The UCI protocol as publiced by Stefan-Meyer Kahlen (ShredderChess):

Description of the universal chess interface (UCI) April 2004

* The specification is independent of the operating system. For Windows,
  the engine is a normal exe file, either a console or "real" windows application.

* all communication is done via standard input and output with text commands,

* The engine should boot and wait for input from the GUI,
  the engine should wait for the "isready" or "setoption" command to set up its internal parameters
  as the boot process should be as quick as possible.

* the engine must always be able to process input from stdin, even while thinking.

* all command strings the engine receives will end with '\n',
  also all commands the GUI receives should end with '\n',
  Note: '\n' can be 0x0c or 0x0a0c or any combination depending on your OS.
  If you use Engine und GUI in the same OS this should be no problem if you cummunicate in text mode,
  but be aware of this when for example running a Linux engine in a Windows GUI.

* The engine will always be in forced mode which means it should never start calculating
  or pondering without receiving a "go" command first.

* Before the engine is asked to search on a position, there will always be a position command
  to tell the engine about the current position.

* by default all the opening book handling is done by the GUI,
  but there is an option for the engine to use its own book ("OwnBook" option, see below)

* if the engine or the GUI receives an unknown command or token it should just ignore it and try to
  parse the rest of the string.

* if the engine receives a command which is not supposed to come, for example "stop" when the engine is
  not calculating, it should also just ignore it.

Move format:

The move format is in long algebraic notation.
A nullmove from the Engine to the GUI should be send as 0000.
Examples: e2e4, e7e5, e1g1 (white short castling), e7e8q (for promotion)

GUI to engine:

These are all the command the engine gets from the interface.

* uci
  tell engine to use the uci (universal chess interface),
  this will be send once as a first command after program boot
  to tell the engine to switch to uci mode.
  After receiving the uci command the engine must identify itself with the "id" command
  and sent the "option" commands to tell the GUI which engine settings the engine supports if any.
  After that the engine should sent "uciok" to acknowledge the uci mode.
  If no uciok is sent within a certain time period, the engine task will be killed by the GUI.

* debug [on | off]
  switch the debug mode of the engine on and off.
  In debug mode the engine should sent additional infos to the GUI, e.g. with the "info string" command,
  to help debugging, e.g. the commands that the engine has received etc.
  This mode should be switched off by default and this command can be sent
  any time, also when the engine is thinking.

* isready
  this is used to synchronize the engine with the GUI. When the GUI has sent a command or
  multiple commands that can take some time to complete,
  this command can be used to wait for the engine to be ready again or
  to ping the engine to find out if it is still alive.
  E.g. this should be sent after setting the path to the tablebases as this can take some time.
  This command is also required once before the engine is asked to do any search
  to wait for the engine to finish initializing.
  This command must always be answered with "readyok" and can be sent also when the engine is calculating
  in which case the engine should also immediately answer with "readyok" without stopping the search.

* setoption name [value]
  this is sent to the engine when the user wants to change the internal parameters
  of the engine. For the "button" type no value is needed.
  One string will be sent for each parameter and this will only be sent when the engine is waiting.
  The name of the option in should not be case sensitive and can inludes spaces like also the value.
  The substrings "value" and "name" should be avoided in and to allow unambiguous parsing,
  for example do not use = "draw value".
  Here are some strings for the example below:
     "setoption name Nullmove value true\n"
  "setoption name Selectivity value 3\n"
     "setoption name Style value Risky\n"
     "setoption name Clear Hash\n"
     "setoption name NalimovPath value c:\chess\tb\4;c:\chess\tb\5\n"

* register
  this is the command to try to register an engine or to tell the engine that registration
  will be done later. This command should always be sent if the engine has send "registration error"
  at program startup.
  The following tokens are allowed:
  * later
     the user doesn't want to register the engine now.
  * name
     the engine should be registered with the name
  * code
     the engine should be registered with the code

```
        Example:
           "register later"
           "register name Stefan MK code 4359874324"

 * ucinewgame
    this is sent to the engine when the next search (started with "position" and "go") will be from
    a different game. This can be a new game the engine should play or a new game it should analyse but
    also the next position from a testsuite with positions only.
    If the GUI hasn't sent a "ucinewgame" before the first "position" command, the engine shouldn't
    expect any further ucinewgame commands as the GUI is probably not supporting the ucinewgame command.
    So the engine should not rely on this command even though all new GUIs should support it.
    As the engine's reaction to "ucinewgame" can take some time the GUI should always send "isready"
    after "ucinewgame" to wait for the engine to finish its operation.

 * position [fen  | startpos ]  moves  ....
        set up the position described in fenstring on the internal board and
        play the moves on the internal chess board.
        if the game was played  from the start position the string "startpos" will be sent
        Note: no "new" command is needed. However, if this position is from a different game than
        the last position sent to the engine, the GUI should have sent a "ucinewgame" inbetween.

 * go
        start calculating on the current position set up with the "position" command.
        There are a number of commands that can follow this command, all will be sent in the same string.
        If one command is not send its value should be interpreted as it would not influence the search.
        * searchmoves  ....
                restrict search to this moves only
                Example: After "position startpos" and "go infinite searchmoves e2e4 d2d4"
                the engine should only search the two moves e2e4 and d2d4 in the initial position.
        * ponder
                start searching in pondering mode.
                Do not exit the search in ponder mode, even if it's mate!
                This means that the last move sent in in the position string is the ponder move.
                The engine can do what it wants to do, but after a "ponderhit" command
                it should execute the suggested move to ponder on. This means that the ponder move sent by
                the GUI can be interpreted as a recommendation about which move to ponder. However, if the
                engine decides to ponder on a different move, it should not display any mainlines as they are
                likely to be misinterpreted by the GUI because the GUI expects the engine to ponder
           on the suggested move.
        * wtime
                white has x msec left on the clock
        * btime
                black has x msec left on the clock
        * winc
                white increment per move in mseconds if x > 0
        * binc
                black increment per move in mseconds if x > 0
        * movestogo
      there are x moves to the next time control,
                this will only be sent if x > 0,
                if you don't get this and get the wtime and btime it's sudden death
        * depth
                search x plies only.
        * nodes
           search x nodes only,
        * mate
                search for a mate in x moves
        * movetime
                search exactly x mseconds
        * infinite
                search until the "stop" command. Do not exit the search without being told so in this mode!

 * stop
        stop calculating as soon as possible,
        don't forget the "bestmove" and possibly the "ponder" token when finishing the search

 * ponderhit
        the user has played the expected move. This will be sent if the engine was told to ponder on the same move
        the user has played. The engine should continue searching but switch from pondering to normal search.

 * quit
        quit the program as soon as possible


 Engine to GUI:
 --------------

 * id
        * name
                this must be sent after receiving the "uci" command to identify the engine,
                e.g. "id name Shredder X.Y\n"
        * author
                this must be sent after receiving the "uci" command to identify the engine,
                e.g. "id author Stefan MK\n"

 * uciok
        Must be sent after the id and optional options to tell the GUI that the engine
        has sent all infos and is ready in uci mode.

 * readyok
        This must be sent when the engine has received an "isready" command and has
        processed all input and is ready to accept new commands now.
        It is usually sent after a command that can take some time to be able to wait for the engine,
        but it can be used anytime, even when the engine is searching,
        and must always be answered with "isready".

 * bestmove  [ ponder  ]
        the engine has stopped searching and found the move  best in this position.
        the engine can send the move it likes to ponder on. The engine must not start pondering automatically.
        this command must always be sent if the engine stops searching, also in pondering mode if there is a
        "stop" command, so for every "go" command a "bestmove" command is needed!
        Directly before that the engine should send a final "info" command with the final search information,
        the the GUI has the complete statistics about the last search.
```

* copyprotection
this is needed for copyprotected engines. After the uciok command the engine can tell the GUI,
that it will check the copy protection now. This is done by "copyprotection checking".
If the check is ok the engine should sent "copyprotection ok", otherwise "copyprotection error".
If there is an error the engine should not function properly but should not quit alone.
If the engine reports "copyprotection error" the GUI should not use this engine
and display an error message instead!
The code in the engine can look like this
TellGUI("copyprotection checking\n");
// ... check the copy protection here ...
if(ok)
TellGUI("copyprotection ok\n");
else
TellGUI("copyprotection error\n");

* registration
this is needed for engines that need a username and/or a code to function with all features.
Analog to the "copyprotection" command the engine can send "registration checking"
after the uciok command followed by either "registration ok" or "registration error".
Also after every attempt to register the engine it should answer with "registration checking"
and then either "registration ok" or "registration error".
In contrast to the "copyprotection" command, the GUI can use the engine after the engine has
reported an error, but should inform the user that the engine is not properly registered
and might not use all its features.
In addition the GUI should offer to open a dialog to
enable registration of the engine. To try to register an engine the GUI can send
the "register" command.
The GUI has to always answer with the "register" command if the engine sends "registration error"
at engine startup (this can also be done with "register later")
and tell the user somehow that the engine is not registered.
This way the engine knows that the GUI can deal with the registration procedure and the user
will be informed that the engine is not properly registered.

* info
the engine wants to send infos to the GUI. This should be done whenever one of the info has changed.
The engine can send only selected infos and multiple infos can be send with one info command,
e.g. "info currmove e2e4 currmovenumber 1" or
"info depth 12 nodes 123456 nps 100000".
Also all infos belonging to the pv should be sent together
e.g. "info depth 2 score cp 214 time 1242 nodes 2124 nps 34928 pv e2e4 e7e5 g1f3"
I suggest to start sending "currmove", "currmovenumber", "currline" and "refutation" only after one second
to avoid too much traffic.
Additional info:
* depth
search depth in plies
* seldepth
selective search depth in plies,
if the engine sends seldepth there must also a "depth" be present in the same string.
* time
the time searched in ms, this should be sent together with the pv.
* nodes
x nodes searched, the engine should send this info regularly
* pv ...
the best line found
* multipv
this for the multi pv mode.
for the best move/pv add "multipv 1" in the string when you send the pv.
in k-best mode always send all k variants in k strings together.
* score
* cp
the score from the engine's point of view in centipawns.
* mate
mate in y moves, not plies.
If the engine is getting mated use negativ values for y.
* lowerbound
the score is just a lower bound.
* upperbound
the score is just an upper bound.
* currmove
currently searching this move
* currmovenumber
currently searching move number x, for the first move x should be 1 not 0.
* hashfull
the hash is x permill full, the engine should send this info regularly
* nps
x nodes per second searched, the engine should send this info regularly
* tbhits
x positions where found in the endgame table bases
* cpuload
the cpu usage of the engine is x permill.
* string
any string str which will be displayed be the engine,
if there is a string command the rest of the line will be interpreted as .
* refutation ...
move is refuted by the line ... , i can be any number >= 1.
Example: after move d1h5 is searched, the engine can send
"info refutation d1h5 g6h5"
if g6h5 is the best answer after d1h5 or if g6h5 refutes the move d1h5.
if there is norefutation for d1h5 found, the engine should just send
"info refutation d1h5"
The engine should only send this if the option "UCI_ShowRefutations" is set to true.
* currline ...
this is the current line the engine is calculating. is the number of the cpu if
the engine is running on more than one cpu. = 1,2,3....
if the engine is just using one cpu, can be omitted.
If is greater than 1, always send all k lines in k strings together.
The engine should only send this if the option "UCI_ShowCurrLine" is set to true.

* option
This command tells the GUI which parameters can be changed in the engine.
This should be sent once at engine startup after the "uci" and the "id" commands
if any parameter can be changed in the engine.
The GUI should parse this and build a dialog for the user to change the settings.

Note that not every option needs to appear in this dialog as some options like
"Ponder", "UCI_AnalyseMode", etc. are better handled elsewhere or are set automatically.
If the user wants to change some settings, the GUI will send a "setoption" command to the engine.
Note that the GUI need not send the setoption command when starting the engine for every option if
it doesn't want to change the default value.
For all allowed combinations see the example below,
as some combinations of this tokens don't make sense.
One string will be sent for each parameter.

* name
  The option has the name id.
  Certain options have a fixed value for , which means that the semantics of this option is fixed.
  Usually those options should not be displayed in the normal engine options window of the GUI but
  get a special treatment. "Pondering" for example should be set automatically when pondering is
  enabled or disabled in the GUI options. The same for "UCI_AnalyseMode" which should also be set
  automatically by the GUI. All those certain options have the prefix "UCI_" except for the
  first 6 options below. If the GUI get an unknown Option with the prefix "UCI_", it should just
  ignore it and not display it in the engine's options dialog.
  * = Hash, type is spin
    the value in MB for memory for hash tables can be changed,
    this should be answered with the first "setoptions" command at program boot
    if the engine has sent the appropriate "option name Hash" command,
    which should be supported by all engines!
    So the engine should use a very small hash first as default.
  * = NalimovPath, type string
    this is the path on the hard disk to the Nalimov compressed format.
    Multiple directories can be concatenated with ";"
  * = NalimovCache, type spin
    this is the size in MB for the cache for the nalimov table bases
    These last two options should also be present in the initial options exchange dialog
    when the engine is booted if the engine supports it
  * = Ponder, type check
    this means that the engine is able to ponder.
    The GUI will send this whenever pondering is possible or not.
    Note: The engine should not start pondering on its own if this is enabled, this option is only
    needed because the engine might change its time management algorithm when pondering is allowed.
  * = OwnBook, type check
    this means that the engine has its own book which is accessed by the engine itself.
    if this is set, the engine takes care of the opening book and the GUI will never
    execute a move out of its book for the engine. If this is set to false by the GUI,
    the engine should not access its own book.
  * = MultiPV, type spin
    the engine supports multi best line or k-best mode. the default value is 1
  * = UCI_ShowCurrLine, type check, should be false by default,
    the engine can show the current line it is calculating. see "info currline" above.
  * = UCI_ShowRefutations, type check, should be false by default,
    the engine can show a move and its refutation in a line. see "info refutations" above.
  * = UCI_LimitStrength, type check, should be false by default,
    The engine is able to limit its strength to a specific Elo number,
    This should always be implemented together with "UCI_Elo".
  * = UCI_Elo, type spin
    The engine can limit its strength in Elo within this interval.
    If UCI_LimitStrength is set to false, this value should be ignored.
    If UCI_LimitStrength is set to true, the engine should play with this specific strength.
    This should always be implemented together with "UCI_LimitStrength".
  * = UCI_AnalyseMode, type check
    The engine wants to behave differently when analysing or playing a game.
    For example when playing it can use some kind of learning.
    This is set to false if the engine is playing a game, otherwise it is true.
  * = UCI_Opponent, type string
    With this command the GUI can send the name, title, elo and if the engine is playing a human
    or computer to the engine.
    The format of the string has to be [GM|IM|FM|WGM|WIM|none] [|none] [computer|human]
    Example:
    "setoption name UCI_Opponent value GM 2800 human Gary Kasparow"
    "setoption name UCI_Opponent value none none computer Shredder"

* type
  The option has type t.
  There are 5 different types of options the engine can send
  * check
    a checkbox that can either be true or false
  * spin
    a spin wheel that can be an integer in a certain range
  * combo
    a combo box that can have different predefined strings as a value
  * button
    a button that can be pressed to send a command to the engine
  * string
    a text field that has a string as a value,
    an empty string has the value ""
* default
  the default value of this parameter is x
* min
  the minimum value of this parameter is x
* max
  the maximum value of this parameter is x
* var
  a predefined value of this parameter is x

Example:
Here are 5 strings for each of the 5 possible types of options
"option name Nullmove type check default true\n"
"option name Selectivity type spin default 2 min 0 max 4\n"
"option name Style type combo default Normal var Solid var Normal var Risky\n"
"option name NalimovPath type string default c:\\n"
"option name Clear Hash type button\n"

Example:

This is how the communication when the engine boots can look like:

```
GUI     engine

// tell the engine to switch to UCI mode
uci

// engine identify
      id name Shredder
               id author Stefan MK

// engine sends the options it can change
// the engine can change the hash size from 1 to 128 MB
               option name Hash type spin default 1 min 1 max 128

// the engine supports Nalimov endgame tablebases
               option name NalimovPath type string default
               option name NalimovCache type spin default 1 min 1 max 32

// the engine can switch off Nullmove and set the playing style
          option name Nullmove type check default true
               option name Style type combo default Normal var Solid var Normal var Risky

// the engine has sent all parameters and is ready
               uciok

// Note: here the GUI can already send a "quit" command if it just wants to find out
//       details about the engine, so the engine should not initialize its internal
//       parameters before here.
// now the GUI sets some values in the engine
// set hash to 32 MB
setoption name Hash value 32

// init tbs
setoption name NalimovCache value 1
setoption name NalimovPath value d:\tb;c\tb

// waiting for the engine to finish initializing
// this command and the answer is required here!
isready

// engine has finished setting up the internal values
               readyok

// now we are ready to go

// if the GUI is supporting it, tell the engine that is is
// searching on a game that is hasn't searched on before
ucinewgame

// if the engine supports the "UCI_AnalyseMode" option and the next search is supposted to
// be an analysis, the GUI should set "UCI_AnalyseMode" to true if it is currently
// set to false with this engine
setoption name UCI_AnalyseMode value true

// tell the engine to search infinite from the start position after 1.e4 e5
position startpos moves e2e4 e7e5
go infinite

// the engine starts sending infos about the search to the GUI
// (only some examples are given)


               info depth 1 seldepth 0
               info score cp 13  depth 1 nodes 13 time 15 pv f1b5
               info depth 2 seldepth 2
               info nps 15937
               info score cp 14  depth 2 nodes 255 time 15 pv f1c4 f8c5
               info depth 2 seldepth 7 nodes 255
               info depth 3 seldepth 7
               info nps 26437
               info score cp 20  depth 3 nodes 423 time 15 pv f1c4 g8f6 b1c3
               info nps 41562
               ....


// here the user has seen enough and asks to stop the searching
stop

// the engine has finished searching and is sending the bestmove command
// which is needed for every "go" command sent to tell the GUI
// that the engine is ready again
               bestmove g1f3 ponder d8f6
```