galaxyutils

Release v0.1.2

Tyler Nichols

Jan 01, 2020

CONTENTS

CHAPTER

ONE

CONFIG_PARSER

1.1 Overview

This particular file allows for the use of a configuration file for a plugin (known as `config.cfg`), which is created from a default configuration file (known as `default_config.cfg`).

These configuration files have the following format:

- Lines beginning with `#` are comments, and are not interpreted by the underlying parser. These can be used to inform users about an option's default value, allowed values, and purpose.
- Lines beginning with `##` are default-only comments, meaning that they will not be copied from `default_config.cfg` into `config.cfg` when creating a duplicated configuration file. These can be used to discourage users from editing the `default_config.cfg` file, among other reasons.
- Lines not preceded by a `#` or a `##` are interpreted to have both an option's name and its value, separated by an assignment operator (=).

Here is an example of a valid `default_config.cfg` file:

```
## DO NOT EDIT OR DELETE THIS FILE! If you need to make changes, then copy this file,␣
↪save it as "config.cfg" in the
## plugin root directory, and edit that file instead!

log_sensitive_data=False
# Default Value: False
# Allowed Values:
#   - True
#   - False
# If set to true, this setting will add sensitive information to the log file␣
↪generated by this plugin. Since this
# information can compromise the security of your account, this setting is best left␣
↪to false, unless you are
# debugging the plugin.
```

Within a plugin (preferably in a file designated to constants, like `consts.py`), the defined setting can then be accessed like this:

```
CONFIG_OPTIONS = get_config_options([
    Option(option_name="log_sensitive_data")
])

LOG_SENSITIVE_DATA = CONFIG_OPTIONS["log_sensitive_data"]
```

For more details regarding accessing the user's defined configuration settings, see the documentation below.

1.2 Documentation

CHAPTER

TWO

TIME_TRACKER

2.1 Overview

This particular file allows for the creation of a `TimeTracker` object, which contains methods that allow an integration developer to keep track of the user's play time for games in their library.

It is the developer's responsibility to do the following:

- Store the user's play time cache for another session. A good implementation for doing so is to both save the play time cache to the `persistent_cache` and to write it locally to a file on the user's disk. This ensures that the user's play time is safe if the `persistent_cache` is erased (i.e., the user loses authentication). An example is shown below.

```
async def shutdown(self):
    if self.game_time_cache:  # Do not overwrite the file if the cache is empty. This␣
↪prevents accidentally erasing
                              # the user's previous game time cache and resetting␣
↪their play time.
        file = open("PlayTimeCache.txt", "w+")
        # Consider informing the user to not modify the game time cache file.
        file.write("# DO NOT EDIT THIS FILE\n")
        file.write(self.game_time_tracker.get_time_cache_hex())
        file.close()
    await self._http_client.close()
    await super().shutdown()

def game_times_import_complete(self):
    self.game_time_cache = self.game_time_tracker.get_time_cache()
    self.persistent_cache["game_time_cache"] = self.game_time_tracker.get_time_cache_
↪hex()
    self.push_cache()
```

- Retrieve the user's play time cache from the previous session. Supposing that the cache was saved according to the previous example, a good implementation would be to retrieve it from the `persistent_cache` if possible, and then fall back to the locally stored file if it cannot be found there. In general, it is best to use a locally stored file to retrieve the game time cache only if all other methods fail. An example is shown below.

```
def handshake_complete(self):
    # Check the persistent cache first.
    if "game_time_cache" in self.persistent_cache:
        self.game_time_cache = pickle.loads(bytes.fromhex(self.persistent_cache["game_
↪time_cache"]))
    # If the game time cache cannot be found in the persistent cache, then check a␣
↪local file for it.
```

(continues on next page)

(continued from previous page)

```
    else:
        try:
            file = open("PlayTimeCache.txt", "r")
            for line in file.readlines():
                if line[:1] != "#":
                    self.game_time_cache = pickle.loads(bytes.fromhex(line))
                    break
        except FileNotFoundError:
            # If the file does not exist, then use an empty game time cache.
            self.game_time_tracker = TimeTracker()
            return
    self.game_time_tracker = TimeTracker(game_time_cache=self.game_time_cache)
```

2.2 Documentation

This module contains some utilities that Galaxy 2.0 plugin developers may find useful. The utilities are designed to be platform- and operating system-independent.

CHAPTER

THREE

FEATURES

Currently, the features of this module include the following:

- Internal Play Time Tracker (`time_tracker.py`): Keep track of a user's play time for each game manually, and save it locally to the user's disk.
- Configuration File Support (`config_parser.py`): Create and utilize a customized config.cfg file, which can contain settings that users and developers can alter to affect how a plugin functions.