Runtime Options

Pyglet offers a way to change runtime behavior through options. These options provide ways to modify specific modules, behavior for specific operating systems, or adding more debugging information. Options can be specified as a key, or as an attribute with the `pyglet.options` dataclass instance.

To change an option from its default, you must import `pyglet` before any sub-packages.

For example:

```
import pyglet
pyglet.options['debug_gl'] = False
pyglet.options.debug_media = True
```

The default options can be overridden from the OS environment as well. The corresponding environment variable for each option key is prefaced by `PYGLET_`. For example, in Bash you can set the `debug_gl` option with:

```
PYGLET_DEBUG_GL=True; export PYGLET_DEBUG_GL
```

For options requiring a tuple of values, separate each value with a comma.

---

class **Options**

Dataclass for global pyglet options.

audio: *Sequence* [*str*] = *('xaudio2', 'directsound', 'openal', 'pulse', 'silent')*

A **Sequence** of valid audio modules names. They will be tried from first to last until either a driver loads or no entries remain. See Choosing the audio driver for more information.

Valid driver names are:

- `'xaudio2'`, the Windows Xaudio2 audio module (Windows only)
- `'directsound'`, the Windows DirectSound audio module (Windows only)
- `'pulse'`, the PulseAudio module

; (Linux only, otherwise nearly ubiquitous. Limited features; use `'openal'` for more.)

- `'openal'`, the OpenAL audio module (A library may need to be installed on Windows and Linux)
- `'silent'`, no audio

`com_mta`: *bool* = *False*

If `True`, this will enforce COM Multithreaded Apartment Mode for Windows applications. By default, pyglet has opted to go for Single-Threaded Apartment (STA) for compatibility reasons. Many other third party libraries used with Python explicitly set STA. However, Windows recommends MTA with a lot of their API's such as Windows Media Foundation (WMF).

See: https://learn.microsoft.com/en-us/windows/win32/cossdk/com-threading-models

New in version 2.0.5.

`debug_com`: *bool* = *False*

If `True`, prints information on COM calls. This can potentially help narrow down issues with certain libraries that utilize COM calls. Only applies to the Windows platform.

`debug_font`: *bool* = *False*

If `True`, will print more verbose information when `Font`'s are loaded.

`debug_gl`: *bool* = *True*

If `True`, all calls to OpenGL functions are checked afterwards for errors using `glGetError`. This will severely impact performance, but provides useful exceptions at the point of failure. By default, this option is enabled if `__debug__` is enabled (i.e., if Python was not run with the -O option). It is disabled by default when pyglet is "frozen", such as within pyinstaller or nuitka.

`debug_gl_shaders`: *bool* = *False*

If `True`, prints shader compilation information such as creation and deletion of shader's. Also includes information on shader ID's, attributes, and uniforms.

`debug_gl_trace`: *bool* = *False*

If `True`, will print the names of OpenGL calls being executed. For exampl

`debug_gl_trace_args`: *bool* = *False*

If `True`, in addition to printing the names of OpenGL calls, it will also print the arguments passed into those calls. For example, `glBlendFunc(770, 771)`

> **Note**
>
> Requires `debug_gl_trace` to be enabled.

`debug_graphics_batch`: *`bool`* = *False*

If `True`, prints batch information being drawn, including **`Group`**'s, VertexDomains, and **`Texture`** information. This can be useful to see how many Group's are being consolidated.

`debug_input`: *`bool`* = *False*

If `True`, prints information on input devices such as controllers, tablets, and more.

`debug_lib`: *`bool`* = *False*

If `True`, prints the path of each dynamic library loaded.

`debug_media`: *`bool`* = *False*

If `True`, prints more detailed media information for audio codecs and drivers. Will be very verbose.

`debug_texture`: *`bool`* = *False*

If `True`, prints information on **`Texture`** size (in bytes) when they are allocated and deleted.

`debug_win32`: *`bool`* = *False*

If `True`, prints error messages related to Windows library calls. Usually get's information from `Kernel32.GetLastError`. This information is output to a file called `debug_win32.log`.

`debug_x11`: *`bool`* = *False*

If `True`, prints information related to Linux X11 calls. This can potentially help narrow down driver or operating system issues.

`dpi_scaling`: *`Literal`* [*`'real'`* , *`'scaled'`* , *`'stretch'`* , *`'platform'`*] = *'real'*

For 'HiDPI' displays, Window behavior can differ between operating syste
'real'.

The current options are an attempt to create consistent behavior across all of the operating systems.

'real' (default): Provides a 1:1 pixel for Window frame size and framebuffer. Primarily used for game applications to ensure you are getting the exact pixels for the resolution. If you provide an 800x600 window, you can ensure it will be 800x600 pixels when the user chooses it.

'scaled': Window size is scaled based on the DPI ratio. Window size and content (projection) size matches the full framebuffer. Primarily used for any applications that wish to become DPI aware. You must rescale and reposition your content to take advantage of the larger framebuffer. An 800x600 with a 150% DPI scaling would be changed to 1200x900 for both *window.get_size* and *window.get_framebuffer_size()*.

Keep in mind that pyglet objects may not be scaled proportionately, so this is left up to the developer. The `scale` & `dpi` attributes can be queried as a reference when determining object creation.

'stretch': Window is scaled based on the DPI ratio. However, content size matches original requested size of the window, and is stretched to fit the full framebuffer. This mimics behavior of having no DPI scaling at all. No rescaling and repositioning of content will be necessary, but at the cost of blurry content depending on the extent of the stretch. For example, 800x600 at 150% DPI will be 800x600 for *window.get_size()* and 1200x900 for *window.get_framebuffer_size()*.

'platform': A DPI aware window is created, however window sizing and framebuffer sizing is not interfered with by Pyglet. Final sizes are dictated by the platform the window was created on. It is up to the user to make any platform adjustments themselves such as sizing on a platform, mouse coordinate adjustments, or framebuffer size handling. On Windows and X11, the framebuffer and the requested window size will always match in pixels 1:1. On MacOS, depending on a Hi-DPI display, you may get a different sized framebuffer than the window size. This option does allow *window.dpi* and *window.scale* to return their respective values.

`dw_legacy_naming`: `bool` *= False*

If `True`, will enable legacy naming support for the default Windows font renderer (`DirectWrite`). Attempt to parse fonts by the passed name, to best match legacy RBIZ naming.

See: https://learn.microsoft.com/en-us/windows/win32/directwrite/font-selection#rbiz-font-family-model

For example, this allows specifying `"Arial Narrow"` rather than `"Arial"` with a `"condensed"` stretch or `"Arial Black"` instead of `"Arial"` with a weight of `black`. This may enhance naming compatibility cross-platform for select fonts as older font renderers went by this naming scheme.

Starts by parsing the string for any known style names, and searches all font collections for a matching RBIZ name. If a perfect match is not found, it will choose a second best match.

Note

Due to the high variation of styles and limited capability of some fonts, there is no guarantee the second closest match will be exactly what the user wants.

Note

The `debug_font` option can provide information on what settings are being selected.

New in version 2.0.3.

`headless`: `bool` *= False*

If `True`, visible Windows are not created and a running desktop environment is not required. This option is useful when running pyglet on a headless server, or compute cluster. OpenGL drivers with `EGL` support are required for this mode.

`headless_device`: `int` *= 0*

If using `headless` mode (`pyglet.options['headless'] = True`), this option allows you to set which GPU to use. This is only useful on multi-GPU systems.

`osx_alt_loop`: `bool` *= False*

If `True`, this will enable an alternative loop for Mac OSX. This is enforced for all ARM64 architecture Mac's.

Due to various issues with the ctypes interface with Objective C, Python, and Mac ARM64 processors, the standard event loop eventually starts breaking down to where inputs are either missed or delayed. Even on Intel based Mac's other odd behavior can be seen with the standard event loop such as randomly crashing from events.

New in version 2.0.5.

`search_local_libs`: `bool` *= True*

If `False`, pyglet won't try to search for libraries in the script directory and its `lib` subdirectory. This is useful to load a local library instead of the system installed version.

`shader_bind_management`: *`bool`* = *True*

If `True`, this will enable internal management of Uniform Block bindings for `ShaderProgram`'s.

If `False`, bindings will not be managed by Pyglet. The user will be responsible for either setting the binding points through GLSL layouts (4.2 required) or manually through `UniformBlock.set_binding`.

New in version 2.0.16.

`shadow_window`: *`bool`* = *True*

By default, pyglet creates a hidden window with a GL context when pyglet.gl is imported. This allows resources to be loaded before the application window is created, and permits GL objects to be shared between windows even after they've been closed. You can disable the creation of the shadow window by setting this option to False.

Some OpenGL driver implementations may not support shared OpenGL contexts and may require disabling the shadow window (and all resources must be loaded after the window using them was created). Recommended for advanced developers only.

New in version 1.1.

`text_antialiasing`: *`bool`* = *True*

If `True`, font renderers will improve text quality by adding antialiasing to the rendered characters. If `False`, text quality will appear pixelated.

`text_shaping`: *`Literal`*[*`'platform'`*, *`'harfbuzz'`*, *`False`*] = *'platform'*

Determines how text is processed and displayed based on features of the font.

Valid option names are:

- `False`, **Disables the shaping process for text. This may increase performance as it reduces the amount**

  of calls during rendering. If your font is simple, monospaced, or you require no advanced OpenType features, this option may be useful.

- `'platform'`, Uses platform's font system for shaping. Supported by Windows (DirectWrite) and Mac (CoreText).
- `'harfbuzz'`, Utilize the harfbuzz library for font shaping. This requires an optional dependency, if not

found, it will fallback to platform shaping.

New in version 2.0.

vsync: `bool` | `None` = *None*

If set, the *pyglet.window.Window.vsync* property is ignored, and this option overrides it (to either force vsync on or off). If unset, or set to None, the *pyglet.window.Window.vsync* property behaves as documented.

win32_disable_xinput: `bool` = *False*

If `True`, this will disable the `XInput` controller usage in Windows and fallback to `DirectInput`. Can be useful for debugging or special uses cases. A controller can only be controlled by either `XInput` or `DirectInput`, not both.

New in version 2.0.

win32_gdi_font: `bool` = *False*

If `True`, pyglet will fallback to the legacy `GDIPlus` font renderer for Windows. This may provide better font compatibility for older fonts. The legacy renderer does not support shaping, colored fonts, substitutions, or other OpenType features. It may also have issues with certain languages.

Due to those lack of features, it can potentially be more performant.

New in version 2.0.

xlib_fullscreen_override_redirect: `bool` = *False*

If `True`, pass the xlib.CWOverrideRedirect flag when creating a fullscreen window. This option is generally not necessary anymore and is considered deprecated.

xsync: `bool` = *True*

If `True` (the default), pyglet will attempt to synchronise the drawing of double-buffered windows to the border updates of the X11 window manager. This improves the appearance of the window during resize operations. This option only affects double-buffered windows on X11 servers supporting the Xsync extension with a window manager that implements the _NET_WM_SYNC_REQUEST protocol.

New in version 1.1.

`options`: *Options*
= Options(audio=('xaudio2', 'directsound', 'openal', 'pulse', 'silent'), debug_font=False, debug_gl=True, debug_gl_trace=False, debug_gl_trace_args=False, debug_gl_shaders=False, debug_graphics_batch=False, debug_lib=False, debug_media=False, debug_texture=False, debug_trace=False, debug_trace_args=False, debug_trace_depth=1, debug_trace_flush=True, debug_com=False, debug_win32=False, debug_input=False, debug_x11=False, shadow_window=True, vsync=None, xsync=True, xlib_fullscreen_override_redirect=False, search_local_libs=True, win32_gdi_font=False, text_antialiasing=True, headless=False, headless_device=0, text_shaping='platform', dw_legacy_naming=False, win32_disable_xinput=False, com_mta=False, osx_alt_loop=False, dpi_scaling='real', shader_bind_management=True)

Instance of `Options` used to set runtime options.

Environment settings

The default `pyglet.Options` instance (`pyglet.options`) can read default values from the operating system's environment variable. The following table shows which environment variable is used for each option:

Environment variable	`pyglet.options` key	Type	Default value
`PYGLET_AUDIO`	`audio`	List of strings	`directsound,openal,alsa,silent`
`PYGLET_DEBUG_GL`	`debug_gl`	Boolean	`1` [1]

[1] Defaults to `1` unless Python is run with `-O` or from a frozen executable.