The Variant Call Format Specification

VCFv4.4 and BCFv2.2

(Superseded by the VCF v4.5 specification)

12 Aug 2025

The master version of this document can be found at `https://github.com/samtools/hts-specs`.
This printing is version a90058f from that repository, last modified on the date shown above.

Contents

1 The VCF specification

VCF is a text file format (most likely stored in a compressed manner). It contains meta-information lines (prefixed with "##"), a header line (prefixed with "#"), and data lines each containing information about a position in the genome and genotype information on samples for each position (text fields separated by tabs). Zero length fields are not allowed, a dot (".") must be used instead. In order to ensure interoperability across platforms, VCF compliant implementations must support both LF ("\n") and CR+LF ("\r\n") newline conventions.

1.1 An example

```
##fileformat=VCFv4.4
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF ALT    QUAL FILTER INFO                             FORMAT      NA00001        NA00002        NA00003
20     14370   rs6054257 G   A      29   PASS   NS=3;DP=14;AF=0.5;DB;H2          GT:GQ:DP:HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20     17330   .         T   A      3    q10    NS=3;DP=11;AF=0.017             GT:GQ:DP:HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A   G,T    67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
20     1230237 .         T   .      47   PASS   NS=3;DP=13;AA=T                  GT:GQ:DP:HQ 0|0:54:7:56,60 0|0:48:4:51,51 0/0:61:2
20     1234567 microsat1 GTC G,GTCT 50   PASS   NS=3;DP=9;AA=G                   GT:GQ:DP    0/1:35:4       0/2:17:2       1/1:40:3
```

This example shows (in order): a good simple SNP, a possible SNP that has been filtered out because its quality is below 10, a site at which two alternate alleles are called, with one of them (T) being ancestral (possibly a reference sequencing error), a site that is called monomorphic reference (i.e. with no alternate alleles), and a microsatellite with two alternative alleles, one a deletion of 2 bases (TC), and the other an insertion of one base (T). Genotype data are given for three samples, two of which are phased and the third unphased, with per sample genotype quality, depth and haplotype qualities (the latter only for the phased samples) given as well as the genotypes. The microsatellite calls are unphased.

1.2 Character encoding, non-printable characters and characters with special meaning

The character encoding of VCF files is UTF-8. UTF-8 is a multi-byte character encoding that is a strict superset of 7-bit ASCII and has the property that none of the bytes in any multi-byte characters are 7-bit ASCII bytes. As a result, most software that processes VCF files does not have to be aware of the possible presence of multi-byte UTF-8 characters. VCF files must not contain a byte order mark. Note that non-printable characters U+0000–U+0008, U+000B–U+000C, U+000E–U+001F are disallowed. Line separators must be CR+LF or LF and they are allowed only as line separators at end of line. Some characters have a special meaning when they appear (such as field delimiters ';' in INFO or ':' FORMAT fields), and for any other meaning they must be represented with the capitalized percent encoding:

%3A	:	(colon)
%3B	;	(semicolon)
%3D	=	(equal sign)
%25	%	(percent sign)
%2C	,	(comma)
%0D	CR	
%0A	LF	
%09	TAB	

1.3 Data types

Data types supported by VCF are: Integer (32-bit, signed), Float (32-bit IEEE-754, formatted to match one of the regular expressions `^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$` or `^[-+]?(INF|INFINITY|NAN)$` case insensi-

tively),* Flag, Character, and String. For the Integer type, the values from -2^{31} to $-2^{31}+7$ cannot be stored in the binary version and therefore are disallowed in both VCF and BCF, see 6.3.3.

1.4 Meta-information lines

File meta-information lines start with "`##`" and must appear first in the VCF file, before the header line (section 1.5) and data record lines (section 1.6). They may be either *unstructured* or *structured*.

An *unstructured* meta-information line consists of a *key* (denoting the type of meta-information recorded) and a *value* (which may not be empty and must not start with a '<' character), separated by an '=' character:

> `##`*key*=*value*

Several unstructured meta-information lines are defined in this specification, notably `##fileformat`. Others not defined by this specification, e.g. `##fileDate` and `##source`, are commonly found in VCF files. These typically have meanings that are obvious, or they are immaterial for processing the file, or both.

A *structured* meta-information line is similar, but the value is itself a comma-separated list of key=value pairs, enclosed within '<' and '>' characters:

> `##`*key*=<*key*=*value*,*key*=*value*,*key*=*value*,...>

All structured lines require an ID which must be unique within their type, i.e., within all the meta-information lines with the same "`##`*key*=" prefix. For all of the structured lines (`##INFO`, `##FORMAT`, `##FILTER`, etc.) described in this specification, optional fields can be included. For example:

`##INFO=<ID=ALLELEID,Number=A,Type=String,Description="Allele ID",Source="ClinVar",Version="20220804">`

In the above example, the optional fields of "Source" and "Version" are provided. The values of optional fields must be written as quoted strings, even for numeric values. Other structured lines not defined by this specification may also be used; the only required field for such lines is the required `ID` field.

It is recommended in VCF and required in BCF that the header includes tags describing the reference and contigs backing the data contained in the file. These tags are based on the SQ field from the SAM spec; all tags are optional (see the VCF example above).

To aid human readability, the order of fields should be ID, Number, Type, Description, then any optional fields. Implementations must not rely on the order of the fields within structured lines and are not required to preserve field ordering.

Meta-information lines are optional, but if they are present then they must be completely well-formed. Other than `##fileformat`, they may appear in any order. Note that BCF, the binary counterpart of VCF, requires that all entries are present. It is recommended to include meta-information lines describing the entries used in the body of the VCF file.

1.4.1 File format

A single 'fileformat' line is always required, must be the first line in the file, and details the VCF format version number. For VCF version 4.4, this line is:

`##fileformat=VCFv4.4`

1.4.2 Information field format

INFO meta-information lines are structured lines with required fields ID, Number, Type, and Description, and recommended optional fields Source and Version:

`##INFO=<ID=ID,Number=number,Type=type,Description="description",Source="source",Version="version">`

Possible Types for INFO fields are: Integer, Float, Flag, Character, and String. The Number entry is an Integer that describes the number of values that can be included with the INFO field. For example, if the INFO field contains a single number, then this value must be 1; if the INFO field describes a pair of numbers, then this value must be 2 and so on. There are also certain special characters used to define special cases:

---

*Note Java's `Double.valueOf` is particular about capitalisation, so additional code is needed to parse all VCF infinite/NaN values.

- A: The field has one value per alternate allele. The values must be in the same order as listed in the ALT column (described in section 1.6).

- R: The field has one value for each possible allele, including the reference. The order of the values must be the reference allele first, then the alternate alleles as listed in the ALT column.

- G: The field has one value for each possible genotype. The values must be in the same order as prescribed in section 1.6.2 (see GENOTYPE ORDERING).

- . (dot): The number of possible values varies, is unknown or unbounded.

The 'Flag' type indicates that the INFO field does not contain a Value entry, and hence the Number must be 0 in this case. The Description value must be surrounded by double-quotes. Double-quote character must be escaped with backslash \ and backslash as \\. Source and Version values likewise must be surrounded by double-quotes and specify the annotation source (case-insensitive, e.g. `"dbsnp"`) and exact version (e.g. `"138"`), respectively for computational use.

1.4.3 Filter field format

FILTER meta-information lines are structured lines with required fields ID and Description that define the possible content of the FILTER column in the VCF records:

```
##FILTER=<ID=ID,Description="description">
```

1.4.4 Individual format field format

FORMAT meta-information lines are structured lines with required fields ID, Number, Type, and Description that define the possible content of the per-sample/genotype columns in the VCF records:

```
##FORMAT=<ID=ID,Number=number,Type=type,Description="description">
```

Possible Types for FORMAT fields are: Integer, Float, Character, and String (this field is otherwise defined precisely as the INFO field). The Number field is defined as per the INFO Number field with the following additional possibilities:

- P: The field has one value for each allele value defined in *GT*.

1.4.5 Alternative allele field format

ALT meta-information lines are structured lines with require fields of ID and Description that describe the possible symbolic alternate alleles in the ALT column of the VCF records:

```
##ALT=<ID=type,Description="description">
```

Structural Variants
In symbolic alternate alleles for structural variants, the ID field indicates the type of structural variant, and can be a colon-separated list of types and subtypes. ID values are case sensitive strings and must not contain whitespace, commas or angle brackets (See 1.6.1.5) The first level type must be one of the following:

- DEL Region of lowered copy number relative to the reference, or a deletion breakpoint

- INS Insertion of novel sequence relative to the reference

- DUP Region of elevated copy number relative to the reference, or a tandem duplication breakpoint

- INV Inversion of reference sequence

- CNV Region of uniform copy number (may be deletion, duplication or copy number neutral)

The CNV symbolic allele should not be used when a more specific one (e.g. DEL, CNV:TR) can be applied.
Implementations are free to define their own subtypes. The presence of a subtype does not change either the copy number or breakpoint interpretation of a symbolic structural variant allele. The following subtypes are recommended:

- CNV:TR Tandem repeat. See 5.7 for further details.

- DUP:TANDEM Tandem duplication

- DEL:ME Deletion of mobile element relative to the reference

- INS:ME Insertion of a mobile element relative to the reference

Note that the position of symbolic structural variant alleles is the position of the base immediately preceding the variant.

IUPAC ambiguity codes
Symbolic alleles can be used also to represent genuinely ambiguous data in VCF, for example:

```
##ALT=<ID=R,Description="IUPAC code R = A/G">
##ALT=<ID=M,Description="IUPAC code M = A/C">
```

1.4.6 Assembly field format

Breakpoint assemblies for structural variations may use an external file:

`##assembly=url`

The URL field specifies the location of a fasta file containing breakpoint assemblies referenced in the VCF records for structural variants via the BKPTID INFO key.

1.4.7 Contig field format

It is recommended for VCF, and required for BCF, that the header includes tags describing the contigs referred to in the file. The structured `contig` field must include the ID attribute and can include additional optional attributes with the following ones reserved:

- length: the length of the sequence

- md5: MD5 checksum of the sequence as defined in the Sam specification v1[†] Briefly, the digest is calculated excluding all characters outside of the inclusive range 33 ('!') to 126 ('~'). and all lowercase characters converted to uppercase. The MD5 digest is calculated as described in *RFC 1321* and presented as a 32 character lowercase hexadecimal number.

- URL: tag to indicate where the sequence can be found

For example:

`##contig=<ID=ctg1,length=81195210,URL=ftp://somewhere.example/assembly.fa,md5=f126cdf8a6e0c7f379d618ff66beb2da,...>`

Contig names follow the same rules as the SAM format's reference sequence names: they may contain any printable ASCII characters in the range [!-~] apart from '\ , "' () [] {} <>' and may not start with '*' or '='. Thus they match the following regular expression:

`[0-9A-Za-z!#$%&+./:;?@^_|~-][0-9A-Za-z!#$%&*+./:;=?@^_|~-]*`

In particular, excluding commas facilitates parsing `##contig` lines, and excluding the characters '<>[]' and initial '*' avoids clashes with symbolic alleles. The contig names must not use a reserved symbolic allele name.

1.4.8 Sample field format

It is possible to define sample to genome mappings as shown below:

```
##META=<ID=Assay,Type=String,Number=.,Values=[WholeGenome, Exome]>
##META=<ID=Disease,Type=String,Number=.,Values=[None, Cancer]>
##META=<ID=Ethnicity,Type=String,Number=.,Values=[AFR, CEU, ASN, MEX]>
##META=<ID=Tissue,Type=String,Number=.,Values=[Blood, Breast, Colon, Lung, ?]>
##SAMPLE=<ID=Sample1,Assay=WholeGenome,Ethnicity=AFR,Disease=None,Description="Patient germline genome from unaffected",DOI=url>
##SAMPLE=<ID=Sample2,Assay=Exome,Ethnicity=CEU,Disease=Cancer,Tissue=Breast,Description="European patient exome from breast cancer">
```

[†]See Reference MD5 calculation section in `SAM Format Specification`.

1.4.9 Pedigree field format

It is possible to record relationships between genomes using the following syntax:

```
##PEDIGREE=<ID=TumourSample,Original=GermlineID>
##PEDIGREE=<ID=SomaticNonTumour,Original=GermlineID>
##PEDIGREE=<ID=ChildID,Father=FatherID,Mother=MotherID>
##PEDIGREE=<ID=SampleID,Name_1=Ancestor_1,...,Name_N=Ancestor_N>
```

or a link to a database:

```
##pedigreeDB=URL
```

See 5.4.11 for details.

1.5 Header line syntax

The header line names the 8 fixed, mandatory columns. These columns are as follows:

#CHROM   POS   ID   REF   ALT   QUAL   FILTER   INFO

If genotype data is present in the file, these are followed by a FORMAT column header, then an arbitrary number of sample IDs. Duplicate sample IDs are not allowed. The header line is tab-delimited and there must be no tab characters at the end of the line.

1.6 Data lines

All data lines are tab-delimited with no tab character at the end of the line. The last data line must end with a line separator. In all cases, missing values are specified with a dot ('.').

1.6.1 Fixed fields

There are 8 fixed fields per record. Fixed fields are:

1. CHROM — chromosome: An identifier from the reference genome or an angle-bracketed ID String ("<ID>") pointing to a contig in the assembly file (cf. the ##assembly line in the header). All entries for a specific CHROM must form a contiguous block within the VCF file. (String, no whitespace permitted, Required).

2. POS — position: The reference position, with the 1st base having position 1. Positions are sorted numerically, in increasing order, within each reference sequence CHROM. It is permitted to have multiple records with the same POS. Telomeres are indicated by using positions 0 or N+1, where N is the length of the corresponding chromosome or contig. (Integer, Required)

3. ID — identifier: Semicolon-separated list of unique identifiers where available. If this is a dbSNP variant the rs number(s) should be used. No identifier should be present in more than one data record. If there is no identifier available, then the MISSING value should be used. (String, no whitespace or semicolons permitted, duplicate values not allowed.)

4. REF — reference base(s): Each base must be one of A,C,G,T,N (case insensitive). Multiple bases are permitted. The value in the POS field refers to the position of the first base in the String. For simple insertions and deletions in which either the REF or one of the ALT alleles would otherwise be null/empty, the REF and ALT Strings must include the base before the variant (which must be reflected in the POS field), unless the variant occurs at position 1 on the contig in which case it must include the base after the variant; this padding base is not required (although it is permitted) e.g. for complex substitutions or other variants where all alleles have at least one base represented in their Strings. If any of the ALT alleles is a symbolic allele (an angle-bracketed ID String "<ID>") then the padding base is required and POS denotes the coordinate of the base preceding the polymorphism. The exception to this is the <*> symbolic allele for which the reference call interval includes the POS base. Tools processing VCF files are not required to preserve case in the REF allele Strings. (String, Required).

   If the reference sequence contains IUPAC ambiguity codes not allowed by this specification (such as R = A/G), the ambiguous reference base must be reduced to a concrete base by using the one that is first alphabetically (thus R as a reference base is converted to A in VCF.)

5. ALT — alternate base(s): Comma-separated list of alternate non-reference alleles. These alleles do not have to be called in any of the samples. Each allele in this list must be one of: a non-empty String of bases (A,C,G,T,N; case insensitive); the '*' symbol (allele missing due to overlapping deletion); the MISSING value '.' (no variant); an angle-bracketed ID String ("<ID>"); the unspecified allele "<*>" as described in Section 5.5; or a breakend replacement string as described in Section 5.4. If there are no alternative alleles, then the MISSING value must be used. Tools processing VCF files are not required to preserve case in the allele String, except for IDs, which are case sensitive. (String; no whitespace, commas, or angle-brackets are permitted in the ID String itself)

6. QUAL — quality: Phred-scaled quality score for the assertion made in ALT. i.e. $-10log_{10}$ prob(call in ALT is wrong). If ALT is '.' (no variant) then this is $-10log_{10}$ prob(variant), and if ALT is not '.' this is $-10log_{10}$ prob(no variant). If unknown, the MISSING value must be specified. (Float)

7. FILTER — filter status: PASS if this position has passed all filters, i.e., a call is made at this position. Otherwise, if the site has not passed all filters, a semicolon-separated list of codes for filters that fail. e.g. "q10;s50" might indicate that at this site the quality is below 10 and the number of samples with data is below 50% of the total number of samples. '0' is reserved and must not be used as a filter String. If filters have not been applied, then this field must be set to the MISSING value. (String, no whitespace or semicolons permitted, duplicate values not allowed.)

8. INFO — additional information: Semicolon-separated series of additional information fields, or the MISSING value '.' if none are present. Each subfield consists of a short *key* with optional *values* in the format: key[=value[,...,value]]. Literal semicolon (';') and equals sign ('=') characters are not permitted in these values, and literal commas (',') are permitted only as delimiters for lists of values; characters with special meaning can be encoded using percent encoding, see Section 1.2. Space characters are allowed in values.

   INFO keys must match the regular expression `^([A-Za-z_][0-9A-Za-z_.]*|1000G)$`, please note that "1000G" is allowed as a special legacy value. Duplicate keys are not allowed. Arbitrary keys are permitted, although those listed in Table 1 and described below are reserved (albeit optional).

   The exact format of each INFO key should be specified in the meta-information (as described above). Example of a complete INFO field: `DP=154;MQ=52;H2`. Keys without corresponding values may be used to indicate group membership (e.g. H2 indicates the SNP is found in HapMap 2). See Section 3 for additional reserved INFO keys used to encode structural variants.

Key	Number	Type	Description
AA	1	String	Ancestral allele
AC	A	Integer	Allele count in genotypes, for each ALT allele, in the same order as listed
AD	R	Integer	Total read depth for each allele
ADF	R	Integer	Read depth for each allele on the forward strand
ADR	R	Integer	Read depth for each allele on the reverse strand
AF	A	Float	Allele frequency for each ALT allele in the same order as listed (estimated from primary data, not called genotypes)
AN	1	Integer	Total number of alleles in called genotypes
BQ	1	Float	RMS base quality
CIGAR	A	String	Cigar string describing how to align an alternate allele to the reference allele
DB	0	Flag	dbSNP membership
DP	1	Integer	Combined depth across samples
END	1	Integer	End position on CHROM (used with symbolic alleles; see below)
H2	0	Flag	HapMap2 membership
H3	0	Flag	HapMap3 membership
MQ	1	Float	RMS mapping quality
MQ0	1	Integer	Number of MAPQ == 0 reads
NS	1	Integer	Number of samples with data
SB	4	Integer	Strand bias
SOMATIC	0	Flag	Somatic mutation (for cancer genomics)

Continued on next page...

Table 1: Reserved INFO keys

9

Key	Number	Type	Description
VALIDATED	0	Flag	Validated by follow-up experiment
1000G	0	Flag	1000 Genomes membership

Table 1: Reserved INFO keys

- END: End reference position (1-based), indicating the variant spans positions POS–END on reference/contig CHROM. Normally this is the position of the last base in the REF allele, so it can be derived from POS and the length of REF, and no END INFO field is needed. However when symbolic alleles are used, e.g. in gVCF or structural variants, an explicit END INFO field provides variant span information that is otherwise unknown. If a record containing a symbolic structural variant allele does not have an END field, it must be computed from the SVLEN field as per Section 3.

  This field is used to compute BCF's `rlen` field (see 6.3.1) and is important when indexing VCF/BCF files to enable random access and querying by position.

1.6.2 Genotype fields

If genotype information is present, then the same types of data must be present for all samples. First a FORMAT field is given specifying the data types and order (colon-separated FORMAT keys matching the regular expression `^[A-Za-z_][0-9A-Za-z_.]*$`, duplicate keys are not allowed). This is followed by one data block per sample, with the colon-separated data corresponding to the types specified in the format. The first key must always be the genotype (GT) if it is present. There are no required keys. Additional Genotype keys can be defined in the meta-information, however, software support for them is not guaranteed.

If any of the fields is missing, it is replaced with the MISSING value. For example if the FORMAT is GT:GQ:DP:HQ then $0\,|\,0:.\,.:23:23,34$ indicates that GQ is missing. If a field contains a list of missing values, it can be represented either as a single MISSING value ('.') or as a list of missing values (e.g. '.,.,.' if the field was Number=3). Trailing fields can be dropped, with the exception of the GT field, which should always be present if specified in the FORMAT field.

As with the INFO field, there are several common, reserved keywords that are standards across the community. See their detailed definitions below, as well as Table 2 for their reference Number, Type and Description. See also Section 4 for a list of genotype keys reserved for structural variants.

Field	Number	Type	Description
AD	R	Integer	Read depth for each allele
ADF	R	Integer	Read depth for each allele on the forward strand
ADR	R	Integer	Read depth for each allele on the reverse strand
DP	1	Integer	Read depth
EC	A	Integer	Expected alternate allele counts
FT	1	String	Filter indicating if this genotype was "called"
GL	G	Float	Genotype likelihoods
GP	G	Float	Genotype posterior probabilities
GQ	1	Integer	Conditional genotype quality
GT	1	String	Genotype
HQ	2	Integer	Haplotype quality
MQ	1	Integer	RMS mapping quality
PL	G	Integer	Phred-scaled genotype likelihoods rounded to the closest integer
PP	G	Integer	Phred-scaled genotype posterior probabilities rounded to the closest integer
PQ	1	Integer	Phasing quality
PS	1	Integer	Phase set
PSL	P	String	Phase set list
PSO	P	Integer	Phase set list ordinal

PSQ	P	Integer	Phase set list quality

- AD, ADF, ADR (Integer): Per-sample read depths for each allele; total (AD), on the forward (ADF) and the reverse (ADR) strand.

- DP (Integer): Read depth at this position for this sample.

- EC (Integer): Comma separated list of expected alternate allele counts for each alternate allele in the same order as listed in the ALT field. Typically used in association analyses.

- FT (String): Sample genotype filter indicating if this genotype was "called" (similar in concept to the FILTER field). Again, use PASS to indicate that all filters have been passed, a semicolon-separated list of codes for filters that fail, or '.' to indicate that filters have not been applied. These values should be described in the meta-information in the same way as FILTERs. No whitespace or semicolons permitted.

- GQ (Integer): Conditional genotype quality, encoded as a phred quality $-10log_{10}$ p(genotype call is wrong, conditioned on the site's being variant).

- GP (Float): Genotype posterior probabilities in the range 0 to 1 using the same ordering as the GL field; one use can be to store imputed genotype probabilities.

- GT (String): Genotype, encoded as allele value preceded by either of / or | depending on whether that allele is considered phased. The first phasing indicator may be omitted and is implicitly defined as / if any phasing indicators are / and | otherwise. The allele values are 0 for the reference allele (what is in the REF field), 1 for the first allele listed in ALT, 2 for the second allele list in ALT and so on. For diploid calls examples could be 0/1, 1 | 0, /0/1, or 1/2, etc. Haploid calls, e.g. on Y, male non-pseudoautosomal X, or mitochondria, should be indicated by having only one allele value. A triploid call might look like 0/0/1, and a partially phased triploid call could be |0/1/2 to indicate that the first allele is phased with another variant in the VCF. If a call cannot be made for a sample at a given locus, '.' must be specified for each missing allele in the GT field (for example './.' for a diploid genotype and '.' for haploid genotype). The meanings of the phasing indicators are as follows (see the PS and PSL fields below for more details on incorporating phasing information into the genotypes):

  ○ / : allele is unphased
  ○ | : allele is phased (according to the phase-set indicated in PS or PSL)

  For symbolic structural variant alleles, GT=0 indicates the absence of any of the ALT symbolic structural variants defined in the record. Implementer should note that merging a VCF record containing only symbolic structural variant ALT alleles with a record containing other alleles will result a change of the meaning of the GT=0 haplotypes from the record containing only symbolic SVs.

- GL (Float): Genotype likelihoods comprised of comma separated floating point log_{10}-scaled likelihoods for all possible genotypes given the set of alleles defined in the REF and ALT fields. In presence of the GT field the same ploidy is expected; without GT field, diploidy is assumed.

  GENOTYPE ORDERING. In general case of ploidy P and N alternate alleles (0 is the REF and $1\ldots N$ the alternate alleles), the ordering of genotypes for the likelihoods can be expressed by the following pseudocode with as many nested loops as ploidy: ‡

```
for aP = 0...N
   for aP-1 = 0...aP
       ...
       for a1 = 0...a2
          println a1a2...aP
```

  Alternatively, the same can be achieved recursively with the following pseudocode:

---

‡Note that we use inclusive for loop boundaries.

```
Ordering(P, N, suffix=""):
    for a in 0...N
        if (P == 1) println str(a) + suffix
        if (P > 1) Ordering(P-1, a, str(a) + suffix)
```

Conversely, the index of the value corresponding to the genotype $k_1 \leq k_2 \leq \ldots \leq k_P$ is

$$\texttt{Index}(k_1/k_2/\ldots/k_P) \;=\; \sum_{m=1}^{P} \binom{k_m+m-1}{m}$$

Examples:

- for $P=2$ and $N=1$, the ordering is 00,01,11
- for $P=2$ and $N=2$, the ordering is 00,01,11,02,12,22
- for $P=3$ and $N=2$, the ordering is 000, 001, 011, 111, 002, 012, 112, 022, 122, 222
- for $P=1$, the index of the genotype a is a
- for $P=2$, the index of the genotype "a/b", where $a \leq b$, is $b(b+1)/2 + a$
- for $P=2$ and arbitrary N, the ordering can be easily derived from a triangular matrix

$b \setminus a$	0	1	2	3
0	0			
1	1	2		
2	3	4	5	
3	6	7	8	9

- HQ (Integer): Haplotype qualities, two comma separated phred qualities.

- MQ (Integer): RMS mapping quality, similar to the version in the INFO field.

- PL (Integer): The phred-scaled genotype likelihoods rounded to the closest integer, and otherwise defined in the same way as the GL field.

- PP (Integer): The phred-scaled genotype posterior probabilities rounded to the closest integer, and otherwise defined in the same way as the GP field.

- PQ (Integer): Phasing quality, the phred-scaled probability that alleles are ordered incorrectly in a heterozygote (against all other members in the phase set). We note that we have not yet included the specific measure for precisely defining "phasing quality"; our intention for now is simply to reserve the PQ tag for future use as a measure of phasing quality.

- PS (non-negative 32-bit Integer): Phase set, defined as a set of phased genotypes to which this genotype belongs. Phased genotypes for an individual that are on the same chromosome and have the same PS value are in the same phased set. A phase set specifies multi-marker haplotypes for the phased genotypes in the set. All phased genotypes that do not contain a PS subfield are assumed to belong to the same phased set. If the genotype in the GT field is unphased, the corresponding PS field is ignored. The recommended convention is to use the position of the first variant in the set as the PS identifier (although this is not required).

- PSL (List of Strings): The list of phase sets, one for each allele specified in the GT. Unphased alleles (without a | separator before them) must have the value '.' in their corresponding position in the list. Unlike PS (which is defined per CHROM), records with different CHROM but the same phase-set name are considered part of the same phase set. If an implementation cannot guarantee uniqueness of phase-set names across the VCF (for example, phasing a streaming VCF or each CHROM is processed independently in parallel), new phase-set names should be of the format CHROM*POS*ALLELE-NUMBER of the "first" allele which is included in this set, with ALLELE-NUMBER being the index of the allele in the GT field, since multiple distinct phase-sets could start at the same position. § A given sample-genotype must not have values for both PS and PSL. In addition, PS and PSL are not interoperable, in that a PS mentioned in one variant cannot be referenced in a PSL in another, since when used in PS it isn't connected to any specific haplotype (i.e. first or second), but PSL is.

  Example:

---

§The '*' character is used as a separator since ':' is not reserved in the CHROM column.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SAMPLE1
chr19	5	.	T	G	.	PASS	DP=100	GT:PSL	|0/1:chr9*5*1,.
chr20	10	.	A	T,G	.	PASS	DP=100	GT:PSL	|1/2|3:chr20*10*1,.,chr9*5*1
chr20	15	.	G	C	.	PASS	DP=100	GT:PSL	1|2:.,chr20*10*1

- PSO (List of integers): List of phase set ordinals. For each phase-set name, defines the order in which variants are encountered when traversing a derivative chromosome. The missing value '.' should be used when the corresponding PSO value is missing. For each phase-set name, PSO should be defined if any allele with that phase-set name on any record is symbolic structural variant or in breakpoint notation. Variants in breakpoint notation must have the same PSL and PSO on both records.

  Without explicitly specifying the derivative chromosome traversal order, multiple derivative chromosome reconstructions are possible. Take for example this tandem duplication in a triploid organism with SNVs (ID/QUAL/FILTER columns removed for clarity):

#CHROM	POS	REF	ALT	INFO	FORMAT	SAMPLE1
chr1	10	T	<DUP>	SVCLAIM=DJ	GT:PSL:PSO	/0/0|1:.,.,chr1*10*1:.,.,3
chr1	20	A	G	.	GT:PSL:PSO	/0/0|0|1:.,.,chr1*10*1,chr1*10*1:.,.,4,
chr1	30	G	T	.	GT:PSL:PSO	/0/0|0|1:.,.,chr1*10*1,chr1*10*1:.,.,2,

  Without defining PSO, it would be ambiguous as to which copy of the duplicated region the SNVs occur on. In this example, the presence of the PSO field clarifies that the SNVs are cis phased with the duplication, the first SNV occurs on the first copy of the duplicated region, and second SNV on the second copy.

- PSQ (List of integers): The list of PQs, one for each phase set in PSL (encoded like PQ). The missing value '.' should be used when the corresponding PSL value is missing, or when the phasing is of unknown quality.

2 Understanding the VCF format and the haplotype representation

VCF records use a single general system for representing genetic variation data composed of:

- Allele: representing single genetic haplotypes (A, T, ATC).

- Genotype: an assignment of alleles for each chromosome of a single named sample at a particular locus.

- VCF record: a record holding all segregating alleles at a locus (as well as genotypes, if appropriate, for multiple individuals containing alleles at that locus).

VCF records use a simple haplotype representation for REF and ALT alleles to describe variant haplotypes at a locus. ALT haplotypes are constructed from the REF haplotype by taking the REF allele bases at the POS in the reference genotype and replacing them with the ALT bases. In essence, the VCF record specifies a-REF-t and the alternative haplotypes are a-ALT-t for each alternative allele.

2.1 VCF tag naming conventions

Several tag names follow conventions indicating how their values are represented numerically:

- The 'L' suffix means *likelihood* as log-likelihood in the sampling distribution, $\log_{10} \Pr(\text{Data}|\text{Model})$. Likelihoods are represented as $\log_{10}$ scale, thus they are negative numbers (e.g. GL, CNL). The likelihood can be also represented in some cases as phred-scale in a separate tag (e.g. PL).

- The 'P' suffix means *probability* as linear-scale probability in the posterior distribution, which is $\Pr(\text{Model}|\text{Data})$. Examples are GP, CNP.

- The 'Q' suffix means *quality* as log-complementary-phred-scale posterior probability, $-10 \log_{10} \Pr(\text{Data}|\text{Model})$, where the model is the most likely genotype that appears in the GT field. Examples are GQ, CNQ. The fixed site-level QUAL field follows the same convention (represented as a phred-scaled number).

3   INFO keys used for structural variants

The following INFO keys are reserved for encoding structural variants. In general, when these keys are used by imprecise variants, the values should be best estimates. When present, per allele values must be specified for all ALT alleles (including non-structural alleles). Except in lists of strings, the missing value should be used as a placeholder for the ALT alleles for which the key does not have a meaningful value. The empty string should be used to encode missing values in lists of strings.

```
##INFO=<ID=IMPRECISE,Number=0,Type=Flag,Description="Imprecise structural variation">
```

Indicates that this record contains an imprecise structural variant ALT allele. ALT alleles missing $CIPOS$ are to be interpreted as imprecise variants with an unspecified confidence interval.

If a precise ALT allele is present in a record with the $IMPRECISE$ flag, $CIPOS$ must be explicitly set for that allele, even if it is '0,0'.

```
##INFO=<ID=NOVEL,Number=0,Type=Flag,Description="Indicates a novel structural variation">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the longest variant described in this record">
```

END position of the longest variant described in this record. The END of each allele is defined as:

Non-symbolic alleles: $\text{POS} + \text{length of REF allele} - 1$.

<INS> symbolic structural variant alleles: $\text{POS} + \text{length of REF allele} - 1$.

<DEL>, <DUP>, <INV>, and <CNV> symbolic structural variant alleles:, $\text{POS} + \text{SVLEN}$.

<*> symbolic allele: the last reference call position.

END must be present for all records containing the <*> symbolic allele and, for backwards compatibility, should be present for records containing any symbolic structural variant alleles.

To prevent loss of information, any VCF record containing the <*> symbolic allele must have END set to the last reference call position of the <*> symbolic allele. When a record contains both the <*> symbolic allele, the END position of the longest allele should be used as the record end position for indexing purposes.

```
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
```

This field has been deprecated due to redundancy with ALT. Refer to section 1.4.5 for the set of valid ALT field symbolic structural variant alleles.

```
##INFO=<ID=SVLEN,Number=A,Type=Integer,Description="Length of structural variant">
```

One value for each ALT allele.

SVLEN must be specified for symbolic structural variant alleles. SVLEN is defined for INS, DUP, INV, and DEL symbolic alleles as the number of the inserted, duplicated, inverted, and deleted bases respectively. SVLEN is defined for CNV symbolic alleles as the length of the segment over which the copy number variant is defined. The missing value . should be used for all other ALT alleles, including ALT alleles using breakend notation.

For backwards compatibility, a missing SVLEN should be inferred from the END field of VCF records whose ALT field contains a single symbolic allele.

For backwards compatibility, the absolute value of SVLEN should be taken and a negative SVLEN should be treated as positive values.

Note that for structural variant symbolic alleles, POS corresponds to the base immediately preceding the variant.

```
##INFO=<ID=CIPOS,Number=.,Type=Integer,Description="Confidence interval around POS for symbolic structural variants">
```

If present, the number of entries must be twice the number of ALT alleles. $CIPOS$ consists of successive pairs of records indicating the start and end offsets relative to POS of the confidence interval for each ALT allele. For example, $CIPOS = -5, 5, 0, 0$ indicates a 5bp confidence interval in each direction for the first ALT allele, and an exact position for the second alt allele.

When breakpoint sequence homology exists, $CIPOS$ should be used in conjunction with $HOMSEQ$ to specify the interval of homology.

If both $IMPRECISE$ and $CIPOS$ are omitted, $CIPOS$ is implicitly defined as 0,0 for all alleles.

Each $CIPOS$ interval must span 0. That is, the lower bound cannot be greater than 0, and the upper bound cannot be less than 0.

```
##INFO=<ID=CIEND,Number=.,Type=Integer,Description="Confidence interval around END for symbolic structural variants">
```

If present, the number of entries must be twice the number of ALT alleles. $CIEND$ consists of successive pairs of records encoding the confidence interval start and end offsets relative to the END position inferred by $SVLEN$ for each ALT allele. For symbolic structural variants, the first in the pair must not be greater than 0, and the second must not be less than 0. For all other alleles, both should be the missing value .. For example, $CIEND = -5, 5, ., .$ indicates a 5bp confidence interval in each direction around the end position for the first ALT allele, and no $CIEND$ is defined for the second alt allele.

If $CIEND$ is missing, it is assumed to match $CIPOS$.

```
##INFO=<ID=HOMLEN,Number=A,Type=Integer,Description="Length of base pair identical micro-homology at breakpoints">

##INFO=<ID=HOMSEQ,Number=A,Type=String,Description="Sequence of base pair identical micro-homology at breakpoints">

##INFO=<ID=BKPTID,Number=A,Type=String,Description="ID of the assembled alternate allele in the assembly file">
```

For precise variants, the consensus sequence the alternate allele assembly is derivable from the REF and ALT fields. However, the alternate allele assembly file may contain additional information about the characteristics of the alt allele contigs.

```
##INFO=<ID=MEINFO,Number=.,Type=String,Description="Mobile element info of the form NAME,START,END,POLARITY">
```

If present, the number of entries must be four (4) times the number of ALT alleles. $MEINFO$ consists of successive quadruplets of records for each ALT allele.

```
##INFO=<ID=METRANS,Number=.,Type=String,Description="Mobile element transduction info of the form CHR,START,END,POLARITY">
```

If present, the number of entries must be four (4) times the number of ALT alleles. $METRANS$ consists of successive quadruplets of records for each ALT allele.

```
##INFO=<ID=DGVID,Number=A,Type=String,Description="ID of this element in Database of Genomic Variation">
##INFO=<ID=DBVARID,Number=A,Type=String,Description="ID of this element in DBVAR">
##INFO=<ID=DBRIPID,Number=A,Type=String,Description="ID of this element in DBRIP">
##INFO=<ID=MATEID,Number=A,Type=String,Description="ID of mate breakend">
##INFO=<ID=PARID,Number=A,Type=String,Description="ID of partner breakend">
##INFO=<ID=EVENT,Number=A,Type=String,Description="ID of associated event">
##INFO=<ID=EVENTTYPE,Number=A,Type=String,Description="Type of associated event">
```

Whilst simple events such as deletions and duplications can be wholly represented by a single VCF record, complex rearrangements such as chromothripsis result in a large number of breakpoints. VCF uses the $EVENT$ field to group such related records together, and $EVENTTYPE$ to classify these events. All records with the same $EVENT$ value are considered to be part of the same event.

The following $EVENTTYPE$ values are reserved and should be used when appropriate:

- DEL - Deletion

- DEL:ME - Deletion of mobile element with respect to the reference

- INS - Insertion

- INS:ME - Insertion of mobile element

- DUP - Duplication

- DUP:TANDEM - Tandem duplication

- DUP:DISPERSED - Dispersed duplication

- INV - Inversion

- TRA - Translocation

- TRA:BALANCED - Balanced inter-chromosomal translocation

- TRA:UNBALANCED - Unbalanced inter-chromosomal translocation

- CHROMOTHRIPSIS - Chromothripsis

- CHROMOPLEXY - Chromoplexy

- BFB - breakage fusion bridge

- DOUBLEMINUTE - Double minute

The semantics of other $EVENTTYPE$ values is implementation-defined. The use of $EVENT$ is not restricted to structural variation and can also be used to associate non-symbolic alleles. Such linking is useful for scenarios such as kataegis or when there is variant position ambiguity in segmentally duplicated regions.

```
##INFO=<ID=CILEN,Number=.,Type=Integer,Description="Confidence interval for the SVLEN field">
```

If present, the number of entries must be twice the number of ALT alleles. $CILEN$ consists of successive pairs of records indicating the lower and upper bounds of the $SVLEN$ confidence interval.

```
##INFO=<ID=CN,Number=A,Type=Float,Description="Copy number of CNV/breakpoint">
##INFO=<ID=CICN,Number=.,Type=Float,Description="Confidence interval around copy number">
```

If present, the number of entries must be twice the number of ALT alleles. $CICN$ consists of successive pairs of records indicating the lower and upper copy number bounds.

```
##INFO=<ID=SVCLAIM,Number=A,Type=String,Description="Claim made by the structural variant call. Valid values are D, J, DJ for abunda
```

This field disambiguates the claim being made by DEL and DUP structural variants.

The distinction between different claim types is made necessary by the fact that many structural variants affect both the overall abundance of a region and its adjacency structure, but often only one of those effects is being directly observed. It is important to specify which one, because if this is not done, other variants in a genome may make the call ambiguous.

For example, a "conventional" deletion usually both decreases the DNA abundance of a region and creates an adjacency between two sequence locations, which are the breakpoints of that deletion. However, if a deletion or duplication is accompanied by a compensatory variant elsewhere in the genome, it may only cause changes in the adjacency structure, but not the overall copy number. On the contrary, combinations of certain complex events in a genome may cause the overall copy number decrease, but not the linear excision of a sequence fragment.

A copy number based duplication call does not make any claims on where the duplicated region is in the genome. It could be a simple tandem duplication, a dispersed duplication, retrotransposition, or even form part of extra-chromosomal DNA such as a double minute. In contrast, a breakpoint based duplication call requires that the end of the duplicated region be adjacent to the start of the duplicated region. This rules out events such as dispersed duplication or retrotransposition, but is consistent with simple tandem duplications as well as double minutes. Both deletion and duplication breakpoint calls can form part of more complex events that result in no copy number change in the deleted/duplicated region.

To resolve any ambiguity in the interpretation of deletion and duplication symbolic alleles, three claim types are defined:

- D (abun**d**ance / read **D**epth) claim indicates that the call has been made based only on a measure of DNA abundance of the called region, with no evidence to support changes in breakpoint structure. This includes indirect claims of abundance made using SNV variant allele frequency.

- J (ad**j**acency / break **J**unction) claim indicates that the call has been made based on the detection of a non-reference DNA adjacency, with no evidence to support overall changes in DNA abundance.

- DJ indicates that there is evidence for both DNA abundance and adjacency changes, which are consistent with each other and suggest the structural variant of the type being reported.

Rules of applying SVCLAIM field are different depending on symbolic allele type:

- DEL/DUP: SVCLAIM must be specified and can be be D, J, or DJ. J and DJ claims indicate a breakpoint between the start and end of the DEL/DUP.

- CNV: implicit abundance claim (SVCLAIM=D). SVCLAIM is optional but, if specified, must be D, or the missing value ".", which is equivalent to D.

- INV/INS: implicit adjacency claims (SVCLAIM=J). SVCLAIM is optional but, if specified, must be J, DJ, or the missing value ".", which is equivalent to J.

- breakend alleles are implicit adjacency claims (SVCLAIM=J). SVCLAIM is optional but, if specified, must be J, or the missing value ".", which is equivalent to J.

- Non-SV alleles should use the missing value ".".

If all alleles contain the missing value ".", the SVCLAIM field can be omitted.

Note that the scope of SVCLAIM is the single ALT allele taken in isolation. Complex events such as a cut and paste transposition of a mobile element result in multiple interrelated copy number and breakpoint claims. These records should be linked using EVENT/EVENTTYPE, with the SVCLAIM field on each record specifying the claim of that individual allele record. That is, a J claim must not be made on a CNV record even if the corresponding start/end breakpoints are known, and a DJ claim on a DEL or DUP is a claim of a simple, isolated deletion or tandem duplication.

```
##INFO=<ID=RN,Number=A,Type=Integer,Description="Total number of repeat sequences in this allele">
```

Used by <CNV:TR> tandem repeat alleles to encode the number of repeat sequences. This field determines the number of values encoded in the RUS, RUL, RB, CIRB, RUC and CIRUC fields for each allele. The length of these fields must match the sum of all RN values for the record. For the purposes of determining RUS, RUL, RB, CIRB, RUC and CIRUC lengths, the missing value "." should be treated as 0. If this field is missing, the RN for each <CNV:TR> allele is assumed to be 1 and 0 otherwise.

See section 5.7 for further details.

```
##INFO=<ID=RUS,Number=.,Type=String,Description="Repeat unit sequence of the corresponding repeat sequence">
##INFO=<ID=RUL,Number=.,Type=Integer,Description="Repeat unit length of the corresponding repeat sequence">
##INFO=<ID=RUC,Number=.,Type=Float,Description="Repeat unit count of corresponding repeat sequence">
##INFO=<ID=RB,Number=.,Type=Integer,Description="Total number of bases in the corresponding repeat sequence">
```

Used by <CNV:TR> tandem repeat alleles to encode information about the nature of the tandem repeats contained for ALT alleles. Conceptually, these fields each contain a list of values for each ALT allele. The length of these inner lists are determined by the RN field for that ALT allele and the length must match the sum of RN for the record. These fields contain the flattened and concatenated list contents in the same order as either corresponding ALT allele.

Each <CNV:TR> allele consists of RN repeat sequences each containing RUC repeat units with sequence RUS.

For example, if a <CNV:TR> allele sequence is $(CAG)_{10}(TG)_7(CAGG)_3$, the RN for that ALT allele would be 3, the RUS $CAG, TG, CAGG$, the RUL $3, 2, 4$, the RUC $10, 7, 3$ and the RB $30, 14, 12$.

RUS may contain only IUPAC nucleotide codes (ambiguous bases are allowed) or the missing value ('.'). If both RUS and RUL are present and not missing, the length of each RUS entry must match the corresponding RUL. If both RUC and RB are present and not missing, RB should approximately equal RUL*RUC but is not required to match exactly. If RUS is present but RUL is not, RUL should be inferred to be equal to the length of the corresponding RUS.

See section 5.7 for an example and further details.

```
##INFO=<ID=CIRUC,Number=.,Type=Float,Description="Confidence interval around RUC">
##INFO=<ID=CIRB,Number=.,Type=Integer,Description="Confidence interval around RB">
```

Confidence interval around RUC and RB respectively. The length of these fields must be twice that of their corresponding fields. CIRUC/CIRB must not be non-missing for any alleles with no corresponding RUC/RB value.

These fields are defined in the same manner as $CIPOS$ and contain the difference between the lower and upper confidence interval bounds and the value of the corresponding field. The lower bound must be less than or equal to zero and the upper bound must be greater than or equal to zero. If the lower bound is the missing value ".", is assumed to be 0 and if the upper bound is the missing value ".", it is assumed to be an unbounded estimate. That is, the length of repeat has been determined to be at least a certain length but a reasonable limit of total length of the repeat could not be determined.

See section 5.7 for an example and further details.

```
##INFO=<ID=RUB,Number=.,Type=Integer,Description="Number of bases in each individual repeat unit">
```

This field encodes the length of each repeat unit for situations where the length of each repeat unit varies greatly. If this field is present, RUC must also be present and must contain only integer values. This field uses the same list-of-list encoding as RUS/RUL/RUC/RB but contains a list for each RC entry, the length of which is determined

by the corresponding integer RUC value. This field contains the length of each individual repeat unit for each RUC entry. If RUB is missing or not specified, the RUB for each individual repeat unit is considered to be equal to the RUL for the corresponding repeat sequence.

For the vast majority of tandem repeats, this field can be omitted and is only required in complex situations such as when a VNTR contains one or more variable length STRs.

See section 5.7 for an example and further details.

4 FORMAT keys used for structural variants

```
##FORMAT=<ID=CN,Number=1,Type=Float,Description="Copy number">
##FORMAT=<ID=CICN,Number=2,Type=Float,Description="Confidence interval around copy number">
##FORMAT=<ID=CNQ,Number=1,Type=Float,Description="Copy number genotype quality">
##FORMAT=<ID=CNL,Number=G,Type=Float,Description="Copy number genotype likelihood">
##FORMAT=<ID=CNP,Number=G,Type=Float,Description="Copy number posterior probabilities">
##FORMAT=<ID=NQ,Number=1,Type=Integer,Description="Phred style probability score that the variant is novel">
##FORMAT=<ID=HAP,Number=1,Type=Integer,Description="Unique haplotype identifier">
##FORMAT=<ID=AHAP,Number=1,Type=Integer,Description="Unique identifier of ancestral haplotype">
```

CN specifies the total copy number over the region defined in the <CNV>, <DEL>, <DUP> alleles. When FORMAT CN is present, all <CNV>, <DEL> and <DUP> alleles must have the same SVLEN. CNQ, CNL, and CNP are analogous to GQ/GL/GP fields. CNQ is encoded as a phred quality $-10log_{10}$ p(copy number genotype call is wrong). CNL specifies a list of log_{10} likelihoods for each potential copy number, starting from zero. CNP is 0 to 1-scaled copy number posterior probabilities (and otherwise defined precisely as the CNL field), intended to store imputed genotype probabilities. When possible, GT/GQ/GL/GP should be used instead of (or in addition to) these keys.

5 Representing variation in VCF records

5.1 Creating VCF entries for SNPs and small indels

5.1.1 Example 1

For example, suppose we are looking at a locus in the genome:

Example	Sequence	Alteration
Ref	a t C g a	C is the reference base
1	a t G g a	C base is a G in some individuals
2	a t - g a	C base is deleted w.r.t. the reference sequence
3	a t CAg a	A base is inserted w.r.t. the reference sequence

Representing these as VCF records would be done as follows:

1. A SNP polymorphism of C/G at position 3 where C is the reference base and G is the alternate base becomes REF=C, ALT=G

2. A single base deletion of C at position 3 becomes REF=TC, ALT=T

3. A single base insertion of A after position 3 becomes REF=C, ALT=CA

Note that the positions must be sorted in increasing order:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	2	.	TC	T	.	PASS	DP=100
20	3	.	C	G	.	PASS	DP=100
20	3	.	C	CA	.	PASS	DP=100

5.1.2 Example 2

Suppose I see a the following in a population of individuals and want to represent these three segregating alleles:

Example	Sequence	Alteration
Ref	a t C g a	C is the reference base
1	a t G g a	C base is a G in some individuals
2	a t - g a	C base is deleted w.r.t. the reference sequence

In this case there are three segregating alleles: $\{tC, tG, t\}$ with a corresponding VCF record:

```
#CHROM   POS   ID   REF   ALT    QUAL   FILTER   INFO
20       2     .    TC    TG,T   .      PASS     DP=100
```

5.1.3 Example 3

Now suppose I have this more complex example:

Example	Sequence	Alteration
Ref	a t C g a	C is the reference base
1	a t - g a	C base is deleted w.r.t. the reference sequence
2	a t -- a	C and G bases are deleted w.r.t. the reference sequence
3	a t CAg a	A base is inserted w.r.t. the reference sequence

There are actually four segregating alleles: $\{tCg, tg, t, tCAg\}$ over bases 2–4. This complex set of allele is represented in VCF as:

```
#CHROM   POS   ID   REF   ALT          QUAL   FILTER   INFO
20       2     .    TCG   TG,T,TCAG    .      PASS     DP=100
```

Note that in VCF records, the molecular equivalence explicitly listed above in the per-base alignment is discarded, so the actual placement of equivalent g isn't retained. For completeness, VCF records are dynamically typed, so whether a VCF record is a SNP, Indel, Mixed, or Reference site depends on the properties of the alleles in the record.

5.2 Decoding VCF entries for SNPs and small indels

5.2.1 SNP VCF record

Suppose I receive the following VCF record:

```
#CHROM   POS   ID   REF   ALT   QUAL   FILTER   INFO
20       3     .    C     T     .      PASS     DP=100
```

This is a SNP since its only single base substitution and there are only two alleles so I have the two following segregating haplotypes:

Example	Sequence	Alteration
Ref	a t C g a	C is the reference base
1	a t T g a	C base is a T in some individuals

5.2.2 Insertion VCF record

Suppose I receive the following VCF record:

```
#CHROM   POS   ID   REF   ALT    QUAL   FILTER   INFO
20       3     .    C     CTAG   .      PASS     DP=100
```

This is a insertion since the reference base C is being replaced by C [the reference base] plus three insertion bases TAG. Again there are only two alleles so I have the two following segregating haplotypes:

Example	Sequence	Alteration
Ref	a t C - - - g a	C is the reference base
1	a t C T A G g a	following the C base is an insertion of 3 bases

5.2.3 Deletion VCF record

Suppose I receive the following VCF record:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	2	.	TCG	T	.	PASS	DP=100

This is a deletion of two reference bases since the reference allele TCG is being replaced by just the T [the reference base]. Again there are only two alleles so I have the two following segregating haplotypes:

Example	Sequence	Alteration
Ref	a T C G a	T is the (first) reference base
1	a T - - a	following the T base is a deletion of 2 bases

5.2.4 Mixed VCF record for a microsatellite

Suppose I receive the following VCF record:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	4	.	GCG	G,GCGCG	.	PASS	DP=100

This is a mixed type record containing a 2 base insertion and a 2 base deletion. There are are three segregating alleles so I have the three following haplotypes:

Example	Sequence	Alteration
Ref	a t c G C G - - a	G is the (first) reference base
1	a t c G - - - - a	following the G base is a deletion of 2 bases
2	a t c G C G C G a	following the G base is an insertion of 2 bases

Note that in all of these examples dashes have been added to make the haplotypes clearer but of course the equivalence among bases isn't provided by the VCF. Technically the following is an equivalent alignment:

Example	Sequence	Alteration
Ref	a t c G - - C G a	G is the (first) reference base
1	a t c G - - - - a	following the G base is a deletion of 2 bases
2	a t c G C G C G a	following the G base is an insertion of 2 bases

5.3 Encoding Structural Variants

The following page contains examples of structural variants encoded in VCF, showing in order:

1. A simple deletion

2. The same deletion in symbolic notation

3. The same deletion split into constituent breakpoint and copy number records

4. A poly A expansion

5. The same poly A expansion represented as a duplication

6. An insertion with both length and position imprecise

7. A single breakend

The sequence of $chrA$ in this example is ATGCGAAAAAAATGT.

VCF STRUCTURAL VARIANT EXAMPLE

```
##fileformat=VCFv4.4
##ALT=<ID=INV,Description="Inversion">
##ALT=<ID=INS,Description="Insertion">
##ALT=<ID=DUP,Description="Duplication">
##ALT=<ID=DUP:TANDEM,Description="Tandem Duplication">
##ALT=<ID=DEL,Description="Deletion">
##ALT=<ID=CNV,Description="Copy number variable region">
##INFO=<ID=MATEID,Number=A,Type=String,Description="ID of mate breakend">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the longest variant described in this record">
##INFO=<ID=CIPOS,Number=.,Type=Integer,Description="Confidence interval around POS for symbolic structural variants">
##INFO=<ID=SVLEN,Number=.,Type=Integer,Description="Length of structural variant">
##INFO=<ID=CILEN,Number=.,Type=Integer,Description="Confidence interval for the SVLEN field">
##INFO=<ID=EVENT,Number=A,Type=String,Description="ID of associated event">
##INFO=<ID=EVENTTYPE,Number=A,Type=String,Description="Type of associated event">
##INFO=<ID=CN,Number=A,Type=Float,Description="Copy number of CNV/breakpoint">
##INFO=<ID=SVCLAIM,Number=A,Type=String,Description="Claim made by the structural variant call. Valid values are D, J, DJ for abundance, adjacency and both respectively">
##INFO=<ID=IMPRECISE,Number=0,Type=Flag,Description="Imprecise structural variation">
##contig=<ID=chrA,length=1000000>
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
#CHROM POS    ID     REF ALT       QUAL FILTER INFO                                                      FORMAT sample
chrA   2      .      TGC T         .    .      EVENT=DEL_seq                                             GT     0/1
chrA   2      .      T   <DEL>     .    .      SVLEN=2;SVCLAIM=DJ;EVENT=DEL_symbolic;END=4               GT     0/1
chrA   2      delbp1 T   T[chrA:5[ .    .      MATEID=delbp2;EVENT=DEL_split_bp_cn                       GT     0/1
chrA   2      delbp2 A   ]chrA:2]A .    .      MATEID=delbp1;EVENT=DEL_split_bp_cn                       GT     0/1
chrA   2      .      T   <DEL>     .    .      SVLEN=2;SVCLAIM=D;EVENT=DEL_split_bp_cn;END=4             GT     0/1
chrA   5      .      G   GAAA      .    .      EVENT=homology_seq                                        GT     1/1
chrA   5      .      G   <DUP>     .    .      SVLEN=3;CIPOS=0,5;EVENT=homology_dup                      GT     0/1
chrA   14     .      T   <INS>     .    .      IMPRECISE;SVLEN=100;CILEN=-50,50;CIPOS=-10,10;END=14      GT     0/1
chrA   14     .      G   .CCCCCCG  .    .      EVENT=single_breakend                                     GT     0/1
```

21

5.4   Specifying complex rearrangements with breakends

An arbitrary rearrangement can be summarized as a set of novel **adjacencies**. Each adjacency ties together 2 **breakends**. The two breakends at either end of a novel adjacency are called **mates**.

There is one line of VCF (i.e. one record) for each of the two breakends in a novel adjacency. The REF field of a breakend record indicates a base or sequence s of bases beginning at position POS, as in all VCF records. The ALT field of a breakend record indicates a replacement for s. This "breakend replacement" has three parts:

1. The string t that replaces places s. The string t may be an extended version of s if some novel bases are inserted during the formation of the novel adjacency.

2. The position p of the mate breakend, indicated by a string of the form "chr:pos". This is the location of the first mapped base in the piece being joined at this novel adjacency.

3. The direction that the joined sequence continues in, starting from p. This is indicated by the orientation of square brackets surrounding p.

These 3 elements are combined in 4 possible ways to create the ALT. In each of the 4 cases, the assertion is that s is replaced with t, and then some piece starting at position p is joined to t. The cases are:

REF	ALT	Meaning
s	t[p[	piece extending to the right of p is joined after t
s	t]p]	reverse comp piece extending left of p is joined after t
s	]p]t	piece extending to the left of p is joined before t
s	[p[t	reverse comp piece extending right of p is joined before t

The example in Figure 1 shows a 3-break operation involving 6 breakends. It exemplifies all possible orientations of breakends in adjacencies. Notice how the ALT field expresses the orientation of the breakends.
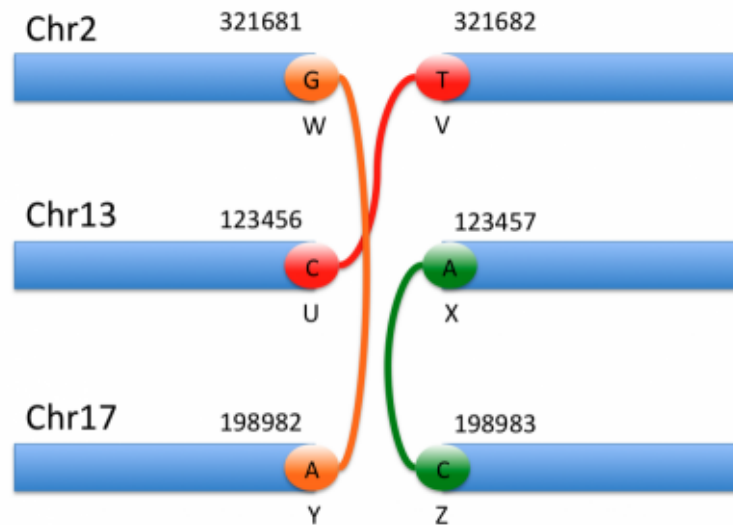
Figure 1: All possible orientations of breakends

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_W	G	G]17:198982]	6	PASS	.
2	321682	bnd_V	T	]13:123456]T	6	PASS	.
13	123456	bnd_U	C	C[2:321682[	6	PASS	.
13	123457	bnd_X	A	[17:198983[A	6	PASS	.
17	198982	bnd_Y	A	A]2:321681]	6	PASS	.
17	198983	bnd_Z	C	[13:123457[C	6	PASS	.

5.4.1 Inserted Sequence

Sometimes, as shown in Figure 2, some bases are inserted between the two breakends, this information is also carried in the ALT column:

Figure 2: Inserted sequence between breakends
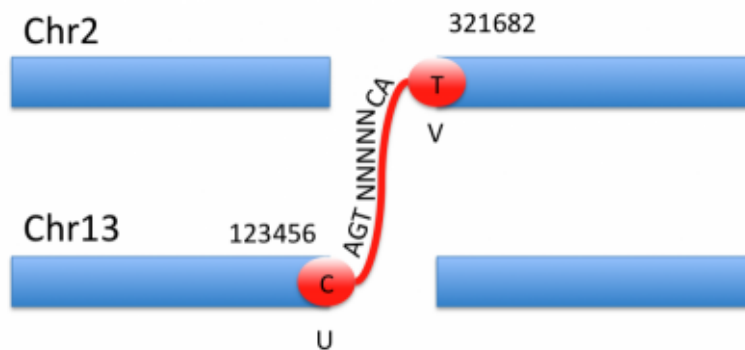
#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321682	bnd_V	T	]13 : 123456]AGTNNNNNCAT	6	PASS	MATEID=bnd_U
13	123456	bnd_U	C	CAGTNNNNNCA[2 : 321682[	6	PASS	MATEID=bnd_V

5.4.2 Large Insertions

If the insertion is too long to be conveniently stored in the ALT column, as in the 329 base insertion shown in Figure 3, it can be represented by a contig from the assembly file:
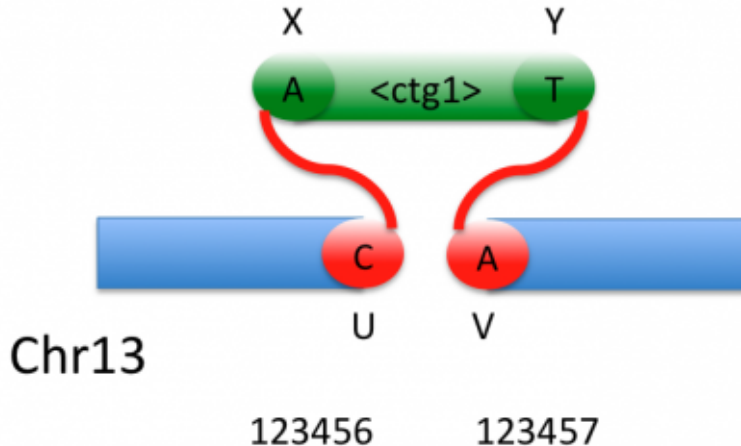
Figure 3: Inserted contig

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
13	123456	bnd_U	C	C[<ctg1>: 1[	6	PASS	MATEID=bnd_V
13	123457	bnd_V	A	] <ctg1 >: 329]A	6	PASS	MATEID=bnd_U

Note: In the special case of the complete insertion of a sequence between two base pairs, it is recommended to use the shorthand notation described below:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
13	123456	INS0	C	<ctg1 >	6	PASS	.

If only a portion of <ctg1 >, say from position 7 to position 214, is inserted, the VCF would be:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
13	123456	bnd_U	C	C[<ctg1>: 7[	6	PASS	MATEID=bnd_U
13	123457	bnd_V	A	] <ctg1 >: 214]A	6	PASS	MATEID=bnd_V

If <ctg1 > is circular and a segment from position 229 to position 45 is inserted, i.e., continuing from position 329 on to position 1, this is represented by adding a circular adjacency:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
13	123456	bnd_U	C	C[<ctg1 >: 229[	6	PASS	MATEID=bnd_U;EVENT=INS0
13	123457	bnd_V	A	] <ctg1 >: 45]A	6	PASS	MATEID=bnd_V;EVENT=INS0
<ctg1 >	1	bnd_X	A	] <ctg1 >: 329]A	6	PASS	MATEID=bnd_Y;EVENT=INS0
<ctg1 >	329	bnd_Y	T	T[<ctg1 >: 1[	6	PASS	MATEID=bnd_C;EVENT=INS0

5.4.3 Multiple mates

If a breakend has multiple mates such as in Figure 4 (either because of breakend reuse or of uncertainty in the measurement), these alternate adjacencies are treated as alternate alleles:
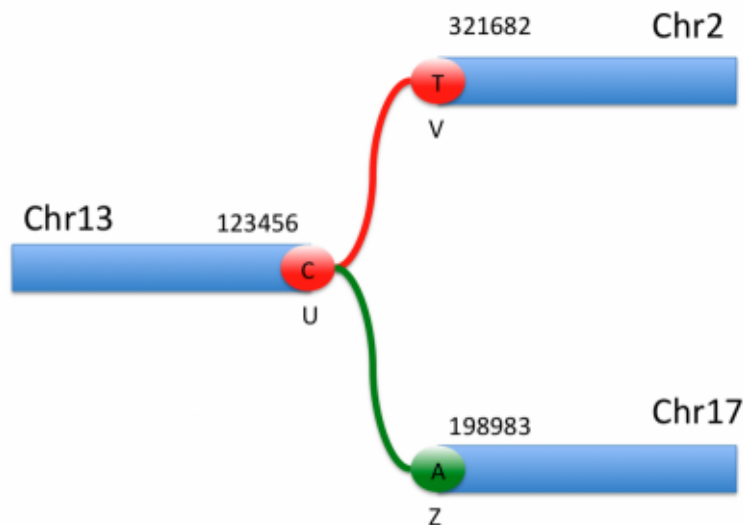
Figure 4: Breakend with multiple mates

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321682	bnd_V	T	]13 : 123456]T	6	PASS	MATEID=bnd_U
13	123456	bnd_U	C	C[2 : 321682[,C[17 : 198983[	6	PASS	MATEID=bnd_V,bnd_Z
17	198983	bnd_Z	A	]13 : 123456]A	6	PASS	MATEID=bnd_U

5.4.4 Explicit partners

Two breakends which are connected in the reference genome but disconnected in the variants are called partners. Each breakend only has one partner, typically one basepair left or right. However, it is not uncommon to observe loss of a few basepairs during the rearrangement. A breakend's partner may be explicitly named as in Figure 5:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_W	G	G[13 : 123460[	6	PASS	PARID=bnd_V;MATEID=bnd_X
2	321682	bnd_V	T	]13 : 123456]T	6	PASS	PARID=bnd_W;MATEID=bnd_U
13	123456	bnd_U	C	C[2 : 321682[	6	PASS	PARID=bnd_X;MATEID=bnd_V
13	123460	bnd_X	A	]2 : 321681]A	6	PASS	PARID=bnd_U;MATEID=bnd_W

5.4.5 Telomeres

For a rearrangement involving the telomere end of a reference chromosome, we define a virtual telomeric breakend that serves as a breakend partner for the breakend at the telomere. That way every breakend has a partner. If the
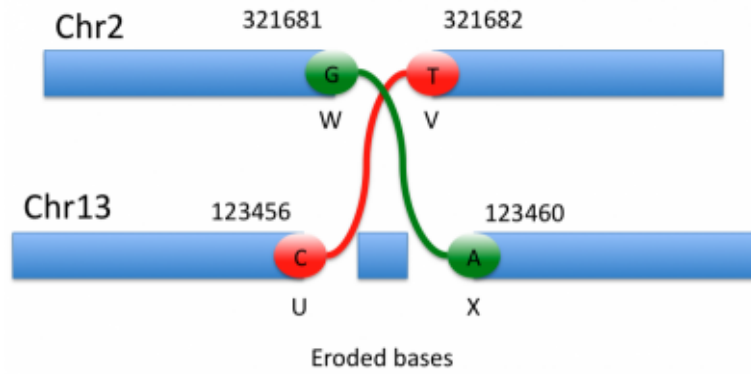
Figure 5: Partner breakends

chromosome extends from position 1 to N, then the virtual telomeric breakends are at positions 0 and N+1. For example, to describe the reciprocal translocation of the entire chromosome 1 into chromosome 13, as illustrated in Figure 6:
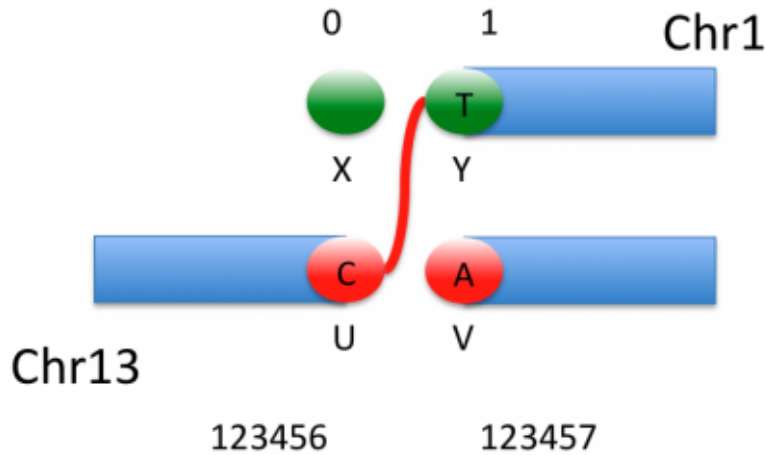

Figure 6: Telomeres

the records would look like:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
1	0	bnd_X	N	.[13 : 123457[	6	PASS	MATEID=bnd_V
1	1	bnd_Y	T	]13 : 123456]T	6	PASS	MATEID=bnd_U
13	123456	bnd_U	C	C[1 : 1[	6	PASS	MATEID=bnd_Y
13	123457	bnd_V	A	]1 : 0]A	6	PASS	MATEID=bnd_X

5.4.6 Event modifiers

As mentioned previously, a single rearrangement event can be described as a set of novel adjacencies. For example, a reciprocal rearrangement such as in Figure 7:
would be described as:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_W	G	G[13 : 123457[	6	PASS	MATEID=bnd_X;EVENT=RR0
2	321682	bnd_V	T	]13 : 123456]T	6	PASS	MATEID=bnd_U;EVENT=RR0
13	123456	bnd_U	C	C[2 : 321682[	6	PASS	MATEID=bnd_V;EVENT=RR0
13	123457	bnd_X	A	]2 : 321681]A	6	PASS	MATEID=bnd_W;EVENT=RR0

Figure 7: Rearrangements

Figure 8: Inversion

5.4.7 Inversions

Similarly an inversion such as in Figure 8:
can be described equivalently in two ways. Either one uses the short hand notation described previously (recommended for simple cases):

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	INV0	T	<INV>	6	PASS	END=421681

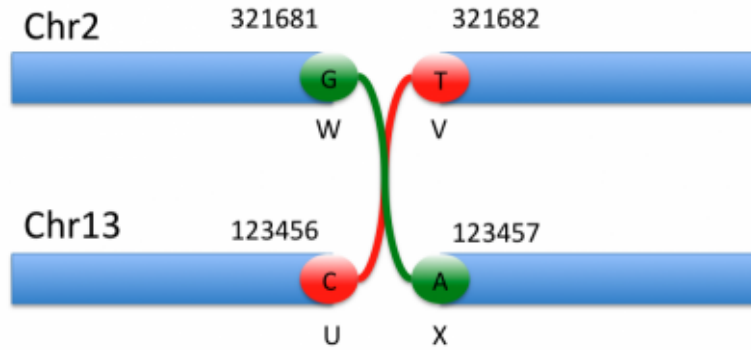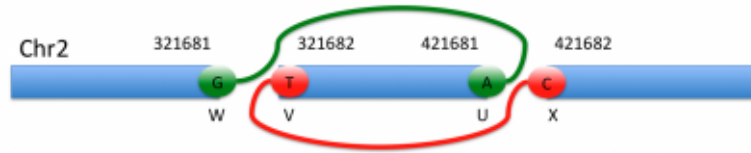or one describes the breakends:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_W	G	G]2 : 421681]	6	PASS	MATEID=bnd_U;EVENT=INV0
2	321682	bnd_V	T	[2 : 421682[T	6	PASS	MATEID=bnd_X;EVENT=INV0
2	421681	bnd_U	A	A]2 : 321681]	6	PASS	MATEID=bnd_W;EVENT=INV0
2	421682	bnd_X	C	[2 : 321682[C	6	PASS	MATEID=bnd_V;EVENT=INV0

5.4.8 Uncertainty around breakend location

It sometimes is difficult to determine the exact position of a break, generally because of homologies between the sequences being modified, such as in Figure 9. The breakend is then placed arbitrarily at the left most position, and the uncertainty is represented with the CIPOS tag. The ALT string is then constructed assuming this arbitrary breakend choice.

The figure above represents a nonreciprocal translocation with microhomology. Even if we know that breakend U is rearranged with breakend V, actually placing these breaks can be extremely difficult. The red and green dashed lines represent the most extreme possible recombination events which are allowed by the sequence evidence available. We therefore place both U and V arbitrarily within the interval of possibility:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_V	T	T]13 : 123462]	6	PASS	MATEID=bnd_U;CIPOS=0,6
13	123456	bnd_U	A	A]2 : 321687]	6	PASS	MATEID=bnd_V;CIPOS=0,6

Note that the coordinate in breakend U's ALT string does not correspond to the designated position of breakend V, but to the position that V would take if U's position were fixed (and vice-versa). The CIPOS tags describe the uncertainty around the positions of U and V.

The fact that breakends U and V are mates is preserved thanks to the MATEID tags. If this were a reciprocal translocation, then there would be additional breakends X and Y, say with X the partner of V on Chr 2 and Y
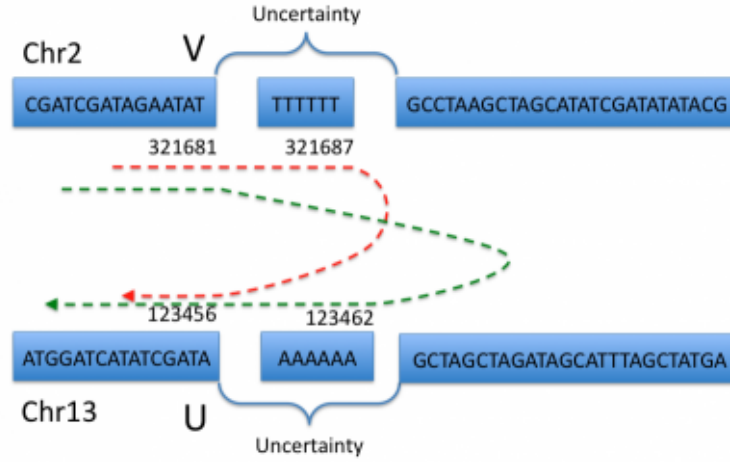
Figure 9: Homology

the partner of U on Chr 13, and there would be two more lines of VCF for the XY novel adjacency. Depending on which positions are chosen for the breakends X and Y, it might not be obvious that X is the partner of V and Y is the partner of U from their locations alone. This partner relationship can be specified explicitly with the tag PARID=bnd_X in the VCF line for breakend V and PARID=bnd_Y in the VCF line for breakend U, and vice versa.

5.4.9   Single breakends

We allow for the definition of a breakend that is not part of a novel adjacency. We call these single breakends, because they lack a mate. Breakends that are unobserved partners of breakends in observed novel adjacencies are one kind of single breakend. For example, if the true situation is known to be either as depicted back in Figure 1, and we only observe the adjacency (U,V), and no adjacencies for W, X, Y, or Z, then we cannot be sure whether we have a simple reciprocal translocation or a more complex 3-break operation. Yet we know the partner X of U and the partner W of V exist and are breakends. In this case we can specify these as single breakends, with unknown mates. The 4 lines of VCF representing this situation would be:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
2	321681	bnd_W	G	G.	6	PASS	.
2	321682	bnd_V	T	]13 : 123456]T	6	PASS	MATEID=bnd_U
13	123456	bnd_U	C	C[2 : 321682[	6	PASS	MATEID=bnd_V
13	123457	bnd_X	A	.A	6	PASS	.

On the other hand, if we know a simple reciprocal translocation has occurred as in Figure 7, then even if we have no evidence for the (W,X) adjacency, for accounting purposes an adjacency between W and X may also be recorded in the VCF file. These two breakends W and X can still be crossed-referenced as mates. The 4 VCF records describing this situation would look exactly as in section 5.4.4, but perhaps with a special quality or filter value for the breakends W and X.

Another possible reason for calling single breakends is an observed but unexplained change in copy number along a chromosome.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
3	12665	bnd_X	A	.A	6	PASS	CIPOS=-50,50
3	12665	.	A	<DUP>	14	PASS	END=13686;CIPOS=-50,50;CIEND=-50,50
3	13686	bnd_Y	T	T.	6	PASS	CIPOS=-50,50

Finally, if an insertion is detected but only the first few base-pairs provided by overhanging reads could be assembled, then this inserted sequence can be provided on that line, in analogy to paired breakends:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
3	12665	bnd_X	A	.TGCA	6	PASS	CIPOS=-50,50
3	12665	.	A	<DUP>	14	PASS	END=13686;CIPOS=-50,50;CIEND=-50,50
3	13686	bnd_Y	T	TCC.	6	PASS	CIPOS=-50,50

5.4.10 Sample mixtures

It may be extremely difficult to obtain clinically perfect samples, with only one type of cell. Let's imagine that two samples are taken from a cancer patient: healthy blood, and some tumor tissue with an estimated 30% stromal contamination. This would then be expressed in the header as:

```
##SAMPLE=<ID=Blood,Genomes=Germline,Mixture=1.,Description="Patient germline genome">
##SAMPLE=<ID=TissueSample,Genomes=Germline;Tumor,Mixture=.3;.7,Description="Patient germline genome;Patient tumor genome">
```

Because of this distinction between sample and genome, it is possible to express the data along both distinctions. For example, in a first pass, a structural variant caller would simply report counts per sample. Using the example of the inversion just above, the VCF code could become:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	Blood	TissueSample
2	321681	bnd_W	G	G]2 : 421681]	6	PASS	MATEID=bnd_U	GT:AD	0:32,0	0|1 : 9, 21
2	321682	bnd_V	T	[2 : 421682[T	6	PASS	MATEID=bnd_X	GT:AD	0:29,0	0|1 : 11, 25
13	421681	bnd_U	A	A]2 : 321681]	6	PASS	MATEID=bnd_W	GT:AD	0:34,0	0|1 : 10, 23
13	421682	bnd_X	C	[2 : 321682[C	6	PASS	MATEID=bnd_V	GT:AD	0:31,0	0|1 : 8, 20

However, a more evolved algorithm could attempt actually deconvolving the two genomes and generating copy number estimates based on the raw data:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	Blood	TumorSample
2	321681	bnd_W	G	G]2 : 421681]	6	PASS	MATEID=bnd_U	GT:CN	0:1	1:1
2	321682	bnd_V	T	[2 : 421682[T	6	PASS	MATEID=bnd_X	GT:CN	0:1	1:1
13	421681	bnd_U	A	A]2 : 321681]	6	PASS	MATEID=bnd_W	GT:CN	0:1	1:1
13	421682	bnd_X	C	[2 : 321682[C	6	PASS	MATEID=bnd_V	GT:CN	0:1	1:1

5.4.11 Clonal derivation relationships

In cancer, each VCF file represents several genomes from a patient, but one genome is special in that it represents the germline genome of the patient. This genome is contrasted to a second genome, the cancer tumor genome. In the simplest case the VCF file for a single patient contains only these two genomes. This is assumed in most of the discussion of the sections below.

In general there may be several tumor genomes from the same patient in the VCF file. Some of these may be secondary tumors derived from an original primary tumor. We suggest the derivation relationships between genomes in a cancer VCF file be represented in the header with PEDIGREE tags.

Analogously, there might also be several normal genomes from the same patient in the VCF (typically double normal studies with blood and solid tissue samples). These normal genomes are then considered to be derived from the original germline genome, which has to be inferred by parsimony.

The general format of a PEDIGREE line describing asexual, clonal derivation is:

```
##PEDIGREE=<ID=DerivedID,Original=OriginalID>
```

This line asserts that the DNA in genome DerivedID is asexually or clonally derived with mutations from the DNA in genome OriginalID. This is the asexual analog of the VCF format that has been proposed for family relationships between genomes, i.e., there is one entry per trio of the form:

```
##PEDIGREE=<ID=ChildID,Mother=MotherID,Father=FatherID>
```

Let's consider a cancer patient VCF file with 4 genomes: germline, primary_tumor, secondary_tumor1, and secondary_tumor2 as illustrated in Figure 10. The primary_tumor is derived from the germline and the secondary tumors are each derived independently from the primary_tumor, in all cases by clonal derivation with mutations. The PEDIGREE lines would look like:

```
##PEDIGREE=<ID=PrimaryTumorID,Original=GermlineID>
##PEDIGREE=<ID=Secondary1TumorID,Original=PrimaryTumorID>
##PEDIGREE=<ID=Secondary2TumorID,Original=PrimaryTumorID>
```

Alternately, if data on the genomes is compiled in a database, a simple pointer can be provided:

```
##pedigreeDB=URL
```
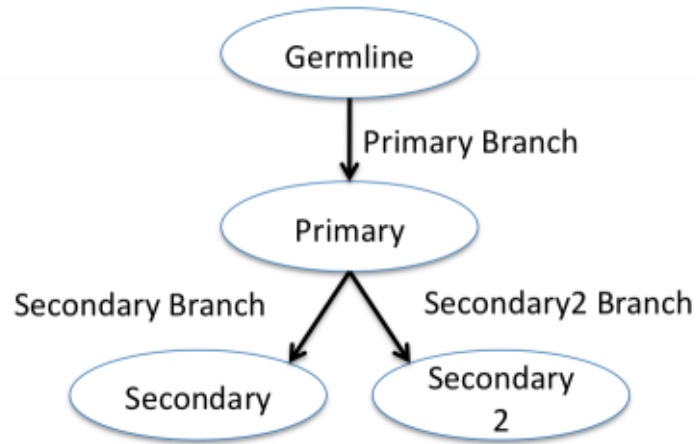
Figure 10: Pedigree example

The most general form of a pedigree line is:

```
##PEDIGREE=<ID=SampleID,Name_1=Ancestor1,...,Name_N=AncestorN>
```

This means that the genome SampleID is derived from the $N \geq 1$ genomes Ancestor1, ..., AncestorN. Based on these derivation relationships two new pieces of information can be specified.

Firstly, we wish to express the knowledge that a variant is novel to a genome, with respect to its parent genome. Ideally, this could be derived by simply comparing the features on either genomes. However, insufficient data or sample mixtures might prevent us from clearly determining at which stage a given variant appeared. This would be represented by a mutation quality score.

Secondly, we define a **haplotype** as a set of variants which are known to be on the same chromosome in the germline genome. Haplotype identifiers must be unique across the germline genome, and are conserved along clonal lineages, regardless of mutations, rearrangements, or recombination. In the case of the duplication of a region within a haplotype, one copy retains the original haplotype identifier, and the others are considered to be novel haplotypes with their own unique identifiers. All these novel haplotypes have in common their **haplotype ancestor** in the parent genome.

5.5   Representing unspecified alleles and REF-only blocks (gVCF)

In order to report sequencing data evidence for both variant and non-variant positions in the genome, the VCF specification allows to represent blocks of reference-only calls in a single record using the END INFO tag, an idea originally introduced by the gVCF file format[¶].

The convention adopted here is to represent reference evidence as likelihoods against an unknown alternate allele represented as $<*>$. Think of this as the likelihood for reference as compared to any other possible alternate allele (both SNP, indel, or otherwise). The $<*>$ representation is preferred over the symbolic allele $<$NON_REF$>$.

Example records are given below:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	Sample
1	4370	.	G	$<*>$	.	.	END=4383	GT:DP:GQ:MIN_DP:PL	0/0:25:60:23:0,60,900
1	4384	.	C	$<*>$	.	.	END=4388	GT:DP:GQ:MIN_DP:PL	0/0:25:45:25:0,42,630
1	4389	.	T	TC,$<*>$	213.73	.	.	GT:DP:GQ:PL	0/1:23:99:51,0,36,93,92,86
1	4390	.	C	$<*>$	.	.	END=4390	GT:DP:GQ:MIN_DP:PL	0/0:26:0:26:0,0,315
1	4391	.	C	$<*>$	.	.	END=4395	GT:DP:GQ:MIN_DP:PL	0/0:27:63:27:0,63,945
1	4396	.	G	C,$<*>$	0	.	.	GT:DP:GQ:P	0/0:24:52:0,52,95,66,95,97
1	4397	.	T	$<*>$	.	.	END=4416	GT:DP:GQ:MIN_DP:PL	0/0:22:14:22:0,15,593

[¶]https://help.basespace.illumina.com/articles/descriptive/gvcf-files/

5.6   Representing copy number variation

To encode copy number variation, VCF uses <CNV>, <DEL> and <DUP> symbolic structural variant alleles, CN INFO and FORMAT fields.

Allele specific copy number is specified through a <CNV> ALT allele for each distinct allelic copy number. INFO CN defines the allele specific copy number with FORMAT CN defining the overall copy number for that sample. POS and INFO SVLEN specify the genomic interval over which the copy number is defined. <DEL> and <DUP> copy number (SVCLAIM=D) alleles should be treated as <CNV> alleles that implicitly define INFO CN=0 and CN=2, respectively. As with all symbolic structural variants, the starting position of the interval is the base immediately after POS. For example, a region on chr1 from position 101 to 130 (both inclusive) with allele-specific copy numbers of 1 and 2 can be represented as follows:

```
chr1 100 . T <CNV>,<CNV> . . END=130;SVLEN=30,30;CN=1,2 GT:CN 1/2:3
```

All <CNV> alleles in the same VCF record should have the same SVLEN. To eliminate genotype ambiguity, copy number ALT alleles should not be mixed with other ALT alleles. When only copy number ALT alleles are present in a VCF record, GT=0 is equivalent to a <CNV> ALT allele with INFO CN of 1 and should be treated identically.

If only total copy number is known, the copy number of the segment should be defined with a single <CNV> ALT allele with a missing INFO CN field. In the above example this corresponds to the following:

```
chr1 100 . T <CNV> . . END=130;SVLEN=30 GT:CN .:3
```

The granularity of copy number representation is explicitly not defined in these specifications. Copy number segmentation can be base-pair accurate with even 1bp changes deletions resulting in new copy number segments, be at a highly granular megabase level of resolution, or anywhere in between. When the bounds of a copy number segment is not known precisely, this should be encoded in the CIPOS and CILEN INFO fields.

5.7 Representing tandem repeats

The repetitive nature of both Short Tandem Repeats (STRs) and Variable Number Tandem Repeats (VNTRs) makes identifying variants in these regions difficult. As a result, specialised techniques have been developed to estimate the length and composition of these regions. Many of these techniques result in imprecise variant calls which cannot be unambiguously represented with non-symbolic alleles. To accommodate such variant calls, a the <CNV:TR> symbolic allele can be used.

In general, tandem repeats can be represented in one or both of two complementary representations. When the exact sequence is known, the variant can be represented as a non-symbolic ALT allele. The variant can be represented either as a single VCF record containing the entire sequence in the ALT field, or over multiple phased records. When the exact sequence is not known, or when reporting tandem repeat 'summary' information, the variant can be represented as <CNV:TR> copy number variants with the RN, RS, RL, RB, RC and RUL INFO fields encoding the nature of the repeat expansion/contraction. The allele sequence of a <CNV:TR> record consists of the one or more repeat sequences, each of which consist of a single repeat unit repeated one or more times. That is, <CNV:TR> records can encode multiple different repeat motifs in a single allele but do not support "nested" repeats.

Figure 11 outlines the field encoding for a STR locus with alleles of sequence $CAGCAGCAGTTGTTG$ ($(CAG)_4(TTG)_2$), and $CACACA$ ($(CA)_3$). $RN = 2, 1$ indicates that the first allele has two repeat sequences and the second allele one. RUS, RUL, and RUC fields encode the repeat unit sequence, length, and count within each repeat sequence. RB encodes the length of each repeat sequence. For loci whether the repeat length variable (such as VNTRs containing STRs), RUB encodes the length of each individual repeat unit which, in this example, is unnecessary since they all match their corresponding RUL.
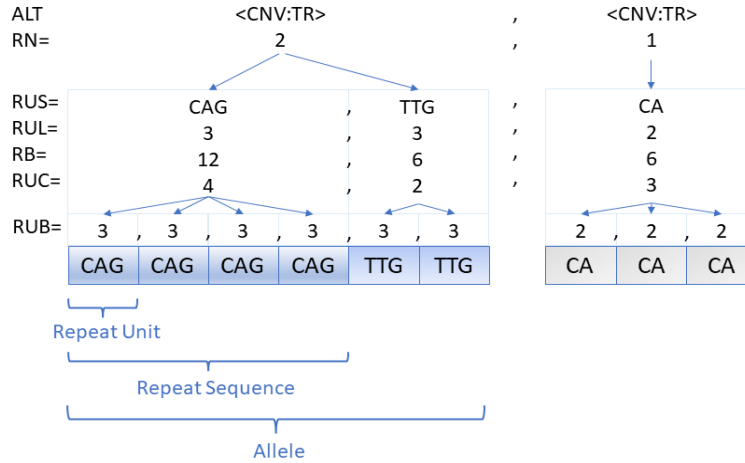
Figure 11: <CNV:TR> Tandem Repeats

A tandem repeat allele can be described by both a <CNV:TR> 'summary' record as well as non-symbolic records. When possible, these records should be phased with their corresponding <CNV:TR> record as outlined in the following example. In the following example, the reference genome contains a $(CAG)_{10}$ repeat at positions 101 to 130 inclusive, the first allele expands this to $(CAG)_{30}$ and the second allele is missing a G from the 6th repeat unit $(CAG)_5(CA)_1(CAG)_4$:

```
##fileformat=VCFv4.4
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the longest variant described in this record">
##INFO=<ID=SVLEN,Number=A,Type=Integer,Description="Length of structural variant">
##INFO=<ID=CN,Number=A,Type=Float,Description="Copy number of allele">
##INFO=<ID=RN,Number=A,Type=Integer,Description="Total number of repeat sequences in this allele">
##INFO=<ID=RUS,Number=.,Type=String,Description="Repeat unit sequence of the corresponding repeat sequence">
##INFO=<ID=RUL,Number=.,Type=Integer,Description="Repeat unit length of the corresponding repeat sequence">
##INFO=<ID=RUC,Number=.,Type=Float,Description="Repeat unit count of corresponding repeat sequence">
##INFO=<ID=RB,Number=.,Type=Integer,Description="Total number of bases in the corresponding repeat sequence">
##INFO=<ID=CIRUC,Number=.,Type=Float,Description="Confidence interval around RUC">
##INFO=<ID=CIRB,Number=.,Type=Integer,Description="Confidence interval around RB">
##INFO=<ID=RUB,Number=.,Type=Integer,Description="Number of bases in each individual repeat unit">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PS,Number=1,Type=Integer,Description="Phase set">
##FORMAT=<ID=CN,Number=1,Type=Float,Description="Copy number">
##ALT=<ID=CNV:TR,Description="Tandem repeat determined based on DNA abundance">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT sample
chr1 100 cnv_notation T <CNV:TR>,<CNV:TR> . . END=130;SVLEN=30,30;CN=3,0.9666;RUS=CAG,CAG,CA,CAG;RN=1,3;RB=90,15,2,12  GT:PS:CN 1|2:100:3.9666
chr1 117 precise_alt2 AG A . . GT:PS 0|1:100
chr1 130 precise_alt1 G GCAGCAGCAGCAGCAGCAGCAGCACGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG . . GT:PS 1|0:100
```

Note the following:

- As with all symbolic structural variant alleles, the POS of the <CNV:TR> record is the base immediately preceding the tandem repeat

- The SVLEN of the <CNV:TR> is the length of the reference allele. It is not the length of the <CNV:TR> allele.

- The SVLEN of the <CNV:TR> allele of a novel (with respect to the reference) tandem repeat should be 1.

- The POS of the <CNV:TR> allele of a novel (with respect to the reference) tandem repeat should be the base immediately preceding the inserted tandem repeat sequence.

- Both a <CNV:TR> and one or more non-symbolic records encoding the tandem repeat can be present.

- <CNV:TR> and the non-symbolic records encoding the tandem repeat should be phased if possible.

- When both <CNV:TR> and the equivalent non-symbolic records are present, the <CNV:TR> should approximately encode the sequence but is not required to encode the sequence exactly. For example, SNVs and indels may be omitted in the <CNV:TR> record.

- Variant callers which do not report allele-specific tandem repeats should use a single <CNV:TR> ALT allele and the missing genotype for the GT field (for example, ./. if diploid).

- The INFO and FORMAT CN fields should be present for <CNV:TR> records (as they are <CNV> records) and, when present, must correspond to the sample allelic length divided by the reference allelic length. Note that CN FORMAT field represents the overall copy number and the INFO CN the allele-specific copy number.

- When benchmarking tandem repeats, the <CNV:TR> interval provides a region over which a set of (preferably phased) non-symbolic records can be compared against for length and sequence composition.

- RN encodes the number of records for each allele in the RUS, RUL, RUC, RB fields. Conceptually, this a mechanism to encode a list-of-list with flat list format used in VCF.

- STRs should use RUS, whereas VNTRs should use RUL to ensure the VCF records are not excessively large.

- RUL should be omitted when RUS is present (as it is redundant when RS is present).

- RUS or RUL must be specified for each <CNV:TR>.

- Support for multiple levels of repeat nesting (such as STRs within VNTRs) is limited to the RUL repeat unit length field which allows the overall length of each top-level repeat unit to be encoded.

- The POS and END of <CNV:TR> records should match the STR/VNTR reference catalog sizes for catalog-based callers.

- Variant normalisation has limited utility in regions of low complexity as almost identical haplotypes can have very different normalised representations.

In some cases, it is desirable to report the full repeat sequence of all alleles at a given repeat locus in a single VCF records. There are no restrictions on doing so and in the above example, instead of reporting *precise_alt*1 and *precise_alt*2, the variants can be represented directly in a single record with a *REF* of:

`CAGCAGCAGCAGCAGCAGCAGCAGCAGCAG`

and an *ALT* of:

`CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG,CAGCAGCAGCAGCAGCACAGCAGCAGCAG`

When the length or number of repeat units in a repeat sequence cannot be determined precisely, CIRB and/or CIRUC can be used to define the bounds. For example, if the total number of CAG repeats at the above locus was at least 50 $((CAG)_{50-})$ and the mostly likely number of repeats was 65, then the <CNV:TR> could be encoded as follows:

`chr1 100 . T <CNV:TR> . . END=130;SVLEN=30;CN=6.5;RUS=CAG;RUC=65;CIRUC=-15,. GT ./.`

Note that:

- RN was omitted as it is only required if at least one <CNV:TR> allele has RN greater than 1.

- The confidence interval bounds are relative to the nominal value.

- A missing upper bound indicates the maximum length is not known.

Exactly representing nested repeats results in the loss of some repeat information when representing with a <CNV:TR> record. For repeats such as $((ACCGGC)_4(ACCAGT))_{3-5}$, summarising the repeat structure in a <CNV:TR> record requires either unrolling the inner repeats, or treating each outer repeat as a separate repeat sequence (the full repeat structure can be stored in a caller-specific non-standard INFO field). For many VNTRs, the critical information to retain is the length of each repeat unit. This length information can be encoded in the RUB field. For example, a 10,000bp VNTRs domain repeated 5 times, each repeat 500bp longer than the previous can be encoded as follows:

```
chr1 1000000 . T <CNV:TR> . . END=20000;SVLEN=20000;CN=1.25;RUL=10000;RUC=5;RUB=10000,10500,11000,11500,12000 GT ./.
```

6   BCF specification

VCF is very expressive, accommodates multiple samples, and is widely used in the community. Its biggest drawback is that it is big and slow. Files are text and therefore require a lot of space on disk. A normal batch of a hundred exomes is a few gigabytes, but large-scale VCFs with thousands of exome samples quickly become hundreds of gigabytes. Because the file is text, it is extremely slow to parse.

Overall, the idea behind is BCF2 is simple. BCF2 is a binary, compressed equivalent of VCF that can be indexed with tabix and can be efficiently decoded from disk or streams. For efficiency reasons BCF2 only supports a subset of VCF, in that all info and genotype fields must have their full types specified. That is, BCF2 requires that if e.g. an info field `AC` is present then it must contain an equivalent VCF header line noting that `AC` is an allele indexed array of type integer.

6.1   Overall file organization

A BCF2 file is composed of a mandatory header, followed by a series of BGZF compressed blocks of binary BCF2 records. The BGZF blocks allow BCF2 files to be indexed with tabix.

BGZF blocks are composed of a VCF header with a few additional records and a block of records. Following the last BGZF BCF2 record block is an empty BGZF block (a block containing zero type of data), indicating that the records are done.

A BCF2 header follows exactly the specification as VCF, with a few extensions/restrictions:

- All BCF2 files must have fully specified contigs definitions. No record may refer to a contig not present in the header itself.

- All INFO and GENOTYPE fields must be fully typed in the BCF2 header to enable type-specific encoding of the fields in records. An error must be thrown when converting a VCF to BCF2 when an unknown or not fully specified field is encountered in the records.

6.2   Header

The BCF2 header contains the following items:

Field	Type	Notes
magic	char[3]	The characters "BCF"
major_version	uint8_t	2
minor_version	uint8_t	2
l_text	uint32_t	Length of the "text" field, including the terminating NUL character
text	char[l_text]	VCF format header text, NUL-terminated

The "magic" field and version numbers can be used to quickly examine the file to determine that it's a BCF2.2 file. The "text" field contains the standard VCF header lines in text format, from `##fileformat=VCFv4.4` to `#CHROM ...` inclusive, terminated by a NUL character.

Because the type is encoded directly in the header, the recommended extension for BCF2 formatted files is *.bcf*. BCF2 supports encoding values in a dictionary of strings. The string map is provided by the keyword `##dictionary=S0,S1,...,SN` as a comma-separate ordered list of strings. See the "Dictionary of strings" section for more details.

6.2.1   Dictionary of strings

Throughout the BCF file most string values are be specified by integer reference to their dictionary values. For example, the following VCF record:

```
##INFO=<ID=ASP,Number=0,Type=Flag,Description="X">
##INFO=<ID=RSPOS,Number=1,Type=Integer,Description="Y">
##INFO=<ID=dbSNPBuildID,Number=1,Type=Integer,Description="Z">
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens">
#CHROM POS ID REF ALT QUAL FILTER INFO
20 10144 rs144773400 TA T . PASS ASP;RSPOS=10145,dbSNPBuildID=134
20 10228 rs143255646 TA T . PASS ASP;RSPOS=10229;dbSNPBuildID=134
```

would be encoded inline in BCF2 by reference to the relative position of the header line in the header (ASP=1, RSPOS=2, dbSNPBuildID=3, and PASS implicitly encoded in the first offset PASS=0).

```
##INFO=<ID=ASP,Number=0,Type=Flag,Description="X">
##INFO=<ID=RSPOS,Number=1,Type=Integer,Description="Y">
##INFO=<ID=dbSNPBuildID,Number=1,Type=Integer,Description="Z">
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens">
#CHROM POS ID REF ALT QUAL FILTER INFO
0 10144 rs144773400 TA T . s0 s1;s2=10145;s3=134
0 10228 rs143255646 TA T . s0 s1;s2=10229;s3=134
```

Defined this way, the dictionary of strings depends on the order and the presence of all preceding header lines. If an existing tag needs to be removed from a BCF, also all consequent tags throughout the whole BCF would have to be recoded. In order to avoid this costly operation, a new IDX field can be used to explicitly define the position which is dropped on BCF-to-VCF conversion. If not present, the implicit relative position is assumed. If the IDX field is present in one record, it must be present also in all other dictionary-defining records. The IDX tag is not necessary in newly created BCF files, but if present, the numbering must match the implicit dictionary of tags.

Note that the dictionary encoding has the magic prefix 's' here to indicate that the field's value is actually in the dictionary entry giving by the subsequent offset. This representation isn't actually the one used in BCF2 records but it provides a clean visual guide for the above example. Note also how the contig has been recoded as a offset into the list of contig declarations.

Note that "PASS" is always implicitly encoded as the first entry in the header dictionary. This is because VCF allows FILTER fields to be PASS without explicitly listing this in the FILTER field itself.

6.2.2 Dictionary of contigs

The CHROM field in BCF2 is encoded as an integer offset into the list of `##contig` field headers in the VCF header. The offsets begin, like the dictionary of strings, at 0. So for example if in BCF2 the contig value is 10, this indicates that the actual chromosome is the 11th element in the ordered list of `##contig` elements. Here's a more concrete example:

```
##contig=<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species="Homo sapiens">
##contig=<ID=21,length=46944323,assembly=B36,md5=f1b74b7f9f4cdbaeb6832ee86cb426c6,species="Homo sapiens">
##contig=<ID=22,length=49691432,assembly=B36,md5=2041e6a0c914b48dd537922cca63acb8,species="Homo sapiens">
#CHROM POS ID REF ALT QUAL FILTER INFO
20 1 . T A . PASS .
21 2 . T A . PASS .
22 3 . T A . PASS .
```

the actual CHROM field values in the encoded BCF2 records would be 0, 1, and 2 corresponding to the first (offset 0) `##contig` element, etc.

6.3 BCF2 records

In BCF2, the original VCF records are converted to binary and encoded as BGZF blocks. Each record is conceptually two parts. First is the site information (chr, pos, INFO field). Immediately after the sites data is the genotype data for every sample in the BCF2 file. The genotype data may be omitted entirely from the record if there is no genotype data in the VCF file. Compression of a BCF file is recommended but not required.

6.3.1 Site encoding

Field	Type	Notes
l_shared	uint32_t	Data length from CHROM to the end of INFO
l_indiv	uint32_t	Data length of FORMAT and individual genotype fields
CHROM	int32_t	Given as an offset into the mandatory contig dictionary
POS	int32_t	0-based leftmost coordinate
rlen	int32_t	Length of the record as projected onto the reference sequence. Must be the maximum of the length of the REF allele and the lengths inferred from the SVLEN/END of any symbolic alleles
QUAL	float	Variant quality; 0x7F800001 for a missing value
n_info	uint16_t	The number of INFO fields in this record
n_allele	uint16_t	The number of REF+ALT alleles in this record
n_sample	uint24_t	The number of samples in this record, stored as a **three** byte little-endian value. Note that n_sample must be equal to the number of samples in the header
n_fmt	uint8_t	The number of FORMAT keys. See 6.3.2
ID	typed string	Variant identifier; 0x07 for a missing value
REF+ALT	list of n_allele typed strings	the first allele is REF (mandatory) followed by n_alleles - 1 ALT alleles, all encoded as typed strings
FILTER	Typed vector of integers	a vector of integer offsets into dictionary, one for each FILTER field value. "." is encoded as MISSING
INFO	field key/value pairs	n_info pairs of typed vectors. The first value must be a typed atomic integer giving the offset of the INFO field key into the dictionary. The second value is a typed vector giving the value of the field
Genotype values	see below	see below

6.3.2 Genotype encoding

Genotype fields are encoded not by sample as in VCF but rather by field, with a vector of values for each sample following each field. In BCF2, the following VCF line:

```
FORMAT     NA00001   NA00002   NA00003
GT:GQ:DP   0/0:48:1  0/1:9:8   1/1:43:5
```

would encoded as the equivalent of:

```
GT=0/0,0/1,1/1   GQ=48,9,43   DP=1,8,5
```

Suppose there are i genotype fields in a specific record. Each i is encoded by a triplet:
BCF2 site information encoding

Field	Type	Notes
fmt_key	typed int	Format key as an offset into the dictionary
fmt_type	uint8_t+	Typing byte of each individual value, possibly followed by a typed int for the vector length. In effect this is the same as the typing value for a single vector, but for genotype values it appears only once before the array of genotype field values
fmt_values (by fmt type)	Array of values	The information of each individual is concatenated in the vector. Every value is of the same fmt type. Variable-length vectors are padded with END_OF_VECTOR values; a string is stored as a vector of char

The value is always implicitly a vector of N values, where N is the number of samples. The type byte of the value field indicates the type of each value of the N length vector. For atomic values this is straightforward (size = 1). But if the type field indicates that the values are themselves vectors (as often occurs, such as with the PL field) then each of the N values in the outer vector is itself a vector of values. This encoding is efficient when every value in the genotype field vector has the same length and type.

It is recommended to respect the ordering as specified in the input VCF/BCF2 file, but parsers should not rely on a specific ordering.

If there are no sample records (genotype data) in this VCF/BCF2 file, the size of the genotypes block will be 0.

6.3.3 Type encoding

In BCF2 values are all strongly typed in the file. The type information is encoded in a prefix byte before the value, which contains information about the low-level type of the value(s) such as int32 or float, as well as the number of elements in the value. The encoding is as follows:

BCF2 type descriptor byte

Bit	Meaning
5,6,7,8 bits	The number of elements of the upcoming type. For atomic values, the size must be 1. If the size is set to 15, this indicates that the vector has 15 or more elements, and that the subsequent BCF2 byte stream contains a typed Integer indicating the true size of the vector. If the size is between 2–14, then this Integer is omitted from the stream and the upcoming stream begins immediately with the first value of the vector. A size of 0 indicates that the value is MISSING.
1,2,3,4 bits	Type

The final four bits encodes an unsigned integer that indicates the type of the upcoming value in the data stream.
BCF2 types

Lowest 4 bits	Hexadecimal encoding	Corresponding atomic type
0	0x00	A MISSING value without an explicit type provided
1	0x?1	Integer [8 bit]
2	0x?2	Integer [16 bit]
3	0x?3	Integer [32 bit]
5	0x?5	Float [32 bit]
7	0x?7	Character, ASCII encoded in 8 bits

In BCF2 strings are type=7 with an appropriate length. Single characters are represented as strings with length=1. Type codes 4,6,8–15 are reserved for future use.

Integers may be encoded as 8, 16, or 32 bit values, in little-endian order. It is up to the encoder to determine the appropriate ranged value to use when writing the BCF2 file. For integer types, the values 0x80, 0x8000, 0x80000000 are interpreted as missing values and 0x81, 0x8001, 0x80000001 as END_OF_VECTOR indicators (for 8, 16, and 32 bit values, respectively). Note that the END_OF_VECTOR byte is not part of the vector itself and only END_OF_VECTOR bytes can follow. In total, eight values are reserved for future use: 0x80–0x87, 0x8000–0x8007, 0x80000000–0x80000007.

Floats are encoded as single-precision (32 bit) in the basic format defined by the IEEE-754-1985 standard. This is the standard representation for floating point numbers on modern computers, with direct support in programming languages like C and Java (see Java's Double class for example). BCF2 supports the full range of values from -Infinity to +Infinity, including NaN. BCF2 needs to represent missing values for single precision floating point numbers. This is accomplished by writing the NaN value as the quiet NaN (qNaN), while the MISSING value is encoded as a signaling NaN. From the NaN wikipedia entry, we have:

> For example, a bit-wise example of a IEEE floating-point standard single precision (32-bit) NaN would be: s111 1111 1axx xxxx xxxx xxxx xxxx xxxx where s is the sign (most often ignored in applications), a determines the type of NaN, and x is an extra payload (most often ignored in applications). If a = 1, it is a quiet NaN; if a is zero and the payload is nonzero, then it is a signaling NaN.

A good way to understand these values is to play around with the IEEE encoder website.

Similarly to integers, the float value of 0x7F800001 is interpreted as a MISSING value and 0x7F800002 as the END_OF_VECTOR indicator. Note that the END_OF_VECTOR byte is not part of the vector itself and only END_OF_VECTOR bytes can follow. In total, eight values are reserved for future use:

Value	32-bit precision	Hexadecimal representation
NaN	0b0111 1111 1100 0000 0000 0000 0000 0000	0x7FC00000
MISSING	0b0111 1111 1000 0000 0000 0000 0000 0001	0x7F800001
END_OF_VECTOR	0b0111 1111 1000 0000 0000 0000 0000 0010	0x7F800002
reserved	0b0111 1111 1000 0000 0000 0000 0000 0011	0x7F800003
. . .	. . .	. . .
reserved	0b0111 1111 1000 0000 0000 0000 0000 0111	0x7F800007

Character values are not explicitly typed in BCF2. Instead, VCF Character values must be encoded by a single character string. See also 1.2.

Flags values — which can only appear in INFO fields — in BCF2 should be encoded by any non-reserved value. The recommended best practice is to encode the value as an 1-element INT8 (type 0x11) with value of 1 to indicate present. Because FLAG values can only be encoded in INFO fields, BCF2 provides no mechanism to encode FLAG values in genotypes, but could be easily extended to do so if allowed in a future VCF version.

String values have two basic encodings. For INFO, FORMAT, and FILTER keys these are encoded by integer offsets into the header dictionary. For string values, such as found in the ID, REF, ALT, INFO, and FORMAT fields, strings are encoded as typed array of ASCII encoded bytes. The array isn't terminated by a NUL byte. The length of the string is given by the length of the type descriptor.

Suppose you want to encode the string "`ACAC`". First, we need the type descriptor byte, which is the string type 0x07 or'd with inline size (4) yielding the type byte of 0x40 | 0x07 = 0x47. Immediately following the type byte is the four byte ASCII encoding of "`ACAC`": 0x41 0x43 0x41 0x43. So the final encoding is:

0x47 0x41 0x43 0x41 0x43	String type with inline size of 4 followed by ACAC in ASCII

Suppose you want to encode the string "`VariantCallFormatSampleText`", a string of size 27. First, we need the type descriptor byte, which is the string type 0x07. Because the size exceeds the inline size limit (27 ≥ 15) we set the size to overflow, yielding the type byte of 0xF0 | 0x07 = 0xF7. Immediately following the type byte is the typed size of 27, which we encode by the atomic INT8 value: 0x11 followed by the actual size 0x1B. Finally comes the actual bytes of the string: 0x56 0x61 0x72 0x69 0x61 0x6E 0x74 0x43 0x61 0x6C 0x6C 0x46 0x6F 0x72 0x6D 0x61 0x74 0x53 0x61 0x6D 0x70 0x6C 0x65 0x54 0x65 0x78 0x74. So the final encoding is:

0xF7	string with overflow size
0x11 0x1B	overflow size encoded as INT8 with value 27
0x56 0x61 0x72 0x69 0x61 0x6E 0x74 0x43 0x61 0x6C 0x6C 0x46 0x6F 0x72 0x6D 0x61 0x74 0x53 0x61 0x6D 0x70 0x6C 0x65 0x54 0x65 0x78 0x74	message in ASCII

Suppose you want to encode the missing value '.'. This is simply a string of size 0 = 0x07.

In VCF there are sometimes fields of type list of strings, such as a number field of unbounded size encoding the amino acid changes due to a mutation. Since BCF2 doesn't directly support vectors of strings (a vector of character is already a string) we collapse the list of strings into a single comma-separated string, encode it as a regular BCF2 vector of characters, and on reading explode it back into the list of strings. This works because strings in VCF cannot contain ',' (it's a field separator) and so we can safely use ',' to separate the individual strings.

Vectors — The BCF2 type byte may indicate that the upcoming data stream contains not a single value but a fixed length vector of values. The vector values occur in order (1st, 2nd, 3rd, etc) encoded as expected for the type declared in the vector's type byte. For example, a vector of 3 16-bit integers would be laid out as first the vector type byte, followed immediately by 3 2-byte values for each integer, including a total of 7 bytes.

Missing values in vectors are handled slightly differently from atomic values. There are two possibilities for missing values:

One (or more) of the values in the vector may be missing, but others in the vector are not. Here each value should be represented in the vector, and each corresponding BCF2 vector value either set to its present value or the type equivalent MISSING value. Alternatively the entire vector of values may be missing. In this case the correct encoding is as a type byte with size 0 and the appropriate type MISSING. Suppose we are encoding the record "AC=[1,2,3]" from the INFO field. The AC key is encoded in the standard way. This would be immediately followed by a typed 8-bit integer vector of size 3, which is encoded by the type descriptor 0x31. The type descriptor is immediately

followed by the three 8-bit integer values: 0x01 0x02 0x03, for a grand total of 4 bytes: 0x31010203.

Suppose we are at a site with many alternative alleles so AC=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]. Since there are 16 values, we have to use the long vector encoding. The type of this field is 8 bit integer with the size set to 15 to indicate that the size is the next stream value, so this has type of 0xF1. The next value in the stream is the size, as a typed 8-bit atomic integer: 0x11 with value 16 0x10. Each integer AC value is represented by it's value as a 8 bit integer. The grand total representation here is:

0xF1 0x01 0x10	8 bit integer vector with overflow size
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x10	1–16 as hexadecimal 8 bit integers

Suppose this INFO field contains the "AC=.", indicating that the AC field is missing from a record with two alt alleles. The correct representation is as the typed pair of AC followed by a MISSING vector of type 8-bit integer: 0x01.

Vectors of mixed length — In some cases genotype fields may be vectors whose length differs among samples. For example, some CNV call sets encode different numbers of genotype likelihoods for each sample, given the large number of potential copy number states, rather padding all samples to have the same number of fields. For example, one sample could have CN0:0,CN1:10 and another CN0:0,CN1:10,CN2:10. In the situation when a genotype field contain vector values of different lengths, these are represented in BCF2 by a vector of the maximum length per sample, with all values in the each vector aligned to the left, and END_OF_VECTOR values assigned to all values not present in the original vector. The BCF2 encoder / decoder must automatically add and remove these END_OF_VECTOR values from the vectors. Note that the use of END_OF_VECTOR means that it is legal to encode a vector VCF field with MISSING values.

For example, suppose I have two samples, each with a FORMAT field X. Sample A has values [1], while sample B has [2,3]. In BCF2 this would be encoded as [1, END_OF_VECTOR] and [2, 3]. Diving into the complete details, suppose X is at offset 3 in the dictionary, which is encoded by the typed INT8 descriptor 0x11 followed by the value 0x03. Next we have the type of the each format field, which here is a 2 element INT8 vector: 0x21. Next we have the encoding for each sample, A = 0x01 0x81 followed by B = 0x02 0x03. All together we have:

0x11 0x03	X dictionary offset
0x21	each value is a 2 element INT8 value
0x01 0x81	A is [1, END_OF_VECTOR]
0x02 0x03	B is [2, 3]

A **Genotype (GT) field** is encoded in a typed integer vector (can be 8, 16, or even 32 bit if necessary) with the number of elements equal to the maximum ploidy among all samples at a site. For one individual, each integer in the vector is organized as $(allele + 1) << 1 \mid phased$ where allele is set to -1 if the allele in GT is a dot '.' (thus the higher bits are all 0). The vector is padded with the END_OF_VECTOR values if the GT having fewer ploidy. We note specifically that except for the END_OF_VECTOR byte, no other negative values are allowed in the GT array. When processing VCF version 4.3 or earlier files, the phasing of the first allele should be treated as missing and inferred from the remaining alleles.

Examples:

0/1	in standard format $(0 + 1) << 1 \mid 0$ followed by $(1 + 1) << 1 \mid 0$	0x02 04
0/1, 1/1, and 0/0	three samples encoded consecutively	0x02 04 04 04 02 02
0 | 1	$(1 + 1) << 1 \mid 1 = $ 0x05 preceded by the phased first byte value 0x03	0x03 05
./.	where both alleles are missing	0x00 00
0	as an implicitly phased haploid it is represented by a single byte	0x03
1	as an implicitly phased haploid it is represented by a single byte	0x05
0/1/2	is tetraploid, with alleles	0x02 04 06
0/1 | 2	is tetraploid with a single phased allele	0x02 04 07
0 and 0/1	pad out the final allele for the haploid individual	0x02 81 02 04

The final example is something seen on chrX when we have a haploid male and a diploid female. The male genotype vector is terminated prematurely by the END_OF_VECTOR value.

6.4 Encoding a VCF record example

Let's encode a realistic (but made-up) VCF record. This is a A/C SNP in HM3 (not really) called in 3 samples. In this section we'll build up the BCF2 encoding for this record.

```
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA00001 NA00002 NA00003
chr1 101 rs123 A C 30.1 PASS HM3;AC=3;AN=6;AA=C GT:GQ:DP:AD:PL 0/0:10:32:32,0:0,10,100 0/1:10:48:32,16:10,0,100 1/1:10:64:0,64:100,10,0
```

6.4.1 Encoding CHROM and POS

First, let's assume that `chr1` is the second chromosome to appear in the contig list—right after `chrM` (MT). So its offset is 1. The `POS` BCF2 field value is 101 (obviously). Because these are both typed values in the BCF2 record, we encode both in their most compact 8-bit value form. The type byte for an atomic 8-bit integer is 0x11. The value for the contig offset is 1 = 0x01. The value 101 is encoded as the single byte 0x65. So in total these are represented as:

0x01000000	CHROM offset is at 1 in 32 bit little endian
0x64000000	POS in 0 base 32 bit little endian
0x01000000	rlen = 1 (it's just a SNP)

6.4.2 Encoding QUAL

The QUAL field value is 30.1, which we encode as an untyped single precision 32-bit float:

0x41 0xF0 0xCC 0xCD	QUAL = 30.1 as 32-bit float

6.4.3 Encoding ID

This ID value is a 5-element string, so is encoded as type descriptor 0x57 followed by the five bytes for the string of `0x72 0x73 0x31 0x32 0x33`. The full encoding is:

0x57 0x72 0x73 0x31 0x32 0x33	ID = rs123

6.4.4 Encoding REF/ALT fields

We encode each of REF and ALT as typed strings, first REF followed immediately by ALT. Each is a 1 element string (0x17), which would then be followed by the single bytes for the bases of 0x43 and 0x41:

0x17 0x41	REF A
0x17 0x43	ALT C

Just for discussion, suppose instead that ALT was ALT=C,T. The only thing that could change is that there would be another typed string following immediately after C encoding 0x17 (1 element string) with the value of 0x54.

6.4.5 Encoding FILTER

"PASS" is implicitly encoded as the first entry in the header dictionary (see dictionary of strings). Here we encode the PASS FILTER field as a vector of size 1 of type 8-bit, which has type byte is 0x11. The value is the offset 0:

0x11 0x00	FILTER field PASS

6.4.6 Encoding the INFO fields

HM3;AC=3;AN=6;AA=C Let's assume that the header dictionary elements for HM3, AC, AN, and AA are at 80, 81, 82, and 83 respectively. All of these can be encoded by 1-element INT8 values (0x11), with associated hex values of 0x50, 0x51, 0x52, and 0x53 respectively.

First is HM3. The entry begins with the key: 0x11 0x50. The HM3 field is a flag with no associated value and therefore the rest of the entry is the special case of a missing value without an explicit type provided. Altogether we have:

0x11 0x50 0x00	HM3 flag is present

Now let's encode the two atomic 8-bit integer fields AC and AN:

0x11 0x51	AC key
0x11 0x03	with value of 3
0x11 0x52	AN key
0x11 0x06	with value of 6

The ancestral allele (AA) tell us that among other primates the original allele is C, a Character here. Because we represent Characters as single element strings in BCF2 (0x17) with value 0x43 (C). So the entire key/value pair is:

0x11 0x53	AA key
0x17 0x43	with value of C

6.4.7 Encoding Genotypes

Continuing with our example:

```
FORMAT          NA00001              NA00002                 NA00003
GT:GQ:DP:AD:PL  0/0:10:32:32,0:0,10,100  0/1:10:48:32,16:10,0,100  1/1:10:64:0,64:100,10,0
```

Here we have the specially encoded GT field. We have two integer fields GQ and DP. We have the AD field, which is a vector of 2 values per sample. And finally we have the PL field which is 3 values per sample. Let's say that the FORMAT keys for GT, GQ, DP, AD, and PL are at offsets 1, 2, 3, and 4, 5, respectively. Now let's encode each of the genotype fields in order of the VCF record (GT, GQ, DP, AD, and then PL):

GT triplet begins with the key: 0x1101. Next is the type of the field, which will be a 2-element (diploid) INT8 type: 0x21. This is followed by 3 2-byte arrays of values 0x0202 0x0204 0x0404 (see genotype encoding example for details). The final encoding is 0x1101 0x21 0x020202040404

GQ triplet begins with the key 0x1102. Because these values are small, we encode them as 8 bit atomic integers with type code 0x11. As each value is the same (10 = 0x0A) the GQ field is encoded as 0x1102 0x11 0x0A0A0A

DP almost identical to GQ. First is the 0x1103 key, followed by 3 8-bit atomic integers encoded as 0x11 (the type) 0x20 (DP=32), 0x30 (DP=48) and 0x40 (DP=64). So we have: 0x1103 0x11203040

AD is more complex. The key is simple, just like the others, with 0x1104. Because the AD field is a vector of 2 values for each genotype, the value of key/value pair a vector type. Because the integer values in each AD field of each sample are small they are encoded by 8 bit values. So the value type is = 0x21. For sample one there are two values: 32,0 which are 0x30 and 0x00. Samples two and three are 0x30 0x20 and 0x00 0x40 respectively. So ultimately this field is encoded as 0x1104 0x21 0x300030200040

PL is just like AD but with three values per sample. The key is 0x1105. Because the PL field is a vector of 3 values for each genotype, the value of key/value pair a vector type, and because the size is 3 it's encoded in the size field of the type. Again, because the integer values in each PL field of each sample are small they are encoded by 8 bit values. So the value type 0x31. For sample one there are three values: 0, 10, and 100 which are 0x00, 0x0A, and 0x64. Samples two and three have the same values but in a slightly different order. So ultimately the PL field is encoded as 0x1105 0x31 0x000A64 0x0A0064 0x640A00

So the genotype block contains:

0x1101 0x21 0x020202040404	GT
0x1102 0x11 0x0A0A0A	GQ
0x1103 0x11 0x203040	DP
0x1104 0x21 0x300030200040	AD
0x1105 0x31 0x000A640A0064640A00	PL

Putting it all together
We need to determine a few values before writing out the final block:
l_shared = 51 (Data length from CHROM to the end of INFO)
l_indiv = 42 (Data length of FORMAT and individual genotype fields)
n_info = 4 (Number of INFO fields)
n_allele = 2 (Number of alleles, including REF)
n_sample = 3 (Number of samples)
n_fmt = 5 (Number of FORMAT keys)

0x33000000	l_shared as 32-bit little endian hex
0x2A000000	l_indiv as 32-bit little endian hex
0x01000000	CHROM offset is at 1 in 32 bit little endian
0x64000000	POS in 0-based 32-bit little endian
0x01000000	rlen = 1 (it's just a SNP)
0x41 0xF0 0xCC 0xCD	QUAL = 30.1 as 32-bit float
0x0400	n_info as 16-bit little-endian
0x0200	n_allele as 16-bit little-endian
0x030000	n_sample as 24-bit little-endian
0x05	n_fmt
0x57 0x72 0x73 0x31 0x32 0x33	ID = rs123
0x17 0x41	REF A
0x17 0x43	ALT C
0x11 0x00	FILTER field PASS
0x11 0x50 0x00	HM3 flag is present
0x11 0x51	AC key
0x11 0x03	with value of 3
0x11 0x52	AN key
0x11 0x06	with value of 6
0x11 0x53	AA key
0x17 0x43	with value of C
0x1101 0x21 0x020202040404	GT
0x1102 0x11 0x0A0A0A	GQ
0x1103 0x11 0x203040	DP
0x1104 0x21 0x300030200040	AD
0x1105 0x31 0x000A640A0064640A00	PL

That's quite a lot of information encoded in only 96 bytes!

6.5 BCF2 block gzip and indexing

These raw binary records may be subsequently encoded into BGZF blocks following the BGZF compression format, section 3 of the SAM format specification. BCF2 records can be raw, though, in cases where the decoding/encoding costs of bgzipping the data make it reasonable to process the data uncompressed, such as streaming BCF2s through pipes with samtools and bcftools. Here the files should be still compressed with BGZF but with compression 0. Implementations should perform BGZF encoding and must support the reading of both raw and BGZF encoded BCF2 files.

BCF2 files are expected to be indexed through the same index scheme, section 4 as BAM files and other block-compressed files with BGZF.

7 List of changes

7.1 VCFv4.4 Errata

- Type=P added to FORMAT field type list (used by PSL/PSO fields)

7.2 Changes between VCFv4.4 and VCFv4.3

- Added tandem repeat support (<CNV:TR>, RN, RUS, RUL, RB, CIRB, RUC, CIRUC, RUB)

- Redefined INFO CN as allele-specific copy number and FORMAT CN as total copy number.

- Redefined INFO and FORMAT CN to support non-integer copy numbers.

- Added support for phasing and derivative chromosome reconstruction in the presence of SVs (PSL, PSO, PSQ)

- Added SVCLAIM to disambiguate copy number based <DEL> and <DUP> variants from breakpoint based ones.

- Conceptually separated variant detection and interpretation.

- Added EVENTTYPE/EVENT to enable the multiple records encoding complex genomic rearrangements to be grouped together.

- Added polyploid partial phasing support (e.g. GT |0|0/1/2). GT now defined as a prefix notation with the first phasing indicator optional.

- Redefined $Number =$ for SVLEN, CIPOS, CIEND, HOMLEN, HOMSEQ, BKPTID, MEINFO, METRANS, DGVID, DBVARID, DBRIPID, MATEID, PARID, EVENT, CN, CICN to support multiple symbolic alleles.

- Redefined END as the end position of the longest ALT allele. Note that END remains $Number = 1$.

- Redefined SVLEN to always be positive and be meaningful for INV variants.

- Redefined SVLEN, END and BCF rlen to support both $< * >$ and symbolic structural variant alleles in the same record.

- Number, Type and Description required only for INFO meta-information lines

- Clarified CIPOS should be used to define micro-homology bounds

- Clarified INFO lines field ordering

- Clarified that symbolic and breakpoint notation ALT alleles are case sensitive.

- Deprecated SVTYPE

- Removed DPADJ, CNADJ, CICNADJ

- Disallowed BND ALT records (these should use breakpoint notation anyway)

- Removed definition of DP from Section 3 as it was already defined in Section 1.6.1.8.

- Deprecated bundles in favour of PSL-based phasing.

- Updated SV examples.

- BCF parsers should perform BGZF encoding and must support both raw and BGZF encoded files.

7.3   Changes to VCFv4.3

- More strict language: "should" replaced with "must" where appropriate

- Tables with Type and Number definitions for INFO and FORMAT reserved keys

- The set of characters allowed in VCF contig names is now the same as that allowed in SAM reference sequence names, which was restricted in January 2019. The characters '\ , " ' ' () {}' are now invalid in VCF contig names, while '*' is now valid when not the first character. (The characters '[] <>' and initial '*'/'=' were already invalid and remain so.)

  The VCF specification previously disallowed colons (':') in contig names to avoid confusion when parsing breakends, but this was unnecessary. Even with contig names containing colons, the breakend mate position notation can be unambiguously parsed because the ":pos" part is **always** present.

- Added PP tag which is the phred-scaled analogue to GP

7.4 Changes between VCFv4.2 and VCFv4.3

- VCF compliant implementations must support both LF and CR+LF newline conventions

- INFO and FORMAT tag names must match the regular expression `^[A-Za-z_][0-9A-Za-z_.]*$`

- Spaces are allowed in INFO field values

- Characters with special meaning (such as ';' in INFO, ':' in FORMAT, and '%' in both) can be encoded using percent encoding (see Section 1.2)

- The character encoding of VCF files is UTF-8.

- The SAMPLE field can contain optional DOI URL for the source data file

- Introduced ##META header lines for defining phenotype metadata

- New reserved tag "CNP" analogous to "GP" was added. Both CNP and GP use 0 to 1 encoding, which is a change from previous phred-scaled GP.

- In order for VCF and BCF to have the same expressive power, we state explicitly that Integers and Floats are 32-bit numbers. Integers are signed.

- We state explicitly that zero length strings are not allowed, this includes the CHROM and ID column, INFO IDs, FILTER IDs and FORMAT IDs. Meta-information lines can be in any order, with the exception of ##fileformat which must come first.

- All header lines of the form ##key=<ID=xxx,...> must have an ID value that is unique for a given value of "key". All header lines whose value starts with "<" must have an ID field. Therefore, also ##PEDIGREE newly requires a unique ID.

- We state explicitly that duplicate IDs, FILTER, INFO or FORMAT keys are not valid.

- A section about gVCF was added, introduced the <*> symbolic allele.

- A section about tag naming conventions was added.

- New reserved AD, ADF, and ADR INFO and FORMAT fields added.

- Removed unused and ill-defined GLE FORMAT tag.

- Chromosome names cannot use reserved symbolic alleles and contain characters used by breakpoints (Section 1.4.7).

- IUPAC ambiguity codes should be converted to a concrete base.

- Symbolic ALTs for IUPAC codes.

7.5 Changes between BCFv2.1 and BCFv2.2

- BCF header lines can include optional IDX field

- We introduce END_OF_VECTOR byte and reserve 8 values for future use

- Clarified that except the END_OF_VECTOR byte, no other negative values are allowed in the GT array

- String vectors in BCF do not need to start with comma, as the number of values is indicated already in the definition of the tag in the header.

- The implicit filter PASS was described inconsistently throughout BCFv2.1: It is encoded as the first entry in the dictionary, not the last.

7.6 Changes between VCFv4.1 and VCFv4.2

- In the HEADER the INFO definition allow additional field after the 4 first mandatory fields

- INFO fields definition can use Number=R to specify one per allele including the reference.

- Allow star allele in the ALT for describing upstream deletion