Standard: Portable Game Notation Specification and Implementation Guide

Revised: 1994.03.12

Authors: Interested readers of the Internet newsgroup rec.games.chess

Coordinator: Steven J. Edwards (send comments to sje@world.std.com)

0: Preface

From the Tower of Babel story:

"If now, while they are one people, all speaking the same language, they have started to do this, nothing will later stop them from doing whatever they propose to do."

Genesis XI, v.6, *New American Bible*

1: Introduction

PGN is "Portable Game Notation", a standard designed for the representation of chess game data using ASCII text files. PGN is structured for easy reading and writing by human users and for easy parsing and generation by computer programs. The intent of the definition and propagation of PGN is to facilitate the sharing of public domain chess game data among chessplayers (both organic and otherwise), publishers, and computer chess researchers throughout the world.

PGN is not intended to be a general purpose standard that is suitable for every possible use; no such standard could fill all conceivable requirements. Instead, PGN is proposed as a universal portable representation for data interchange. The idea is to allow the construction of a family of chess

interchange. The idea is to allow the construction of a family of chess applications that can quickly and easily process chess game data using PGN for import and export among themselves.

2: Chess data representation

Computer usage among chessplayers has become quite common in recent years and a variety of different programs, both commercial and public domain, are used to generate, access, and propagate chess game data. Some of these programs are rather impressive; most are now well behaved in that they correctly follow the Laws of Chess and handle users' data with reasonable care. Unfortunately, many programs have had serious problems with several aspects of the external representation of chess game data. Sometimes these problems become more visible when a user attempts to move significant quantities of data from one program to another; if there has been no real effort to ensure portability of data, then the chances for a successful transfer are small at best.

2.1: Data interchange incompatibility

The reasons for format incompatibility are easy to understand. In fact, most of them are correlated with the same problems that have already been seen with commercial software offerings for other domains such as word processing, spreadsheets, fonts, and graphics. Sometimes a manufacturer deliberately designs a data format using encryption or some other secret, proprietary technique to "lock in" a customer. Sometimes a designer may produce a format that can be deciphered without too much difficulty, but at the same time publicly discourage third party software by claiming trade secret protection. Another software producer may develop a non-proprietary system, but it may work well only within the scope of a single program or application because it is not easily expandable. Finally, some other software may work very well for many purposes, but it uses symbols and language not easily understood by people or computers available to those outside the country of its development.

2.2: Specification goals

A specification for a portable game notation must observe the lessons of

history and be able to handle probable needs of the future. The design

criteria for PGN were selected to meet these needs. These criteria include:

1) The details of the system must be publicly available and free of unnecessary complexity. Ideally, if the documentation is not available for some reason, typical chess software developers and users should be able to understand most of the data without the need for third party assistance.

2) The details of the system must be non-proprietary so that users and software developers are unrestricted by concerns about infringing on intellectual property rights. The idea is to let chess programmers compete in a free market where customers may choose software based on their real needs and not based on artificial requirements created by a secret data format.

3) The system must work for a variety of programs. The format should be such that it can be used by chess database programs, chess publishing programs, chess server programs, and chessplaying programs without being unnecessarily specific to any particular application class.

4) The system must be easily expandable and scalable. The expansion ability must include handling data items that may not exist currently but could be expected to emerge in the future. (Examples: new opening classifications and new country names.) The system should be scalable in that it must not have any arbitrary restrictions concerning the quantity of stored data. Also, planned modes of expansion should either preserve earlier databases or at least allow for their automatic conversion.

5) The system must be international. Chess software users are found in many countries and the system should be free of difficulties caused by conventions local to a given region.

6) Finally, the system should handle the same kinds and amounts of data that are already handled by existing chess software and by print media.

```
2.3: A sample PGN game
```

```
Although its description may seem rather lengthy, PGN is actually fairly
simple.  A sample PGN game follows; it has most of the important features
described in later sections of this document.
```

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]
```

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3
O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15.
Nb1 h6 16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21.
Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7
27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33.
f3 Bc8 34. Kf2 Bf5 35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5
40. Rd6 Kc5 41. Ra6 Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

```
3: Formats: import and export
```

```
There are two formats in the PGN specification.  These are the "import" format
and the "export" format.  These are the two different ways of formatting the
same PGN data according to its source.  The details of the two formats are
described throughout the following sections of this document.
```

```
Other than formats, there is the additional topic of PGN presentation.  While
both PGN import and export formats are designed to be readable by humans, there
is no recommendation that either of these be an ultimate mode of chess data
presentation.  Rather, software developers are urged to consider all of the
various techniques at their disposal to enhance the display of chess data at
```

the presentation level (i.e., highest level) of their programs. This means that the use of different fonts, character sizes, color, and other tools of computer aided interaction and publishing should be explored to provide a high quality presentation appropriate to the function of the particular program.

3.1: Import format allows for manually prepared data

The import format is rather flexible and is used to describe data that may have been prepared by hand, much like a source file for a high level programming language. A program that can read PGN data should be able to handle the somewhat lax import format.

3.2: Export format used for program generated output

The export format is rather strict and is used to describe data that is usually prepared under program control, something like a pretty printed source program reformatted by a compiler.

3.2.1: Byte equivalence

For a given PGN data file, export format representations generated by different PGN programs on the same computing system should be exactly equivalent, byte for byte.

3.2.2: Archival storage and the newline character

Export format should also be used for archival storage. Here, "archival" storage is defined as storage that may be accessed by a variety of computing systems. The only extra requirement for archival storage is that the newline character have a specific representation that is independent of its value for a particular computing system's text file usage. The archival representation of a newline is the ASCII control character LF (line feed, decimal value 10, hexadecimal value 0x0a).

```
Sadly, there are some accidents of history that survive to this day that have
baroque representations for a newline: multicharacter sequences, end-of-line
record markers, start-of-line byte counts, fixed length records, and so forth.
It is well beyond the scope of the PGN project to reconcile all of these to the
unified world of ANSI C and the those enjoying the bliss of a single '\n'
convention.  Some systems may just not be able to handle an archival PGN text
file with native text editors.  In these cases, an indulgence of sorts is
granted to use the local newline convention in non-archival PGN files for those
text editors.
```

```
3.2.3: Speed of processing
```

```
Several parts of the export format deal with exact descriptions of line and
field justification that are absent from the import format details.  The main
reason for these restrictions on the export format are to allow the
construction of simple data translation programs that can easily scan PGN data
without having to have a full chess engine or other complex parsing routines.
The idea is to encourage chess software authors to always allow for at least a
limited PGN reading capability.  Even when a full chess engine parsing
capability is available, it is likely to be at least two orders of magnitude
slower than a simple text scanner.
```

```
3.2.4: Reduced export format
```

```
A PGN game represented using export format is said to be in "reduced export
format" if all of the following hold: 1) it has no commentary, 2) it has only
the standard seven tag roster identification information ("STR", see below), 3)
it has no recursive annotation variations ("RAV", see below), and 4) it has no
numeric annotation glyphs ("NAG", see below).  Reduced export format is used
for bulk storage of unannotated games.  It represents a minimum level of
standard conformance for a PGN exporting application.
```

```
4: Lexicographical issues
```

PGN data is composed of characters; non-overlapping contiguous sequences of characters form lexical tokens.

4.1: Character codes

PGN data is represented using a subset of the eight bit ISO 8859/1 (Latin 1) character set. ("ISO" is an acronym for the International Standards Organization.) This set is also known as ECMA-94 and is similar to other ISO Latin character sets. ISO 8859/1 includes the standard seven bit ASCII character set for the 32 control character code values from zero to 31. The 95 printing character code values from 32 to 126 are also equivalent to seven bit ASCII usage. (Code value 127, the ASCII DEL control character, is a graphic character in ISO 8859/1; it is not used for PGN data representation.)

The 32 ISO 8859/1 code values from 128 to 159 are non-printing control characters. They are not used for PGN data representation. The 32 code values from 160 to 191 are mostly non-alphabetic printing characters and their use for PGN data is discouraged as their graphic representation varies considerably among other ISO Latin sets. Finally, the 64 code values from 192 to 255 are mostly alphabetic printing characters with various diacritical marks; their use is encouraged for those languages that require such characters. The graphic representations of this last set of 64 characters is fairly constant for the ISO Latin family.

Printing character codes outside of the seven bit ASCII range may only appear in string data and in commentary. They are not permitted for use in symbol construction.

Because some PGN users' environments may not support presentation of non-ASCII characters, PGN game authors should refrain from using such characters in critical commentary or string values in game data that may be referenced in such environments. PGN software authors should have their programs handle such environments by displaying a question mark ("?") for non-ASCII character codes. This is an important point because there are many computing systems that can

display eight bit character data, but the display graphics may differ among machines and operating systems from different manufacturers.

Only four of the ASCII control characters are permitted in PGN import format; these are the horizontal and vertical tabs along with the linefeed and carriage return codes.

The external representation of the newline character may differ among platforms; this is an acceptable variation as long as the details of the implementation are hidden from software implementors and users. When a choice is practical, the Unix "newline is linefeed" convention is preferred.

4.2: Tab characters

Tab characters, both horizontal and vertical, are not permitted in the export format. This is because the treatment of tab characters is highly dependent upon the particular software in use on the host computing system. Also, tab characters may not appear inside of string data.

4.3: Line lengths

PGN data are organized as simple text lines without any special bytes or markers for secondary record structure imposed by specific operating systems. Import format PGN text lines are limited to having a maximum of 255 characters per line including the newline character. Lines with 80 or more printing characters are strongly discouraged because of the difficulties experienced by common text editors with long lines.

In some cases, very long tag values will require 80 or more columns, but these are relatively rare. An example of this is the "FEN" tag pair; it may have a long tag value, but this particular tag pair is only used to represent a game that doesn't start from the usual initial position.

5: Commentary

Comment text may appear in PGN data. There are two kinds of comments. The
first kind is the "rest of line" comment; this comment type starts with a
semicolon character and continues to the end of the line. The second kind
starts with a left brace character and continues to the next right brace
character. Comments cannot appear inside any token.

Brace comments do not nest; a left brace character appearing in a brace comment
loses its special meaning and is ignored. A semicolon appearing inside of a
brace comment loses its special meaning and is ignored. Braces appearing
inside of a semicolon comments lose their special meaning and are ignored.

*** Export format representation of comments needs definition work.

6: Escape mechanism

There is a special escape mechanism for PGN data. This mechanism is triggered
by a percent sign character ("%") appearing in the first column of a line; the
data on the rest of the line is ignored by publicly available PGN scanning
software. This escape convention is intended for the private use of software
developers and researchers to embed non-PGN commands and data in PGN streams.

A percent sign appearing in any other place other than the first position in a
line does not trigger the escape mechanism.

7: Tokens

PGN character data is organized as tokens. A token is a contiguous sequence of
characters that represents a basic semantic unit. Tokens may be separated from
adjacent tokens by white space characters. (White space characters include
space, newline, and tab characters.) Some tokens are self delimiting and do
not require white space characters.

A string token is a sequence of zero or more printing characters delimited by a pair of quote characters (ASCII decimal value 34, hexadecimal value 0x22). An empty string is represented by two adjacent quotes. (Note: an apostrophe is not a quote.) A quote inside a string is represented by the backslash immediately followed by a quote. A backslash inside a string is represented by two adjacent backslashes. Strings are commonly used as tag pair values (see below). Non-printing characters like newline and tab are not permitted inside of strings. A string token is terminated by its closing quote. Currently, a string is limited to a maximum of 255 characters of data.

An integer token is a sequence of one or more decimal digit characters. It is a special case of the more general "symbol" token class described below. Integer tokens are used to help represent move number indications (see below). An integer token is terminated just prior to the first non-symbol character following the integer digit sequence.

A period character (".") is a token by itself. It is used for move number indications (see below). It is self terminating.

An asterisk character ("*") is a token by itself. It is used as one of the possible game termination markers (see below); it indicates an incomplete game or a game with an unknown or otherwise unavailable result. It is self terminating.

The left and right bracket characters ("[" and "]") are tokens. They are used to delimit tag pairs (see below). Both are self terminating.

The left and right parenthesis characters ("(" and ")") are tokens. They are used to delimit Recursive Annotation Variations (see below). Both are self terminating.

The left and right angle bracket characters ("<" and ">") are tokens. They are reserved for future expansion. Both are self terminating.

A Numeric Annotation Glyph ("NAG", see below) is a token; it is composed of a dollar sign character ("$") immediately followed by one or more digit characters. It is terminated just prior to the first non-digit character following the digit sequence.

A symbol token starts with a letter or digit character and is immediately followed by a sequence of zero or more symbol continuation characters. These continuation characters are letter characters ("A-Za-z"), digit characters ("0-9"), the underscore ("_"), the plus sign ("+"), the octothorpe sign ("#"), the equal sign ("="), the colon (":"), and the hyphen ("-"). Symbols are used for a variety of purposes. All characters in a symbol are significant. A symbol token is terminated just prior to the first non-symbol character following the symbol character sequence. Currently, a symbol is limited to a maximum of 255 characters in length.

8: Parsing games

A PGN database file is a sequential collection of zero or more PGN games. An empty file is a valid, although somewhat uninformative, PGN database.

A PGN game is composed of two sections. The first is the tag pair section and the second is the movetext section. The tag pair section provides information that identifies the game by defining the values associated with a set of standard parameters. The movetext section gives the usually enumerated and possibly annotated moves of the game along with the concluding game termination marker. The chess moves themselves are represented using SAN (Standard Algebraic Notation), also described later in this document.

8.1: Tag pair section

The tag pair section is composed of a series of zero or more tag pairs.

A tag pair is composed of four consecutive tokens: a left bracket token, a symbol token, a string token, and a right bracket token. The symbol token is

the tag name and the string token is the tag value associated with the tag name. (There is a standard set of tag names and semantics described below.) The same tag name should not appear more than once in a tag pair section.

A further restriction on tag names is that they are composed exclusively of letters, digits, and the underscore character. This is done to facilitate mapping of tag names into key and attribute names for use with general purpose database programs.

For PGN import format, there may be zero or more white space characters between any adjacent pair of tokens in a tag pair.

For PGN export format, there are no white space characters between the left bracket and the tag name, there are no white space characters between the tag value and the right bracket, and there is a single space character between the tag name and the tag value.

Tag names, like all symbols, are case sensitive. All tag names used for archival storage begin with an upper case letter.

PGN import format may have multiple tag pairs on the same line and may even have a tag pair spanning more than a single line. Export format requires each tag pair to appear left justified on a line by itself; a single empty line follows the last tag pair.

Some tag values may be composed of a sequence of items. For example, a consultation game may have more than one player for a given side. When this occurs, the single character ":" (colon) appears between adjacent items. Because of this use as an internal separator in strings, the colon should not otherwise appear in a string.

The tag pair format is designed for expansion; initially only strings are allowed as tag pair values. Tag value formats associated with the STR (Seven Tag Roster, see below) will not change; they will always be string values.

However, there are long term plans to allow general list structures as tag values for non-STR tag pairs. Use of these expanded tag values will likely be restricted to special research programs. In all events, the top level structure of a tag pair remains the same: left bracket, tag name, tag value, and right bracket.

8.1.1: Seven Tag Roster

There is a set of tags defined for mandatory use for archival storage of PGN data. This is the STR (Seven Tag Roster). The interpretation of these tags is fixed as is the order in which they appear. Although the definition and use of additional tag names and semantics is permitted and encouraged when needed, the STR is the common ground that all programs should follow for public data interchange.

For import format, the order of tag pairs is not important. For export format, the STR tag pairs appear before any other tag pairs. (The STR tag pairs must also appear in order; this order is described below). Also for export format, any additional tag pairs appear in ASCII order by tag name.

The seven tag names of the STR are (in order):

1) Event (the name of the tournament or match event)

2) Site (the location of the event)

3) Date (the starting date of the game)

4) Round (the playing round ordinal of the game)

5) White (the player of the white pieces)

6) Black (the player of the black pieces)

7) Result (the result of the game)

A set of supplemental tag names is given later in this document.

For PGN export format, a single blank line appears after the last of the tag pairs to conclude the tag pair section. This helps simple scanning programs to quickly determine the end of the tag pair section and the beginning of the movetext section.

8.1.1.1: The Event tag

The Event tag value should be reasonably descriptive. Abbreviations are to be avoided unless absolutely necessary. A consistent event naming should be used to help facilitate database scanning. If the name of the event is unknown, a single question mark should appear as the tag value.

Examples:

[Event "FIDE World Championship"]

[Event "Moscow City Championship"]

[Event "ACM North American Computer Championship"]

[Event "Casual Game"]

8.1.1.2: The Site tag

The Site tag value should include city and region names along with a standard name for the country. The use of the IOC (International Olympic Committee) three letter names is suggested for those countries where such codes are available. If the site of the event is unknown, a single question mark should appear as the tag value. A comma may be used to separate a city from a region. No comma is needed to separate a city or region from the IOC country code. A later section of this document gives a list of three letter nation codes along with a few additions for "locations" not covered by the IOC.

Examples:

```
[Site "New York City, NY USA"]
```

```
[Site "St. Petersburg RUS"]
```

```
[Site "Riga LAT"]
```

8.1.1.3: The Date tag

The Date tag value gives the starting date for the game. (Note: this is not necessarily the same as the starting date for the event.) The date is given with respect to the local time of the site given in the Event tag. The Date tag value field always uses a standard ten character format: "YYYY.MM.DD". The first four characters are digits that give the year, the next character is a period, the next two characters are digits that give the month, the next character is a period, and the final two characters are digits that give the day of the month. If the any of the digit fields are not known, then question marks are used in place of the digits.

Examples:

```
[Date "1992.08.31"]
```

```
[Date "1993.??.??"]
```

```
[Date "2001.01.01"]
```

8.1.1.4: The Round tag

The Round tag value gives the playing round for the game. In a match competition, this value is the number of the game played. If the use of a round number is inappropriate, then the field should be a single hyphen character. If the round is unknown, a single question mark should appear as the tag value.

Some organizers employ unusual round designations and have multipart playing rounds and sometimes even have conditional rounds. In these cases, a multipart round identifier can be made from a sequence of integer round numbers separated by periods. The leftmost integer represents the most significant round and succeeding integers represent round numbers in descending hierarchical order.

Examples:

```
[Round "1"]
```

```
[Round "3.1"]
```

```
[Round "4.1.2"]
```

8.1.1.5: The White tag

The White tag value is the name of the player or players of the white pieces. The names are given as they would appear in a telephone directory. The family or last name appears first. If a first name or first initial is available, it

is separated from the family name by a comma and a space. Finally, one or more middle initials may appear. (Wherever a comma appears, the very next character should be a space. Wherever an initial appears, the very next character should be a period.) If the name is unknown, a single question mark should appear as the tag value.

The intent is to allow meaningful ASCII sorting of the tag value that is independent of regional name formation customs. If more than one person is playing the white pieces, the names are listed in alphabetical order and are separated by the colon character between adjacent entries. A player who is also a computer program should have appropriate version information listed after the name of the program.

The format used in the FIDE Rating Lists is appropriate for use for player name tags.

Examples:

[White "Tal, Mikhail N."]

[White "van der Wiel, Johan"]

[White "Acme Pawngrabber v.3.2"]

[White "Fine, R."]

8.1.1.6: The Black tag

The Black tag value is the name of the player or players of the black pieces. The names are given here as they are for the White tag value.

Examples:

```
[Black "Lasker, Emmanuel"]
```

```
[Black "Smyslov, Vasily V."]
```

```
[Black "Smith, John Q.:Woodpusher 2000"]
```

```
[Black "Morphy"]
```

8.1.1.7: The Result tag

```
The Result field value is the result of the game.  It is always exactly the
same as the game termination marker that concludes the associated movetext.  It
is always one of four possible values: "1-0" (White wins), "0-1" (Black wins),
"1/2-1/2" (drawn game), and "*" (game still in progress, game abandoned, or
result otherwise unknown).  Note that the digit zero is used in both of the
first two cases; not the letter "O".
```

All possible examples:

```
[Result "0-1"]
```

```
[Result "1-0"]
```

```
[Result "1/2-1/2"]
```

```
[Result "*"]
```

8.2: Movetext section

The movetext section is composed of chess moves, move number indications, optional annotations, and a single concluding game termination marker.

Because illegal moves are not real chess moves, they are not permitted in PGN movetext. They may appear in commentary, however. One would hope that illegal moves are relatively rare in games worthy of recording.

8.2.1: Movetext line justification

In PGN import format, tokens in the movetext do not require any specific line justification.

In PGN export format, tokens in the movetext are placed left justified on successive text lines each of which has less than 80 printing characters. As many tokens as possible are placed on a line with the remainder appearing on successive lines. A single space character appears between any two adjacent symbol tokens on the same line in the movetext. As with the tag pair section, a single empty line follows the last line of data to conclude the movetext section.

Neither the first or the last character on an export format PGN line is a space. (This may change in the case of commentary; this area is currently under development.)

8.2.2: Movetext move number indications

A move number indication is composed of one or more adjacent digits (an integer token) followed by zero or more periods. The integer portion of the indication gives the move number of the immediately following white move (if present) and also the immediately following black move (if present).

8.2.2.1: Import format move number indications

PGN import format does not require move number indications. It does not prohibit superfluous move number indications anywhere in the movetext as long as the move numbers are correct.

PGN import format move number indications may have zero or more period characters following the digit sequence that gives the move number; one or more white space characters may appear between the digit sequence and the period(s).

8.2.2.2: Export format move number indications

There are two export format move number indication formats, one for use appearing immediately before a white move element and one for use appearing immediately before a black move element. A white move number indication is formed from the integer giving the fullmove number with a single period character appended. A black move number indication is formed from the integer giving the fullmove number with three period characters appended.

All white move elements have a preceding move number indication. A black move element has a preceding move number indication only in two cases: first, if there is intervening annotation or commentary between the black move and the previous white move; and second, if there is no previous white move in the special case where a game starts from a position where Black is the active player.

There are no other cases where move number indications appear in PGN export format.

8.2.3: Movetext SAN (Standard Algebraic Notation)

SAN (Standard Algebraic Notation) is a representation standard for chess moves using the ASCII Latin alphabet.

Examples of SAN recorded games are found throughout most modern chess publications. SAN as presented in this document uses English language single character abbreviations for chess pieces, although this is easily changed in the source. English is chosen over other languages because it appears to be the most widely recognized.

An alternative to SAN is FAN (Figurine Algebraic Notation). FAN uses miniature piece icons instead of single letter piece abbreviations. The two notations are otherwise identical.

8.2.3.1: Square identification

SAN identifies each of the sixty four squares on the chessboard with a unique two character name. The first character of a square identifier is the file of the square; a file is a column of eight squares designated by a single lower case letter from "a" (leftmost or queenside) up to and including "h" (rightmost or kingside). The second character of a square identifier is the rank of the square; a rank is a row of eight squares designated by a single digit from "1" (bottom side [White's first rank]) up to and including "8" (top side [Black's first rank]). The initial squares of some pieces are: white queen rook at a1, white king at e1, black queen knight pawn at b7, and black king rook at h8.

8.2.3.2: Piece identification

SAN identifies each piece by a single upper case letter. The standard English values: pawn = "P", knight = "N", bishop = "B", rook = "R", queen = "Q", and king = "K".

The letter code for a pawn is not used for SAN moves in PGN export format movetext. However, some PGN import software disambiguation code may allow for the appearance of pawn letter codes. Also, pawn and other piece letter codes are needed for use in some tag pair and annotation constructs.

It is admittedly a bit chauvinistic to select English piece letters over those from other languages. There is a slight justification in that English is a de facto universal second language among most chessplayers and program users. It is probably the best that can be done for now. A later section of this document gives alternative piece letters, but these should be used only for local presentation software and not for archival storage or for dynamic interchange among programs.

8.2.3.3: Basic SAN move construction

A basic SAN move is given by listing the moving piece letter (omitted for pawns) followed by the destination square. Capture moves are denoted by the lower case letter "x" immediately prior to the destination square; pawn captures include the file letter of the originating square of the capturing pawn immediately prior to the "x" character.

SAN kingside castling is indicated by the sequence "O-O"; queenside castling is indicated by the sequence "O-O-O". Note that the upper case letter "O" is used, not the digit zero. The use of a zero character is not only incompatible with traditional text practices, but it can also confuse parsing algorithms which also have to understand about move numbers and game termination markers. Also note that the use of the letter "O" is consistent with the practice of having all chess move symbols start with a letter; also, it follows the convention that all non-pwn move symbols start with an upper case letter.

En passant captures do not have any special notation; they are formed as if the captured pawn were on the capturing pawn's destination square. Pawn promotions are denoted by the equal sign "=" immediately following the destination square with a promoted piece letter (indicating one of knight, bishop, rook, or queen) immediately following the equal sign. As above, the piece letter is in upper case.

8.2.3.4: Disambiguation

In the case of ambiguities (multiple pieces of the same type moving to the same square), the first appropriate disambiguating step of the three following steps is taken:

First, if the moving pieces can be distinguished by their originating files, the originating file letter of the moving piece is inserted immediately after the moving piece letter.

Second (when the first step fails), if the moving pieces can be distinguished by their originating ranks, the originating rank digit of the moving piece is inserted immediately after the moving piece letter.

Third (when both the first and the second steps fail), the two character square coordinate of the originating square of the moving piece is inserted immediately after the moving piece letter.

Note that the above disambiguation is needed only to distinguish among moves of the same piece type to the same square; it is not used to distinguish among attacks of the same piece type to the same square. An example of this would be a position with two white knights, one on square c3 and one on square g1 and a vacant square e2 with White to move. Both knights attack square e2, and if both could legally move there, then a file disambiguation is needed; the (nonchecking) knight moves would be "Nce2" and "Nge2". However, if the white king were at square e1 and a black bishop were at square b4 with a vacant square d2 (thus an absolute pin of the white knight at square c3), then only one white knight (the one at square g1) could move to square e2: "Ne2".

8.2.3.5: Check and checkmate indication characters

If the move is a checking move, the plus sign "+" is appended as a suffix to the basic SAN move notation; if the move is a checkmating move, the octothorpe sign "#" is appended instead.

Neither the appearance nor the absence of either a check or checkmating indicator is used for disambiguation purposes. This means that if two (or more) pieces of the same type can move to the same square the differences in checking status of the moves does not allieviate the need for the standard rank and file disabiguation described above. (Note that a difference in checking status for the above may occur only in the case of a discovered check.)

Neither the checking or checkmating indicators are considered annotation as they do not communicate subjective information. Therefore, they are qualitatively different from move suffix annotations like "!" and "?". Subjective move annotations are handled using Numeric Annotation Glyphs as described in a later section of this document.

There are no special markings used for double checks or discovered checks.

There are no special markings used for drawing moves.

8.2.3.6: SAN move length

SAN moves can be as short as two characters (e.g., "d4"), or as long as seven characters (e.g., "Qa6xb7#", "fxg1=Q+"). The average SAN move length seen in realistic games is probably just fractionally longer than three characters. If the SAN rules seem complicated, be assured that the earlier notation systems of LEN (Long English Notation) and EDN (English Descriptive Notation) are much more complex, and that LAN (Long Algebraic Notation, the predecessor of SAN) is unnecessarily bulky.

8.2.3.7: Import and export SAN

PGN export format always uses the above canonical SAN to represent moves in the movetext section of a PGN game. Import format is somewhat more relaxed and it makes allowances for moves that do not conform exactly to the canonical format. However, these allowances may differ among different PGN reader programs. Only

data appearing in export format is in all cases guaranteed to be importable into all PGN readers.

There are a number of suggested guidelines for use with implementing PGN reader software for permitting non-canonical SAN move representation. The idea is to have a PGN reader apply various transformations to attempt to discover the move that is represented by non-canonical input. Some suggested transformations include: letter case remapping, capture indicator insertion, check indicator insertion, and checkmate indicator insertion.

8.2.3.8: SAN move suffix annotations

Import format PGN allows for the use of traditional suffix annotations for moves. There are exactly six such annotations available: "!", "?", "!!", "!?", "?!", and "??". At most one such suffix annotation may appear per move, and if present, it is always the last part of the move symbol.

When exported, a move suffix annotation is translated into the corresponding Numeric Annotation Glyph as described in a later section of this document. For example, if the single move symbol "Qxa8?" appears in an import format PGN movetext, it would be replaced with the two adjacent symbols "Qxa8 $2".

8.2.4: Movetext NAG (Numeric Annotation Glyph)

An NAG (Numeric Annotation Glyph) is a movetext element that is used to indicate a simple annotation in a language independent manner. An NAG is formed from a dollar sign ("$") with a non-negative decimal integer suffix. The non-negative integer must be from zero to 255 in value.

8.2.5: Movetext RAV (Recursive Annotation Variation)

An RAV (Recursive Annotation Variation) is a sequence of movetext containing one or more moves enclosed in parentheses. An RAV is used to represent an alternative variation. The alternate move sequence given by an RAV is one that

may be legally played by first unplaying the move that appears immediately prior to the RAV. Because the RAV is a recursive construct, it may be nested.

*** The specification for import/export representation of RAV elements needs further development.

8.2.6: Game Termination Markers

Each movetext section has exactly one game termination marker; the marker always occurs as the last element in the movetext. The game termination marker is a symbol that is one of the following four values: "1-0" (White wins), "0-1" (Black wins), "1/2-1/2" (drawn game), and "*" (game in progress, result unknown, or game abandoned). Note that the digit zero is used in the above; not the upper case letter "O". The game termination marker appearing in the movetext of a game must match the value of the game's Result tag pair. (While the marker appears as a string in the Result tag, it appears as a symbol without quotes in the movetext.)

9: Supplemental tag names

The following tag names and their associated semantics are recommended for use for information not contained in the Seven Tag Roster.

9.1: Player related information

Note that if there is more than one player field in an instance of a player (White or Black) tag, then there will be corresponding multiple fields in any of the following tags. For example, if the White tag has the three field value "Jones:Smith:Zacharias" (a consultation game), then the WhiteTitle tag could have a value of "IM:-:GM" if Jones was an International Master, Smith was untitled, and Zacharias was a Grandmaster.

9.1.1: Tags: WhiteTitle, BlackTitle

These use string values such as "FM", "IM", and "GM"; these tags are used only for the standard abbreviations for FIDE titles. A value of "-" is used for an untitled player.

9.1.2: Tags: WhiteElo, BlackElo

These tags use integer values; these are used for FIDE Elo ratings. A value of "-" is used for an unrated player.

9.1.3: Tags: WhiteUSCF, BlackUSCF

These tags use integer values; these are used for USCF (United States Chess Federation) ratings. Similar tag names can be constructed for other rating agencies.

9.1.4: Tags: WhiteNA, BlackNA

These tags use string values; these are the e-mail or network addresses of the players. A value of "-" is used for a player without an electronic address.

9.1.5: Tags: WhiteType, BlackType

These tags use string values; these describe the player types. The value "human" should be used for a person while the value "program" should be used for algorithmic (computer) players.

9.2: Event related information

The following tags are used for providing additional information about the event.

9.2.1: Tag: EventDate

This uses a date value, similar to the Date tag field, that gives the starting date of the Event.

9.2.2: Tag: EventSponsor

This uses a string value giving the name of the sponsor of the event.

9.2.3: Tag: Section

This uses a string; this is used for the playing section of a tournament (e.g., "Open" or "Reserve").

9.2.4: Tag: Stage

This uses a string; this is used for the stage of a multistage event (e.g., "Preliminary" or "Semifinal").

9.2.5: Tag: Board

This uses an integer; this identifies the board number in a team event and also in a simultaneous exhibition.

9.3: Opening information (locale specific)

The following tag pairs are used for traditional opening names. The associated tag values will vary according to the local language in use.

9.3.1: Tag: Opening

This uses a string; this is used for the traditional opening name. This will vary by locale. This tag pair is associated with the use of the EPD opcode "v0" described in a later section of this document.

9.3.2: Tag: Variation

This uses a string; this is used to further refine the Opening tag. This will vary by locale. This tag pair is associated with the use of the EPD opcode "v1" described in a later section of this document.

9.3.3: Tag: SubVariation

This uses a string; this is used to further refine the Variation tag. This will vary by locale. This tag pair is associated with the use of the EPD opcode "v2" described in a later section of this document.

9.4: Opening information (third party vendors)

The following tag pairs are used for representing opening identification according to various third party vendors and organizations. References to these organizations does not imply any endorsement of them or any endorsement by them.

9.4.1: Tag: ECO

This uses a string of either the form "XDD" or the form "XDD/DD" where the "X" is a letter from "A" to "E" and the "D" positions are digits; this is used for an opening designation from the five volume *Encyclopedia of Chess Openings*. This tag pair is associated with the use of the EPD opcode "eco" described in a later section of this document.

9.4.2: Tag: NIC

This uses a string; this is used for an opening designation from the _New in Chess_ database. This tag pair is associated with the use of the EPD opcode "nic" described in a later section of this document.

9.5: Time and date related information

The following tags assist with further refinement of the time and data information associated with a game.

9.5.1: Tag: Time

This uses a time-of-day value in the form "HH:MM:SS"; similar to the Date tag except that it denotes the local clock time (hours, minutes, and seconds) of the start of the game. Note that colons, not periods, are used for field separators for the Time tag value. The value is taken from the local time corresponding to the location given in the Site tag pair.

9.5.2: Tag: UTCTime

This tag is similar to the Time tag except that the time is given according to the Universal Coordinated Time standard.

9.5.3: Tag:; UTCDate

This tag is similar to the Date tag except that the date is given according to the Universal Coordinated Time standard.

9.6: Time control

The follwing tag is used to help describe the time control used with the game.

9.6.1: Tag: TimeControl

This uses a list of one or more time control fields. Each field contains a descriptor for each time control period; if more than one descriptor is present then they are separated by the colon character (":"). The descriptors appear in the order in which they are used in the game. The last field appearing is considered to be implicitly repeated for further control periods as needed.

There are six kinds of TimeControl fields.

The first kind is a single question mark ("?") which means that the time control mode is unknown. When used, it is usually the only descriptor present.

The second kind is a single hyphen ("-") which means that there was no time control mode in use. When used, it is usually the only descriptor present.

The third Time control field kind is formed as two positive integers separated by a solidus ("/") character. The first integer is the number of moves in the period and the second is the number of seconds in the period. Thus, a time control period of 40 moves in 2 1/2 hours would be represented as "40/9000".

The fourth TimeControl field kind is used for a "sudden death" control period. It should only be used for the last descriptor in a TimeControl tag value. It is sometimes the only descriptor present. The format consists of a single integer that gives the number of seconds in the period. Thus, a blitz game would be represented with a TimeControl tag value of "300".

The fifth TimeControl field kind is used for an "incremental" control period. It should only be used for the last descriptor in a TimeControl tag value and is usually the only descriptor in the value. The format consists of two positive integers separated by a plus sign ("+") character. The first integer gives the minimum number of seconds allocated for the period and the second integer gives the number of extra seconds added after each move is made. So,

an incremental time control of 90 minutes plus one extra minute per move would be given by "4500+60" in the TimeControl tag value.

The sixth TimeControl field kind is used for a "sandclock" or "hourglass" control period. It should only be used for the last descriptor in a TimeControl tag value and is usually the only descriptor in the value. The format consists of an asterisk ("*") immediately followed by a positive integer. The integer gives the total number of seconds in the sandclock period. The time control is implemented as if a sandclock were set at the start of the period with an equal amount of sand in each of the two chambers and the players invert the sandclock after each move with a time forfeit indicated by an empty upper chamber. Electronic implementation of a physical sandclock may be used. An example sandclock specification for a common three minute egg timer sandclock would have a tag value of "*180".

Additional TimeControl field kinds will be defined as necessary.

9.7: Alternative starting positions

There are two tags defined for assistance with describing games that did not start from the usual initial array.

9.7.1: Tag: SetUp

This tag takes an integer that denotes the "set-up" status of the game. A value of "0" indicates that the game has started from the usual initial array. A value of "1" indicates that the game started from a set-up position; this position is given in the "FEN" tag pair. This tag must appear for a game starting with a set-up position. If it appears with a tag value of "1", a FEN tag pair must also appear.

9.7.2: Tag: FEN

This tag uses a string that gives the Forsyth-Edwards Notation for the starting position used in the game. FEN is described in a later section of this document. If a SetUp tag appears with a tag value of "1", the FEN tag pair is also required.

9.8: Game conclusion

There is a single tag that discusses the conclusion of the game.

9.8.1: Tag: Termination

This takes a string that describes the reason for the conclusion of the game. While the Result tag gives the result of the game, it does not provide any extra information and so the Termination tag is defined for this purpose.

Strings that may appear as Termination tag values:

* "abandoned": abandoned game.

* "adjudication": result due to third party adjudication process.

* "death": losing player called to greater things, one hopes.

* "emergency": game concluded due to unforeseen circumstances.

* "normal": game terminated in a normal fashion.

* "rules infraction": administrative forfeit due to losing player's failure to observe either the Laws of Chess or the event regulations.

* "time forfeit": loss due to losing player's failure to meet time control requirements.

* "unterminated": game not terminated.

9.9: Miscellaneous

These are tags that can be briefly described and that doon't fit well inother sections.

9.9.1: Tag: Annotator

This tag uses a name or names in the format of the player name tags; this identifies the annotator or annotators of the game.

9.9.2: Tag: Mode

This uses a string that gives the playing mode of the game. Examples: "OTB" (over the board), "PM" (paper mail), "EM" (electronic mail), "ICS" (Internet Chess Server), and "TC" (general telecommunication).

9.9.3: Tag: PlyCount

This tag takes a single integer that gives the number of ply (moves) in the game.

10: Numeric Annotation Glyphs

NAG zero is used for a null annotation; it is provided for the convenience of software designers as a placeholder value and should probably not be used in external PGN data.

NAGs with values from 1 to 9 annotate the move just played.

NAGs with values from 10 to 135 modify the current position.

NAGs with values from 136 to 139 describe time pressure.

Other NAG values are reserved for future definition.

Note: the number assignments listed below should be considered preliminary in nature; they are likely to be changed as a result of reviewer feedback.

NAG	Interpretation
0	null annotation
1	good move (traditional "!")
2	poor move (traditional "?")
3	very good move (traditional "!!")
4	very poor move (traditional "??")
5	speculative move (traditional "!?")
6	questionable move (traditional "?!")
7	forced move (all others lose quickly)
8	singular move (no reasonable alternatives)
9	worst move
10	drawish position
11	equal chances, quiet position
12	equal chances, active position
13	unclear position
14	White has a slight advantage
15	Black has a slight advantage
16	White has a moderate advantage
17	Black has a moderate advantage
18	White has a decisive advantage
19	Black has a decisive advantage
20	White has a crushing advantage (Black should resign)

```
21    Black has a crushing advantage (White should resign)
22    White is in zugzwang
23    Black is in zugzwang
24    White has a slight space advantage
25    Black has a slight space advantage
26    White has a moderate space advantage
27    Black has a moderate space advantage
28    White has a decisive space advantage
29    Black has a decisive space advantage
30    White has a slight time (development) advantage
31    Black has a slight time (development) advantage
32    White has a moderate time (development) advantage
33    Black has a moderate time (development) advantage
34    White has a decisive time (development) advantage
35    Black has a decisive time (development) advantage
36    White has the initiative
37    Black has the initiative
38    White has a lasting initiative
39    Black has a lasting initiative
40    White has the attack
41    Black has the attack
42    White has insufficient compensation for material deficit
43    Black has insufficient compensation for material deficit
44    White has sufficient compensation for material deficit
45    Black has sufficient compensation for material deficit
46    White has more than adequate compensation for material deficit
47    Black has more than adequate compensation for material deficit
48    White has a slight center control advantage
49    Black has a slight center control advantage
50    White has a moderate center control advantage
51    Black has a moderate center control advantage
52    White has a decisive center control advantage
53    Black has a decisive center control advantage
54    White has a slight kingside control advantage
55    Black has a slight kingside control advantage
56    White has a moderate kingside control advantage
57    Black has a moderate kingside control advantage
58    White has a decisive kingside control advantage
59    Black has a decisive kingside control advantage
```

```
60    White has a slight queenside control advantage
61    Black has a slight queenside control advantage
62    White has a moderate queenside control advantage
63    Black has a moderate queenside control advantage
64    White has a decisive queenside control advantage
65    Black has a decisive queenside control advantage
66    White has a vulnerable first rank
67    Black has a vulnerable first rank
68    White has a well protected first rank
69    Black has a well protected first rank
70    White has a poorly protected king
71    Black has a poorly protected king
72    White has a well protected king
73    Black has a well protected king
74    White has a poorly placed king
75    Black has a poorly placed king
76    White has a well placed king
77    Black has a well placed king
78    White has a very weak pawn structure
79    Black has a very weak pawn structure
80    White has a moderately weak pawn structure
81    Black has a moderately weak pawn structure
82    White has a moderately strong pawn structure
83    Black has a moderately strong pawn structure
84    White has a very strong pawn structure
85    Black has a very strong pawn structure
86    White has poor knight placement
87    Black has poor knight placement
88    White has good knight placement
89    Black has good knight placement
90    White has poor bishop placement
91    Black has poor bishop placement
92    White has good bishop placement
93    Black has good bishop placement
84    White has poor rook placement
85    Black has poor rook placement
86    White has good rook placement
87    Black has good rook placement
98    White has poor queen placement
```

```
 99    Black has poor queen placement
100    White has good queen placement
101    Black has good queen placement
102    White has poor piece coordination
103    Black has poor piece coordination
104    White has good piece coordination
105    Black has good piece coordination
106    White has played the opening very poorly
107    Black has played the opening very poorly
108    White has played the opening poorly
109    Black has played the opening poorly
110    White has played the opening well
111    Black has played the opening well
112    White has played the opening very well
113    Black has played the opening very well
114    White has played the middlegame very poorly
115    Black has played the middlegame very poorly
116    White has played the middlegame poorly
117    Black has played the middlegame poorly
118    White has played the middlegame well
119    Black has played the middlegame well
120    White has played the middlegame very well
121    Black has played the middlegame very well
122    White has played the ending very poorly
123    Black has played the ending very poorly
124    White has played the ending poorly
125    Black has played the ending poorly
126    White has played the ending well
127    Black has played the ending well
128    White has played the ending very well
129    Black has played the ending very well
130    White has slight counterplay
131    Black has slight counterplay
132    White has moderate counterplay
133    Black has moderate counterplay
134    White has decisive counterplay
135    Black has decisive counterplay
136    White has moderate time control pressure
137    Black has moderate time control pressure
```

```
138    White has severe time control pressure
139    Black has severe time control pressure
```

```
11: File names and directories
```

```
File names chosen for PGN data should be both informative and portable.  The
directory names and arrangements should also be chosen for the same reasons and
also for ease of navigation.
```

```
Some of suggested file and directory names may be difficult or impossible to
represent on certain computing systems.  Use of appropriate conversion customs
is encouraged.
```

```
11.1: File name suffix for PGN data
```

```
The use of the file suffix ".pgn" is encouraged for ASCII text files containing
PGN data.
```

```
11.2: File name formation for PGN data for a specific player
```

```
PGN games for a specific player should have a file name consisting of the
player's last name followed by the ".pgn" suffix.
```

```
11.3: File name formation for PGN data for a specific event
```

```
PGN games for a specific event should have a file name consisting of the
event's name followed by the ".pgn" suffix.
```

```
11.4: File name formation for PGN data for chronologically ordered games
```

PGN data files used for chronologically ordered (oldest first) archives use
date information as file name root strings. A file containing all the PGN
games for a given year would have an eight character name in the format
"YYYY.pgn". A file containing PGN data for a given month would have a ten
character name in the format "YYYYMM.pgn". Finally, a file for PGN games for a
single day would have a twelve character name in the format "YYYYMMDD.pgn".
Large files are split into smaller files as needed.

As game files are commonly arranged by chronological order, games with missing
or incomplete Date tag pair data are to be avoided. Any question mark
characters in a Date tag value will be treated as zero digits for collation
within a file and also for file naming.

Large quantities of PGN data arranged by chronological order should be
organized into hierarchical directories. A directory containing all PGN data
for a given year would have a four character name in the format "YYYY";
directories containing PGN files for a given month would have a six character
name in the format "YYYYMM".

11.5: Suggested directory tree organization

A suggested directory arrangement for ftp sites and CD-ROM distributions:

* PGN: master directory of the PGN subtree (pub/chess/Game-Databases/PGN)

* PGN/Events: directory of PGN files, each for a specific event

* PGN/Events/News: news and status of the event collection

* PGN/Events/ReadMe: brief description of the local directory contents

* PGN/MGR: directory of the Master Games Repository subtree

* PGN/MGR/News: news and status of the entire PGN/MGR subtree

* PGN/MGR/ReadMe: brief description of the local directory contents

* PGN/MGR/YYYY: directory of games or subtrees for the year YYYY

* PGN/MGR/YYYY/ReadMe: description of local directory for year YYYY

* PGN/MGR/YYYY/News: news and status for year YYYY data

* PGN/News: news and status of the entire PGN subtree

* PGN/Players: directory of PGN files, each for a specific player

* PGN/Players/News: news and status of the player collection

* PGN/Players/ReadMe: brief description of the local directory contents

* PGN/ReadMe: brief description of the local directory contents

* PGN/Standard: the PGN standard (this document)

* PGN/Tools: software utilities that access PGN data

12: PGN collating sequence

There is a standard sorting order for PGN games within a file. This collation
is based on eight keys; these are the seven tag values of the STR and also the
movetext itself.

The first (most important, primary key) is the Date tag. Earlier dated games appear prior to games played at a later date. This field is sorted by ascending numeric value first with the year, then the month, and finally the day of the month. Query characters used for unknown date digit values will be treated as zero digit characters for ordering comparison.

The second key is the Event tag. This is sorted in ascending ASCII order.

The third key is the Site tag. This is sorted in ascending ASCII order.

The fourth key is the Round tag. This is sorted in ascending numeric order based on the value of the integer used to denote the playing round. A query or hyphen used for the round is ordered before any integer value. A query character is ordered before a hyphen character.

The fifth key is the White tag. This is sorted in ascending ASCII order.

The sixth key is the Black tag. This is sorted in ascending ASCII order.

The seventh key is the Result tag. This is sorted in ascending ASCII order.

The eighth key is the movetext itself. This is sorted in ascending ASCII order with the entire text including spaces and newline characters.

13: PGN software

This section describes some PGN software that is either currently available or expected to be available in the near future. The entries are presented in rough chronological order of their being made known to the PGN standard coordinator. Authors of PGN capable software are encouraged to contact the coordinator (e-mail address listed near the start of this document) so that the information may be included here in this section.

In addition to the PGN standard, there are two more chess standards of interest to the chess software community. These are the FEN standard (Forsyth-Edwards Notation) for position notation and the EPD standard (Extended Position Description) for comprehensive position description for automated interprogram processing. These are described in a later section of this document.

Some PGN software is freeware and can be gotten from ftp sites and other sources. Other PGN software is payware and appears as part of commercial chessplaying programs and chess database managers. Those who are interested in the propagation of the PGN standard are encouraged to support manufacturers of chess software that use the standard. If a particular vendor does not offer PGN compatibility, it is likely that a few letters to them along with a copy of this specification may help them decide to include PGN support in their next release.

The staff at the University of Oklahoma at Norman (USA) have graciously provided an ftp site (chess.uoknor.edu) for the storage of chess related data and programs. Because file names change over time, those accessing the site are encouraged to first retrieve the file "pub/chess/ls-lR.gz" for a current listing. A scan of this listing will also help locate versions of PGN programs for machine types and operating systems other than those listed below. Further information about this archive can be gotten from its administrator, Chris Petroff (chris@uoknor.edu).

For European users, the kind staff at the University of Hamburg (Germany) have provided the ftp site ftp.math.uni-hamburg.de; this carries a daily mirror of the pub/chess directory at the chess.uoknor.edu site.

13.1: The SAN Kit

The "SAN Kit" is an ANSI C source chess programming toolkit available for free from the ftp site chess.uoknor.edu in the directory pub/chess/Unix as the file "SAN.tar.gz" (a gzip tar archive). This kit contains code for PGN import and export and can be used to "regularize" PGN data into reduced export format by use of its "tfgg" command. The SAN Kit also supports FEN I/O. Code from this

kit is freely redistributable for anyone as long as future distribution is unhindered for everyone. The SAN Kit is undergoing continuous development, although dates of future deliveries are quite difficult to predict and releases sometimes appear months apart. Suggestions and comments should be directed to its author, Steven J. Edwards (sje@world.std.com).

13.2: pgnRead

The program "pgnRead" runs under MS Windows 3.1 and provides an interactive graphical user interface for scanning PGN data files. This program includes a colorful figurine chessboard display and scrolling controls for game and game text selection. It is available from the chess.uoknor.edu ftp site in the pub/chess/DOS directory; several versions are available with names of the form "pgnrd**.exe"; the latest at this writing is "PGNRD130.EXE". Suggestions and comments should be directed to its author, Keith Fuller (keithfx@aol.com).

13.3: mail2pgn/GIICS

The program "mail2pgn" produces a PGN version of chess game data generated by the ICS (Internet Chess Server). It can be found at the chess.uoknor.edu ftp site in the pub/chess/DOS directory as the file "mail2pgn.zip" A C language version is in the directory pub/chess/Unix as the file "mail2pgn.c". Suggestions and comments should be directed to its author, John Aronson (aronson@helios.ece.arizona.edu). This code has been reportedly incorporated into the GIICS (Graphical Interface for the ICS); suggestions and comments should be directed to its author, Tony Acero (ace3@midway.uchicago.edu).

There is a report that mail2pgn has been superseded by the newer program "MV2PGN" described below.

13.4: XBoard

"XBoard" is a comprehensive chess utility running under the X Window System that provides a graphical user interface in a portable manner. A new version

```
now handles PGN data.  It is available from the chess.uoknor.edu ftp site in
the pub/chess/X directory as the file "xboard-3.0.pl9.tar.gz".  Suggestions and
comments should be directed to its author, Tim Mann (mann@src.dec.com).
```

```
13.5: cupgn
```

```
The program "cupgn" converts game data stored in the ChessBase format into PGN.
It is available from the chess.uoknor.edu ftp site in the
pub/chess/Game-Databases/CBUFF directory as the file "cupgn.tar.gz".  Another
version is in the directory pub/chess/DOS as the file "cupgn120.exe".
Suggestions and comments should be directed to its author, Anjo Anjewierden
(anjo@swi.psy.uva.nl).
```

```
13.6: Zarkov
```

```
The current version (3.0) of the commercial chessplaying program "Zarkov" can
read and write games using PGN.  This program can also use the EPD standard for
communication with other EPD capable programs.  Historically, Zarkov is the
very first program to use EPD.  Suggestions and comments should be directed to
its author, John Stanback (jhs@icbdfcs1.fc.hp.com).
```

```
A vendor for North America is:
```

```
    International Chess Enterprises
    P.O. Box 19457
    Seattle, WA 98109
    USA
    (800) 262-4277
```

```
A vendor for Europe is:
```

```
    Gambit-Soft
    Feckenhauser Strasse 27
    D-78628 Rottweil
```

```
    GERMANY
    49-741-21573
```

13.7: Chess Assistant

The upcoming version of the multifunction commercial database program "Chess Assistant" will be able to use the PGN standard as an import and export option. There is a report of a freeware program, "PGN2CA", that will convert PGN databases into Chess Assistant format. For more information, the contact is Victor Zakharov, one of the members of the Chess Assistant development team (VICTOR@ldis.cs.msu.su).

A vendor for North America is:

```
    International Chess Enterprises
    P.O. Box 19457
    Seattle, WA 98109
    USA
    (800) 262-4277
```

13.8: BOOKUP

The MS-DOS edition of the multifunction commercial program BOOKUP, version 8.1, is able to use the EPD standard for communication with other EPD capable programs. It may also be PGN capable as well.

The BOOKUP 8.1.1 Addenda notes dated 1993.12.17 provide comprehensive information on how to use EPD in conjunction with "analyst" programs such as Zarkov and HIARCS. Specifically, the search and evaluation abilities of an analyst program are combined with the information organization abilities of the BOOKUP database program to provide position scoring. This is done by first having BOOKUP export a database in EPD format, then having an analyst program annotate each EPD record with a numeric score, and then having BOOKUP import the changed EPD file. BOOKUP can then apply minimaxing to the imported

database; this results in scores from terminal positions being propagated back to earlier positions and even back to moves from the starting array.

For some reason, BOOKUP calls this process "backsolving", but it's really just standard minimaxing. In any case, it's a good example of how different programs from different authors performing different types of tasks can be integrated by use of a common, non-proprietary standard. This allows for a new set of powerful features that are beyond the capabilities of any one of the individual component programs.

BOOKUP allows for some customizing of EPD actions. One such customization is to require the positional evaluations to follow the EPD standard; this means that the score is always given from the viewpoint of the active player. This is explained more fully in the section on the "ce" (centipawn evaluation) opcode in the EPD description in a later section of this document. To ensure that BOOKUP handles the centipawn evaluations in the "right" way, the EPD setting "Positive for White" must be set to "N". This makes BOOKUP work correctly with Zarkov and with all other programs that use the "right" centipawn evaluation convention. There is an apparent problem with HIARCS that requires this option to be set to "Y"; but this really means that, if true, HIARCS needs to be adjusted to use the "right" centipawn evaluation convention.

A vendor in North America is:

    BOOKUP
    2763 Kensington Place West
    Columbus, OH 43202
    USA
    (800) 949-5445
    (614) 263-7219

13.9: HIARCS

The current version (2.1) of the commercial chessplaying program "HIARCS" is able to use the EPD standard for communication with other EPD capable programs.

It may also be PGN capable as well. More details will appear here as they become available.

A vendor in North America is:

    HIARCS
    c/o BOOKUP
    2763 Kensington Place West
    Columbus, OH 43202
    USA
    (800) 949-5445
    (614) 263-7219

13.10: Deja Vu

The chess database "Deja Vu" from ChessWorks is a PGN compatible collection of over 300,000 games. It is available only on CD-ROM and is scheduled for release in 1994.05 with periodic revisions thereafter. The introductory price is US$329. For further information, the authors are John Crayton and Eric Schiller and they can be contacted via e-mail (chesswks@netcom.com).

13.11: MV2PGN

The program "MV2PGN" can be used to convert game data generated by both current and older versions of the GIICS (Graphical Interface - Internet Chess Server). The program is included in the self extracting archive available from chess.uoknor.edu in the directory pub/chess/DOS as the file "ics2pgn.exe". Source code is also included. This program is reported to supersede the older "mail2pgn" and was needed due to a change in ICS recording format in late 1993. For further information about MV2PGN, the contact person is Gary Bastin (gbastin@x102a.ess.harris.com).

13.12: The Hansen utilities (cb2pgn, nic2pgn, pgn2cb, pgn2nic)

The Hansen utilities are used to convert among various chess data representation formats. The PGN related programs include: "cb2pgn.exe" (convert ChessBase to PGN), "nic2pgn.exe" (convert NIC to PGN), "pgn2cb.exe" (convert PGN to ChessBase), and "pgn2nic.exe" (convert PGN to NIC).

The ChessBase related utilities (cb2pgn/pgn2cb) are found at chess.uoknor.edu in the pub/chess/Game-Databases/ChessBase directory.

The NIC related utilities (nic2pgn/pgn2nic) are found at chess.uoknor.edu in the pub/chess/Game-Databases/NIC directory.

For further information about the Hansen utilities, the contact person is the author, Carsten Hansen (ch0506@hdc.hha.dk).

13.13: Slappy the Database

"Slappy the Database" is a commercial chess database and translation program scheduled for release no sooner than late 1994. It is a low cost utility with a simple character interface intended for those who want a supported product but who do not need (or cannot afford) a comprehensive, feature-laden program with a graphical user interface. Slappy's two most important features are its batch processing ability and its full implementation of each and every standard described in this document. Versions of Slappy the Database will be provided for various platforms including: Intel 386/486 Unix, Apple Macintosh, and MS-DOS.

Slappy may also be useful to those who have a full feature program who also need to run time consuming chess database tasks on a spare computer.

Suggestions and comments should be directed to its author, Steven J. Edwards (sje@world.std.com). More details will appear here as they become available.

13.14: CBASCII

"CBASCII" is a general utility for converting chess data between ChessBase format and ASCII representations. It has PGN capability, and it is available from the chess.uoknor.edu ftp site in the pub/chess/DOS directory as the file "cba1_2.zip". The contact person is the program's author, Andy Duplain (duplain@btcs.bt.co.uk).

13.15: ZZZZZZ

"ZZZZZZ" is a chessplaying program, complete with source, that also includes some database functions. A recent version is reported to have both PGN and EPD capabilities. It is available from the chess.uoknor.edu ftp site in the pub/chess/Unix directory as the file "zzzzzz-3.2b1.tar.gz". The contact person is its author, Gijsbert Wiesenecker (wiesenecker@sara.nl).

13.16: icsconv

The program "icsconv" can be used to convert Internet Chess Server games, both old and new format, to PGN. It is available from the chess.uoknor.edu site in the pub/chess/Game-Databases/PGN/Tools directory as the file "icsconv.exe". The contact person is the author, Kevin Nomura (chow@netcom.com).

13.17: CHESSOP (CHESSOPN/CHESSOPG)

CHESSOP is an openings database and viewing tool with support for reading PGN games. It runs under MS-DOS and displays positions rather than games. For each position, both good and bad moves are listed with appropriate annotation. Transpositions are handled as well. The distributed database contains over 100,000 positions covering all the common openings. Users can feed in their own PGN data as well. CHESSOP takes 3 Mbyte of hard disk, costs US$39 and can be obtained from:

    CHESSX Software
    12 Bluebell Close

```
    Glenmore Park
    AUSTRALIA 2745.
```

```
The ideas behind CHESSOP can be seen in CHESSOPN (alias CHESSOPG), a free
version on the ICS server which has a reduced openings database (25,000
positions) and no PGN or transposition support but is otherwise the same as
CHESSOP.  (These are the files "chessopg.zip" in the directory pub/chess/DOS at
the chess.uoknor.edu ftp site.)
```

```
13.18: CAT2PGN
```

```
The program "CAT2PGN" is a utility that translates data from the format used by
Chess Assistant into PGN.  It is available from the chess.uoknor.edu ftp site.
The contact person for CAT2PGN is its author, David Myers
(myers@frodo.biochem.duke.edu).
```

```
13.19: pgn2opg
```

```
The utility "pgn2opg" can be used to convert PGN files into a text format used
by the "CHESSOPG" program mentioned above.  Although it does not perform any
semantic analysis on PGN input, it has been demonstrated to handle known
correct PGN input properly.  The file can be found in the pub/chess/PGN/Tools
directory at the chess.uoknor.edu ftp site.  For more information, the author
is David Barnes (djb@ukc.ac.uk).
```

```
14: PGN data archives
```

```
The primary PGN data archive repository is located at the ftp site
chess.uoknor.edu as the directory "pub/chess/Game-Databases/PGN".  It is
organized according to the description given in section C.5 of this document.
The European site ftp.math.uni-hamburg.de is also reported to carry a regularly
updated copy of the repository.
```

```
15: International Olympic Committee country codes
```

International Olympic Committee country codes are employed for Site nation information because of their traditional use with the reporting of international sporting events. Due to changes in geography and linguistic custom, some of the following may be incorrect or outdated. Corrections and extensions should be sent via e-mail to the PGN coordinator whose address listed near the start of this document.

AFG: Afghanistan
AIR: Aboard aircraft
ALB: Albania
ALG: Algeria
AND: Andorra
ANG: Angola
ANT: Antigua
ARG: Argentina
ARM: Armenia
ATA: Antarctica
AUS: Australia
AZB: Azerbaijan
BAN: Bangladesh
BAR: Bahrain
BHM: Bahamas
BEL: Belgium
BER: Bermuda
BIH: Bosnia and Herzegovina
BLA: Belarus
BLG: Bulgaria
BLZ: Belize
BOL: Bolivia
BRB: Barbados
BRS: Brazil
BRU: Brunei
BSW: Botswana
CAN: Canada
CHI: Chile
COL: Columbia
CRA: Costa Rica

```
CRO: Croatia
CSR: Czechoslovakia
CUB: Cuba
CYP: Cyprus
DEN: Denmark
DOM: Dominican Republic
ECU: Ecuador
EGY: Egypt
ENG: England
ESP: Spain
EST: Estonia
FAI: Faroe Islands
FIJ: Fiji
FIN: Finland
FRA: France
GAM: Gambia
GCI: Guernsey-Jersey
GEO: Georgia
GER: Germany
GHA: Ghana
GRC: Greece
GUA: Guatemala
GUY: Guyana
HAI: Haiti
HKG: Hong Kong
HON: Honduras
HUN: Hungary
IND: India
IRL: Ireland
IRN: Iran
IRQ: Iraq
ISD: Iceland
ISR: Israel
ITA: Italy
IVO: Ivory Coast
JAM: Jamaica
JAP: Japan
JRD: Jordan
JUG: Yugoslavia
```

```
KAZ: Kazakhstan
KEN: Kenya
KIR: Kyrgyzstan
KUW: Kuwait
LAT: Latvia
LEB: Lebanon
LIB: Libya
LIC: Liechtenstein
LTU: Lithuania
LUX: Luxembourg
MAL: Malaysia
MAU: Mauritania
MEX: Mexico
MLI: Mali
MLT: Malta
MNC: Monaco
MOL: Moldova
MON: Mongolia
MOZ: Mozambique
MRC: Morocco
MRT: Mauritius
MYN: Myanmar
NCG: Nicaragua
NET: The Internet
NIG: Nigeria
NLA: Netherlands Antilles
NLD: Netherlands
NOR: Norway
NZD: New Zealand
OST: Austria
PAK: Pakistan
PAL: Palestine
PAN: Panama
PAR: Paraguay
PER: Peru
PHI: Philippines
PNG: Papua New Guinea
POL: Poland
POR: Portugal
```

```
PRC: People's Republic of China
PRO: Puerto Rico
QTR: Qatar
RIN: Indonesia
ROM: Romania
RUS: Russia
SAF: South Africa
SAL: El Salvador
SCO: Scotland
SEA: At Sea
SEN: Senegal
SEY: Seychelles
SIP: Singapore
SLV: Slovenia
SMA: San Marino
SPC: Aboard spacecraft
SRI: Sri Lanka
SUD: Sudan
SUR: Surinam
SVE: Sweden
SWZ: Switzerland
SYR: Syria
TAI: Thailand
TMT: Turkmenistan
TRK: Turkey
TTO: Trinidad and Tobago
TUN: Tunisia
UAE: United Arab Emirates
UGA: Uganda
UKR: Ukraine
UNK: Unknown
URU: Uruguay
USA: United States of America
UZB: Uzbekistan
VEN: Venezuela
VGB: British Virgin Islands
VIE: Vietnam
VUS: U.S. Virgin Islands
WLS: Wales
```

YEM: Yemen
YUG: Yugoslavia
ZAM: Zambia
ZIM: Zimbabwe
ZRE: Zaire

16: Additional chess data standards

While PGN is used for game storage, there are other data representation standards for other chess related purposes. Two important standards are FEN and EPD, both described in this section.

16.1: FEN

FEN is "Forsyth-Edwards Notation"; it is a standard for describing chess positions using the ASCII character set.

A single FEN record uses one text line of variable length composed of six data fields. The first four fields of the FEN specification are the same as the first four fields of the EPD specification.

A text file composed exclusively of FEN data records should have a file name with the suffix ".fen".

16.1.1: History

FEN is based on a 19th century standard for position recording designed by the Scotsman David Forsyth, a newspaper journalist. The original Forsyth standard has been slightly extended for use with chess software by Steven Edwards with assistance from commentators on the Internet. This new standard, FEN, was first implemented in Edwards' SAN Kit.

16.1.2: Uses for a position notation

Having a standard position notation is particularly important for chess programmers as it allows them to share position databases. For example, there exist standard position notation databases with many of the classical benchmark tests for chessplaying programs, and by using a common position notation format many hours of tedious data entry can be saved. Additionally, a position notation can be useful for page layout programs and for confirming position status for e-mail competition.

Many interesting chess problem sets represented using FEN can be found at the chess.uoknor.edu ftp site in the directory pub/chess/SAN_testsuites.

16.1.3: Data fields

FEN specifies the piece placement, the active color, the castling availability, the en passant target square, the halfmove clock, and the fullmove number. These can all fit on a single text line in an easily read format. The length of a FEN position description varies somewhat according to the position. In some cases, the description could be eighty or more characters in length and so may not fit conveniently on some displays. However, these positions aren't too common.

A FEN description has six fields. Each field is composed only of non-blank printing ASCII characters. Adjacent fields are separated by a single ASCII space character.

16.1.3.1: 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 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. A solidus character "/" is
used to separate data of adjacent ranks.

16.1.3.2: 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.

16.1.3.3: Castling availability

The third field represents castling availability. This indicates potential
future castling that may of 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.

16.1.3.4: 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
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.

16.1.3.5: Halfmove clock

The fifth field is a nonnegative integer representing the halfmove clock. This
number is the count of halfmoves (or ply) since the last pawn advance or
capturing move. This value is used for the fifty move draw rule.

16.1.3.6: Fullmove number

The sixth and last field is a positive integer that gives the fullmove number.
This will have the value "1" for the first move of a game for both White and
Black. It is incremented by one immediately after each move by Black.

16.1.4: Examples

Here's the FEN for the starting position:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

And after the move 1. e4:

rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1

And then after 1. ... c5:

rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq c6 0 2

And then after 2. Nf3:

```
rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2
```

For two kings on their home squares and a white pawn on e2 (White to move) with thirty eight full moves played with five halfmoves since the last pawn move or capture:

```
4k3/8/8/8/8/8/4P3/4K3 w - - 5 39
```

16.2: EPD

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.

A single EPD uses one text line of variable length composed of four data field followed by zero or more operations. The four fields of the EPD specification are the same as the first four fields of the FEN specification.

A text file composed exclusively of EPD data records should have a file name with the suffix ".epd".

16.2.1: History

EPD is based in part on the earlier FEN standard; it 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 is in Stanback's master strength chessplaying program Zarkov.

16.2.2: Uses for an extended position notation

Like FEN, EPD can also be used for general position description. However, unlike FEN, EPD is designed to be expandable by the addition of new operations that provide new functionality as needs arise.

Many interesting chess problem sets represented using EPD can be found at the chess.uoknor.edu ftp site in the directory pub/chess/SAN_testsuites.

16.2.3: Data fields

EPD specifies 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 descriptions pass among programs only and these 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 1024 characters long.)

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

16.2.3.1: 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 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. A solidus character "/" is used to separate data of adjacent ranks.

16.2.3.2: 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.

16.2.3.3: 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.

16.2.3.4: 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 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.

16.2.4: 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.

16.2.4.1: General format

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.

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 character at each end. A string value must have less than 256 bytes of data.

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. All 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.

Some opcodes require one or more operands that are chess moves. These moves should be represented using SAN. If a different representation is used, there is no guarantee that the EPD will be read correctly during subsequent processing.

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 indeed discouraged.

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).

16.2.4.2: 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). These mnemonics will also all be at least two characters in length.

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.

16.2.5: Opcode list

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

16.2.5.1: 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. Note that the value may be quite large for some extended searches and so use of (at least) a long (four byte) representation is suggested.

16.2.5.2: 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. Note that the value may be quite large for some extended searches and so use of (at least) a long (four byte) representation is suggested.

16.2.5.3: 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 in the opinion of the EPD writer. Each operand is a SAN move; they appear in ASCII order.

16.2.5.4: 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. Each operand is a SAN move; they appear in ASCII order.

16.2.5.5: 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 n 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.

16.2.5.6: 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 -32767 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 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.

16.2.5.7: 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.

16.2.5.8: 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.

16.2.5.9: 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.

16.2.5.10: 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.

16.2.5.11: 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.

16.2.5.12: 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.

16.2.5.13: Opcode "fmvn": fullmove number

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

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.

16.2.5.14: 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.

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.

16.2.5.15: Opcode "id": position identification

The opcode "id" is used to provide a simple identifying 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".

16.2.5.16: 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 opening name) 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.

16.2.5.17: 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.

16.2.5.18: 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.

16.2.5.19: 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.

16.2.5.20: 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.

16.2.5.21: Opcode "resign": game resignation

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

16.2.5.22: 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.

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.

16.2.5.23: 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.

16.2.5.24: 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.

16.2.5.25: 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.

16.2.5.26: 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 n 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.

17: Alternative chesspiece identifier letters

English language piece names are used to define the letter set for identifying chesspieces in PGN movetext. However, authors of programs which are used only for local presentation or scanning of chess move data may find it convenient to use piece letter codes common in their locales. This is not a problem as long as PGN data that resides in archival storage or that is exchanged among programs still uses the SAN (English) piece letter codes: "PNBRQK".

For the above authors only, a list of alternative piece letter codes are provided:

Language	Piece letters (pawn knight bishop rook queen king)
Czech	P J S V D K
Danish	B S L T D K
Dutch	O P L T D K

```
English      P N B R Q K
Estonian     P R O V L K
Finnish      P R L T D K
French       P C F T D R
German       B S L T D K
Hungarian    G H F B V K
Icelandic    P R B H D K
Italian      P C A T D R
Norwegian    B S L T D K
Polish       P S G W H K
Portuguese   P C B T D R
Romanian     P C N T D R
Spanish      P C A T D R
Swedish      B S L T D K
```

18: Formal syntax

```
<PGN-database> ::= <PGN-game> <PGN-database>
                   <empty>
```

```
<PGN-game> ::= <tag-section> <movetext-section>
```

```
<tag-section> ::= <tag-pair> <tag-section>
                  <empty>
```

```
<tag-pair> ::= [ <tag-name> <tag-value> ]
```

```
<tag-name> ::= <identifier>
```

```
<tag-value> ::= <string>
```

```
<movetext-section> ::= <element-sequence> <game-termination>
```

```
<element-sequence> ::= <element> <element-sequence>
                       <recursive-variation> <element-sequence>
                       <empty>
```

```
<element> ::= <move-number-indication>
              <SAN-move>
              <numeric-annotation-glyph>
```

```
<recursive-variation> ::= ( <element-sequence> )
```

```
<game-termination> ::= 1-0
                       0-1
                       1/2-1/2
                       *
<empty> ::=
```

```
19: Canonical chess position hash coding
```

```
*** This section is under development.
```

```
20: Binary representation (PGC)
```

```
*** This section is under development.
```

```
The binary coded version of PGN is PGC (PGN Game Coding).  PGC is a binary
representation standard of PGN data designed for the dual goals of storage
efficiency and program I/O.  A file containing PGC data should have a name with
a suffix of ".pgc".
```

```
Unlike PGN text files that may have locale dependent representations for
newlines, PGC files have data that does not vary due to local processing
```

environment. This means that PGC files may be transferred among systems using general binary file methods.

PGC files should be used only when the use of PGN is impractical due to time and space resource constraints. As the general level of processing capabilities increases, the need for PGC over PGN will decrease. Therefore, implementors are encouraged not to use PGC as the default representation because it is much more difficult (than PGN) to understand without proper software.

PGC data is composed of a sequence of PGC records. Each record is composed of a sequence of one or more bytes. The first byte is the PGN record marker and it specifies the interpretation of the remaining portion of the record. This remaining portion is composed of zero or more PGN record items. Item types include move sequences, move sets, and character strings.

20.1: Bytes, words, and doublewords

At the lowest level, PGC binary data is organized as bytes, words (two contiguous bytes), and doublewords (four contiguous bytes). All eight bits of a byte are used. Longwords (eight contiguous bytes) are not used. Integer values are stored using two's complement representation. Integers may be signed or unsigned depending on context. Multibyte integers are stored in low-endian format with the least significant byte appearing first.

A one byte integer item is called "int-1". A two byte integer item is called "int-2". A four byte integer item is called "int-4".

Characters are stored as bytes using the ISO 8859/1 Latin-1 (ECMA-94) code set. There is no provision for other characters sets or representations.

20.2: Move ordinals

A chess move is represented using a move ordinal. This is a single unsigned byte quantity with values from zero to 255. A move ordinal is interpreted as an index into the list of legal moves from the current position. This list is constructed by generating the legal moves from the current position, assigning SAN ASCII strings to each move, and then sorting these strings in ascending order. Note that a seven bit ordinal, as used by some inferior representation systems, is insufficient as there are some positions that have more than 128 moves available.

Examples: From the initial position, there are twenty moves. Move ordinal 0 corresponds to the SAN move string "Na3"; move ordinal 1 corresponds to "Nc3", move ordinal 4 corresponds to "a3", and move ordinal 19 corresponds to "h4".

Moves can be organized into sequences and sets. A move sequence is an ordered list of moves that are played, one after another from first to last. A move set is a list of moves that are all playable from the current position.

Move sequence data is represented using a length header followed by move ordinal data. The length header is an unsigned integer that may be a byte or a word. The integer gives the number, possibly zero, of following move ordinal bytes. Most move sequences can be represented using just a byte header; these are called "mvseq-1" items. Move sequence data using a word header are called "mvseq-2" items.

Move set data is represented using a length header followed by move ordinal data. The length header is an unsigned integer that is a byte. The integer gives the number, possibly zero, of following move ordinal bytes. All move sets are be represented using just a byte header; these are called "mvset-1" items. (Note the implied restriction that a move set can only have a maximum of 255 of the possible 256 ordinals present at one time.)

20.3: String data

PGC string data is represented using a length header followed by bytes of character data. The length header is an unsigned integer that may be a byte, a word, or a doubleword. The integer gives the number, possibly zero, of following character bytes. Most strings can be represented using just a byte header; these are called "string-1" items. String data using a word header are called "string-2" items and string data using a doubleword header are called "string-4" items. No special ASCII NUL termination byte is required for PGC storage of a string as the length is explicitly given in the item header.

20.4: Marker codes

PGC marker codes are given in hexadecimal format. PGC marker code zero (marker 0x00) is the "noop" marker and carries no meaning. Each additional marker code defined appears in its own subsection below.

20.4.1: Marker 0x01: reduced export format single game

Marker 0x01 is used to indicate a single complete game in reduced export format. This refers to a game that has only the Seven Tag Roster data, played moves, and no annotations or comments. This record type is used as an alternative to the general game data begin/end record pairs described below. The general marker pair (0x05/0x06) is used to help represent game data that can't be adequately represented in reduced export format. There are eight items that follow marker 0x01 to form the "reduced export format single game" record. In order, these are:

1) string-1 (Event tag value)

2) string-1 (Site tag value)

3) string-1 (Date tag value)

4) string-1 (Round tag value)

5) string-1 (White tag value)

6) string-1 (Black tag value)

7) string-1 (Result tag value)

8) mvseq-2 (played moves)

20.4.2: Marker 0x02: tag pair

Marker 0x02 is used to indicate a single tag pair. There are two items that
follow marker 0x02 to form the "tag pair" record; in order these are:

1) string-1 (tag pair name)

2) string-1 (tag pair value)

20.4.3: Marker 0x03: short move sequence

Marker 0x03 is used to indicate a short move sequence. There is one item that
follows marker 0x03 to form the "short move sequence" record; this is:

1) mvseq-1 (played moves)

20.4.4: Marker 0x04: long move sequence

Marker 0x04 is used to indicate a long move sequence. There is one item that
follows marker 0x04 to form the "long move sequence" record; this is:

1) mvseq-2 (played moves)

20.4.5: Marker 0x05: general game data begin

Marker 0x05 is used to indicate the beginning of data for a game. It has no associated items; it is a complete record by itself. Instead, it marks the beginning of PGC records used to describe a game. All records up to the corresponding "general game data end" record are considered to be part of the same game. (PGC record type 0x01, "reduced export format single game", is not permitted to appear within a general game begin/end record pair. The general game construct is to be used as an alternative to record type 0x01 in those cases where the latter is too restrictive to contain the data for a game.)

20.4.6: Marker 0x06: general game data end

Marker 0x06 is used to indicate the end of data for a game. It has no associated items; it is a complete record by itself. Instead, it marks the end of PGC records used to describe a game. All records after the corresponding (and earlier appearing) "general game data begin" record are considered to be part of the same game.

20.4.7: Marker 0x07: simple-nag

Marker 0x07 is used to indicate the presence of a simple NAG (Numeric Annotation Glyph). This is an annotation marker that has only a short type identification and no operands. There is one item that follows marker 0x07 to form the "simple-nag" record; this is:

1) int-1 (unsigned NAG value, from 0 to 255)

20.4.8: Marker 0x08: rav-begin

Marker 0x08 is used to indicate the beginning of an RAV (Recursive Annotation Variation). It has no associated items; it is a complete record by itself. Instead, it marks the beginning of PGC records used to describe a recursive annotation. It is considered an opening bracket for a later rav-end record; the recursive annotation is completely described between the bracket pair. The rav-begin/data/rav-end structures can be nested.

20.4.9: Marker 0x09: rav-end

Marker 0x09 is used to indicate the end of an RAV (Recursive Annotation Variation). It has no associated items; it is a complete record by itself. Instead, it marks the end of PGC records used to describe a recursive annotation. It is considered a closing bracket for an earlier rav-begin record; the recursive annotation is completely described between the bracket pair. The rav-begin/data/rav-end structures can be nested.

20.4.10: Marker 0x0a: escape-string

Marker 0x0a is used to indicate the presence of an escape string. This is a string represented by the use of the percent sign ("%") escape mechanism in PGN. The data that is escaped is the sequence of characters immediately follwoing the percent sign up to but not including the terminating newline. As is the case with the PGN percent sign escape, the use of a PGC escape-string record is limited to use for non-archival data. There is one item that follows marker 0x0a to form the "escape-string" record; this is the string data being escaped:

1) string-2 (escaped string data)

21: E-mail correspondence usage

*** This section is under development.