

Overview of `ensemblVEP`

Valerie Obenchain

Last modified: December 2012; Compiled: September 11, 2013

Contents

1	Introduction	1
2	Results as R objects	1
3	Write results to a file	4
4	Configuring runtime options	5
5	<code>sessionInfo()</code>	5

1 Introduction

Ensembl provides the facility to predict functional consequences of known and unknown variants using the Variant Effect Predictor (VEP). The `ensemblVEP` package wraps Ensembl VEP and returns the results as R objects or a file on disk. To use this package the Ensembl VEP perl script must be installed in your path. See the package README for details. Downloads:

<http://uswest.ensembl.org/info/docs/variation/vep/index.html>

Complete documentation for runtime options:

http://uswest.ensembl.org/info/docs/variation/vep/vep_script.html

To test that Ensembl VEP is properly installed, enter the name of the script from the command line:

```
variant_effect_predictor.pl
```

2 Results as R objects

```
> library(ensemblVEP)
```

The `ensemblVEP` function can return variant consequences from Ensembl VEP as R objects (`GRanges` or `VCF`) or write them to a file. The default behavior returns a `GRanges`. Runtime options are stored in a `VEPParam` object and allow a great deal of control over the content and format of the results. See the man pages for more details.

```
> ?ensemblVEP
```

```
> ?VEPParam
```

The default runtime options can be inspected by creating a `VEPParam`.

```
> param <- VEPParam()
```

```
> param
```

```
class: VEPParam
```

```
basic(0):
```

```
input(1): species
```

```
cache(3): dir, dir_cache, dir_plugins
```

```
output(1): terms
```

```
identifier(0):
```

```
colocatedVariants(0):
```

```

dataformat(0):
filterqc(0):
database(2): database, host
advanced(1): buffer_size

```

```
> basic(param)
```

```

$verbose
[1] FALSE

```

```

$quiet
[1] FALSE

```

```

$no_progress
[1] FALSE

```

```

$config
character(0)

```

```

$everything
[1] FALSE

```

```

$fork
numeric(0)

```

Using a vcf file from VariantAnnotation as input, we query Ensembl VEP with the default runtime parameters.

```

> fl <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
> gr <- ensemblVEP(fl)

```

Consequence data are parsed into the metadata columns of the GRanges. To control the type and amount of data returned see the options in output(VEPParam()).

```
> head(gr, 3)
```

GRanges with 3 ranges and 12 metadata columns:

	seqnames <Rle>	ranges <IRanges>	strand <Rle>	Allele <factor>	Gene <factor>	
rs58108140	1	[10583, 10583]	*	A	ENSG00000223972	
rs58108140	1	[10583, 10583]	*	A	ENSG00000227232	
rs58108140	1	[10583, 10583]	*	A	ENSG00000227232	
	Feature <factor>	Feature_type <factor>	Consequence <factor>	cDNA_position <factor>		
rs58108140	ENST00000456328	Transcript	upstream_gene_variant	<NA>		
rs58108140	ENST00000488147	Transcript	downstream_gene_variant	<NA>		
rs58108140	ENST00000541675	Transcript	downstream_gene_variant	<NA>		
	CDS_position <factor>	Protein_position <factor>	Amino_acids <factor>	Codons <factor>		
rs58108140	<NA>	<NA>	<NA>	<NA>		
rs58108140	<NA>	<NA>	<NA>	<NA>		
rs58108140	<NA>	<NA>	<NA>	<NA>		
	Existing_variation <factor>	DISTANCE <factor>				
rs58108140	<NA>	1286				
rs58108140	<NA>	3821				
rs58108140	<NA>	3780				

```

---
seqlengths:
 1
NA

```

Next we use a vcf of structural variants as input

```
> fl <- system.file("extdata", "structural.vcf", package="VariantAnnotation")
```

and request that a VCF object be returned by setting the *vcf* option in the *dataformat* slot to TRUE.

```
> param <- VEPParam(dataformat=c(vcf=TRUE))
```

An call to `ensemblVEP` results in an error.

```
> vcf <- ensemblVEP(fl, param)
2012-12-03 16:40:55 - Starting...
ERROR: Could not detect input file format
```

In most situations Ensembl VEP can auto-detect the input format. In this case, however, it cannot so we explicitly set the *format* option to 'vcf'.

```
> input(param)$format <- "vcf"
```

Try again.

```
> vep <- ensemblVEP(fl, param)
```

Success! When a VCF is returned, consequence data are included as an unparsed INFO column labeled *CSQ*.

```
> info(vep)$CSQ
```

```
CharacterList of length 6
[[1]] -||||intergenic_variant|||||
[[2]] deletion|ENSG00000233684|ENST00000430529|Transcript|intron_variant&nc_t...
[[3]] deletion|ENSG00000230448|ENST00000418420|Transcript|intron_variant&nc_t...
[[4]] insertion|ENSG00000134077|ENST00000515662|Transcript|coding_sequence_va...
[[5]] duplication|ENSG00000132155|ENST00000423275|Transcript|intron_variant&N...
[[6]] -||||intergenic_variant|||||
```

The `parseCSQtoGRanges` function parses these data into a `GRanges`. When the rownames of the original VCF are provided as `VCFRowID` a metadata column of the same name is included in the output.

```
> vcf <- readVcf(fl, "hg19")
> csq <- parseCSQtoGRanges(vep, VCFRowID=rownames(vcf))
> head(csq, 3)
```

GRanges with 3 ranges and 13 metadata columns:

	seqnames <Rle>	ranges <IRanges>	strand <Rle>	VCFRowID <integer>	Allele <factor>							
1:	2827693	1 [2827693, 2827762]	*		2	-						
2:	321682	2 [321682, 321682]	*		3	deletion						
2:	321682	2 [321682, 321682]	*		3	deletion						
	Gene <factor>	Feature <factor>	Feature_type <factor>									
1:	2827693	<NA>	<NA>		<NA>							
2:	321682	ENSG00000233684	ENST00000430529		Transcript							
2:	321682	ENSG00000233684	ENST00000436808		Transcript							
					Consequence <factor>							
1:	2827693				intergenic_variant							
2:	321682				intron_variant&nc_transcript_variant&feature_truncation							
2:	321682				intron_variant&nc_transcript_variant&feature_truncation							
	cDNA_position <factor>	CDS_position <factor>	Protein_position <factor>	Amino_acids <factor>	Codons <factor>							
1:	2827693	<NA>	<NA>	<NA>	<NA>	<NA>						
2:	321682	<NA>	<NA>	<NA>	<NA>	<NA>						
2:	321682	<NA>	<NA>	<NA>	<NA>	<NA>						

```

Existing_variation DISTANCE
      <factor> <factor>
1:2827693      <NA>      <NA>
2:321682      <NA>      <NA>
2:321682      <NA>      <NA>
---
seqlengths:
  1 2 3 4
NA NA NA NA

```

The `VCFRowID` columns maps the expanded `CSQ` data back to the rows in the `VCF` object. This index can be used to subset the original VCF.

```

> vcf[csq$"VCFRowID"]

class: CollapsedVCF
dim: 27 1
rowData(vcf):
  GRanges with 5 metadata columns: paramRangeID, REF, ALT, QUAL, FILTER
info(vcf):
  DataFrame with 10 columns: BKPTID, CIEND, CIPOS, END, HOMLEN, HOMSEQ, IMPR...
info(header(vcf)):
  Number Type      Description
BKPTID   .      String  ID of the assembled alternate allele in the ass...
CIEND    2      Integer Confidence interval around END for imprecise va...
CIPOS    2      Integer Confidence interval around POS for imprecise va...
END       1      Integer End position of the variant described in this r...
HOMLEN   .      Integer Length of base pair identical micro-homology at...
HOMSEQ   .      String  Sequence of base pair identical micro-homology ...
IMPRECISE 0      Flag    Imprecise structural variation
MEINFO   4      String  Mobile element info of the form NAME,START,END,...
SVLEN    .      Integer Difference in length between REF and ALT alleles
SVTYPE   1      String  Type of structural variant
geno(vcf):
  SimpleList of length 4: GT, GQ, CN, CNQ
geno(header(vcf)):
  Number Type      Description
GT  1      String  Genotype
GQ  1      Float   Genotype quality
CN  1      Integer Copy number genotype for imprecise events
CNQ 1      Float   Copy number genotype quality for imprecise events

```

3 Write results to a file

In the previous section we saw Ensembl VEP results returned as R objects in the workspace. Alternatively, these results can be written directly to a file. The flag that controls how the data are returned is the `output_file` flag in the `input` options.

When `output_file` is an empty character (default), the results are returned as either a `GRanges` or `VCF` object.

```

> input(param)$output_file

character(0)

```

To write results directly to a file, specify a file name for the `output_file` flag.

```

> input(param)$output_file <- "/mypath/myfile"

```

The file can be written as a `vcf` or `gvf` by setting the options in the `dataformat` slot to `TRUE`. If neither of `vcf` or `gvf` are `TRUE` the file is written out as tab delimited.

```

> ## Write a vcf file to myfile.vcf:
> myparam <- VEPParam(dataformat=c(vcf=TRUE),
+                       input=c(output_file="/path/myfile.vcf"))
> ## Write a gvf file to myfile.gvf:
> myparam <- VEPParam(dataformat=c(gvf=TRUE),
+                       input=c(output_file="/path/myfile.gvf"))
> ## Write a tab delimited file to myfile.txt:
> myparam <- VEPParam(input=c(output_file="/path/myfile.txt"))

```

4 Configuring runtime options

The Ensembl VEP web page has complete descriptions of all runtime options. http://www.ensembl.org/info/docs/variation/vep/vep_script.html#running Below are examples of how to configure the runtime options in the *VEP-Param* for specific situations. Investigate the differences in results using a sample file from *VariantAnnotation*.

```

> fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")

```

- Add regulatory region consequences:

```

> param <- VEPParam(output=c(regulatory=TRUE))
> gr <- ensemblVEP(fl, param)

```

- Specify input file format as VCF, add HGNC gene identifiers, output SO consequence terms:

```

> param <- VEPParam(input=c(format="vcf"),
+                   output=c(terms="so"),
+                   identifiers=c(symbol=TRUE))
> gr <- ensemblVEP(fl, param)

```

- Check for co-located variants, output only coding sequence consequences, output HGVS names:

```

> param <- VEPParam(filterqc=c(coding_only=TRUE),
+                   colocatedVariants=c(check_existing=TRUE),
+                   identifiers=c(symbol=TRUE))
> gr <- ensemblVEP(fl, param)

```

- Add SIFT score and prediction, PolyPhen prediction only, output results as VCF:

```

fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
param <- VEPParam(output=c(sift="b", polyphen="p"),
                  dataformat=c(vcf=TRUE))

```

```

vcf <- ensemblVEP(fl, param)
csq <- parseCSQToGRanges(vcf)

```

```

> head(levels(mcols(csq)$SIFT))
[1] "deleterious(0.01)" "deleterious(0.02)" "deleterious(0.03)"
[4] "deleterious(0.04)" "deleterious(0.05)" "deleterious(0)"

```

```

> levels(mcols(csq)$PolyPhen)
[1] "benign"           "possibly_damaging" "probably_damaging"
[4] "unknown"

```

5 sessionInfo()

```

> sessionInfo()

```

```

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] parallel stats graphics grDevices utils datasets methods
[8] base
```

other attached packages:

```
[1] ensemblVEP_1.0.5          VariantAnnotation_1.6.7 Rsamtools_1.12.4
[4] Biostrings_2.28.0         GenomicRanges_1.12.5   IRanges_1.18.3
[7] BiocGenerics_0.6.0
```

loaded via a namespace (and not attached):

```
[1] AnnotationDbi_1.22.6      BSgenome_1.28.0        Biobase_2.20.1
[4] DBI_0.2-7                 GenomicFeatures_1.12.3 RCurl_1.95-4.1
[7] RSQLite_0.11.4           XML_3.98-1.1           biomaRt_2.16.0
[10] bitops_1.0-6             rtracklayer_1.20.4    stats4_3.0.1
[13] tools_3.0.1              zlibbioc_1.6.0
```