UNIVERSAL CHESS INTERFACE

2024 MAR 00 · DRAFT

Introduction

This specification governs the interaction between two processes named the *client* and the *engine*. Graphical interfaces, terminal emulators, and scripts and utilities are examples of clients. Prescriptive text begins with section 1 *Definitions*.

Getting Started

This section needs to be written! It should contain enough for a beginner to add basic UCI support to their engine. Maybe the specification should define a "basic" subset of UCI?

Conventions

A small H before a number indicates that it is written in hexadecimal; for example, ᴴ10 = 16. Byte sequences of Unicode scalar values encoded with UTF-8 are set in teal; for example, `u¢i` = ⟨ᴴ75, ᴴc2, ᴴa2, ᴴ69⟩.

Special terms defined by the specification are highlighted in purple and inline comments are set ※ after a reference mark, in grey.

> A blue box is used to describe a convention that clients and engines are encouraged to follow, usually to do with the interpretation or meaning of the messages that clients and engines send. A blue box is also used to provide a recommendation, usually to do with implementation-defined behavior.

> A grey box is used to provide an explanative note or comment.

> A green box is used to provide an example of conforming behavior.

Text within colored boxes is nonnormative.

1 Definitions

1·1 The engine's standard input and output must be open file descriptors, and the engine's standard error must be an open file descriptor until closed by the engine. The sequence of bytes that the client sends to the engine via the engine's standard input is the input sequence. The sequence of bytes that the engine sends to the client via the engine's standard output is the output sequence.

> There are no requirements for an engine's standard error except that it be open. For example, the engine's standard error may be directed to a null file, to the client's standard output or standard error, or to a log file. There are accordingly no restrictions on the sequence of bytes that the engine writes to its standard error.

```
let read byte! : File Descriptor → Byte | EOF
```

1·2 A message terminator is the byte pair ⟨"0d, "0a⟩ or the byte "0a alone when it is not preceded by the byte "0d. ※ Decoded as UTF-8, these bytes are U+000D CARRIAGE RETURN and U+000A LINE FEED.

> This disallows some pairs of Unicode scalar values encoded with UTF-16, such as ⟨U+010D LATIN SMALL LETTER C WITH CARON, U+0A05 GURMUKHI LETTER A⟩ encoded as UTF-16 in big-endian byte order. It's assumed that such pairs will rarely occur, and that the inconvenience of this limitation is less than the difficulties that would arise if the specification attempted to require locale awareness.

> In almost all cases, the input and output sequences should be valid UTF-8. However, the input and output sequences are not *required* to be valid UTF-8, which allows clients and engines to send most file system paths (which may be opaque sequences of bytes, as in Linux, or unicode scalar values encoded as UTF-16, as in Windows) in their native representation.

1·3 Message terminators divide the input sequence and output sequence into messages, that is, a message is a (possibly empty) subsequence of bytes that does not contain a message terminator, and every byte of the input and output sequences is either part of a message or a message terminator. The messages within the input sequence are client messages and the messages within the output sequence are engine messages.

```
def read message!(fd : File Descriptor) : List(Byte) | EOF
    seq ← empty
    hd ← read byte!(fd)
    repeat
        match hd
            eof ⇒ return (if empty?(seq) then eof else seq)
            "0d ⇒ match read byte!(fd)
                eof ⇒ append!(seq, hd) • return seq
                "0a ⇒ return seq
                pk ⇒ append!(seq, hd) • hd ← pk
            "0a ⇒ return seq
            ⋆ ⇒ append!(seq, hd) • hd ← read byte!(fd)
```

1·4 The byte "20 divides messages into tokens, that is, a token is a sequence of bytes that are not "20, and every byte of a message is either part of a token or is "20. ※ Decoded as UTF-8, this is U+0020 SPACE.

> In some cases, only the beginning of a message will be viewed as a collection of tokens and the remainder will be viewed as a continuous byte sequence.

```
def get token!(seq : List(Byte)) → List(Byte) | None
    return none if empty?(seq)
    while first(seq) = "20
        shift!(seq) • return none if empty?(seq)
    tok ← empty
    while first(seq) ≠ "20
        append!(tok, first(seq))
        shift!(seq) • return tok if empty?(seq)
    while first(seq) = "20
        shift!(seq) • return tok if empty?(seq)
    return tok
```

1·5 An algebraic move token is a token of the form

[a–h][1–8][a–h][1–8][qrbn]?

1·6 A Forsyth–Edwards Notation (FEN) sequence is a sequence of six tokens ⟨*board*, *side-to-move*, *rights*, EP-*target*, DFZ, *move-number*⟩ of the form

```
       board  =  row/row/row/row/row/row/row/row
         row  =  8 | [1–8]?([KQRBNPkqrbnp][1–8]?)+
side-to-move  =  w | b
      rights  =  - | ((K?Q?k?q?) ∩ [KQkq]+)
   EP-target  =  - | [a–h][36]
         DFZ  =  0 | [1–9][0–9]*
 move-number  =  [1–9][0–9]*
```

※ The dash “-” is U+002D HYPHEN-MINUS.

For each *row* of *board*, map K, Q, …, p to 1 and map 1, 2, …, 8 to 1, 2, …, 8. The sum of the numbers must be 8.

1·7 TODO *FEN corresponds to a chess position in the natural way, as described elsewhere. Note need not be reachable from starting position. Define legality of a move from a position.*

1·8 A violation is any violation of the requirements of the specification by the client or engine. When a violation occurs, or when the requirements of the specification are otherwise not met, the specification imposes no further requirements on the behavior of the client or engine.

1·9 An error is a condition that resembles a violation but that the client and engine are expected to handle.

2 Client Messages

2·1 TODO

3 Engine Messages

3·1 TODO

4 State Machine

4·1 TODO

5 Notation

5·1 TODO *explain the language used for reference implementation*