Introduction to *VariantAnnotation*

Valerie Obenchain

October 1, 2013

Contents

1 Introduction **1**

2 Variant Call Format (VCF) files **2**
2.1 Data import and exploration 2
2.1.1 Header information 2
2.1.2 Genomic positions 3
2.1.3 Genotype data 4
2.1.4 Info data 6
2.2 Import data subsets 8
2.2.1 Select genomic coordinates 8
2.2.2 Select VCF fields 9

3 Locating variants in and around genes **10**

4 Amino acid coding changes **11**

5 SIFT and PolyPhen Databases **13**

6 Other operations **14**
6.1 Create a SnpMatrix 14
6.2 Expand a VCF 16
6.3 Write out VCF files 16

7 References **16**

8 Session Information **16**

1 Introduction

This vignette outlines a work flow for annotating and filtering genetic variants using the *VariantAnnotation*package. Sample data are in VariantCall Format (VCF) and are a subset of chromosome 22 from 1000 Genomes. VCF text files contain meta-information lines, a header line with column names, data lines with information about a position in the genome, and optional genotype information on samples for each position. The 1000 Genomes page describes the VCF format in detail.

Data are read in from a VCF file and variants identified according to region such as `coding`, `intron`, `intergenic`, `spliceSite` etc. Amino acid coding changes are computed for the non-synonymous variants and SIFT and PolyPhen databases provide predictions of how severly the coding changes affect protein function.

2 Variant Call Format (VCF) files

2.1 Data import and exploration

Data are parsed into a `VCF` object with `readVcf`.

```
> library(VariantAnnotation)
> fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
> vcf <- readVcf(fl, "hg19")
> vcf
```

```
class: CollapsedVCF
dim: 10376 5
rowData(vcf):
  GRanges with 5 metadata columns: paramRangeID, REF, ALT, QUAL, FILTER
info(vcf):
  DataFrame with 22 columns: LDAF, AVGPOST, RSQ, ERATE, THETA, CIEND...
info(header(vcf)):
             Number Type    Description
   LDAF      1      Float   MLE Allele Frequency Accounting for LD
   AVGPOST   1      Float   Average posterior probability from MaCH...
   RSQ       1      Float   Genotype imputation quality from MaCH/T...
   ERATE     1      Float   Per-marker Mutation rate from MaCH/Thunder
   THETA     1      Float   Per-marker Transition rate from MaCH/Th...
   CIEND     2      Integer Confidence interval around END for impr...
   CIPOS     2      Integer Confidence interval around POS for impr...
   END       1      Integer End position of the variant described i...
   HOMLEN    .      Integer Length of base pair identical micro-hom...
   HOMSEQ    .      String  Sequence of base pair identical micro-h...
   SVLEN     1      Integer Difference in length between REF and AL...
   SVTYPE    1      String  Type of structural variant
   AC        .      Integer Alternate Allele Count
   AN        1      Integer Total Allele Count
   AA        1      String  Ancestral Allele, ftp://ftp.1000genomes...
   AF        1      Float   Global Allele Frequency based on AC/AN
   AMR_AF    1      Float   Allele Frequency for samples from AMR b...
   ASN_AF    1      Float   Allele Frequency for samples from ASN b...
   AFR_AF    1      Float   Allele Frequency for samples from AFR b...
   EUR_AF    1      Float   Allele Frequency for samples from EUR b...
   VT        1      String  indicates what type of variant the line...
   SNPSOURCE .      String  indicates if a snp was called when anal...
geno(vcf):
  SimpleList of length 3: GT, DS, GL
geno(header(vcf)):
      Number Type   Description
   GT 1      String Genotype
   DS 1      Float  Genotype dosage from MaCH/Thunder
   GL .      Float  Genotype Likelihoods
```

2.1.1 Header information

Header information can be extracted from the VCF with `header()`. We see there are 5 samples, 1 piece of meta information, 22 info fields and 3 geno fields.

```
> header(vcf)

class: VCFHeader
samples(5): HG00096 HG00097 HG00099 HG00100 HG00101
meta(1): fileformat
fixed(1): ALT
info(22): LDAF AVGPOST ... VT SNPSOURCE
geno(3): GT DS GL
```

Data can be further extracted using the named accessors.

```
> samples(header(vcf))

[1] "HG00096" "HG00097" "HG00099" "HG00100" "HG00101"

> geno(header(vcf))

DataFrame with 3 rows and 3 columns
        Number        Type                        Description
   <character> <character>                        <character>
GT           1      String                           Genotype
DS           1       Float Genotype dosage from MaCH/Thunder
GL           .       Float              Genotype Likelihoods
```

2.1.2 Genomic positions

`rowData` contains information from the CHROM, POS, and ID fields of the VCF file, represented as a `GRanges`. The `paramRangeID` column is meaningful when reading subsets of data and is discussed further below.

```
> head(rowData(vcf), 3)

GRanges with 3 ranges and 5 metadata columns:
              seqnames               ranges strand | paramRangeID
                 <Rle>            <IRanges>  <Rle> |     <factor>
    rs7410291       22 [50300078, 50300078]      * |         <NA>
  rs147922003       22 [50300086, 50300086]      * |         <NA>
  rs114143073       22 [50300101, 50300101]      * |         <NA>
                        REF                ALT      QUAL      FILTER
              <DNAStringSet> <DNAStringSetList> <numeric> <character>
    rs7410291              A                  G       100        PASS
  rs147922003              C                  T       100        PASS
  rs114143073              G                  A       100        PASS
  ---
  seqlengths:
   22
   NA
```

Individual fields can be pulled out with named accessors. Here we see `REF` is stored as a `DNAStringSet` and `qual` is a numeric vector.

```
> ref(vcf)[1:5]
```

```
  A DNAStringSet instance of length 5
    width seq
[1]     1 A
[2]     1 C
[3]     1 G
[4]     1 C
[5]     1 C
```

```
> qual(vcf)[1:5]
```

```
[1] 100 100 100 100 100
```

ALT is a DNAStringSetList (allows for multiple alternate alleles per variant) or a DNAStringSet. When structural variants are present it will be a CharacterList.

```
> alt(vcf)[1:5]
```

```
DNAStringSetList of length 5
[[1]] G
[[2]] T
[[3]] A
[[4]] T
[[5]] T
```

2.1.3 Genotype data

Genotype data described in the FORMAT fields are parsed into the geno slot. The data are unique to each sample and each sample may have multiple values variable. Because of this, the data are parsed into matrices or arrays where the rows represent the variants and the columns the samples. Multidimentional arrays indicate multiple values per sample. In this file all variables are matrices.

```
> geno(vcf)
```

```
List of length 3
names(3): GT DS GL
```

```
> sapply(geno(vcf), class)
```

```
      GT       DS       GL
"matrix" "matrix" "matrix"
```

Let's take a closer look at the genotype dosage (DS) variable. The header provides the variable definition and type.

```
> geno(header(vcf))["DS",]
```

```
DataFrame with 1 row and 3 columns
        Number        Type                        Description
   <character> <character>                        <character>
DS           1       Float Genotype dosage from MaCH/Thunder
```

These data are stored as a 10376 x 5 matrix. Each of the five samples (columns) has a single value per variant location (row).

```
> DS <-geno(vcf)$DS
> dim(DS)
```

```
[1] 10376     5
```

```
> DS[1:3,]
```

```
            HG00096 HG00097 HG00099 HG00100 HG00101
rs7410291         0       0       1       0       0
rs147922003       0       0       0       0       0
rs114143073       0       0       0       0       0
```

DS is also known as 'posterior mean genotypes' and range in value from [0, 2]. To get a sense of variable distribution, we compute a five number summary of the minimum, lower-hinge (first quartile), median, upper-hinge (third quartile) and maximum.

```
> fivenum(DS)
```

```
[1] 0 0 0 0 2
```

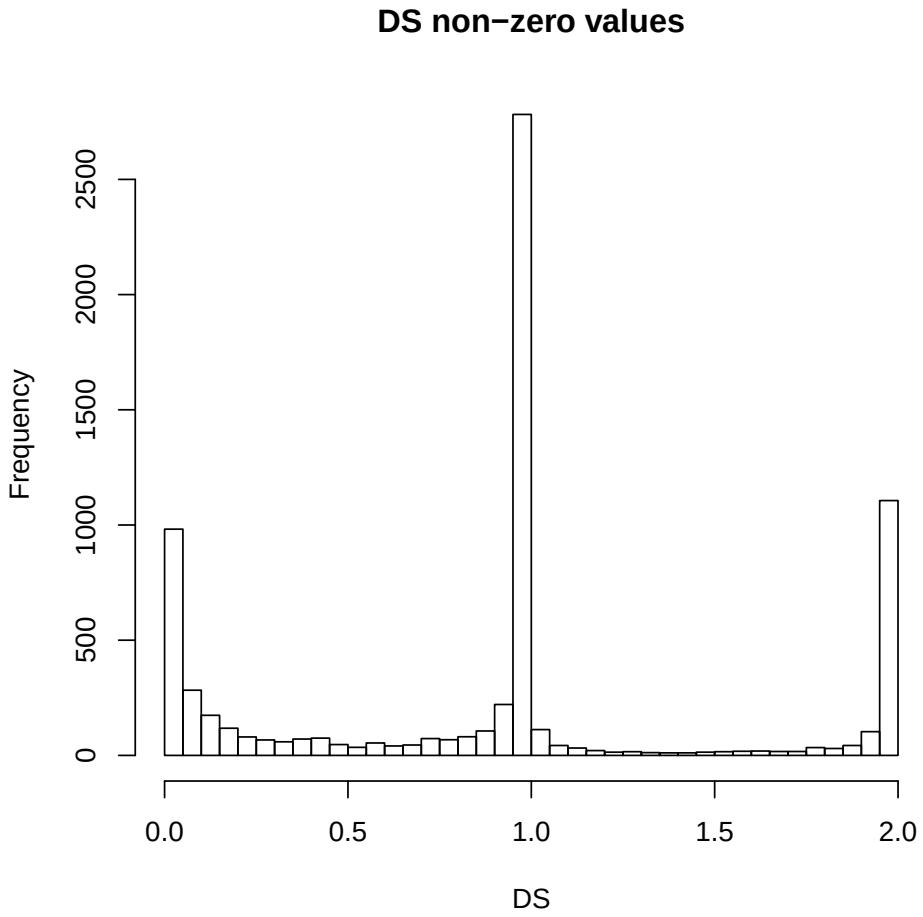
The majority of these values (86%) are zero.

```
> length(which(DS==0))/length(DS)
```

```
[1] 0.8621627
```

View the distribution of the non-zero values.

```
> hist(DS[DS != 0], breaks=seq(0, 2, by=0.05),
+     main="DS non-zero values", xlab="DS")
```



2.1.4 Info data

In contrast to the genotype data, the info data are unique to the variant and the same across samples. All info variables are represented in a single `DataFrame`.

```
> info(vcf)[1:4, 1:5]
```

```
DataFrame with 4 rows and 5 columns
                LDAF   AVGPOST       RSQ     ERATE     THETA
           <numeric> <numeric> <numeric> <numeric> <numeric>
rs7410291      0.3431    0.9890    0.9856     2e-03    0.0005
rs147922003    0.0091    0.9963    0.8398     5e-04    0.0011
rs114143073    0.0098    0.9891    0.5919     7e-04    0.0008
rs141778433    0.0062    0.9950    0.6756     9e-04    0.0003
```

We will use the info data to compare quality measures between novel (i.e., not in dbSNP) and known (i.e., in dbSNP) variants and the variant type present in the file. Variants with membership in dbSNP can be identified by using the appropriate SNPlocs package for hg19.

```
> library(SNPlocs.Hsapiens.dbSNP.20101109)
> dbsnprd <- renameSeqlevels(rowData(vcf), c("22"="ch22"))
```

```
> ch22snps <- getSNPlocs("ch22")
> dbsnpchr22 <- sub("rs", "", names(dbsnprd)) %in% ch22snps$RefSNP_id
> table(dbsnpchr22)

dbsnpchr22
FALSE  TRUE
 6259  4117
```

Info variables of interest are 'VT', 'LDAF' and 'RSQ'. The header offers more details on these variables.

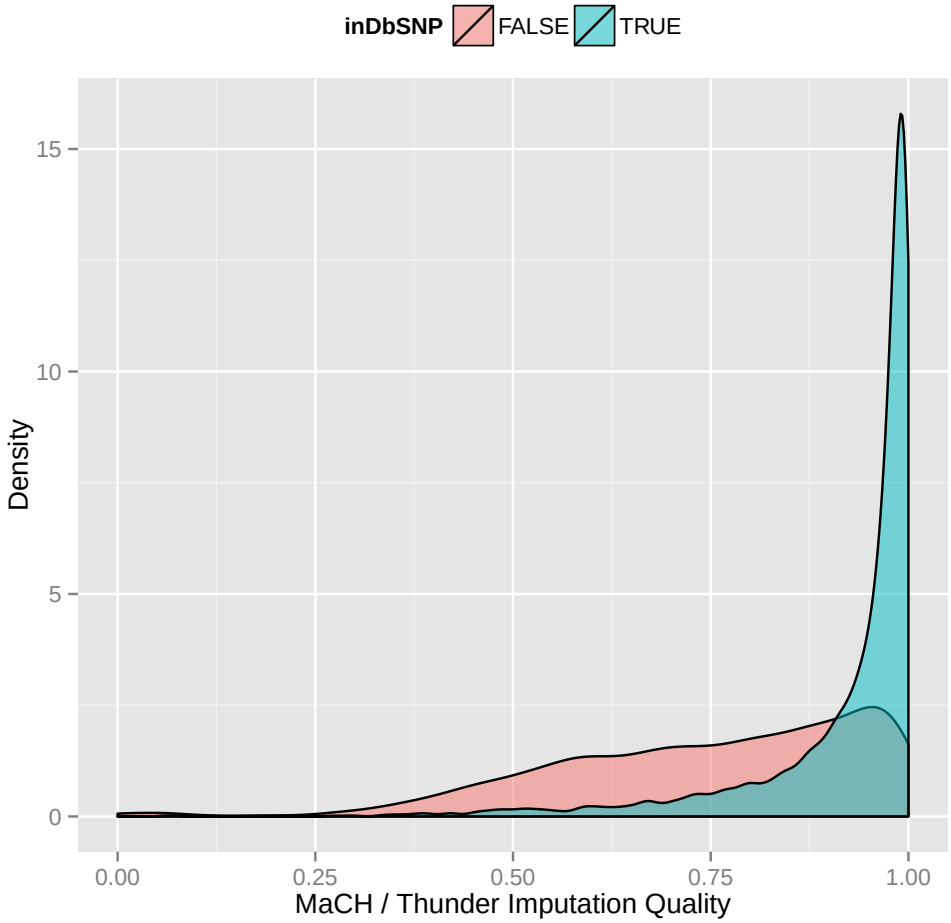
```
> info(header(vcf))[c("VT", "LDAF", "RSQ"),]

DataFrame with 3 rows and 3 columns
          Number        Type
     <character> <character>
VT             1      String
LDAF           1       Float
RSQ            1       Float
                                          Description
                                          <character>
VT   indicates what type of variant the line represents
LDAF             MLE Allele Frequency Accounting for LD
RSQ       Genotype imputation quality from MaCH/Thunder
```

Create a data frame of quality measures of interest ...

```
> metrics <- data.frame(QUAL=qual(vcf), inDbSNP=dbsnpchr22,
+     VT=info(vcf)$VT, LDAF=info(vcf)$LDAF, RSQ=info(vcf)$RSQ)
```

and visualize the distribution of qualities using `ggplot2`. For instance, genotype imputation quality is higher for the known variants in dbSNP.

```
> library(ggplot2)
> ggplot(metrics, aes(x=RSQ, fill=inDbSNP)) +
+     geom_density(alpha=0.5) +
+     scale_x_continuous(name="MaCH / Thunder Imputation Quality") +
+     scale_y_continuous(name="Density") +
+     theme(legend.position="top")
```



2.2 Import data subsets

When working with large VCF files it may be more efficient to read in subsets of the data. This can be accomplished by selecting genomic coordinates (ranges) or by specific fields from the VCF file.

2.2.1 Select genomic coordinates

To read in a portion of chromosome 22, create a `GRanges` with the regions of interest.

```
> rng <- GRanges(seqnames="22", ranges=IRanges(
+            start=c(50301422, 50989541),
+            end=c(50312106, 51001328),
+            names=c("gene_79087", "gene_644186")))
```

When ranges are specified, the VCF file must have an accompanying Tabix index file. See ?`indexTabix` for help creating an index.

```
> tab <- TabixFile(fl)
> vcf_rng <- readVcf(tab, "hg19", param=rng)
```

The `paramRangesID` column distinguishes which records came from which param range.

```
> head(rowData(vcf_rng), 3)
```

```
GRanges with 3 ranges and 5 metadata columns:
              seqnames              ranges strand | paramRangeID
                 <Rle>           <IRanges>  <Rle> |     <factor>
  rs114335781       22 [50301422, 50301422]      * |   gene_79087
    rs8135963       22 [50301476, 50301476]      * |   gene_79087
  22:50301488       22 [50301488, 50301488]      * |   gene_79087
                       REF               ALT      QUAL      FILTER
              <DNAStringSet> <DNAStringSetList> <numeric> <character>
  rs114335781              G                 A       100        PASS
    rs8135963              T                 C       100        PASS
  22:50301488              C                 T       100        PASS
  ---
  seqlengths:
   22
   NA
```

2.2.2 Select VCF fields

Data import can also be defined by the `fixed`, `info` and `geno` fields. Fields available for import are described in the header information. To view the header before reading in the data, use `ScanVcfHeader`.

```
> hdr <- scanVcfHeader(fl)
> ## e.g., INFO and GENO fields
> head(info(hdr), 3)
```

```
DataFrame with 3 rows and 3 columns
            Number        Type
        <character> <character>
LDAF              1       Float
AVGPOST           1       Float
RSQ               1       Float
                                          Description
                                          <character>
LDAF             MLE Allele Frequency Accounting for LD
AVGPOST Average posterior probability from MaCH/Thunder
RSQ       Genotype imputation quality from MaCH/Thunder
```

```
> head(geno(hdr), 3)
```

```
DataFrame with 3 rows and 3 columns
        Number        Type                        Description
   <character> <character>                        <character>
GT           1      String                           Genotype
DS           1       Float Genotype dosage from MaCH/Thunder
GL           .       Float              Genotype Likelihoods
```

To subset on "LDAF" and "GT" we specify them as `character` vectors in the `info` and `geno` arguments to `ScanVcfParam`. This creates a `ScanVcfParam` object which is used as the `param` argument to `readVcf`.

```
> ## Return all 'fixed' fields, "LAF" from 'info' and "GT" from 'geno'
> svp <- ScanVcfParam(info="LDAF", geno="GT")
> vcf1 <- readVcf(fl, "hg19", svp)
> names(geno(vcf1))
```

```
[1] "GT"
```

To subset on both genomic coordinates and fields the `ScanVcfParam` object must contain both.

```
> svp_all <- ScanVcfParam(info="LDAF", geno="GT", which=rng)
> svp_all

class: ScanVcfParam
vcfWhich: 1 elements
vcfFixed: character() [All]
vcfInfo: LDAF
vcfGeno: GT
```

3 Locating variants in and around genes

Variant location with respect to genes can be identified with the `locateVariants` function. Regions are specified in the `region` argument and can be one of the following constructors: CodingVariants, IntronVariants, FiveUTRVariants, ThreeUTRVariants, IntergenicVariants, SpliceSiteVariants or PromoterVariants. Location definitions are shown in Table 1.

Location	Details
coding	falls *within* a coding region
fiveUTR	falls *within* a 5' untranslated region
threeUTR	falls *within* a 3' untranslated region
intron	falls *within* an intron region
intergenic	does not fall *within* a transcript associated with a gene
spliceSite	overlaps any portion of the first 2 or last 2 nucleotides of an intron
promoter	falls *within* a promoter region of a transcript

Table 1: Variant locations

For overlap methods to work properly the chromosome names (seqlevels) must be compatible in the objects being compared. The VCF data chromosome names are represented by number, i.e., '22', but the TxDb chromosome names are preceded with 'chr'. Modify the seqlevels in the VCF object with `renameSeqlevels`.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> vcf <- renameSeqlevels(vcf, c("22"="chr22"))
> rd <- rowData(vcf)
> loc <- locateVariants(rd, txdb, CodingVariants())
> head(loc, 3)

GRanges with 3 ranges and 7 metadata columns:
      seqnames               ranges strand | LOCATION   QUERYID
         <Rle>            <IRanges>  <Rle> | <factor> <integer>
  [1]    chr22 [50301422, 50301422]      * |   coding        24
  [2]    chr22 [50301476, 50301476]      * |   coding        25
  [3]    chr22 [50301488, 50301488]      * |   coding        26
           TXID     CDSID      GENEID   PRECEDEID    FOLLOWID
      <integer> <integer> <character> <character> <character>
  [1]     73482    217009       79087        <NA>        <NA>
  [2]     73482    217009       79087        <NA>        <NA>
```

```
  [3]     73482    217009       79087        <NA>        <NA>
  ---
  seqlengths:
   chr22
      NA
```

Locate variants in all regions with the `AllVariants()` constructor,

```
> allvar <- locateVariants(rd, txdb, AllVariants())
```

To answer gene-centric questions data can be summarized by gene reguardless of transcript.

```
> ## Did any coding variants match more than one gene?
> splt <- split(mcols(loc)$GENEID, mcols(loc)$QUERYID)
> table(sapply(splt, function(x) length(unique(x)) > 1))

FALSE  TRUE
  956    15

> ## Summarize the number of coding variants by gene ID.
> splt <- split(mcols(loc)$QUERYID, mcols(loc)$GENEID)
> head(sapply(splt, function(x) length(unique(x))), 3)

113730   1890  23209
    22     15     30
```

4 Amino acid coding changes

`predictCoding` computes amino acid coding changes for non-synonymous variants. Only ranges in `query` that overlap with a coding region in the `subject` are considered. Reference sequences are retrieved from either a `BSgenome` or fasta file specified in `seqSource`. Variant sequences are constructed by substituting, inserting or deleting values in the `varAllele` column into the reference sequence. Amino acid codes are computed for the variant codon sequence when the length is a multiple of 3.

The `query` argument to `predictCoding` can be a `GRanges` or `VCF`. When a `GRanges` is supplied the `varAllele` argument must be specified. In the case of a `VCF`, the alternate alleles are taken from `alt(<VCF>)` and the `varAllele` argument is not specified.

The result is a modified `query` containing only variants that fall within coding regions. Each row represents a variant-transcript match so more than one row per original variant is possible.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> coding <- predictCoding(vcf, txdb, seqSource=Hsapiens)
> coding[5:7]

GRanges with 3 ranges and 17 metadata columns:
              seqnames               ranges strand | paramRangeID
                 <Rle>            <IRanges>  <Rle> |     <factor>
  22:50301584    chr22 [50301584, 50301584]      - |         <NA>
  rs114264124    chr22 [50302962, 50302962]      - |         <NA>
  rs149209714    chr22 [50302995, 50302995]      - |         <NA>
                         REF                ALT      QUAL      FILTER
              <DNAStringSet> <DNAStringSetList> <numeric> <character>
  22:50301584              C                  T       100        PASS
  rs114264124              C                  T       100        PASS
```

```
  rs149209714              C                   G       100        PASS
                  varAllele     CDSLOC    PROTEINLOC   QUERYID
              <DNAStringSet>  <IRanges> <IntegerList> <integer>
  22:50301584             A [777, 777]           259        28
  rs114264124             A [698, 698]           233        57
  rs149209714             C [665, 665]           222        58
                    TXID     CDSID      GENEID   CONSEQUENCE
              <character> <integer> <character>      <factor>
  22:50301584       73482    217009       79087    synonymous
  rs114264124       73482    217010       79087 nonsynonymous
  rs149209714       73482    217010       79087 nonsynonymous
                  REFCODON       VARCODON         REFAA         VARAA
              <DNAStringSet> <DNAStringSet> <AAStringSet> <AAStringSet>
  22:50301584           CCG            CCA             P             P
  rs114264124           CGG            CAG             R             Q
  rs149209714           GGA            GCA             G             A
  ---
  seqlengths:
   chr22
      NA
```

Using variant rs114264124 as an example, we see varAllele `A` has been substituted into the `refCodon` `CGG` to produce `varCodon` `CAG`. The `refCodon` is the sequence of codons necessary to make the variant allele substitution and therefore often includes more nucleotides than indicated in the range (i.e. the range is 50302962, 50302962, width of 1). Notice it is the second position in the `refCodon` that has been substituted. This position in the codon, the position of substitution, corresponds to genomic position 50302962. This genomic position maps to position 698 in coding region-based coordinates and to triplet 233 in the protein. This is a non-synonymous coding variant where the amino acid has changed from `R` (Arg) to `Q` (Gln).

When the resulting `varCodon` is not a multiple of 3 it cannot be translated. The consequence is considered a `frameshift` and `varAA` will be missing.

```
> ## CONSEQUENCE is 'frameshift' where translation is not possible
> coding[mcols(coding)$CONSEQUENCE == "frameshift"]

GRanges with 1 range and 17 metadata columns:
              seqnames               ranges strand | paramRangeID
                 <Rle>            <IRanges>  <Rle> |     <factor>
  22:50317001    chr22 [50317001, 50317001]      + |         <NA>
                        REF               ALT      QUAL      FILTER
              <DNAStringSet> <DNAStringSetList> <numeric> <character>
  22:50317001              G              GCACT       233        PASS
                  varAllele     CDSLOC    PROTEINLOC   QUERYID
              <DNAStringSet>  <IRanges> <IntegerList> <integer>
  22:50317001          GCACT [808, 808]           270       359
                    TXID     CDSID      GENEID CONSEQUENCE
              <character> <integer> <character>    <factor>
  22:50317001       72592    214765       79174  frameshift
                  REFCODON       VARCODON         REFAA         VARAA
              <DNAStringSet> <DNAStringSet> <AAStringSet> <AAStringSet>
  22:50317001           GCC            GCC             A
  ---
  seqlengths:
```

```
chr22
   NA
```

5 SIFT and PolyPhen Databases

From `predictCoding` we identified the amino acid coding changes for the non-synonymous variants. For this subset we can retrieve predictions of how damaging these coding changes may be. SIFT (Sorting Intolerant From Tolerant) and PolyPhen (Polymorphism Phenotyping) are methods that predict the impact of amino acid substitution on a human protein. The SIFT method uses sequence homology and the physical properties of amino acids to make predictions about protein function. PolyPhen uses sequence-based features and structural information characterizing the substitution to make predictions about the structure and function of the protein.

Collated predictions for specific dbSNP builds are available as downloads from the SIFT and PolyPhen web sites. These results have been packaged into *SIFT.Hsapiens.dbSNP132.db* and *PolyPhen.Hapiens.dbSNP131.db* and are designed to be searched by rsid. Variants that are in dbSNP can be searched with these database packages. When working with novel variants, SIFT and PolyPhen must be called directly. See references for home pages.

Identify the non-synonymous variants and obtain the rsids.

```
> nms <- names(coding)
> idx <- mcols(coding)$CONSEQUENCE == "nonsynonymous"
> nonsyn <- coding[idx]
> names(nonsyn) <- nms[idx]
> rsids <- unique(names(nonsyn)[grep("rs", names(nonsyn), fixed=TRUE)])
```

Detailed descriptions of the database columns can be found with `?SIFTDbColumns` and `?PolyPhenDbColumns`. Variants in these databases often contain more than one row per variant. The variant may have been reported by multiple sources and therefore the source will differ as well as some of the other variables.

It is important to keep in mind the pre-computed predictions in the SIFT and PolyPhen packages are based on specific gene models. SIFT is based on Ensembl and PolyPhen on UCSC Known Gene. The `TranscriptDb` we used to identify the coding snps was based on UCSC Known Gene so we will use PolyPhen for predictions. PolyPhen provides predictions using two different training datasets and has considerable information about 3D protein structure. See `?PolyPhenDbColumns` or the PolyPhen web site listed in the references for more details.

Query the PolyPhen database,

```
> library(PolyPhen.Hsapiens.dbSNP131)
> pp <- select(PolyPhen.Hsapiens.dbSNP131, keys=rsids,
+           cols=c("TRAININGSET", "PREDICTION", "PPH2PROB"))
> head(pp[!is.na(pp$PREDICTION), ])

         RSID TRAININGSET        PREDICTION PPH2PROB
11  rs8139422      humdiv possibly damaging    0.228
12  rs8139422      humvar possibly damaging    0.249
13 rs74510325      humdiv possibly damaging    0.475
14 rs74510325      humvar possibly damaging    0.335
15 rs73891177      humdiv            benign    0.001
16 rs73891177      humvar            benign    0.005
```

6 Other operations

6.1 Create a SnpMatrix

The 'GT' element in the FORMAT field of the VCF represents the genotype. These data can be converted into a SnpMatrix object which can then be used with the functions offered in *snpStats* and other packages making use of the SnpMatrix class.

The genotypeToSnpMatrix function converts the genotype calls in geno to a SnpMatrix. No dbSNP package is used in this computation. The return value is a named list where 'genotypes' is a SnpMatrix and 'map' is a DataFrame with SNP names and alleles at each loci. The ignore column in 'map' indicates which variants were set to NA (missing) because they met one or more of the following criteria,

- variants with >1 ALT allele are set to NA
- only single nucleotide variants are included; others are set to NA
- only diploid calls are included; others are set to NA

See ?genotypeToSnpMatrix for more details.

```
> res <- genotypeToSnpMatrix(vcf)
> res
```

```
$genotypes
A SnpMatrix with  5 rows and  10376 columns
Row names:  HG00096 ... HG00101
Col names:  rs7410291 ... rs114526001

$map
DataFrame with 10376 rows and 4 columns
        snp.names       allele.1           allele.2    ignore
      <character> <DNAStringSet> <DNAStringSetList> <logical>
1       rs7410291              A                  G     FALSE
2     rs147922003              C                  T     FALSE
3     rs114143073              G                  A     FALSE
4     rs141778433              C                  T     FALSE
5     rs182170314              C                  T     FALSE
...           ...            ...                ...       ...
10372 rs187302552              A                  G     FALSE
10373   rs9628178              A                  G     FALSE
10374   rs5770892              A                  G     FALSE
10375 rs144055359              G                  A     FALSE
10376 rs114526001              G                  C     FALSE
```

In the map DataFrame, allele.1 represents the reference allele and allele.2 is the alternate allele.

```
> allele2 <- res$map[["allele.2"]]
> ## number of alternate alleles per variant
> unique(elementLengths(allele2))
```

```
[1] 1
```

In addition to the called genotypes, genotype likelihoods or probabilities can also be converted to a SnpMatrix, using the *snpStats* encoding of posterior probabilities as byte values. To use the values in the 'GL' or 'GP' FORMAT field instead of the called genotypes, use the uncertain=TRUE option in genotypeToSnpMatrix.

```
> fl.gl <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
> vcf.gl <- readVcf(fl.gl, "hg19")
> geno(vcf.gl)

List of length 3
names(3): GT DS GL

> ## Convert the "GL" FORMAT field to a SnpMatrix
> res <- genotypeToSnpMatrix(vcf.gl, uncertain=TRUE)
> res

$genotypes
A SnpMatrix with  85 rows and  9 columns
Row names:  NA06984 ... NA12890
Col names:  rs58108140 ... rs200430748

$map
DataFrame with 9 rows and 4 columns
    snp.names       allele.1           allele.2    ignore
  <character> <DNAStringSet> <DNAStringSetList> <logical>
1  rs58108140              G                  A     FALSE
2 rs189107123              C                         TRUE
3 rs180734498              C                  T     FALSE
4 rs144762171              G                         TRUE
5 rs201747181             TC                         TRUE
6 rs151276478              T                         TRUE
7 rs140337953              G                  T     FALSE
8 rs199681827              C                         TRUE
9 rs200430748              G                         TRUE

> t(as(res$genotype, "character"))[c(1,3,7), 1:5]

            NA06984     NA06986     NA06989     NA06994     NA07000
rs58108140  "Uncertain" "Uncertain" "A/B"       "Uncertain" "Uncertain"
rs180734498 "Uncertain" "Uncertain" "Uncertain" "Uncertain" "Uncertain"
rs140337953 "Uncertain" "Uncertain" "Uncertain" "Uncertain" "Uncertain"

> ## Compare to a SnpMatrix created from the "GT" field
> res.gt <- genotypeToSnpMatrix(vcf.gl, uncertain=FALSE)
> t(as(res.gt$genotype, "character"))[c(1,3,7), 1:5]

            NA06984 NA06986 NA06989 NA06994 NA07000
rs58108140  "A/B"   "A/B"   "A/B"   "A/A"   "A/A"
rs180734498 "A/B"   "A/A"   "A/A"   "A/A"   "A/B"
rs140337953 "B/B"   "B/B"   "A/B"   "B/B"   "A/B"

> ## What are the original likelihoods for rs58108140?
> geno(vcf.gl)$GL["rs58108140", 1:5]

$NA06984
[1] -4.70 -0.58 -0.13

$NA06986
```

```
[1] -1.15 -0.10 -0.84

$NA06989
[1] -2.05  0.00 -3.27

$NA06994
[1] -0.48 -0.48 -0.48

$NA07000
[1] -0.28 -0.44 -0.96
```

For variant rs58108140 in sample NA06989, the "A/B" genotype is much more likely than the others, so the `SnpMatrix` object displays the called genotype.

6.2 Expand a VCF

Coming soon ... CollapsedVCF and ExpandedVCF classes and expand,CollapsedVCF-method.

6.3 Write out VCF files

A VCF file can be written out from data stored in a `VCF` class. Methods to write out from more general structures are in progress.

```
> fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
> out1.vcf <- tempfile()
> out2.vcf <- tempfile()
> in1 <- readVcf(fl, "hg19")
> writeVcf(in1, out1.vcf)
> in2 <- readVcf(out1.vcf, "hg19")
> writeVcf(in2, out2.vcf)
> in3 <- readVcf(out2.vcf, "hg19")
> identical(in2, in3)

[1] FALSE
```

7 References

Wang K, Li M, Hakonarson H, (2010), ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Research, Vol 38, No. 16, e164.

McLaren W, Pritchard B, RiosD, et. al., (2010), Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. Bioinformatics, Vol. 26, No. 16, 2069-2070.

SIFT home page : `http://sift.bii.a-star.edu.sg/`

PolyPhen home page : `http://genetics.bwh.harvard.edu/pph2/`

8 Session Information

```
R version 3.0.1 (2013-05-16)
Platform: x86_64-unknown-linux-gnu (64-bit)
```

```
locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] splines   parallel  stats     graphics  grDevices utils
[7] datasets  methods   base

other attached packages:
 [1] snpStats_1.10.0
 [2] Matrix_1.0-14
 [3] lattice_0.20-23
 [4] survival_2.37-4
 [5] PolyPhen.Hsapiens.dbSNP131_1.0.2
 [6] RSQLite_0.11.4
 [7] DBI_0.2-7
 [8] BSgenome.Hsapiens.UCSC.hg19_1.3.19
 [9] BSgenome_1.28.0
[10] TxDb.Hsapiens.UCSC.hg19.knownGene_2.9.2
[11] GenomicFeatures_1.12.4
[12] AnnotationDbi_1.22.6
[13] Biobase_2.20.1
[14] ggplot2_0.9.3.1
[15] SNPlocs.Hsapiens.dbSNP.20101109_0.99.6
[16] VariantAnnotation_1.6.8
[17] Rsamtools_1.12.4
[18] Biostrings_2.28.0
[19] GenomicRanges_1.12.5
[20] IRanges_1.18.4
[21] BiocGenerics_0.6.0

loaded via a namespace (and not attached):
 [1] MASS_7.3-29        RColorBrewer_1.0-5 RCurl_1.95-4.1
 [4] XML_3.98-1.1       biomaRt_2.16.0     bitops_1.0-6
 [7] colorspace_1.2-4   dichromat_2.0-0    digest_0.6.3
[10] grid_3.0.1         gtable_0.1.2       labeling_0.2
[13] munsell_0.4.2      plyr_1.8           proto_0.3-10
[16] reshape2_1.2.2     rtracklayer_1.20.4 scales_0.2.3
[19] stats4_3.0.1       stringr_0.6.2      tools_3.0.1
[22] zlibbioc_1.6.0
```