Skip to content

Application Timers (bpy.app.timers)

Run a Function in x Seconds

```
import bpy


def in_5_seconds():
    print("Hello World")


bpy.app.timers.register(in_5_seconds, first_interval=5)
```

Run a Function every x Seconds

```
import bpy


def every_2_seconds():
    print("Hello World")
    return 2.0


bpy.app.timers.register(every_2_seconds)
```

Run a Function n times every x seconds

```
import bpy

counter = 0


def run_10_times():
    global counter
    counter += 1
    print(counter)
    if counter == 10:
        return None
    return 0.1


bpy.app.timers.register(run_10_times)
```

Assign parameters to functions

```
import bpy
import functools


def print_message(message):
```

```
    print("Message:", message)


bpy.app.timers.register(functools.partial(print_message, "Hello"), first_interval=2.0)
bpy.app.timers.register(functools.partial(print_message, "World"), first_interval=3.0)
```

Use a Timer to react to events in another thread

You should never modify Blender data at arbitrary points in time in separate threads. However you can use a queue to collect all the actions that should l executed when Blender is in the right state again. Pythons *queue.Queue* can be used here, because it implements the required locking semantics.

```
import bpy
import queue

execution_queue = queue.Queue()


# This function can safely be called in another thread.
# The function will be executed when the timer runs the next time.
def run_in_main_thread(function):
    execution_queue.put(function)


def execute_queued_functions():
    while not execution_queue.empty():
        function = execution_queue.get()
        function()
    return 1.0


bpy.app.timers.register(execute_queued_functions)
```

bpy.app.timers.**is_registered(function)**

Check if this function is registered as a timer.

PARAMETERS:

function (*Callable[[], float | None]*) – Function to check.

RETURNS:

True when this function is registered, otherwise False.

RETURN TYPE:

bool

bpy.app.timers.**register(function, first_interval=0, persistent=False)**

Add a new function that will be called after the specified amount of seconds. The function gets no arguments and is expected to return either None a float. If `None` is returned, the timer will be unregistered. A returned number specifies the delay until the function is called again. `functools.partial` can be used to assign some parameters.

PARAMETERS:

- **function** (*Callable[[], float | None]*) – The function that should called.
- **first_interval** (*float*) – Seconds until the callback should be called the first time.
- **persistent** (*bool*) – Don't remove timer when a new file is loaded.

bpy.app.timers.**unregister(function)**

Unregister timer.

Unregister all.

PARAMETERS:

function (*Callable[[], float | None]*) – Function to unregister.