Chess Engine Communication Protocol

Tim Mann & H.G.Muller

Version 2; implemented in xboard/WinBoard 4.2.1 and later. (Sept 3, 2009)
Changes since version 1 are indicated in red.
Changes for WinBoard 4.3.xx are indicated in green.
Changes for WinBoard 4.4.xx are indicated in blue.

---

1. INTRODUCTION

This document is a set of rough notes on the protocol that xboard and WinBoard use to communicate with

gnuchessx and other chess engines. These notes may be useful if you want to connect a different chess engine to xboard. Throughout the notes, "xboard" means both xboard and WinBoard except where they are specifically contrasted.

There are two reasons I can imagine someone wanting to do this:

1. You have, or are developing, a chess engine but you don't want to write your own graphical interface.

2. You have, or are developing,a chess engine, and you want to interface it to the Internet Chess Server.

In case (2), if you are using xboard, you will need to configure the "Zippy" code into it, but WinBoard includes this code already. See the file zippy.README in the xboard or WinBoard distribution for more information.

These notes are unpolished, but I've attempted to make them complete in this release. If you notice any errors, omissions, or misleading statements, let me know.

I'd like to hear from everyone who is trying to interface their own chess engine to xboard/WinBoard. Please join the mailing list for authors of xboard/WinBoard compatible chess engines and post a message about what you're doing. The list is now hosted by Yahoo Groups; you can join at http://groups.yahoo.com/group/chess-engines, or you can read the list there without joining. The list is filtered to prevent spam.

Note that the WinBoard 4.3.xx line was developed independently of the original GNU project, by H.G.Muller. If you have questions about WinBoard 4.3.xx, or want to report bugs in it, report them in the appropriate section of the WinBoard forum.

2. CONNECTION

An xboard chess engine runs as a separate process from xboard itself, connected to xboard through a pair of anonymous pipes. The engine does not have to do anything special to set up these pipes. xboard sets up the pipes itself and starts the engine with one pipe as its standard input and the other as its standard output. The engine then reads commands from its standard input and writes responses to its standard output. This is, unfortunately, a little more complicated to do right than it sounds; see section 6 below.

And yes, contrary to some people's expectations, exactly the same thing is true for WinBoard. Pipes and standard input/output are implemented in Win32 and work fine. You don't have to use DDE, COM, DLLs, BSOD, or any of the other infinite complexity that Microsoft has created just to talk between two programs. A WinBoard chess engine is a Win32 console program that simply reads from its standard input and writes to its standard output. See sections 5 and 6 below for additional details.

3. DEBUGGING

To diagnose problems in your engine's interaction with xboard, use the -debug flag on xboard's command line to see the messages that are being exchanged. In WinBoard, these messages are written to the file WinBoard.debug instead of going to the screen.

You can turn debug mode on or off while WinBoard is running by pressing Ctrl+Alt+F12. You can turn debug mode on or off while xboard is running by binding DebugProc to a shortcut key (and pressing the key!); see the instructions on shortcut keys in the xboard man page.

While your engine is running under xboard/WinBoard, you can send a command directly to the engine by pressing Shift+1 (xboard) or Alt+1 (WinBoard 4.0.3 and later). This brings up a dialog that you can type your command into. Press Shift+2 (Alt+2) instead to send to the second chess engine in Two Machines mode. On WinBoard 4.0.2 and earlier, Ctrl+Alt is used in place of Alt; this had to be changed due to a conflict with typing the @-sign on some European keyboards.

4. HOW IT GOT THIS WAY

Originally, xboard was just trying to talk to the existing command-line interface of GNU Chess 3.1+ and 4, which was designed for people to type commands to. So the communication protocol is very ad-hoc. It might have been good to redesign it early on, but because xboard and GNU Chess are separate programs, I didn't want to force people to upgrade them together to versions that matched. I particularly wanted to keep new versions of xboard working with old versions of GNU Chess, to make it easier to compare the play of old and new gnuchess versions. I didn't foresee the need for a clean protocol to be used with other chess engines in the future.

Circumstances have changed over the years, and now there are many more engines that work with xboard. I've had to make the protocol description more precise, I've added some features that GNU Chess does not support, and I've specified the standard semantics of a few features to be slightly different from what GNU Chess 4 does.

This release of the protocol specification is the first to carry a version number of its own -- version 2. Previous releases simply carried a last-modified date and were loosely tied to specific releases of xboard and WinBoard. The version number "1" applies generally to all those older versions of the protocol.

Protocol version 2 remains compatible with older engines but has several new capabilities. In particular, it adds the "feature" command, a new mechanism for making backward-compatible changes and extensions to the protocol. Engines that do not support a particular new feature do not have to use it; new features are not

enabled unless the engine specifically requests them using the feature command. If an engine does not send the feature command at all, the protocol behavior is nearly identical to version 1. Several new features can be selected by the feature command in version 2, including the "ping" command (recommended for all engines), the "setboard" command, and many optional parameters. Additional features will probably be added in future versions.

If it is necessary to have a separate name, it would be best to refer to the protocol including the green additions as version 2f. I really don't think it is a different protocol from version 2, though. I just tried to clarify some ambiguities in the original definition, now that the WinBoard 4.3.xx line has implemented them in a specific way. The hand-shaking protocol for features as defined in protocol 2 perfectly allows addition of an occasional new features without any need for stepping up the protocol version number, and I think refraining from the latter would enormously lower the barrier for actual implementation of these features in engines.
The two really new things are the engine debug comments, and the "nps" command. The former merely tries to regulate an extremely common existing pactice of having engines dump debug messages on WinBoard in an unprotected way, as usually you get away with that.

5. WINBOARD REQUIRES WIN32 ENGINES

Due to some Microsoft brain damage that I don't understand, WinBoard does not work with chess engines that were compiled to use a DOS extender for 32-bit addressing. (Probably not with 16-bit DOS or Windows programs either.) WinBoard works only with engines that are compiled for the Win32 API. You can get a free compiler that targets the Win32 API from http://sources.redhat.com/cygwin/. I think DJGPP 2.x should also work if you use the RSXNTDJ extension, but I haven't tried it. Of course, Microsoft Visual C++ will work. Most likely the other commercial products that support Win32 will work too (Borland, etc.), but I have not tried them. Delphi has been successfully used to write engines for WinBoard; if you want to do this, Tony Werten has donated some sample code that should help you get started.

6. HINTS ON INPUT/OUTPUT

Beware of using buffered I/O in your chess engine. The C stdio library, C++ streams, and the I/O packages in most other languages use buffering both on input and output. That means two things. First, when your engine tries to write some characters to xboard, the library stashes them in an internal buffer and does not actually write them to the pipe connected to xboard until either the buffer fills up or you call a special library routine asking for it to be flushed. (In C stdio, this routine is named `fflush`.) Second, when your engine tries to read some characters from xboard, the library does not read just the characters you asked for -- it reads all the characters that are currently available (up to some limit) and stashes any characters you are not yet ready for in an internal buffer. The next time you ask to read, you get the characters from the buffer (if any) before the library tries to read more data from the actual pipe.

Why does this cause problems? First, on the output side, remember that your engine produces output in small quantities (say, a few characters for a move, or a line or two giving the current analysis), and that data always needs to be delivered to xboard/WinBoard for display immediately. If you use buffered output, the data you print will sit in a buffer in your own address space instead of being delivered.

You can usually fix the output buffering problem by asking for the buffering to be turned off. In C stdio, you do this by calling `setbuf(stdout, NULL)`. A more laborious and error-prone method is to carefully call `fflush(stdout)` after every line you output; I don't recommend this. In C++, you can try `cout.setf(ios::unitbuf)`, which is documented in current editions of "The C++ Programming Language," but not older ones. Another C++ method that might work is `cout.rdbuf()->setbuf(NULL, 0)`. Alternatively, you can carefully call `cout.flush()` after every line you output; again, I don't recommend this.

Another way to fix the problem is to use unbuffered operating system calls to write directly to the file descriptor for standard output. On Unix, this means `write(1, ...)` -- see the man page for write(2). On Win32, you can use either the Unix-like `_write(1, ...)` or Win32 native routines like `WriteFile`.

Second, on the input side, you are likely to want to poll during your search and stop it if new input has come in. If

you implement pondering, you'll need this so that pondering stops when the user makes a move. You should also poll during normal thinking on your move, so that you can implement the "?" (move now) command, and so that you can respond promptly to a "result", "force", or "quit" command if xboard wants to end the game or terminate your engine. Buffered input makes polling more complicated -- when you poll, you must stop your search if there are *either* characters in the buffer *or* characters available from the underlying file descriptor.

The most direct way to fix this problem is to use unbuffered operating system calls to read (and poll) the underlying file descriptor directly. On Unix, use `read(0, ...)` to read from standard input, and use `select()` to poll it. See the man pages read(2) and select(2). (Don't follow the example of GNU Chess 4 and use the FIONREAD ioctl to poll for input. It is not very portable; that is, it does not exist on all versions of Unix, and is broken on some that do have it.) On Win32, you can use either the Unix-like `_read(0, ...)` or the native Win32 `ReadFile()` to read. Unfortunately, under Win32, the function to use for polling is different depending on whether the input device is a pipe, a console, or something else. (More Microsoft brain damage here -- did they never hear of device independence?) For pipes, you can use `PeekNamedPipe` to poll (even when the pipe is unnamed). For consoles, you can use `GetNumberOfConsoleInputEvents`. For sockets only, you can use `select()`. It might be possible to use `WaitForSingleObject` more generally, but I have not tried it. Some code to do these things can be found in Crafty's utility.c, but I don't guarantee that it's all correct or optimal.

A second way to fix the problem might be to ask your I/O library not to buffer on input. It should then be safe to poll the underlying file descriptor as described above. With C, you can try calling `setbuf(stdin, NULL)`. However, I have never tried this. Also, there could be problems if you use `scanf()`, at least with certain patterns, because `scanf()` sometimes needs to read one extra character and "push it back" into the buffer; hence, there is a one-character pushback buffer even if you asked for stdio to be unbuffered. With C++, you can try `cin.rdbuf()->setbuf(NULL, 0)`, but again, I have never tried this.

A third way to fix the problem is to check whether there are characters in the buffer whenever you poll. C I/O libraries generally do not provide any portable way to do this. Under C++, you can use `cin.rdbuf()->in_avail()`. This method has been reported to work with EXchess. Remember that if there are no characters in the buffer, you still have to poll the underlying file descriptor too, using the method described above.

A fourth way to fix the problem is to use a separate thread to read from stdin. This way works well if you are familiar with thread programming. This thread can be blocked waiting for input to come in at all times, while the main thread of your engine does its thinking. When input arrives, you have the thread put the input into a buffer and set a flag in a global variable. Your search routine then periodically tests the global variable to see if there is input to process, and stops if there is. WinBoard and my Win32 ports of ICC timestamp and FICS timeseal use threads to handle multiple input sources.

7. SIGNALS

Engines that run on Unix need to be concerned with two Unix signals: `SIGTERM` and `SIGINT`. This applies both to engines that run under xboard and (the unusual case of) engines that WinBoard remotely runs on a Unix host using the -firstHost or -secondHost feature. It does not apply to engines that run on Windows, because Windows does not have Unix-style signals. Beginning with version 2, you can now turn off the use of either or both signals. See the "feature" command in section 9 below.

First, when an engine is sent the "quit" command, it is also given a `SIGTERM` signal shortly afterward to make sure it goes away. If your engine reliably responds to "quit", and the signal causes problems for you, you should either ignore it by calling `signal(SIGTERM, SIG_IGN)` at the start of your program, or disable it with the "feature" command.

Second, xboard will send an interrupt signal (`SIGINT`) at certain times when it believes the engine may not be listening to user input (thinking or pondering). WinBoard currently does this only when the engine is running remotely using the -firstHost or -secondHost feature, not when it is running locally. You probably need to know only enough about this grungy feature to keep it from getting in your way.

The `SIGINT`s are basically tailored to the needs of GNU Chess 4 on systems where its input polling code is broken or disabled. Because they work in a rather peculiar way, it is recommended that you either ignore

SIGINT by having your engine call signal(SIGINT, SIG_IGN), or disable it with the "feature" command.

Here are details for the curious. If xboard needs to send a command when it is the chess engine's move (such as before the "?" command), it sends a SIGINT first. If xboard needs to send commands when it is not the chess engine's move, but the chess engine may be pondering (thinking on its opponent's time) or analyzing (analysis or analyze file mode), xboard sends a SIGINT before the first such command only. Another SIGINT is not sent until another move is made, even if xboard issues more commands. This behavior is necessary for GNU Chess 4. The first SIGINT stops it from pondering until the next move, but on some systems, GNU Chess 4 will die if it receives a SIGINT when not actually thinking or pondering.

There are two reasons why WinBoard does not send the Win32 equivalent of SIGINT (which is called CTRL_C_EVENT) to local engines. First, the Win32 GNU Chess 4 port does not need it. Second, I could not find a way to get it to work. Win32 seems to be designed under the assumption that only console applications, not windowed applications, would ever want to send a CTRL_C_EVENT.

8. COMMANDS FROM XBOARD TO THE ENGINE

All commands from xboard to the engine end with a newline (\n), even where that is not explicitly stated. All your output to xboard must be in complete lines; any form of prompt or partial line will cause problems.

At the beginning of each game, xboard sends an initialization string. This is currently "new\nrandom\n" unless the user changes it with the initString or secondInitString option.

xboard normally reuses the same chess engine process for multiple games. At the end of a game, xboard will send the "force" command (see below) to make sure your engine stops thinking about the current position. It will later send the initString again to start a new game. If your engine can't play multiple games, you can disable reuse either with the "feature" command (beginning in protocol version 2; see below) or with xboard's -xreuse (or -xreuse2) command line option. xboard will then ask the process to quit after each game and start a new

process for the next game.

xboard

This command will be sent once immediately after your engine process is started. You can use it to put your engine into "xboard mode" if that is needed. If your engine prints a prompt to ask for user input, you must turn off the prompt and output a newline when the "xboard" command comes in.

protover N

Beginning in protocol version 2 (in which N=2), this command will be sent immediately after the "xboard" command. If you receive some other command immediately after "xboard" (such as "new"), you can assume that protocol version 1 is in use. The "protover" command is the only new command that xboard always sends in version 2. All other new commands to the engine are sent only if the engine first enables them with the "feature" command. Protocol versions will always be simple integers so that they can easily be compared.

Your engine should reply to the protover command by sending the "feature" command (see below) with the list of non-default feature settings that you require, if any.

Your engine should never refuse to run due to receiving a higher protocol version number than it is expecting! New protocol versions will always be compatible with older ones by default; the larger version number is simply a hint that additional "feature" command options added in later protocol versions may be accepted.

accepted

rejected

These commands may be sent to your engine in reply to the "feature" command; see its documentation below.

new

Reset the board to the standard chess starting position. Set White on move. Leave force mode and set the engine to play Black. Associate the engine's clock with Black and the opponent's clock with White. Reset clocks and time controls to the start of a new game. Use wall clock for time measurement. Stop clocks. Do not ponder on this move, even if pondering is on. Remove any search depth limit previously set by the sd command.

variant VARNAME

If the game is not standard chess, but a variant, this command is sent after "new" and before the first move or "edit" command. Currently defined variant names are:

wildcastle	Shuffle chess where king can castle from d file
nocastle	Shuffle chess with no castling at all
fischerandom	Fischer Random
bughouse	Bughouse, ICC/FICS rules
crazyhouse	Crazyhouse, ICC/FICS rules
losers	Win by losing all pieces or getting mated (ICC)
suicide	Win by losing all pieces including king, or by having fewer pieces when one player

suicide	has no legal moves (FICS)
giveaway	Win by losing all pieces including king, or by having no legal moves (ICC)
twokings	Weird ICC wild 9
kriegspiel	Kriegspiel (engines not supported)
atomic	Atomic
3check	Win by giving check 3 times
xiangqi	Chinese Chess (9x10 board)
shogi	Japanese Chess (9x9 bord)
capablanca	Capablanca Chess (10x8 board, with Archbishop and Chancellor)
gothic	Gothic Chess (10x8 board, same with better opening setup)
falcon	Falcon Chess (10x8 board, with two Falcon pieces)
shatranj	ancient Arabic Chess, with Elephants and General in stead of B and Q
courier	Courier Chess (12x8 board, a medieval precursor of modern Chess
knightmate	King moves as Knight and vice versa
berolina	Pawns capture straight ahead, and move diagonally

janus	Janus Chess (10x8, with two Archbishops)
caparandom	shuffle variant like FRC (10x8 board)
cylinder	Pieces wrap around between side edges, like board is a cylinder
super	Superchess: a shuffle variant with 4 fairy pieces on 8x8 board
great	Great Shatranj: sliders are replaced by corresponding short-range pieces on a 10x8 board
lion	Mighty-Lion Chess, with a super-knight, more powerful than a Queen
elven	Elven Chess: hybrid between Chess and Chu Shogi on 10x10 board
chu	Chu Shogi: Edo-period Japanese Chess on a 12x12 board
unknown	Unknown variant (not supported)

As of XBoard 4.8, engines can define arbitrary variant names; see the "feature" and "setup" commands in section 9.

quit

The chess engine should immediately exit. This command is used when xboard is itself exiting, and also between games if the -xreuse command line option is given (or -xreuse2 for the second engine). See also Signals above.

random

This command is specific to GNU Chess 4. You can either ignore it completely (that is, treat it as a no-op) or implement it as GNU Chess does. The command toggles "random" mode (that is, it sets random = !random). In random mode, the engine adds a small random value to its evaluation function to vary its play. The "new" command sets random mode off.

force

Set the engine to play neither color ("force mode"). Stop clocks. The engine should check that moves received in force mode are legal and made in the proper turn, but should not think, ponder, or make moves of its own.

go

Leave force mode and set the engine to play the color that is on move. Associate the engine's clock with the color that is on move, the opponent's clock with the color that is not on move. Start the engine's clock. Start thinking and eventually make a move.

playother

(This command is new in protocol version 2. It is not sent unless you enable it with the feature command.) Leave force mode and set the engine to play the color that is *not* on move. Associate the opponent's clock with the color that is on move, the engine's clock with the color that is not on move. Start the opponent's clock. If pondering is enabled, the engine should begin pondering. If the engine later receives a move, it should start thinking and eventually reply.

white

(This command is obsolete as of protocol version 2, but is still sent in some situations to accommodate older engines unless you disable it with the feature command.) Set White on move. Set the engine to play Black. Stop clocks.

black

(This command is obsolete as of protocol version 2, but is still sent in some situations to accommodate older engines unless you disable it with the feature command.) Set Black on move. Set the engine to play White. Stop clocks.

level MPS BASE INC

Set time controls. See the Time Control section below.

st TIME

Set time controls. See the Time Control section below.

sd DEPTH

The engine should limit its thinking to DEPTH ply. The commands "level" or "st" and "sd" can be used together in an orthogonal way. If both are issued, the engine should observe both limitations: In the protocol, the "sd" command isn't a time control. It doesn't say that your engine has unlimited time but must search to exactly the given depth. It says that you should pay attention to the time control as normal, but cut off the search at the specified depth even if you have time to search deeper. If you don't have time to search to the specified depth, given your normal time management algorithm, then you will want to stop sooner than the given depth.

The "new" command should set the search depth back to unlimited. This is already stated in the spec. The "level" command should not affect the search depth. As it happens, xboard/WinBoard currently always sends sd (if needed) right after level, but that isn't part of the spec.

nps NODE_RATE

The engine should not use wall-clock time to make its timing decisions, but an own internal time measure based on the number of nodes it has searched (and will report as "thinking output", see section 10), converted to seconds through dividing by the given NODE_RATE. Example: after receiving the commands "st 8" and "nps 10000", the engine should never use more that 80,000 nodes in the search for any move. In this mode, the engine should report user CPU time used (in its thinking output), rather than wall-clock time. This even holds if NODE_RATE is given as 0, but in that case it should also use the user CPU time for its timing decisions. The effect of an "nps" command should persist until the next "new" command.

time N

Set a clock that always belongs to the engine. N is a number in centiseconds (units of 1/100 second). Even if the engine changes to playing the opposite color, this clock remains with the engine.

otim N

Set a clock that always belongs to the opponent. N is a number in centiseconds (units of 1/100 second). Even if the opponent changes to playing the opposite color, this clock remains with the opponent.

If needed for purposes of board display in force mode (where the engine is not participating in the game) the time clock should be associated with the last color that the engine was set to play, the otim clock with the opposite color.

This business of "clocks remaining with the engine" is apparently so ambiguous that many engines implement it wrong. The clocks in fact always remain with the color. Which clock reading is relayed with "time", and which by "otim", is determined by which side the engine plays. Note that the way the clocks operate and receive extra time (in accordance with the selected time control) is not affected in any way by which moves are made by the engine, which by the opponent, and which were forced.

Beginning in protocol version 2, if you can't handle the time and otim commands, you can use the "feature" command to disable them; see below. The following techniques from older protocol versions also work: You can ignore the time and otim commands (that is, treat them as no-ops), or send back "Error (unknown command): time" the first time you see "time".

MOVE

See below for the syntax of moves. If the move is illegal, print an error message; see the section "Commands from the engine to xboard". If the move is legal and in turn, make it. If not in force mode, stop the opponent's clock, start the engine's clock, start thinking, and eventually make a move.

When xboard sends your engine a move, it normally sends coordinate algebraic notation. Examples:

Normal moves:	e2e4
Pawn promotion:	e7e8q
Castling:	e1g1, e1c1, e8g8, e8c8
Bughouse/crazyhouse drop:	P@h3
ICS Wild 0/1 castling:	d1f1, d1b1, d8f8, d8b8

FischerRandom castling:	O-O, O-O-O (oh, not zero)
Multi-leg move:	c4d5,d5e4 (legs separated by comma)
Null move:	@@@@

Note that on boards with exactly 10 ranks, counting of the ranks starts at 0.

Beginning in protocol version 2, you can use the feature command to select SAN (standard algebraic notation) instead; for example, e4, Nf3, exd5, Bxf7+, Qxf7#, e8=Q, O-O, or P@h3. Note that the last form, P@h3, is a extension to the PGN standard's definition of SAN, which does not support bughouse or crazyhouse.

xboard doesn't reliably detect illegal moves, because it does not keep track of castling unavailability due to king or rook moves, or en passant availability. If xboard sends an illegal move, send back an error message so that xboard can retract it and inform the user; see the section "Commands from the engine to xboard".

usermove MOVE

By default, moves are sent to the engine without a command name; the notation is just sent as a line by itself. Beginning in protocol version 2, you can use the feature command to cause the command name "usermove" to be sent before the move. Example: "usermove e2e4".

?

Move now. If your engine is thinking, it should move immediately; otherwise, the command should be ignored (treated as a no-op). It is permissible for your engine to always ignore the ? command. The only bad consequence is that xboard's Move Now menu command will do nothing.

It is also permissible for your engine to move immediately if it gets any command while thinking, as long as it processes the command right after moving, but it's preferable if you don't do this. For example, xboard may send post, nopost, easy, hard, force, quit, or other commands while the engine is on move.

ping N

In this command, N is a decimal number. When you receive the command, reply by sending the string **pong N**, where N is the same number you received. Important: You must not reply to a "ping" command until you have finished executing all commands that you received before it. Pondering does not count; if you receive a ping while pondering, you should reply immediately and continue pondering. Because of the way xboard uses the ping command, if you implement the other commands in this protocol, you should never see a "ping" command when it is your move; however, if you do, you must not send the "pong" reply to xboard until after you send your move. For example, xboard may send "?" immediately followed by "ping". If you implement the "?" command, you will have moved by the time you see the subsequent ping command. Similarly, xboard may send a sequence like "force", "new", "ping". You must not send the pong response until after you have finished executing the "new" command and are ready for the new game to start.

The ping command is new in protocol version 2 and will not be sent unless you enable it with the "feature" command. Its purpose is to allow several race conditions that could occur in previous versions of the protocol to be fixed, so it is highly recommended that you implement it. It is especially important in simple engines that do not ponder and do not poll for input while thinking, but it is needed in all engines.

draw

The engine's opponent offers the engine a draw. To accept the draw, send "offer draw". To decline, ignore the offer (that is, send nothing). If you're playing on ICS, it's possible for the draw offer to have

been withdrawn by the time you accept it, so don't assume the game is over because you accept a draw offer. Continue playing until xboard tells you the game is over. See also "offer draw" below.

result RESULT {COMMENT}

After the end of each game, xboard will send you a result command. You can use this command to trigger learning. RESULT is either 1-0, 0-1, 1/2-1/2, or *, indicating whether white won, black won, the game was a draw, or the game was unfinished. The COMMENT string is purely a human-readable comment; its content is unspecified and subject to change. In ICS mode, it is passed through from ICS uninterpreted. Example:

```
result 1-0 {White mates}
```

Here are some notes on interpreting the "result" command. Some apply only to playing on ICS ("Zippy" mode).

If you won but did not just play a mate, your opponent must have resigned or forfeited. If you lost but were not just mated, you probably forfeited on time, or perhaps the operator resigned manually. If there was a draw for some nonobvious reason, perhaps your opponent called your flag when he had insufficient mating material (or vice versa), or perhaps the operator agreed to a draw manually.

You will get a result command even if you already know the game ended -- for example, after you just checkmated your opponent. In fact, if you send the "RESULT {COMMENT}" command (discussed below), you will simply get the same thing fed back to you with "result" tacked in front. You might not always get a "result *" command, however. In particular, you won't get one in local chess engine mode when the user stops playing by selecting Reset, Edit Game, Exit or the like.

setboard FEN

The setboard command is the new way to set up positions, beginning in protocol version 2. It is not used unless it has been selected with the feature command. Here FEN is a position in Forsythe-Edwards Notation, as defined in the PGN standard. Note that this PGN standard referred to here only applies to normal Chess; Obviously in variants that cannot be described by a FEN for normal Chess, e.g. because the board is not 8x8, other pieces then PNBRQK participate, there are holdings that need to be specified, etc., xboard will use a FEN format that is standard or suitable for that variant. In particular, in FRC or CRC, WinBoard will use Shredder-FEN or X-FEN standard, i.e. it can use the rook-file indicator letter to represent a castling right (like HAha) whenever it wants, but if it uses KQkq, this will always refer to the outermost rook on the given side.

Illegal positions: Note that either setboard or edit can be used to send an illegal position to the engine. The user can create any position with xboard's Edit Position command (even, say, an empty board, or a board with 64 white kings and no black ones). If your engine receives a position that it considers illegal, I suggest that you send the response "tellusererror Illegal position", and then respond to any attempted move with "Illegal move" until the next new, edit, or setboard command.

edit

The edit command is the old way to set up positions. For compatibility with old engines, it is still used by default, but new engines may prefer to use the feature command (see below) to cause xboard to use setboard instead. The edit command puts the chess engine into a special mode, where it accepts the following subcommands:

c	change current piece color, initially white
Pa4 (for example)	place pawn of current color on a4
xa4 (for example)	empty the square a4 (not used by xboard)

#	clear board
.	leave edit mode

See the Idioms section below for additional subcommands used in ChessBase's implementation of the protocol.

The edit command does not change the side to move. To set up a black-on-move position, xboard uses the following command sequence:

```
new
force
a2a3
edit
<edit commands>
.
```

This sequence is used to avoid the "black" command, which is now considered obsolete and which many engines never did implement as specified in this document.

After an edit command is complete, if a king and a rook are on their home squares, castling is assumed to be available to them. En passant capture is assumed to be illegal on the current move regardless of the positions of the pawns. The clock for the 50 move rule starts at zero, and for purposes of the draw by repetition rule, no prior positions are deemed to have occurred. In FRC or CRC, any rook and king put on the back rank should be considered to have castling rights, even if it later becomes apparent that they cannot be both in the initial position, because the position just set up is asymmetric. It is upto WinBoard to find work-around in cases where this is not desired, similar to the "black kludge" shown above, by setting up an earlier position, and then do a move to destroy castling rights or create e.p. rights. (Don't bet your life on it...)

hint

If the user asks for a hint, xboard sends your engine the command "hint". Your engine should respond with "Hint: xxx", where xxx is a suggested move. If there is no move to suggest, you can ignore the hint command (that is, treat it as a no-op).

bk

If the user selects "Book" from the xboard menu, xboard will send your engine the command "bk". You can send any text you like as the response, as long as each line begins with a blank space or tab (\t) character, and you send an empty line at the end. The text pops up in a modal information dialog.

undo

If the user asks to back up one move, xboard will send you the "undo" command. xboard will not send this command without putting you in "force" mode first, so you don't have to worry about what should happen if the user asks to undo a move your engine made. (GNU Chess 4 actually switches to playing the opposite color in this case.)

remove

If the user asks to retract a move, xboard will send you the "remove" command. It sends this command only when the user is on move. Your engine should undo the last two moves (one for each player) and continue playing the same color.

hard

Turn on pondering (thinking on the opponent's time, also known as "permanent brain"). xboard will not

make any assumption about what your default is for pondering or whether "new" affects this setting.

easy

Turn off pondering.

post

Turn on thinking/pondering output. See Thinking Output section.

nopost

Turn off thinking/pondering output.

analyze

Enter analyze mode. See Analyze Mode section.

name X

This command informs the engine of its opponent's name. When the engine is playing on a chess server, xboard obtains the opponent's name from the server. When the engine is playing locally against a human user, xboard obtains the user's login name from the local operating system. When the engine is playing locally against another engine, xboard uses either the other engine's filename or the name that the other engine supplied in the myname option to the feature command. By default, xboard uses the name command only when the engine is playing on a chess server. Beginning in protocol version 2, you can change this with the name option to the feature command; see below.

rating

In ICS mode, xboard obtains the ICS opponent's rating from the "Creating:" message that appears before each game. (This message may not appear on servers using outdated versions of the FICS code.) In Zippy mode, it sends these ratings on to the chess engine using the "rating" command. The chess engine's own rating comes first, and if either opponent is not rated, his rating is given as 0. In the future this command may also be used in other modes, if ratings are known. Example:

```
rating 2600 1500
```

ics HOSTNAME

If HOSTNAME is "-", the engine is playing against a local opponent; otherwise, the engine is playing on an Internet Chess Server (ICS) with the given hostname. This command is new in protocol version 2 and is not sent unless the engine has enabled it with the "feature" command. Example: "ics freechess.org"

computer

The opponent is also a computer chess engine. Some engines alter their playing style when they receive this command.

pause

resume

(These commands are new in protocol version 2 and will not be sent unless feature pause=1 is set. At this writing, xboard actually does not use the commands at all, but it or other interfaces may use them in the future.) The "pause" command puts the engine into a special state where it does not think, ponder, or otherwise consume significant CPU time. The current thinking or pondering (if any) is suspended and

both player's clocks are stopped. The only command that the interface may send to the engine while it is in the paused state is "resume". The paused thinking or pondering (if any) resumes from exactly where it left off, and the clock of the player on move resumes running from where it stopped.

memory N

This command informs the engine on how much memory it is allowed to use maximally, in MegaBytes. On receipt of this command, the engine should adapt the size of its hash tables accordingly. This command does only fix the total memory use, the engine has to decide for itself (or be configured by the user by other means) how to divide up the available memory between the various tables it wants to use (e.g. main hash, pawn hash, tablebase cache, bitbases). This command will only be sent to engines that have requested it through the memory feature, and only at the start of a game, as the first of the commands to relay engine option settings just before each "new" command.

cores N

This command informs the engine on how many CPU cores it is allowed to use maximally. This could be interpreted as the number of search threads for SMP engines. (Threads that do not consume significant amounts of CPU time, like I/O threads, need not be included in the count.) This command will only be sent to engines that have requested it through the smp feature. The engine should be able to respond to the "cores" command any time during a game, but it is allowed to finish a search in progress before procesing the command. (Obeying the command should take priority over finishing a ponder search, though.) In any case it will be sent at the start of every game as the last command to relay engine option settings before the "new" command.

egtpath TYPE PATH

This command informs the engine in which directory (given by the PATH argument) it can find end-game

tables of the specified TYPE. The TYPE argument can be any character string which does not contain spaces. Currently **nalimov** and **scorpio** are defined types, for Nalimov tablebases and Scorpio bitbases, respectively, but future developers of other formats are free to define their own format names. The GUI simply matches the TYPE names the engine says it supports with those that the user supplied when configuring xboard. For every match, it sends a separate "y" command. The PATH argument would normally (for Nalimov) be the pathname of the directory the EGT files are in, but could also be the name of a file, or in fact anything the particular EGT type requires. It is upto the developer of the EGT format to specify the syntax of this parameter. This command will only be sent to engines that have told the GUI they support EGTs of the given TYPE through the egt feature. It will be sent at the start of each game, before the "new" command.

option NAME[=VALUE]

This command changes the setting of the option NAME defined by the engine (through an earlier feature command) to the given VALUE. XBoard will in general have no idea what the option means, and will send the command only when a user changes the value of this option through a menu, or at startup of the engine (before the first 'cores' command or, if that is not sent, the first 'new' command) in reaction to command-line options. The NAME echoes back to the engine the string that was identified as an option NAME in the feature command defining the option. The VALUE is of the type (numeric or text or absent) that was implied by the option type specified in this feature command, i.e. with 'spin' and 'check' options VALUE will be a decimal integer (in the latter case 0 or 1), with 'combo' and 'string' options VALUE will be a text string, and with 'button' and 'save' options no VALUE will be sent at all.

exclude MOVE

include MOVE

exclude all

include all

These commands change the set of moves that the engine should consider in the root node of its search, by removing or adding the mentioned MOVE from this set. After reaching a new position, (e.g. through a usermove, undo, new or setboard command), or after receiving "include all", this set should always be reset to all legal moves from that position. If the set of moves changes during a search, the engine could start a new search from scratch, or it can try to be smart, and continue the current search with the new set of moves (e.g. after exclusion of a move that has not been searched yet in the current iteration). After "exclude all", the engine would have no legal moves in the root, which logically should make it behave as if it is (stale)mated, but it is allowed to defer any effects of this command on a search in progress to when the set gets non-empty again through addition of a move. These commands will only be sent to engines that have requested such through the exclude feature.

setscore SCORE DEPTH

This command instructs the engine to treat future search requests on the current position (also when it is encountered inside a larger search tree) upto the given DEPTH as if these result is SCORE centi-Pawn in favor of the side that has the move in this position. It is entirely up to the engine to decide when the effect of this option should expire. (E.g. it could last upto the next "new" or "quit" command, or even into future sessions until the user explicitly clears it through an engine-defined option.) This command will only be sent to engines that have requested it through the setscore feature.

lift SQUARE

put SQUARE

hover SQUARE

These commands are only important for variants the GUI does not know the rules of, and keep the engine

aware of how the user is manipulating pieces in the GUI, so that it can supply relevant rule information. The "lift" command is sent by the GUI when the user 'picks up' (or selects) a piece from the mentioned SQUARE, so that the engine can reply with a "highlight" command to mark the squares where that piece can move to. The "put" command similarly indicates where the user releases that piece; as the GUI clears the highlights on that event by itself, usually no engine response would be required. The "hover" command is sent whenever the mouse pointer enters a square that is currently marked in red, (reserved for captures) so that the engine can (optionally) reply with a "highlight" command to mark victims of non-standard capture (such as e.p. capture in Chess, or jump capture in Checkers) when the user would move the currently selected piece there. These commands will only be sent to engines that have requested such through the highlight feature.

Bughouse commands:

xboard now supports bughouse engines when in Zippy mode. See zippy.README for information on Zippy mode and how to turn on the bughouse support. The bughouse move format is given above. xboard sends the following additional commands to the engine when in bughouse mode. Commands to inform your engine of the partner's game state may be added in the future.

partner <player>

<player> is now your partner for future games. Example:

```
partner mann
```

partner

Meaning: You no longer have a partner.

ptell <text>

Your partner told you <text>, either with a ptell or an ordinary tell.

holding [<white>] [<black>]

White currently holds <white>; black currently holds <black>. Example:

```
holding [PPPRQ] []
```

holding [<white>] [<black>] <color><piece>

White currently holds <white>; black currently holds <black>, after <color> acquired <piece>. Example:

```
holding [PPPRQ] [R] BR
```

9. COMMANDS FROM THE ENGINE TO XBOARD

In general, an engine should not send any output to xboard that is not described in this document. As the protocol is extended, newer versions of xboard may recognize additional strings as commands that were previously not assigned a meaning.

feature FEATURE1=VALUE1 FEATURE2=VALUE2 ...

Beginning with version 2, the protocol includes the "feature" command, which lets your engine control certain optional protocol features. Feature settings are written as FEATURE=VALUE, where FEATURE is a name from the list below and VALUE is the value to be assigned. Features can take string, integer, or boolean values; the type of value is listed for each feature. String values are written in double quotes (for example, `feature myname="Miracle Chess 0.9"`), integers are written in decimal, and boolean

values are written as 0 for false, 1 for true. Any number of features can be set in one feature command, or multiple feature commands can be given.

Your engine should send one or more feature commands immediately after receiving the "protover" command, since xboard needs to know the values of some features before sending further commands to the engine. Because engines that predate protocol version 2 do not send "feature", xboard uses a timeout mechanism: when it first starts your engine, it sends "xboard" and "protover N", then listens for feature commands for two seconds before sending any other commands. To end this timeout and avoid the wait, set the feature "done=1" at the end of your last feature command. To increase the timeout, if needed, set the feature "done=0" before your first feature command and "done=1" at the end. If needed, it is okay for your engine to set done=0 soon as it starts, even before it receives the xboard and protover commands. This can be useful if your engine takes a long time to initialize itself. It should be harmless even if you are talking to a (version 1) user interface that does not understand the "feature" command, since such interfaces generally ignore commands from the engine that they do not understand.

The feature command is designed to let the protocol change without breaking engines that were written for older protocol versions. When a new feature is added to the protocol, its default value is always chosen to be compatible with older versions of the protocol that did not have the feature. Any feature that your engine does not set in a "feature" command retains its default value, so as the protocol changes, you do not have to change your engine to keep up with it unless you want to take advantage of a new feature. Because some features are improvements to the protocol, while others are meant to cater to engines that do not implement all the protocol features, the recommended setting for a feature is not always the same as the default setting. The listing below gives both default and recommended settings for most features.

You may want to code your engine so as to be able to work with multiple versions of the engine protocol. Protocol version 1 does not send the protover command and does not implement the feature command; if you send a feature command in protocol version 1, it will have no effect and there will be no response. In protocol version 2 or later, each feature F that you set generates the response "accepted F" if the

feature is implemented, or "rejected F" if it is not. Thus an engine author can request any feature without having to keep track of which protocol version it was introduced in; you need only check whether the feature is accepted or rejected. This mechanism also makes it possible for a user interface author to implement a subset of a protocol version by rejecting some features that are defined in that version; however, you should realize that engine authors are likely to code for xboard and may not be prepared to have a feature that they depend on be rejected. If the GUI rejects an option feature because of the syntax of the value, it should print the value string with the "rejected" command, e.g. "rejected option nonsense" in response to receiving feature option="nonsense".

Here are the features that are currently defined.

ping (boolean, default 0, recommended 1)

If ping=1, xboard may use the protocol's new "ping" command; if ping=0, xboard will not use the command.

setboard (boolean, default 0, recommended 1)

If setboard=1, xboard will use the protocol's new "setboard" command to set up positions; if setboard=0, it will use the older "edit" command.

playother (boolean, default 0, recommended 1)

If playother=1, xboard will use the protocol's new "playother" command when appropriate; if playother=0, it will not use the command.

san (boolean, default 0)

If san=1, xboard will send moves to the engine in standard algebraic notation (SAN); for example, Nf3. If san=0, xboard will send moves in coordinate notation; for example, g1f3. See MOVE in section 8 above for more details of both kinds of notation.

usermove (boolean, default 0)

If usermove=1, xboard will send moves to the engine with the command "usermove MOVE"; if usermove=0, xboard will send just the move, with no command name.

time (boolean, default 1, recommended 1)

If time=1, xboard will send the "time" and "otim" commands to update the engine's clocks; if time=0, it will not.

draw (boolean, default 1, recommended 1)

If draw=1, xboard will send the "draw" command if the engine's opponent offers a draw; if draw=0, xboard will not inform the engine about draw offers. Note that if draw=1, you may receive a draw offer while you are on move; if this will cause you to move immediately, you should set draw=0.

sigint (boolean, default 1)

If sigint=1, xboard may send SIGINT (the interrupt signal) to the engine as section 7 above; if sigint=0, it will not.

sigterm (boolean, default 1)

If sigterm=1, xboard may send SIGTERM (the termination signal) to the engine as section 7 above;

if sigterm=0, it will not.

reuse (boolean, default 1, recommended 1)

If reuse=1, xboard may reuse your engine for multiple games. If reuse=0 (or if the user has set the -xreuse option on xboard's command line), xboard will kill the engine process after every game and start a fresh process for the next game.

analyze (boolean, default 1, recommended 1)

If analyze=0, xboard will not try to use the "analyze" command; it will pop up an error message if the user asks for analysis mode. If analyze=1, xboard will try to use the command if the user asks for analysis mode.

myname (string, default determined from engine filename)

This feature lets you set the name that xboard will use for your engine in window banners, in the PGN tags of saved game files, and when sending the "name" command to another engine.

variants (string, see text below)

This feature indicates which chess variants your engine accepts. It should be a comma-separated list of variant names. See the table under the "variant" command in section 8 above. If you do not set this feature, xboard will assume by default that your engine supports all variants. (However, the -zippyVariants command-line option still limits which variants will be accepted in Zippy mode.) It is recommended that you set this feature to the correct value for your engine (just "normal" in most cases) rather than leaving the default in place, so that the user will get an appropriate error message if he tries to play a variant that your engine does not support. As of XBoard 4.8 a variant

name not known to the GUI will be considered an engine-defined variant. The user will be given the opportunity to select such variants, but when this happens, the engine should define its meaning in detail with the aid of a "setup" command defined below, in order to avoid an error.
If your engine can play variants on a deviating board size, like capablanca on an 8x8 board, or capablanca crazyhouse, it can list them amongst the variants with a prefix specifying board size plus holdings size, like 8x8+0_capablanca or 10x8+7_capablanca. If it is capable of playing any variant with an arbitrary board size, it should list "boardsize" as one of the variants. If there is a maximum to the board size, this can be prefixed, e.g. "12x10+0_boardsize".

colors (boolean, default 1, recommended 0)

If colors=1, xboard uses the obsolete "white" and "black" commands in a stylized way that works with most older chess engines that require the commands. See the "Idioms" section below for details. If colors=0, xboard does not use the "white" and "black" commands at all.

ics (boolean, default 0)

If ics=1, xboard will use the protocol's new "ics" command to inform the engine of whether or not it is playing on a chess server; if ics=0, it will not.

name (boolean, see text below)

If name=1, xboard will use the protocol's "name" command to inform the engine of the opponent's name; if name=0, it will not. By default, name=1 if the engine is playing on a chess server; name=0 if not.

pause (boolean, default 0)

If pause=1, xboard may use the protocol's new "pause" command; if pause=0, xboard assumes that the engine does not support this command.

nps (boolean, default ?)

If nps=1, it means the engine supports the nps command. If nps=0, it means the engine does not support it, and WinBoard should refrain from sending it. Default is that WinBoard sends it, in an attempt to try out if the engine understand it. The engine should properly respond with "Error (unkown command): nps" if it does not implement it, (as any protocol version pre-scribes), or WinBoard might assume that the engine did understand the command. In that case the use of different time standards that ensues could lead to time forfeits for the engine.

debug (boolean, default 0)

If debug=1, it means the engine wants to send debug output prefixed by '#', which WinBoard should ignore, except for including it in the winboard.debug file. As this feature is added to protocol 2 ony late, so that not all protocol-2 supporting versions of WinBoard might implement it, it is important that engines check if WinBoard accepts the feature. If the feature is rejected, engines must refrain from sending the debug output, or do so at their own risk.

memory (boolean, default 0)

If memory=1, the size of the total amount of memory available for the memory-consuming tables of the engine (e.g. hash, EGTB cache) will be set by the GUI through the "memory" command.

smp (boolean, default 0)

If smp=1, the GUI will send the "cores" command to the engine to inform it how many CPU cores it

can use. Note that sending smp=1 does not imply the engine can use more than one CPU; just that it wants to receive the "cores" command.

egt (string, see text below)

This feature indicates which end-game table formats the engine supports. It should be a comma-separated list of format names. See under the "egtpath" command in section 8 above. If you do not set this feature, xboard will assume the engine does not support end-game tables, and will not send any "egtpath" commands to inform the engine about their whereabouts.

option (string, see text below)

This feature is used by the engine to define an option command to appear in a GUI menu, so that the user can change the corresponding setting of the engine through the GUI interactively. The string describes the option by defining a name, type, current value and (sometimes) the acceptable value range. Unlike other features, option features are accumulated by the GUI, and the GUI must be able to add a new option to the list at any time, even after having received feature done=1.
There are ten different options types, each requiring a slighly different syntax of the defining string:
feature option="NAME -button"
feature option="NAME -save"
feature option="NAME -reset"
feature option="NAME -check VALUE"
feature option="NAME -string VALUE"
feature option="NAME -spin VALUE MIN MAX"
feature option="NAME -combo CHOICE1 /// CHOICE2 ..."
feature option="NAME -slider VALUE MIN MAX"
feature option="NAME -file VALUE"
feature option="NAME -path VALUE"

NAME is an arbitrary alphanumeric string which can contain spaces; the other words in capitals would be replaced by the current (default) setting of the option, (a character string for -string options, a decimal number for -spin and -check options, were the latter uses 1=checked, 0=unchecked), the minimum or maximum value of numeric (-spin) options, or arbitrary text labels (for -combo option). In the latter case, the current value will be preceded by an asterisk. The -file and -path options are similar to -string, but can be used to inform the GUI that the text represents a file name or folder name respectively, so the GUI dialog could add the appropriate browse button to the text-edit field. Similarly, a -slider option is like a -spin, but the GUI might make a different graphical representation for it. A -save option is like a -button, and defines an immediate command to be sent by the engine. With -save the GUI will make sure all current option settings are flushed to the engine before it sends this command. A -reset option is like a -button, but use of it purges the list of options before sending the corresponding option command to the engine. This enables the engine to completely redefine its options or their current settings, by sending a new set of option feature commands to the GUI, terminated by feature done=1. (The effect of sending an option feature for an option with the same name as was defined before, without first receiving a -reset option command, is undefined.)

exclude (boolean, default 0)

If exclude=1 the GUI can send "exclude" and "include" commands to control which moves from the root position should be searched.

setscore (boolean, default 0)

If setscore=1 the GUI can send "setscore" commands to define the score of the current position.

highlight (boolean, default 0)

If highlight=1 the GUI will send "lift", "put" and "hover" commands to the engine, to keep the latter aware of the user's piece manipulation before the move entry is completed, so it can respond with the proper "highlight" and "click" commands.

done (integer, no default)

If you set done=1 during the initial two-second timeout after xboard sends you the "xboard" command, the timeout will end and xboard will not look for any more feature commands before starting normal operation. If you set done=0, the initial timeout is increased to one hour; in this case, you must set done=1 before xboard will enter normal operation.

Illegal move: MOVE

Illegal move (REASON): MOVE

If your engine receives a MOVE command that is recognizably a move but is not legal in the current position, your engine must print an error message in one of the above formats so that xboard can pass the error on to the user and retract the move. The (REASON) is entirely optional. Examples:

```
Illegal move: e2e4
Illegal move (in check): Nf3
Illegal move (moving into check): e1g1
```

Generally, xboard will never send an ambiguous move, so it does not matter whether you respond to such a move with an Illegal move message or an Error message.

Error (ERRORTYPE): COMMAND

If your engine receives a command it does not understand or does not implement, it should print an error

message in the above format so that xboard can parse it. Examples:

```
Error (ambiguous move): Nf3
Error (unknown command): analyze
Error (command not legal now): undo
Error (too many parameters): level 1 2 3 4 5 6 7
```

move MOVE

Your engine is making the move MOVE. Do not echo moves from xboard with this command; send only new moves made by the engine.

For the actual move text from your chess engine (in place of MOVE above), your move should be either

- in coordinate notation (e.g., e2e4, e7e8q) with castling indicated by the King's two-square move (e.g., e1g1), or
- in Standard Algebraic Notation (SAN) as defined in the Portable Game Notation standard (e.g, e4, Nf3, O-O, cxb5, Nxe4, e8=Q), with the extension piece@square (e.g., P@f7) to handle piece placement in bughouse and crazyhouse.

xboard itself also accepts some variants of SAN, but for compatibility with non-xboard interfaces, it is best not to rely on this behavior.

Warning: Even though all versions of this protocol specification have indicated that xboard accepts SAN moves, some non-xboard interfaces are known to accept only coordinate notation. See the Idioms section for more information on the known limitations of some non-xboard interfaces. It should be safe to send SAN moves if you receive a "protover 2" (or later) command from the interface, but otherwise it is best to stick to coordinate notation for maximum compatibility. An even more conservative approach would be for your engine to send SAN to the interface only if you have set feature san=1 (which causes the interface to send SAN to you) and have received "accepted san" in reply.

For a multi-leg move, each leg will have to be sent in a separate "move" command, a comma at the end of all non-final legs indicating there is more to follow.

RESULT {COMMENT}

When your engine detects that the game has ended by rule, your engine must output a line of the form "RESULT {comment}" (without the quotes), where RESULT is a PGN result code (1-0, 0-1, or 1/2-1/2), and comment is the reason. Here "by rule" means that the game is definitely over because of what happened on the board. In normal chess, this includes checkmate, stalemate, triple repetition, the 50 move rule, or insufficient material; it does not include loss on time or the like. Examples:

```
0-1 {Black mates}
1-0 {White mates}
1/2-1/2 {Draw by repetition}
1/2-1/2 {Stalemate}
```

xboard relays the result to the user, the ICS, the other engine in Two Machines mode, and the PGN save file as required. Note that "definitey over" above means that sending this command will be taken by WinBoard as an unconditional refusal of the engine to play on, which might cause you to forfeit if the game was in fact not over. This command should thus not be used to offer draws, accept draws, or make draw-by-rule claims that are not yet valid in the current position (but will be after you move). For offering and claiming such draws, "offer draw" should be used.

Note that (in accordance with FIDE rules) only KK, KNK, KBK and KBKB with all bishops on the same color can be claimed as draws on the basis of insufficient mating material. The end-games KNNK, KBKN, KNKN and KBKB with unlike bishops do have mate positions, and cannot be claimed. Complex draws based on locked Pawn chains will not be recognized as draws by most interfaces, so do not claim in such positions, but just offer a draw or play on.

Note to GUI programmers: RESULT commands that the engine sends immediately after its move might be detected by the GUI only after the opponent has moved, because of communication and scheduling delays, no matter how fast the engine sent it. Any judgement of the validity of RESULT claims based on te "current" board position will have to account for this uncertainty.

resign

If your engine wants to resign, it can send the command "resign". Alternatively, it can use the "RESULT {comment}" command if the string "resign" is included in the comment; for example "0-1 {White resigns}". xboard relays the resignation to the user, the ICS, the other engine in Two Machines mode, and the PGN save file as required. Note that many interfaces work more smoothly if you resign *before* you move.

offer draw

If your engine wants to offer a draw by agreement (as opposed to claiming a draw by rule), it can send the command "offer draw". xboard relays the offer to the user, the ICS, the other engine in Two Machines mode, and the PGN save file as required. In Machine White, Machine Black, or Two Machines mode, the offer is considered valid until your engine has made two more moves. This command must also be used to accept a draw offer. Do not use the 1/2-1/2 command for that, as the offer might be no longer valid, in which case a refusal to play on implied by the RESULT command might make you forfeit the game. "offer draw" should also be used to claim 50-move and 3-fold-repetition draws that will occur *after* your move, by sending it *before* making the move. WinBoard will grant draw offers without the opponent having any say in it in situations where draws can be claimed. Only if the draw cannot be claimed, the offer will be passed to your opponent after you make your next move, just before WinBoard relays this move to the opponent.

tellopponent MESSAGE

This command lets the engine give a message to its opponent, independent of whether the opponent is a user on the local machine or a remote ICS user (Zippy mode). MESSAGE consists of any characters, including whitespace, to the end of the line. When the engine is playing against a user on the local machine, xboard pops up an information dialog containing the message. When the engine is playing against an opponent on the ICS (Zippy mode), xboard sends "say MESSAGE\n" to the ICS.

tellothers MESSAGE

This command lets the engine give a message to people watching the game other than the engine's opponent. MESSAGE consists of any characters, including whitespace, to the end of the line. When the engine is playing against a user on the local machine, this command does nothing. When the engine is playing against an opponent on the ICS (Zippy mode), xboard sends "whisper MESSAGE\n" to the ICS.

tellall MESSAGE

This command lets the engine give a message to its opponent and other people watching the game, independent of whether the opponent is a user on the local machine or a remote ICS user (Zippy mode). MESSAGE consists of any characters, including whitespace, to the end of the line. When the engine is playing against a user on the local machine, xboard pops up an information dialog containing the message. When the engine is playing against an opponent on the ICS (Zippy mode), xboard sends "kibitz MESSAGE\n" to the ICS.

telluser MESSAGE

xboard pops up an information dialog containing the message. MESSAGE consists of any characters, including whitespace, to the end of the line.

tellusererror MESSAGE

xboard pops up an error dialog containing the message. MESSAGE consists of any characters, including whitespace, to the end of the line.

askuser REPTAG MESSAGE

Here REPTAG is a string containing no whitespace, and MESSAGE consists of any characters, including whitespace, to the end of the line. xboard pops up a question dialog that says MESSAGE and has a typein box. If the user types in "bar", xboard sends "REPTAG bar" to the engine. The user can cancel the dialog and send nothing.

tellics MESSAGE

In Zippy mode, xboard sends "MESSAGE\n" to ICS. MESSAGE consists of any characters, including whitespace, to the end of the line.

tellicsnoalias MESSAGE

In Zippy mode, xboard sends "xMESSAGE\n" to ICS, where "x" is a character that prevents the ICS from expanding command aliases, if xboard knows of such a character. (On chessclub.com and chess.net, "/" is used; on freechess.org, "$" is used.) MESSAGE consists of any characters, including whitespace, to the end of the line.

COMMENT

The engine can send any string of printable characters, terminated by a newline, for inclusion in the winboard.debug file, provided the line starts with a '#' character. If the engine has set feature debug=1, it is guaranteed that WinBoard (and any future version of it) will completely ignore these lines in any other respect.

Commands to configure the GUI for arbitrary Chess variants:

setup FEN

setup (PIECETOCHAR) FEN

setup (PIECETOCHAR) WxH+S_PARENTVARIANT FEN

The engine can optionally send a setup command to the GUI in reply to the variant command. In the simplest form this sends the FEN of the initial position. This can be used to implement engines for non-standard variants that only differ from standard variants through the initial position. (E.g. many of the 'wild' boards you can play on an ICS.) Whether the GUI should obey or ignore this command depends on the situation. Normally it would ignore it in variants where it knows the standard initial position and legality testing is on, or when the user specified an initial position. In other cases it will use the FEN sent by the first engine for setting up the initial position, as if it was an externally supplied position. Such a position will always be sent to a second engine that might be involved, and any setup commands received from the latter will always be ignored. (This to allow for shuffle games, where the two engines might pick different setups.) When no initial position is known, such as for 'catch-all' variants like fairy, or whenever the board width is overruled to a non-standard value, the FEN will be used as default initial position even when legality testing is on.

Optionally the meaning of the piece ID letters in the FEN can be defined between parentheses; this will be interpreted as if it was the value of a -pieceToCharTable command-line option, mapping letters to GUI piece types. Also optionally behind that, the setup command can specify board width W, board height H and holdings size S, as well as a 'parent variant'. This is typically done in response to a variant command with a non-standard name, about which the GUI is not supposed to know anything at all. The engine can then specify board size, participating pieces, initial setup, and other rule details (inherited from the parent

variant), saving the user the trouble to configure the GUI for this non-standard variant. Example:

```
setup (PN.RQKpn.rqk) 6x6+0_fairy rnqknr/pppppp/6/6/PPPPPP/RNQKNR w - - 0 1
```

could be used by an engine for Los-Alamos Chess in response to 'variant losalamos', and would automatically switch the GUI to this variant as soon as the user selected it from the GUI menu. The PIECETOCHAR element would ensure a Bishop would not be accepted as promotion choice.

piece ID PIECEDESC

The engine can send one or more piece commands in response to a variant command, in order to specify that the piece indicated by ID moves in a non-standard way in this variant. (This to enable the GUI to reliably perform mate detection, and produce good SAN.) Like in FEN the ID is a case-sensitive letter, specifying the color. When it is a capital suffixed by &, the description is valid for both colors. PIECEDESC describes the moves in 'Betza notation', basically a concatenation of one-letter (upper-case) codes for all of its moves. These codes can be prefixed with lower-case 'modifiers' to indicate directional sub-sets (combinations of fblrvs, if the piece is not totally symmetric), move modality (non-capture, capture, e.p. capture; mce), and whether the move can jump directly to its destination, or can be blocked (n). Moves only valid for a virgin piece are prefixed by 'i'. An optional numeric suffix on the move indicates the maximum number of times the move can be repeated in the same direction, to indicate sliders / riders (with the convention 0 = infinite).

highlight COLORFEN

Through this command the engine can apply markers to the board squares, of the same type as the GUI uses for indicating where the user could put down a piece he grabs. The COLORFEN is a construct similar to the board part of a FEN, in which the letters indicate colors rather than piece types. Eight colors are available, through the single-letter codes: RYGCBMWD, for red, yellow, green, cyan, blue, magenta, white, black ('dark'), respectively. For example, "highlight 8/8/8/8/4y3/4yr2/8/8" would mark e3 and e4

yellow, and f3 red. Some colors have special meaning to the GUI:

color	used for	effect
red	capture	hovering over the square makes the GUI send a "hover" command
magenta	promotion	moving to the square will be treated by the GUI as a promotion
cyan	multi-move	moving to the square will not complete the move entry
green	victims	no real effect, but used by convention to indicate capture victims for the latest "hover"

The GUI will use the markers for legality checking, and will consider moves to squares left non-marked in a highlight command as illegal even when legality checking is off. This way the GUI can be made aware of the rules of unknown variants, popping up promotion dialogs where it would otherwise not, and knowing where to wait for more input on multi-leg moves. When it would be necessary to mark squares where no legal moves go to (e.g. to indicate side effects), the corresponding lower-case character can be used for the color. For indicating a legal destination square without visibly marking it, T (transparent) can be used.

click SQUARE

The GUI will treat this command as if the user had clicked the mentioned SQUARE. This can be used to implement one-click moving in variants the GUI does not know the rules of (having the engine send the clicks needed to complete the move). It can also be used to implement side effects of the move the GUI would not know about (e.g. moving the Rook in a non-standard castling).

10. THINKING OUTPUT

If the user asks your engine to "show thinking", xboard sends your engine the "post" command. It sends "nopost" to turn thinking off. In post mode, your engine sends output lines to show the progress of its thinking. The engine can send as many or few of these lines as it wants to, whenever it wants to. Typically they would be sent when the PV (principal variation) changes or the depth changes. The thinking output should be in the following format:

```
ply score time nodes pv
```

Where:

ply	Integer giving current search depth.
score	Integer giving current evaluation in centipawns.
time	Current search time in centiseconds (ex:1028 = 10.28 seconds).
nodes	Nodes searched.
***selective depth**	Maximium length of any branch in the current search.
***speed**	Nodes per second in last measured time interval.
*	Reserved for future extensions.
***tbhits**	Number of tablebase probes made in the current search.

pv	Freeform text giving current "best" line. You can continue the pv onto another line if you start each continuation line with at least four space characters.

The items marked with * are optional. If any of these items is present, the **pv** field must be preceeded directly by a tab character; if no tab character preceeds the first non-integer token, the **pv** field will start at the first non-blank character after **nodes**. Otherwise it will start after the last tab that is not behind any non-integer token. Of all integers between **nodes** and **pv** the last one is intepreted as **tbhits**. Of any remaining ones the first is interpreted as **selective depth**, and a second as **speed**. More infos could be added to this in the future. Note that older interfaces might consider the optional infos to be part of the **pv** field, and display them exactly as sent. It is therefore encouraged that engines use tabs or spaces to format this optional info so that it will display nicely in (not too wide) columns.

A question mark as the last character in the **pv** field should be used to indicate the reported score is from a fail low, and thus represents an upper bound only. Similarly, an exclamation point should be used to indicate a fail high / lower bound.

Mate scores should be indicated as 100000 + N for "mate in N moves", and -100000 - N for "mated in N moves".

Example:

```
  9 156 1084 48000 Nf3 Nc6 Nc3 Nf6
```

Meaning:

```
9 ply, score=1.56, time = 10.84 seconds, nodes=48000, PV = "Nf3 Nc6 Nc3 Nf6"
```

Longer example from actual Crafty output:

```
  4    109      14   1435  1. e4 d5 2. Qf3 dxe4 3. Qxe4 Nc6
```

```
 4    116      23   2252  1. Nf3 Nc6 2. e4 e6
 4    116      27   2589  1. Nf3 Nc6 2. e4 e6
 5    141      44   4539  1. Nf3 Nc6 2. O-O e5 3. e4
 5    141      54   5568  1. Nf3 Nc6 2. O-O e5 3. e4
```

You can use the PV to show other things; for instance, while in book, Crafty shows the observed frequency of different reply moves in its book. In situations like this where your engine is not really searching, start the PV with a '(' character:

```
 0      0       0      0  (e4 64%, d4 24%)
```

GNU Chess output is very slightly different. The ply number is followed by an extra nonblank character, and the time is in seconds, not hundredths of seconds. For compatibility, xboard accepts the extra character and takes it as a flag indicating the different time units. Example:

```
2.     14    0       38   d1d2  e8e7
3+     78    0       65   d1d2  e8e7  d2d3
3&     14    0       89   d1d2  e8e7  d2d3
3&     76    0      191   d1e2  e8e7  e2e3
3.     76    0      215   d1e2  e8e7  e2e3
4&     15    0      366   d1e2  e8e7  e2e3  e7e6
4.     15    0      515   d1e2  e8e7  e2e3  e7e6
5+     74    0      702   d1e2  f7f5  e2e3  e8e7  e3f4
5&     71    0     1085   d1e2  e8e7  e2e3  e7e6  e3f4
5.     71    0     1669   d1e2  e8e7  e2e3  e7e6  e3f4
6&     48    0     3035   d1e2  e8e7  e2e3  e7e6  e3e4  f7f5  e4d4
6.     48    0     3720   d1e2  e8e7  e2e3  e7e6  e3e4  f7f5  e4d4
7&     48    0     6381   d1e2  e8e7  e2e3  e7e6  e3e4  f7f5  e4d4
7.     48    0    10056   d1e2  e8e7  e2e3  e7e6  e3e4  f7f5  e4d4
8&     66    1    20536   d1e2  e8e7  e2e3  e7e6  e3d4  g7g5  a2a4  f7f5
8.     66    1    24387   d1e2  e8e7  e2e3  e7e6  e3d4  g7g5  a2a4  f7f5
9&     62    2    38886   d1e2  e8e7  e2e3  e7e6  e3d4  h7h5  a2a4  h5h4
                         d4e4
9.     62    4    72578   d1e2  e8e7  e2e3  e7e6  e3d4  h7h5  a2a4  h5h4
```

```
                          d4e4
10&     34    7   135944   d1e2  e8e7  e2e3  e7e6  e3d4  h7h5  c2c4  h5h4
                          d4e4  f7f5  e4f4
10.     34    9   173474   d1e2  e8e7  e2e3  e7e6  e3d4  h7h5  c2c4  h5h4
                          d4e4  f7f5  e4f4
```

If your engine is pondering (thinking on its opponent's time) in post mode, it can show its thinking then too. In this case your engine may omit the hint move (the move it is assuming its opponent will make) from the thinking lines *if and only if* it sends xboard the move in the usual "Hint: xxx" format before sending the first line.

11. TIME CONTROL

xboard supports three styles of time control: conventional chess clocks, the ICS-style incremental clock, and an exact number of seconds per move.

In conventional clock mode, every time control period is the same. That is, if the time control is 40 moves in 5 minutes, then after each side has made 40 moves, they each get an additional 5 minutes, and so on, ad infinitum. At some future time it would be nice to support a series of distinct time controls. This is very low on my personal priority list, but code donations to the xboard project are accepted, so feel free to take a swing at it. I suggest you talk to me first, though.

The command to set a conventional time control looks like this:

```
level 40 5 0
level 40 0:30 0
```

The 40 means that there are 40 moves per time control. The 5 means there are 5 minutes in the control. In the second example, the 0:30 means there are 30 seconds. The final 0 means that we are in conventional clock mode.

Note that the time parameter in this command is not a pure numeric argument, but in general is a character string, in order to pass the number of seconds. Engines are encouraged to ignore any unexpected characters at the end of this string, i.e. following the MIN or MIN:SEC specification. Future protocol versions might (under control of an appropriate feature) append such extra characters to this argument, in order to inform the engine in advance of the time control it can expect after the current session completes. E.g. "level 40 25+5 0" could mean that the engine has to play 40 moves in 25 minutes, but should expect to get only 5 minutes for the entire remainder of the game after that, rather than another 25 minutes for the next 40 moves. When the time comes, (i.e. after the 40 moves), it will be informed of the time-control change by receiving a new "level 0 5 0" command, but engines with advanced time management might want to plan for this in advance.

The command to set an incremental time control looks like this:

```
level 0 2 12
```

Here the 0 means "play the whole game in this time control period", the 2 means "base=2 minutes", and the 12 means "inc=12 seconds". As in conventional clock mode, the second argument to level can be in minutes and seconds.

At the start of the game, each player's clock is set to base minutes. Immediately after a player makes a move, inc seconds are added to his clock. A player's clock counts down while it is his turn. Your flag can be called whenever your clock is zero or negative. (Your clock can go negative and then become positive again because of the increment.)

The number of moves given in the level command (when non-zero) should be taken as the number of moves still to do before the specified time will be added to the clock, if the "level" command is received after some moves have already been played. The time given should be interpreted as the time left on its clock (including any time left over from the previous sessions), and not necessarily the time that will be added to the clock after the specified number of moves has been played. This is only relevant in WinBoard 4.3.xx, which might send the engine "level" commands during a game, just before the engine has to start thinking about the first move of a

new time-control session. Example: if at the start of the game "level 40 60 0" was given (40 moves per hour), and the engine receives "level 20 22 0" just before move 41, it should understand that it should do the next 20 moves in 22 minutes (pehaps because the secondary session was 20 moves per 15 minutes, and it had 7 minutes left on its clock after the first 40 moves).

A special rule on some ICS implementations: if you ask for a game with base=0, the clocks really start at 10 seconds instead of 0. xboard itself does not know about this rule, so it passes the 0 on to the engine instead of changing it to 0:10.

ICS also has time odds games. With time odds, each player has his own (base, inc) pair, but otherwise things work the same as in normal games. The Zippy xboard accepts time odds games but ignores the fact that the opponent's parameters are different; this is perhaps not quite the right thing to do, but gnuchess doesn't understand time odds. Time odds games are always unrated.

The command to set an exact number of seconds per move looks like this:

```
st 30
```

This means that each move must be made in at most 30 seconds. Time not used on one move does not accumulate for use on later moves.

12. ANALYZE MODE

xboard supports analyzing fresh games, edited positions, and games from files. However, all of these look the same from the chess engine's perspective. Basically, the engine just has to respond to the "analyze" command. Beginning in protocol version 2, if your engine does not support analyze mode, it should use the feature command to set analyze=0. The older method of printing the error message "Error (unknown command): analyze" in response to the "analyze" command will also work, however.

To enter analyze mode, xboard sends the command sequence "post", "analyze". Analyze mode in your engine should be similar to force mode, except that your engine thinks about what move it would make next if it were on move. Your engine should accept the following commands while in analyze mode:

- Any legal move, as in force mode
- **undo** Back up one move and analyze previous position.
- **new** Reset position to start of game but stay in analyze mode.
- **setboard** if you have set feature setboard=1; otherwise **edit**. Exiting edit mode returns to analyze mode.
- **exit** Leave analyze mode.
- **.** Send a search status update (optional); see below.
- **bk** Show book moves from this position, if any; see above.
- **hint** Show the predicted move from this position, if any; see above.

If the user selects "Periodic Updates", xboard will send the string ".\n" to the chess engine periodically during analyze mode, unless the last PV received began with a '(' character.

The chess engine should respond to ".\n" with a line like this:

```
stat01: time nodes ply mvleft mvtot mvname
```

Where:

time	Elapsed search time in centiseconds (ie: 567 = 5.67 seconds).
nodes	Nodes searched so far.

ply	Search depth so far.
mvleft	Number of moves left to consider at this depth.
mvtot	Total number of moves to consider.
mvname	Move currently being considered (SAN or coordinate notation). Optional; added in protocol version 2.

Examples:

```
stat01: 1234 30000 7 5 30
stat01: 1234 30000 7 5 30 Nf3
```

Meaning:

After 12.34 seconds, I've searched 7 ply/30000 nodes, there are a total of 30 legal moves, and I have 5 more moves to search before going to depth 8. In the second example, of the 30 legal moves, the one I am currently searching is Nf3.

Implementation of the "." command is optional. If the engine does not respond to the "." command with a "stat01..." line, xboard will stop sending "." commands. If the engine does not implement this command, the analysis window will use a shortened format to display the engine info.

To give the user some extra information, the chess engine can output the strings "++\n" and "--\n", to indicate that the current search is failing high or low, respectively. You don't have to send anything else to say "Okay, I'm not failing high/low anymore." xboard will figure this out itself.

13. IDIOMS AND BACKWARD COMPATIBILITY FEATURES

Some engines have variant interpretations of the force/go/white/black, time/otim, and hard/easy command sets. In order to accommodate these older engines, xboard uses these commands only according to the stylized patterns ("idioms") given in this section. The obsolete white and black commands have historically been particularly troublesome, and it is recommended that new engines set the feature colors=0 and/or ignore the commands.

time N

otim N

MOVE

Sent when the opponent makes a move and the engine is already playing the opposite color.

white

go

Sent when the engine is in force mode or playing Black but should switch to playing White. This sequence is sent only when White is already on move. If you set the feature colors=0, "white" is not sent.

black

go

Sent when the engine is in force mode or playing White but should switch to playing Black. This sequence is sent only when Black is already on move. If you set the feature colors=0, "black" is not sent.

white

time N

otim N

black

go

Sent when Black is on move, the engine is in force mode or playing White, and the engine's clock needs to be updated before it starts playing. The initial "white" is a kludge to accommodate GNU Chess 4's variant interpretation of these commands. If you set the feature colors=0, "white" and "black" are not sent.

black

time N

otim N

white

go

Sent when White is on move, the engine is in force mode or playing Black, and the engine's clock needs to be updated before it starts playing. See previous idiom. The initial "black" is a kludge to accommodate GNU Chess 4's variant interpretation of these commands. If you set the feature colors=0, "black" and "white" are not sent.

hard

easy

> Sent in sequence to turn off pondering if xboard is not sure whether it is on. When xboard is sure, it will send "hard" or "easy" alone. xboard does this because "easy" is a toggle in GNU Chess 4 but "hard" is an absolute on.

To support older engines, certain additional commands from the engine to xboard are also recognized. (These are commands by themselves, not values to be placed in the comment field of the PGN result code.) These forms are not recommended for new engines; use the PGN result code commands or the resign command instead.

Command	Interpreted as
White resigns	0-1 {White resigns}
Black resigns	1-0 {Black resigns}
White	1-0 {White mates}
Black	0-1 {Black mates}
Draw	1/2-1/2 {Draw}
computer mates	1-0 {White mates} or 0-1 {Black mates}
opponent mates	1-0 {White mates} or 0-1 {Black mates}
computer resigns	0-1 {White resigns} or 1-0 {Black resigns}

game is a draw	1/2-1/2 {Draw}
checkmate	1-0 {White mates} or 0-1 {Black mates}

Commands in the above table are recognized if they begin a line and arbitrary characters follow, so (for example) "White mates" will be recognized as "White", and "game is a draw by the 50 move rule" will be recognized as "game is a draw". All the commands are case-sensitive.

An alternative move syntax is also recognized:

Command	Interpreted as
NUMBER ... MOVE	move MOVE

Here NUMBER means any string of decimal digits, optionally ending in a period. MOVE is any string containing no whitespace. In this command format, xboard requires the "..." even if your engine is playing White. A command of the form NUMBER MOVE will be ignored. This odd treatment of the commands is needed for compatibility with gnuchessx. The original reasons for it are lost in the mists of time, but I suspect it was originally a bug in the earliest versions of xboard, before I started working on it, which someone "fixed" in the wrong way, by creating a special version of gnuchess (gnuchessx) instead of changing xboard.

Any line that contains the words "offer" and "draw" is recognized as "offer draw".

The "Illegal move" message is recognized even if spelled "illegal move" and even if the colon (":") is omitted. This accommodates GNU Chess 4, which prints messages like "Illegal move (no matching move)e2e4", and old versions of Crafty, which print just "illegal move".

In Zippy mode, for compatibility with older versions of Crafty, xboard passes through to ICS any line that begins "kibitz", "whisper", "tell", or "draw". Do not use this feature in new code. Instead, use the commands "tellall",

"tellothers", "tellopponent", "tellics" (if needed), "1/2-1/2 {COMMENT}", or "offer draw", as appropriate.

If the engine responds to the "sd DEPTH" command with an error message indicating the command is not supported (such as "Illegal move: sd"), xboard sets an internal flag and subsequently uses the command "depth\nDEPTH" instead, for the benefit of GNU Chess 4. Note the newline in the middle of this command! New engines should not rely on this feature.

If the engine responds to the "st TIME" command with an error message indicating the command is not supported (such as "Illegal move: st"), xboard sets an internal flag and subsequently uses the command "level 1 TIME" instead, for the benefit of GNU Chess 4. Note that this is not a standard use of the level command, as TIME seconds are not added after each player makes 1 move; rather, each move is made in at most TIME seconds. New engines should not implement or rely on this feature.

In support of the -firstHost/-secondHost features, which allow a chess engine to be run on another machine using the rsh protocol, xboard recognizes error messages that are likely to come from rsh as fatal errors. The following messages are currently recognized:

- unknown host
- No remote directory
- not found
- No such file
- can't alloc
- Permission denied

ChessBase/Fritz now implements the xboard/winboard protocol and can use WinBoard-compatible engines in its GUI. ChessBase's version of the protocol is generally the same as version 1, except that they have added the commands **fritz**, **reset**, and **ponder**, and the edit subcommands **castle** and **ep**. If you want your engine to

work well with the ChessBase/Fritz GUI, you may need to implement these additional commands, and you should also be aware of the peculiar way that ChessBase uses the protocol. See their web page for documentation.

ChessMaster 8000 also implements version 1 of the xboard/winboard protocol and can use WinBoard-compatible engines. The original release of CM8000 also has one additional restriction: only pure coordinate notation (e.g., e2e4) is accepted in the move command. A patch to correct this should be available from The Learning Company (makers of CM8000) in February 2001.

"Our mission is to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of Free Software users."

The Free Software Foundation is the principal organizational sponsor of the GNU Operating System. **Support GNU and the FSF** by buying manuals and gear, **joining the FSF** as an associate member, or making a **donation**, either directly to the FSF or via Flattr.

back to top

Please send general FSF & GNU inquiries to <gnu@gnu.org>. There are also other ways to contact the FSF. Broken links and other corrections or suggestions can be sent to <bug-xboard@gnu.org>.
Please see the Translations README for information on coordinating and submitting translations of this article.

Updated: $Date: 2016/05/15 17:44:19 $