SGD MODULES DOCUMENTATION

Backend2121

December 17, 2021

1 Introduction

All the modules go inside the *Modules* folder, the main script *gui.py* will handle automatically all the imports as long as the modules follow the structure described in paragraph 4.
A working example is already integrated inside the *Modules* folder called **TemplateModule.py**, you can take a look at that to see examples on how to implement the functions.

2 Disclaimer

By no means this software nor any on the modules used must cause excessive traffic to any website, this is an open source project and must not be used as an attack vector nor cause any sort of damage to anyone

3 Pre-Requisites

It is required to have some knowledge in:

- Python
- PyQt5
- Selenium/Requests (if needed)
- BeautifulSoup (if needed)

4 Structure

All the modules MUST follow this general structure, you can still add as many new fuction as you want:

- class ***Settings***
  - **init**(self)
- class ***module***
  - **init**(self)
  - **searchGame**(self, game: str) -> None
  - **listGames**(self) -> tuple[list, list]
  - **listIcons**(self) -> tuple[list, list]
  - **listLinks**(self, link: str) -> tuple[list, list]
  - **cleanLink**(self, link: str) -> str

5 Class Settings

This class gets instantiated by *guy.py* and by the *module* class (paragraph 6).

The *init* function MUST contain:

- A variable containing the name of the module **strictly named self.name**.

```
self.name = "TEST_MODULE"
```

- An empty string **strictly named self.logPath** where *gui.py* will store the current time to allow the module's logger to log to the same file as *gui.py*.

```
self.logPath = ""
```

- A method to load the module's settings as well as a way to load a default configuration in case of first time use eg.

```
with open("config.json",) as f:
    try:
        self.data = json.load(f)[self.name]
    except KeyError:
        # Load default config
        self.data = {
            'isEnabled': 1
        }
```

- At least a bare-metal widget with *PyQt*5, *gui.py* will create a tab inside the preferences menu to list the module's settings.

```
# Main widget used to display all of this inside a tab
self.widget = QWidget()

# Define main layout used to house all the sub-layouts
# with respective tickBoxes/textBoxes etc.
self.mainLayout = QVBoxLayout()

# Give layout to widget
self.widget.setLayout(self.mainLayout)
```

- A method to register changes in the settings (eg. a tickbox getting ticked).

```
def stateChange(self) -> None:
    """Register changes to data array"""
    if self.enableThis.isChecked(): self.data["isEnabled"] = 1
    else: self.data["isEnabled"] = 0
```

6 Class module

This will contain all the scripting part of the module.
The *init* function **must** contain:

- A reference to the Settings class **strictly named self.preferences**.

```
self.preferences = Settings()
```

- A reference to the module's name **strictly named self.name** defined in the *Settings* class.

```
self.name = self.preferences.name
```

- An empty string **strictly named self.toSearch** to store the user input.

```
self.toSearch = ""
```

- The definition of the module's logger (eg. python's logging module).

```
# Define logger and set the default config for it
self.log = logging.getLogger("Template_Logger")
logging.basicConfig(
    filename=self.preferences.logPath,
    filemode='a',
    format='%(levelname)s - %(name)s - "%(asctime)s": %(message)s',
    level="INFO"
    )
self.log.info("INITIALIZED {0}".format(self.name)) #Log init complete
```

- All other variables that the module needs (eg. a *Selenium* browser).

```
self.options = Options()

# Headless + silent selenium config
self.options.add_experimental_option("detach", True)
self.options.add_argument("--headless")
self.options.add_argument('log-level=1')

self.browser = Chrome(
    chrome_options=self.options,
    executable_path=os.getcwd() + "/chromedriver.exe"
    )
```

- A function **strictly named self.searchGame()**.

```
def searchGame(self, game: str) -> None:
    """Called by searchGamesWorker.py

    Instantiated in SEARCH PHASE 1 of gui.py"""
    # Use this function to reach the website through a proxy
    # if needed or use it as a set method for self.toSearch
```

- A function **strictly named self.listGames()**.

  This is used to pass to the gui all the titles + links of the games found in a webpage, the user will then select one of them and the script will continue.

```
def listGames(self) -> tuple[list, list]:
    """Called by listGamesWorker.py

    Instantiated in SEARCH PHASE 2 of gui.py

    Gets all games listed in the first page of results

    tuple[0] = Games's Titles

    tuple[1] = Games's Page Links
    """
```

- A function **strictly named self.listIcons()**.

  This function should be runned only if the user has an hypotetical *loadIcons* tickBox inside *Settings*.

```
def listIcons(self) -> tuple[list, list]:
    """Called by iconGamesWorker.py

    Instantiated in SEARCH PHASE 3 of gui.py"""
```

- A function **strictly named self.listLinks()**

  This function is run after the user selected the desired game/rom from the list on the left, the function must return all the entries + links that the webpage lists for the selected game.

```
def listLinks(self, link: str) -> tuple[list, list]:
    """Called by listLinksWorker.py

    Instantiated in SELECTION PHASE 1 of gui.py"""
```

- A function **strictly named self.cleanLink()**

  This function is the last one to be called, and must return the string containing the direct download link (or the furthest link the module can reach if **Captchas** are used by the website).

  If a proxy was used in the *searchGame* function then this function must return a cleaned version of the link.

  (**hide.me** for example messes up the clear text link creating a one-time only link that only the browser that created it can use, so this function must return the real link)

```
def cleanLink(self, link: str) -> str:
    """Called by cleanLinkWorker.py

    Instantiated in LINK_FETCHING PHASE 1 of gui.py"""
```