Note

You are not reading the most recent version of this documentation. v1.10.0 is the latest version available.

UCI engine communication

The Universal Chess Interface is a protocol for communicating with engines.

chess.uci.popen_engine(*command*, *, *engine_cls=<class 'chess.uci.Engine'>*, *setpgrp=False*, ***kwargs*)

Opens a local chess engine process.

No initialization commands are sent, so do not forget to send the mandatory *uci* command.

```
>>> engine = chess.uci.popen_engine("/usr/bin/stockfish")
>>> engine.uci()
>>> engine.name
'Stockfish 8 64 POPCNT'
>>> engine.author
'T. Romstad, M. Costalba, J. Kiiski, G. Linscott'
```

Parameters:

- **command** –
- **engine_cls** –
- **setpgrp** – Open the engine process in a new process group. This will stop signals (such as keyboard interrupts) from propagating from the parent process. Defaults to `False`.

chess.uci.spur_spawn_engine(*shell*, *command*, *, *engine_cls=<class 'chess.uci.Engine'>*)

Spawns a remote engine using a Spur shell.

```
>>> import spur
>>>
>>> shell = spur.SshShell(hostname="localhost", username="username", password="pw")
>>> engine = chess.uci.spur_spawn_engine(shell, ["/usr/bin/stockfish"])
>>> engine.uci()
```

class **chess.uci.Engine**(*, *Executor=<class 'concurrent.futures.thread.ThreadPoolExecutor'>*)

process

The underlying operating system process.

name

The name of the engine. Conforming engines should send this as *id name* when they receive the initial *uci* command.

author

The author of the engine. Conforming engines should send this as *id author* after the initial *uci* command.

options

A case-insensitive dictionary of Options. The engine should send available options when it receives the initial *uci* command.

uciok

`threading.Event()` that will be set as soon as *uciok* was received. By then `name`, `author` and `options` should be available.

return_code

The return code of the operating system process.

terminated

`threading.Event()` that will be set as soon as the underyling operating system process is terminated and the `return_code` is available.

terminate(*, *async_callback=None*)

Terminate the engine.

This is not an UCI command. It instead tries to terminate the engine on operating system level, like sending SIGTERM on Unix systems. If possible, first try the *quit* command.

Returns: The return code of the engine process (or a Future).

kill(*, *async_callback=None*)

Kill the engine.

Forcefully kill the engine process, like by sending SIGKILL.

Returns: The return code of the engine process (or a Future).

is_alive()

Poll the engine process to check if it is alive.

UCI commands

class **chess.uci.Engine**(*, *Executor=<class 'concurrent.futures.thread.ThreadPoolExecutor'>*)

uci(*, *async_callback=None*)

Tells the engine to use the UCI interface.

This is mandatory before any other command. A conforming engine will send its name, authors and available options.

Returns: Nothing

debug(*on*, *, *async_callback=None*)

Switch the debug mode on or off.

In debug mode, the engine should send additional information to the GUI to help with the debugging. Usually, this mode is off by default.

Parameters: **on** – bool

Returns: Nothing

isready(*, *async_callback=None*)

Command used to synchronize with the engine.

The engine will respond as soon as it has handled all other queued commands.

Returns: Nothing

setoption(*options*, *, *async_callback=None*)

Set values for the engine's available options.

Parameters: **options** – A dictionary with option names as keys.

Returns: Nothing

ucinewgame(*, *async_callback=None*)

Tell the engine that the next search will be from a different game.

This can be a new game the engine should play or if the engine should analyse a position from a different game. Using this command is recommended, but not required.

Returns: Nothing

position(*board*, ***, *async_callback=None*)

Set up a given position.

Rather than sending just the final FEN, the initial FEN and all moves leading up to the position will be sent. This will allow the engine to use the move history (for example to detect repetitions).

If the position is from a new game, it is recommended to use the *ucinewgame* command before the *position* command.

Parameters: **board** – A *chess.Board*.

Returns: Nothing

Raises: `EngineStateException` if the engine is still calculating.

go(***, *searchmoves=None*, *ponder=False*, *wtime=None*, *btime=None*, *winc=None*, *binc=None*, *movestogo=None*, *depth=None*, *nodes=None*, *mate=None*, *movetime=None*, *infinite=False*, *async_callback=None*)

Start calculating on the current position.

All parameters are optional, but there should be at least one of *depth*, *nodes*, *mate*, *infinite* or some time control settings, so that the engine knows how long to calculate.

Note that when using *infinite* or *ponder*, the engine will not stop until it is told to.

Parameters:

- **searchmoves** – Restrict search to moves in this list.
- **ponder** – Bool to enable pondering mode. The engine will not stop pondering in the background until a *stop* command is received.
- **wtime** – Integer of milliseconds White has left on the clock.
- **btime** – Integer of milliseconds Black has left on the clock.
- **winc** – Integer of white Fisher increment.
- **binc** – Integer of black Fisher increment.
- **movestogo** – Number of moves to the next time control. If this is not set, but wtime or btime are, then it is sudden death.
- **depth** – Search *depth* ply only.
- **nodes** – Search so many *nodes* only.
- **mate** – Search for a mate in *mate* moves.
- **movetime** – Integer. Search exactly *movetime* milliseconds.
- **infinite** – Search in the background until a *stop* command is received.

Returns: A tuple of two elements. The first is the best move according to the engine. The second is the ponder move. This is the reply as sent by the engine. Either of the elements may be `None`.

Raises: `EngineStateException` if the engine is already calculating.

stop(*, *async_callback=None*)

Stop calculating as soon as possible.

Returns: Nothing.

ponderhit(*, *async_callback=None*)

May be sent if the expected ponder move has been played.

The engine should continue searching, but should switch from pondering to normal search.

Returns: Nothing.

Raises: `EngineStateException` if the engine is not currently searching in ponder mode.

quit(*, *async_callback=None*)

Quit the engine as soon as possible.

Returns: The return code of the engine process.

`EngineTerminatedException` is raised if the engine process is no longer alive.

Asynchronous communication

By default, all operations are executed synchronously and their result is returned. For example

```
>>> import chess.uci
>>>
>>> engine = chess.uci.popen_engine("stockfish")
>>>
>>> engine.go(movetime=2000)
BestMove(bestmove=Move.from_uci('e2e4'), ponder=None)
```

will take about 2000 milliseconds. All UCI commands have an optional *async_callback* argument. They will then immediately return a Future and continue.

```
>>> command = engine.go(movetime=2000, async_callback=True)
>>> command.done()
False
>>> command.result()  # Synchronously wait for the command to finish
BestMove(bestmove=Move.from_uci('e2e4'), ponder=None)
>>> command.done()
True
```

Instead of just passing *async_callback=True*, a callback function may be passed. It will be invoked **possibly on a different thread** as soon as the command is completed. It takes the command future as a single argument.

```
>>> def on_go_finished(command):
...     # Will likely be executed on a different thread.
...     bestmove, ponder = command.result()
...
>>> command = engine.go(movetime=2000, async_callback=on_go_finished)
```

Info handler

class **chess.uci.Score**

A *cp* (centipawns) or *mate* score sent by an UCI engine.

cp

Evaluation in centipawns or `None`.

mate

Mate in x or `None`. Negative number if the engine thinks it is going to be mated.

class **chess.uci.InfoHandler**

Chess engines may send information about their calculations with the *info* command. An `InfoHandler` instance can be used to aggregate or react to this information.

```
>>> import chess.uci
>>>
>>> engine = chess.uci.popen_engine("stockfish")
>>>
>>> # Register a standard info handler.
>>> info_handler = chess.uci.InfoHandler()
>>> engine.info_handlers.append(info_handler)
>>>
>>> # Start a search.
>>> engine.position(chess.Board())
>>> engine.go(movetime=1000)
BestMove(bestmove=Move.from_uci('e2e4'), ponder=Move.from_uci('e7e6'))
>>>
>>> # Retrieve the score of the mainline (PV 1) after search is completed.
>>> # Note that the score is relative to the side to move.
>>> info_handler.info["score"][1]
Score(cp=34, mate=None)
```

See `info` for a way to access this dictionary in a thread-safe way during search.

If you want to be notified whenever new information is available, you would usually subclass the `InfoHandler` class:

```
>>> class MyHandler(chess.uci.InfoHandler):
...     def post_info(self):
...         # Called whenever a complete info line has been processed.
...         print(self.info)
...         super().post_info()  # Release the lock
```

info

The default implementation stores all received information in this dictionary. To get a consistent snapshot, use the object as if it were a `threading.Lock()`.

```
>>> # Start thinking.
>>> engine.go(infinite=True, async_callback=True)
```

```
>>> # Wait a moment, then access a consistent snapshot.
>>> time.sleep(3)
>>> with info_handler:
...     if 1 in info_handler.info["score"]:
...         print("Score: ", info_handler.info["score"][1].cp)
...         print("Mate: ", info_handler.info["score"][1].mate)
Score: 34
Mate: None
```

depth(*x*)

Receives the search depth in plies.

seldepth(*x*)

Receives the selective search depth in plies.

time(*x*)

Receives a new time searched in milliseconds.

nodes(*x*)

Receives the number of nodes searched.

pv(*moves*)

Receives the principal variation as a list of moves.

In *MultiPV* mode, this is related to the most recent *multipv* number sent by the engine.

multipv(*num*)

Receives a new *multipv* number, starting at 1.

If *multipv* occurs in an info line, this is guaranteed to be called before *score* or *pv*.

score(*cp*, *mate*, *lowerbound*, *upperbound*)

Receives a new evaluation in *cp* (centipawns) or a *mate* score.

cp may be `None` if no score in centipawns is available.

mate may be `None` if no forced mate has been found. A negative number means the engine thinks it will get mated.

lowerbound and *upperbound* are usually `False`. If `True`, the sent score is just a *lowerbound* or *upperbound*.

In *MultiPV* mode, this is related to the most recent *multipv* number sent by the engine.

currmove(*move*)

Receives a move the engine is currently thinking about.

The move comes directly from the engine, so the castling move representation depends on the *UCI_Chess960* option of the engine.

currmovenumber(*x*)

Receives a new current move number.

hashfull(*x*)

Receives new information about the hash table.

The hash table is *x* permill full.

nps(*x*)

Receives a new nodes per second (nps) statistic.

tbhits(*x*)

Receives a new information about the number of tablebase hits.

cpuload(*x*)

Receives a new *cpuload* information in permill.

string(*string*)

Receives a string the engine wants to display.

refutation(*move*, *refuted_by*)

Receives a new refutation of a move.

refuted_by may be a list of moves representing the mainline of the refutation or `None` if no refutation has been found.

Engines should only send refutations if the *UCI_ShowRefutations* option has been enabled.

currline(*cpunr*, *moves*)

Receives a new snapshot of a line that a specific CPU is calculating.

cpunr is an integer representing a specific CPU and *moves* is a list of moves.

ebf(*ebf*)

Receives the effective branching factor.

pre_info(*line*)

Receives new info lines before they are processed.

When subclassing, remember to call this method on the parent class to keep the locking intact.

post_info()

Processing of a new info line has been finished.

When subclassing, remember to call this method on the parent class to keep the locking intact.

on_bestmove(*bestmove*, *ponder*)

Receives a new *bestmove* and a new *ponder* move.

on_go()

Notified when a *go* command is beeing sent.

Since information about the previous search is invalidated, the dictionary with the current information will be cleared.

Options

class **chess.uci.Option**

Information about an available option for an UCI engine.

name

The name of the option.

type

The type of the option.

Officially documented types are `check` for a boolean value, `spin` for an integer value between a minimum and a maximum, `combo` for an enumeration of predefined string values (one of which can be selected), `button` for an action and `string` for a text field.

default

The default value of the option.

There is no need to send a *setoption* command with the defaut value.

min

The minimum integer value of a *spin* option.

max

The maximum integer value of a *spin* option.

var

A list of allows string values for a *combo* option.