# Making and Utilizing TxDb Objects

Marc Carlson, Patrick Aboyoun, Herv Pags, Seth Falcon, Martin Morgan

March 30, 2017

## 1 Introduction

The *GenomicFeatures* package retrieves and manages transcript-related features from the UCSC Genome Bioinformatics[1] and BioMart[2] data resources. The package is useful for ChIP-chip, ChIP-seq, and RNA-seq analyses.

```
library("GenomicFeatures")
```

```
## Loading required package:  BiocGenerics
```

```
## Loading required package:  parallel
```

```
##
## Attaching package:  'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ, clusterExport,
##    clusterMap, parApply, parCapply, parLapply, parLapplyLB, parRapply,
##    parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##    IQR, mad, xtabs
```

```
## The following objects are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, append, as.data.frame,
##    cbind, colnames, do.call, duplicated, eval, evalq, get, grep, grepl,
##    intersect, is.unsorted, lapply, lengths, mapply, match, mget, order, paste,
##    pmax, pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply, setdiff,
##    sort, table, tapply, union, unique, unsplit, which, which.max, which.min
```

```
## Loading required package:  S4Vectors
```

```
## Loading required package:  stats4
```

```
##
## Attaching package:  'S4Vectors'
```

[1] http://genome.ucsc.edu/
[2] http://www.biomart.org/

```
## The following objects are masked from 'package:base':
##
##    colMeans, colSums, expand.grid, rowMeans, rowSums

## Loading required package:  IRanges

## Loading required package:  GenomeInfoDb

## Loading required package:  GenomicRanges

## Loading required package:  AnnotationDbi

## Loading required package:  Biobase

## Welcome to Bioconductor
##
##    Vignettes contain introductory material; view with 'browseVignettes()'.  To
##    cite Bioconductor, see 'citation("Biobase")', and for packages
##    'citation("pkgname")'.
```

# 2 TxDb Objects

The *GenomicFeatures* package uses *TxDb* objects to store transcript metadata. This class maps the 5' and 3' untranslated regions (UTRs), protein coding sequences (CDSs) and exons for a set of mRNA transcripts to their associated genome. *TxDb* objects have numerous accessors functions to allow such features to be retrieved individually or grouped together in a way that reflects the underlying biology.

All *TxDb* objects are backed by a SQLite database that manages genomic locations and the relationships between pre-processed mRNA transcripts, exons, protein coding sequences, and their related gene identifiers.

# 3 Retrieving Data from TxDb objects

## 3.1 Loading Transcript Data

There are two ways that users can load pre-existing data to generate a *TxDb* object. One method is to use the `loadDb` method to load the object directly from an appropriate .sqlite database file.

Here we are loading a previously created *TxDb* object based on UCSC known gene data. This database only contains a small subset of the possible annotations for human and is only included to demonstrate and test the functionality of the *GenomicFeatures* packageas a demonstration.

```
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package="GenomicFeatures")
txdb <- loadDb(samplefile)
txdb

## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
```

```
## # Data source: UCSC
## # Genome: hg19
## # Organism: Homo sapiens
## # UCSC Table: knownGene
## # Resource URL: http://genome.ucsc.edu/
## # Type of Gene ID: Entrez Gene ID
## # Full dataset: no
## # miRBase build ID: NA
## # transcript_nrow: 178
## # exon_nrow: 620
## # cds_nrow: 523
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2014-10-08 10:31:15 -0700 (Wed, 08 Oct 2014)
## # GenomicFeatures version at creation time: 1.17.21
## # RSQLite version at creation time: 0.11.4
## # DBSCHEMAVERSION: 1.0
```

In this case, the *TxDb* object has been returned by the `loadDb` method.

More commonly however, we expect that users will just load a TxDb annotation package like this:

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene #shorthand (for convenience)
txdb
```

```
## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Data source: UCSC
## # Genome: hg19
## # Organism: Homo sapiens
## # Taxonomy ID: 9606
## # UCSC Table: knownGene
## # Resource URL: http://genome.ucsc.edu/
## # Type of Gene ID: Entrez Gene ID
## # Full dataset: yes
## # miRBase build ID: GRCh37
## # transcript_nrow: 82960
## # exon_nrow: 289969
## # cds_nrow: 237533
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2015-10-07 18:11:28 +0000 (Wed, 07 Oct 2015)
## # GenomicFeatures version at creation time: 1.21.30
## # RSQLite version at creation time: 1.0.0
## # DBSCHEMAVERSION: 1.1
```

Loading the package like this will also create a *TxDb* object, and by default that object will have the same name as the package itself.

## 3.2 Pre-filtering data based on Chromosomes

It is possible to filter the data that is returned from a *TxDb* object based on it's chromosome. This can be a useful way to limit the things that are returned if you are only interested in studying a handful of chromosomes.

To determine which chromosomes are currently active, use the `seqlevels` method. For example:

```
head(seqlevels(txdb))
```

```
## [1] "chr1" "chr2" "chr3" "chr4" "chr5" "chr6"
```

Will tell you all the chromosomes that are active for the TxDb.Hsapiens.UCSC.hg19.knownGene *TxDb* object (by default it will be all of them).

If you then wanted to only set Chromosome 1 to be active you could do it like this:

```
seqlevels(txdb) <- "chr1"
```

So if you ran this, then from this point on in your R session only chromosome 1 would be consulted when you call the various retrieval methods... If you need to reset back to the original seqlevels (i.e. to the seqlevels stored in the db), then set the seqlevels to `seqlevels0(txdb)`.

```
seqlevels(txdb) <- seqlevels0(txdb)
```

**Exercise 1**
*Use `seqlevels` to set only chromsome 15 to be active. BTW, the rest of this vignette will assume you have succeeded at this.*

**Solution:**

```
seqlevels(txdb) <- "chr15"
```

## 3.3 Retrieving data using the select method

The *TxDb* objects inherit from *AnnotationDb* objects (just as the *ChipDb* and *OrgDb* objects do). One of the implications of this relationship is that these object ought to be used in similar ways to each other. Therefore we have written supporting `columns`, `keytypes`, `keys` and `select` methods for *TxDb* objects.

These methods can be a useful way of extracting data from a *TxDb* object. And they are used in the same way that they would be used to extract information about a *ChipDb* or an *OrgDb* object. Here is a simple example of how to find the UCSC transcript names that match with a set of gene IDs.

```
keys <- c("100033416", "100033417", "100033420")
columns(txdb)
```

```
##  [1] "CDSCHROM"   "CDSEND"     "CDSID"      "CDSNAME"    "CDSSTART"   "CDSSTRAND"
##  [7] "EXONCHROM"  "EXONEND"    "EXONID"     "EXONNAME"   "EXONRANK"   "EXONSTART"
## [13] "EXONSTRAND" "GENEID"     "TXCHROM"    "TXEND"      "TXID"       "TXNAME"
## [19] "TXSTART"    "TXSTRAND"   "TXTYPE"
```

```
keytypes(txdb)
```

```
## [1] "CDSID"    "CDSNAME"  "EXONID"   "EXONNAME" "GENEID"   "TXID"     "TXNAME"
```

```
select(txdb, keys = keys, columns="TXNAME", keytype="GENEID")
```

```
## 'select()' returned 1:1 mapping between keys and columns
```

```
##      GENEID     TXNAME
## 1 100033416 uc001yxl.4
## 2 100033417 uc001yxo.3
## 3 100033420 uc001yxr.3
```

**Exercise 2**
*For the genes in the example above, find the chromosome and strand information that will go with each of the transcript names.*

**Solution:**

```
columns(txdb)
```

```
##  [1] "CDSCHROM"   "CDSEND"     "CDSID"      "CDSNAME"    "CDSSTART"   "CDSSTRAND"
##  [7] "EXONCHROM"  "EXONEND"    "EXONID"     "EXONNAME"   "EXONRANK"   "EXONSTART"
## [13] "EXONSTRAND" "GENEID"     "TXCHROM"    "TXEND"      "TXID"       "TXNAME"
## [19] "TXSTART"    "TXSTRAND"   "TXTYPE"
```

```
cols <- c("TXNAME", "TXSTRAND", "TXCHROM")
select(txdb, keys=keys, columns=cols, keytype="GENEID")
```

```
## 'select()' returned 1:1 mapping between keys and columns
```

```
##      GENEID     TXNAME TXCHROM TXSTRAND
## 1 100033416 uc001yxl.4   chr15        +
## 2 100033417 uc001yxo.3   chr15        +
## 3 100033420 uc001yxr.3   chr15        +
```

## 3.4 Methods for returning GRanges objects

Retrieving data with select is useful, but sometimes it is more convenient to extract the result as *GRanges* objects. This is often the case when you are doing counting or specialized overlap operations downstream. For these use cases there is another family of methods available.

Perhaps the most common operations for a *TxDb* object is to retrieve the genomic coordinates or *ranges* for exons, transcripts or coding sequences. The functions `transcripts`, `exons`, and `cds` return the coordinate information as a *GRanges* object.

As an example, all transcripts present in a *TxDb* object can be obtained as follows:

```
GR <- transcripts(txdb)
GR[1:3]
```

```
## GRanges object with 3 ranges and 2 metadata columns:
##       seqnames              ranges strand |     tx_id     tx_name
##          <Rle>           <IRanges>  <Rle> | <integer> <character>
##   [1]    chr15 [20362688, 20364420]      + |     53552  uc001yte.1
```

```
##   [2]    chr15 [20487997, 20496811]      + |     53553  uc001ytf.1
##   [3]    chr15 [20723929, 20727150]      + |     53554  uc001ytj.3
##   -------
##   seqinfo: 1 sequence from hg19 genome
```

The `transcripts` function returns a *GRanges* class object. You can learn a lot more about the manipulation of these objects by reading the *GenomicRanges* introductory vignette. The `show` method for a *GRanges* object will display the ranges, seqnames (a chromosome or a contig), and strand on the left side and then present related metadata on the right side. At the bottom, the seqlengths display all the possible seqnames along with the length of each sequence.

In addition, the `transcripts` function can also be used to retrieve a subset of the transcripts available such as those on the +-strand of chromosome 1.

```
GR <- transcripts(txdb, filter=list(tx_chrom = "chr15", tx_strand = "+"))
length(GR)
```

```
## [1] 1732
```

```
unique(strand(GR))
```

```
## [1] +
## Levels: + - *
```

The `exons` and `cds` functions can also be used in a similar fashion to retrive genomic coordinates for exons and coding sequences.

**Exercise 3**
*Use `exons` to retrieve all the exons from chromosome 15. How does the length of this compare to the value returned by `transcripts`?*

**Solution:**

```
EX <- exons(txdb)
EX[1:4]
```

```
## GRanges object with 4 ranges and 1 metadata column:
##       seqnames              ranges strand |   exon_id
##          <Rle>           <IRanges>  <Rle> | <integer>
##   [1]    chr15 [20362688, 20362858]      + |    192986
##   [2]    chr15 [20362943, 20363123]      + |    192987
##   [3]    chr15 [20364397, 20364420]      + |    192988
##   [4]    chr15 [20487997, 20488227]      + |    192989
##   -------
##   seqinfo: 1 sequence from hg19 genome
```

```
length(EX)
```

```
## [1] 10771
```

```
length(GR)
```

```
## [1] 1732
```

## 3.5 Working with Grouped Features

Often one is interested in how particular genomic features relate to each other, and not just their location. For example, it might be of interest to group transcripts by gene or to group exons by transcript. Such groupings are supported by the `transcriptsBy`, `exonsBy`, and `cdsBy` functions.

The following call can be used to group transcripts by genes:

```
GRList <- transcriptsBy(txdb, by = "gene")
length(GRList)
```

```
## [1] 799
```

```
names(GRList)[10:13]
```

```
## [1] "100033424" "100033425" "100033427" "100033428"
```

```
GRList[11:12]
```

```
## GRangesList object of length 2:
## $100033425
## GRanges object with 1 range and 2 metadata columns:
##       seqnames               ranges strand |     tx_id     tx_name
##          <Rle>            <IRanges>  <Rle> | <integer> <character>
##   [1]    chr15 [25324204, 25325381]      + |     53638  uc001yxw.4
##
## $100033427
## GRanges object with 1 range and 2 metadata columns:
##       seqnames               ranges strand | tx_id    tx_name
##   [1]    chr15 [25326433, 25326526]      + | 53640 uc001yxz.3
##
## -------
## seqinfo: 1 sequence from hg19 genome
```

The `transcriptsBy` function returns a *GRangesList* class object. As with *GRanges* objects, you can learn more about these objects by reading the *GenomicRanges* introductory vignette. The `show` method for a *GRangesList* object will display as a list of *GRanges* objects. And, at the bottom the seqlengths will be displayed once for the entire list.

For each of these three functions, there is a limited set of options that can be passed into the `by` argument to allow grouping. For the `transcriptsBy` function, you can group by gene, exon or cds, whereas for the `exonsBy` and `cdsBy` functions can only be grouped by transcript (tx) or gene.

So as a further example, to extract all the exons for each transcript you can call:

```
GRList <- exonsBy(txdb, by = "tx")
length(GRList)
```

```
## [1] 3337
```

```
names(GRList)[10:13]
```

```
## [1] "53561" "53562" "53563" "53564"
```

```
GRList[[12]]
```

```
## GRanges object with 1 range and 3 metadata columns:
##       seqnames              ranges strand |   exon_id   exon_name exon_rank
##          <Rle>           <IRanges>  <Rle> | <integer> <character> <integer>
##   [1]    chr15 [22043463, 22043502]      + |    193028        <NA>         1
##   -------
##   seqinfo: 1 sequence from hg19 genome
```

As you can see, the *GRangesList* objects returned from each function contain locations and identifiers grouped into a list like object according to the type of feature specified in the `by` argument. The object returned can then be used by functions like `findOverlaps` to contextualize alignments from high-throughput sequencing.

The identifiers used to label the *GRanges* objects depend upon the data source used to create the *TxDb* object. So the list identifiers will not always be Entrez Gene IDs, as they were in the first example. Furthermore, some data sources do not provide a unique identifier for all features. In this situation, the group label will be a synthetic ID created by *GenomicFeatures* to keep the relations between features consistent in the database this was the case in the 2nd example. Even though the results will sometimes have to come back to you as synthetic IDs, you can still always retrieve the original IDs.

**Exercise 4**
*Starting with the tx_ids that are the names of the GRList object we just made, use `select` to retrieve that matching transcript names. Remember that the list used a `by` argument = "tx", so the list is grouped by transcript IDs.*

**Solution:**

```
GRList <- exonsBy(txdb, by = "tx")
tx_ids <- names(GRList)
head(select(txdb, keys=tx_ids, columns="TXNAME", keytype="TXID"))
```

```
## 'select()' returned 1:1 mapping between keys and columns
```

```
##    TXID     TXNAME
## 1 53552 uc001yte.1
## 2 53553 uc001ytf.1
## 3 53554 uc001ytj.3
## 4 53555 uc021sex.1
## 5 53556 uc010tzb.1
## 6 53557 uc021sey.1
```

Finally, the order of the results in a *GRangesList* object can vary with the way in which things were grouped. In most cases the grouped elements of the *GRangesList* object will be listed in the order that they occurred along the chromosome. However, when exons or CDS are grouped by transcript, 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.

## 3.6 Predefined grouping functions

The `intronsByTranscript`, `fiveUTRsByTranscript` and `threeUTRsByTranscript` are convenience functions that provide behavior equivalent to the grouping functions, but in prespecified form. These functions

return a *GRangesList* object grouped by transcript for introns, 5' UTR's, and 3' UTR's, respectively. Below are examples of how you can call these methods.

```
length(intronsByTranscript(txdb))
```

```
## [1] 3337
```

```
length(fiveUTRsByTranscript(txdb))
```

```
## [1] 1825
```

```
length(threeUTRsByTranscript(txdb))
```

```
## [1] 1803
```

## 3.7 Getting the actual sequence data

The *GenomicFeatures* package also provides provides functions for converting from ranges to actual sequence (when paired with an appropriate *BSgenome* package).

```
library(BSgenome.Hsapiens.UCSC.hg19)
```

```
## Loading required package:  BSgenome
```

```
## Loading required package:  Biostrings
```

```
## Loading required package:  XVector
```

```
## Loading required package:  rtracklayer
```

```
tx_seqs1 <- extractTranscriptSeqs(Hsapiens, TxDb.Hsapiens.UCSC.hg19.knownGene,
                                  use.names=TRUE)
```

And, once these sequences have been extracted, you can translate them into proteins with `translate`:

```
suppressWarnings(translate(tx_seqs1))
```

```
##   A AAStringSet instance of length 3337
##        width seq                                                  names
##    [1]   125 EDQDDEARVQYEGFRPGMYVRVEIENV...QRLLKYTPQHMHCGAAFWA*FSDSCH uc001yte.1
##    [2]   288 RIAS*GRAEFSSAQTSEIQRRRSSVLL...IFLFFESVFYSVYFNYGNNCFFTVTD uc001ytf.1
##    [3]   588 RSGQRLPEQPEAEGGDPGKQRRRAEHR...KVICERDLLENETHLYLCSIKICFSS uc001ytj.3
##    [4]    10 HHLNCRPQTG                                              uc021sex.1
##    [5]     9 STVTLPHSQ                                               uc010tzb.1
##    ...   ... ...
## [3333]    10 QVPMRVQVGQ                                              uc021syy.1
## [3334]   306 MVTEFIFLGLSDSQELQTFLFMLFFVF...TLRNKDMKTAIRRLRKWDAHSSVKF* uc002cdf.1
## [3335]   550 LAVSLFFDLFFLFMCICCLLAQTSRVL...RRQSLTPRRLHPAQLEILY*KHTVGF uc002cds.2
## [3336]   496 LAVSLFFDLFFLFMCICCLLAQTSRVL...EAVTDPETFASCTARDPLLKAHCWFL uc010utv.1
## [3337]   531 LAVSLFFDLFFLFMCICCLLAQTSRVL...RRQSLTPRRLHPAQLEILY*KHTVGF uc010utw.1
```

**Exercise 5**

*But of course this is not a meaningful translation, because the call to `extractTranscriptSeqs` will have extracted all the transcribed regions of the genome regardless of whether or not they are translated. Look*

*at the manual page for `extractTranscriptSeqs` and see how you can use cdsBy to only translate only the coding regions.*

**Solution:**

```
cds_seqs <- extractTranscriptSeqs(Hsapiens,
                                  cdsBy(txdb, by="tx", use.names=TRUE))
translate(cds_seqs)
```

```
##   A AAStringSet instance of length 1875
##        width seq                                                 names
##    [1]   102 MYVRVEIENVPCEFVQNIDPHYPIILG...EDHNGRQRLLKYTPQHMHCGAAFWA* uc001yte.1
##    [2]   435 MEWKLEQSMREQALLKAQLTQLKESLK...QEHPGLGSNCCVPFFCWAWPPRRRR* uc010tzc.1
##    [3]   317 MKIANNTVVTEFILLGLTQSQDIQLLV...QEVKTSMKRLLSRHVVCQVDFIIRN* uc001yuc.1
##    [4]   314 METANYTKVTEFVLTGLSQTPEVQLVL...YTLRNKEVKAAMRKLVTKYILCKEK* uc010tzu.2
##    [5]   317 MKIANNTVVTEFILLGLTQSQDIQLLV...QEVKTSMKRLLSRHVVCQVDFIIRN* uc010tzv.2
##     ...   ... ...
## [1871]   186 MAGGVLPLRGLRALCRVLLFLSQFCIL...RDHVHCLGRSEFKDICQQNVFLQVY* uc010ush.1
## [1872]   258 MYNSKLWEASGHWQHYSENMFTFEIEK...GGKWYPVNFLKKDLWLTLTWITVVH* uc002bxl.3
## [1873]   803 MAAEALAAEAVASRLERQEEDIRWLWS...ILVTSAIDKLKNLRKTRTLNAEEAF* uc002bxm.3
## [1874]   306 MVTEFIFLGLSDSQELQTFLFMLFFVF...TLRNKDMKTAIRRLRKWDAHSSVKF* uc002cdf.1
## [1875]   134 MSESINFSHNLGQLLSPPRCVVMPGMP...QGSCYKGETQESVESRVLPGPRHRH* uc010utv.1
```

# 4 Creating New TxDb Objects or Packages

The *GenomicFeatures* package provides functions to create *TxDb* objects based on data downloaded from UCSC Genome Bioinformatics or BioMart. The following subsections demonstrate the use of these functions. There is also support for creating *TxDb* objects from custom data sources using `makeTxDb`; see the help page for this function for details.

## 4.1 Using `makeTxDbFromUCSC`

The function `makeTxDbFromUCSC` 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 tables known to work with `makeTxDbFromUCSC`.

```
supportedUCSCtables(genome="mm9")
```

```
##         tablename          track           subtrack
## 1       knownGene     UCSC Genes               <NA>
## 2   knownGeneOld8 Old UCSC Genes               <NA>
## 3   knownGeneOld7 Old UCSC Genes               <NA>
## 4   knownGeneOld6 Old UCSC Genes               <NA>
## 5   knownGeneOld4 Old UCSC Genes               <NA>
## 6   knownGeneOld3 Old UCSC Genes               <NA>
## 7        ccdsGene           CCDS               <NA>
```

```
## 8         refGene   RefSeq Genes                <NA>
## 9      xenoRefGene   Other RefSeq                <NA>
## 10       vegaGene     Vega Genes Vega Protein Genes
## 11 vegaPseudoGene     Vega Genes   Vega Pseudogenes
## 12        ensGene  Ensembl Genes                <NA>
## 13        acembly  AceView Genes                <NA>
## 14  nscanPasaGene         N-SCAN    N-SCAN PASA-EST
## 15      nscanGene         N-SCAN              N-SCAN
## 16        sgpGene      SGP Genes                <NA>
## 17         geneid   Geneid Genes                <NA>
## 18        genscan  Genscan Genes                <NA>
## 19       exoniphy       Exoniphy                <NA>
```

```
mm9KG_txdb <- makeTxDbFromUCSC(genome="mm9", tablename="knownGene")
```

The function `makeTxDbFromUCSC` also takes an important argument called `circ_seqs` to label which chromosomes are circular. The argument is a character vector of strings that correspond to the circular chromosomes (as labeled by the source). To discover what the source calls their chromosomes, use the `getChromInfoFromUCSC` function to list them. By default, there is a supplied character vector that will attempt to label all the mitochondrial chromosomes as circular by matching to them. This is the `DEFAULT_CIRC_SEQS` vector. It contains strings that usually correspond to mitochondrial chromosomes. Once the database has been generated with the circular chromosomes tagged in this way, all subsequent analysis of these chromosomes will be able to consider their circularity for analysis. So it is important for the user to make sure that they pass in the correct strings to the `circ_seqs` argument to ensure that the correct sequences are tagged as circular by the database.

```
head(getChromInfoFromUCSC("hg19"))
```

```
## Download and preprocess the 'chrominfo' data frame ...
```

```
## OK
```

```
##   chrom    length
## 1  chr1 249250621
## 2  chr2 243199373
## 3  chr3 198022430
## 4  chr4 191154276
## 5  chr5 180915260
## 6  chr6 171115067
```

## 4.2 Using `makeTxDbFromBiomart`

Retrieve data from BioMart by specifying the mart and the data set to the `makeTxDbFromBiomart` function (not all BioMart data sets are currently supported):

```
mmusculusEnsembl <- makeTxDbFromBiomart(dataset="mmusculus_gene_ensembl")
```

As with the `makeTxDbFromUCSC` function, the `makeTxDbFromBiomart` function also has a `circ_seqs` argument that will default to using the contents of the `DEFAULT_CIRC_SEQS` vector. And just like those UCSC sources, there is also a helper function called `getChromInfoFromBiomart` that can show what the different chromosomes are called for a given source.

Using the `makeTxDbFromBiomart` `makeTxDbFromUCSC` functions can take a while and may also require some bandwidth as these methods have to download and then assemble a database from their respective sources. It is not expected that most users will want to do this step every time. Instead, we suggest that you save your annotation objects and label them with an appropriate time stamp so as to facilitate reproducible research.

### 4.3 Using `makeTxDbFromGFF`

You can also extract transcript information from either GFF3 or GTF files by using the `makeTxDbFromGFF` function. Usage is similar to `makeTxDbFromBiomart` and `makeTxDbFromUCSC`.

### 4.4 Saving and Loading a TxDb Object

Once a *TxDb* object has been created, it can be saved to avoid the time and bandwidth costs of recreating it and to make it possible to reproduce results with identical genomic feature data at a later date. Since *TxDb* objects are backed by a SQLite database, the save format is a SQLite database file (which could be accessed from programs other than *R*if desired). Note that it is not possible to serialize a *TxDb* object using *R*'s `save` function.

```
saveDb(mm9KG_txdb, file="fileName.sqlite")
```

And as was mentioned earlier, a saved *TxDb* object can be initialized from a .sqlite file by simply using `loadDb`.

```
mm9KG_txdb <- loadDb("fileName.sqlite")
```

### 4.5 Using `makeTxDbPackageFromUCSC` and `makeTxDbPackageFromBiomart`

It is often much more convenient to just make an annotation package out of your annotations. If you are finding that this is the case, then you should consider the convenience functions: `makeTxDbPackageFromUCSC` and `makeTxDbPackageFromBiomart`. These functions are similar to `makeTxDbFromUCSC` and `makeTxDbFromBiomart` except that they will take the extra step of actually wrapping the database up into an annotation package for you. This package can then be installed and used as of the standard TxDb packages found on in the Bioconductor repository.

## 5 Session Information

```
## R version 3.3.3 (2017-03-06)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.2 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C                 LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8      LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                    LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
```

```
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
##  [1] BSgenome.Hsapiens.UCSC.hg19_1.4.0       BSgenome_1.42.0
##  [3] rtracklayer_1.34.2                      Biostrings_2.42.1
##  [5] XVector_0.14.1                          TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [7] GenomicFeatures_1.26.4                  AnnotationDbi_1.36.2
##  [9] Biobase_2.34.0                          GenomicRanges_1.26.4
## [11] GenomeInfoDb_1.10.3                     IRanges_2.8.2
## [13] S4Vectors_0.12.2                        BiocGenerics_0.20.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10               knitr_1.15.1               magrittr_1.5
##  [4] GenomicAlignments_1.10.1   zlibbioc_1.20.0            BiocParallel_1.8.1
##  [7] lattice_0.20-35            highr_0.6                  stringr_1.2.0
## [10] tools_3.3.3                grid_3.3.3                 SummarizedExperiment_1.4.0
## [13] DBI_0.6                    digest_0.6.12              Matrix_1.2-8
## [16] bitops_1.0-6               RCurl_1.95-4.8             biomaRt_2.30.0
## [19] memoise_1.0.0              evaluate_0.10              RSQLite_1.1-2
## [22] stringi_1.1.3              Rsamtools_1.26.1           XML_3.98-1.5
## [25] BiocStyle_2.2.1
```