Debugging tools

pyglet includes a number of debug paths that can be enabled during or before application startup. These were primarily developed to aid in debugging pyglet itself, however some of them may also prove useful for understanding and debugging pyglet applications.

Each debug option is a key in the `pyglet.options` dictionary. Options can be set directly on the dictionary before any other modules are imported:

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

They can also be set with environment variables before pyglet is imported. The corresponding environment variable for each option is the string `PYGLET_` prefixed to the uppercase option key. For example, the environment variable for `debug_gl` is `PYGLET_DEBUG_GL`. Boolean options are set or unset with `1` and `0` values.

A summary of the debug environment variables appears in the table below.

Option	Environment variable	Type
`debug_font`	`PYGLET_DEBUG_FONT`	bool
`debug_gl`	`PYGLET_DEBUG_GL`	bool
`debug_gl_trace`	`PYGLET_DEBUG_GL_TRACE`	bool
`debug_gl_trace_args`	`PYGLET_DEBUG_GL_TRACE_ARGS`	bool
`debug_graphics_batch`	`PYGLET_DEBUG_GRAPHICS_BATCH`	bool
`debug_lib`	`PYGLET_DEBUG_LIB`	bool
`debug_media`	`PYGLET_DEBUG_MEDIA`	bool
`debug_trace`	`PYGLET_DEBUG_TRACE`	bool
`debug_trace_args`	`PYGLET_DEBUG_TRACE_ARGS`	bool
`debug_trace_depth`	`PYGLET_DEBUG_TRACE_DEPTH`	int
`debug_win32`	`PYGLET_DEBUG_WIN32`	bool
`debug_x11`	`PYGLET_DEBUG_X11`	bool

The `debug_media` and `debug_font` options are used to debug the **`pyglet.media`** and **`pyglet.font`** modules, respectively. Their behaviour is platform-dependent and useful only for pyglet developers.

The remaining debug options are detailed below.

Debugging OpenGL

The `debug_graphics_batch` option causes all **`Batch`** objects to dump their rendering tree to standard output before drawing, after any change (so two drawings of the same tree will only dump once). This is useful to debug applications making use of **`Group`** and **`Batch`** rendering.

Error checking

The `debug_gl` option intercepts most OpenGL calls and calls `glGetError` afterwards (it only does this where such a call would be legal). If an error is reported, an exception is raised immediately.

This option is enabled by default unless the `-O` flag (optimisation) is given to Py script is running from within a py2exe or py2app package.

Tracing

The `debug_gl_trace` option causes all OpenGL functions called to be dumped to standard out. When combined with `debug_gl_trace_args`, the arguments given to each function are also printed (they are abbreviated if necessary to avoid dumping large amounts of buffer data).

Tracing execution

The `debug_trace` option enables Python-wide function tracing. This causes every function call to be printed to standard out. Due to the large number of function calls required just to initialise pyglet, it is recommended to redirect standard output to a file when using this option.

The `debug_trace_args` option additionally prints the arguments to each function call.

When `debug_trace_depth` is greater than 1 the caller(s) of each function (and their arguments, if `debug_trace_args` is set) are also printed. Each caller is indented beneath the callee. The default depth is 1, specifying that no callers are printed.

Platform-specific debugging

The `debug_lib` option causes the path of each loaded library to be printed to standard out. This is performed by the undocumented `pyglet.lib` module, which on Linux and Mac OS X must sometimes follow complex procedures to find the correct library. On Windows not all libraries are loaded via this module, so they will not be printed (however, loading Windows DLLs is sufficiently simple that there is little need for this information).

Linux

X11 errors are caught by pyglet and suppressed, as there are plenty of X servers in the wild that generate errors that can be safely ignored. The `debug_x11` option causes these errors to be dumped to standard out, along with a traceback of the Python stack (this may or may not correspond to the error, depending on whether or not it was reported asynchronously).

Windows

The `debug_win32` option causes all library calls into `user32.dll`, `kernel32.dll` and `gdi32.dll` to be intercepted. Before each library call `SetLastError(0)` is called, and afterwards `GetLastError()` is called. Any errors discovered are written to a file named `debug_win32.log`. Note that an error is only valid if the function called returned an error code, but some function does not check this.