are no longer downloaded (izharus in #137)
- `utils.get_minecraft_news()` was removed without deprecation period, as Mojang shut down the API endpoint (#131)
- Added `news` module as replacement for `utils.get_minecraft_news()`
- Fixed launching of Fabric 1.21 (#143)
- Fixed Bug with install mrpack (#124)

13.4 6.5

- Add *get_version_runtime_information()*
- Fix *extract_natives()* for newer versions
- Fix installation of Forge 1.20.4
- Add examples to documentation

13.5 6.4

- Added *AccountNotOwnMinecraft* exception

13.6 6.3

- Fixed #92
- Fixed #93

13.7 6.2

- Fix raising InvalidChecksum exception

13.8 6.1

- Added *AzureAppNotPermitted* exception
- Added *is_minecraft_installed()*
- Added callbacks to the *mrpack* module
- Fix some Bugs

13.9 6.0

- Added *vanilla_launcher* module
- Added *mrpack* module
- Added *quilt* module
- Added *get_jvm_runtime_information()*
- Added *InvalidChecksum* exception
- Remove account module (deprecated since 4.4 which was released 2022-02-16)
- Move module documentation into Code
- Change account type to msa
- Add support for Quick Play
- Add internal types
- Refactor Code

13.10 5.3

- Move minecraft-launcher-lib to Codeberg
- Add defaultExecutablePath option
- Add disableMultiplayer and disableChat options
- Change get_java_executable to use javaw.exe on Windows (osfanbuff63)

13.11 5.2

- Added a secure login option using pkce (get_secure_login_data)(Manuel Quarneti)
- Add forge_to_installed_version()
- Fix setMax callback

13.12 5.1

- Fix crash when custom clients use invalid releaseTime

13.13 5.0

- The minimum Python version is now 3.8
- All public APIs are now complety static typed (with help of Manuel Quarneti)
- minecraft-launcher-lib has now a py.typed file
- Installs now using requests.session for faster installing
- Add types and microsoft_types module
- Add is_platform_supported()
- Add get_installed_jvm_runtimes()
- The client secret is now optional for Microsoft Accounts
- Include release time in version list
- install_jvm_runtime() does now support symlinks
- Fix launching custom clients

13.14 4.6

- Add is_vanilla_version()
- Install version that is inherited from
- Fix command for 1.19-pre1
- Fix type annotations
- Cache requests
- Rewrite Maven parsing

13.15 4.5

- Fix Forge installation for 1.18 again (catnip)

13.16 4.4

- Fix Forge installation for 1.18
- Do not use bare except

- Add DeprecationWarning to the account module

13.17 4.3

- Add get_executable_path()
- Fix using Java Runtime on Windows

13.18 4.2

- Fix launching Forge 1.17.1

13.19 4.1

- Add get_minecraft_news()
- Replace deprecated distutils.spawn.find_executable() with shutil.which()
- Add support for using a custom Java runtime in different functions (BobDotCom)
- Fix Forge for 1.12.2
- Fix find_forge_version() (BobDotCom)
- Packages can now be built without requests being installed (BobDotCom)
- Fix finding Java runtime on Mac (BobDotCom)

13.20 4.0

- Add Support for Microsoft Accounts
- All functions with a Path as Argument can now take a os.PathLike
- Fix crash in get_installed_versions() when a directory has no json file
- Fix Bug in install_forge_version()

13.21 3.6

- Fix install_forge_version() for 1.17.1

13.22 3.5

- Fix crash when logging is empty

13.23 3.4

- Add runtime module
- The runtime is now automatic installed if needed

13.24 3.3

- Add is_forge_version_valid()
- Add supports_automatic_install()
- Add UnsupportedVersion exception
- Add ExternalProgramError exception
- Add callbacks to install_fabric()
- Make install_forge_version() raise VersionNotFound exception
- Fix install_fabric()
- Better codestyle

13.25 3.2

- Use custom user agent for all requests
- Fix typo that causes crash (DiamondsBattle)

13.26 3.1

- Fix Bug in install_minecraft_version()

13.27 3.0

- Add fabric module
- install_minecraft_version supports now custom libraries urls
- Add VersionNotFound exception
- Add type annotations
- Add docstrings
- Add is_version_valid()
- Add generate_test_options()

13.28 2.1

- Add support for log4j configuration file
- Fix Bug with files in versions directory

13.29 2.0

- Add forge modul
- Add hash validation

13.30 1.4

- Fix downloading libraries on windows

13.31 1.3

- Fix downloading libraries without url
- Fix get_available_versions()
- Improve get_java_executable()

13.32 1.2

- Fix Typo

13.33 1.1

- Fix Forge for older versions

13.34 1.0

- Add function to extract natives
- Add functions for upload and reset a skin

13.35 0.5

- Better support for older versions
- Add new functions to utils

13.36 0.4

- The natives are now extracted
- Fix running older versions of Forge

13.37 0.3

- The classpath has now the correct seperator on windows
- Add option to set the executable path
- Add support for {arch} in natives

13.38 0.2

- Add support for Forge
- Add more options
- Add callback functions

13.39 0.1

- First Release

CHAPTER

FOURTEEN

HISTORY

I like the Design of the old Minecraft Launcher (2013-2016), but I don't like the Design of the new one which was released in 2016, so I kept using the old one. I was aware that the old Launcher is not going to be supported forever. There are many 3rd party launcher out there so I thought: Why not writing my own Launcher that has the same GUI as the old Launcher? So I started developing a Launcher in Python using QtWidgets from PyQt5. I chose Python, because it's an easy to use language. QtWidgets also allows me to write a classical styles user interface and is cross platform as a Bonus. I finished the GUI soon. Everything except the launching was working. I looked how t do that, but is was complicated and I run into a few problems, so I was looking for a library to do that or working Python code that supports all edge cases. I haven't found anything that was useable, so I decided to use mclauncher-api, which is a Java library. I wrote a small wrapper program in Java that uses the lib and was called by my Launcher. I released it and called it jdMinecraftLauncher, so it can also used by other who like the good old design.

I knew, this was not a good solution so I kept working on a Python code that can launch and install Minecraft. I was able to archive this and install and launch the latest version at this time. I tried to use the code with older versions, but I realized that the Way how to launch Minecraft changed a bit over time. I also got this part working. I remembered, how hard it was for me to do this, so I decided to move all the functions that could also be used by others out of jdMinecraftLauncher into a library. I am not very creative when it comes to names, so I just called the library minecraft-launcher-lib, because it's a library for Minecraft launchers. minecraft-launcher-lib was primarily made to meet the needs of jdMinecraftLauncher, but I designed to library in a Way that it could be used by anyone out there and with any GUI toolkit.

Version 0.1 was published on 2019-11-18 on GitLab together with a very minimalistic documentation and uploaded to PyPI.

I kept working on it and added support for more and more Versions. I also tested the library more, as it was now integreated in jdMinecraftLauncher. Version 1.0, which finally official supports all existing Versions was released on 2020-05-16.

After that, I added functions for automatically installing Forge, which was a lot of reverse engineering, and Farbric, which is just downloading and executing the Installer. I also improved the Documentation and added tests using Pytest.

A big change came with the introduction of Microsoft Accounts. WIth old Mojang Accounts you just needed a Username and Password to log in. Now you need to create a Azure App before you can get started. The login also needs to be done in a Web browser. This causes a lot of Problems.

It's gotten even worse: Since 2023 you need to apply to get the Permission to use the Minecraft API. this makes it way harder for people who want to try out this library.

Another big change happened in 2023: GitLab announces that inactive Repos will be deleted after one Year. This was reversed later, but I felt like GitLab, which I was using since Microsoft bought GitHub, is no longer a good and safe place for my Projects. So I looked for another Code hosting service and found Codeberg, which is non profit hosting service for OpenSource Projects. After GitLab was blocking the CI for new Users, I decided that it was time to ditch GitLab and moved the Repo to Codeberg.

PYTHON MODULE INDEX

m

INDEX

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X