Description of the biomaRt package

Steffen Durinck[‡*], Wolfgang Huber[¶†],
Yves Moreau[‡], Bart De Moor[‡]

July 25, 2006

[‡]Department of Electronical Engineering, ESAT-SCD, K.U.Leuven,
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium,
`http://www.esat.kuleuven.ac.be/~dna/BioI`
and [¶]European Bioinformatics Institute, Hinxton, UK

Contents

*Steffen.Durinck@esat.kuleuven.ac.be
†huber@ebi.ac.uk

1 Introduction

The BioConductor *biomaRt* package provides and API in R to query BioMart databases such as Ensembl (http://www.ensembl.org), a software system which produces and maintains automatic annotation on metazoan genomes. Two sets of functions are currently implemented. A first set of functions is tailored towards Ensembl and are a set of commonly used queries in microarray data analysis. A second set of functions aims to mimic functionality of other BioMart APIs such as Martshell, Martview, etc. (see http://www.biomart.org for more information). These functions are very general, and can be used with any BioMart system. They allow retrieval of all information that other BioMart APIs provide. With these two sets of functions, one can for example annotate the features on your array with the latest annotations starting from identifiers such as affy ids, RefSeq, entrezgene,.. Annotation includes gene names, GO, OMIM annotation, etc. On top of this, biomaRt enables you to retrieve any type of information available from the BioMart databases from R.

2 objects

2.1 Mart-class

An object of the `Mart` class stores connections to BioMart databases and aditional information about the BioMarts. It has the following slots:

- `mysql`: Logical indicating if access to BioMart database should use MySQL or use the BioMart webservice over HTTP (default)
- `connections`: Stores the MySQLConnections
- `mysqldriver`: Stores the MySQL driver
- `mainTables`: List of the main tables in the BioMart database
- `biomart`: Name of the BioMart database
- `host`: Hostname of the BioMart database

- `dataset`: Name of the dataset that is in use
- `filters`: Environment that stores information on BioMart filters
- `attributes`: Environment that stores information on BioMart attributes

3 Simple biomaRt functions for frequently used queries to Ensembl

In this section we describe a set of simple functions which are frequently used in the microarray community. More powerful functions and data retrieval from all BioMart databases is described in the next section "Advanced data retrieval with BioMart API functions".

3.1 Selecting a BioMart database to use

A first step in using the biomaRt package is to connect to a BioMart database. The function martConnect establishes a connection with one or more of the following BioMart databases: snp, ensembl, sequence and vega. Default this function will connect to public BioMart databases. If no biomart is specified, only a connection to ensembl will be established. If you want to use local BioMart install you have to set the local argument to TRUE and specify host, password and user details in the corresponding arguments.

3.1.1 `useMart`

```
Loading required package: Biobase
```

```
> library(biomaRt)
```

```
Loading required package: XML
Loading required package: RCurl

Attaching package: 'biomaRt'


        The following object(s) are masked from package:annotate :

         getGO
```

```
> mart <- useMart("ensembl")
```

3.1.2 `martDisconnect`

When using MySQL access, you can only hold a limited number of connections with different BioMarts. The function martDisconnect can be used to close a mart connection.

```
> martDisconnect(mart)
```

3.2 Annotating identifiers with gene information

The function `getGene` uses a query id to look up the name, description and chromosomal information of the corresponding gene. Currently the `getGene` function takes identifiers from entrezgene, ensembl, refseq, affy, hugo, unigene and embl.
The *id* argument is either a vector of identifiers or a single identifier to be annotated.
The *array* argument takes affy array identifiers as values. A list of possible identifiers supported by the package can be obtained by executing the function `getAffyArrays`.
The *mart* argument is a mart connection, which was obtained using the method `martConnect`.
The *type* takes the values of 'entrezgene','refseq','hugo', 'ensembl' and 'embl' to clarify which type of identifier is specified in the id argument.

First we select the BioMart databases and the dataset we want to use.

```
> mart <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")

Checking attributes and filters ... ok
```

Then we check which affy arrays are available:

```
> getAffyArrays(mart)

 [1] "affy_hc_g110"        "affy_hg_focus"       "affy_hg_u133a"
 [4] "affy_hg_u133a_2"     "affy_hg_u133b"       "affy_hg_u133_plus_2"
 [7] "affy_hg_u95av2"      "affy_hg_u95b"        "affy_hg_u95c"
[10] "affy_hg_u95d"        "affy_hg_u95e"        "affy_hugenefl"
[13] "affy_u133_x3p"
```

Assume now that we have some upregulated features that we want to annotate. To get the gene information on a certain affy array do:

```
> upregulated <- c("210708_x_at", "202763_at", "211464_x_at")
> getGene(id = upregulated, array = "affy_hg_u133_plus_2", mart = mart)

           ID symbol
1   202763_at  CASP3
2 210708_x_at
3 210708_x_at
4 210708_x_at CASP10
5 210708_x_at
6 210708_x_at
7 211464_x_at  CASP6
8 211464_x_at
```

```
1                                                    Caspase-3 precursor (EC 3.4.22.-)
2 Caspase-10 precursor (EC 3.4.22.-) (CASP-10) (ICE-like apoptotic protease 4) (Apoptoti
3 Caspase-10 precursor (EC 3.4.22.-) (CASP-10) (ICE-like apoptotic protease 4) (Apoptoti
4 Caspase-10 precursor (EC 3.4.22.-) (CASP-10) (ICE-like apoptotic protease 4) (Apoptoti
5 Caspase-10 precursor (EC 3.4.22.-) (CASP-10) (ICE-like apoptotic protease 4) (Apoptoti
6 Caspase-10 precursor (EC 3.4.22.-) (CASP-10) (ICE-like apoptotic protease 4) (Apoptoti
7
8
  chromosome  band strand chromosome_start chromosome_end ensembl_gene_id
1          4 q35.1     -1        185785845      185807623 ENSG00000164305
2          2 q33.1      1        201756100      201802372 ENSG00000003400
3          2 q33.1      1        201756100      201802372 ENSG00000003400
4          2 q33.1      1        201756100      201802372 ENSG00000003400
5          2 q33.1      1        201756100      201802372 ENSG00000003400
6          2 q33.1      1        201756100      201802372 ENSG00000003400
7          4   q25     -1        110829234      110844078 ENSG00000138794
8          4   q25     -1        110829234      110844078 ENSG00000138794
  ensembl_transcript_id
1       ENST00000308394
2       ENST00000272879
3       ENST00000360132
4       ENST00000286186
5       ENST00000346817
6       ENST00000313728
7       ENST00000265164
8       ENST00000352981
```

When using other id's we have to specify the `type` and `species`, use the function `getSpecies` to find valid species names.

```
> getGene(id = 100, type = "entrezgene", mart = mart)

   ID symbol
1 100    ADA
2 100

1 Adenosine deaminase (EC 3.5.4.4) (Adenosine aminohydrolase). [Source:Uniprot/SWISSPROT
2 Adenosine deaminase (EC 3.5.4.4) (Adenosine aminohydrolase). [Source:Uniprot/SWISSPROT
  chromosome   band strand chromosome_start chromosome_end ensembl_gene_id
1         20 q13.12     -1         42681577       42713797 ENSG00000196839
2         20 q13.12     -1         42681577       42713797 ENSG00000196839
  ensembl_transcript_id
1       ENST00000372874
2       ENST00000359372
```

3.2.1 GO annotation

Gene Onotology annotation can be retrieved with the function `getGO`. The arguments are the same as the function `getGene`.

```
> go <- getGO(id = "203140_at", array = "affy_hg_u133_plus_2",
+     mart = mart)
> go
```

```
          ID      go_id
1  203140_at GO:0003700
2  203140_at GO:0005515
3  203140_at GO:0008270
4  203140_at GO:0046872
5  203140_at GO:0000122
6  203140_at GO:0006350
7  203140_at GO:0006355
8  203140_at GO:0006954
9  203140_at GO:0008284
10 203140_at GO:0000119
11 203140_at GO:0005634
12 203140_at GO:0043066
13 203140_at GO:0043066
14 203140_at GO:0007283
15 203140_at GO:0007283
16 203140_at GO:0016564
17 203140_at GO:0016564
                                                    go_description
1                                        transcription factor activity
2                                                    protein binding
3                                                   zinc ion binding
4                                                  metal ion binding
5  negative regulation of transcription from RNA polymerase II promoter
6                                                      transcription
7                          regulation of transcription, DNA-dependent
8                                                inflammatory response
9                           positive regulation of cell proliferation
10                                                 mediator complex
11                                                         nucleus
12                                  negative regulation of apoptosis
13                                  negative regulation of apoptosis
14                                                   spermatogenesis
15                                                   spermatogenesis
16                                transcriptional repressor activity
```

```
17                                    transcriptional repressor activity
   evidence_code ensembl_gene_id ensembl_transcript_id
1            NAS ENSG00000113916       ENST00000232014
2            IPI ENSG00000113916       ENST00000232014
3            IEA ENSG00000113916       ENST00000232014
4            IEA ENSG00000113916       ENST00000232014
5             NR ENSG00000113916       ENST00000232014
6            IEA ENSG00000113916       ENST00000232014
7            TAS ENSG00000113916       ENST00000232014
8            TAS ENSG00000113916       ENST00000232014
9            TAS ENSG00000113916       ENST00000232014
10            NR ENSG00000113916       ENST00000232014
11           IDA ENSG00000113916       ENST00000232014
12           IMP ENSG00000113916       ENST00000232014
13           IEA ENSG00000113916       ENST00000232014
14           IMP ENSG00000113916       ENST00000232014
15           IEA ENSG00000113916       ENST00000232014
16           IDA ENSG00000113916       ENST00000232014
17           IEA ENSG00000113916       ENST00000232014
```

3.2.2 OMIM annotation

OMIM annotation can be retrieved with the function `getOMIM`. The arguments are the same as the function `getGene`.

3.2.3 INTERPRO protein domains

INTERPRO protein domains of the corresponding proteins can be searched with the function `getINTERPRO`. Again the arguments are the same as the function `getGene`.

```
> getINTERPRO(id = "1939_at", array = "affy_hg_u95av2", mart = mart)

       ID interpro_id          description ensembl_gene_id
1 1939_at   IPR002117    p53 tumor antigen ENSG00000141510
2 1939_at   IPR011615     p53, DNA-binding ENSG00000141510
3 1939_at   IPR010991 p53, tetramerisation ENSG00000141510
4 1939_at   IPR000694  Proline-rich region ENSG00000141510
  ensembl_transcript_id
1       ENST00000269305
2       ENST00000269305
3       ENST00000269305
4       ENST00000269305
```

3.3 Homology mapping

This function maps homologs of genes of one species to another species. To use the function one needs two instances of a mart object where two different datasets are selected e.g. hsapiens_gene_ensembl and mmusculus_gene_ensembl if you want to map homologues between these two species. Now we can look for homologs:

```
> from.mart = useMart("ensembl", dataset = "hsapiens_gene_ensembl")

Checking attributes and filters ... ok

> to.mart = useMart("ensembl", dataset = "mmusculus_gene_ensembl")

Checking attributes and filters ... ok

> getHomolog(id = 2, from.type = "entrezgene", to.type = "refseq",
+     from.mart = from.mart, to.mart = to.mart)

                  V1                  V2        V3
1 ENSMUSG00000030111 ENSMUST00000032203 NM_175628
```

3.4 Identify subsets of genes for further analysis with the `getFeature` function

The function `getFeature` is a general function to look up identifiers which pass a certain filter. A first such a filter is to look for identifiers that correspond to genes with a given symbol. If the array argument is given then affy identifiers from that array will be returned. For retrieving other identifiers one has to specify the species and the type of identifier to retrieve.

```
> getFeature(symbol = "BRCA2", array = "affy_hg_u133_plus_2", mart = mart)

  hgnc_symbol affy_hg_u133_plus_2
1       BRCA2         208368_s_at
2       BRCA2         208368_s_at
```

A second possible filter is to look for ids which have a certain OMIM disease term attached to them (this only works for hsapiens). Similarly one can look for ids that have a certain GO annotation e.g. retrieve all affy id's on the hgu133plus2 array which have protein-tyrosine kinase activity.

An other filter uses the position of genes on the genome. One can query for all genes on a certain chromosome:

```
> ychrom <- getFeature(chromosome = "Y", type = "entrezgene", mart = mart)
> ychrom[1:10, ]
```

```
   ensembl_transcript_id chromosome_name entrezgene
1        ENST00000381670               Y      55344
2        ENST00000381663               Y      55344
3        ENST00000331098               Y      55344
4        ENST00000381657               Y      55344
5        ENST00000381656               Y      55344
6        ENST00000326153               Y       8225
7        ENST00000381625               Y      28227
8        ENST00000381619               Y      28227
9        ENST00000381612               Y      28227
10       ENST00000381610               Y      28227
```

Or query for genes that lay in a particular region:

```
> getFeature(chromosome = 1, start = 2800000, end = 3200000, type = "entrezgene",
+     mart = mart)
```

```
   ensembl_transcript_id chromosome_name start_position end_position entrezgene
1        ENST00000378404               1        2927907      2929327     140625
2        ENST00000304706               1        2927907      2929327     140625
3        ENST00000321336               1        2970496      2974193     440556
4        ENST00000378398               1        2975621      3345045      63976
5        ENST00000378398               1        2975621      3345045     647868
6        ENST00000270722               1        2975621      3345045      63976
7        ENST00000270722               1        2975621      3345045     647868
8        ENST00000378391               1        2975621      3345045      63976
9        ENST00000378391               1        2975621      3345045     647868
10       ENST00000378389               1        2975621      3345045         NA
11       ENST00000378388               1        2975621      3345045         NA
```

3.5 Sequence information

The function `getSequence` retrieves the sequence given its chromosome, start and end position.

3.6 Single Nucleotide Polymorphisms

The function `getSNP` retrieves all SNP's between a given a start and end position on a gives chromosome.. Note: make sure you have a Mart object with connections to ensembl and snp

```
> mart = useMart("snp", dataset = "hsapiens_snp")

Checking attributes and filters ... ok

> getSNP(chromosome = 8, start = 148350, end = 148612, mart = mart)

        tscid  refsnp_id allele chrom_start chrom_strand
1  TSC1723456  rs3969741    C/A      148394            1
2  TSC1421398  rs4046274    C/A      148394            1
3  TSC1421399  rs4046275    A/G      148411            1
4               rs13291    C/T      148462            1
5  TSC1421400  rs4046276    C/T      148462            1
6              rs4483971    C/T      148462            1
7             rs17355217    C/T      148462            1
8             rs12019378    T/G      148471            1
9  TSC1421401  rs4046277    G/A      148499            1
10            rs11136408    G/A      148525            1
11 TSC1421402  rs4046278    G/A      148533            1
12            rs17419210    C/T      148533           -1
13            rs28735600    G/A      148533            1
14 TSC1737607  rs3965587    C/T      148535            1
15             rs4378731    G/A      148601            1
```

3.7 More exotic functions

3.7.1 getPossibleXrefs

This function retrieves the possible cross-references present in Ensembl. This is a very general function to see what can be extracted from Ensembl. The results of this function can be used in the getXref function to extract the data of interest.

3.7.2 getXref

This function retrieves any cross reference in Ensembl. It can for example be used to map different affymetrix array within one species. E.g. starting from an affy id of chip hgu95av2 and id 1939_at, look for corresponding affy identifiers on the affy hgu133plus2 chip.

4 Advanced data retrieval with BioMart API functions

In this section we'll discuss functions that resemble other BioMart APIs such as Martshell (see: http://www.biomart.org for more info). These functions are very general and can be used on all BioMart databases. The order in which the functions are discussed is the usual order of how you should use them.

4.1 listMarts

The `listMarts` lists the possible BioMarts where we can connect to.

```
> library(biomaRt)
> marts <- listMarts()
> marts

$biomart
[1] "dicty"    "ensembl"  "snp"      "vega"     "uniprot"  "msd"      "wormbase"

$version
[1] "DICTYBASE (NORTHWESTERN)"    "ENSEMBL 39 (SANGER)"
[3] "SNP 39 (SANGER)"             "VEGA 39 (SANGER)"
[5] "UNIPROT PROTOTYPE 4-5 (EBI)" "MSD PROTOTYPE 4 (EBI)"
[7] "WORMBASE CURRENT (CSHL)"

$host
[1] "www.dictybase.org" "www.biomart.org"   "www.biomart.org"
[4] "www.biomart.org"   "www.biomart.org"   "www.biomart.org"
[7] "www.biomart.org"

$path
[1] ""                      "/biomart/martservice" "/biomart/martservice"
[4] "/biomart/martservice" "/biomart/martservice" "/biomart/martservice"
[7] "/biomart/martservice"

$vschema
[1] "dicty"   "default" "default" "default" "default" "default" "default"
```

4.2 useMart

Here we select from the list of possible BioMart databases, a BioMart that we want to use.

```
> mart <- useMart("ensembl")
```

4.3 listDatasets

Next we want to select a specific dataset of the selected BioMart. To see which dataset is available we use the function `listDatasets`.

```
> listDatasets(mart)
```

```
                      dataset    version
1    rnorvegicus_gene_ensembl    RGSC3.4
2    scerevisiae_gene_ensembl       SGD1
3       celegans_gene_ensembl     CEL150
4      trubripes_gene_ensembl      FUGU4
5  cintestinalis_gene_ensembl       JGI2
6   ptroglodytes_gene_ensembl    CHIMP1A
7       agambiae_gene_ensembl     AgamP3
8       hsapiens_gene_ensembl     NCBI36
9        ggallus_gene_ensembl    WASHUC1
10   xtropicalis_gene_ensembl     JGI4.1
11        drerio_gene_ensembl     ZFISH6
12 tnigroviridis_gene_ensembl TETRAODON7
13      mmulatta_gene_ensembl   MMUL_0_1
14    mdomestica_gene_ensembl    BROAD03
15 dmelanogaster_gene_ensembl    BDGP4.2
16     mmusculus_gene_ensembl    NCBIM36
17       btaurus_gene_ensembl   Btau_2.0
18   cfamiliaris_gene_ensembl    BROADD1
```

4.4 useDataset

To actually use a dataset we use the function `useDataset` to update our Mart object so it contains the configuration information of the dataset of interest.

```
> mart <- useDataset(dataset = "hsapiens_gene_ensembl", mart = mart)

Checking attributes and filters ... ok
```

4.5 Filter, Values and Attributes

In BioMart, a filter is used to search a set of attributes that have a specified value for that filter. To explain this better lets consider the following use case. We want to get the gene symbol, chromosome name and band of the following features on the affy hgu95av2 chip: 1939_at,2082_s_at and 1454_at. In this case the attributes are gene symbol, chromosome name and band, they are the information we want to retrieve. The filter is the hgu95av2 chip and as values for this filter we use the affy identifiers we want to retrieve the information from. In BioMart a list of possible attributes that we can query for can be retrieved by using the function `listAttributes`

```
> attributes <- listAttributes(mart)
> attributes[1:10]
```

```
 [1] "adf_embl"       "adf_entrezgene" "adf_go"         "adf_pdb"
 [5] "adf_refseq"     "adf_swall"      "adf_swissprot"  "affy_hcg110"
 [9] "affy_hg_focus"  "affy_hg_u133a"
```

Similarly a list of possible filters can be obtained with the function `listFilters`.

```
> filters <- listFilters(mart)
> filters[1:10]
```

```
 [1] "affy_hc_g110"        "affy_hg_focus"       "affy_hg_u133a"
 [4] "affy_hg_u133a_2"     "affy_hg_u133b"       "affy_hg_u133_plus_2"
 [7] "affy_hg_u95av2"      "affy_hg_u95b"        "affy_hg_u95c"
[10] "affy_hg_u95d"
```

To get the information from our example we can use the function `getBM`, using valid attributes and filter.

```
> getBM(attributes = c("affy_hg_u95av2", "hgnc_symbol"), filter = "affy_hg_u95av2",
+     values = c("1939_at", "1000_at"), mart = mart)
```

```
  affy_hg_u95av2 hgnc_symbol
1        1000_at       MAPK3
2        1000_at
3        1939_at        TP53
```

As you see multiple attributes can be retrieved at once but in the current version of biomaRt there is the restriction that the attributes which are queried together, should somehow be of a similar type, e.g. chromosome band and chromosome name or e.g. allele, SNP, and frequency of snp.

5 Local BioMart databases

The biomaRt package can be used with a local install of a public BioMart database or a locally developed BioMart database. In order for biomaRt to recognize the database as a BioMart, make sure that the local database you create has a name conform with

```
database_mart_version
```

where database is the name of the database and version is a version number. No more underscores than the ones showed should be present in this name. A possible name is for example

```
ensemblLocal_mart_36
```

. For more information on how to install a public BioMart database see: http://www.biomart.org/install.htm and follow link databases.