q3py Documentation

Release 0.0.1.18-374c

robo9k

April 12, 2015

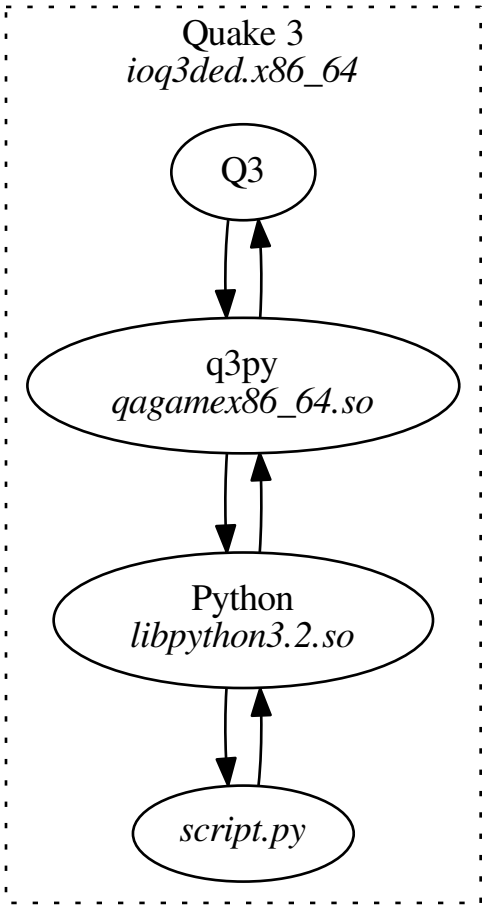
Contents

`q3py` is a Quake 3 to Python bridge.

Warning: This is a work in progress and not meant for public use (yet)!

The first-person shooter "Quake 3" uses separate modules for its game logic, i.e. client, server and user interface.

`q3py` is such a Quake 3 module, but it relays all calls to a Python module. To do so, q3py embedds `libpython3` and provides a Python extension module to allow the Python code to call back into Quake 3.



Todo

The SVG/PNG generated by Sphinx has a random name and thus can not be linked to from the GitHub README.rst

CHAPTER 1

Contents

1.1 Hello world!

This is the classical "Hello world!" example for q3py, `q3py_hello.py`.

```
import q3py
import ctypes


# Game module initialization function.
# Defined in ioq3/code/game/g_public.h
GAME_INIT = 0

# Engine system trap to print errors from game module.
# Defined in ioq3/code/game/g_public.h
G_ERROR = 1


# Compare with trap_Error() from ioq3/code/game/g_syscalls.c
def qerror(msg):
    c_msg = ctypes.create_string_buffer(msg)

    return q3py.syscall(G_ERROR, ctypes.addressof(c_msg))


# Compare with G_InitGame() from ioq3/code/game/g_main.c
def init_game(level_time, random_seed, restart):
    print("Python init_game(level_time={level_time}, "
          "random_seed={random_seed}, "
          "restart={restart})".format(level_time=level_time,
                                      random_seed=random_seed,
                                      restart=restart))
    qerror(b"Hello, Quake 3!")


# Compare with vmMain() from ioq3/code/game/g_main.c
def vm_main(cmd, arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7, arg8,
            arg9, arg10, arg11):

    if (cmd == GAME_INIT):
        init_game(arg0, arg1, bool(arg2))

    return -1
```

```
39
40
41  # Related to dllEntry() in ioq3/code/game/g_syscalls.c
42  def dll_entry():
43      print("Python dll_entry() called")
44
45      return vm_main
```

As you can see, this is a little more involved, but to be fair it is not the most simple way to write a hello world program with q3py.

To run this example, take a look at the *Configuration* page. The output might look like this:

```
Q3PY [INFO]: Entry point is 'q3py_hello:dll_entry'
Python dll_entry() called
Q3PY [INFO]: v0.0.1 initialized
Python init_game(level_time=0, random_seed=42, restart=False)
********************
ERROR: Hello, Quake 3!
********************
----- Server Shutdown (Server crashed: Hello, Quake 3!) -----
forcefully unloading qagame vm
---------------------------
]
```

Hey wait, did it just say we crashed the server?

Yes, but there is a reason to this. The purpose of a Quake 3 module is not to print "Hello world", but to implement game logic. We only implemented a tiny little part of the `game` logic (we could have done this for `cgame` or `ui` as well).

When Quake 3 loads this Python module via q3py, it calls the `vmMain` function of q3py, which in turn calls our `vm_main`, since that's what we told q3py to do. The `vmMain` function acts as a generic dispatcher for all the calls of the engine, hence the ugly arguments and argument names.

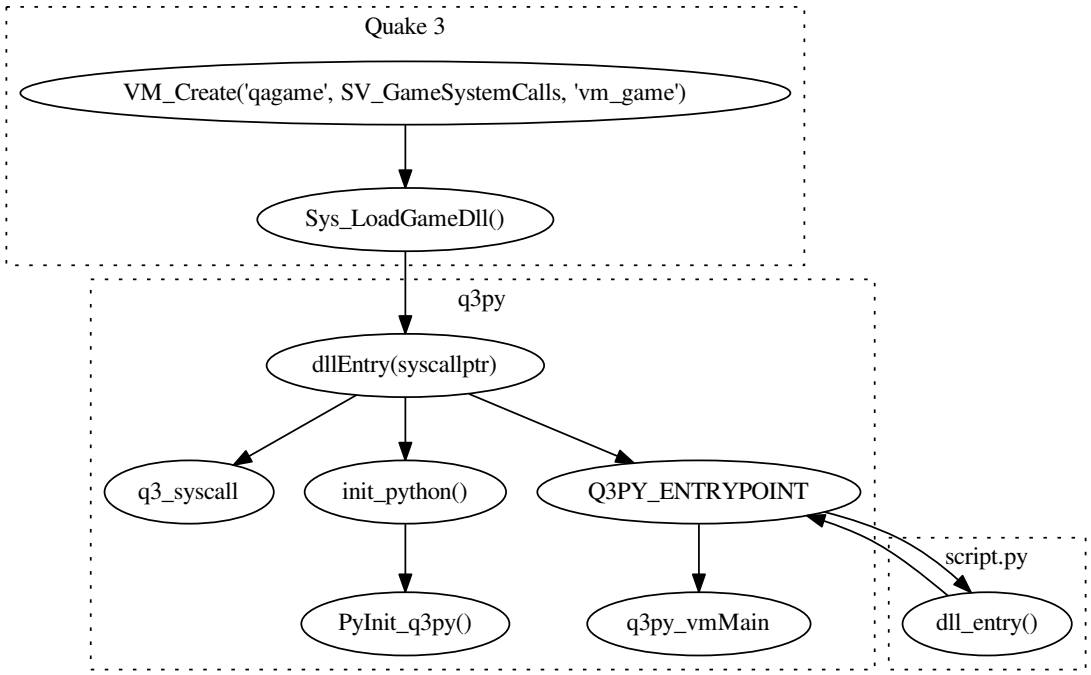
Either way, upon initialization Quake 3 calls `vmMain` with the command `GAME_INIT` and some additional arguments specific to this call, which are getting passed into `init_game` here.

In `init_game`, we call a `qerror` method with the message that we saw during the server crash. So now in `qerror` we use q3py to do a "syscall" back into the Quake 3 engine and as you might have guessed, we call the `G_ERROR` syscall which justs prints the message as we've seen and exits the game :)

1.2 Call graphs

The following graphs give an overview of how Quake 3, q3py, Python and finally your Python module interact with each other.

1.2.1 Initialization



When the Quake 3 dedicated server or client binary is launched, it will attempt to load several game modules, i.e. `game`, `cgame` and `ui`. This example is looking at the server and thus `game` module only.

Quake 3 calls `VM_Create()` to load a game module by its name ("qagame"), the syscall function provided by the engine (`SV_GameSystemCalls()`) and the mode in which to load the game module ("vm_game", see *Quake 3 configuration*).

Depending on the VM mode, Quake 3 then looks for a matching native library file via `Sys_LoadGameDll()` and shall find q3py. Quake 3 invokes the exported `dllEntry` function of the game module and passes the syscall pointer (`SV_GameSystemCalls`) as an argument.

Upon invocation of its `dllEntry`, q3py stores this syscall pointer into a global variable `q3_syscall` to be used lateron. Furthermore it initializes Python (`init_python()`) and its own Python extension module (`PyInit_q3py()`).

q3py then uses its `Q3PY_ENTRYPOINT` setting to call the given method of your Python module and stores the result in the global variable `q3py_vmMain` for later use.

Once all functions and methods have returned to their caller, all of Quake 3, q3py, Python and your Python module should have done their minimal initialization. Depending on the Quake 3 fork, the engine will now do an additional syscall into the game module (`GAME_INIT`) to initialize its state.

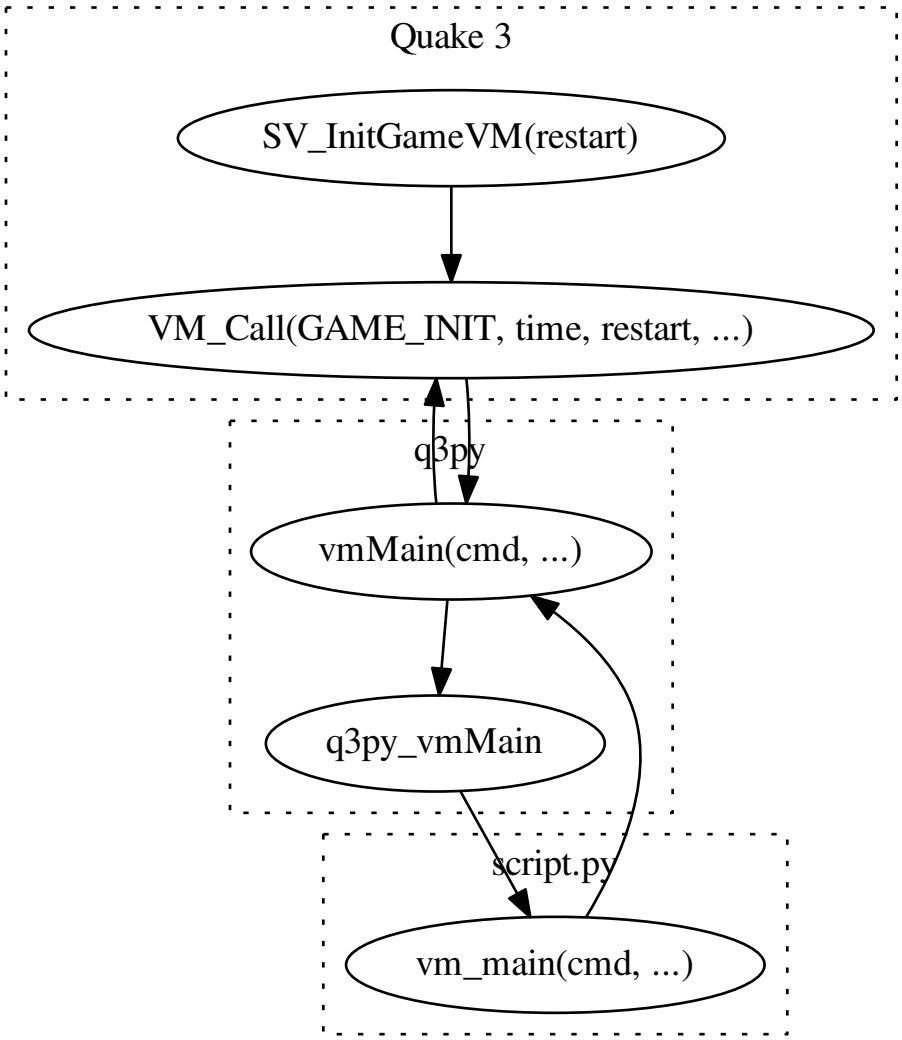
Note: Quake 3 calls `dllEntry` only when a map is loaded, not when one is restarted (it calls `GAME_INIT` in both cases and passes a `restart` argument).

Furthermore there is no inverse operation such as `dllExit`. Quake 3 invokes syscalls such as `GAME_SHUTDOWN` and unloads the shared library. q3py does not implement `_fini` or such (see `man dlclose(3)`).

Todo

Investigate whether we need `_fini` or such to shutdown Python or cleanup any other shared state.

1.2.2 Quake 3 calling q3py



Todo

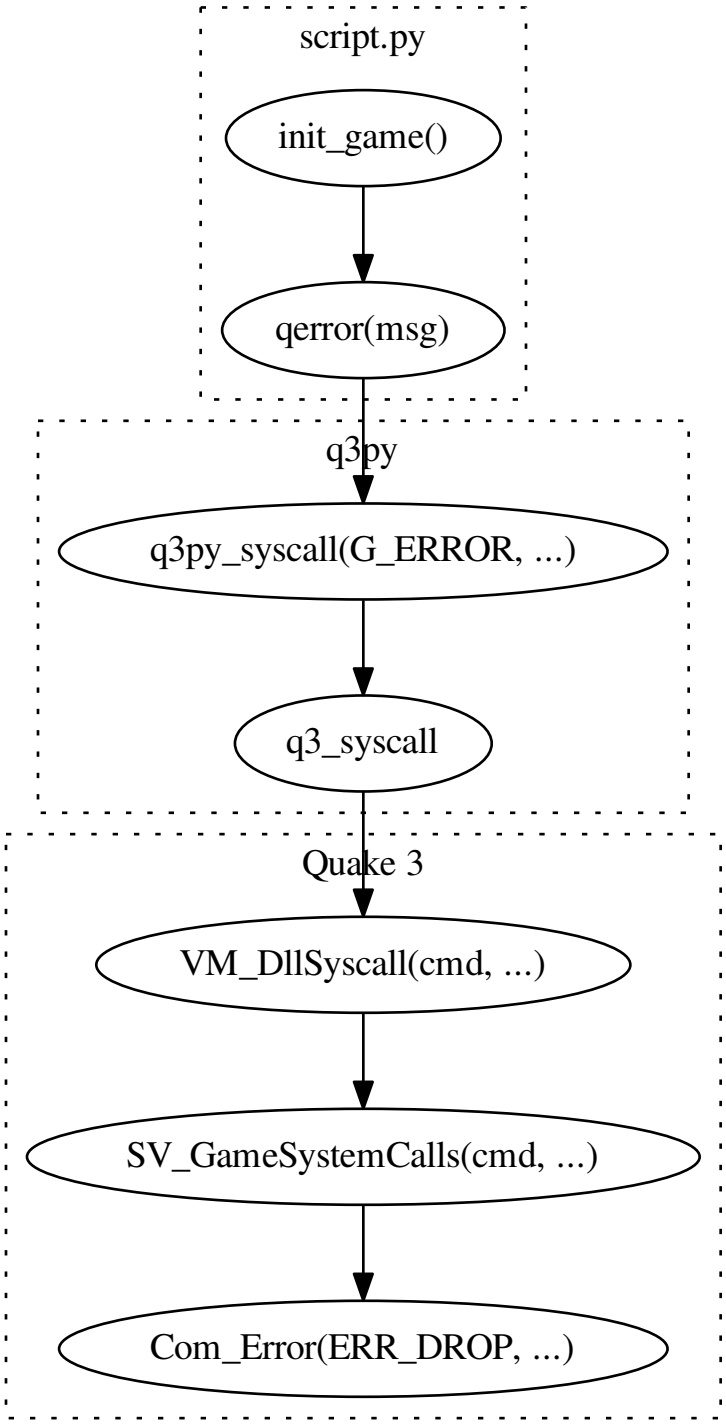
Graphviz layout with subgraph labels is unreadable.

Quake 3 calls into the game modules via syscalls, for example to initialize the game state with `GAME_INIT`. Do to so, the engine calls `VM_Call()` on a VM it created previously and passes the syscall command (`GAME_INIT`)

and arguments depending on the syscall (in the case of `GAME_INIT` those are `levelTime`, `randomSeed` and `restart`).

With a shared library as game module, its exported `vmMain` function is invoked with the syscall command and arguments. q3py looks up the configured Python callable in its global `q3py_vmMain` variable and calls it with the syscall command and arguments. q3py then passes the return value of the Python callable back to Quake 3.

1.2.3 script.py calling q3py



The game modules call back into Quake 3 via syscalls, for example to indicate a critical error.

Your Python module can either use the q3py C or Python API. In both cases the `q3py_syscall()` function is invoked with the syscall command and arguments. q3py passes those to its `q3_syscall` function pointer which it obtained during initialization earlier on.

This function pointer aims at `VM_DllSyscall()`, a compability function which invokes the engine system call dispatcher, e.g. `SV_GameSystemCalls()`, with the syscall command and arguments. The dispatcher then calls a function which matches the syscall command, e.g. `Com_Error()` for `G_ERROR`.

1.3 Installation

As of now, q3py is only available as a source code download. As such you will need to install all the dependencies to build it.

1.3.1 Download

```
wget https://github.com/robo9k/q3py/archive/master.zip
unzip q3py-master.zip && cd q3py-master/
```

You need to download the source code of q3py. The easiest way is to grab the current master branch tarball (a zip actually) from GitHub. After downloading, extract the source code somewhere.

1.3.2 Compile

```
sudo apt-get install build-essential autotools-dev python3-dev
```

To compile q3py, you need a C99 compiler such as GCC (clang works as well). Furthermore you need the GNU autotools and the Python 3 header files and library.

```
autoreconf --install && ./configure && make
```

You then run the autotools to create a `Makefile` and such, configure q3py and finally run `make` to compile everything.

1.3.3 Install

```
sudo make install
```

To install q3py after compilation, just run the `install` Make target as root, which will install q3py as a system package.

Todo

Install to local directory without root

Since q3py is meant to work with any Quake 3 fork, you now need to install it for your game.

```
ln -s /usr/local/lib64/q3py.so ~/.q3a/q3py/qagamex86_64.so
```

This creates a symbolic link from the q3py system package to a "q3py" game directory for the Quake 3 installation of the current user.

Note: You need to configure Quake 3 to use the game directory in which you just installed q3py (unless you use the `BASEGAME`, e.g. "baseq3").

1.4 Configuration

To run q3py, you need to configure the components that it consists of; i.e. q3py itself, Quake 3 and the embedded Python.

As an example, you can run q3py like this;

```
Q3PY_ENTRYPOINT="q3py_hello:dll_entry" \
PYTHONPATH="q3py/doc/examples/" \
ioq3ded +set vm_game 0 \
+set fs_game "q3py" +map "q3dm6"
```

Note: This example assumes that you installed q3py in a "q3py" game directory (the `fs_game` option for Quake 3). Furthermore it starts the dedicated server of ioquake3 and loads the map "q3dm6".

The output from running this example might look like this:

```
Try loading dll file /home/robo9k/.q3a/q3py/qagamex86_64.so
Loading DLL file: /home/robo9k/.q3a/q3py/qagamex86_64.so
Sys_LoadGameDll(/home/robo9k/.q3a/q3py/qagamex86_64.so) found vmMain function at 0xdeadbeef
]Q3PY [INFO]: dllEntry called with syscall 0xcafebabe
Q3PY [INFO]: Entry point is 'q3py_hello:dll_entry'
Q3PY [INFO]: v0.0.1 initialized
```

1.4.1 q3py

Since q3py is loaded as a shared library by Quake 3, you can not pass command line options to it. Instead, q3py is configured with environment variables.

Q3PY_ENTRYPOINT
New in version 0.0.1.

The Python entry point which q3py shall use, in the form of *`module:method`*.

> **Warning:** This is a required setting. Without it, q3py could not do anything and thus exits with an error.

See also:

`Q3PY_ENV_ENTRYPOINT` `init_python()`

1.4.2 Quake 3

In its default configuration Quake 3 does not load shared libraries (such as q3py) for its game modules. Therefor you need to configure the *ioquake3 cvar vm_game* to use q3py as the `game` module, `vm_cgame` for the `cgame` module and `vm_ui` for the `ui` module respectively.

You can do so either via a `+set vm_game 0` command line option at Quake 3 startup or by adding `set vm_game 0` to your Quake 3 `.cfg`.

1.4.3 Python

In order for Python to find the module you specified as your q3py entry point above, you need to tell it where to look at.

You do so by adding the folder that contains the Python module to the `PYTHONPATH` environment variable.

Todo

virtualenv, venv, activate_this.py

1.5 Python API

q3py acts as a C extension module to the embedded Python. This allows the Python module to call back into Quake 3 and to configure q3py.

`q3py.`**`set_vmmain`**(*callback*)
New in version 0.0.1.

Sets the `vmMain` callable that q3py invokes when being called by Quake 3.

Parameters **callback** (*callable*) – The Python callable to use

Raises TypeError: if `callback` is not a `callable()`

`q3py.`**`syscall`**(*num*[, *args*])
New in version 0.0.1.

Invokes a Quake 3 `syscall`.

Parameters

- **num** (*int*) – The number of the syscall
- **args** (*int*) – The arguments for the syscall

Returns The result of the syscall

Return type int

`q3py.`**`__version__`**
New in version 0.0.1.

Contains the version of q3py as a read-only string of the form *MAJOR*`.`*MINOR*`.`*PATCH*, e.g. "0.0.1".

`q3py.`**`_C_API`**
New in version 0.0.1.

Contains a read-only `PyCapsule` of q3py's C API.

1.6 C API

q3py exposes a C API as a Python Capsule, which can be used directly by other native Python extensions.

All of the following can be found in q3py's public header file `include/q3py.h`.

intptr_t **`q3py_syscall`**(intptr_t *number*, ...)
New in version 0.0.1.

Invokes a Quake 3 `syscall`.

Parameters

- **number** (*intptr_t*) – The number of the syscall
- **...** (*intptr_t*) – The arguments for the syscall

Returns The result of the syscall

int **import_q3py**()

New in version 0.0.1.

Imports the q3py Python capsule into `Q3Py_API`.

> **Warning:** This needs to be called before any other q3py function, otherwise you will get a nice `SEGFAULT`.

Returns `0` on success, `-1` on error

void** **Q3Py_API**

New in version 0.0.1.

Holds a pointer to the q3py Python capsule.

> **Warning:** Do not set this directly, use `import_q3py()` instead!

CHAPTER 2

License

All of q3py (the code, its documentation and build scripts) is licensed under the MIT license, see the `LICENSE` file.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*

q

Symbols

E

I

P

Q

S