# Extended Position Description Specification

Revised: 1995.11.26

Technical contact: sje@mv.mv.com (Steven J. Edwards)

# CONTENTS

## 1: Introduction

EPD is "Extended Position Description". It is a standard for describing chess positions along with an extended set of structured attribute values using the ASCII character set. It is intended for data and command interchange among chessplaying programs. It is also intended for the representation of portable opening library repositories and for problem test suites.

EPD is an open standard and is freely available for use by both research and commercial programs without charge. The only requirement for use is that any proposed extensions be coordinated through the technical contact given at the start of this document.

A single EPD record uses one text line of variable length composed of four data fields followed by zero or more operations. A text file composed exclusively of EPD data records should have a file name with the suffix ".epd".

## 2: History

EPD was created in 1993 and is based in part on the earlier FEN standard (Forsyth-Edwards Notation) for representing chess positions. Compared to FEN, EPD has added extensions for use with opening library preparation and also for general data and command interchange among advanced chess programs. EPD was developed by John Stanback and Steven Edwards; its first implementation was in Stanback's commercial chessplaying program Zarkov and its second implementation was in Edwards' research chessplaying program Spector. So many programs have since adopted EPD that no one knows the exact sequence thereafter.

EPD is employed for storing test suites for chessplaying programs and for recording the results of programs running these test suites.

Example test suites are available for researchers via anonymous ftp from the chess.onenet.net site in the pub/chess/Tests directory. The ASCII text file pub/chess/Tests/Manifest gives descriptions of the contents of the various test suite files.

EPD is used to provide a linkage mechanism between chessplaying programs and position database programs to support the automated direction of analysis generation.

# 3: EPD tools and applications

To encourage development of EPD capable applications, a free EPD tool kit is available for program authors working with the ANSI C language.

To further encourage usage of EPD, a number of free applications are also available.

## 3.1: The EPD Kit

Work is currently in progress on developing an EPD Kit. This tool kit is a collection of portable ANSI C source code files that provide routines to create and manipulate EPD data for arbitrarily complex records. It is designed to handle all common EPD related tasks so as to assist chess program developers with EPD implementation. A secondary goal is to ensure that every implementation of EPD processing have the same set of operational semantics.

The EPD Kit will be made freely available to all chess software authors without charge and can be used in both research and commercial applications. As with EPD itself, the only requirement for use is that any proposed extensions be coordinated through the technical contact given at the start of this document.

## 3.2: Argus, the automated tournament referee

Work is currently in progress on developing Argus, an automated tournament referee program for computer chess events. Argus uses IP (Internet Protocol) communications to act as a mediator for multiple pairs of chessplaying programs and to provide an interactive interface for a human tournament supervisor. Argus uses the EPD Kit along with other routines to perform the following tasks:

1) Starting chessplaying programs (IP clients) with proper initialization data;

2) Relaying position/move data (using EPD) from each program to its opponent;

3) Providing all chess clock data as part of the relay process;

4) Record all games using PGN (Portable Game Notation) to assist in the production of the tournament final report;

5) Record all moves and other transmitted data in log files for later

analysis;

6) Detect and report time forfeit conditions;

7) Mediate draw offers and responses between each pair of opponents;

8) Recognize and handle game termination conditions due to draws, resignations, time forfeits, and checkmates;

9) Allow for chessplaying program restart and game resumption as directed by the human supervisor;

10) Allow for a second instance of itself to operate in observer mode to be ready to take over in case of primary machine failure;

11) Support display of games in progress for the benefit of the human supervisor and for the general viewing audience.

In its usual configuration, Argus runs on an IP network that connects it with all of the participating machines. It acts like a Unix style server using TCP/IP; the chessplaying programs connect to Argus as TCP/IP clients. Unlike a typical Unix style server, it runs in the foreground instead of the background when operated by a human supervisor.

One variant mode of operation allows for Argus to be started by the host system and run in the background. This use is intended for events where human supervision is not required. Any operating information usually provided manually may instead be supplied by configuration files.

Another variant mode of operation allows for Argus to mediate communication between a single pair of chessplaying programs using regular (unstructured) bidirectional asynchronous serial communication instead of IP. While less reliable than IP operation, unstructured serial communication can be used on common inexpensive hardware platforms that lack IP support. An example would be to use common PC machines with each chessplaying program running on a separate machine and a third machine running Argus in serial mode. Each of the two machines with chessplaying programs connect to the Argus machine via a null modem cable. Note that the Argus machine needs two free serial ports while each of the chessplaying machines needs only a single free serial port.

The Argus program will be made freely available to all chess software authors without charge and can be used in both research and commercial applications. As with EPD itself, the only requirement for use is that any proposed extensions be coordinated through the technical contact given at the start of this document.

### 3.3: Gastric, an EPD based report generator

Work is in progress on Gastric, an application that reads EPD files and produces statistical reports. The main use of Gastric is to assist in the process of benchmarking chessplaying program performance on EPD test suites. The resulting reports contain summaries of raw performance, identification of solved/missed problems, distribution information for node count, time consumption, and other items. Advanced functions of Gastric may be used to produce comparative analysis of different programs or different versions of the same program. Some work is also planned to allow Gastric output to be used as feedback into self-adjusting chessplaying programs.

The Gastric program will be made freely available to all chess software authors without charge and can be used in both research and commercial applications. As with EPD itself, the only requirement for use is that any proposed extensions be coordinated through the technical contact given at the start of this document.

## 4: The four EPD data fields

Each EPD record contains four data filed that describe the current position. From left to right starting at the beginning of the record, these are the piece placement, the active color, the castling availability, and the en passant target square of a position. These can all fit on a single text line in an easily read format. The length of an EPD position description varies somewhat according to the position and any associated operations. In some cases, the description could be eighty or more characters in length and so may not fit conveniently on some displays. However, most EPD records pass among programs only and so are not usually seen by program users.

Note: due to the likelihood of future expansion of EPD, implementors are encouraged to have their programs handle EPD text lines of up to 4096 characters long including the traditional ASCII NUL character as a terminator. This is an increase from the earlier suggestion of a maximum length of 1024 characters. Depending on the host operating system, the external representation of EPD records will include one or more bytes to indicate the end of a line. These do not count against the length limit as the internal representation of an EPD text record is stripped of end of line bytes and instead is terminated by the traditional ASCII NUL character.

Each of the four EPD data fields are composed only of non-blank printing ASCII characters. Adjacent data fields are separated by a single ASCII space character.

## 4.1: Piece placement data

The first field represents the placement of the pieces on the board.

The board contents are specified starting with the eighth rank and ending with the first rank. For each rank, the squares are specified from file a to file h. White pieces are identified by uppercase SAN (Standard Algebraic Notation) piece letters ("PNBRQK") and black pieces are identified by lowercase SAN piece letters ("pnbrqk"). Empty squares are represented by the digits one through eight; the digit used represents the count of contiguous empty squares along a rank. The contents of all eight squares on each rank must be specified; therefore, the count of piece letters plus the sum of the vacant square counts must always equal eight. The solidus character "/" (forward slash) is used to separate data of adjacent ranks. There is no leading or trailing solidus in the piece placement data; hence there are exactly seven of solidus characters in the placement field.

The piece placement data for the starting array is:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR

## 4.2: Active color

The second field represents the active color. A lower case "w" is used if White is to move; a lower case "b" is used if Black is the active player.

The piece placement and active color data for the starting array is:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w

## 4.3: Castling availability

The third field represents castling availability. This indicates potential future castling that may or may not be possible at the moment due to blocking pieces or enemy attacks. If there is no castling availability for either side, the single character symbol "-" is used. Otherwise, a combination of from one to four characters are present. If White has kingside castling availability, the uppercase letter "K" appears. If White has queenside castling availability, the uppercase letter "Q" appears. If Black has kingside castling availability, the lowercase letter "k" appears. If Black has queenside castling availability, then the lowercase letter "q" appears. Those letters which appear will be ordered first uppercase before lowercase and second kingside before queenside. There is no white space between the letters.

The piece placement, active color, and castling availability data for the starting array is:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq
```

### 4.4: En passant target square

The fourth field is the en passant target square. If there is no en passant target square then the single character symbol "-" appears. If there is an en passant target square then is represented by a lowercase file character (one of "abcdefgh") immediately followed by a rank digit. Obviously, the rank digit will be "3" following a white pawn double advance (Black is the active color) or else be the digit "6" after a black pawn double advance (White being the active color).

An en passant target square is given if and only if the last move was a pawn advance of two squares. Therefore, an en passant target square field may have a square name even if there is no pawn of the opposing side that may immediately execute the en passant capture.

The piece placement, active color, castling availability, and en passant target square data for the starting array is:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq -
```

## 5: Operations

An EPD operation is composed of an opcode followed by zero or more operands and is concluded by a semicolon.

Multiple operations are separated by a single space character. If there is at least one operation present in an EPD line, it is separated from the last (fourth) data field by a single space character.

### 5.1: General format of opcodes and operands

An opcode is an identifier that starts with a letter character and may be followed by up to fourteen more characters. Each additional character may be a letter or a digit or the underscore character. Traditionally, no uppercase letters are used in opcode names that are to be used by more than one program.

An operand is either a set of contiguous non-white space printing characters or a string. A string is a set of contiguous printing characters delimited by a quote (ASCII code: 34 decimal, 0x22 hexadecimal) character at each end. A string value must have less than 256 bytes of data. This count does not include the traditional

ASCII NUL character terminator.

If at least one operand is present in an operation, there is a single space between the opcode and the first operand. If more than one operand is present in an operation, there is a single blank character between every two adjacent operands. If there are no operands, a semicolon character is appended to the opcode to mark the end of the operation. If any operands appear, the last operand has an appended semicolon that marks the end of the operation.

Any given opcode appears at most once per EPD record. Multiple operations in a single EPD record should appear in ASCII order of their opcode names (mnemonics). However, a program reading EPD records may allow for operations not in ASCII order by opcode mnemonics; the semantics are the same in either case.

Some opcodes that allow for more than one operand may have special ordering requirements for the operands. For example, the "pv" (predicted variation) opcode requires its operands (moves) to appear in the order in which they would be played. Most other opcodes that allow for more than one operand should have operands appearing in ASCII order. An example of the latter set is the "bm" (best move[s]) opcode; its operands are moves that are all immediately playable from the current position.

## 5.2: Operand basetypes

Operand values are represented using a variety of basetypes.

### 5.2.1: Identifier basetype

Some opcodes require one of more operands that are identifiers. An identifier is an unquoted sequence of one to fifteen characters. The characters are selected from the upper and lower case letters, the ten digits, and the underscore character. Most identifiers that may appear in EPD are taken from predefined sets as explained in the sections covering opcode semantics.

Identifiers are most often used to select one value from a list of possible values for a general attribute. They are also used to represent PGN tag attributes.

### 5.2.2: Chess move basetype

Some opcodes require one or more operands that are chess moves. These moves should be represented using SAN (Standard Algebraic Notation). If a different representation is used, there is no guarantee that the EPD will be read correctly during subsequent processing. In particular, EDN (English Descriptive Notation), CCN (Computer Coordinate Notation), and LAN (Long Algebraic Notation) are explicitly not supported.

Chess moves are used most often in single operand operations to select one move from the available moves. They are also used in multiple operand operations to define a set of moves (all taken from available moves) and in multiple operand operations to express a sequence of moves (taken from moves available at each point in a forward sequence of play).

Note that some chess moves also qualify as identifiers. However, the semantics of a particular opcode dictate the exact basetype interpretation of its operands, so there is no ambiguity.

### 5.2.3: Integer basetype

Some opcodes require one or more operands that are integers. Some opcodes may require that an integer operand must be within a given range; the details are described in the opcode list given below. A negative integer is formed with a hyphen (minus sign) preceding the integer digit sequence. An optional plus sign may be used for indicating a non-negative value, but such use is not required and is discouraged. Support for integers in the range -2147483648 to 2147483647 (32 bit two's complement signed extrema) is required.

Integers are used to represent centipawn scores and also for various counts, limits, and totals.

### 5.2.4: Floating basetype

Some opcodes require one or more operands that are floating point numbers. Some opcodes may require that a floating point operand must be within a given range; the details are described in the opcode list given below. A floating point operand is constructed from an optional sign character ("+" or "-"), a digit sequence (with at least one digit), a radix point (always "."), and a final digit sequence (with at least one digit). There is currently no provision for scientific representation of numeric values.

The floating basetype in not in current use.

### 5.2.5: Date basetype

Some opcodes require one or more operands that represent dates. These are given in a special date format composed of ten characters. The first four characters are digits that give the year (0001-9999), the fifth character is a period, the sixth and seventh characters are digits that give the month number (01-12), the eighth character is a period, and the ninth and tenth characters are digits that give the day number in the month (01-31).

The date basetype is used to specify date values in timestamps.

### 5.2.6: Time of day basetype

Some opcodes require one or more operands that represent a time of day. These are given in a special time of day format composed of eight characters. The first two characters are digits that give the hour (00-23), the third character is a colon, the fourth and fifth characters are digits that give the minute (00-59), the sixth character is a colon, and the seventh and eighth characters are digits that give the second (00-59).

The time of day basetype is used to specify time of day values in timestamps.

### 5.2.7: Clock basetype

Some opcodes require one or more operands that represent a total amount of time as would be measured by a traditional digital clock. These are given in a special clock format composed of 12 characters. The first three characters are digits giving a count of days (000-999), the fourth character is a colon, the fifth and sixth characters are digits giving a count of hours (00-23), the seventh character is a colon, the eighth and ninth characters are digits giving a count of minutes (00-59), the tenth character is a colon, and the eleventh and twelfth characters are digits giving a count of seconds (00-59).

The clock basetype is used to specify clock values for chess clock information. It is not used to measure time consumption for a search; an integer count of seconds is used instead.

## 5.3: Opcode mnemonics

An opcode mnemonic used for archival storage and for interprogram communication starts with a lower case letter and is composed of only lower case letters, digits, and the underscore character (i.e., no upper case letters). Mnemonics are all at least two characters long.

Opcode mnemonics used only by a single program or an experimental suite of programs should start with an upper case letter. This is so they may be easily distinguished should they be inadvertently be encountered by other programs. When a such a "private" opcode be demonstrated to be widely useful, it should be brought into the official list (appearing below) in a lower case form.

If a given program does not recognize a particular opcode, that operation is simply ignored; it is not signaled as an error.

## 6: Opcode list

The opcodes are listed here in ASCII order of their mnemonics. Suggestions for new opcodes should be sent to the technical contact listed near the start of this document.

### 6.1: Opcode "acn": analysis count: nodes

The opcode "acn" takes a single non-negative integer operand. It is used to represent the number of nodes examined in an analysis or search. Note that the value may be quite large for some extended searches and so use of a long (four byte) representation is suggested.

### 6.2: Opcode "acs": analysis count: seconds

The opcode "acs" takes a single non-negative integer operand. It is used to represent the number of seconds used for an analysis or search. Note that the value may be quite large for some extended searches and so use of a long (four byte) representation is suggested. Also note that the special clock format is not used for this operand. Some systems can distinguish between elapsed time and processor time; in such cases, the processor time should be used as its value is usually more indicative of search effort than wall clock time.

### 6.3: Opcode "am": avoid move(s)

The opcode "am" indicates a set of zero or more moves, all immediately playable from the current position, that are to be avoided as a search result. Each operand is a SAN move; they appear in ASCII order.

### 6.4: Opcode "bm": best move(s)

The opcode "bm" indicates a set of zero or more moves, all immediately playable from the current position, that are judged to the best available by the EPD writer and so each is allowable as a search result. Each operand is a SAN move; they appear in ASCII order.

## 6.5: Opcode "c0": comment (primary, also "c1" though "c9")

The opcode "c0" (lower case letter "c", digit character zero) indicates a top level comment that applies to the given position. It is the first of ten ranked comments, each of which has a mnemonic formed from the lower case letter "c" followed by a single decimal digit. Each of these opcodes takes either a single string operand or no operand at all.

This ten member comment family of opcodes is intended for use as descriptive commentary for a complete game or game fragment. The usual processing of these opcodes are as follows:

1) At the beginning of a game (or game fragment), a move sequence scanning program initializes each element of its set of ten comment string registers to be null.

2) As the EPD record for each position in the game is processed, the comment operations are interpreted from left to right. (Actually, all operations in an EPD record are interpreted from left to right.) Because operations appear in ASCII order according to their opcode mnemonics, opcode "c0" (if present) will be handled prior to all other opcodes, then opcode "c1" (if present), and so forth until opcode "c9" (if present).

3) The processing of opcode "cN" (0 <= N <= 9) involves two steps. First, all comment string registers with an index equal to or greater than N are set to null. (This is the set "cN" though "c9".) Second, and only if a string operand is present, the value of the corresponding comment string register is set equal to the string operand.

## 6.6: Opcode "cc": chess clock values

The opcode "cc" is used to indicate the amount of time used for each side at the time of the writing of the opcode to the EPD record. This opcode always takes two values. Both values are in clock format. The first is the amount of time consumed by White and the second is the amount of time consumed by Black. Note that these values are not simple integers. Also, there is no provision for recording at a resolution of less than one second.

This opcode is most commonly used by a mediation program as a source of impartial time information for a pair of opposing players.

## 6.7: Opcode "ce": centipawn evaluation

The opcode "ce" indicates the evaluation of the indicated position in centipawn units. It takes a single operand, an optionally signed integer that gives an evaluation of the position from the viewpoint of the active player; i.e., the player with the move. Positive values indicate a position favorable to the moving player while negative values indicate a position favorable to the passive player; i.e., the player without the move. A centipawn evaluation value close to zero indicates a neutral positional evaluation.

Values are restricted to integers that are equal to or greater than -32768 and are less than or equal to 32766.

A value greater than 32000 indicates the availability of a forced mate to the active player. The number of plies until mate is given by subtracting the evaluation from the value 32767. Thus, a winning mate in N fullmoves is a mate in ((2 * N) - 1) halfmoves (or ply) and has a corresponding centipawn evaluation of (32767 - ((2 * N) - 1)). For example, a mate on the move (mate in one) has a centipawn evaluation of 32766 while a mate in five has a centipawn evaluation of 32758.

A value less than -32000 indicates the availability of a forced mate to the passive player. The number of plies until mate is given by subtracting the evaluation from the value -32767 and then negating the result. Thus, a losing mate in N fullmoves is a mate in (2 * N) halfmoves (or ply) and has a corresponding centipawn evaluation of (-32767 + (2 * N)). For example, a mate after the move (losing mate in one) has a centipawn evaluation of -32765 while a losing mate in five has a centipawn evaluation of -32757.

A value of -32767 indicates that the side to move is checkmated. A value of -32768 indicates an illegal position. A stalemate position has a centipawn evaluation of zero as does a position drawn due to insufficient mating material. Any other position known to be a certain forced draw also has a centipawn evaluation of zero.

## 6.8: Opcode "dm": direct mate fullmove count

The "dm" opcode is used to indicate the number of fullmoves until checkmate is to be delivered by the active color for the indicated position. It always takes a single operand which is a positive integer giving the fullmove count. For example, a position known to be a "mate in three" would have an operation of "dm 3;" to indicate this.

This opcode is intended for use with problem sets composed of positions requiring direct mate answers as solutions.

## 6.9: Opcode "draw_accept": accept a draw offer

The opcode "draw_accept" is used to indicate that a draw offer made after the move that lead to the indicated position is accepted by the active player. This opcode takes no operands.

The "draw_accept" opcode should not appear on the same EPD record as a "draw_reject" opcode.

## 6.10: Opcode "draw_claim": claim a draw

The opcode "draw_claim" is used to indicate claim by the active player that a draw exists. The draw is claimed because of a third time repetition or because of the fifty move rule or because of insufficient mating material. A supplied move (see the opcode "sm") is also required to appear as part of the same EPD record. The "draw_claim" opcode takes no operands.

The "draw_claim" opcode should not appear on the same EPD record as a "draw_offer" opcode.

## 6.11: Opcode "draw_offer": offer a draw

The opcode "draw_offer" is used to indicate that a draw is offered by the active player. A supplied move (see the opcode "sm") is also required to appear as part of the same EPD record; this move is considered played from the indicated position. The "draw_offer" opcode takes no operands.

The "draw_offer" opcode should not appear on the same EPD record as a "draw_claim" opcode.

## 6.12: Opcode "draw_reject": reject a draw offer

The opcode "draw_reject" is used to indicate that a draw offer made after the move that lead to the indicated position is rejected by the active player. This opcode takes no operands.

The "draw_reject" opcode should not appear on the same EPD record as a "draw_accept" opcode.

## 6.13: Opcode "eco": _Encyclopedia of Chess Openings_ opening code

The opcode "eco" is used to associate an opening designation from the _Encyclopedia of Chess Openings_ taxonomy with the indicated position. The opcode takes either a single string operand (the ECO opening name) or no operand at all. If an operand is present, its value is associated with an "ECO" string register of the scanning program. If there is no operand, the ECO string register of the scanning program is set to null.

The usage is similar to that of the "ECO" tag pair of the PGN standard.

## 6.14: Opcode "fmvn": fullmove number

The opcode "fmvn" represents the fullmove number associated with the position. It always takes a single operand that is the positive integer value of the move number. The value of the fullmove number for the starting array is one.

This opcode is used to explicitly represent the fullmove number in EPD that is present by default in FEN as the sixth field. Fullmove number information is usually omitted from EPD because it does not affect move generation (commonly needed for EPD-using tasks) but it does affect game notation (commonly needed for FEN-using tasks). Because of the desire for space optimization for large EPD files, fullmove numbers were dropped from EPD's parent FEN. The halfmove clock information was similarly dropped.

## 6.15: Opcode "hmvc": halfmove clock

The opcode "hmvc" represents the halfmove clock associated with the position. The halfmove clock of a position is equal to the number of plies since the last pawn move or capture. This information is used to implement the fifty move draw rule. It always takes a single operand that is the non-negative integer value of the halfmove clock. The value of the halfmove clock for the starting array is zero.

This opcode is used to explicitly represent the halfmove clock in EPD that is present by default in FEN as the fifth field. Halfmove clock information is usually omitted from EPD because it does not affect move generation (commonly needed for EPD-using tasks) but it does affect game termination issues (commonly needed for FEN-using tasks). Because of the desire for space optimization for large EPD files, halfmove clock values were dropped from EPD's parent FEN. The fullmove number information was similarly dropped.

## 6.16: Opcode "id": position identification

The opcode "id" is used to provide a simple identification label for the indicated position. It takes a single string operand.

This opcode is intended for use with test suites used for measuring chessplaying program strength. An example "id" operand for the seven hundred fifty seventh position of the one thousand one problems in Reinfeld's _1001 Winning Chess Sacrifices and Combinations_ would be "WCSAC.0757" while the fifteenth position in the twenty four problem Bratko-Kopec test suite would have an "id" operand of "BK.15".

## 6.17: Opcode "nic": _New In Chess_ opening code

The opcode "nic" is used to associate an opening designation from the _New In Chess_ taxonomy with the indicated position. The opcode takes either a single string operand (the NIC code for the opening) or no operand at all. If an operand is present, its value is associated with an "NIC" string register of the scanning program. If there is no operand, the NIC string register of the scanning program is set to null.

The usage is similar to that of the "NIC" tag pair of the PGN standard.

## 6.18: Opcode "noop": no operation

The "noop" opcode is used to indicate no operation. It takes zero or more operands, each of which may be of any type. The operation involves no processing. It is intended for use by developers for program testing purposes.

## 6.19: Opcode "pm": predicted move

The "pm" opcode is used to provide a single predicted move for the indicated position. It has exactly one operand, a move playable from the position. This move is judged by the EPD writer to represent the best move available to the active player.

If a non-empty "pv" (predicted variation) line of play is also present in the same EPD record, the first move of the predicted variation is the same as the predicted move.

The "pm" opcode is intended for use as a general "display hint" mechanism.

## 6.20: Opcode "ptp": PGN tag pair

The "ptp" opcode is used to record a PGN tag pair. It always takes an even number of operands. For each pair of operands (from left to right), the first operand in the pair is always an identifier and is interpreted as the name of a PGN tag; the second operand in the pair is always a string and is the value associated with the tag given by the first operand.

Any given PGN tag name should only appear once as a tag identifier operand in a "ptp" operation.

## 6.21: Opcode "pv": predicted variation

The "pv" opcode is used to provide a predicted variation for the indicated position. It has zero or more operands which represent a sequence of moves playable from the position. This sequence is judged by the EPD writer to represent the best play available.

If a "pm" (predicted move) operation is also present in the same EPD record, the predicted move is the same as the first move of the predicted variation.

## 6.22: Opcode "rc": repetition count

The "rc" opcode is used to indicate the number of occurrences of the indicated position. It takes a single, positive integer operand. Any position, including the initial starting position, is considered to have an "rc" value of at least one. A value of three indicates a candidate for a draw claim by the position repetition rule.

## 6.23: Opcode "refcom": referee command

The "refcom" opcode is used to represent a command from a referee program to a client program during automated competition. It takes a single identifier operand which is to be interpreted as a command by the receiving program. Note that as the operand is an identifier and not a string value, it is not enclosed in quote characters.

There are seven available operand values: conclude, disconnect, execute, fault, inform, reset, and respond.

Further details of "refcom" usage are given in the section on referee semantics later in this document.

## 6.24: Opcode "refreq": referee request

The "refreq" opcode is used to represent a request from a client program to the referee program during automated competition. It takes a single identifier operand which is to be interpreted as a request to the referee from a client program. Note that as the operand is an identifier and not a string value, it is not enclosed in quote characters.

There are four available operand values: fault, reply, sign_off, and sign_on.

Further details of "refreq" usage are given in the section on referee semantics later in this document.

## 6.25: Opcode "resign": game resignation

The opcode "resign" is used to indicate that the active player has resigned the game. This opcode takes no operands.

The "resign" opcode should not appear on the same EPD record with any of the following opcodes: "draw_accept", "draw_claim", "draw_decline', and "draw_offer".

## 6.26: Opcode "sm": supplied move

The "sm" opcode is used to provide a single supplied move for the indicated position. It has exactly one operand, a move playable from the position. This move is the move to be played from the position.

If a "sv" (supplied variation) operation is present on the same record and has at least one operand, then its first operand must match the single operand of the "sm" opcode.

The "sm" opcode is intended for use to communicate the most recent played move in an active game. It is used to communicate moves between programs in automatic play via a network. This includes correspondence play using e-mail and also programs acting as network front ends to human players.

## 6.27: Opcode "sv": supplied variation

The "sv" opcode is used to provide zero or more supplied moves for the indicated position. The operands are a move sequence playable from the position.

If an "sm" (supplied move) operation is also present on the same record and the "sv" operation has at least one operand, then the "sm" operand must match the first operand of the "sv" operation.

## 6.28: Opcode "tcgs": telecommunication: game selector

The "tcgs" opcode is one of the telecommunication family of opcodes used for games conducted via e-mail and similar means. This opcode takes a single operand that is a positive integer. It is used to select among various games in progress between the same sender and receiver.

Details of e-mail implementation await further development.

## 6.29: Opcode "tcri": telecommunication: receiver identification

The "tcri" opcode is one of the telecommunication family of opcodes used for games conducted via e-mail and similar means. This opcode takes two order dependent string operands. The first operand is the e-mail address of the receiver of the EPD record. The second operand is the name of the player (program or human) at the address who is the actual receiver of the EPD record.

Details of e-mail implementation await further development.

## 6.30: Opcode "tcsi": telecommunication: sender identification

The "tcsi" opcode is one of the telecommunication family of opcodes used for games conducted via e-mail and similar means. This opcode takes two order dependent string operands. The first operand is the e-mail address of the sender of the EPD record. The second operand is the name of the player (program or human) at the address who is the actual sender of the EPD record.

Details of e-mail implementation await further development.

## 6.31: Opcode "ts": timestamp

The "ts" opcode is used to record a timestamp value. It takes two operands. The first operand is in date format and the second operand is in time of day format. The interpretation of the combined operand values gives the time of the last modification of the EPD record. The timestamp is interpreted to be in UTC (Universal Coordinated Time, formerly known as GMT).

## 6.32: Opcode "v0": variation name (primary, also "v1" though "v9")

The opcode "v0" (lower case letter "v", digit character zero) indicates a top level variation name that applies to the given position. It is the first of ten ranked variation names, each of which has a mnemonic formed from the lower case letter "v" followed by a single decimal digit. Each of these opcodes takes either a single string operand or no operand at all.

This ten member variation name family of opcodes is intended for use as traditional variation names for a complete game or game fragment. The usual processing of these opcodes are as follows:

1) At the beginning of a game (or game fragment), a move sequence scanning program initializes each element of its set of ten variation name string registers to be null.

2) As the EPD record for each position in the game is processed, the variation name operations are interpreted from left to right. (Actually, all operations in an EPD record are interpreted from left to right.) Because operations appear in ASCII order according to their opcode mnemonics, opcode "v0" (if present) will be handled prior to all other opcodes, then opcode "v1" (if present), and so forth until opcode "v9" (if present).

3) The processing of opcode "vN" (0 <= N <= 9) involves two steps. First, all variation name string registers with an index equal to or greater than N are set to null. (This is the set "vN" though "v9".) Second, and only if a string operand is present, the value of the corresponding variation name string register is set equal to the string operand.

## 7: EPD processing verbs

An EPD processing verb is a command to an EPD capable program used to direct processing of one or more EPD files. Standardization of verb semantics among EPD capable programs is important to helping reduce confusion among program users and to better insure overall interoperatibilty.

Each EPD processing verb that requires the reading of EPD records has a specific set of required opcodes that must be on each input record. Each EPD processing verb that requires the writing of EPD records has a specific set of required opcodes that must be on each output record. Some EPD processing verbs imply both reading and writing EPD records; these will have requirements for both input and output opcode sets.

The names of the EPD processing verbs in this section are for use for specification purposes only. Program authors are free to select different names as appropriate for the needs of a program's user interface.

### 7.1: EPD verb: pfdn (process file: data normalization)

The "pfdn" (process file: data normalization) verb reads an EPD input file and produces a normalized copy of the data on as the EPD output file. The output file retains the record ordering of the input file. The noramlization is used to produce a canonical representation of the EPD. The input records are also checked for legality. There is no minimum set of operations requires on the input records. For each input record, all of the operations present are reproduced in the corresponding output record.

The normalization of each EPD record consists of the following actions:

1) Any leading whitespace characters are removed.

2) Any trailing whitespace characters are removed.

3) Any unneeded whitespace characters used as data separators are removed; a single blank is used to separate adjacent fields, adjacent operations, and adjacent operands. Also, a single blank character is used to separate the fourth position data field (the en passant target square indication) from the first operation (if present).

4) Operations are reordered in increasing ASCII order by opcode mnemonic.

5) Operands for each opcode that does not require a special order of interpretation are reordered in increasing ASCII order by external representation.

Data normalization is useful for making a canonical version from data produced by programs or other sources that do not completely conform to the lexigraphical and ordering rules of the EPD standard. It also helps when comparing two EPD files from different sources on a line by line basis; the non-semantic differences are removed so that different text lines indicate true semantic difference.

## 7.2: EPD verb: pfga (process file: general analysis)

The "pfga" (process file: general analysis) verb is used to instruct a chessplaying program to perform an analysis for each EPD input record and produce an EPD output file containing this analysis. The output file retains the record ordering of the input file. The current position given by each input record is not changed; it is copied to the output.

Each input EPD record receives the same analysis effort. The level of effort is indicated as a command (separate from EPD) to the analysis program prior to the start of the EPD processing. Usually, the level is given as a time limit or depth limit per each position. The limit can be either a hard limit or a soft limit. A hard limit represents an absolute maximum effort per position, while a soft limit allows the program to spend more or less effort per position. The hard limit interpretation is preferred for comparing programs. The soft limit interpretation is used to help test time allocation strategy where a program can choose to take more or less time depending on the complexity of a position.

Each EPD output record is a copy of the corresponding EPD input record with new analysis added as a result of the verb processing.

There is no minimum set of operations required for the EPD input records.

Each output EPD record must contain:

1) A "pv" (predicted variation) operation. The operands of this form a sequence of chess moves to be played from the given position. The length of this may vary from record to record due to the level of anaylsis effort and the complexity of each position. However, unless the current position represents a checkmate or stalemate for the side to move, the pv operation must include at least one move. If the current position represents a checkmate or stalemate for the side to move, then the pv operation still appears, but has no operands.

2) A "ce" (centipawn evaluation) operation. The value of its operand is the value in hundredths of a pawn of the current position. Note that the evaluation is assigned to the position before the predicted move (or any other move) is made. Thus, a positive centipawn score indicates an advantage for the side to move in the current position while a negative score indicates a disadvantage for the side to move.

Each output EPD record may also contain:

1) A "pm" (predicted move) operation, unless the current position represents a checkmate or stalemate for the side to move. (If the side to move has no moves, then the "pm" operation will not appear.) The single operand of the "pm" opcode must be the same as the first operand of the "pv" sequence.

2) A "sm" (supplied move) operation, unless the current position represents a checkmate or stalemate for the side to move. (If the side to move has no moves, then the "sm" operation will not appear.) The single operand of the "sm" opcode must be the same as the first operand of the "pv" sequence.

3) An "acn" (analysis count: nodes) operation. The single operand is the number of nodes visited in the analysis search for the position.

4) An "acs" (analysis count: seconds) operation. The single operand is the number of seconds used for the analysis search for the position.

## 7.3: EPD verb: pfms (process file: mate search)

The "pfms" verb is used to conduct searches for forced checkmating sequences. The length of the forced mate sequence is provided (outside of EPD) to the program prior to the beginning of "pfms" processing. The length is specified using a fullmove count. For example, a fullmove mate length of three would instruct the program to search for all mates in three. An analysis program reads and input EPD file and looks for forced mates in each position where no forced mate of equal or lesser length has been recorded. The output file retains the record ordering of the input file.

The action of the "pfms" command on each record is governed by the pre-specified fullmove count and, if present on the record, the value of the "dm" (direct mate fullmove count) operand. A particular record will be subject to a search for a forced mate if either:

1) There is no "dm" operation on the input record, or

2) The value of the "dm" operand on the input record is greater than the value of the pre-specified fullmove analysis length.

If the analysis program finds a forced mate, it produces two additional operations on the corresponding output EPD record:

1) A "dm" operation with an operand equal to the pre-specified fullmove mate length.

2) A "pm" operation with the first move of the mating sequence as its operand. If two or more such moves exist, the program selects the first one it located to appear as the "pm" operand.

The idea is that a set of positions can be repeatedly scanned by a mate finding program with the fullmove analysis depth starting with a value of one and being increased by one with each pass. For any given pass, the positions solved by an earlier pass are skipped.

The output EPD records may also contain other (optional) information such as "acn", "acs", and "pv" operations.

### 7.4: EPD verb: pfop (process file: operation purge)

The "pfop" verb is used to purge a particular operation from each of the records in an EPD file that contain the operation. The output file retains the record ordering of the input file. Prior to processing, the opcode of the operation to be purged is specified.

The records of the input file are copied to the output file. If the pre-specified operation is present on a record, the operation is removed prior to copying the record to the output.

### 7.5: EPD verb: pfts (process file: target search)

The "pfts" (process file: target search) verb is similar to the "pfga" (process file: general analysis) verb in that each position on the EPD input file is subject to a general analysis. The difference is that each input record contains a set of target moves and a set of avoidance moves. Either of these two sets, but not both, may be empty. The set of avoidance moves is given by the operands of a "am" opcode (if present). The set of target moves is given by the operands of a "bm" opcode (if present).

Prior to processing the target search, the program is given a search effort limit such as a limit on the amount of search time or search nodes per position. The "pfts" verb causes each input EPD record to be read, subjected to analysis, and then written to output file with the predicted move attached with the "pm" opcode. (No "pm" operation is added is the current position is a checkmate or stalemate of the side to play.)

The output EPD records may also contain other (optional) information such as "acn", "acs", and "pv" operations.

## 8: EPD referee semantics

Communication between a chessplaying program and a referee program is performed by exchanging EPD records. Each EPD record emitted by a chessplaying program to be received by the referee has a "refreq" EPD opcode with an operand that describes the request. Each EPD record emitted by a referee to be received by a chessplaying program has a "refcom" EPD opcode with an operand that describes the command.

The usual operation sequence in a referee mediated event is as follows:

1) The referee server program is started and the human event supervisor provides it with any necessary tournament information including the names of the chessplaying programs, the name of the event, and various other data.

2) The referee program completes its initialization by performing pairing operations as required.

3) Once the server has its initial data, it then opens a socket and binds it to the appropriate port. It then starts listening for input from clients. For a serial implementation, an analogous function is performed.

4) The competing chessplaying programs (clients) are started (if not already running) and are given the name of the referee host machine along with the port number. For a serial implementation, an analogous function is performed.

5) Each client program transmits an EPD record to the referee requesting registration. This causes each client to be signed on to the referee.

6) The referee program replies to each client signing on with an EPD record commanding a reset operation to set up for a new game.

7) The referee program sends an EPD record to each client informing each client about the values for each of the tag values for the PGN Seven Tag Format.

8) For each client on the move, the referee will send an EPD record commanding a response. This causes each receiving client to calculate a move. If there has been a prior move, it along with the position from which the move is played is sent. If there has been no prior move, the current position is sent but no move is included.

9) For each client receiving a command to respond, the current position indicated by the record is set as the current position in the receiving program. (It should already be the current position in the receiver.) If a supplied move was given, it is executed on the current position. Finally, the receiving program calculates a move.

10) As each program on the move completes its calculation, it sends a reply to the referee which includes the result of the calculation. The position sent back on the reply is the result of applying the move received on the referee record to the position on the same received record. If a move was produced as the result of the calculation, it is also sent. (A

move will not be produced or sent if the receving client was checkmated, or if it was stalemated, of if it resigns, or claims a draw due to insufficient material.)

11) As the referee receives a reply from a client, it produces a respond command record to the client's opponent. (This step will be skipped if an end of game condition is detected and no further moves need to be communicated.)

12) The referee continues with the respond/reply cycle for each pair of opponent clients until the game concludes for that pair.

13) For each game conclusion, the referee sends a conclude command to each of the clients involved.

14) When a client is to be removed from competition, it sends a sign off request. This eliminates that program from being paired until it re-registers with a sign on request.

15) When the referree server is to be removed from network operations, it will send a disconnect command to each client that is currently signed on to the referee.

## 8.1: Referee commands (client directives)

The referee communicates the command of interest as the single operand of the "refcom" opcode. The refcom opcode will be on each record sent by the referee. Each possible refcom operand is sent as an identifier (and not as a string).

EPD records sent by the referee will include check clock data as appropriate. Whenever a client program receives a record with the "cc" (chess clock) opcode, the client should set the values of its internal clocks to the values specified by the cc operands. Note that the clock values for both White and Black are present in a cc operation.

All EPD records carry the four data fields describing the current position. In most cases, this position should also be the current position of the receiving client. If the position sent by the referee matches the client's current position, then the client can assume that all of the game history leading to the current position is valid. Thus, every client keeps track of the game history internally and uses this to detect repetition draws and so there is no need for each EPD record to contain a complete copy of the game history.

If the position sent by the referee does not match the receiving program's current position, then the receiving program must set its current position to be the same as the one it received. Unless an explicit game history move sequence is also sent on the same EPD record, the receiving program is to assume that the new (different) position received has no game history. In this case the receiving program cannot check for repetition of positions prior to the new position as there aren't any previous positions in the game.

Each client is expected to maintain its own copy of the halfmove clock (plies since last irreversible move; starts at zero for the initial position) and the fullmove number (which has a value of one for the initial position). If the referee sends a halfmove clock value or a fullmove number which is different from that kept by the program, then the receiving program is to treat it as a new position and clear any game history. As noted above, a halfmove clock is sent using the "hmvc" opcode and a fullmove number is sent using a "fmvn" opcode.

If a supplied move (always using the "sm" opcode) is sent by the referee, the receiving program must execute this move on the current position. This is done after the program's current position is set to the position sent by the referee (remember that the two will usually match). The resulting position becomes the new current position. This new current position is used for all further calculations. The new current position is also the position to be sent to the referee if a move response is commanded. When a client program produces a move to be played, it uses the sm opcode with its operand being the supplied move. The position sent is alwasy the position from which the supplied move is to be played. Thus, the semantics of the current position and the supplied move are symmetric with respect to the client and the server.

### 8.1.1: Referee command: conclude

The "conclude" refcom operand instructs the client to conclude the current game in progress. The position sent is the final position of the game. There is no supplied move sent. No further EPD records concerning the game will be sent by the referee. The client should perform any end of game activity required for its normal operation. No response from the client is made.

To allow for client game conclusion processing time, the referee will avoid sending any more EPD records to a client concluding a game for a time period set by the human supervisor. The default delay will be five seconds.

### 8.1.2: Referee command: disconnect

The "disconnect" refcom operand instructs the client that the referee is terminating service operations. The client should close its communication channel with the server. This command is sent at the end of an event or whenever the referee is to be brought down for some reason. No further EPD records will be sent until the server is cycled. It provides an opportunity for a client to gracefully disconnect from network operations with the server. No supplied move is sent. The position sent is irrelevant. No response from the client is made.

### 8.1.3: Referee command: execute

The "execute" refcom operand instructs the client to set up a position. If a move is supplied (it usually is), then that move is executed from the position. The sent position will usually be the receiver's current position. This command is used only to play through the initial sequence of moves from a game to support a restart capability. No response is made by the receiver.

### 8.1.4: Referee command: fault

The "fault" refcom operand is used to indicate that the referee has detected an unrecoverable fault. The reciever should signal for human intervention to assist with corrective action. The human supervisor will be notified by the referee regarding the nature of the fault. No response is made by the receiver.

A future version of the referee protocol will support some form of automated fault recovery.

### 8.1.5: Referee command: inform

The "inform" refcom operand is used to convey PGN tag pair data to the receiver. The "ptp" opcode will carry the PGN tag data to be set on the receiving client. This command may be sent at any time. It will usually be sent prior to the first move of a game. It will also be sent after the last move of a game to communicate the result of the game via the PGN "Result" tag pair. No response is made by the receiver.

The main purpose for the inform referee command is to be able to communcate tag pair data to a client without having to send a move or other command. Note that the ptp opcode may also appear on EPD records from the referee that are not inform commands; its operands are processed in the same way.

The usual information sent includes the values for the Seven Tag Roster. The PGN tag names are "Event", "Site", "Date", "Round", "White", "Black", and "Result".

Future versions of the referee will likely send more than just the Seven Tag Roster of PGN tag pairs. One probable addition will be to send the "TimeControl" tag pair prior to the start of a game; this will allow a receiving program to have its time control parameters set automatically rather than manually.

### 8.1.6: Referee command: reset

The "reset" refcom operand is used to command the receiving client to set up for a new game. Any previous information about a game in progress is deleted. This command will be sent to mark the beginning of a game. It will also be sent if there is a need to abort the game currently in progress. No response is made by the receiver.

To allow for client reset processing time, the referee will avoid sending any more EPD records to a resetting client for a time period set by the human supervisor. The default delay will be five seconds.

### 8.1.7: Referee command: respond

The "respond" refcom operand is used to command the receiving client to respond to the move (if any) played by its opponent. The position to use for calculation is the position sent which is modified by a supplied move (if present; uses the "sm" opcode). The client program calculates a response and sends it to the referee using the "reply" operand of the "refreq" opcode.

## 8.2: Referee requests (server directives)

The referee communicates the command of interest as the single operand of the "refcom" opcode. The refcom opcode will be on each record sent by the referee. Each possible refcom operand is sent as an identifier (and not as a string).

### 8.2.1: Referee request: fault

The "fault" refreq operand is used to indicate that the client has detected an unrecoverable fault. The receiver should signal for human intervention to assist with corrective action. The human supervisor will be notified by the referee regarding the nature of the fault. No response is made by the referee.

A future version of the referee protocol will support some form of automated fault recovery.

### 8.2.2: Referee request: reply

The "reply" refreq operand is used to carry a reply by the client program.

Usually, a move (the client's reply) is included as the operand of the "sm" opcode.

### 8.2.3: Referee request: sign_off

The "sign_off" refreq operand is used to indicate that the client program is signing off from the referee connection and no further operations will be made on the communication channel. The channel in use is then closed by both the referee and the client.

A new connection must be established and a new "sign_on" referee request needs to be made for further referee operations with the client.

### 8.2.4: Referee request: sign_on

The "sign_on" refreq operand is used to indicate that the client program is signing on to the referee connection. This request is required before any further operations can be made on the communication channel. The channel in use remains open until it is closed by either side.

# 9: EPD report generation semantics

[TBD]