

Using refGenome package

Wolfgang Kaisers, CIBs HHU Dusseldorf

March 22, 2016

1 refGenome package

The **refGenome** package provides functionality for managing of genome annotation data, especially for Ensembl and UCSC data.

2 Object types inside refGenome package

The central classes inside this package are **refGenome** derived (S4) classes. The class contains two slots: **ev** (**environment**) and **basedir** (**character**). All annotation data is kept in data.frames inside the **ev** slot. Saving and loading **refGenome** derived objects works on the complete content of the environment. This mechanism also avoids generation of copies and allows addition of new data inside of member functions. The **basedir** slot keeps a path on a hard-disc which is intended as location where data files and object versions can be kept.

The package contains three derived class lineages **refGenome**, **refExons** and **refJunctions**. For each lineage there are classes for Ensembl and UCSC defined, e.g. **ensemblGenome** and **ucscGenome**. The exon classes focus on annotated exon positions and the junction classes focus on adjacent exons.

2.1 Creation of empty refGenome objects

Empty objects of **refGenome** derived classes can be created with **ensembleGenome()** or **ucscGenome()**. After creation of an empty object the first step usually is to set the **basedir** address:

```
> library(refGenome)
> beg <- ensemblGenome()
> basedir(beg) <- system.file("extdata", package="refGenome")
```

The "basedir" folder is intended to contain all data which is associated with the current annotation set, e.g. downloaded gtf files, saved object data, saved SQLite versions of the data and potentially sequence information. In order to fill an empty object, annotation data has to be imported from external files.

2.2 Importing annotation data

The basic importing mechanism for `refGenome` objects is to import a "gtf" file. Therefore, the "gtf" files have to be downloaded. The download source and mechanism is explained for `ensemblGenome` and `ucscGenome` separately. There are specialized mechanisms in order to provide additional information either from within the gtf file (ensembl) or via other external files (ucsc).

2.3 Saving and loading data

The data content of `refGenome` objects can be saved and re-loaded in several ways. One way is the `saveGenome` method where the content is written into a compressed ".RData" file. One alternative is to write the content into a SQLite database via `writeDB`.

3 Ensembl Genomes

The `ensemblGenome` class is specialized for managing annotation data for ensemble Genomes.

3.1 Download and import data

For `ensemblGenome` objects, gtf files can be downloaded from Ensemble servers. Therefore, go to

```
http://www.ensembl.org/info/data/ftp/index.html
```

and choose a file from the "Gene sets" column. They are labeled "GTF". For example Version 62 of human genomic annotation can be downloaded from

```
ftp://ftp.ensembl.org/pub/release-80/gtf/homo_sapiens/Homo_sapiens.GRCh38.80.gtf.gz
```

A copy of the obtained file should then be placed in the the "basedir" directory. With the appropriate setting of basedir, annotation data can be imported with:

```
> ens_gtf <- "hs.ensembl.62.small.gtf"
> read.gtf(beg, ens_gtf)
```

```
[read.gtf.refGenome] Reading file 'hs.ensembl.62.small.gtf'.
```

```
[GTF]          135 lines processed.
[read.gtf.refGenome] Extracting genes table.
[read.gtf.refGenome] Finished.
```

```
> beg
```

```
Object of class 'ensemblGenome' with 135 rows and 15 columns.
```

	id	seqid	source	feature	start	end
1	1	GL000213.1	protein_coding	exon	138767	139339

```

2 2 GL000213.1 protein_coding      CDS 138767 139287
3 3 GL000213.1 protein_coding start_codon 139285 139287
4 4 GL000213.1 protein_coding      exon 134276 134390
5 5 GL000213.1 protein_coding      CDS 134276 134390
6 6 GL000213.1 protein_coding      exon 133943 134116
  score strand frame      protein_id transcript_name
1      .      -      .      <NA>    BX072566.1-201
2      .      -      0 ENSP00000329990 BX072566.1-201
3      .      -      0      <NA>    BX072566.1-201
4      .      -      .      <NA>    BX072566.1-201
5      .      -      1 ENSP00000329990 BX072566.1-201
6      .      -      .      <NA>    BX072566.1-201
      gene_id      transcript_id gene_name exon_number
1 ENSG00000237375 ENST00000327822 BX072566.1      1
2 ENSG00000237375 ENST00000327822 BX072566.1      1
3 ENSG00000237375 ENST00000327822 BX072566.1      1
4 ENSG00000237375 ENST00000327822 BX072566.1      2
5 ENSG00000237375 ENST00000327822 BX072566.1      2
6 ENSG00000237375 ENST00000327822 BX072566.1      3

```

The top lines of the contained table are shown when the object is printed.

4 UCSC Genomes

Downloading of annotation data for UCSC genomes is a bit more complicated than for Ensemble Genomes because additional data must be downloaded in separate files. The Homepage for UCSC browser can be found under:

<http://genome.ucsc.edu/>

In order to import UCSC annotation data into **refGenome** objects files containing the data have to be downloaded from the USCS Table Browser which can be found under:

<http://genome.ucsc.edu/cgi-bin/hgTables>

or by following the "Table Browser" link in the left panel on the homepage. On the Table Browser:

- Select genome, assembly and track (UCSC genes)
- Choose table (knownGene)
- Choose output format (GTF -gene transfer format for knownGene table)
- Insert a name for the output file
- Download the file (get output)

The basic table to be imported is "knownGene". The knownGene table has to be downloaded in GTF format (otherwise the read.gtf function will complain about "wrong number of columns").

In order to extend the available information additionally the tables "kgXref", "knownToEnsembl" and "knownIsoforms" can be downloaded and imported. These tables come in plain "csv" format. Select "all fields from selected table" as output format.

Do not use "add custom tracks" or modify the tables elsewhere tracks because the importing functions will check for appropriate number of columns.

After downloading, all tables should be placed into a separate folder which we from now on call "basedir". ucscGenome objects keep a **basedir** as standard location for all writing and reading procedures.

```
> uc <- ucscGenome()
> basedir(uc) <- "/my/ucsc/basedir"
> read.gtf(uc, "ucsc_knownGene.gtf")
> addXref(uc, "kgXref.csv")
> addEnsembl(uc, "knownToEnsembl.csv")
> addIsoforms(uc, "ucsc_knownisoforms.csv")
```

4.1 Load stored data

Once, annotation data is imported and stored, ucscGenome objects can be re-stored with the loadGenome function which is shown below on example data:

```
> ucfile <- system.file("extdata", "hs.ucsc.small.RData", package="refGenome")
> uc <- loadGenome(ucfile)
> ensfile <- system.file("extdata", "hs.ensembl.62.small.RData", package="refGenome")
> ens <- loadGenome(ensfile)
```

5 Extracting data subsets

There are specialized functions for extracting data for multiple purposes.

5.1 Extracting data for sets of seqid's

For preparation of seqid based extraction, the contained seqid's can be tabled:

```
> tableSeqids(ens)

      1 GL000213.1
111      24
```

Extraction of subsets based on seqid can be done with extractSeqids. The sequence id's for extraction are specified as regular expression:

```
> en1 <- extractSeqids(ens, "^1$")
> en1
```

Object of class 'ensemblGenome' with 111 rows and 15 columns.

	id	seqid	start	end	feature	score	strand	frame
25	1	1	11869	12227	exon	.	+	.
34	2	1	11872	12227	exon	.	+	.
41	3	1	11874	12227	exon	.	+	.
28	4	1	12010	12057	exon	.	+	.
29	5	1	12179	12227	exon	.	+	.
35	6	1	12190	12227	CDS	.	+	0

	gene_id	transcript_id	source
25	ENSG00000223972	ENST00000456328	pseudogene
34	ENSG00000249291	ENST00000515242	protein_coding
41	ENSG00000253101	ENST00000518655	protein_coding
28	ENSG00000223972	ENST00000450305	pseudogene
29	ENSG00000223972	ENST00000450305	pseudogene
35	ENSG00000249291	ENST00000515242	protein_coding

	gene_name	transcript_name	exon_number
25	DDX11L1	DDX11L1-002	1
34	AL627309.2	AL627309.2-201	1
41	DDX11L11	DDX11L11-201	1
28	DDX11L1	DDX11L1-001	1
29	DDX11L1	DDX11L1-001	2
35	AL627309.2	AL627309.2-201	1

	transcript_biotype
25	processed_transcript
34	nonsense_mediated_decay
41	nonsense_mediated_decay
28	transcribed_unprocessed_pseudogene
29	transcribed_unprocessed_pseudogene
35	nonsense_mediated_decay

It looks cumbersome for single chromosomes but allows extraction of complex patterns.

5.2 Extracting primary assembly data

Usually the interesting part of the annotation data is the the primary assembly (where alternative haplotypes are excluded). Therefore functions which return the proper terms are supplied:

```
> ensPrimAssembly()

[1] "^([0-9]{1,2})$|^[XY]|MT$"

> ucPrimAssembly()

[1] "^chr[0-9XYM]{1,2}$"
```

Extraction of primary assembly `seqid`'s `i` is done by:

```
> enpa<-extractSeqids(ens,ensPrimAssembly())
> tableSeqids(enpa)
```

```

1
111

> ucpa<-extractSeqids(uc,ucPrimAssembly())
> tableSeqids(ucpa)

chr1
6

```

5.3 Extract features

Subsets defined by features can also be tabled and extracted:

```

> tableFeatures(enpa)

      CDS      exon start_codon stop_codon
      8      98      3      2

> enpf<-extractFeature(enpa,"exon")
> enpf

Object of class 'ensemblGenome' with 98 rows and 15 columns.
  id seqid start  end feature score strand frame
25  1     1 11869 12227  exon     .      +     .
34  2     1 11872 12227  exon     .      +     .
41  3     1 11874 12227  exon     .      +     .
28  4     1 12010 12057  exon     .      +     .
29  5     1 12179 12227  exon     .      +     .
42  8     1 12595 12721  exon     .      +     .
      gene_id  transcript_id      source
25 ENSG00000223972 ENST00000456328  pseudogene
34 ENSG00000249291 ENST00000515242  protein_coding
41 ENSG00000253101 ENST00000518655  protein_coding
28 ENSG00000223972 ENST00000450305  pseudogene
29 ENSG00000223972 ENST00000450305  pseudogene
42 ENSG00000253101 ENST00000518655  protein_coding
      gene_name transcript_name exon_number
25  DDX11L1      DDX11L1-002      1
34 AL627309.2  AL627309.2-201      1
41  DDX11L11    DDX11L11-201      1
28  DDX11L1      DDX11L1-001      1
29  DDX11L1      DDX11L1-001      2
42  DDX11L11    DDX11L11-201      2
      transcript_biotype
25      processed_transcript
34      nonsense_mediated_decay
41      nonsense_mediated_decay
28 transcribed_unprocessed_pseudogene
29 transcribed_unprocessed_pseudogene
42      nonsense_mediated_decay

```

5.4 Extract data for single genes and transcripts

There are some functions which extract objects that contain data for single genes (or transcripts). These functions provide a closer insight into specific regions.

Objects which contain data for vectors of gene-names can be extracted with

```
> dxe <- extractByGeneName(enpa, "DDX11L1")
> dxu <- extractByGeneName(ucpa, "DDX11L1")
```

When gene-names did not match in the gtf-table of the object, a message including all names of not matching gene-names will be printed. When no gene-name matches, a message will be printed and the function returns `NULL`, which can be tested for later on.

Additionally subsets can also be extracted based on gene-id

```
> dxe <- extractByGeneId(enpa, "ENSG00000223972")
> dxu <- extractByGeneId(ucpa, "ENSG00000223972")
```

```
[1] "ENSG00000223972"
```

From these extracts we can view the contained transcripts with the `tableTranscript.id` function:

```
> tableTranscript.id(enpa)

ENST00000408384 ENST00000417324 ENST00000423562
              1              8              10
ENST00000430492 ENST00000438504 ENST00000450305
              9              12              6
ENST00000456328 ENST00000461467 ENST00000469289
              3              2              2
ENST00000473358 ENST00000488147 ENST00000515242
              3              11              7
ENST00000518655 ENST00000537342 ENST00000538476
              8              7              13
ENST00000541675
              9
```

```
> tableTranscript.id(ucpa)
```

```
uc001aaa.3 uc010nrx.1
          3          3
```

Data for interesting transcripts can be extracted by `extractTranscript`:

```
> extractTranscript(ens, "ENST00000456328")
```

Object of class 'ensemblGenome' with 3 rows and 15 columns.

	transcript_id	id	seqid	start	end	feature	score	strand
1	ENST00000456328	1	1	11869	12227	exon	.	+
2	ENST00000456328	9	1	12613	12721	exon	.	+
3	ENST00000456328	14	1	13221	14409	exon	.	+

```

      frame      gene_id      source gene_name
1      . ENSG00000223972 pseudogene  DDX11L1
2      . ENSG00000223972 pseudogene  DDX11L1
3      . ENSG00000223972 pseudogene  DDX11L1
      transcript_name exon_number transcript_biotype
1      DDX11L1-002          1 processed_transcript
2      DDX11L1-002          2 processed_transcript
3      DDX11L1-002          3 processed_transcript

> extractTranscript(uc, "uc010nxr.1")

Object of class 'ucscGenome' with 3 rows and 14 columns.
      transcript_id id seqid start  end feature score strand
1      uc010nxr.1  4  chr1 11874 12227  exon      0      +
2      uc010nxr.1  5  chr1 12646 12697  exon      0      +
3      uc010nxr.1  6  chr1 13221 14409  exon      0      +
      frame      gene_id      source gene_name      ensembl
1      . uc010nxr.1 hg19_knownGene  DDX11L1 ENST00000456328
2      . uc010nxr.1 hg19_knownGene  DDX11L1 ENST00000456328
3      . uc010nxr.1 hg19_knownGene  DDX11L1 ENST00000456328
      clusterId
1      1
2      1
3      1

```

6 Accumulate data for whole genes

The function `getGenePositions` accumulates position data for whole genes. Genes are grouped by `gene_name`. For both, *ensemblGenome* and *ucscGenome* the `gene_name` column is not present after the standard gtf-import. For *ucscGenome*, `addXref` must be used. Respective warnings are thrown.

```

> gpe <- getGenePositions(ens)
> gpe

      id      gene_id gene_name      seqid start  end
2  2 ENSG00000223972  DDX11L1          1  11869 14409
7  7 ENSG00000249291 AL627309.2         1  11872 14412
8  8 ENSG00000253101  DDX11L1          1  11874 14409
3  3 ENSG00000227232  WASH7P           1  14363 29806
6  6 ENSG00000243485 MIR1302-10         1  29554 31109
1  1 ENSG00000221311 MIR1302-10         1  30366 30503
5  5 ENSG00000237613  FAM138A           1  34554 36081
4  4 ENSG00000237375 BX072566.1 GL000213.1 108007 139339
      strand start_codon stop_codon
2      +      NA      NA
7      +     12190      NA
8      +     13548    13817
3      -      NA      NA
6      +      NA      NA

```



```

1      +      NA      NA
5      -      35736    35140
4      -      139287   108028

> gpu <- getGenePositions(uc)
> gpu

   id   gene_id gene_name seqid start   end strand
1  1 uc001aaa.3  DDX11L1  chr1 11874 14409      +
   start_codon stop_codon
1           NA      NA

```

There is a slight difference between both results: The last column is `gene_id` for *ensemblGenome* and `clusterID` for *ucscGenome*. This is due to different information which is available for each.

7 Exon and splice-junction based views (only for Ensembl genomes)

7.1 Extract exon based table

Exon based view on annotation data can be obtained with `ensemblExons` which returns an object of class `ensemblExons`. Basically `ensemblExons` calls `extractFeature` for feature type "exon". Information about presence of cds start or end and start-codon or stop-codon is added.

```

[refExons.refGenome] Extracting tables.
[refExons.refGenome] Adding 'CDS'.
[refExons.refGenome] Adding 'start_codon'.
[refExons.refGenome] Adding 'stop_codon'.
[refExons.refGenome] Finished.

```

```

[refExons.refGenome] Extracting tables.
[refExons.refGenome] Adding 'CDS'.
[refExons.refGenome] Adding 'start_codon'.
[refExons.refGenome] Adding 'stop_codon'.
[refExons.refGenome] Finished.

```

```
> enex
```

Object of class 'ensemblExons' with 109 rows and 18 columns.

```

   id seqid start   end score strand frame   gene_id
53  1     1 11869 12227     .      +     . ENSG00000223972
74  2     1 11872 12227     .      +     . ENSG00000249291
77  3     1 11874 12227     .      +     . ENSG00000253101
47  4     1 12010 12057     .      +     . ENSG00000223972
48  5     1 12179 12227     .      +     . ENSG00000223972
78  8     1 12595 12721     .      +     . ENSG00000253101
   transcript_id      source gene_name
53 ENST00000456328    pseudogene   DDX11L1

```

74	ENST00000515242	protein_coding	AL627309.2	
77	ENST00000518655	protein_coding	DDX11L11	
47	ENST00000450305	pseudogene	DDX11L1	
48	ENST00000450305	pseudogene	DDX11L1	
78	ENST00000518655	protein_coding	DDX11L11	
	transcript_name	exon_number		
53	DDX11L1-002	1		
74	AL627309.2-201	1		
77	DDX11L11-201	1		
47	DDX11L1-001	1		
48	DDX11L1-001	2		
78	DDX11L11-201	2		
	transcript_biotype	cds_start	cds_end	
53	processed_transcript	NA	NA	
74	nonsense_mediated_decay	318	0	
77	nonsense_mediated_decay	NA	NA	
47	transcribed_unprocessed_pseudogene	NA	NA	
48	transcribed_unprocessed_pseudogene	NA	NA	
78	nonsense_mediated_decay	NA	NA	
	start_codon	stop_codon		
53	NA	NA		
74	318	NA		
77	NA	NA		
47	NA	NA		
48	NA	NA		
78	NA	NA		

7.2 Extract splice-junction based views from `ensemblExons`

From `ensemblExons` information about adjacency of exons (which defines annotated splice-sites) can be obtained by putting exons with equal `transcript_id` and subsequent `exon_number` side by side.

The start and end positions of adjacent exons are renamed to `lstart`, `lend` and `rstart` and `rend`. The "l" prefix refers to the exon with lower start and end coordinates (i.e. left, lower `exon_number`). The "r" prefix refers to the exons with higher start and end coordinates (i.e. right, higher `exon_number`).

Setting `coding=TRUE` will restrict the result to exons for which `source` and `gene_biotype` equal "protein_coding".

```
> jens <- getSpliceTable(ens)
> jens
```

Object of class 'ensemblJunctions' with 92 rows and 13 columns.

	id	seqid	lstart	lend	rstart	rend	gene_id
1	1	GL000213.1	108007	108247	109884	110007	ENSG000000237375
2	2	GL000213.1	109884	110007	118422	118588	ENSG000000237375
3	3	GL000213.1	118422	118588	119629	119673	ENSG000000237375
4	4	GL000213.1	119629	119673	121073	121143	ENSG000000237375
5	5	GL000213.1	121073	121143	126648	126718	ENSG000000237375
6	6	GL000213.1	126648	126718	129228	129365	ENSG000000237375
	gene_name	strand	transcript_id	lexid	rexid		

```

1 BX072566.1      - ENST00000327822    112    115
2 BX072566.1      - ENST00000327822    115    117
3 BX072566.1      - ENST00000327822    117    119
4 BX072566.1      - ENST00000327822    119    121
5 BX072566.1      - ENST00000327822    121    123
6 BX072566.1      - ENST00000327822    123    125

```

```

      transcript_biotype
1 nonsense_mediated_decay
2 nonsense_mediated_decay
3 nonsense_mediated_decay
4 nonsense_mediated_decay
5 nonsense_mediated_decay
6 nonsense_mediated_decay

```

```

> juc <- getSpliceTable(uc)
> juc

```

Object of class 'ucscJunctions' with 4 rows and 12 columns.

	id	seqid	lstart	lend	rstart	rend	gene_id	gene_name
1	1	chr1	11874	12227	12613	12721	uc001aaa.3	DDX11L1
2	2	chr1	12613	12721	13221	14409	uc001aaa.3	DDX11L1
3	3	chr1	11874	12227	12646	12697	uc010nxr.1	DDX11L1
4	4	chr1	12646	12697	13221	14409	uc010nxr.1	DDX11L1

	strand	transcript_id	lexid	rexid
1	+	uc001aaa.3	1	2
2	+	uc001aaa.3	2	3
3	+	uc010nxr.1	4	5
4	+	uc010nxr.1	5	6

This generally leads to repeated occurrence of start and end positions when a splice-junction is contained in multiple transcripts. Additionally a handful splice-sites with multiple gene-id's are present.

The `unifyJunc` therefore calculates `nGenes` which represents the multiplicity of each gene-id at each splice-site and then selects a gene-id for which `nGenes` is maximal.

`unifyJuncs` adds a `uid` column to the basic `gtf` table which identifies each `seqid`, left-end, right-start combination uniquely. `unifyJuncs` also adds a new `ujc` table inside the contained environment.

`getUnifiedJuncs` takes the result of `unifyJuncs` and adds `gene_name` and strand information.

```

> ujens <- unifyJuncs(jens)
> ujuc <- unifyJuncs(juc)
> jeg <- getGenePositions(jens)
> jug <- getGenePositions(juc)
> ujens

```

Object of class 'ensemblJunctions' with 51 rows and 12 columns.

	id	seqid	lstart	lend	rstart	rend	nSites	gene_id
1	1	1	12010	12057	12179	12227	1	ENSG00000223972
2	2	1	11874	12227	12595	12721	1	ENSG00000253101

```

3 3      1 11869 12227 12613 12721      3 ENSG00000223972
4 4      1 12613 12697 12975 13052      1 ENSG00000223972
5 5      1 12613 12721 13221 14409      1 ENSG00000223972
6 6      1 12613 12721 13225 14412      1 ENSG00000249291

```

```

strand fexid cnNmd gene_name
1      +   41     1   DDX11L1
2      +   64     0   DDX11L1
3      +   42     2   DDX11L1
4      +   43     1   DDX11L1
5      +   47     1   DDX11L1
6      +   63     0 AL627309.2

```

```
> jug
```

```

id      gene_id gene_name seqid start  end strand
1 1 uc001aaa.3   DDX11L1 chr1 11874 14409      +
start_codon stop_codon
1          NA          NA

```

The result tables of `unifyJuncs` and `getGenePositions` are stored inside the internal environment of `ensemblJunctions`. From there, the results can easily be reproduced without recalculation. The tables are automatically included in `saveGenome` and `load.ensembl.juncs` mechanisms.

8 Overlapping

The `overlap` function is used to supply annotation for genomic ranges. The function takes two `data.frame`'s which contain query (`qry`) and reference (`ref`) ranges respectively. Each dataset will be identified by its id.

The routine assumes that query and reference tables are ascending sorted by column 'start'. Otherwise the result will be incorrect (i.e. missing hits). The function assumes that there is no overlap between reference ranges. It will otherwise return only one, possibly arbitrary, hit per query range.

The function returns a `data.frame`. For each query range, there will be one row.

```

> qry<-data.frame(
+               id=1:6,
+               start=c(10,18,61,78,82,110),
+               end=c(15,22,63,87,90,120))
> ref<-data.frame(
+               id=1:5,
+               start=c(20,40,60,80,100),
+               end=c(25,45,65,85,105))
> overlap(qry,ref)

overlap leftDiff rightDiff queryid refid
0      no        0         5         1      0

```

1	l	2	3	2	1
2	n	1	2	3	3
3	b	2	2	4	4
4	r	2	5	5	4
5	no	5	0	6	0

The query and reference record are identified by "queryid" and "refid". The type of overlap is encoded in the "overlap" column. The overlap encodings are explained as follows:

- **no**. The query range does not overlap with any reference ranges.
- **l** The query range overhangs the matching reference range on the left (lower coordinate) side.
- **n** The query range is completely contained within a reference range. There is no overhang.
- **b** The query range overhangs the matching reference range on both sides.
- **r** The query range overhangs the matching reference range on the right (higher coordinate) side.

The added "leftDiff" and "rightDiff" columns contain the distance between the query and reference range boundaries: leftDiff is the difference between the left (lower coordinate) margins and rightDiff is the difference between the right (higher coordinate) margins.

8.1 Overlapping for splice-junctions

The `overlapJuncs` function is specialized for annotation of splice events in BAM alignment data. The function takes two arguments: a `data.frame` providing query data and a `refJunctions` object providing annotation data.

8.1.1 Query data

Query data for `overlapJuncs` are BAM alignment gap-sites. These gap-sites are defined by one or multiple gapped alignments of RNA-seq reads to a reference genome. Due to the splicing process, a variable fraction of the reads align in two or more fractions to the reference genome. Two adjacent alignment fractions define the position of a gap-site.

The position of the alignment fractions are named **lstart** and **lend** for the left (i.e. lower genomic coordinates) alignment interval and **rstart** and **rend** for the right (i.e. higher genomic coordinates) alignment interval.

The two genomic intervals of a gap-site implicitly define a third interval: The gap in between (alignment gap).

The biological meaning of gap-sites is that alignment gaps correspond to introns and the left and right align regions correspond to adjacent exon boundaries.

Due to the fact that gapped alignments do not cover whole exons, the inner boundaries of gap-sites are biological meaningful and the outer boundaries of gap-sites usually are technical artifacts.

Therefore the `overlapJuncs` function searches and rates overlaps by inner gap boundaries (`lend`, `rstart`) whereas the outer boundaries solely define whether

any overlap with annotated splice-sites is present.

The **query data.frame** defines one gap-site in each row. Therefore the table is expected to contain the following columns:

- **id**: Consecutive (unique) integral values.
- **seqid**: Character values for reference sequence identifiers (e.g. 'chr1' or '1').
- **lstart**: Position of first nucleotide of left alignment region
- **lend**: Position of last nucleotide of left alignment region.
- **rstart**: Position of first nucleotide of right alignment region.
- **rend**: Position of last nucleotide of right alignment region.

All genomic positions are 1-based (i.e. first position of sequence has position 1).

```
> # + + + + + + + + + + + + + + + #
> # A) Example query data
> # + + + + + + + + + + + + + + + #
> ##              1          2          3          4          5          6          7 ##
> qry <- data.frame(id = 1:7, seqid = "1",
+                   lstart = c(10100L, 11800L, 12220L, 12220L, 12220L, 32000L, 40000L),
+                   lend = c(10100L, 12000L, 12225L, 12227L, 12227L, 32100L, 40100L),
+                   rstart = c(10200L, 12200L, 12057L, 12613L, 12650L, 32200L, 40200L),
+                   rend = c(10300L, 12250L, 12179L, 12620L, 12700L, 32300L, 40300L))
> ##              1          2          3          4          5          6          7 ##
```

8.1.2 Reference data

The reference data for the `overlapJuncs` function is can be obtained from a `refGenome` object with the `getSpliceJuncs` function.

```
> ensfile <- system.file("extdata", "hs.ensembl.62.small.RData",
+                         package="refGenome")
> # Load Ensembl genome
> ens <- loadGenome(ensfile)
> # Calculate junction positions:
> junc <- getSpliceTable(ens)
```

8.1.3 Overlapping results

Junction based annotation of annotation gap-sites is done by the `overlapJuncs` function.

```
> res <- overlapJuncs(qry, junc)
```

The `overlapJuncs` function returns a `data.frame`. As an overlap result is returned for every query record, the output and the query `data.frame` contain the same number of rows.

The first two columns (`qid` and `refid`) provide the query id (the id from the query `data.frame`) and a `refid` (the id for the identified optimal overlap). Records without overlap can be identified with a `refid` value of 0.

The validity of the overlap is defined by the distance of the inner boundaries between the query and the reference site. The distance for the left alignment region is given in `ldiff` and the distance for the right alignment region is given in `rdiff`. The sum of the absolute `ldiff` and `rdiff` values is given by `sod` (sum of distances). The optimal hit is defined as the one with the lowest `sod` value. An exact hit will have `sod=0`.

The result table provides `gene_id`, `transcript_id`, `gene_name` and `strand`.

9 Workflows

9.1 Establish a standard set of `refGenome` objects for Ensembl

The following example assumes, that a downloaded *GTF* file has been downloaded and unzipped into a target location (`endir`):

```
> library(refGenome)
> endir <- "/.../refGenomes/hsEns76"
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> # Read GTF
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> en76 <- ensemblGenome()
> basedir(en76) <- endir
> read.gtf(en76, "Homo_sapiens.GRCh38.76.gtf")
> saveGenome(en76, "en76.RData")
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> # Extract primary assembly
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> enpa76 <- extractSeqids(en76, ensPrimAssembly())
> saveGenome(enpa76, "enpa76.RData")
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> # Extract Exons
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> enex76 <- refExons(enpa76)
> saveGenome(enex76, "enex76.RData")
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> # Extract Junctions
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> enjc76 <- getSplICEtable(enpa76)
> saveGenome(enjc76, "enjc76.RData")
>
> # + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + #
> # Extract data.frame
```

[illegible]

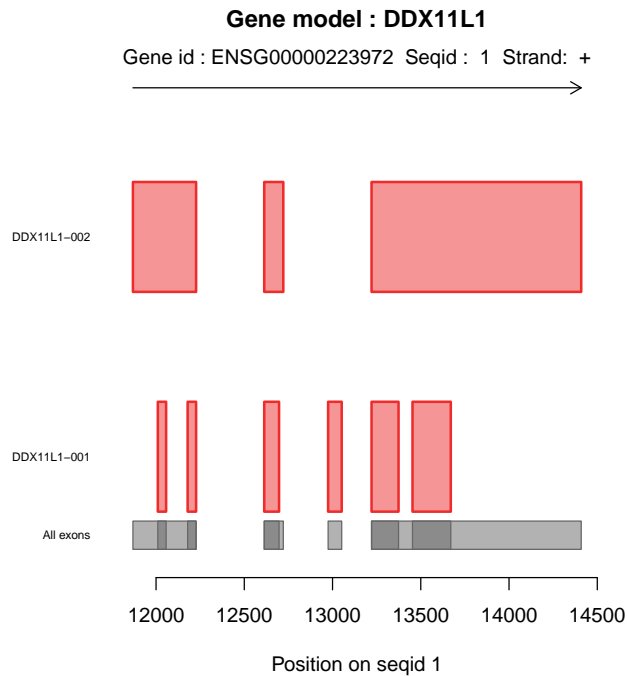
10 Gene model

Genomic annotation data expands into a complex variety of qualities for each gene. The `geneModel` and `transcriptModel` classes capture qualities for single genes and transcripts.

10.1 geneModel objects

Gene model objects contain collected data for one single gene for example gene-name, gene-id and strand as well as tables with coordinates for single exons and a list of transcript objects.

```
> ensfile<-system.file("extdata",
+      "hs.ensembl.62.small.RData", package="refGenome")
> ens<-loadGenome(ensfile)
> gt <- getGeneTable(ens)
> gene_id <- as.character(gt$gene_id[1])
> gm <- geneModel(ens, gene_id)
> plot(gm)
```



Further exploration of exons can be done using `transcriptModel` objects which is the appropriate method when a gene consists of many transcripts or many exons.

10.2 transcriptModel object

Objects of class `transcriptModel` contain data on single transcripts like gene-id, gene-name, strand or coordinates.

Transcript objects can be extracted from `geneModel` objects by name and by index which is the usual way to create `transcriptModel` objects.

10.2.1 Extraction by index

```
> tr <- getTranscript(gm, 1)
> tr
```

An object of class 'transcriptModel'.

```
ID           : ENST00000450305
Name          : DDX11L1-001
Gene ID       : ENSG00000223972
Gene Name     : DDX11L1
Start         : 12.010                      End           : 13.670
Start codon   : NA                          Stop codon    : NA
5' prime utr : NA                          3' prime utr : NA
Seq Name      : 1
Strand        : +
```

10.2.2 Extraction by ID

```
> tr <- getTranscript(gm, "ENST00000450305")
> tr
```

An object of class 'transcriptModel'.

```
ID           : ENST00000450305
Name          : DDX11L1-001
Gene ID       : ENSG00000223972
Gene Name     : DDX11L1
Start         : 12.010                      End           : 13.670
Start codon   : NA                          Stop codon    : NA
5' prime utr : NA                          3' prime utr : NA
Seq Name      : 1
Strand        : +
```

From a `transcriptModel` object, exon and CDS tables can be extracted.

```
> getExonData(tr)
```

	start	end	exon_number	seqid	strand
1	12010	12057	1	1	+
2	12179	12227	2	1	+
3	12613	12697	3	1	+
4	12975	13052	4	1	+
5	13221	13374	5	1	+
6	13453	13670	6	1	+

```
> getCdsData(tr)
```

	start	end	exon_number	seqid	strand
1	12010	12057	1	1	+
2	12179	12227	2	1	+
3	12613	12697	3	1	+
4	12975	13052	4	1	+
5	13221	13374	5	1	+
6	13453	13670	6	1	+

11 Gene model objects

11.1 Gene model

Gene model objects organize data for single genes. The Gene model objects are created using gene-id's from `refGenome` objects as shown below.

```
> ensfile<-system.file("extdata",
+   "hs_en82_grch38_small.gtf", package="refGenome")
> en82s <- ensemblGenome()
> basedir(en82s) <- dirname(ensfile)
> read.gtf(en82s, "hs_en82_grch38_small.gtf")

[read.gtf.refGenome] Reading file 'hs_en82_grch38_small.gtf'.

[GTf]          64 lines processed.
[read.gtf.refGenome] Extracting genes table.
[read.gtf.refGenome] Found 9 gene records.
[read.gtf.refGenome] Finished.

> gt <- getGeneTable(en82s)

> gm <- geneModel(en82s, "ENSG00000227232")
```

11.2 Transcript model

Transcript model objects contain data for a single transcript. They are intended for use inside `geneModel` objects.

```
> tr <- getTranscript(gm, 1)
> tr

An object of class 'transcriptModel'.
ID          : ENST00000488147
Name        : WASH7P-001
Gene ID     : ENSG00000227232
Gene Name   : WASH7P
Start       : 14.404                      End           : 29.570
Start codon : NA                          Stop codon      : NA
5' prime utr: NA                          3' prime utr    : NA
Seq Name    : 1
Strand      : -
```