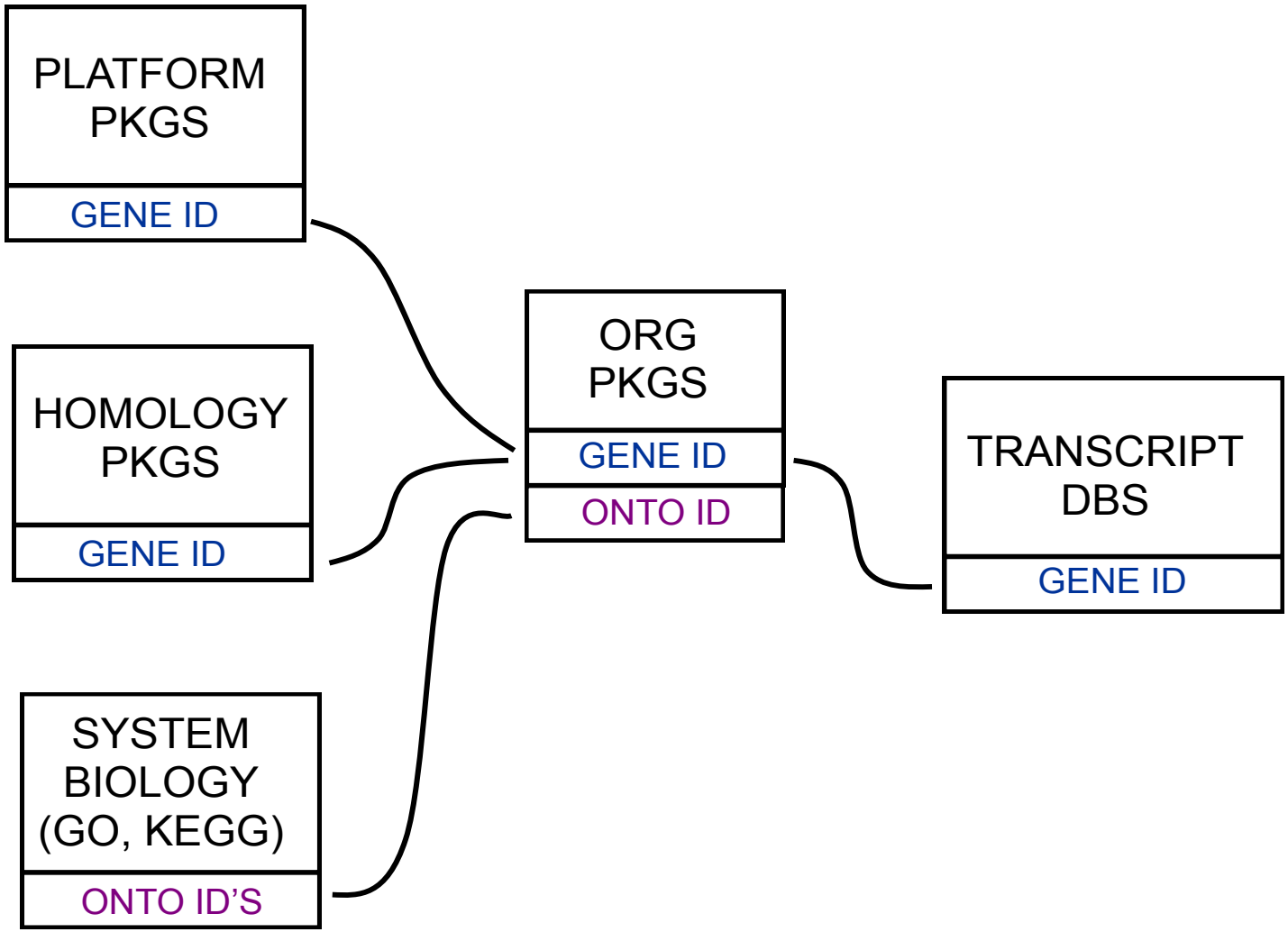
Using the *GenomicFeatures* package

Marc Carlson

Fred Hutchinson Cancer Research Center

December 10th 2010

Bioconductor Annotation Packages: a bigger picture



Bioconductor Sequence Annotations

The GenomicFeatures package

- Transcript information stored in SQLite databases
- These databases attempt to represent the relational nature of splicing data correctly
- Transcript information exposed to user via the *GenomicRanges* infrastructure
- Parsers are available to construct these databases from biomaRt or the UCSC genome browser. Also there is a generic parser for custom jobs.

GenomicFeatures transcript sources

Constructors

`makeTranscriptDbFromBiomart`, `makeTranscriptDbFromUCSC`

```
> library(GenomicFeatures)
> nrow(supportedUCSCtables())
```

```
[1] 24
```

```
> head(supportedUCSCtables(), 10)
```

```
                                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
```

TranscriptDb basics

Making a *TranscriptDb* object

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

Saving and Loading

```
> saveFeatures(mm9KG, file="mm9_knownGene.sqlite")
```

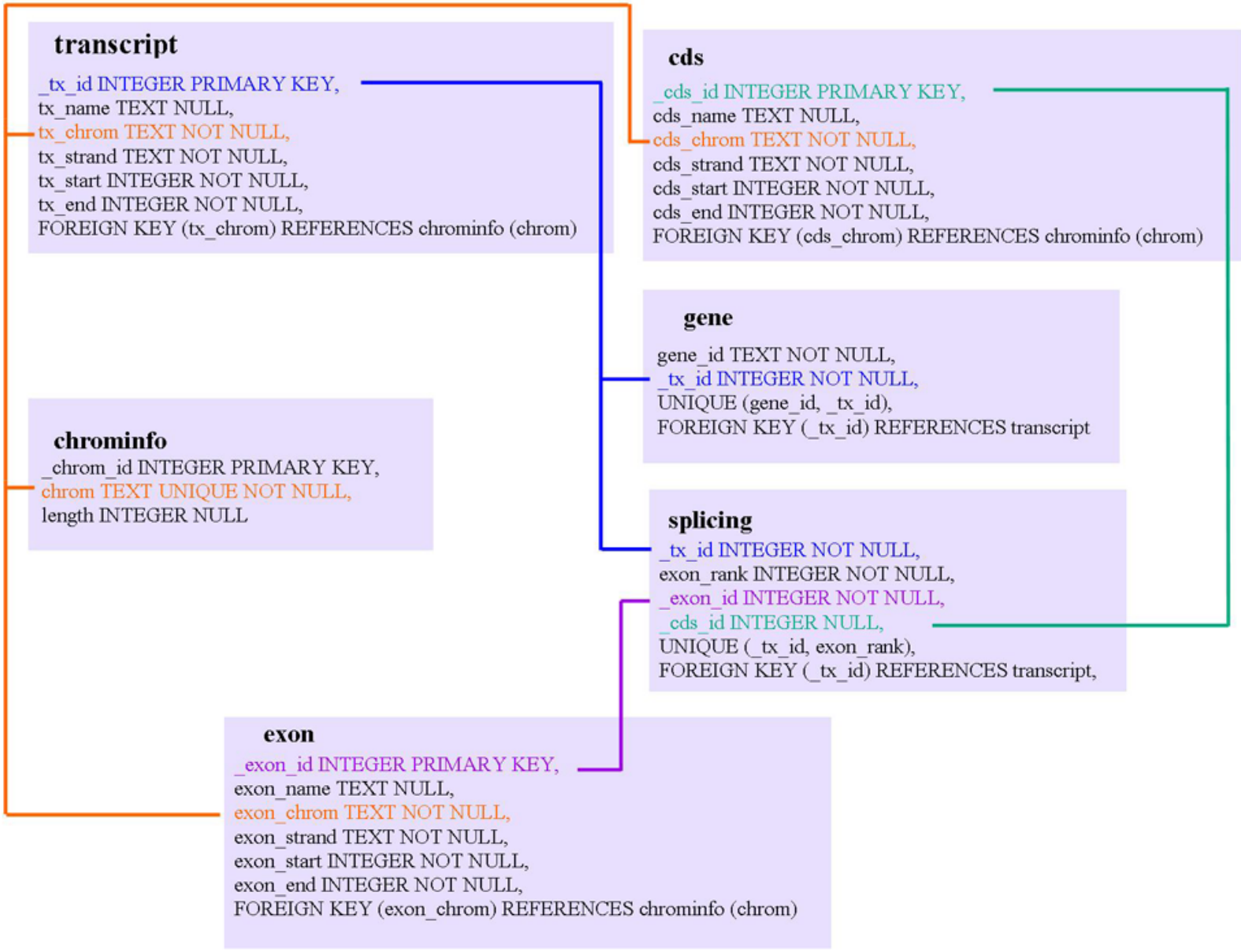
```
> txdb <-
+   loadFeatures(system.file("extdata", "mm9_knownGene.sqlite",
+                            package = "SeattleIntro2010"))
```

TranscriptDb class

```
> txdb

TranscriptDb object:
| Db type: TranscriptDb
| Data source: UCSC
| Genome: mm9
| UCSC Table: knownGene
| Type of Gene ID: Entrez Gene ID
| Full dataset: yes
| transcript_nrow: 49409
| exon_nrow: 237551
| cds_nrow: 204831
| Db created by: GenomicFeatures package from Bioconductor
| Creation time: 2010-12-07 20:14:52 -0800 (Tue, 07 Dec 2010)
| GenomicFeatures version at creation time: 1.2.3
| RSQLite version at creation time: 0.9-4
| DBSCHEMAVERSION: 1.0
```

TranscriptDb schema



Ungrouped transcript-related information

Extractors

`transcripts`, `exons`, `cds`

```
> tx <- transcripts(txdb)
> length(tx)
```

```
[1] 49409
```

```
> head(tx, 5)
```

```
GRanges with 5 ranges and 2 elementMetadata values
    seqnames              ranges strand |     tx_id     tx_name
       <Rle>           <IRanges>  <Rle> | <integer> <character>
[1]     chr1 [4797974, 4832908]      + |        14  uc007afg.1
[2]     chr1 [4797974, 4836816]      + |        15  uc007afh.1
[3]     chr1 [4847895, 4887985]      + |        16  uc007afi.1
[4]     chr1 [4848119, 4880877]      + |        17  uc007afj.1
[5]     chr1 [5073254, 5089858]      + |        20  uc007afm.1

seqlengths
      chr1       chr2 ...  chrX_random  chrY_random
 197195432  181748087 ...       1785075     58682461
```

Grouped transcript-related information

Extractors

`transcriptsBy`, `exonsBy`, `cdsBy`, `intronsByTranscript`, `fiveUTRsByTranscript`, `threeUTRsByTranscript`

```
> txExons <- exonsBy(txdb)
> txIntrons <- intronsByTranscript(txdb)
> txExons[6]
```

```
GRangesList of length 1
$6
GRanges with 2 ranges and 3 elementMetadata values
    seqnames              ranges strand |   exon_id   exon_name
       <Rle>           <IRanges>  <Rle> | <integer> <character>
[1]     chr1 [4483181, 4483816]      - |        15          NA
[2]     chr1 [4481009, 4482749]      - |        14          NA
    exon_rank
    <integer>
[1]         1
[2]         2
```

`seqlengths`

Standard GRanges accessors can be used

Accessors

`names`, `length`, `elementMetaData`, `width`, `ranges`,`start`,`end`

Examples:

```
> head(start(tx))

[1] 4797974 4797974 4847895 4848119 5073254 5073254

> head(ranges(txExons), n=1)

CompressedIRangesList of length 1
$`1`
IRanges of length 2
      start     end width
[1] 3203520 3205713  2194
[2] 3195985 3197398  1414

> head(elementMetadata(tx), n=2)

DataFrame with 2 rows and 2 columns
      tx_id     tx_name
  <integer> <character>
```

GenomicFeatures exercise 1

Load the transciptDB object above and then gather the annotations for transcripts as grouped by their genes.

GenomicFeatures exercise 1 solution

```
> library(GenomicFeatures)
> txdb <-
+   loadFeatures(system.file("extdata", "mm9_knownGene.sqlite",
+                            package = "SeattleIntro2010"))
> myTranscripts <- transcriptsBy(txdb, by="gene")
```

Overlapping with transcripts

Methods

`findOverlaps`, `countOverlaps`, `match`, `%in%`, `subsetByOverlaps`

Usage and help:

```
> findOverlaps(query, subject, maxgap = 0L, minoverlap = 1L,
+              type = c("any", "start", "end"),
+              select = c("all", "first"))
```

Example

```
> grngs <- GRanges("chr1", gaps(ranges(txIntrons[[7]])), "-")
> countOverlaps(grngs, tx)
```

```
[1] 5 4 4
```

Using Annotations and Genomic Ranges

Let's begin by learning what we need to extract data for chromosomes 3 and 5

Example

```
> library(GenomicFeatures)
> txdb <- makeTranscriptDbFromUCSC(genome="sacCer2", tablename="sgdGene
> saveFeatures(txdb, "sacCer2_sgdGene.sqlite")
> txdb <- loadFeatures(system.file("extdata", "sacCer2_sgdGene.sqlite",
+                      package = "SeattleIntro2010"))
> t <- transcripts(txdb)
> seqlengths(t)
> ## length of chr3 and 5 is 316617 and 576869
> chr3Len <- seqlengths(t)["chrIII"]
> chr5Len <- seqlengths(t)["chrV"]
> ## Define a GRanges object that describes the sequences we want to re
> gr <- GRanges(seqnames = Rle(c("chrIII","chrV")),
+               ranges = IRanges(1,width=c(chr3Len,chr5Len)),
+               seqlengths = c(chr3Len, chr5Len))
> ## And of course we also need a bam file
> fl <- system.file("extdata", "SRR002051.chrI-V.bam",
+                   package = "SeattleIntro2010")
```

Using Annotations and Genomic Ranges

- `readGappedAlignments` calls `readBamGappedAlignments`
- This is a good demo of the "..." argument
- This also shows some of the complexity of the R help system

Example

```
> ## A convenient way:
> library(ShortRead)
> grGappedBam <- readGappedAlignments(fl)
> grbam <- grg(grGappedBam)
> grbam <- grbam[seqnames(grbam) %in% c("chrIII","chrV")]
> seqlengths(grbam) <- c(chr3Len, chr5Len)
>
+ ## Look at the help here
+ # ?readGappedAlignments
+ ## and here
+ # ?readBamGappedAlignments
```

Using Annotations and Genomic Ranges

- `readGappedAlignments` calls `readBamGappedAlignments`
- This is a good demo of the "..." argument
- This also shows some of the complexity of the R help system

Example

```
> ## The most convenient way
> grGappedBam <- readGappedAlignments(fl, index=fl,
+                                     which=gr)
> grbam <- grg(grGappedBam)
> seqlengths(grbam) <- c(chr3Len, chr5Len)
> ## I saved this for you
> save(grbam, file= "bamReadsChr3andChr5.rda")
```

Practical examples: Ura3 and TIM9

Load in the Data and Annotations

Example

```
> load(system.file("data", "bamReadsChr3andChr5.rda",
+                  package = "SeattleIntro2010"))
> ## 1st get the annotations for all exons grouped by gene
> library(GenomicFeatures)
> txdb <- loadFeatures(system.file("extdata", "sacCer2_sgdGene.sqlite",
+                  package = "SeattleIntro2010"))
> txs <- transcriptsBy(txdb, by="gene")
```

Practical examples: Ura3 and TIM9

Subset the annotations down to just two genes so that we can overlap with just those ranges.

Example

```
> ## Check if the gene IDs "YEL021W" & "YEL020W-A" are present
> c("YEL021W","YEL020W-A") %in% names(txs)
> ## Lets extract only two genes
> ind = names(txs) %in% c("YEL021W","YEL020W-A")
> txpair = txs[ind]
> ## drop strand info. (so we will match both + and - strands)
> sgrbam <- grbam
> strand(sgrbam) <- rep("*",length(strand(sgrbam)))
```

Practical examples: Ura3 and TIM9

Use findOverlaps to see which things match

Example

```
> OL <- findOverlaps(sgrbam, txpair)
> head(matchMatrix(OL))
> ind <- matchMatrix(OL)[,"query"]
> ## Subset to just the data that overlapped
> subData <- grbam[ind]
> seqlengths(subData) <- chr5Len
```

Practical examples: Ura3 and TIM9

We can also do this process in reverse (starting with ranges)

Example

```
> ##starting with the data object
> subData
> ## call findOverlaps()
> OLrev <- findOverlaps(subData, txs)
> matchMatrix(OLrev)
> ## Then extract the genes that overlapped
> ind <- unique(matchMatrix(OLrev)[,"subject"])
> txs[ind]
```

GenomicFeatures exercise 2

Use the code below to load the results from the DE data from Martins talk earlier in the day. Use your gene centric annotation skills to find any gene names for the top 6 gene IDs, and then use your new skills to find the ranges that correspond to these genes.

```
> load(system.file("data", "nbTopTable.rda",
+                  package = "SeattleIntro2010"))
```

GenomicFeatures exercise 2 solution

```
> ## 1st lets see if any of these genes is even named
> ids = head(nbTopTable)[,1]
> library(org.Sc.sgd.db)
> mget(ids, org.Sc.sgdGENENAME, ifnotfound=NA)
> ## Now lets get the ranges
> ind = names(txs) %in% ids
> txSub = txs[ind]
```

Visualizing the Data

- Working with rtracklayer often involves wrapping things by using the `GRangesForUCSCGenome` wrapper
- Then you can launch a genome browser from R

Example

```
> library(rtracklayer)
> ## construct a range that describes where you want to look
> range <- GRangesForUCSCGenome(genome="sacCer2", chrom="chrV",
+                               ranges=ranges(subData),
+                               strand = strand(subData))
> # browseGenome(range = range)
> NA
```

Visualizing the Data

- You can also save files as .bed .gff or .wig files
- Those files can then be imported into the genome browser

Example

```
> ## You can export your tracks (bed, gff and wig are supported)
> export( range, "GeneRange.bed")
> ## We also saved this for you
> loadedBedFile <- system.file("extdata", "GeneRange.bed",
+                   package = "SeattleIntro2010")
> loadedBedFile
> ## Can upload this into genome browser
```

Visualizing the Data

Example

```
> ## lets look at another one:
> ind = names(txs) %in% "YEL022W"
> tx = txs[ind]
> OL <- findOverlaps(grbam, tx)
> ind <- matchMatrix(OL)[,"query"]
> subDat <- grbam[ind]
```

Visualizing the Data

`coverage` returns an rle with values computed to fit within the entire chromosome. Therefore we have to adjust for that.

Example

```
> ## a shift solution
> cov <- coverage(subData, shift=list(chrV=1-min(start(subData))))
> x <- cov[["chrV"]]
> ## or a slice solution
> x <- slice(coverage(subData), 1L)[["chrV"]][[1]]
```

Visualizing the Data

- This just defines a plotting function so we can look at the data
- Both the start and the runValues of the rle have to be plotted

Example

```
> plotCoverage <- function(x, xlab = "Position", ylab = "Coverage",
+   ...) {
+  plot(c(start(x)), c(runValue(x)), type = "s", col = "blue",
+       xlab = xlab, ylab = ylab, ...)
+ }
> ## then call out plot function
> plotCoverage(x)
```

Visualizing the Data

Of course we also could have plotted this using UCSC's genome browser

Example

```
> ## or wrap it up and then put into the genome browser
> sd <- GRangesForUCSCGenome(genome="sacCer2", chrom="chrV",
+                            ranges=ranges(subData),
+                            strand = strand(subData))
> export(sd, "subData.bed")
```

GenomicFeatures exercise 3

Use the data provided to look up data reads that map to LEU2(YCL018W) and NFS1 (YCL017C) from chromosome 3. Then either plot them or display the data in a browser. What do you notice about these two genes?

GenomicFeatures exercise 3 solution

```
> ## Get annots
> ind = names(txs) %in% c("YCL018W","YCL017C")
> txpair = txs[ind]
> ## Overlap and subset
> OL <- findOverlaps(grbam, txpair)
> ind <- matchMatrix(OL)[,"query"]
> subData <- grbam[ind]
> seqlengths(subData) <- chr3Len
> ## Save as a track:
> range <- GRangesForUCSCGenome(genome="sacCer2", chrom="chrIII"
+                               ranges=ranges(subData),
+                               strand = strand(subData))
> export( range, "GeneRange2.bed")
> ## or just plot
> x <- slice(coverage(subData), 1L)[["chrIII"]][[1]]
> plotCoverage(x)
```

Retrieving other data from rtracklayer

Next release watch for FeatureDb objects

But for now we can use rtracklayer

```
> ## create a session object
> session <- browserSession()
> ## then ucscGenomes will list the genomes
> head(ucscGenomes())
> ## Choose one, and assign it into your session object
> genome(session) <- "sacCer2"
> ## NOW you can list the trackNames
> head(trackNames(session))
```

Retrieving other data from rtracklayer

Next release watch for FeatureDb objects

But for now we can use rtracklayer

```
> ## this information is used to create a query object
> query <- ucscTableQuery(session, "oreganno")
> ## and for each track, you can list the tables it contains
> tableNames(query)
> ## This information can be used to refine/modify your query ob
> tableName(query) <- "oreganno"
> ## Which is all then used to getTable to retrieve the data
> head(getTable(query))
```