Making and Utilizing TranscriptDb Objects

Marc Carlson, Patrick Aboyoun, Herve Pages, Seth Falcon and Martin Morgaan

August 17, 2010

1 Introduction

The *GenomicFeatures* package retrieves and manages transcript-related features from UCSC Genome Bioinformatics and BioMart data resources. The package is useful for ChIP-chip, ChIP-seq, and RNA-seq analyses.

```
> library(GenomicFeatures)
```

2 Transcript Metadata

2.1 *TranscriptDb* Objects

The *GenomicFeatures* package uses *TranscriptDb* objects to store transcript metadata. This class is designed to map the 5' untranslated regions (UTRs), protein coding sequence (CDSs), and 3' UTRs for a set mRNA transcripts to their associated genome, where the combined 5' UTR, CDS, and 3' UTR region for an mRNA transcript either originated from a single exon or from multiple exons that were post-transcriptionally spliced.

As the suffix of the class name suggests, *TranscriptDb* objects are backed by a SQLite database. This database manages genomic locations and the relationships between pre-processed mRNA transcripts, exons, protein coding sequences, and their related gene IDs.

2.2 Creating New *TranscriptDb* Objects

There are three methods for creating new *TranscriptDb* objects in the *GenomicFeatures* package:

1. Use `makeTranscriptDbFromUCSC` to download from UCSC Genome Bioinformatics.

2. Use `makeTranscriptDbFromBiomart` to download from BioMart.

3. Use a *data.frame* containing transcript metadata with `makeTranscriptDb` to make a custom database.

The function `makeTranscriptDbFromUCSC` downloads UCSC Genome Bioinformatics transcript tables (e.g. `"knownGene"`, `"refGene"`, `"ensGene"`) for a genome build (e.g. `"mm9"`, `"hg19"`). Use the `supportedUCSCtables` utility function to get the list of supported tables.

```
> supportedUCSCtables()
```

```
                                track           subtrack
knownGene                   UCSC Genes               <NA>
knownGeneOld3           Old UCSC Genes               <NA>
wgEncodeGencodeManualV3  Gencode Genes    Genecode Manual
wgEncodeGencodeAutoV3    Gencode Genes      Genecode Auto
wgEncodeGencodePolyaV3   Gencode Genes     Genecode PolyA
ccdsGene                         CCDS               <NA>
refGene                   RefSeq Genes               <NA>
xenoRefGene               Other RefSeq               <NA>
vegaGene                    Vega Genes Vega Protein Genes
vegaPseudoGene              Vega Genes   Vega Pseudogenes
ensGene                   Ensembl Genes               <NA>
acembly                   AceView Genes               <NA>
sibGene                       SIB Genes               <NA>
nscanPasaGene                    N-SCAN    N-SCAN PASA-EST
nscanGene                        N-SCAN             N-SCAN
sgdGene                       SGD Genes               <NA>
sgpGene                       SGP Genes               <NA>
geneid                     Geneid Genes               <NA>
genscan                   Genscan Genes               <NA>
exoniphy                       Exoniphy               <NA>
augustusHints                  Augustus     Augustus Hints
augustusXRA                    Augustus   Augustus De Novo
augustusAbinitio               Augustus Augustus Ab Initio
acescan                         ACEScan               <NA>
```

```
> mm9KG <- makeTranscriptDbFromUCSC(genome = "mm9", tablename = "knownGene")
```

Retrieve data from BioMart by specifying the 'mart' and data set (not all BioMart data sets are currently supported):

```
> mmusculusEnsembl <-
+   makeTranscriptDbFromBiomart(biomart = "ensembl",
+                               dataset = "mmusculus_gene_ensembl")
```

The function `makeTranscriptDb` creates *TransctriptDb* objects from *data.frame* objects.

2.3 Saving and Loading a *TranscriptDb* Object

TranscriptDb objects can be saved as SQLite files for future access (e.g., to easily reproduce results with identical genomic feature data at a later date, or for access from programs other than R).

```
> saveFeatures(mm9KG, file="fileName.sqlite")
```

Load a saved *TranscriptDb* object with `loadFeatures`:

```
> mm9KG <- loadFeatures("fileName.sqlite")
```

For instance, a sample of UCSC known genes is included in *GenomicFeatures*.

```
> exampleFile <-
+     system.file("extdata", "UCSC_knownGene_sample.sqlite",
+                 package="GenomicFeatures")
> txdb <- loadFeatures(exampleFile)
> txdb

TranscriptDb object:
| Db type: TranscriptDb
| Data source: UCSC
| Genome: hg18
| UCSC Table: knownGene
| Type of Gene ID: Entrez Gene ID
| Full dataset: no
| transcript_nrow: 135
| exon_nrow: 544
| cds_nrow: 324
| Db created by: GenomicFeatures package from Bioconductor
| Creation time: 2010-03-25 19:49:07 -0700 (Thu, 25 Mar 2010)
| GenomicFeatures version at creation time: 0.4.9
| RSQLite version at creation time: 0.8-4
```

In addition to transcript data, the object contains information about how it was created (e.g., the number of transcripts, exons, and coding sequence rows) and about software versions and creation dates.

3 Retrieving Transcript, Exon, and Coding Sequence Ranges

3.1 Working with Basic Features

The most basic operation for getting data out of a transcriptDb object is to simple retrieve the ranges of exons, transcripts or coding sequences into a GRanges object. For this purpose, the functions `transcripts`, `exons`, and `cds` have been provided.

So as an example, you can use `transcripts` to simply retrieve all the transcripts present in a *TranscriptDb* object and return them as a single *GRanges* object like this:

```
> txdb <- loadFeatures(system.file("extdata", "UCSC_knownGene_sample.sqlite",
+                                  package="GenomicFeatures"))
> GR <- transcripts(txdb)
> GR

GRanges with 135 ranges and 2 elementMetadata values
          seqnames            ranges strand   |     tx_id     tx_name
             <Rle>         <IRanges>  <Rle>   | <integer> <character>
  [1]        chr1 [  1116,   4121]      +   |         1  uc001aaa.2
  [2]        chr1 [  1116,   4272]      +   |         2  uc009vip.1
  [3]        chr1 [  4269,   6628]      -   |         3  uc009vis.1
  [4]       chr10 [170643, 285201]      +   |        39  uc001ifj.1
  [5]       chr10 [ 82997,  85178]      -   |        37  uc001ifi.1
  [6]       chr10 [ 82997,  85178]      -   |        38  uc009xhe.1
  [7]       chr11 [117131, 118022]      -   |        40  uc009ybr.1
  [8]       chr11 [121174, 121242]      -   |        41  uc001lnw.1
  [9]       chr11 [123614, 129117]      -   |        42  uc001lnx.2
  ...         ...             ...    ... ...       ...         ...
[127]        chrX [132991, 160020]      +   |        32  uc004cpa.1
[128]        chrX [138079, 156125]      +   |        31  uc004cpb.1
[129]        chrX [157539, 160020]      +   |        33  uc004cpc.1
[130] chrX_random [ 46059, 438262]      +   |       105  uc004fny.2
[131] chrX_random [ 68697,  80051]      +   |       103  uc004fnz.2
```

```
[132] chrX_random [281521, 289293]      +   |       104  uc004foa.2
[133]        chrY [132991, 160020]      +   |        35  uc004fon.1
[134]        chrY [138079, 156125]      +   |        34  uc004foo.1
[135]        chrY [157539, 160020]      +   |        36  uc004fop.1

seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

Now suppose that you wanted to further refine things and retrieve only the things that are present on the plus strand of chromosome 1. Well the `transcripts` function will allow you to do things like that too.

```
> GR <- transcripts(txdb, vals <- list(tx_chrom = "chr1", tx_strand = "+"))
> GR

GRanges with 2 ranges and 2 elementMetadata values
    seqnames       ranges strand |     tx_id     tx_name
       <Rle>    <IRanges>  <Rle> | <integer> <character>
[1]     chr1 [1116, 4121]      + |         1  uc001aaa.2
[2]     chr1 [1116, 4272]      + |         2  uc009vip.1

seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

The `exons`, and `cds` functions have a similar role to play and are used to retrieve just the exons or just the coding sequences.

3.2 Working with Grouped Features

Often it will not be enough to just get exons, cds or transcripts only. Sometimes you will want to give greater consideration to the context of the data. In these cases, you will want to think of the ranged annotation data as being grouped in a specific way, for example, you might want to consider that the transcripts are each associated with a specific gene or the exon composition of specific transcripts. These relationships are maintained with *TranscriptDb* objects and can be accessed through the `transcriptsBy`, `exonsBy`, and `cdsBy` functions.

So for example, to extract all the transcripts grouped by their exons you can call:

```
> txdb <- loadFeatures(system.file("extdata", "UCSC_knownGene_sample.sqlite",
+                                  package="GenomicFeatures"))
> GRList <- transcriptsBy(txdb, "gene")
> GRList

GRangesList of length 51
$100132288
GRanges with 1 range and 2 elementMetadata values
        seqnames           ranges strand |     tx_id     tx_name
           <Rle>        <IRanges>  <Rle> | <integer> <character>
[1] chr21_random [103280, 164670]      - |       120  uc002zka.1

$10752
GRanges with 3 ranges and 2 elementMetadata values
    seqnames           ranges strand |     tx_id     tx_name
       <Rle>        <IRanges>  <Rle> | <integer> <character>
[1]     chr3 [213650, 426097]      + |         7  uc003bot.1
[2]     chr3 [213650, 426097]      + |         8  uc003bou.1
[3]     chr3 [214327, 265280]      + |         9  uc003bov.1

$10771
GRanges with 1 range and 2 elementMetadata values
    seqnames           ranges strand |     tx_id     tx_name
       <Rle>        <IRanges>  <Rle> | <integer> <character>
[1]    chr10 [170643, 285201]      + |        39  uc001ifj.1

...
<48 more elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

Similarly, to extract all the exons for each transcript you can call:

```
> txdb <- loadFeatures(system.file("extdata", "UCSC_knownGene_sample.sqlite",
+                                  package="GenomicFeatures"))
> GRList <- exonsBy(txdb, "tx")
> GRList
```

```
GRangesList of length 135
$1
GRanges with 3 ranges and 3 elementMetadata values
    seqnames       ranges strand |   exon_id   exon_name exon_rank
       <Rle>    <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr1 [1116, 2090]      + |         1          NA         1
[2]     chr1 [2476, 2584]      + |         2          NA         2
[3]     chr1 [3084, 4121]      + |         3          NA         3

$2
GRanges with 2 ranges and 3 elementMetadata values
    seqnames       ranges strand |   exon_id   exon_name exon_rank
       <Rle>    <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr1 [1116, 2090]      + |         1          NA         1
[2]     chr1 [2476, 4272]      + |         4          NA         2

$3
GRanges with 4 ranges and 3 elementMetadata values
    seqnames       ranges strand |   exon_id   exon_name exon_rank
       <Rle>    <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr1 [6470, 6628]      - |         8          NA         1
[2]     chr1 [5659, 5805]      - |         7          NA         2
[3]     chr1 [4833, 4901]      - |         6          NA         3
[4]     chr1 [4269, 4692]      - |         5          NA         4

...
<132 more elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

These functions return *GRangesList* objects that contain locations and identifiers all grouped according to the type of feature specified. These objects can be used in downstream analyses such as using `findOverlaps` contextualize the alignments from high-throughput sequencing.

It is important to consider the context created when grouping by a particular feature. For example, in the 1st example, where we grouped by genes, the name of the features is an Entrez Gene ID. If the database had been

based instead on Ensembl sources, it would be an Ensembl Gene ID. However, in the second example where we group by transcript, we see that the groups are labeled by an ID that is not a traditional transcript ID. In this second case, we have been given an internally assigned database ID. This is because some sources may choose to overload their use of traditional transcript IDs in ways that would make the existence of our database schema impossible. So in the second case we have to use an internal id to guarantee uniqueness. This will happen whenever you group by anything that is not a gene. So when you want to use traditional transcript IDs you can look them up using the appropriate basic accessors described in the preceding section.

Another case where context can matter is when considering the order of the elements returned. In most cases the grouped elements will be listed in the order that they occur along the chromosome. But in the context where you have grouped exons or CDS by transcripts, they will instead be grouped according to their position along the transcript itself. This is important because alternative splicing can mean that the order along the transcript can be different from that along the chromosome.

3.3 Prespecfied grouping functions

An important kind of grouping functions are the ones that group in a prespecified manner. The `intronsByTranscript`, `fiveUTRsByTranscript` and `threeUTRsByTranscript` functions are like this. These functions retrieve the *GRangesList* objects for the introns, 5' UTR's, and 3' UTR's grouped by transcript respectively.

```
> intronsByTranscript(txdb)

GRangesList of length 135
$1
GRanges with 2 ranges and 0 elementMetadata values
    seqnames       ranges strand |
       <Rle>    <IRanges>  <Rle> |
[1]     chr1 [2091, 2475]      + |
[2]     chr1 [2585, 3083]      + |

$2
GRanges with 1 range and 0 elementMetadata values
    seqnames       ranges strand |
       <Rle>    <IRanges>  <Rle> |
```

```
[1]     chr1 [2091, 2475]      + |

$3
GRanges with 3 ranges and 0 elementMetadata values
    seqnames       ranges strand |
       <Rle>    <IRanges>  <Rle> |
[1]     chr1 [4693, 4832]      - |
[2]     chr1 [4902, 5658]      - |
[3]     chr1 [5806, 6469]      - |

...
<132 more elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

```
> fiveUTRsByTranscript(txdb)

GRangesList of length 61
$7
GRanges with 3 ranges and 3 elementMetadata values
    seqnames           ranges strand |   exon_id   exon_name exon_rank
       <Rle>        <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr3 [213650, 213746]      + |        20          NA         1
[2]     chr3 [261296, 261375]      + |        21          NA         2
[3]     chr3 [336366, 336459]      + |        22          NA         3

$8
GRanges with 3 ranges and 3 elementMetadata values
    seqnames           ranges strand |   exon_id   exon_name exon_rank
       <Rle>        <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr3 [213650, 213746]      + |        20          NA         1
[2]     chr3 [261296, 261375]      + |        21          NA         2
[3]     chr3 [336366, 336459]      + |        22          NA         3

$10
GRanges with 1 range and 3 elementMetadata values
    seqnames         ranges strand |   exon_id   exon_name exon_rank
```

```
       <Rle>      <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr4 [43227, 43382]      + |        50          NA         1

...
<58 more elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954

> threeUTRsByTranscript(txdb)

GRangesList of length 58
$7
GRanges with 1 range and 3 elementMetadata values
    seqnames           ranges strand |   exon_id   exon_name exon_rank
       <Rle>        <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr3 [422395, 426097]      + |        47          NA        28

$8
GRanges with 1 range and 3 elementMetadata values
    seqnames           ranges strand |   exon_id   exon_name exon_rank
       <Rle>        <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr3 [422395, 426097]      + |        47          NA        27

$10
GRanges with 1 range and 3 elementMetadata values
    seqnames           ranges strand |   exon_id   exon_name exon_rank
       <Rle>        <IRanges>  <Rle> | <integer> <character> <integer>
[1]     chr4 [145913, 146490]      + |        53          NA         4

...
<55 more elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

3.4 Convenience functions

The `transcriptsByOverlaps`, `exonsByOverlaps` and `cdsByOverlaps` functions return a *GRangesList* object containing data about transcripts, exons, or coding sequences that overlap genomic coordinates specified by a *GRanges* object.

```
> library(org.Mm.eg.db)
> set.seed(0L)
> idx <- sample(length(org.Mm.egCHRLOC), 50)
> tbl <- unique(merge(toTable(org.Mm.egCHRLOC[idx]),
+                     toTable(org.Mm.egCHRLOCEND[idx])))
> gr <- with(tbl, {
+     lvls <- paste("chr", c(1:19, "X", "Y", "MT", "Un"), sep="")
+     GRanges(seqnames=factor(paste("chr", Chromosome, sep=""),
+                             levels=lvls),
+             ranges=IRanges(abs(start_location), abs(end_location)),
+             strand=ifelse(start_location >= 0, "+", "-"),
+             egid=gene_id)
+ })
> transcriptsByOverlaps(txdb, gr)

GRanges with 0 ranges and 2 elementMetadata values
     seqnames ranges strand | tx_id tx_name

seqlengths
          chr1   chr1_random          chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954
```

The convenience functions can be a great shortcut, but because they have to make assumptions about how the results are compared and represented, they are ultimately not as flexible as just using the basic and grouping accessors in combination with `findOverlaps`.

4 Examples

Let's suppose that you have run an experiment. After mapping all your reads to a genome and collapsing them into a set of ranges, you want to find out what genomic Features a particular range overlaps with. How would be the usual way to proceed? Here is an example:

4.1 Retrieving data from an RNA-seq experiment

Let's consider the case where you have some RNA-seq data and you want to convert your ranges into counts representing how hits per transcript. For this example, let's also assume that you are only interested in counting ranges that overlap with exons (not introns).

First lets say that this is your data:

```
> gr <- GRanges(seqnames = rep("chr5",4),
+               ranges = IRanges(start = c(244620,244670,245804,247502),
+                                end = c(244652,244702,245836,247534)),
+               strand = rep("+",4))
```

From our *TranscriptDb* object, we want to recover the annotations for all of the relevant exons, but grouped according to their transcripts. Therefore, we want to use `exonsby` and group them by transcripts.

```
> annotGr <- exonsBy(txdb, "tx")
```

Then we need to used `findOverlaps` to learn which of our data ranges, `gr`, will overlap with the in exons that we have grouped by transcripts.

```
> OL <- findOverlaps(annotGr, gr)
```

Finally, once we have called `findOverlaps` we can subset out the annotations that meet our criteria. The `subjectHits` method will allow us to retrieve only the things that overlapped with the subject in our original `findOverlaps` call. And once we have subsetted out annotations in this way, the length of the resulting *GRangesList* object is also the number of transcripts that overlap with our data.

```
> tdata <- annotGr[subjectHits(OL),]
> tdata

GRangesList of length 0
<0 elements>


seqlengths
          chr1   chr1_random         chr10 ...   chrX_random          chrY
     247249719       1663265     135374737 ...       1719168      57772954

> length(tdata)
```

```
[1] 0
```

By using `findOverlaps` along with the different accessors in this way, it is possible to connect any data that has been represented as a *GRanges* object with the annotations stored in a *TranscriptDb* object. Calling `findOverlaps` along with the appropriate *GRanges* object not only allows users to quickly determine what has overlapped, but also controls what criteria are used for determining whether an overlap has occurred. This can be done by passing in an alternate `type` parameter to `findOverlaps`. In addition, because the basic accesors allow for the users to retrieve data grouped in different ways, the user has control over which parts of a transcript or gene are included in the overlap.

5 Session Information

The version number of R and packages loaded for generating the vignette were:

```
R version 2.11.1 (2010-05-31)
i386-pc-mingw32

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] org.Mm.eg.db_2.4.1     RSQLite_0.9-2          DBI_0.2-5
 [4] AnnotationDbi_1.10.2   Biobase_2.8.0          GenomicFeatures_1.0.10
 [7] GenomicRanges_1.0.7    IRanges_1.6.14         rtracklayer_1.8.1
[10] RCurl_1.4-3            bitops_1.0-4.1

loaded via a namespace (and not attached):
[1] biomaRt_2.4.0     Biostrings_2.16.9 BSgenome_1.16.5   XML_3.1-1
```