CrazyHusk

Release 0.1.6.dev57+g7aa13b6

Nick Haines

Jul 27, 2022

CONTENTS

Dependency-free Python object wrappers for working with Unreal Engine

CHAPTER

ONE

FEATURES

- TODO

CHAPTER

TWO

REQUIREMENTS

- TODO

CHAPTER

THREE

INSTALLATION

You can install *CrazyHusk* via pip from PyPI:

```
$ pip install crazyhusk
```

CHAPTER

FOUR

USAGE

Please see the *Command-line Reference* for details.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

SIX

LICENSE

Distributed under the terms of the *MIT license*, *CrazyHusk* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

7.1 Usage

7.2 Reference

7.2.1 crazyhusk

Initializes the crazyhusk package on import.

crazyhusk.build

Wrapper objects for Unreal Engine builds.

class crazyhusk.build.**Buildable**

Abstract base class for objects buildable by Unreal's build tools.

default_build_configuration()

Get the default build configuration for this Buildable.

Return type
str

default_build_target()

Get the default build target for this Buildable.

Return type
str

default_local_platform()

Get the default build platform for the local system.

Return type
str

abstract property engine: Optional[*UnrealEngine*]

Get the associated UnrealEngine object for this Buildable.

abstract get_build_command(*target=None*, *configuration=None*, *platform=None*, **extra_switches*, ***extra_parameters*)

Iterate strings of subprocess arguments to execute the build.

Parameters

- **target** (*Optional[str]*) –
- **configuration** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
Iterable[str]

abstract is_buildable()

Get whether this object is buildable in its current configuration.

Return type
bool

is_valid_build_configuration(*configuration*)

Get whether a given build configuration is valid for this Buildable.

Parameters
configuration (*str*) –

Return type
bool

is_valid_build_platform(*platform*)

Get whether a given platform is valid for this Buildable.

Parameters
platform (*str*) –

Return type
bool

is_valid_build_target(*target*)

Get whether a given build target is valid for this Buildable.

Parameters
target (*str*) –

Return type
bool

is_valid_static_analyzer(*static_analyzer*)

Get whether a given c++ static analyzer is valid for this Buildable.

Parameters
static_analyzer (*str*) –

Return type
bool

class crazyhusk.build.**UnrealBuild**(*buildable*, *target=None*, *configuration=None*, *build_platform=None*, *static_analyzer=None*)

Object wrapper for composing and running an Unreal build subroutine.

Parameters

- **buildable** (Buildable) –
- **target** (*Optional[str]*) –
- **configuration** (*Optional[str]*) –
- **build_platform** (*Optional[str]*) –
- **static_analyzer** (*Optional[str]*) –

property **configuration: str**

Get the build configuration for this UnrealBuild.

property **platform: str**

Get the build platform for this UnrealBuild.

run(**extra_switches*, ***extra_parameters*)

Execute the currently configured build subprocess for this UnrealBuild.

Parameters

- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

property **static_analyzer: Optional[str]**

Get the c++ static analyzer platform for this UnrealBuild.

property **target: str**

Get the build target for this UnrealBuild.

crazyhusk.cli

Expose crazyhusk functionality to the commandline.

exception crazyhusk.cli.**CommandError**

Custom exception representing errors encountered with CLI.

crazyhusk.cli.**parse_cli_args**(*args*)

Parse crazyhusk CLI arguments.

Parameters
args (*List[str]*) –

Return type
Namespace

crazyhusk.cli.**run**(*args=['-T', '-E', '-b', 'readthedocssinglehtmllocalmedia', '-d', '_build/doctrees', '-D', 'language=en', '.', '_build/localmedia']*)

Run the crazyhusk CLI entrypoint.

Parameters
args (*List[str]*) –

Return type
None

crazyhusk.cli.**set_subcommand_arguments**(*parser*, *command*)

Dynamically set argparse.Parser subcommand arguments by inspecting a callable function.

Parameters

- **parser** (*ArgumentParser*) –
- **command** (*Any*) –

Return type
ArgumentParser

crazyhusk.code

Wrapper objects for Unreal code templates.

class crazyhusk.code.**CodeTemplate**(*name*, *template_string=''*)

Object wrapper for working with Unreal's code templating system for C++.

Parameters

- **name** (*str*) –
- **template_string** (*str*) –

make_instance(***tokens*)

Create a templated string using the supplied tokens with this CodeTemplate.

Parameters
tokens (*str*) –

Return type
str

property tokens: Set[str]

Get the set of string replacement tokens expressed by this CodeTemplate.

exception crazyhusk.code.**CodeTemplateError**

Custom exception representing errors encountered with CodeTemplate.

crazyhusk.config

Object wrappers for working with Unreal Engine config files.

exception crazyhusk.config.**UnrealConfigError**

Custom exception representing errors encountered with Unreal config files.

class crazyhusk.config.**UnrealConfigParser**

Object wrapper representing a configuration stack.

optionxform(*optionstr*)

Transform the string used by ConfigParsers for use with key expression of options.

Parameters
optionstr (*str*) –

Return type
str

crazyhusk.engine

Object wrappers for working with Unreal Engine installations.

class crazyhusk.engine.**UnrealEngine**(*base_dir*, *association_name=None*)

Object wrapper representing an Unreal Engine.

Parameters

- **base_dir** (*str*) –
- **association_name** (*Optional[str]*) –

property build_dir: str

Path to this Engine's Build directory.

property build_targets: Dict[str, str]

Get a mapping of this UnrealEngine's available build targets.

property build_type: Optional[str]

Type of build available for this Engine.

property code_templates: Dict[str, *CodeTemplate*]

Get a mapping of this UnrealEngine's available C++ code templates.

config(*config_category=None*, *platform=None*)

Create a configuration object associated with this engine by category and platform.

Parameters

- **config_category** (*Optional[str]*) –
- **platform** (*Optional[str]*) –

Return type
UnrealConfigParser

property config_dir: str

Path to this Engine's Config directory.

config_files(*config_category=None*, *platform=None*)

Iterate configuration file paths associated with this engine by category and platform.

Parameters

- **config_category** (*Optional[str]*) –
- **platform** (*Optional[str]*) –

Return type
Iterable[str]

property content_dir: str

Path to this Engine's Content directory.

default_build_target()

Get the default build target for this Buildable.

Return type
str

property engine: Optional[*UnrealEngine*]

Get the associated UnrealEngine object for this Buildable.

property engine_dir: str

Path to this Engine's Engine directory.

static engine_dir_exists(*engine*)

Raise exception if this instance is not available on disk.

Parameters
engine (UnrealEngine) –

Return type
None

static engine_exe_common_path(*engine*, *executable*, **args*)

Raise exception if the executable does not resolve to a path owned by the given engine.

Parameters

- **engine** (UnrealEngine) –
- **executable** (*str*) –
- **args** (*str*) –

Return type
None

static engine_exe_exists(*engine*, *executable*, **args*)

Raise exception if the executable is not available on disk.

Parameters

- **engine** (UnrealEngine) –
- **executable** (*str*) –
- **args** (*str*) –

Return type
None

executable_path(*executable_name*)

Resolve an expected real path for an executable member of this engine for a given executable name.

Parameters
executable_name (*str*) –

Return type
Optional[str]

property feature_packs_dir: str

Path to this Engine's FeaturePacks directory.

static find_engine(*association*)

Find an engine distribution from EngineAssociation string.

Parameters
association (*str*) –

Return type
Optional[UnrealEngine]

static format_commandline_options(**switches*, ***parameters*)

Convert input arguments from Pythonic expansions to commandline strings.

Parameters

- **switches** (*str*) –
- **parameters** (*str*) –

Return type
Iterable[str]

get_build_command(*target=None*, *configuration=None*, *platform=None*, **extra_switches*, ***extra_parameters*)

Get the default build configuration for this Buildable.

Parameters

- **target** (*Optional[str]*) –
- **configuration** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
Iterable[str]

is_buildable()

Get whether this object is buildable in its current configuration.

Return type
bool

is_installed_build()

Determine if this engine is an Installed distribution.

Return type
bool

is_source_build()

Determine if this engine is a Source distribution.

Return type
bool

is_valid_build_target(*target*)

Get whether a given build target is valid for this Buildable.

Parameters
target (*str*) –

Return type
bool

static list_all_engines()

List all available engine installations.

Return type
Iterable[UnrealEngine]

static list_engine_code_templates(*engine*)

Iterate over a given UnrealEngine's available C++ code templates.

Parameters
engine (UnrealEngine) –

Return type
Iterable[CodeTemplate]

static log_engine_list()

Log all found engines.

Return type
None

property plugins: Optional[Dict[str, *UnrealPlugin*]]

Get a mapping of the available plugins installed with this Engine.

property plugins_dir: str

Path to this Engine's Plugins directory.

run(*executable*, **args*, *expected_retcodes=None*)

Run an associated Unreal executable in a subprocess, and process output line by line.

Parameters

- **executable** (*str*) –
- **args** (*str*) –
- **expected_retcodes** (*Optional[Set[int]]*) –

Return type
int

run_commandlet(*commandlet*, **extra_switches*, ***extra_parameters*)

Run a commandlet for this project.

Parameters

- **commandlet** (*UnrealCommandlet*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

property samples_dir: str

Path to this Engine's Samples directory.

sanitize_commandline(*executable*, **args*)

Raise exceptions if we are about to run unsafe commands in the subprocess.

Parameters

- **executable** (*str*) –
- **args** (*str*) –

Return type
List[str]

property source_dir: str

Path to this Engine's Source directory.

property templates_dir: str

Path to this Engine's Templates directory.

unreal_path_from_file_path(*file_path*)

Convert a file path to an appropriate Unreal object path for use with this engine.

Parameters
file_path (*str*) –

Return type
str

unreal_path_to_file_path(*unreal_path*, *ext='.uasset'*)

Convert an Unreal object path to a file path relative to this engine.

Parameters

- **unreal_path** (*str*) –
- **ext** (*str*) –

Return type
Optional[str]

validate()

Raise exceptions if this instance is misconfigured.

Return type
None

property version: Optional[UnrealVersion]

Engine version, as UnrealVersion.

exception crazyhusk.engine.**UnrealEngineError**

Custom exception representing errors encountered with UnrealEngine.

crazyhusk.logs

Logging utilities for crazyhusk Unreal Engine object wrappers.

class crazyhusk.logs.**FilterEngineRun**(*executable*, **args*)

Filter to enhance log records when using UnrealEngine.run().

Parameters

- **executable** (*str*) –
- **args** (*str*) –

filter(*record*)

Enhance a loggable record's attributes.

Parameters
record (*Any*) –

Return type
Literal[True]

class crazyhusk.logs.**FilterUBTWarnings**(*name=''*)

Filter to enhance log records generated by UnrealBuildTool.

filter(*record*)

Filter LogRecords emitted from UnrealBuildTool which are warnings/errors/notes.

Parameters
record (*Any*) –

Return type
Literal[True]

class crazyhusk.logs.**FilterUE4Logs**(*name=''*)

Filter to enhance log records generated by UE4Editor/UE4Game.

filter(*record*)

Filter LogRecords emitted from UE4Editor/UE4Game.

Parameters
record (*Any*) –

Return type
Literal[True]

crazyhusk.module

Wrapper objects for Unreal code modules.

class crazyhusk.module.**ModuleDescriptor**

Object wrapper representation of Unreal code module, equivalent to FModuleDescriptor.

https://docs.unrealengine.com/en-US/API/Runtime/Projects/FModuleDescriptor/index.html

find_definition_file(*owner*)

Find a .Build.cs file for this module relative to an owner object's Source directory, if one exists.

Parameters
owner (*Any*) –

Return type
Optional[str]

`is_valid()`

Get wehther this ModuleDescriptor is properly constructed.

Return type
bool

`to_dict()`

Format this ModuleDescriptor as a dictionary for JSON.

Return type
Dict[str, *Any*]

`static to_object(dct)`

Convert a dictionary to an instance of ModuleDescriptor.

Parameters
dct (*Dict[str, Any]*) –

Return type
Union[ModuleDescriptor, *Dict*[str, *Any*]]

crazyhusk.plugin

Wrapper objects for Unreal plugins.

`class crazyhusk.plugin.UnrealPlugin(plugin_file)`

Object wrapper representation of an Unreal Engine plugin.

Parameters
plugin_file (*str*) –

`property code_templates: Dict[str, CodeTemplate]`

Get a mapping of this UnrealPlugin's available C++ code templates.

`property config_dir: str`

Directory path of this plugin's Config.

`property content_dir: str`

Directory path of this plugin's Content.

`property descriptor: PluginDescriptor`

Get an instance of this UnrealPlugin's associated PluginDescriptor.

`property engine: Optional[UnrealEngine]`

Get the associated UnrealEngine object for this Buildable.

`get_build_command(target=None, configuration=None, platform=None, *extra_switches, **extra_parameters)`

Iterate strings of subprocess arguments to execute the build.

Parameters

- **target** (*Optional[str]*) –

- **configuration** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
Iterable[str]

`is_buildable()`

Get whether this object is buildable in its current configuration.

Return type
bool

`static list_plugin_code_templates(plugin)`

Iterate over a given UnrealPlugin's available C++ code templates.

Parameters
plugin (UnrealPlugin) –

Return type
Iterable[CodeTemplate]

`property modules: Dict[str, ModuleDescriptor]`

Get a mapping of this UnrealPlugin's associated ModuleDescriptors.

`property name: str`

Get the name of this UnrealPlugin.

`property plugin_dir: str`

Directory path of this plugin.

`static plugin_file_exists(plugin)`

Raise exception if UnrealPlugin instance is not available on disk.

Parameters
plugin (UnrealPlugin) –

Return type
None

`property plugin_refs: Dict[str, PluginReferenceDescriptor]`

Get a mapping of PluginReferenceDescriptors for this UnrealPlugin.

`property source_dir: str`

Path to this plugin's Source directory.

`unreal_path_from_file_path(file_path)`

Convert a file path to an appropriate Unreal object path for use with this plugin.

Parameters
file_path (*str*) –

Return type
Optional[str]

`unreal_path_to_file_path(unreal_path, ext='.uasset')`

Convert an Unreal object path to a file path relative to this plugin.

Parameters

- **unreal_path** (*str*) –
- **ext** (*str*) –

Return type
Optional[str]

static valid_plugin_file_extension(*plugin*)

Raise exception if UnrealPlugin instance does not have the correct file extension.

Parameters
plugin (UnrealPlugin) –

Return type
None

validate()

Raise exceptions if this instance is misconfigured.

Return type
None

crazyhusk.project

Object wrappers for Unreal projects.

class crazyhusk.project.**UnrealProject**(*project_file*)

Object wrapper representation of an Unreal Engine project.

Parameters
project_file (*str*) –

property code_templates: Dict[str, *CodeTemplate*]

Get a mapping of this UnrealProject's available C++ code templates.

config(*config_category=None*, *platform=None*)

Create a configuration object associated with this project by category and platform.

Parameters

- **config_category** (*Optional[str]*) –
- **platform** (*Optional[str]*) –

Return type
UnrealConfigParser

property config_dir: str

Get the project's Config directory.

config_files(*config_category=None*, *platform=None*)

Iterate configuration file paths associated with this project by category and platform.

Parameters

- **config_category** (*Optional[str]*) –
- **platform** (*Optional[str]*) –

Return type
Iterable[str]

property content_dir: str

Get the project's Content directory.

property descriptor: Optional[ProjectDescriptor]

Get an instance of this UnrealProject's associated ProjectDescriptor.

property engine: Optional[*UnrealEngine*]

Get the associated UnrealEngine object for this Buildable.

get_build_command(*target=None*, *configuration=None*, *platform=None*, **extra_switches*, ***extra_parameters*)

Iterate strings of subprocess arguments to execute the build.

Parameters

- **target** (*Optional[str]*) –
- **configuration** (*Optional[str]*) –
- **platform** (*Optional[str]*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
Iterable[str]

is_buildable()

Get whether this object is buildable in its current configuration.

Return type
bool

static list_project_code_templates(*project*)

Iterate over a given UnrealProject's available C++ code templates.

Parameters
project (UnrealProject) –

Return type
Iterable[CodeTemplate]

list_tests(*editor=True*, **extra_switches*, ***extra_parameters*)

List available automation tests for this project.

Parameters

- **editor** (*bool*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

property modules: Optional[Dict[str, *ModuleDescriptor*]]

Get a mapping of this UnrealProject's associated ModuleDescriptors.

property plugins: Optional[Dict[str, *UnrealPlugin*]]

Get a mapping of the available plugins installed with this UnrealProject.

property plugins_dir: str

Get the project's Plugins directory.

property project_dir: str

Get the base directory for .uproject file.

static project_file_exists(*project*)

Raise exception if UnrealProject instance is not available on disk.

Parameters
project (UnrealProject) –

Return type
None

render(*map_path*, *LevelSequence*, *vsync=False*, **extra_switches*, ***extra_parameters*)

Run this project in movie scene capture mode.

Parameters

- **map_path** (*str*) –
- **LevelSequence** (*str*) –
- **vsync** (*bool*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

property reports_dir: str

Get the project's default Reports directory.

run_commandlet(*commandlet*, **extra_switches*, ***extra_parameters*)

Run a commandlet for this project.

Parameters

- **commandlet** (*UnrealCommandlet*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

run_tests(*tests*, *report_path=None*, *editor=True*, *rhi='nullrhi'*, **extra_switches*, ***extra_parameters*)

Run named automation tests for this project.

Parameters

- **tests** (*List[str]*) –
- **report_path** (*Optional[str]*) –
- **editor** (*bool*) –
- **rhi** (*str*) –
- **extra_switches** (*str*) –
- **extra_parameters** (*str*) –

Return type
int

property saved_dir: str

Get the project's Saved directory.

property source_dir: str

Path to this project's Source directory.

unreal_path_from_file_path(*file_path*)

Convert a file path to an appropriate Unreal object path for use with this project.

Parameters
file_path (*str*) –

Return type
Optional[str]

unreal_path_to_file_path(*unreal_path*, *ext='.uasset'*)

Convert an Unreal object path to a file path relative to this project.

Parameters

- **unreal_path** (*str*) –
- **ext** (*str*) –

Return type
Optional[str]

static valid_project_file_extension(*project*)

Raise exception if UnrealProject instance does not have the correct file extension.

Parameters
project (UnrealProject) –

Return type
None

validate()

Raise exceptions if this instance is misconfigured.

Return type
None

crazyhusk.reports

Utilities for working with report formats generated by Unreal Engine.

crazyhusk.reports.**json_report_to_dict**(*report_file*)

Deserialize Unreal JSON report file into a dictionary.

Parameters
report_file (*str*) –

Return type
Dict[str, *Any*]

crazyhusk.reports.**json_reports_to_junit_xml**(*junit_file*, **json_reports*)

Convert a JSON report from Unreal automation to jUnit XML format.

Parameters

- **junit_file** (*str*) –
- **json_reports** (*str*) –

Return type
None

crazyhusk.reports.**report_entry_to_entry_xml**(*entry*)

Convert Unreal JSON report entry into jUnit failure XML.

Parameters
entry (*Dict[str, Any]*) –

Return type
Element

crazyhusk.reports.**report_object_to_testsuite_xml**(*report*)

Convert Unreal JSON report into jUnit testsuite.

Parameters
report (*Dict[str, Any]*) –

Return type
Element

crazyhusk.reports.**report_test_to_testcase_xml**(*test*)

Convert Unreal JSON report test into jUnit testcase.

Parameters
test (*Dict[str, Any]*) –

Return type
Element

crazyhusk.reports.**report_timestamp_to_iso8601_timestamp**(*timestamp*)

Convert Unreal JSON report formatted timestamp to ISO8601 timestamp.

Parameters
timestamp (*str*) –

Return type
str

crazyhusk.reports.**write_junit_xml_report**(*report_file*, *test_suites*)

Write XML test suites to a file.

Parameters

- **report_file** (*str*) –
- **test_suites** (*Element*) –

Return type
None

7.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

7.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

7.3.2 How to request a feature

Request features on the Issue Tracker.

7.3.3 How to set up your development environment

Install the package with development requirements:

```
$ pip install .[dev]
```

7.3.4 How to test the project

Run the full test suite:

```
$ pytest
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

7.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The test suite must pass without errors and warnings.
- Include unit tests. This project does its best to achieve excellent coverage statistics.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ pre-commit install
```

Or you may run the formatting checks at any time on staged changes by running the following command:

```
$ pre-commit run
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

7.4 Contributor Covenant Code of Conduct

7.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission

- Other conduct which could reasonably be considered inappropriate in a professional setting

7.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at nhaines.pro@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

7.5 License

PYTHON MODULE INDEX

C

INDEX

N

O

P

R

S

T

U

V

W