# NVLib's AutoGUI Python Library - API Documentation

This document provides a complete reference for the AutoGUI Python library, which uses customtkinter to build user interfaces from JSON files created with the NVLib GUI Builder.

## 1. The Main AutoGUI Class

The AutoGUI class is the main entry point for your application. It creates and manages the main window and all the components within it.

### Initialization

First, create an instance of the class. This will automatically create your main application window.

```
from NVLib.Components.GUI.AutoGUI import AutoGUI

app = AutoGUI()
```

### Core Methods

| Method | Description |
|---|---|
| app.build_gui('path/to.json') | Loads a layout file and builds the entire user interface. |
| app.run() | Starts the application's main event loop. This must be called at the end. |
| app.set_title("My App") | Sets the title of the application window. |
| app.set_background("#color") | Sets the background color of the main window. |
| app.load_new_gui() | Opens a file dialog to let the user select and load a new layout file. |
| app.close_gui() | Closes and terminates the application window. |

## 2. Accessing Components

You can access any component from your layout by its **ID**. The component ID

becomes an attribute of your app object.

**Example:** If you have a button with the ID login_button:

```
# Access the button
my_button = app.login_button
```

## 3. Understanding Event-Driven Programming (Callbacks)

GUI applications are different from simple scripts that run from top to bottom. They are **event-driven**, which means they wait for the user to do something (like click a button) and then react to that event. The code that runs in reaction to an event is called a **callback function**.

### Example Explained

Consider this code block:

```
def process_login():
    """This function is called when the login button is clicked."""
    username = app.username_text.text()
    password = app.password_text.text()

    print("--- Login Attempt ---")
    print(f"Username: '{username}'")
    print(f"Password: '{password}'")

    if username:
        app.welcome_label.text(f"Welcome, {username}!")
    else:
        app.welcome_label.text("Login Failed!")

# ... later in the code ...
app.login_button.on_click(process_login)
```

### What's Happening Here?

1. **Defining the Instructions:** The process_login() function is a set of instructions. It defines *what* should happen: get the text from the username and password fields, print them, and update a welcome label. At this point, the code has **not run yet**.

It's just been defined.
2. **Assigning the Event:** The line app.login_button.on_click(process_login) is the crucial part. It tells the button component: "When you are clicked, execute the instructions inside the process_login function."

Think of it like setting an alarm. You define the action (play music) and set the event (7:00 AM). You don't have to constantly check the clock; the alarm system handles waiting for the event and then performs the action for you. This is what makes GUI programming efficient.

## 4. Component-Specific API

**Button**

- **my_button.on_click(my_function)**: Assigns a function to be called when the button is clicked.
- **my_button.text("New Text") / my_button.text()**: Gets or sets the button's text.
- **my_button.text_color("#ff0000")**: Sets the text color.
- **my_button.background_color("#blue")**: Sets the button's background color.
- **my_button.bold(True)**: Makes the button's text bold.
- **my_button.toggle_visibility()**: Hides or shows the button.

**TextBox & TextArea**

- **my_textbox.text("Some text")**: Sets the text content.
- **current_text = my_textbox.text()**: Gets the current text.
- **my_textbox.text_color("#ff0000")**: Sets the text color.
- **my_textbox.background_color("#blue")**: Sets the background color.
- **my_textbox.bold(True)**: Makes the text bold.
- **my_textbox.toggle_visibility()**: Hides or shows the component.

**Label**

- **my_label.text("New Text") / my_label.text()**: Gets or sets the label's text.
- **my_label.text_color("#ff0000")**: Sets the text color.
- **my_label.bold(True)**: Makes the label's text bold.
- **my_label.toggle_visibility()**: Hides or shows the label.

**Checkbox**

- **is_checked = my_checkbox.is_checked()**: Returns True or False.
- **my_checkbox.on_toggle(my_function)**: Assigns a function to run when clicked.
- **my_checkbox.toggle_visibility()**: Hides or shows the checkbox.

**RadioGroup**

- **selected_value = my_radiogroup.get()**: Returns the value of the selected option.
- **my_radiogroup.on_select(my_function)**: Assigns a function to run when the selection changes.
- **my_radiogroup.toggle_visibility()**: Hides or shows the entire radio group.

**Dropdown**

- **selected_option = my_dropdown.get()**: Returns the currently selected option.
- **my_dropdown.on_select(my_function)**: Assigns a function to run when the selection changes.
- **my_dropdown.toggle_visibility()**: Hides or shows the dropdown.

**Slider**

- **current_value = my_slider.get()**: Returns the current numeric value.
- **my_slider.set(50)**: Sets the slider's value.
- **my_slider.toggle_visibility()**: Hides or shows the slider.

**ProgressBar**

- **my_progressbar.set(75)**: Sets the progress bar's value (0-100).
- **my_progressbar.toggle_visibility()**: Hides or shows the progress bar.

**Spinner**

- **current_value = my_spinner.get()**: Returns the current numeric value.
- **my_spinner.set(42)**: Sets the spinner's value.
- **my_spinner.toggle_visibility()**: Hides or shows the spinner.

**ToggleButton (Switch)**

- **is_on = my_toggle.is_on()**: Returns True if the switch is on.
- **my_toggle.on_toggle(my_function)**: Assigns a function to run when toggled.
- **my_toggle.toggle_visibility()**: Hides or shows the toggle button.

**Image**

- **my_image.toggle_visibility()**: Hides or shows the image.

**CardView & Panel**

- **my_cardview.toggle_visibility()**: Hides or shows the container and all components inside it.
- **my_cardview.background_color("#blue")**: Sets the container's background color.

## 5. Understanding if __name__ == '__main__':

In Python, every file is technically a "module." You might have one main script that you run (main.py), and other files that contain helper functions or classes (like autogui.py).

**What does it do?**

The line if __name__ == '__main__': is a standard Python construct that checks **how the script is being used**.

- **When you run the file directly** (e.g., python main.py), Python sets a special internal variable, __name__, to the string "__main__". The code inside this if block will then execute.
- **When** you **import the file** into another script (e.g., import main), Python sets __name__ to the name of the file (e.g., "main"). In this case, the code inside the if block will **not** execute.

**Why is it necessary?**

It allows you to create code that is both **runnable** and **importable**.

- **Runnable:** It defines the main starting point of your application. When you run main.py, the code inside this block is what kicks everything off—it creates the AutoGUI instance, builds the UI, and starts the app.
- **Importable:** Imagine you have a useful function inside main.py that you want to use in another project. If you import main into your new project, you don't want the entire login window to pop up unexpectedly. The if __name__ == '__main__': block prevents this from happening, allowing you to safely reuse your code without side effects.