Introduction to the biomartr Package

2016-12-15

Getting Started

A major problem in bioinformatics research is consistent data retrieval. The biomartr package therefore, aims to provide users with easy to use and diverse interfaces to well curated genomic databases such as:

- NCBI
- ENSEMBL
- ENSEMBLGENOMES
- BioMart
- Gene Ontology

The collection of functions implemented in biomartr enable fast and consistent functional annotation and data retrieval queries for a set of genes, entire genomes or meta-genome projects.

Biological Sequence Retrieval

In the post-genomic era, biological sequences are used to investigate most phenomena of molecular biology. The growing number of databases and their entries allows us to design meta-studies in a new dimension and to re-investigate known phenomena from a new perspective. Nevertheless, from a data science point of view this vast amount of heterogenous data, coming from very different data resources and data standards is very hard to transform from heterogenous to homogenous data. The detection of significant patterns within meta-analyses, therefore relies on high quality data analysis and data science.

Another aspect is reproducibility. Even in cases where a high degree of data homogeneity is achieved, the aspect of scientific reproducibility adds up to a new layer of complexity. Much effort is now being invested to enable high standards of reproducibility in data driven sciences (e.g. ROpenSci). The biomartr package aims to be a part of this data science movement. It’s functions implement interfaces (Application Programming Interfaces, short APIs) to major databases such as NCBI, ENSEMBL and ENSEMBLGENOMES allowing users to access curated data from major genome data sources.

The Sequence Retrieval and Metagenome Retrieval vignettes will introduce users to the process of genomic sequence retrieval using biomartr. All functions were designed to allow users to achieve the highest (yet possible) degree of reproducibility and transparency for their own analyses.

Installation

Before users can download and install biomartr they need to install the following packages from Bioconductor:

```r
# install Bioconductor base packages
source("http://bioconductor.org/biocLite.R")
biocLite()

# load the biomaRt package
source("http://bioconductor.org/biocLite.R")
biocLite("biomaRt")

# load the Biostrings package
source("http://bioconductor.org/biocLite.R")
biocLite("Biostrings")
```
Users might be asked during the installation process of Biostrings and biomart whether or not they would like to update all package dependencies of the corresponding packages. Please type a specifying that all package dependencies of the corresponding packages shall be updated. This is important for the sufficient functionality of biomart.

Now users can download biomart from CRAN:

```r
install.packages("biomart", dependencies = TRUE)
```

**Please Note**

When using the biomart functions please be aware that an unstable internet connection can cause that some functions will not terminate properly. In that case, please re-run the corresponding function and try to use biomart with a stable internet connection. In other cases, the NCBI or ENSEMBL servers are overloaded and not very responsive causing some biomart functions to fail as well. When this happens, it is best to re-run the functions a few hours later when the query load to the NCBI or ENSEMBL servers are reduced.

**Retrieve Sequence Databases from NCBI**

NCBI stores a variety of specialized database such as Genbank, RefSeq, Taxonomy, SNP, etc. on their servers. The `download.database()` and `download.database.all()` functions implemented in biomart allows users to download these databases from NCBI.

**Search for available databases**

Before downloading specific databases from NCBI users might want to list available databases. Using the `listDatabases()` function users can retrieve a list of available databases stored on NCBI.

```r
library("biomart")

# retrieve a list of available sequence databases at NCBI
listDatabases(db = "all")
```

```
[1] "16SMicrobial"     "cdd_delta"
[3] "cloud"           "env_nr"
[5] "env_nt"          "est"
[7] "est_human"       "est_mouse"
[9] "est_others"      "FASTA"
[11] "gene_info"       "gss"
[13] "gss_annot"       "htge"
[15] "human_genomic"   "human_genomic_transcript"
[17] "landmark"        "mouse_genomic_transcript"
[19] "nr"              "nt"
[21] "other_genomic"   "pataa"
[23] "patnt"           "pdbaa"
[25] "pdbnt"           "ref_prok_rep_genomes"
[27] "ref_viroids_rep_genomes" "ref_viruses_rep_genomes"
[29] "refseq_genomic"  "refseq_protein"
[31] "refseq_rna"      "refseqgene"
[33] "sts"             "swissprot"
[35] "taxdb"           "tsa_nr"
[37] "tsa_nt"          "vector"
```
However, in case users already know which database they would like to retrieve they can filter for the exact files by specifying the NCBI database name. In the following example all sequence files that are part of the NCBI nr database shall be retrieved.

First, the `listDatabases(db = "nr")` allows to list all files corresponding to the nr database.

```
# show all NCBI nr files
listDatabases(db = "nr")
```

```
[1] "nr.00.tar.gz" "nr.01.tar.gz" "nr.02.tar.gz" "nr.03.tar.gz" "nr.04.tar.gz" "nr.05.tar.gz"
[7] "nr.16.tar.gz" "nr.06.tar.gz" "nr.15.tar.gz" "nr.30.tar.gz" "nr.07.tar.gz" "nr.08.tar.gz"
[13] "nr.09.tar.gz" "nr.10.tar.gz" "nr.11.tar.gz" "nr.12.tar.gz" "nr.13.tar.gz" "nr.14.tar.gz"
[19] "nr.28.tar.gz" "nr.29.tar.gz" "nr.31.tar.gz" "nr.17.tar.gz" "nr.18.tar.gz" "nr.19.tar.gz"
[25] "nr.20.tar.gz" "nr.21.tar.gz" "nr.22.tar.gz" "nr.23.tar.gz" "nr.32.tar.gz" "nr.24.tar.gz"
[31] "nr.25.tar.gz" "nr.26.tar.gz" "nr.27.tar.gz" "nr.33.tar.gz" "nr.34.tar.gz" "nr.35.tar.gz"
[37] "nr.36.tar.gz" "nr.37.tar.gz" "nr.38.tar.gz" "nr.39.tar.gz" "nr.40.tar.gz" "nr.41.tar.gz"
```

The output illustrates that the NCBI nr database has been separated into 41 files.

Further examples are:

```
# show all NCBI nt files
listDatabases(db = "nt")
```

```
[1] "nt.00.tar.gz" "nt.01.tar.gz" "nt.02.tar.gz" "nt.03.tar.gz" "nt.04.tar.gz" "nt.05.tar.gz"
[7] "nt.06.tar.gz" "nt.07.tar.gz" "nt.08.tar.gz" "nt.09.tar.gz" "nt.10.tar.gz" "nt.11.tar.gz"
[13] "nt.12.tar.gz" "nt.13.tar.gz" "nt.14.tar.gz" "nt.15.tar.gz" "nt.16.tar.gz" "nt.17.tar.gz"
[19] "nt.18.tar.gz" "nt.19.tar.gz" "nt.20.tar.gz" "nt.21.tar.gz" "nt.22.tar.gz" "nt.23.tar.gz"
[25] "nt.24.tar.gz" "nt.25.tar.gz" "nt.26.tar.gz" "nt.27.tar.gz" "nt.28.tar.gz" "nt.29.tar.gz"
[31] "nt.30.tar.gz" "nt.31.tar.gz" "nt.32.tar.gz" "nt.33.tar.gz"
```

```
# show all NCBI ESTs others
listDatabases(db = "est_others")
```

```
[1] "est_others.00.tar.gz" "est_others.01.tar.gz" "est_others.02.tar.gz" "est_others.03.tar.gz"
[5] "est_others.04.tar.gz" "est_others.05.tar.gz" "est_others.06.tar.gz" "est_others.07.tar.gz"
[9] "est_others.08.tar.gz" "est_others.09.tar.gz" "est_others.10.tar.gz"
```

```
# show all NCBI RefSeq (only genomes)
head(listDatabases(db = "refseq_genomic"), 20)
```

```
[1] "refseq_genomic.00.tar.gz" "refseq_genomic.01.tar.gz" "refseq_genomic.02.tar.gz"
[4] "refseq_genomic.03.tar.gz" "refseq_genomic.04.tar.gz" "refseq_genomic.05.tar.gz"
[7] "refseq_genomic.06.tar.gz" "refseq_genomic.07.tar.gz" "refseq_genomic.08.tar.gz"
[10] "refseq_genomic.09.tar.gz" "refseq_genomic.10.tar.gz" "refseq_genomic.11.tar.gz"
[13] "refseq_genomic.12.tar.gz" "refseq_genomic.13.tar.gz" "refseq_genomic.14.tar.gz"
[16] "refseq_genomic.15.tar.gz" "refseq_genomic.16.tar.gz" "refseq_genomic.17.tar.gz"
[19] "refseq_genomic.18.tar.gz" "refseq_genomic.19.tar.gz"
```

```
# show all NCBI RefSeq (only proteomes)
listDatabases(db = "refseq_protein")
```

```
[1] "refseq_protein.00.tar.gz" "refseq_protein.01.tar.gz" "refseq_protein.02.tar.gz"
[4] "refseq_protein.03.tar.gz" "refseq_protein.04.tar.gz" "refseq_protein.05.tar.gz"
[7] "refseq_protein.06.tar.gz" "refseq_protein.07.tar.gz" "refseq_protein.08.tar.gz"
[10] "refseq_protein.09.tar.gz" "refseq_protein.10.tar.gz" "refseq_protein.11.tar.gz"
[13] "refseq_protein.12.tar.gz" "refseq_protein.13.tar.gz" "refseq_protein.14.tar.gz"
[16] "refseq_protein.15.tar.gz" "refseq_protein.16.tar.gz" "refseq_protein.17.tar.gz"
[19] "refseq_protein.18.tar.gz" "refseq_protein.19.tar.gz"
```
# show all NCBI RefSeq (only RNA)
listDatabases(db = "refseq_rna")

[1] "refseq_rna.00.tar.gz" "refseq_rna.01.tar.gz" "refseq_rna.02.tar.gz" "refseq_rna.05.tar.gz"
[5] "refseq_rna.03.tar.gz" "refseq_rna.06.tar.gz" "refseq_rna.04.tar.gz" "refseq_rna.07.tar.gz"

# show NCBI swissprot
listDatabases(db = "swissprot")

[1] "swissprot.tar.gz"

# show NCBI PDB
listDatabases(db = "pdb")

[1] "pdbnt.00.tar.gz" "pdbnt.04.tar.gz" "pdbnt.05.tar.gz" "pdbnt.06.tar.gz" "pdbnt.07.tar.gz"
[6] "pdbnt.08.tar.gz" "pdbnt.09.tar.gz" "pdbnt.10.tar.gz" "pdbnt.11.tar.gz" "pdbnt.12.tar.gz"
[11] "pdbnt.13.tar.gz" "pdbnt.14.tar.gz" "pdbnt.15.tar.gz" "pdbnt.16.tar.gz" "pdbnt.17.tar.gz"
[16] "pdbnt.18.tar.gz" "pdbnt.20.tar.gz" "pdbnt.21.tar.gz" "pdbnt.22.tar.gz" "pdbnt.23.tar.gz"
[21] "pdbnt.24.tar.gz" "pdbnt.25.tar.gz" "pdbnt.26.tar.gz" "pdbnt.27.tar.gz" "pdbnt.28.tar.gz"
[26] "pdbnt.29.tar.gz" "pdbnt.30.tar.gz" "pdbnt.31.tar.gz" "pdbnt.32.tar.gz" "pdbnt.33.tar.gz"
[31] "pdbaa.tar.gz" "pdbnt.00.tar.gz" "pdbnt.01.tar.gz" "pdbnt.02.tar.gz"

# show NCBI Human database
listDatabases(db = "human")

[1] "human_genomic.00.tar.gz" "human_genomic.01.tar.gz"
[3] "human_genomic.02.tar.gz" "human_genomic.03.tar.gz"
[5] "human_genomic.04.tar.gz" "human_genomic.05.tar.gz"
[7] "human_genomic.06.tar.gz" "human_genomic.07.tar.gz"
[9] "human_genomic.08.tar.gz" "human_genomic_transcript.tar.gz"
[11] "human_genomic.10.tar.gz" "human_genomic.11.tar.gz"
[13] "human_genomic.12.tar.gz" "human_genomic.13.tar.gz"
[15] "human_genomic.14.tar.gz" "human_genomic.15.tar.gz"

# show NCBI EST databases
listDatabases(db = "est")

[1] "est.tar.gz" "est_human.00.tar.gz" "est_human.01.tar.gz" "est_mouse.tar.gz"
[5] "est_others.00.tar.gz" "est_others.01.tar.gz" "est_others.02.tar.gz" "est_others.03.tar.gz"
[9] "est_others.04.tar.gz" "est_others.05.tar.gz" "est_others.06.tar.gz" "est_others.07.tar.gz"
[13] "est_others.08.tar.gz" "est_others.09.tar.gz" "est_others.10.tar.gz"

Please note that all lookup and retrieval functions will only work properly when a sufficient internet connection is provided.

In a next step users can use the listDatabases() and download.database.all() functions to retrieve all files corresponding to a specific NCBI database.

Download available databases

Using the same search strategy by specifying the database name as described above, users can now download these databases using the download.database() function.

For downloading only single files users can type:

# select NCBI nr version 27 = "nr.27.tar.gz"
# and download it into the folder
Using this command first a folder named `nr` is created and the file `nr.27.tar.gz` is downloaded to this folder. This command will download the pre-formatted (by makeblastdb formatted) database version is retrieved.

Alternatively, users can retrieve all `nr` files with one command by typing:

```r
# download the entire NCBI nr database
download.database.all(db = "nr", path = "nr")
```

Using this command, all NCBI `nr` files are loaded into the `nr` folder (`path = "nr"`).

The same approach can be applied to all other databases mentioned above, e.g.:

```r
# download the entire NCBI nt database
download.database.all(db = "nt", path = "nt")
```

```r
# download the entire NCBI refseq (protein) database
download.database.all(db = "refseq_protein", path = "refseq_protein")
```

```r
# download the entire NCBI PDB database
download.database.all(db = "pdb", path = "pdb")
```

Please notice that most of these databases are very large, so users should take of of providing a stable internet connection throughout the download process.

**Biological Sequence Retrieval**

The `biomartr` package allows users to retrieve biological sequences in a very simple and intuitive way.

Using `biomartr`, users can retrieve either genomes, proteomes, or CDS data using the specialized functions:

- `getGenome()`
- `getProteome()`
- `getCDS()`
- `getGFF()`

**Getting Started with Sequence Retrieval**

First users can check whether or not the genome, proteome, or CDS of their interest is available for download.

Using the scientific name of the organism of interest, users can check whether the corresponding genome is available via the `is.genome.available()` function.

Please note that the first time you run this command it might take a while, because during the initial execution of this function all necessary information is retrieved from NCBI and then stored locally. All further runs are then much faster.

Example `refseq`:

```r
# checking whether or not the Homo sapiens
# genome is available for download
is.genome.available("Homo sapiens", db = "refseq")
```

```
[1] TRUE
```

Example `genbank`:
# checking whether or not the Homo sapiens genome is available for download
is.genome.available("Homo sapiens", db = "genbank")

[1] TRUE

Using is.genome.available() with ENSEMBL and ENSEMBLGENOMES

Users can also specify db = "ensembl" or db = "ensemblgenomes" to retrieve available organisms provided by ENSEMBL or ENSEMBLGENOMES. Again, users might experience a delay in the execution of this function when running it for the first time. This is due to the download of ENSEMBL or ENSEMBLGENOMES information which is then stored internally to enable a much faster execution of this function in following runs. The corresponding information files are stored at file.path(tempdir(), "ensembl_summary.txt") and file.path(tempdir(), "ensemblgenomes_summary.txt").

Example ENSEMBL:

# checking whether Homo sapiens is available in the ENSEMBL database
is.genome.available("Homo sapiens", db = "ensembl")

[1] TRUE

# retrieve details for Homo sapiens
is.genome.available("Homo sapiens", db = "ensembl", details = TRUE)

   division taxon_id name release display_name accession common_name aliases
group1 Ensembl 9606 homo_sapiens 86 Human GCA_000001405.22 human
       Homo sapiens, h_sapiens, enshs, human, hsap, 9606, homsap, hsapiens
    groups assembly
       core, cdna, vega, otherfeatures, rnaseq, variation, funcgen GRCh38

Example ENSEMBLGENOMES:

For example, some species that cannot be found at db = "ensembl" might be available at db = "ensemblgenomes" and vice versa. So I recommend users to to check in both databases ENSEMBL and ENSMBLGENOMES whether or not a particular species is present. In case of "Homo sapiens", the genome is available at db = "ensembl" but not at db = "ensemblgenomes" whereas the genome of "Arabidopsis thaliana" is available at db = "ensemblgenomes" but not at db = "ensembl".

# checking whether Homo sapiens is available in the ENSEMBLGENOMES database
is.genome.available("Homo sapiens", db = "ensemblgenomes")

Error: Unfortunately organism 'Homo sapiens' is not available at ENSEMBLGENOMES. Please check whether or not the organism name is typed correctly.

# checking whether Arabidopsis thaliana is available in the ENSEMBLGENOMES database
is.genome.available("Arabidopsis thaliana", db = "ensemblgenomes")

[1] TRUE

# show details for Arabidopsis thaliana
is.genome.available("Arabidopsis thaliana", db = "ensemblgenomes", details = TRUE)

   division taxon_id name release display_name
<chr>         <int> <chr>          <int>          <chr>
group1 EnsemblPlants 3702 arabidopsis_thaliana 85 Arabidopsis thaliana
Please note that the detailed information provided by ENSEMBL or ENSEMBL genomes differs from the information provided by NCBI.

By specifying the `details = TRUE` argument, the genome file size as well as additional information can be printed to the console.

```r
# printing details to the console
is.genome.available("Homo sapiens", details = TRUE, db = "refseq")
```

```
assembly_accession  bioproject  biosample  wgs_master
  <chr>        <chr>      <chr>        <chr>
1 GCF_000001405.35 PRJNA168      <NA>        <NA>
2 GCF_000306695.2 PRJNA178030 SAMN02205338 AMYH00000000.2
```

The argument `db` specifies from which database (`refseq`, `genbank`, `ensembl` or `ensemblgenomes`) organism information shall be retrieved.

Users can determine the total number of available genomes using the `listGenomes()` function.

Example `refseq`:

```r
length(listGenomes(db = "refseq"))
```

```
[1] 7879
```

Example `genbank`:

```r
length(listGenomes(db = "genbank"))
```

```
[1] 6723
```

Example `ensembl`:

```r
length(listGenomes(db = "ensembl"))
```

```
[1] 85
```

Example `ensemblgenomes`:

```r
length(listGenomes(db = "ensemblgenomes"))
```

```
[1] 42529
```

Hence, currently 7879 genomes (including all kingdoms of life) are stored on NCBI RefSeq.

**Retrieving kingdom, group and subgroup information**

Using this example users can retrieve the number of available species for each kingdom of life:

Example `refseq`:

```r
# the number of genomes available for each kingdom
listKingdoms(db = "refseq")
```
Archaea Bacteria Eukaryota Viroids Viruses
78 1627 425 46 5703

Example genbank:

```python
# the number of genomes available for each kingdom
listKingdoms(db = "genbank")
```

Archaea Bacteria Eukaryota
347 4876 1501

Example ENSEMBL:

```python
# the number of genomes available for each kingdom
listKingdoms(db = "ensembl")
```

Ensembl
85

Example ENSEMBLGENOMES:

```python
# the number of genomes available for each kingdom
listKingdoms(db = "ensemblgenomes")
```

EnsemblBacteria EnsemblFungi EnsemblMetazoa EnsemblPlants
416 10 634 65 44
EnsemblProtists
176

Analogous computations can be performed for groups and subgroups

Unfortunately, ENSEMBL and ENSEMBLGENOMES do not provide group or subgroup information. Therefore, group and subgroup listings are limited to refseq and genbank.

Example refseq:

```python
# the number of genomes available for each group
listGroups(db = "refseq")
```

| Acidobacteria | Animals |
|---------------|---------|
| 11            | 293     |
| Avsunviroidae | Deltaviruses |
| 4             | 1       |
| dsDNA viruses, no RNA stage | dsRNA viruses |
| 2572          | 261     |
| Elusimicrobia | Euryarchaeota |
| 1             | 64      |
| FCB group     | Fungi |
| 155           | 34      |
| Fusobacteria  | Nitrospirae |
| 6             | 3       |
| Other         | Plants |
| 10            | 62      |
| Pospiviroidae | Proteobacteria |
| 34            | 774     |
| Protists      | PVC group |
| 34            | 20      |
| Retro-transcribing viruses | Satellites |
| Subgroup                        | # Genomes Available |
|--------------------------------|---------------------|
| Spirochaetes ssDNA viruses     | 135                 |
| ssRNA viruses Synergistetes    | 12                  |
| TACK group                     | 1432                |
| Thermodesulfobacteria          | 13                  |
| Thermotogae                    | 864                 |
| unassigned viruses             | 3                   |
| unclassified Archaea           | 2                   |
| unclassified archaeal viruses  | 9                   |
| unclassified Bacteria          | 1                   |
| unclassified virophages        | 23                  |
| unclassified viroid viruses    | 3                   |
| unclassified viruses           | 23                  |
| unclassified viruses           | 6                   |
| # the number of genomes available for each subgroup |

```
head(listSubgroups(db = "refseq"), 15)
```

| Subgroup                        | # Genomes Available |
|--------------------------------|---------------------|
| Acidithiobacillia Acidobacteriia| 2                   |
| Actinobacteria Adenoviridae     | 194                 |
| Alloherpesviridae Alphaflexivirida| 7               |
| Alphaproteobacteria Alphatetravirida| 256             |
| Alvernaviridae Amalgavirida     | 1                   |
| Amphibians Ampullavirida        | 3                   |
| Anelloviridae Apicomplexans     | 53                  |
| Apple fruit crinkle viroid      | 1                   |

Example genbank:

```
# the number of genomes available for each group
listGroups(db = "genbank")
```

| Subgroup                        | # Genomes Available |
|--------------------------------|---------------------|
| Acidobacteria                   | 47                  |
| Animals                         | 655                 |
| Caldiserica Deferribacteres     | 1                   |
| DPANN group Elusimicrobia       | 27                  |
| environmental samples Euryarchaeota| 6                |
| FCB group Fungi                 | 545                 |
| Fusobacteria Nitrospinae/Tectomicrobia group| 7            |
| Nitrospirae Other               | 1                  |
# the number of genomes available for each subgroup

```r
tableSubgroups(db = "genbank", 15)
```

Note that when running the `listGenomes()` function for the first time, it might take a while until the function returns any results, because necessary information need to be downloaded from NCBI and ENSEMBL databases. All subsequent executions of `listGenomes()` will then respond very fast, because they will access the corresponding files stored on your hard drive.

## Downloading Biological Sequences and Annotations

After checking for the availability of sequence information for an organism of interest, the next step is to download the corresponding genome, proteome, CDS, or GFF file. The following functions allow users to download proteomes, genomes, CDS and GFF files from several database resources such as: NCBI RefSeq, NCBI Genbank, ENSEMBL and ENSEMBLGENOMES. When a corresponding proteome, genome, CDS or GFF file was loaded to your hard-drive, a documentation *.txt file is generated storing File Name, Organism, Database, URL, DATE, assembly_accession, bioproject, biosample, taxid, version_status, release_type, seq_rel_date etc. information of the retrieved file. This way a better reproducibility of proteome, genome, CDS and GFF versions used for subsequent data analyses can be achieved.

## Genome Retrieval

The easiest way to download a genome is to use the `getGenome()` function.
In this example we will download the genome of *Homo sapiens*.

The `getGenome()` function is an interface function to the NCBI RefSeq, NCBI Genbank, ENSEMBL and ENSEMBLGENOMES databases from which corresponding genomes can be retrieved.

The `db` argument specifies from which database genome assemblies in *.*.fasta file format shall be retrieved. Options are:

- `db = "refseq"` for retrieval from NCBI RefSeq
- `db = "genbank"` for retrieval from NCBI Genbank
- `db = "ensembl"` for retrieval from ENSEMBL
- `db = "ensemblgenomes"` for retrieval from ENSEMBLGENOMES

Furthermore, users need to specify the scientific name of the organism of interest for which a genome assembly shall be downloaded, e.g. `organism = "Homo sapiens"`. Finally, the `path` argument specifies the folder path in which the corresponding assembly shall be locally stored. In case users would like to store the genome file at a different location, they can specify the `path = file.path("put","your","path","here")` argument.

Example RefSeq:
```r
# download the genome of Homo sapiens from refseq
# and store the corresponding genome file in '~/ncbi_downloads/genomes'
HS.genome.refseq <- getGenome(db = "refseq",
    organism = "Homo sapiens",
    path = file.path("_ncbi_downloads","genomes") )
```

In this example, `getGenome()` creates a directory named '~/ncbi_downloads/genomes' into which the corresponding genome named GCF_000001405.34_GRCh38.p8_genomic.fna.gz is downloaded. The return value of `getGenome()` is the folder path to the downloaded genome file that can then be used as input to the `read_genome()` function. The variable `HS.genome.refseq` stores the path to the downloaded genome. Subsequently, users can use the `read_genome()` function to import the genome into the R session. Users can choose to work with the genome sequence in R either as Biostrings object (`obj.type = "Biostrings"`) or data.table object (`obj.type = "data.table"`) by specifying the `obj.type` argument of the `read_genome()` function.

```r
# import downloaded genome as Biostrings object
Human_Genome <- read_genome(file = HS.genome.refseq,
    obj.type = "Biostrings")
```

A DNASTringSet instance of length 551

| width seq | names                  |
|-----------|------------------------|
| [1] 248956422 | NNNNNNNNNNNNNNNNNNNNNNNNNN ... NNNNNNNNNNNNNNNNNNNNNNNN NC_000001.11 Homo ... |
| [2] 175055  | GAATTCAGTGAGAAGAACAGGCA ... TTTTTCTCAGTCATAGAATTC NT_187361.1 Homo ... |
| [3] 32032   | AGGGGTCAGGTAGGGAGGAGGTTCAGT ... GAACGTGACATGCATTAATC NT_187362.1 Homo ... |
| [4] 127682  | GATCGAGACTATCCTGGCTAACAC ... ATGTGCAATGGGACCCTTTGATC NT_187363.1 Homo ... |
| [5] 66860   | GAATTCATCGATGAGGATCCATC ... AAAAACTCTCACGAGAATTC NT_187364.1 Homo ... |
| ...       | ...                    |
| [547] 170148 | TTTCTTTCTTTTTTTTTTTTGT ... GTCAAGAGACTCATGGGGAATTC NT_187685.1 Homo ... |
| [548] 215732 | TGCTTGAGGACTCCATACAGTCT ... GTCAAGAGACTCATGGGGAATTC NT_187686.1 Homo ... |
| [549] 170537 | TCTACTCTCCCATTGCTTGCCTCGG ... GTCAAGAGACTCATGGGGAATTC NT_187687.1 Homo ... |
| [550] 177381 | GATCTATCTGTATCTCCACAGGTG ... GTCAAGAGACTCATGGGGAATTC NT_113949.2 Homo ... |
| [551] 16569  | GATCAAGTGTTATACCCCTATAT ... CCGTAAATAAGACATCAGATG NC_012920.1 Homo ... |

Internally, a text file named `doc_Homo_sapiens_db_refseq.txt` is generated. The information stored in this log file is structured as follows:
In summary, the `getGenome()` and `read_genome()` functions allow users to retrieve genome assemblies by specifying the scientific name of the organism of interest and allow them to import the retrieved genome assembly e.g. as `Biostrings` object. Thus, users can then perform the Biostrings notation to work with downloaded genomes and can rely on the log file generated by `getGenome()` to better document the source and version of genome assemblies used for subsequent studies.

Alternatively, users can perform the pipeline logic of the magrittr package:

```r
# install.packages("magrittr")
library(magrittr)

# import genome as Biostrings object
Human_Genome <- getGenome(
  db = "refseq",
  organism = "Homo sapiens",
  path = file.path("_ncbi_downloads","genomes")) %>%
  read_genome(obj.type = "Biostrings")
```

**Example Genbank:**

```r
# and store the corresponding genome file in '_ncbi_downloads/genomes'
HS.genome.genbank <- getGenome(
  db = "genbank",
  organism = "Homo sapiens",
  path = file.path("_ncbi_downloads","genomes"))
```
# import downloaded genome as Biostrings object

```r
Human_Genome <- read_genome(file = HS.genome.genbank, obj.type = "Biostrings")
```

# look at the Biostrings object

```r
Human_Genome
```

A DNAStringSet instance of length 551

| width | seq | names                          |
|-------|-----|--------------------------------|
| [1]   | 248956422 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN CM000663.2 Homo s... |
| [2]   | 175055 | GAATTCAGCTGAGAAGAACAGGCA...TGTTTGTCACTACATAGAATTC KI270706.1 Homo s... |
| [3]   | 32032  | AGGGGTCTGCTAAGAGGAGGCTCCTC...TGAATCACTGGAGTGGAAATTC KI270707.1 Homo s... |
| [4]   | 127682 | GATCGAGACTATCCTGGCTAACAACAG...ATTGTCAATGGGACCTTTGATC KI270708.1 Homo s... |
| [5]   | 66860  | GAATTCATTGCATGAGATTCGATTCC...AAAAAACTCTCACGCCACAGAATTC KI270709.1 Homo s... |

... ... ...

| [547] | 170148 | TTTCTTTCTTTTTTTTTTTTGT...GTCACAGGACTCATGGGGAATTC KI270931.1 Homo s... |
| [548] | 215732 | TGTGGTGAGGACCCTTAAGATCTA...GTCACAGGACTCATGGGGAATTC KI270932.1 Homo s... |
| [549] | 170537 | TCTACTCTCCATGCTTGCTGCG...GTCACAGGACTCATGGGGAATTC KI270933.1 Homo s... |
| [550] | 177381 | GATCTATCTGTATCTCCACAGGTG...GTCACAGGACTCATGGGGAATTC GL000209.2 Homo s... |
| [551] | 16569  | GAATCACAGGTCTATCACCCTATA...CCCTTAAATAAGACATCACGATG J01415.2 Homo sap... |

Example ENSEMBL:

# download the genome of Homo sapiens from ENSEMBL
# and store the corresponding genome file in '_ncbi_downloads/genomes'

```r
HS.genome.ensembl <- getGenome(db = "ensembl", organism = "Homo sapiens", path = file.path('_ncbi_downloads','genomes'))
```

# import downloaded genome as Biostrings object

```r
Human_Genome <- read_genome(file = HS.genome.ensembl, obj.type = "Biostrings")
```

# look at the Biostrings object

```r
Human_Genome
```

A DNAStringSet instance of length 524

| width | seq | names                          |
|-------|-----|--------------------------------|
| [1]   | 248956422 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 1 dna:chromosome ... |
| [2]   | 242193529 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 2 dna:chromosome ... |
| [3]   | 198295559 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 3 dna:chromosome ... |
| [4]   | 190214555 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 4 dna:chromosome ... |
| [5]   | 181538259 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 5 dna:chromosome ... |

... ... ...

| [520] | 993  | GCCCCACGTCCGGAGGAGGTGG...GAGGGAGGTGCGGGGCTACCGCCCT KI270539.1 dna:sc... |
| [521] | 990  | TTTCTCAGATGGGTGTTTGAACAC...TCAGAAACTCTGGTGATGTTG KI270385.1 dna:sc... |
| [522] | 981  | AGATCTCGTGGGAGCAGGATAAC...TCAGCCTTCAACACTCTCTTTTGC KI270423.1 dna:sc... |
| [523] | 971  | ATTTGTGAGGTGTGGTTCCTCAACCTA...TTGGATAGCTTTGATGTTG KI270392.1 dna:sc... |
| [524] | 970  | AAGTGGATATTTGATGCTCCCCCTG...TCCTCAATACAGGATGGACTAC KI270394.1 dna:sc... |

Example ENSEMBLGENSEMEN:

Due to the unavailability of "Homo sapiens" at db = "ensemblgenomes" here we choose "Arabidopsis thaliana" as example.

# download the genome of Arabidopsis thaliana from ENSEMBLGENSEMEN
# and store the corresponding genome file in '_ncbi_downloads/genomes'

```r
AT.genome.ensemblgenomes <- getGenome(db = "ensemblgenomes",)
```

# look at the Biostrings object

```r
AT.genome.ensemblgenomes
```

A DNAStringSet instance of length 252

| width | seq | names                          |
|-------|-----|--------------------------------|
| [1]   | 248956422 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 1 dna:chromosome ... |
| [2]   | 242193529 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 2 dna:chromosome ... |
| [3]   | 198295559 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 3 dna:chromosome ... |
| [4]   | 190214555 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 4 dna:chromosome ... |
| [5]   | 181538259 | NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 5 dna:chromosome ... |

... ... ...

| [520] | 993  | GCCCCACGTCCGGAGGAGGTGG...GAGGGAGGTGCGGGGCTACCGCCCT KI270539.1 dna:sc... |
| [521] | 990  | TTTCTCAGATGGGTGTTTGAACAC...TCAGAAACTCTGGTGATGTTG KI270385.1 dna:sc... |
| [522] | 981  | AGATCTCGTGGGAGCAGGATAAC...TCAGCCTTCAACACTCTCTTTTGC KI270423.1 dna:sc... |
| [523] | 971  | ATTTGTGAGGTGTGGTTCCTCAACCTA...TTGGATAGCTTTGATGTTG KI270392.1 dna:sc... |
| [524] | 970  | AAGTGGATATTTGATGCTCCCCCTG...TCCTCAATACAGGATGGACTAC KI270394.1 dna:sc... |
organism = "Arabidopsis thaliana",
path = file.path("_ncbi_downloads","genomes")

# import downloaded genome as Biostrings object
Cress_Genome <- read_genome(file = AT.genome.ensemblgenomes,
obj.type = "Biostrings")

# look at the Biostrings object
Cress_Genome

A DNAStringSet instance of length 7

width seq names
[1] 30427671 CCCTAAACCCTAAACCCTAAACCCT...AGGGTTTAGGGTTTAGGGTTTAGGG 1 dna:chromosome...
[2] 19698289 NNNNNNNNNNNNNNNNNNNNNNN...AGGGTTTAGGGTTTAGGGTTTAGGG 2 dna:chromosome...
[3] 23459830 NNNNNNAAAAAANNNNNNNNNNNNN...ACCCTAAACCCTAAACCCTAAACC 3 dna:chromosome...
[4] 18585056 NNNNNNNNNNNNNNNNNNNNNNNNN...AGGGTTTAGGGTTTAGGGTTTAGGG 4 dna:chromosome...
[5] 26975502 TATACCATGTACCCTCAACCTTAAA...GTTTAGGATTTAGGGTTTTTAGATC 5 dna:chromosome...
[6] 366924 GGATCCGTTCGAAACAGGTTAGCCT...TCGCAGAATGGAAACAAACCGGATT Mt dna:chromosome...
[7] 154478 ATGGGCGAACGACGGGAATTGAACC...TCATAAATCTTGTTGGCGGCGATC Pt dna:chromosome...

Proteome Retrieval

The getProteome() function is an interface function to the NCBI RefSeq, NCBI Genbank, ENSEMBL and ENSEMBLGENOMES databases from which corresponding proteomes can be retrieved. It works analogous to getGenome().

The db argument specifies from which database proteomes in *.fasta file format shall be retrieved. Options are:

- db = "refseq" for retrieval from NCBI RefSeq
- db = "genbank" for retrieval from NCBI Genbank
- db = "ensembl" for retrieval from ENSEMBL
- db = "ensemblgenomes" for retrieval from ENSEMBLGENOMES

Furthermore, again users need to specify the scientific name of the organism of interest for which a proteome shall be downloaded, e.g. organism = "Homo sapiens". Finally, the path argument specifies the folder path in which the corresponding proteome shall be locally stored. In case users would like to store the proteome file at a different location, they can specify the path = file.path("put","your","path","here") argument.

Example RefSeq:

# download the proteome of Homo sapiens from refseq
# and store the corresponding proteome file in '~/ncbi_downloads/proteomes'

HS.proteome.refseq <- getProteome(db = "refseq",
organism = "Homo sapiens",
path = file.path("_ncbi_downloads","proteomes"))

In this example, getProteome() creates a directory named ' _ncbi_downloads/proteomes' into which the corresponding genome named GCF_000001405.34_GRCh38.p8_protein.faa.gz is downloaded. The return value of getProteome() is the folder path to the downloaded proteome file that can then be used as input to the read_proteome() function. The variable HS.proteome.refseq stores the path to the downloaded proteome. Subsequently, users can use the read_proteome() function to import the proteome into the R session. Users can choose to work with the proteome sequence in R either as Biostrings object (obj.type = "Biostrings") or data.table object (obj.type = "data.table") by specifying the obj.type argument of the read_proteome() function.
# import proteome as Biostrings object

```r
Human_Proteome <- read_proteome(file = HS.proteome.refseq,
                   obj.type = "Biostrings")
```

Human_Proteome

A AAStringSet instance of length 109018

| width | seq names |
|-------|-----------|
| [1]   | 1474      |
|       | MGKNKLHPSLVLVLLVLPTDAS...DYETDEFAIAEYNAPCSKDLGNA NP_000005.2 alpha... |
| [2]   | 290       |
|       | MDIEAYFERIGYNKSNKLDELT...LNIFKISLRGNLVPKGDSLTI NP_000006.2 aryla... |
| [3]   | 421       |
|       | MAAGFGCCVLRISRFHRWSQHI...QIYEGTSIQRLVAREHIDKYN NP_000007.1 mediu... |
| [4]   | 412       |
|       | MAAALLARASGPARALLCPRAWRQ...EYMGETSEIQLVIACHLLRSYRS NP_000008.1 short... |
| [5]   | 655       |
|       | MQAAARMAASLRQLRLRGSSRL...RFKISKALVERGGVTSNPLGF NP_000009.1 very... |
| [109014]| 98       |
|       | MPLIYMNILAFILSGLMLVYRS...LALLVSINNTYGLDYVHNLLLQC YP_003024034.1 NA... |
| [109015]| 459      |
|       | MLKLIVPTIMLLPLTLWLSDKHMI...LMWHSLSILLSLNPDITGFSS YP_003024035.1 NA... |
| [109016]| 603      |
|       | MTMHTTMTLTTSILPPFLTTL...QKQMIKLYFLSFFFPILLLLIT YP_003024036.1 NA... |
| [109017]| 174      |
|       | HHYFLLSVLWLGVFGFSSSAPS...WLVVTTGWLTVGYYIVIEARGN YP_003024037.1 NA... |
| [109018]| 380      |
|       | MTPMRKTNPLMLKHICFIDLPTP...LYFTTILILMTPSLIENKMLKWA YP_003024038.1 cy... |

Alternatively, users can perform the pipeline logic of the magrittr package:

```r
# install.packages("magrittr")
library(magrittr)

# import proteome as Biostrings object

Human_Proteome <- getProteome(db = "refseq",
                              organism = "Homo sapiens",
                              path = file.path("_ncbi_downloads","proteomes")) %>%
read_proteome(obj.type = "Biostrings")
```

Human_Proteome

A AAStringSet instance of length 109018

| width | seq names |
|-------|-----------|
| [1]   | 1474      |
|       | MGKNKLHPSLVLVLLVLPTDAS...DYETDEFAIAEYNAPCSKDLGNA NP_000005.2 alpha... |
| [2]   | 290       |
|       | MDIEAYFERIGYNKSNKLDELT...LNIFKISLRGNLVPKGDSLTI NP_000006.2 aryla... |
| [3]   | 421       |
|       | MAAGFGCCVLRISRFHRWSQHI...QIYEGTSIQRLVAREHIDKYN NP_000007.1 mediu... |
| [4]   | 412       |
|       | MAAALLARASGPARALLCPRAWRQ...EYMGETSEIQLVIACHLLRSYRS NP_000008.1 short... |
| [5]   | 655       |
|       | MQAAARMAASLRQLRLRGSSRL...RFKISKALVERGGVTSNPLGF NP_000009.1 very... |
| [109014]| 98       |
|       | MPLIYMNILAFILSGLMLVYRS...LALLVSINNTYGLDYVHNLLLQC YP_003024034.1 NA... |
| [109015]| 459      |
|       | MLKLIVPTIMLLPLTLWLSDKHMI...LMWHSLSILLSLNPDITGFSS YP_003024035.1 NA... |
| [109016]| 603      |
|       | MTMHTTMTLTTSILPPFLTTL...QKQMIKLYFLSFFFPILLLLIT YP_003024036.1 NA... |
| [109017]| 174      |
|       | HHYFLLSVLWLGVFGFSSSAPS...WLVVTTGWLTVGYYIVIEARGN YP_003024037.1 NA... |
| [109018]| 380      |
|       | MTPMRKTNPLMLKHICFIDLPTP...LYFTTILILMTPSLIENKMLKWA YP_003024038.1 cy... |

Example Genbank:

```r
# download the proteome of Homo sapiens from genbank
# and store the corresponding proteome file in '_ncbi_downloads/proteomes'

HS.proteome.genbank <- getProteome(db = "genbank",
                                 organism = "Homo sapiens",
                                 path = file.path("_ncbi_downloads","proteomes"))

# install.packages("magrittr")
library(magrittr)

# import proteome as Biostrings object

Human_Proteome <- read_proteome(file = HS.proteome.genbank,
                                obj.type = "Biostrings")
Example ENSEMBL:

# download the proteome of Homo sapiens from ENSEMBL
# and store the corresponding proteome file in
# \_ncbi\_downloads/proteomes

HS.proteome.ensembl <- getProteome( db = "ensembl",
                                   organism = "Homo sapiens",
                                   path = file.path("\_ncbi\_downloads","proteomes"))

# import proteome as Biostrings object

Human_Proteome <- read_proteome(file = HS.proteome.ensembl,
                                 obj.type = "Biostrings")

Example ENSEMBLGENOMES:

Due to the unavailability of "Homo sapiens" at db = "ensemblgenomes" here we choose "Arabidopsis thaliana" as example.

# download the proteome of Arabidopsis thaliana from ENSEMBLGENOMES
# and store the corresponding proteome file in \_ncbi\_downloads/proteomes

AT.proteome.ensemblgenomes <- getProteome( db = "ensemblgenomes",
                                          organism = "Arabidopsis thaliana",
                                          path = file.path("\_ncbi\_downloads","proteomes"))

# import proteome as Biostrings object

Cress_Proteome <- read_proteome(file = AT.proteome.ensemblgenomes,
                                 obj.type = "Biostrings")
Cress_Proteome

A AAStringSet instance of length 35386

| width | seq names |
|-------|-----------|
| [1]   | 415 MGRDETETYITVPSFFKCPISLDVM...IKVLKFNSALAAAYETKTHIMPF AT3G18710.1 pep:k... |
| [2]   | 855 MATENPIRISGNERWSNRKVSVVP...YTGKHIVSRLEQPSIEENQELR AT4G25880.2 pep:k... |
| [3]   | 858 MATENPIRISGNERWSNRKVSVVP...KHHISVLEQPSIEEMKFPKTKN AT4G25880.3 pep:k... |
| [4]   | 861 MATENPIRISGNERWSNRKVSVVP...YTGKHIVSRLEQPSIEENQELR AT4G25880.1 pep:k... |
| [5]   | 358 MTKAYSTRVLTFLILISLMAVTLNL...CSARNTQSFMSVLEEGIEAISMI AT1G71695.1 pep:k... |
| ...   | ... ...   |

[35382] 374 MHSRSALLYRFLPASRCFSSSAV...YKAGEYYYJSMEADRVSAPSTSP AT2G20860.1 pep:k... 
[35383] 392 MADNLNLVSVLGVVLTIFHNPII...WEPNNLAIRRPPSRDFYGLAAYY AT3G14210.1 pep:k... 
[35384] 495 MASLLSPATPTATSAAHSCSTAGF...LHPYFLLGQAAVLSKLSFSK AT5G01920.1 pep:k... 
[35385] 563 MSKKEAEKTVGSTIKPTLNP...YAVQRYLLEEELGLYSEPQAGLRA AT2G26280.1 pep:k... 
[35386] 453 MGVSLLKQQHRQADTFSRFMER...NHQQQQLQRSRELVRGLESANVI AT4G32600.1 pep:k... 

CDS Retrieval

The `getCDS()` function is an interface function to the NCBI RefSeq, NCBI Genbank, ENSEMBL and ENSEMBLGENOMES databases from which corresponding CDS files can be retrieved. It works analogous to `getGenome()` and `getProteome()`.

The `db` argument specifies from which database proteomes in `*.fasta` file format shall be retrieved. Options are:

- `db = "refseq"` for retrieval from NCBI RefSeq
- `db = "genbank"` for retrieval from NCBI Genbank
- `db = "ensembl"` for retrieval from ENSEMBL
- `db = "ensemblgenomes"` for retrieval from ENSEMBLGENOMES

Furthermore, again users need to specify the scientific name of the organism of interest for which a proteomes shall be downloaded, e.g. `organism = "Homo sapiens"`. Finally, the `path` argument specifies the folder path in which the corresponding CDS file shall be locally stored. In case users would like to store the CDS file at a different location, they can specify the `path = file.path("put","your","path","here")` argument.

Example RefSeq:

```r
# download the genome of Homo sapiens from refseq
# and store the corresponding genome CDS file in '_ncbi_downloads/CDs'
HS.cds.refseq <- getCDS(db = "refseq", 
                        organism = "Homo sapiens", 
                        path = file.path("_ncbi_downloads","CDs"))
```

In this example, `getCDS()` creates a directory named `_ncbi_downloads/CDs` into which the corresponding genome named `Homo_sapiens cds_from_genomic_refseq.fna.gz` is downloaded. The return value of `getCDS()` is the folder path to the downloaded genome file that can then be used as input to the `read_cds()` function. The variable `HS.cds.refseq` stores the path to the downloaded CDS file. Subsequently, users can use the `read_cds()` function to import the genome into the R session. Users can choose to work with the genome sequence in R either as Biostrings object (`obj.type = "Biostrings"`) or data.table object (`obj.type = "data.table"`) by specifying the `obj.type` argument of the `read_cds()` function.

```r
# import downloaded CDS as Biostrings object
Human_CDS <- read_cds(file = HS.cds.refseq, 
                      obj.type = "Biostrings")
```
# look at the Biostrings object

Human_CDS

A BStringSet instance of length 114967

| width | seq                              | names                                        |
|-------|----------------------------------|----------------------------------------------|
| [1]   | 918 ATGGTGACTGAATTCTATTTTCTG...CACATTCTAGTGAAATGTTTAG | lcl|NC_000001.11_... |
| [2]   | 402 ATGAGTGACGATCAAAGTTCTTCT...CAGGACACGGCCAGGCATTAG | lcl|NC_000001.11_... |
| [3]   | 402 ATGAGTGACGATCAAAGTTCTTCT...CAGGACACGGCCAGGCATTAG | lcl|NC_000001.11_... |
| [4]   | 402 ATGAGTGACGATCAAAGTTCTTCT...CAGGACACGGCCAGGCATTAG | lcl|NC_000001.11_... |
| [5]   | 402 ATGAGTGACGATCAAAGTTCTTCT...CAGGACACGGCCAGGCATTAG | lcl|NC_000001.11_... |
|       | ...                               | ...                                          |
| [114963] | 297 ATGCCCCTCATTTACATAAATATT...ACCTAAACCTACTCCAAATGCTAA | lcl|NC_012920.1_c... |
| [114964] | 1378 ATGCTAAAAACTAATCGTCCCAACA...CATCATTACCGGGTTTCTCTT | lcl|NC_012920.1_c... |
| [114965] | 1812 ATAACCACATGCACTACTATAAACC...TAACCTCTCCTAATACACTAA | lcl|NC_012920.1_c... |
| [114966] | 525 ATGAGTATATGTTTGGTTTCTGTGTTG...TTGGATTGGCTGGGGGAATAGG | lcl|NC_012920.1_c... |
| [114967] | 1141 ATGACCCCAATACGCAAAACTAAC...AAACAAAATACTCAAATGGGCCT | lcl|NC_012920.1_c... |

Internally, a text file named doc_Homo_sapiens_db_refseq.txt is generated. The information stored in this log file is structured as follows:

File Name: Homo_sapiens cds_from_genomic_refseq.fna.gz
Organism Name: Homo_sapiens
Database: NCBI refseq
URL: ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/405/GCF_000001405.35_GRCh38.p9/GCF_000001405.35_GRCh38.p9 cds_from_genomic.fna.gz
Download Date: Sun Oct 23 17:19:05 2016
refseq_category: reference genome
assembly_accession: GCF_000001405.35
bioproject: PRJNA168
biosample: NA
taxid: 9606
infraspecific_name: NA
version_status: latest
release_type: Patch
genome_rep: Full
seq_rel_date: 2016-09-26
submitter: Genome Reference Consortium

In summary, the `getCDS()` and `read_cds()` functions allow users to retrieve CDS files by specifying the scientific name of the organism of interest and allow them to import the retrieved CDS file e.g. as Biostrings object. Thus, users can then perform the Biostrings notation to work with downloaded CDS and can rely on the log file generated by `getCDS()` to better document the source and version of CDS used for subsequent studies.

Alternatively, users can perform the pipeline logic of the magrittr package:

```r
# install.packages("magrittr")
library(magrittr)
# import CDS as Biostrings object
Human_CDS <- getCDS( db = "refseq",
                   organism = "Homo sapiens",
                   path = file.path("_ncbi_downloads","CDS")) %>%
   read_cds(obj.type = "Biostrings")
```

Human_CDS

A BStringSet instance of length 114967
Example Genbank:

```r
# download the genome of Homo sapiens from genbank
# and store the corresponding genome CDS file in '_ncbi_downloads/CDS'
HS.cds.genbank <- getCDs(
db = "genbank",
organism = "Homo sapiens",
path = file.path('_ncbi_downloads','CDS'))

# import downloaded CDS as Biostrings object
Human_CDS <- read_cds(file = HS.cds.genbank,
obj.type = "Biostrings")

# look at the Biostrings object
Human_CDS
```

Example ENSEMBL:

```r
# download the genome of Homo sapiens from ensembl
# and store the corresponding genome CDS file in '_ncbi_downloads/CDS'
HS.cds.ensembl <- getCDs(
db = "ensembl",
organism = "Homo sapiens",
path = file.path('_ncbi_downloads','CDS'))

# import downloaded CDS as Biostrings object
Human_CDS <- read_cds(file = HS.cds.ensembl,
obj.type = "Biostrings")

# look at the Biostrings object
Human_CDS
```

A BStringSet instance of length 102915

```r
A BStringSet instance of length 13
```
## width seq names

|   |   |   |
|---|---|---|
| 1 | 13 ACTGGGGGATACG | ENST00000448914.1 |
| 2 | 12 GGGACAGGGGCC | ENST00000631435.1 |
| 3 | 12 GGGACAGGGGCC | ENST00000632684.1 |
| 4 | 9 CTTTCTCTAC | ENST00000434970.2 |
| 5 | 8 GAAATAGT | ENST00000415118.1 |

... ... ...

| 102911 | 1665 ATGCTACTGCCACTGCTGCTGTCC...ATGCAGAAGTCAAGTTCCAATGA | ENST00000436984.6 |
| 102912 | 1920 ATGCTACTGCCACTGCTGCTGTCC...ATGCAGAAGTCAAGTTCCAATGA | ENST00000439889.6 |
| 102913 | 2094 ATGCTACTGCCACTGCTGCTGTCC...ATGCAGAAGTCAAGTTCCAATGA | ENST00000033933.7 |
| 102914 | 466 ATGCTACTGCCACTGCTGCTGTCC...AGCCCTGGACCTCTCTGTGCAGT | ENST00000529627.1 |
| 102915 | 599 ATGGCGGAGATGCTACTGCCACTGCTGCTGTCC...CTCACCTGCGGACTGACTTC | ENST00000530476.1 |

Example ENSEMBLGENOMES:

Due to the inavailability of "Homo sapiens" at db = "ensemblgenomes" here we choose "Arabidopsis thaliana" as example.

```r
# download the genome of Homo sapiens from ensemblgenomes
# and store the corresponding genome CDS file in /quotesingle.ts1
_ncbi_downloads/CDS/quotesingle.ts1

AT.cds.ensemblgenomes <- getCDS( db = "ensemblgenomes",
                                 organism = "Arabidopsis thaliana",
                                 path = file.path("_ncbi_downloads","CDS"))

# import downloaded CDS as Biostrings object
Cress_CDS <- read_cds(file = AT.cds.ensemblgenomes,
                       obj.type = "Biostrings")

# look at the Biostrings object
Cress_CDS

A BStringSet instance of length 35386
width seq names
|   |   |   |
|---|---|---|
| 1 | 1248 ATGGGGAGAGATGAAACAGAGACGT...ACTACTCATATTATGCCTTTTTGA | AT3G18710.1 cds:k...
| 2 | 2568 ATGGCAACTGAGAATCCTATTAGGA...GAAAACCAAGAATTGAGGAGATGA | AT4G25880.2 cds:k...
| 3 | 2577 ATGGCAACTGAGAATCCTATTAGGA...TTCCCCAATAAAACCAAGAATTGAGGAGATGA | AT4G25880.3 cds:k...
| 4 | 2586 ATGGCAACTGAGAATCCTATTAGGA...GAAAACCAAGAATTGAGGAGATGA | AT4G25880.1 cds:k...
| 5 | 1077 ATGACAAAGGCTTATTCAACACCTG...GAAGGAAGCTATTTCCATGACTAA | AT1G71695.1 cds:k...

... ... ...

| 35382 | 1125 ATGCATTCGCGCTCCTGCTGCTT...GCTTCTTCTTCTACATCCCCGTAG | AT2G20860.1 cds:k...
| 35383 | 1179 ATGGCGAAGAAAAGGTGCGCTCGCTTCT...TTGGCGCTCGCGGCTCATATTAGGA | AT3G14210.1 cds:k...
| 35384 | 1488 ATGGCGGCTCTCTCTTCTCTCTCGGCGCA...TCAAAGCTCAGTTTCTACACAGAAGTAG | AT5G01920.1 cds:k...
| 35385 | 1689 ATGAGTTAAACAAAGAAAGAAGTAGTG...GAACAcAGGCGCCGCTCCTATTAGGA | AT2G26280.1 cds:k...
| 35386 | 1362 ATGGGTGTTTCTCTCTCGAACAAC...GGTTGGCGCAATATGTTATCTTAG | AT4G32600.1 cds:k...

Retrieve the annotation file of a particular genome

Finally, users can download the corresponding annotation .gff files for particular genomes of interest using the getGFF() function.

Example RefSeq:

```r
# download the GFF file of Homo sapiens from refseq
# and store the corresponding file in '_ncbi_downloads/annotation'

HS.gff.refseq <- getGFF( db = "refseq",
                         organism = "Homo sapiens",
                         path = file.path("_ncbi_downloads","annotation"))
```
After downloading the .gff file, users can import the .gff file with `read_gff()`.

```r
# import downloaded GFF file
Human_GFF <- read_gff(file = HS.gff.refseq)
```

```
| seqid       | source | type   | start  | end     | score | strand | phase |
|-------------|--------|--------|--------|---------|-------|--------|-------|
| NC_000001.11 | RefSeq | region | 1      | 248956422 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | gene | 11874 | 14409 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | transcript | 11874 | 14409 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | exon | 11874 | 12227 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | exon | 12613 | 12721 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | exon | 13221 | 14409 | 0     | +      | 0     |
| NC_000001.11 | BestRefSeq | gene | 14362 | 29370 | 0     | -      | 0     |
| NC_000001.11 | BestRefSeq | transcript | 14362 | 29370 | 0     | -      | 0     |
| NC_000001.11 | BestRefSeq | exon | 29321 | 29370 | 0     | -      | 0     |
| NC_000001.11 | BestRefSeq | exon | 24738 | 24891 | 0     | -      | 0     |
```

### Example Genbank:

```r
# download the GFF file of Homo sapiens from genbank
# and store the corresponding file in '~/ncbi_downloads/annotation'
HS.gff.genbank <- getGFF(db = "genbank",
                         organism = "Homo sapiens",
                         path = file.path("~/ncbi_downloads/annotation"))

# import downloaded GFF file
Human_GFF <- read_gff(file = HS.gff.genbank)
```

```
| seqid       | source | type       | start  | end     | score | strand | phase |
|-------------|--------|------------|--------|---------|-------|--------|-------|
| CM000663.2  | Genbank | region     | 1      | 248956422 | 0     | +      | 0     |
| CM000663.2  | Genbank | centromere | 122026460 | 125184587 | 0     | +      | 0     |
| KI270706.1  | Genbank | region     | 1      | 175055  | 0     | +      | 0     |
| KI270707.1  | Genbank | region     | 1      | 32032   | 0     | +      | 0     |
| KI270708.1  | Genbank | region     | 1      | 127682  | 0     | +      | 0     |
| KI270709.1  | Genbank | region     | 1      | 66860   | 0     | +      | 0     |
| KI270710.1  | Genbank | region     | 1      | 40176   | 0     | +      | 0     |
| KI270711.1  | Genbank | region     | 1      | 42210   | 0     | +      | 0     |
| KI270712.1  | Genbank | region     | 1      | 176043  | 0     | +      | 0     |
| KI270713.1  | Genbank | region     | 1      | 40745   | 0     | +      | 0     |
```

### Example ENSEMBL:

```r
# download the GFF file of Homo sapiens from ENSEMBL
# and store the corresponding file in '~/ncbi_downloads/annotation'
HS.gff.ensembl <- getGFF(db = "ensembl",
                         organism = "Homo sapiens",
                         path = file.path("~/ncbi_downloads/annotation"))

# import downloaded GFF file
Human_GFF <- read_gff(file = HS.gff.ensembl)
```

```
| seqid       | source | type        | start  | end     | score | strand | phase |
|-------------|--------|-------------|--------|---------|-------|--------|-------|
| CM000663.2  | Genbank | region      | 1      | 248956422 | 0     | +      | 0     |
| CM000663.2  | Genbank | centromere  | 122026460 | 125184587 | 0     | +      | 0     |
| KI270706.1  | Genbank | region      | 1      | 175055  | 0     | +      | 0     |
| KI270707.1  | Genbank | region      | 1      | 32032   | 0     | +      | 0     |
| KI270708.1  | Genbank | region      | 1      | 127682  | 0     | +      | 0     |
| KI270709.1  | Genbank | region      | 1      | 66860   | 0     | +      | 0     |
| KI270710.1  | Genbank | region      | 1      | 40176   | 0     | +      | 0     |
| KI270711.1  | Genbank | region      | 1      | 42210   | 0     | +      | 0     |
| KI270712.1  | Genbank | region      | 1      | 176043  | 0     | +      | 0     |
| KI270713.1  | Genbank | region      | 1      | 40745   | 0     | +      | 0     |
```

After downloading the .gff file, users can import the .gff file with `read_gff()`.
# import downloaded GFF file

Human_GFF <- `read_gff(file = HS.gff.ensembl)`

Cress_GFF <- `read_gff(file = AT.gff.ensemblgenomes)`

Example ENSEMBLGENOMES:

Due to the inavailability of "Homo sapiens" at db = "ensemblgenomes" here we choose "Arabidopsis thaliana" as example.

# download the GFF file of Arabidopsis thaliana from ENSEMBLGENOMES
# and store the corresponding file in '/quotesingle.ts1

AT.gff.ensemblgenomes <- getGFF(db = "ensemblgenomes",
organism = "Arabidopsis thaliana",
path = file.path("_ncbi_downloads","annotation"))

After downloading the .gff file, users can import the .gff file with read_gff().

# import downloaded GFF file

Cress_GFF <- `read_gff(file = AT.gff.ensemblgenomes)`

Perform Meta-Genome Retrieval

The number of genome sequences generated and stored in sequence databases is growing exponentially every day. With the availability of this growing amount of data, meta-genomics studies become more popular and
useful for finding patterns within genomes by comparing them to thousands of other genomes. However, the first step in any meta-genomics study is the retrieval of the genomes that shall be compared or investigated.

For this purpose, I implemented the `meta.retrieval()` function to allow users to perform easy meta-genome retrieval in R.

The `getKingdoms()` function stores a list of all available kingdoms of life. Using the argument `db` users can specify from which database kingdom information shall be retrieved.

Example RefSeq:

```r
getKingdoms(db = "refseq")
```

```
[1] "archaea"  "bacteria"  "fungi"    "invertebrate"
[5] "plant"   "protozoa"  "vertebrate_mammalian" "vertebrate_other"
[9] "viral"
```

Example Genbank:

```r
getKingdoms(db = "genbank")
```

```
[1] "archaea"  "bacteria"  "fungi"
[4] "invertebrate" "plant"   "protozoa"
[7] "vertebrate_mammalian" "vertebrate_other"
```

In these examples the difference between `db = "refseq"` and `db = "genbank"` is that `db = "genbank"` does not store viral information.

These kingdoms can be specified in `meta.retrieval()`.

The `meta.retrieval()` function aims to simplify the genome retrieval process for subsequent meta-genomics studies.

Usually this step is performed with `shell` scripts. However, since many meta-genomics packages exist for the R programming language, I implemented this functionality for easy integration into existing workflows.

For example, the pipeline logic of the magrittr package can be used with `meta.retrieval()`.

```r
# download all vertebrate genomes, then apply ...
meta.retrieval(kingdom = "vertebrate_mammalian", db = "refseq", type = "genome") %>% ...
```

Here ... denotes any subsequent meta-genomics analysis. Hence, `meta.retrieval()` enables the pipelining methodology for meta-genomics.

The `meta.retrieval()` function can retrieve genomes, proteomes, and CDS files.

### Retrieve Genomic Sequences

Download all mammalian vertebrate genomes from RefSeq.

```r
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "refseq", type = "genome")
```

All genomes are stored in the folder named according to the kingdom. In this case `vertebrate_mammalian`. Alternatively, users can specify the `out.folder` argument to define a custom output folder path.

Example Bacteria

```r
# download all bacteria genomes
meta.retrieval(kingdom = "bacteria", db = "refseq", type = "genome")
```

Example Viruses
# download all virus genomes
meta.retrieval(kingdom = "viral", db = "refseq", type = "genome")

Example Archaea
# download all archaea genomes
meta.retrieval(kingdom = "archaea", db = "refseq", type = "genome")

Example Fungi
# download all fungi genomes
meta.retrieval(kingdom = "fungi", db = "refseq", type = "genome")

Example Plants
# download all plant genomes
meta.retrieval(kingdom = "plant", db = "refseq", type = "genome")

Example Invertebrates
# download all invertebrate genomes
meta.retrieval(kingdom = "invertebrate", db = "refseq", type = "genome")

Example Protozoa
# download all invertebrate genomes
meta.retrieval(kingdom = "protozoa", db = "refseq", type = "genome")

Alternatively, download all mammalian vertebrate genomes from Genbank, e.g.
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "genbank", type = "genome")

Metagenome project retrieval from NCBI Genbank

NCBI Genbank stores metagenome projects in addition to species specific genome, proteome or CDS sequences. To retrieve these metagenomes users can perform the following combination of commands:

First, users can list the project names of available metagenomes by typing
# list available metagenomes at NCBI Genbank
listMetaGenomes()

```
[1] "metagenome"                        [6] "human gut metagenome"                        [11] "epibiont metagenome"
[4] "marine metagenome"                  [7] "soil metagenome"                              [12] "mine drainage metagenome"
[7] "mouse gut metagenome"               [10] "marine sediment metagenome"                    [13] "termite gut metagenome"
[10] "hot springs metagenome"             [11] "human lung metagenome"                         [14] "fossil metagenome"
[13] "freshwater metagenome"              [16] "saltern metagenome"                            [15] "stromatolite metagenome"
[16] "coral metagenome"                   [19] "mosquito metagenome"                              [16] "fish metagenome"
[19] "bovine gut metagenome"              [22] "chicken gut metagenome"                          [19] "wastewater metagenome"
[22] "microbial mat metagenome"           [25] "freshwater sediment metagenome"                 [22] "human metagenome"
[25] "hydrothermal vent metagenome"       [28] "compost metagenome"                              [25] "wallaby gut metagenome"
[28] "groundwater metagenome"             [31] "gut metagenome"                                 [28] "sediment metagenome"
[31] "ant fungus garden metagenome"       [34] "food metagenome"                                [31] "hypersaline lake metagenome"
[34] "hydrocarbon metagenome"             [37] "activated sludge metagenome"                     [34] "viral metagenome"
[37] "bioreactor metagenome"              [40] "wasp metagenome"                                [37] "permafrost metagenome"
[40] "sponge metagenome"                  [43] "aquatic metagenome"                                [40] "insect gut metagenome"
[43] "activated carbon metagenome"        [46] "anaerobic digestor metagenome"                 [43] "rock metagenome"
[46] "terrestrial metagenome"            [49] "rock porewater metagenome"                       [46] "seawater metagenome"
```

24
Internally the `listMetaGenomes()` function downloads the assembly_summary.txt file from ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/metagenomes/ to retrieve available metagenome information. This procedure might take a few seconds during the first run of `listMetaGenomes()`. Subsequently, the assembly_summary.txt file will be stored in the `tempdir()` directory to achieve a much faster access of this information during following uses of `listMetaGenomes()`.

In case users wish to retrieve detailed information about available metagenome projects they can specify the `details = TRUE` argument.

```r
# detailed information on available metagenomes at NCBI Genbank
listMetaGenomes(details = TRUE)
```

A tibble: 857 x 21

```r
table
```

| assembly_accession | bioproject | biosample | wgs_master | refseq_category | taxid | species_taxid |
|-------------------|------------|-----------|------------|-----------------|-------|--------------|
| `<chr>`            | `<chr>`    | `<chr>`   | `<chr>`    | `<chr>`         | `<int>`| `<int>`      |
| 1                  | GCA_000206185.1 | PRJNA32359 | SAMN02954317 | AAGA00000000.1 | na    | 256318       |
| 2                  | GCA_000206205.1 | PRJNA32355 | SAMN02954315 | AAFZ00000000.1 | na    | 256318       |
| 3                  | GCA_000206225.1 | PRJNA32357 | SAMN02954316 | AAFY00000000.1 | na    | 256318       |
| 4                  | GCA_000208265.2 | PRJNA17779 | SAMN02954240 | AASZ00000000.1 | na    | 256318       |
| 5                  | GCA_000208285.1 | PRJNA17657 | SAMN02954268 | AATD00000000.1 | na    | 256318       |
| 6                  | GCA_000208305.1 | PRJNA17659 | SAMN02954269 | AATN00000000.1 | na    | 256318       |
| 7                  | GCA_000208325.1 | PRJNA16729 | SAMN02954263 | AAQL00000000.1 | na    | 256318       |
| 8                  | GCA_000208345.1 | PRJNA16729 | SAMN02954282 | AAQK00000000.1 | na    | 256318       |
| 9                  | GCA_000208365.1 | PRJNA13699 | SAMN02954283 | AAFX00000000.1 | na    | 256318       |
| 10                 | GCA_900010595.1 | PRJEB11544 | SAMEA3639840 | CZPY00000000.1 | na    | 256318       |
| ...               | with 847 more rows, and 14 more variables: organism_name <chr>, infraspecific_name <chr>, isolate <chr>, version_status <chr>, assembly_level <chr>, release_type <chr>, genome_rep <chr>, seq_rel_date <date>, asm_name <chr>, submitter <chr>, gbrs_paired_asm <chr>, paired_asm_comp <chr>, ftp_path <chr>, excluded_from_refseq <chr> |

Finally, users can retrieve available metagenomes using `getMetaGenomes()` function. The `name` argument receives the metagenome project name retrieved with `listMetaGenomes()` function. The `path` argument specifies the folder path in which corresponding genomes shall be stored.

```r
# retrieve all genomes belonging to the human gut metagenome project
getMetaGenomes(name = "human gut metagenome", path = file.path("_ncbi_downloads","human_gut"))
```

1) "The metagenome of 'human gut metagenome' has been downloaded to '_ncbi_downloads/human_gut'."

[1] "_ncbi_downloads/human_gut/GCA_000205525.2_ASM20552v2_genomic.fna.gz"
[2] "_ncbi_downloads/human_gut/GCA_000205765.1_ASM20576v1_genomic.fna.gz"
[3] "_ncbi_downloads/human_gut/GCA_000205785.1_ASM20578v1_genomic.fna.gz"
[4] "_ncbi_downloads/human_gut/GCA_000207925.1_ASM20792v1_genomic.fna.gz"
[5] "_ncbi_downloads/human_gut/GCA_000207945.1_ASM20794v1_genomic.fna.gz"
[6] "_ncbi_downloads/human_gut/GCA_000207965.1_ASM20796v1_genomic.fna.gz"
[7] "_ncbi_downloads/human_gut/GCA_000207985.1_ASM20798v1_genomic.fna.gz"
[8] "_ncbi_downloads/human_gut/GCA_000208005.1_ASM20800v1_genomic.fna.gz"
[9] "_ncbi_downloads/human_gut/GCA_000208025.1_ASM20802v1_genomic.fna.gz"
[10] "_ncbi_downloads/human_gut/GCA_000208045.1_ASM20804v1_genomic.fna.gz"
[11] "_ncbi_downloads/human_gut/GCA_000208065.1_ASM20806v1_genomic.fna.gz"
[12] "_ncbi_downloads/human_gut/GCA_000208085.1_ASM20808v1_genomic.fna.gz"
[13] "_ncbi_downloads/human_gut/GCA_000208105.1_ASM20810v1_genomic.fna.gz"
[14] "_ncbi_downloads/human_gut/GCA_000208125.1_ASM20812v1_genomic.fna.gz"
[15] "_ncbi_downloads/human_gut/GCA_000208145.1_ASM20814v1_genomic.fna.gz"
[16] "_ncbi_downloads/human_gut/GCA_000208165.1_ASM20816v1_genomic.fna.gz"
...
Internally, `getMetaGenomes()` creates a folder specified in the `path` argument. Genomes associated with the metagenomes project specified in the `name` argument will then be downloaded and stored in this folder. As return value `getMetaGenomes()` returns the file paths to the genome files which can then be used as input to the `read*()` functions.

Alternatively or subsequent to the metagenome retrieval, users can retrieve annotation files of genomes belonging to a metagenome project selected with `listMetaGenomes()` by using the `getMetaGenomeAnnotations()` function.

```r
# retrieve all genomes belonging to the human gut metagenome project
getMetaGenomeAnnotations(name = "human gut metagenome", path = file.path("_ncbi_downloads","human_gut","annotations"))
```

```
[1] "The annotations of metagenome 'human gut metagenome'
have been downloaded and stored at '_ncbi_downloads/human_gut/annotations'.'
[1] "_ncbi_downloads/human_gut/annotations/GCA_000205525.2_ASM20552v2_genomic.gff.gz"
[2] "_ncbi_downloads/human_gut/annotations/GCA_000205765.1_ASM20576v1_genomic.gff.gz"
[3] "_ncbi_downloads/human_gut/annotations/GCA_000205785.1_ASM20578v1_genomic.gff.gz"
[4] "_ncbi_downloads/human_gut/annotations/GCA_000207925.1_ASM20792v1_genomic.gff.gz"
[5] "_ncbi_downloads/human_gut/annotations/GCA_000207945.1_ASM20794v1_genomic.gff.gz"
[6] "_ncbi_downloads/human_gut/annotations/GCA_000207965.1_ASM20796v1_genomic.gff.gz"
[7] "_ncbi_downloads/human_gut/annotations/GCA_000207985.1_ASM20798v1_genomic.gff.gz"
[8] "_ncbi_downloads/human_gut/annotations/GCA_000208005.1_ASM20800v1_genomic.gff.gz"
[9] "_ncbi_downloads/human_gut/annotations/GCA_000208025.1_ASM20802v1_genomic.gff.gz"
[10] "_ncbi_downloads/human_gut/annotations/GCA_000208045.1_ASM20804v1_genomic.gff.gz"
[11] "_ncbi_downloads/human_gut/annotations/GCA_000208065.1_ASM20806v1_genomic.gff.gz"
[12] "_ncbi_downloads/human_gut/annotations/GCA_000208085.1_ASM20808v1_genomic.gff.gz"
[13] "_ncbi_downloads/human_gut/annotations/GCA_000208105.1_ASM20810v1_genomic.gff.gz"
[14] "_ncbi_downloads/human_gut/annotations/GCA_000208125.1_ASM20812v1_genomic.gff.gz"
[15] "_ncbi_downloads/human_gut/annotations/GCA_000208145.1_ASM20814v1_genomic.gff.gz"
[16] "_ncbi_downloads/human_gut/annotations/GCA_000208165.1_ASM20816v1_genomic.gff.gz"
...```

The file paths of the downloaded *.gff are retured by `getMetaGenomeAnnotations()` and can be used as input for the `read.gff()` function in the seqreadr package.

## Retrieve Protein Sequences

Download all mammalian vertebrate proteomes.

Example RefSeq:

```r
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "refseq", type = "proteome")
```

Example Genbank:

```r
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "genbank", type = "proteome")
```

## Retrieve CDS Sequences

Download all mammalian vertebrate CDS from RefSeq (Genbank does not store CDS data).

Example RefSeq:

```r
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "refseq", type = "CDS")
```
Example Genbank:

```r
# download all vertebrate genomes
meta.retrieval(kingdom = "vertebrate_mammalian", db = "genbank", type = "CDS")
```

Retrieve GFF files

Download all mammalian vertebrate gff files.

Example RefSeq:

```r
# download all vertebrate gff files
meta.retrieval(kingdom = "vertebrate_mammalian", db = "refseq", type = "gff")
```

Example Genbank:

```r
# download all vertebrate gff files
meta.retrieval(kingdom = "vertebrate_mammalian", db = "genbank", type = "gff")
```

Users can obtain alternative kingdoms using `getKingdoms()`.

Retrieve Genomes for all kingdoms of life

If users wish to download the all genomes, proteome, CDS, or gff files for all species available in RefSeq or Genbank, they can use the `meta.retrieval.all()` function for this purpose.

Genome Retrieval

Example RefSeq:

```r
# download all geneomes stored in RefSeq
meta.retrieval.all(db = "refseq", type = "genome")
```

Example Genbank:

```r
# download all geneomes stored in Genbank
meta.retrieval.all(db = "genbank", type = "genome")
```

Proteome Retrieval

Example RefSeq:

```r
# download all proteome stored in RefSeq
meta.retrieval.all(db = "refseq", type = "proteome")
```

Example Genbank:

```r
# download all proteome stored in Genbank
meta.retrieval.all(db = "genbank", type = "proteome")
```

Functional Annotation with BioMart

The BioMart project enables users to retrieve a vast diversity of annotation data for specific organisms. Steffen Durinck and Wolfgang Huber provide an powerful interface between the R language and BioMart by providing the R package `biomaRt`. The following sections will introduce users to the functionality and data retrieval
precedures using the `biomaRt` package and will then introduce them to the interface functions `biomart()` and `biomart_organisms()` implemented in `biomart` that are based on the `biomaRt` methodology but aim to introduce an more intuitive way of interacting with BioMart.

### Getting Started with biomaRt

The best way to get started with the methodology presented by the established `biomaRt` package is to understand the workflow of data retrieval. The database provided by BioMart is organized in so called: **marts**, **datasets**, and **attributes**. So when users want to retrieve information for a specific organism of interest, first they need to specify the **marts** and **datasets** in which the information of the corresponding organism can be found. Subsequently they can specify the **attributes** argument that is ought to be returned for the corresponding organism.

The availability of **marts**, **datasets**, and **attributes** can be checked by the following functions:

```r
# install the biomaRt package
source("http://bioconductor.org/biocLite.R")
biocLite("biomaRt")

# load biomaRt
library(biomaRt)

# look at top 10 databases
head(listMarts(host = "www.ensembl.org"), 10)
```

| biomart                  | version |
|--------------------------|---------|
| 1 ENSEMBL_MART_ENSEMBL   | Ensembl Genes 83 |
| 2 ENSEMBL_MART_SNP       | Ensembl Variation 83 |
| 3 ENSEMBL_MART_FUNCGEN   | Ensembl Regulation 83 |
| 4 ENSEMBL_MART_VEGA      | Vega 63 |
| 5 pride                  | PRIDE (EBI UK) |

Users will observe that several **marts** providing annotation for specific classes of organisms or groups of organisms are available.

For our example, we will choose the **hsapiens_gene_ensembl** mart and list all available datasets that are element of this **mart**.

```r
head(listDatasets(useMart("ENSEMBL_MART_ENSEMBL", host = "www.ensembl.org")), 10)
```

| dataset             | description                                      | version  |
|---------------------|--------------------------------------------------|----------|
| oanatinus_gene_ensembl | Ornithorhynchus anatinus genes (OANA5)              | OANA5    |
| cporcellus_gene_ensembl | Cavia porcellus genes (cavPor3)                     | cavPor3  |
| gaculeatus_gene_ensembl | Gasterosteus aculeatus genes (BROADS1)             | BROADS1  |
| lafricana_gene_ensembl | Loxodonta africana genes (loxAfr3)                 | loxAfr3  |
| itridecemlineatus_gene_ensembl | Ictidomyys tridecemlineatus genes (spetri2)        | spetri2  |
| choffmanni_gene_ensembl | Choloepus hoffmanni genes (choHof1)               | choHof1  |
| csavignyi_gene_ensembl | Ciona savignyi genes (CSAV2.0)                     | CSAV2.0  |
| fcatus_gene_ensembl | Felis catus genes (Felis_catus_6.2) Felis_catus_6.2 | Felis_catus_6.2 |
| rnorvegicus_gene_ensembl | Rattus norvegicus genes (Rnor_6.0) Rnor_6.0        | Rnor_6.0 |
| psinensis_gene_ensembl | Pelodiscus sinensis genes (PelSin_1.0)             | PelSin_1.0 |

The **useMart()** function is a wrapper function provided by `biomaRt` to connect a selected BioMart database (**mart**) with a corresponding dataset stored within this **mart**.

We select dataset **hsapiens_gene_ensembl** and now check for available attributes (annotation data) that can be accessed for **Homo sapiens** genes.
Please note the nested structure of this attribute query. For an attribute query procedure an additional wrapper function named `useDataset()` is needed in which `useMart()` and a corresponding dataset needs to be specified. The result is a table storing the name of available attributes for *Homo sapiens* as well as a short description.

Furthermore, users can retrieve all filters for *Homo sapiens* that can be specified by the actual BioMart query process.

After accumulating all this information, it is now possible to perform an actual BioMart query by using the `getBM()` function.

In this example we will retrieve attributes: `start_position`, `end_position` and `description` for the *Homo sapiens* gene "GUCA2A".

Since the input genes are *ensembl* gene ids, we need to specify the `filters` argument `filters = "hgnc_symbol"`.

```r
# 1) select a mart and data set
mart <- useDataset(dataset = "hsapiens_gene_ensembl",
                   mart = useMart("ENSEMBL_MART_ENSEMBL",
                                   host = "www.ensembl.org")), 10)

# 2) run a biomart query using the getBM() function
# and specify the attributes and filter arguments
geneSet <- "GUCA2A"
```
resultTable <- getBM(attributes = c("start_position","end_position","description"),
    filters = "hgnc_symbol",
    values = geneSet,
    mart = mart)

resultTable

    start_position  end_position               description
1    42162691   42164718 guanylate cyclase activator 2A (guanylin) [Source:HGNC Symbol;Acc:HGNC:4682]

When using `getBM()` users can pass all attributes retrieved by `listAttributes()` to the `attributes` argument of the `getBM()` function.

**Getting Started with biomart**

This query methodology provided by BioMart and the biomart package is a very well defined approach for accurate annotation retrieval. Nevertheless, when learning this query methodology it (subjectively) seems non-intuitive from the user perspective. Therefore, the biomart package provides another query methodology that aims to be more organism centric.

Taken together, the following workflow allows users to perform fast BioMart queries for attributes using the `biomart()` function implemented in this biomart package:

1) get attributes, datasets, and marts via: `organismAttributes()`
2) choose available biological features (filters) via: `organismFilters()`
3) specify a set of query genes: e.g. retrieved with `getGenome()`, `getProteome()` or `getCDS()`
4) specify all arguments of the `biomart()` function using steps 1) - 3) and perform a BioMart query

Note that dataset names change very frequently due to the update of dataset versions. So in case some query functions do not work properly, users should check with `organismAttributes(update = TRUE)` whether or not their dataset name has been changed. For example, `organismAttributes("Homo sapiens", topic = "id", update = TRUE)` might reveal that the dataset ENSEMBL_MART_ENSEMBL has changed.

**Retrieve marts, datasets, attributes, and filters with biomart**

Retrieve Available Marts

The `getMarts()` function allows users to list all available databases that can be accessed through BioMart interfaces.

```
# load the biomart package
library(biomart)

# list all available databases
getMarts()
```

| mart                 | version             |
|----------------------|---------------------|
| ENSEMBL_MART_ENSEMBL | Ensembl Genes 87    |
| ENSEMBL_MART_MOUSE   | Mouse strains 87    |
Retrieve Available Datasets from a Specific Mart

Now users can select a specific database to list all available datasets that can be accessed through this database. In this example we choose the ENSEMBL_MART_ENSEMBL database.

```
head(getDatasets(mart = "ENSEMBL_MART_ENSEMBL") , 5)
```

```
dataset                  description             version
1  oanatinus_gene_ensembl Platypus genes (OANA5) OANA5
2  cporcellus_gene_ensembl Guinea Pig genes (cavPor3) cavPor3
3  gaculeatus_gene_ensembl Stickleback genes (BROAD S1) BROAD S1
4  lafricana_gene_ensembl Elephant genes (Loxafr3.0) Loxafr3.0
5  itridecemlineatus_gene_ensembl Squirrel genes (spetri2) spetri2
```

Now you can select the dataset hsapiens_gene_ensembl and list all available attributes that can be retrieved from this dataset.

```
tail(getDatasets(mart = "ENSEMBL_MART_ENSEMBL") , 38)
```

```
dataset
32  hsapiens_gene_ensembl
33  pformosa_gene_ensembl
34  tbelangeri_gene_ensembl
35  mfuro_gene_ensembl
36  ggallus_gene_ensembl
37  xtropicalis_gene_ensembl
38  ecaballus_gene_ensembl
39  pabelii_gene_ensembl
40  xmaculatus_gene_ensembl
41  drerio_gene_ensembl
42  tnigroviridis_gene_ensembl
43  lchalumnae_gene_ensembl
44  amelanoleuca_gene_ensembl
45  mmulatta_gene_ensembl
46  pvampyrus_gene_ensembl
47  panubis_gene_ensembl
48  mdomestica_gene_ensembl
49  acarolinensis_gene_ensembl
50  vpacos_gene_ensembl
51  tsyrichtha_gene_ensembl
52  ogarnettii_gene_ensembl
53  dmelanogaster_gene_ensembl
54  loculatus_gene_ensembl
55  mmurinus_gene_ensembl
56  olatipes_gene_ensembl
57  oprinceps_gene_ensembl
58  ggorilla_gene_ensembl
59  dordii_gene_ensembl
```

31
| Description                  | Version         |
|------------------------------|-----------------|
| Human genes                  | GRCh38.p7       |
| Amazon molly genes            | Poecilia_formosa-5.1.2 |
| Tree Shrew genes              | tupBel1         |
| Ferret genes                  | MusPutFur1.0    |
| Chicken genes                 | Gallus_gallus-5.0 |
| Xenopus genes                 | JGI 4.2         |
| Horse genes                   | Equ Cab 2       |
| Orangutan genes               | PPGY2           |
| Platypus fish genes           | Xipmac4.4.2     |
| Zebrafish genes               | GRCz10          |
| Tetraodon genes               | TETRAODON 8.0   |
| Coelacanth genes              | LatCha1         |
| Panda genes                   | ailMel1         |
| Macaque genes                 | Mmu1_8.0.1      |
| Megabat genes                 | pteVam1         |
| Olive baboon genes            | PapAnu2.0       |
| Opossum genes                 | monDom5         |
| Anole lizard genes            | AnoCar2.0       |
| Alpaca genes                  | vicPac1         |
| Tarsier genes                 | tarSyr1         |
| Bushbaby genes                | OtoGar3         |
| Fruitfly genes                | BDGP6           |
| Spotted gar genes             | LepOcu1         |
| Mouse Lemur genes             | Mmuur_2.0       |
| Medaka genes                  | HdrR            |
| Pika genes                    | OchPri2.0       |
| Gorilla genes                 | gorGor3.1       |
| Kangaroo rat genes            | dipOrd1         |
| Sheep genes                   | Oar_v3.1        |
| Mouse genes                   | GRCh38_p5       |
| Turkey genes                  | Turkey_2.01     |
| Cod genes                     | gadMor1         |
| Shrew genes                   | sorAra1         |
| Duck genes                    | BGI_duck_1.0    |
| Tasmanian devil genes         | Devil_ref v7.0  |
| Cow genes                     | UMD3.1          |
| Wallaby genes                 | Meug_1.0        |
| Dog genes                     | CanFam3.1       |
Retrieve Available Attributes from a Specific Dataset

Now that you have selected a database (hsapiens_gene_ensembl) and a dataset (hsapiens_gene_ensembl), users can list all available attributes for this dataset using the `getAttributes()` function.

```
# list all available attributes for dataset: hsapiens_gene_ensembl
head( getAttributes(mart = "ENSEMBL_MART_ENSEMBL",
                   dataset = "hsapiens_gene_ensembl"), 10 )
```

| name                      | description            |
|--------------------------|------------------------|
| ensembl_gene_id          | Gene ID                |
| ensembl_transcript_id    | Transcript ID          |
| ensembl_peptide_id       | Protein ID             |
| ensembl_exon_id          | Exon ID                |
| description              | Description            |
| chromosome_name          | Chromosome/scaffold name |
| start_position           | Gene Start (bp)        |
| end_position             | Gene End (bp)          |
| strand                   | Strand                 |
| band                     | Band                   |

Retrieve Available Filters from a Specific Dataset

Finally, the `getFilters()` function allows users to list available filters for a specific dataset that can be used for a `biomart()` query.

```
# list all available filters for dataset: hsapiens_gene_ensembl
head( getFilters(mart = "ENSEMBL_MART_ENSEMBL",
                 dataset = "hsapiens_gene_ensembl"), 10 )
```

| name                     | description               |
|--------------------------|---------------------------|
| chromosome_name          | Chromosome name           |
| start                    | Gene Start (bp)           |
| end                      | Gene End (bp)             |
| band_start               | Band Start                |
| band_end                 | Band End                  |
| marker_start             | Marker Start              |
| marker_end               | Marker End                |
| encode_region            | Encode region             |
| strand                   | Strand                    |
| chromosomal_region       | Chromosome Regions (e.g 1:100:10000:-1,1:100000:200000:1) |

Organism Specific Retrieval of Information

In most use cases, users will work with a single or a set of model organisms. In this process they will mostly be interested in specific annotations for this particular model organism. The `organismBM()` function addresses this issue and provides users with an organism centric query to marts and datasets which are available for a particular organism of interest.

**Note** that when running the following functions for the first time, the data retrieval procedure will take some time, due to the remote access to BioMart. The corresponding result is then saved in a *.txt file named _biomart/listDatasets.txt within the tempdir() folder, allowing subsequent queries to be performed much faster. The tempdir() folder, however, will be deleted after a new R session was established. In this case the initial call of the subsequent functions again will take time to retrieve all organism specific data from the BioMart database.
This concept of locally storing all organism specific database linking information available in BioMart into an internal file allows users to significantly speed up subsequent retrieval queries for that particular organism.

```
# retrieving all available datasets and biomart connections for
# a specific query organism (scientific name)
organismBM(organism = "Homo sapiens")
```

| organism_name | description                                                      |
|---------------|------------------------------------------------------------------|
| hsapiens      | Human genes (GRCh38.p7)                                          |
| hsapiens      | homo_sapiens sequences (GRCh38.p7)                               |
| hsapiens      | Human Short Variants (SNPs and indels excluding flagged variants) (GRCh38.p7) |
| hsapiens      | Human Structural Variants (GRCh38.p7)                            |
| hsapiens      | Human Somatic Structural Variants (GRCh38.p7)                    |
| hsapiens      | Human Somatic Short Variants (SNPs and indels excluding flagged variants) (GRCh38.p7) |
| hsapiens      | Human Regulatory Evidence (GRCh38.p7)                            |
| hsapiens      | Human Binding Motifs (GRCh38.p7)                                 |
| hsapiens      | Human Regulatory Features (GRCh38.p7)                            |
| hsapiens      | Human miRNA Target Regions (GRCh38.p7)                           |
| hsapiens      | Human Other Regulatory Regions (GRCh38.p7)                       |
| hsapiens      | Human genes (GRCh38.p7)                                          |

With 3 more variables: mart <chr>, dataset <chr>, version <chr>

The result is a table storing all marts and datasets from which annotations can be retrieved for *Homo sapiens*. Furthermore, a short description as well as the version of the dataset being accessed (very useful for publications) is returned.

Users will observe that 3 different marts provide 6 different datasets storing annotation information for *Homo sapiens*.

Please note, however, that scientific names of organisms must be written correctly! For ex. “Homo Sapiens” will be treated differently (not recognized) than “Homo sapiens” (recognized).

Similar to the biomaRt package query methodology, users need to specify attributes and filters to be able to perform accurate BioMart queries. Here the functions organismAttributes() and organismFilters() provide useful and intuitive concepts to obtain this information.

```
# return available attributes for "Homo sapiens"
head(organismAttributes("Homo sapiens"), 20)
```

| name                      | description                        |
|---------------------------|------------------------------------|
| ensembl_gene_id           | Gene ID                            |
| ensembl_transcript_id     | Transcript ID                      |
| ensembl_peptide_id        | Protein ID                         |
| chromosome_name           | Chromosome/scaffold name            |
| start_position            | Gene Start (bp)                    |
| end_position              | Gene End (bp)                      |
| strand                    | Strand                             |
| band                      | Band                               |
| transcript_start          | Transcript Start (bp)              |
| transcript_end            | Transcript End (bp)                |
| transcription_start_site  | Transcription Start Site (TSS)      |
| transcript_length         | Transcript length (including UTRs and CDS) |
| transcript_tsl            | Transcript Support Level (TSL)      |
| gencode_basic             | GENCODE basic annotation            |
Users will observe that the `organismAttributes()` function returns a data.frame storing attribute names, datasets, and marts which are available for Homo sapiens. After the ENSEMBL release 87 the ENSEMBL_MART_SEQUENCE service provided by Ensembl does not work properly and thus the `organismAttributes()` function prints out warning messages to make the user aware when certain marts provided by Ensembl do not work properly, yet.

An additional feature provided by `organismAttributes()` is the `topic` argument. The `topic` argument allows users to search for specific attributes, topics, or categories for faster filtering.

```r
# search for attribute topic "id"
head(organismAttributes("Homo sapiens", topic = "id"), 20)
```

| name                      | description                                      | dataset                                      |
|---------------------------|--------------------------------------------------|----------------------------------------------|
| ensembl_gene_id           | Gene ID                                          | hsapiens_gene_ensembl                        |
| ensembl_transcript_id     | Transcript ID                                    | hsapiens_gene_ensembl                        |
| ensembl_peptide_id        | Protein ID                                       | hsapiens_gene_ensembl                        |
| ensembl_exon_id           | Exon ID                                          | hsapiens_gene_ensembl                        |
| study_external_id         | Study External Reference                         | hsapiens_gene_ensembl                        |
| go_id                     | GO Term Accession                                | hsapiens_gene_ensembl                        |
| dbass3_id                 | Database of Aberrant 3 Splice Sites (DBASS3) IDs | hsapiens_gene_ensembl                        |
| dbass5_id                 | Database of Aberrant 5 Splice Sites (DBASS5) IDs | hsapiens_gene_ensembl                        |
| hgnc_id                   | HGNC ID(s)                                       | hsapiens_gene_ensembl                        |
| mirbase_id                | miRBase ID(s)                                    | hsapiens_gene_ensembl                        |
| mim_morbid                | MIM MORBID                                       | hsapiens_gene_ensembl                        |
| protein_id                | Protein (Genbank) ID [e.g. AAA02487]             | hsapiens_gene_ensembl                        |
| refseq_peptide            | RefSeq Protein ID [e.g. NP_001005353]            | hsapiens_gene_ensembl                        |
| refseq_peptide_predicted  | RefSeq Predicted Protein ID [e.g. XP_001720922]  | hsapiens_gene_ensembl                        |
| wikigene_id               | WikiGene ID                                      | hsapiens_gene_ensembl                        |
| ensembl_gene_id           | Gene ID                                          | hsapiens_gene_ensembl                        |
| ensembl_transcript_id     | Transcript ID                                    | hsapiens_gene_ensembl                        |
| ensembl_peptide_id        | Protein ID                                       | hsapiens_gene_ensembl                        |
| ensembl_exon_id           | Exon ID                                          | hsapiens_gene_ensembl                        |
| ensembl_gene_id           | Gene ID                                          | hsapiens_gene_ensembl                        |

Now, all attribute names having id as part of their name are being returned.

Another example is `topic = "homolog"`.

```r
# search for attribute topic "homolog"
head(organismAttributes("Homo sapiens", topic = "homolog"), 20)
```

| name                      | description                                      |
|---------------------------|--------------------------------------------------|
| vpacos_homolog_ensembl_gene|                                                 |
### Attributes

VPAcOS homologs include:

- **Gene Name**
- **Ensembl Peptide**
- **Chromosome**
- **Chromosomal Position**
- **Canonical Transcript Protein**
- **Subtype**
- **Orthology Type**
- **Percent ID**
- **Percent ID R1**
- **GOC Score**
- **WGA Coverage**
- **DN**
- **DS**

**Additional Variables**:
- **Description**
- **Dataset**
- **Mart**

#### Example Code

```r
# Search for attribute topic "dn"
head(organismAttributes("Homo sapiens", topic = "dn"))
```

**Tibble**:

- **name**: cdna_coding_start
- **description**: cDNA coding start
- **dataset**: hsapiens_gene_ensembl
- **mart**: ENSEMBL_MART_ENSEMBL

#### Additional Code

**Organism Filters**

- **Chromosome Name**

```r
# Return available filters for "Homo sapiens"
head(organismFilters("Homo sapiens"), 20)
```

**Example Filters**:

- **cds**: CDS ID
- **cds_length**: CDS Length
- **cds_start**: CDS Start
- **cds_end**: CDS End
- **vpacos_homolog_ds**: dS with Alpaca
- **pformosa_homolog_ds**: dS with Amazon molly

**Analogous Functions**

- **organismAttributes()**
- **organismFilters()**
The `organismFilters()` function also allows users to search for filters that correspond to a specific topic or category.

```r
# search for filter topic "id"
head(organismFilters("Homo sapiens", topic = "id"), 20)
```

| name                                      | description                                   |
|-------------------------------------------|-----------------------------------------------|
| with_go_id                                | with GO Term Accession(s)                     |
| with_mim_morbid                           | with MIM MORBID ID(s)                        |
| with_protein_id                           | with protein (Genbank) ID(s)                  |
| with_refseq_peptide                       | with RefSeq predicted protein ID(s)           |
| ensembl_gene_id                           | Gene ID(s) [e.g. ENSG00000139618]            |
| ensembl_transcript_id                     | Transcript ID(s) [e.g. ENST00000380152]      |
| ensembl_peptide_id                        | Protein ID(s) [e.g. ENSP00000369497]         |
| ensembl_exon_id                           | Exon ID(s) [e.g. ENSE00001508081]            |
| hgnc_id                                   | HGNC ID(s) [e.g. HGNC:8030]                  |
| go_id                                     | GO Term Accession(s) [e.g. GO:0005515]        |
| mim_morbid                                | MIM MORBID ID(s) [e.g. 100100]               |
| mirbase_id                                | miRBase ID(s) [e.g. hsa-mir-137]             |
| protein_id                                | Protein (Genbank) ID(s) [e.g. ACU09872]      |
| refseq_peptide                            | RefSeq protein ID(s) [e.g. NP_001005353]     |
| refseq_peptide_predicted                  | RefSeq predicted protein ID(s) [e.g. XP_011520427] |
| wikigene_id                               | WikiGene ID(s) [e.g. 115286]                 |
| go_evidence_code                          | GO Evidence code                             |
| with_itridecemlineatus_homolog            | Orthologous Squirrel Genes                   |
| with_tnigroviridis_homolog                | Orthologous Tetraodon Genes                  |
```

with 2 more variables: `dataset <chr>`, `mart <chr>`
Performing BioMart queries with biomartR

The short introduction to the functionality of `organismBM()`, `organismAttributes()`, and `organismFilters()` will allow users to perform BioMart queries in a very intuitive organism centric way. The main function to perform BioMart queries is `biomart()`.

For the following examples we will assume that we are interested in the annotation of specific genes from the *Homo sapiens* proteome. We want to map the corresponding refseq gene id to a set of other gene ids used in other databases. For this purpose, first we need consult the `organismAttributes()` function.

```r
head(organismAttributes("Homo sapiens", topic = "id"))
```

| name                  | description         | dataset                | mart                      |
|-----------------------|---------------------|------------------------|---------------------------|
| ensembl_gene_id       | Gene ID             | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |
| ensembl_transcript_id | Transcript ID       | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |
| ensembl_peptide_id    | Protein ID          | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |
| ensembl_exon_id       | Exon ID             | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |
| study_external_id     | Study External Reference | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |
| go_id                 | GO Term Accession   | hsapiens_gene_ensembl  | ENSEMBL_MART_ENSEMBL      |

# retrieve the proteome of Homo sapiens from refseq
```r
file_path <- getProteome(db = "refseq",
                          organism = "Homo sapiens",
                          path = file.path("_ncbi_downloads","proteomes") )

Hsapiens_proteome <- read_proteome(file_path, format = "fasta")
```

# remove splice variants from id
```r
gene_set <- unlist(sapply(strsplit(Hsapiens_proteome@ranges@NAMES[1:5], ".",fixed = TRUE), function(x) x[1]))
```

```r
result_BM <- biomart(genes = gene_set,
                     mart = "ENSEMBL_MART_ENSEMBL",
                     dataset = "hsapiens_gene_ensembl",
                     attributes = c("ensembl_gene_id","ensembl_peptide_id"),
                     filters = "refseq_peptide")
```

| refseq_peptide | ensembl_gene_id | ensembl_peptide_id |
|----------------|-----------------|--------------------|
| NP_000005      | ENSG00000175899 | ENSP00000323929    |
| NP_000006      | ENSG00000156006 | ENSP00000286479    |
| NP_000007      | ENSG00000117054 | ENSP00000359878    |
| NP_000008      | ENSG00000122971 | ENSP00000242592    |
| NP_000009      | ENSG00000072778 | ENSP00000349297    |

The `biomart()` function takes as arguments a set of genes (gene ids specified in the `filter` argument), the corresponding `mart` and `dataset`, as well as the `attributes` which shall be returned.

**Gene Ontology**

The `biomart` package also enables a fast and intuitive retrieval of GO terms and additional information via the `getGO()` function. Several databases can be selected to retrieve GO annotation information for a set of query genes. So far, the `getGO()` function allows GO information retrieval from the BioMart database.

In this example we will retrieve GO information for a set of *Homo sapiens* genes stored as `hgnc_symbol`. 
GO Annotation Retrieval via BioMart

The `getGO()` function takes several arguments as input to retrieve GO information from BioMart. First, the scientific name of the organism of interest needs to be specified. Furthermore, a set of gene ids as well as their corresponding filter notation (GUCA2A gene ids have filter notation `hgnc_symbol`; see `organismFilters()` for details) need to be specified. The database argument then defines the database from which GO information shall be retrieved.

```r
# search for GO terms of an example Homo sapiens gene
GO_tbl <- getGO(organism = "Homo sapiens",
                genes = "GUCA2A",
                filters = "hgnc_symbol")
```

Hence, for each gene id the resulting table stores all annotated GO terms found in BioMart.