

# annotate

October 5, 2010

---

ACCNUMStats	<i>Provides statistics on the types of ids used for the ACCNUM environment of a given data package</i>
-------------	--

---

## Description

Given a data package name, ACCNUMStats counts how many of the probe ids are mapped to GenBank Accession numbers, UniGene ids, RefSeq ids, or Image clone ids.

## Usage

```
ACCNUMStats(pkgName)
whatACC(accs)
```

## Arguments

pkgName	pkgName a character string for the name of a BioC data package
accs	accs a vector of character string for the ids whose type will be determined

## Details

The ACCNUM environment of each BioC data package contains mappings between probe ids and a set of public ids based on which mappings of probe ids to other annotation data can be obtained using public data sources. The set of ids were provided by a manufacturer or user at the time when the data package was built. The manufacturer/user provided ids can be of different types of public ids, such as GenBank Accession number, UniGene ids, etc..

ACCNUMStats counts the number of probes that are mapped to different types of public ids and have the results presented in a table.

## Author(s)

Jianhua Zhang

## References

The ACCNUM environment of a platform dependent BioC data package

## Examples

```
library("hgu95av2.db")
ACCNUMStats("hgu95av2")
```

---

GO2heatmap	<i>Compute a heatmap for the specified data, for either a GO category or a KEGG pathway.</i>
------------	--

---

## Description

For a given GO category or KEGG pathway, all probes in the supplied data are mapped to the pathway and a heatmap is produced.

## Usage

```
GO2heatmap(x, eset, data, ...)  
KEGG2heatmap(x, eset, data, ...)
```

## Arguments

x	The name of the category or pathway.
eset	An ExpressionSet providing the data.
data	The name of the chip.
...	Additional parameters to pass to heatmap.

## Details

For the given pathway or GO category all matching probes are determined, these are used to subset the data and heatmap is invoked on that set of data. Extra parameters can be passed through to heatmap using the ... parameter. The annotation slot of the eset argument is used to determine the appropriate annotation data to use.

## Value

The value returned by heatmap is passed back to the user.

## Author(s)

R. Gentleman

## See Also

[heatmap](#)

## Examples

```
library("hgu95av2.db")  
data(sample.ExpressionSet)  
KEGG2heatmap("04810", sample.ExpressionSet, "hgu95av2")
```

---

`G0mnplot`*A function to plot by group means against each other.*

---

**Description**

For a two sample comparison, as determined by `group`, and a specified KEGG pathway or GO category, per group means are computed and plotted against each other.

**Usage**

```
G0mnplot(x, eset, data = "hgu133plus2", group, ...)  
KEGGmnplot(x, eset, data = "hgu133plus2", group, ...)
```

**Arguments**

<code>x</code>	The name of the KEGG pathway or GO category.
<code>eset</code>	An <code>ExpressionSet</code> containing the data.
<code>data</code>	The name of the chip that was used to provide the data.
<code>group</code>	The variable indicating group membership, should have two different values.
<code>...</code>	Extra parameters to pass to the call to <code>plot</code> .

**Details**

All probes in `eset` that map to the given category are determined. Then per group, per probe means are computed and plotted against each other. Extra parameters can be passed to the plot function via the `dots` argument.

**Value**

The matrix of per group means, for each probe.

**Author(s)**

R. Gentleman

**See Also**

[KEGG2heatmap](#)

**Examples**

```
library("hgu95av2.db")  
data(sample.ExpressionSet)  
KEGGmnplot("04810", sample.ExpressionSet, sample.ExpressionSet$sex,  
           data = "hgu95av2")
```

---

`HTMLPage-class`*Classes to represent HTML pages*

---

### Description

Class `HTMLPage` and `FramedHTMLPage` are a pair of experimental classes used to explore concepts of representing HTML pages using S4 objects.

### Slots

**fileName:** Object of class "character" The filename of the HTML page  
**pageText:** Object of class "character" The text of the HTML page  
**pageTitle:** Object of class "character" The title of the HTML page  
**topPage:** Object of class "HTMLPage" The header page for a `FramedHTMLPage`  
**sidePage:** Object of class "HTMLPage" The side index page for a `FramedHTMLPage`  
**mainPage:** Object of class "HTMLPage" The primary page for a `FramedHTMLPage`

### Methods

**show** signature(object = "HTMLPage"): Describes information about the page  
**fileName** signature(object = "HTMLPage"): Gets the `fileName` slot  
**pageText** signature(object = "HTMLPage"): Gets the `pageText` slot  
**pageTitle** signature(object = "HTMLPage"): Gets the `pageTitle` slot  
**toFile** signature(object = "HTMLPage"): Writes the page out to the file designated by the `fileName` slot

### Note

These classes are currently experimental.

`FramedHTMLPage` is modeled after the framing layout of the Bioconductor website ([www.bioconductor.org](http://www.bioconductor.org)).

### Author(s)

Jeff Gentry

### Examples

```
##----- Should be DIRECTLY executable !! -----
```

---

 LL2homology

*Functions that find the homology data for a given set of LocusLink ids or HomoloGeneIDs*


---

### Description

Given a set of LocusLink ids or NCBI HomoloGeneIDs, the functions obtain the homology data and represent them as a list of sub-lists using the homology data package for the organism of interest. A sub-list can be of length 1 or greater depending on whether a LocusLink id can be mapped to one or more HomoloGeneIDs.

### Usage

```
LL2homology(homoPkg, llids)
HGID2homology(hgid, homoPkg)
ACC2homology(accs, homoPkg)
```

### Arguments

llids	llids a vector of character strings or numeric numbers for a set of LocusLink ids whose homologous genes in other organisms are to be found
hgid	hgid a named vector of character strings or numeric numbers for a set of HomoloGeneIDs whose homologous genes in other organisms are to be found. Names of the vector give the code used by NCBI for organisms
accs	accs a vector of character strings for a set of GenBank Accession numbers
homoPkg	homoPkg a character string for the name of the homology data package for a given organism, which is a short version of the scientific name of the organism plus homology (e. g. hsahomology)

### Details

The homology data package has to be installed before executing any of the two functions.

Each sub-list has the following elements:

homoOrg - a named vector of a single character string whose value is the scientific name of the organism and name the numeric code used by NCBI for the organism.

homoLL - an integer for LocusLink id.

homoHGID - an integer for internal HomoloGeneID.

homoACC - a character string for GenBank accession number of the best matching sequence of the organism.

homoType - a single letter for the type of similarity measurement between the homologous genes. homoType can be either B (reciprocal best best between three or more organisms), b (reciprocal best match between two organisms), or c (curated homology relationship between two organisms).

homoPS - a percentage value measured as the percent of identity of base pair alignment between the homologous sequences.

homoURL - a url to the source if the homology relationship is a curated orthology.

Sub-lists with homoType = B or b will not have any value for homoURL and objects with homoType = c will not have any value for homoPS.

**Value**

Both functions returns a list of sub-lists containing data for homologous genes in other organisms.

**Author(s)**

Jianhua Zhang

**References**

<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?=homologene>

**Examples**

```
if(require("hsahomology")){
  llids <- ls(env = hsahomologyLL2HGID)[2:5]
  LL2homology("hsahomology", llids)
}
```

---

PMIDAmat

*A function to compute the probe to PubMed id incidence matrix.*

---

**Description**

For a given chip or a given set of genes, it computes the mapping from probes to PubMed id.

**Usage**

```
PMIDAmat(pkg, gene=NULL)
```

**Arguments**

pkg	The package name of the chip for which the incidence matrix should be computed.
gene	A character vector of interested probe set ids or NULL (default).

**Details**

Not much to say, just find which probes are associated with which PubMed ids and return the incidence matrix, with PubMed ids as rows and probes as columns.

To specify a set of probes to use, let the argument `gene` to be a vector of probe ids. Bt this way, the calculations are not involved with non-interested genes/PubMed ids so that the whole process could finish soon.

**Value**

A matrix containing zero or one, depending on whether the probe (column) is associated with a PubMed id (row).

**Author(s)**

R. Gentleman

## Examples

```
library("hgu95av2.db")
probe <- names(as.list(hgu95av2ACCNUM))
Amat <- PMIDAmat("hgu95av2", gene=sample(probe, 10))
```

---

PWAmat

*A function to compute the probe to KEGG pathway incidence matrix.*

---

## Description

For a given chip we compute the mapping from probes to KEGG pathways.

## Usage

```
PWAmat(data)
```

## Arguments

`data`            The name of the chip for which the incidence matrix should be computed.

## Details

Not much to say, just find which probes are in which pathways and return the incidence matrix, with pathways as rows and probes as columns.

It would be nice to be able to specify a set of probes to use, so that one does not do perform the calculations using all probes if they are not of interest.

## Value

A matrix containing zero or one, depending on whether the probe (row) is in a pathway (column).

## Author(s)

R. Gentleman

## See Also

[KEGG2heatmap](#), [GOMnplot](#)

## Examples

```
library("hgu95av2.db")
Am1 <- PWAmat("hgu95av2")
```

**Description**

Given a set of UniGene identifiers this function creates a set of URLs that can be used to either open a browser to the requested location or that can be used as anchors in the construction of HTML output.

**Usage**

```
UniGeneQuery(query, UGaddress="UniGene/", type="CID")
```

**Arguments**

query	The UniGene identifiers.
UGaddress	The address of UniGene, within the NCBI repository.
type	What type of object is being asked for; either CID or UGID

**Details**

Using published details from NCBI we construct an appropriate string for directing a web browser to the information available at the NCBI for that genomic product (usually an EST).

**Value**

A character vector containing the query string.

**Note**

Be very careful about automatically querying this resource. It is considered antisocial behavior by the owners.

**Author(s)**

Robert Gentleman

**References**

NCBI, <http://www.ncbi.nih.gov/>

**Examples**

```
q1<-UniGeneQuery(c("Hs.293970", "Hs.155650"))
q1
if( interactive())
  browseURL(q1[1])
```



---

accessionToUID      *A function to convert accession values to NCBI UIDs.*

---

## Description

Given one or more accession values, this function will attempt to convert them into NCBI UID values.

## Usage

```
accessionToUID(..., db=c("genbank", "pubmed"))
```

## Arguments

...                  Accession numbers to be transformed.  
db                    Which database this accession number refers to, defaults to Genbank

## Details

Utilizes the PubMed tool pmqty.cgi to convert an accession number into a valid NCBI UID number.

**WARNING:** The powers that be at NCBI have been known to ban the IP addresses of users who abuse their servers (currently defined as less than 2 seconds between queries). Do NOT put this function in a type loop or you may find your access revoked.

## Value

Returns either a valid NCBI UID value or NULL (if there was nothing available).

## Author(s)

Jeff Gentry

## See Also

[pubmed](#), [xmlTreeParse](#)

## Examples

```
## The two returns from genbank should be the same
xdoc <- genbank("U03397", type="accession", disp="data")
x <- accessionToUID("U03397", db="genbank")
xdoc <- genbank(x, type="uid", disp="data")

## Can handle multiple inputs
y <- accessionToUID("M16653", "U892893", db="pubmed")
```

---

annPkgName	<i>Get annotation package name from chip name</i>
------------	---

---

### Description

This function returns the name of the Bioconductor annotation data package that corresponds to the specified chip or genome. The `type` argument is used to request an annotation package with a particular backing store.

### Usage

```
annPkgName(name, type = c("db", "env"))
```

### Arguments

name	string specifying the name of the chip or genome. For example, "hgu133plus2"
type	Either "db" or "env". This will determine whether the package name returned corresponds to the SQLite-based annotation package or environment-based package, respectively.

### Value

a string giving the name of the annotation data package

### Author(s)

Seth Falcon

### See Also

[getAnnMap](#)

### Examples

```
annPkgName("hgu133plus2", type="db")
annPkgName("hgu133plus2", type="env")
```

---

aqListGOIDs	<i>List GO Identifiers by GO Ontology</i>
-------------	---

---

### Description

This function returns a character vector of all GO identifiers in the specified ontologies: Biological Process (BP), Cellular Component (CC), Molecular Function (MF).

### Usage

```
aqListGOIDs(ont)
```

**Arguments**

`ont` A character vector specifying the two-letter codes of the ontologies from which all GO IDs will be retrieved. Entries must be one of "BP", "CC", or "MF".

**Value**

A character vector of GO IDs. The vector will contain all GO IDs in the GO ontologies specified by the `ont` argument.

**Author(s)**

Seth Falcon

**Examples**

```
## all GO IDs in BP
bp_ids = aqListGOIDs("BP")
length(bp_ids)

## all GO IDs in BP or CC
bp_or_cc_ids = aqListGOIDs(c("BP", "CC"))
length(bp_or_cc_ids)
```

---

`buildChromLocation` *A function to generate an instantiation of a `chromLocation` class*

---

**Description**

This function will take the name of a data package and build a `chromLocation` object representing that data set.

**Usage**

```
buildChromLocation(dataPkg)
```

**Arguments**

`dataPkg` The name of the data package to be used

**Details**

The requested data set must be available in the user's `.libPaths()`, and the function will throw an error if this is not the case.

If the data package is present, the necessary information will be extracted from the data package and a `chromLocation` object will be created.

**Value**

A `chromLocation` object representing the specified data set.

**Author(s)**

Jeff Gentry

## Examples

```
library("hgu95av2.db")
z <- buildChromLocation("hgu95av2")
```

---

buildPubMedAbst      *A function to generate an instantiation of a PubMedAbst class*

---

## Description

This function will take in a XML tree object and will create an instance of a PubMedAbst class. This instance is returned to the caller.

## Usage

```
buildPubMedAbst(xml)
```

## Arguments

xml                      A XMLTree object that corresponds to a Pubmed abstract.

## Value

This function returns an instantiation of a PubMedAbst object to the caller.

## Author(s)

Jeff Gentry

## See Also

[pubmed.genbank](#)

## Examples

```
x <- pubmed("9695952", "8325638", "8422497")
a <- xmlRoot(x)
numAbst <- length(xmlChildren(a))
absts <- list()
for (i in 1:numAbst) {
  absts[[i]] <- buildPubMedAbst(a[[i]])
}
```

---

`chrCats`*Returns a list of chromosome locations from a MAP environment*

---

## Description

The `chrCats` function takes a data package that contains a MAP environment and returns a list that contains the locations for each gene (from the chromosome number to more specific locations if they're available). For example, the `hgu95av2MAP` environment gives the location, `14q22-q23`, for Affymetrix identifier: `1114\_at`. This function will return a list with one named element for `1114\_at` and the values it will contain are `14`, `14q`, `14q2`, `14q22`, and `14q23` since the Affy id is located at each of those chromosome locations.

## Usage

```
chrCats(data)
createMAPIncMat(data)
createLLChrCats(data)
```

## Arguments

`data`                    the data package (a character string)

## Details

This function does a lot of string manipulation and there are a few known errors so I want to discuss them here in case someone else would like to improve on this function.

The first thing, `chrCats`, does is only allow one location for each Affymetrix identifier. If the MAP environment has more than one location for an Affy id, then the first location is taken. Currently, the `hgu95av2MAP` environment has only 9 Affy ids (out of 12625) that have more than one location and the `hgu133aMAP` environment has only 16 Affy ids (out of 22283) that have more than one location so this does not affect many identifiers.

Next any spaces are removed from each location as several locations have leading spaces.

Then a for loop (which is not efficient!) is used to look at each location individually and make a list that will be returned. A few particular strings are looked for in each location and these include `'|'` and `'-'`.

Locations that include `'|'` in the string are split based on the `'|'` as though it represents OR. For example, for Affy id, `32273\_at`, in `hgu95av2MAP` the location is given as `5q33|5q31.1` and this function assumes this means `5q33` or `5q31.1` so it will return the values `5`, `5q`, `5q3`, `5q33`, `5q31`, and `5q31.1` for this Affy id.

The `'-'` character is assumed to mean BETWEEN. For example, for Affy id, `1138\_at`, in `hgu95av2MAP` the location is given as `2q11-q14` and this function assumes this means the location is somewhere between `2q11` and `2q14` so it will return the values `2`, `2q`, `2q1`, `2q11`, `2q12`, `2q13`, and `2q14` for this Affy id.

Now here is the first problem with this function. I do not know how to handle the `'-'` when the two strings are not of equal length. For example, for Affy id, `36779\_at`, in `hgu95av2MAP` the location is given as `5q33.3-q34`, but I do not know how to treat this BETWEEN because I do not know how many sub-bands there are between `5q33.3` and `5q34`. Is there a `5q33.4` or `5q33.5`, etc.? I'm not sure. So I treat this `'-'` as an `'|'`. This function will return the values `5`, `5q`, `5q3`, `5q33`, `5q33.3`, and `5q34` for this Affy id and most likely, that is incorrect.

Another problem I have with the '-' occurs when all of the characters up until the last character do not match. For example, for Affy id, 38927\_i\_at, in hgu95av2MAP the location is given as 11q14-q21, but again I'm not sure how to treat this BETWEEN because I don't know the number of sub-bands between 11q14 and 11q21. Does 11q15 exist, etc.? So I again treat this '-' as an '|'. This function will return the values 11, 11q, 11q1, 11q14, 11q2, and 11q21 for this Affy id and this is probably incorrect.

The problem with '-' also occurs when the location is something like 19cen-q13.1 for Affy id, 34670\_at, in hgu95av2MAP. Again I don't know the number of sub-bands between 19cen and 19q13.1 so I treat this BETWEEN as an OR.

Another problem I have with 'cen' in the location is that sometimes the location looks like: 19p13.2-cen and very rarely it looks like: 5p13.1-5cen. In the second case, the chromosome number is included after the '-' and before the 'cen'. This only occurs with the location 5p13.1-5cen in both hgu95av2MAP and hgu133aMAP and all other locations do not include the chromosome number after the '-'. Currently this function returns the wrong information for that one location. It will return the values 5, 5p, 5p1, 5p13, 5p13.1, 5p5, and 5p5cen, but it should return 5, 5p, 5p1, 5p13, 5p13.1, and 5cen so this one location is an error. All other locations that include 'cen' are correct. For example, this function returns the values 19, 19p, 19p1, 19p13, 19p13.2, and 19cen for the location 19p13.2-cen.

This function is very slow because it contains for loops and thus, it would be useful to make it more efficient. Also, it would be nice at some point for someone with more knowledge on chromosome location figure out how to improve some of my string manipulation errors.

`createLLChrCats` is a wrapper that converts probe IDs to Entrez Gene IDs.

`createMAPIncMat` is a wrapper that calls `createLLChrCats` and then returns an incidence matrix with rows being the categories and cols the Entrez Gene IDs.

## Value

A named list with an element for each Affy id. The name will be the Affy id and the values will be the locations for that Affy id. If the Affy id had a location of NA in the MAP environment, then a list element is not returned for that Affy id.

## Author(s)

Elizabeth Whalen

## Examples

```
library("hgu95av2.db")
mapValues <- chrCats("hgu95av2")
```

---

chromLocation-class

*Class chromLocation, a class for describing genes and their chromosome mappings.*

---

## Description

This class provides chromosomal information provided by a Bioconductor metadata package. By creating the object once for a particular package, it can be used in a variety of locations without the need to recompute values repeatedly.

**Creating Objects**

```

new('chromLocation', organism = ....., # Object of class character
dataSource = ....., # Object of class character
chromLocs = ....., # Object of class list
probesToChrom = ....., # Object of class ANY
chromInfo = ....., # Object of class numeric
geneSymbols = ....., # Object of class ANY
)

```

**Slots**

**organism:** Object of class "character". The organism that these genes correspond to.

**dataSource:** Object of class "character". The source of the gene data.

**chromLocs:** Object of class "list". A list which provides specific location information for every gene.

**probesToChrom:** An object with an environment-like API which will translate a probe identifier to chromosome it belongs to.

**chromInfo:** A numerical vector representing each chromosome, where the names are the names of the chromosomes and the values are their lengths

**geneSymbols:** An environment or an object with environment-like API that maps a probe ID to the appropriate gene symbol

**Methods**

**chromLengths** (chromLocation): Gets the lengths of the chromosome for this organism

**chromLocs** (chromLocation): Gets the 'chromLocs' attribute.

**chromNames** (chromLocation): Gets the name of the chromosomes for this organism

**dataSource** (chromLocation): Gets the 'dataSource' attribute.

**probesToChrom** (chromLocation): Gets the 'probesToChrom' attribute.

**nChrom** (chromLocation): gets the number of chromosomes this organism has

**organism** (chromLocation): gets the 'organism' attribute.

**chromInfo** Gets the 'chromInfo' attribute.

**geneSymbols** Gets the 'geneSymbols' attribute.

**See Also**

[buildChromLocation](#)

**Examples**

```

library("hgu95av2.db")

z <- buildChromLocation("hgu95av2")

## find the number of chromosomes
nChrom(z)

## Find the names of the chromosomes
chromNames(z)

```

```
## get the organism this object refers to
organism(z)

## get the lengths of the chromosomes in this object
chromLengths(z)
```

---

compatibleVersions *function to check to see if the packages represented by the names passed have the same version number*

---

### Description

This function takes the names of installed R packages and then checks to see if they all have the same version number.

### Usage

```
compatibleVersions(...)
```

### Arguments

...                    ... character strings for the names of R packages that have been installed

### Details

If all the package have the same version number, the function returns TRUE. Otherwise, the function returns FALSE

### Value

This function returns TRUE or FALSE depending on whether the packages have the same version number

### Author(s)

Jianhua Zhang

### See Also

[packageDescription](#)

### Examples

```
library("hgu95av2.db")
library("GO.db")
compatibleVersions("hgu95av2.db", "GO.db")
```



---

`dropECode`*Drop GO labels for specified Evidence Codes*

---

### Description

Genes are mapped to GO terms on the basis of evidence codes. In some analyses it will be appropriate to drop certain sets of annotations based on specific evidence codes.

### Usage

```
dropECode(inlist, code="IEA")
```

### Arguments

<code>inlist</code>	A list of GO data
<code>code</code>	The set of codes that should be dropped.

### Details

A simple use of `lapply` and `sapply` to find and eliminate those terms that have the specified evidence codes.

This might be used when one is using to GO to validate a sequence matching experiment (for example), then all terms whose mapping was based on sequence similarity (say ISS and IEA) should be removed.

### Value

A list of the same length as the input list retaining only those annotations whose evidence codes were not the ones in the exclusion set `code`.

### Author(s)

R. Gentleman

### See Also

[getEvidence](#), [getOntology](#)

### Examples

```
library("hgu95av2.db")
bb <- hgu95av2GO[["39613_at"]]
getEvidence(bb[1:3])
cc <- dropECode(bb[1:3])
if (length(cc))
  getEvidence(cc)
```

---

`entrezGeneByID`*Create a Query String for an Entrez Gene Identifier*

---

**Description**

Given a set of UniGene identifiers this function creates a set of URLs that can be used to either open a browser to the requested location or that can be used as anchors in the construction of HTML output.

**Usage**

```
entrezGeneByID(query)
```

**Arguments**

`query` Entrez Gene identifiers.

**Details**

Using NCBI we construct appropriate strings for directing a web browser to the Entrez Genes specified by their IDs.

**Value**

A character vector containing the query string.

**Note**

Be very careful about automatically querying this resource. It is considered antisocial behavior by the owners.

**Author(s)**

Marc Carlson

**References**

NCBI, <http://www.ncbi.nih.gov/>

**Examples**

```
q1<-entrezGeneByID(c("100", "1002"))
q1
if( interactive())
  browseURL(q1[1])
```

---

entrezGeneQuery      *Create a Query String for Entrez Genes*

---

### Description

Given a set of search terms this function creates a set of URLs that can be used to either open a browser to the requested location or that can be used as anchors in the construction of HTML output.

### Usage

```
entrezGeneQuery(query)
```

### Arguments

query                  The UniGene identifiers.

### Details

Using NCBI we construct an appropriate string for directing a web browser to information about genes of that type at NCBI.

### Value

A character vector containing the query string.

### Note

Be very careful about automatically querying this resource. It is considered antisocial behavior by the owners.

### Author(s)

Marc Carlson

### References

NCBI, <http://www.ncbi.nih.gov/>

### Examples

```
q1<-entrezGeneQuery(c("leukemia", "Homo sapiens"))
q1
if( interactive())
  browseURL(q1[1])
```

---

`filterGOByOntology` *Filter GO terms by a specified GO ontology*

---

### Description

Given a character vector containing GO identifiers, return a logical vector indicating which GO IDs are in the specified ontology (BP, CC, or MF).

### Usage

```
filterGOByOntology(goids, ontology = c("BP", "CC", "MF"))
```

### Arguments

<code>goids</code>	a character vector of GO IDs
<code>ontology</code>	One of "BP", "CC", or "MF"

### Value

A logical vector with length equal to `goids`. A TRUE indicates that the corresponding GO ID in `goids` is a member of the ontology specified by `ontology`.

### Author(s)

Seth Falcon

### Examples

```
haveGO <- suppressWarnings(require("GO"))
if (haveGO) {
  ids <- c("GO:0001838", "GO:0001839")
  stopifnot(all(filterGOByOntology(ids, "BP")))
  stopifnot(!any(filterGOByOntology(ids, "MF")))
} else cat("Sorry, this example requires the GO package\n")
```

---

`findNeighbors` *A function to locate neighboring genes within a defined range around a target gene represented by a Entrez Gene ID*

---

### Description

Give a data package with mappings between Entrez Gene IDs and their locations on chromosomes, this function locates genes that are within a defined range on a given chromosome. If a Entrez Gene ID is passed as one of the arguments, genes located will be neighbors to the gene represented by the Entrez Gene ID within a defined range on the chromosome the target gene resides

**Usage**

```

findNeighbors(chrLoc, llID, chromosome, upBase, downBase, mergeOrNot = TRUE)
checkArgs(llID, chromosome, upBase, downBase)
findChr4LL(llID, chrEnv, organism)
getValidChr(organism)
getBoundary(loc, base, lower = TRUE)
weightByConfi(foundLLs)

```

**Arguments**

chrLoc	chrLoc a character string for the name of the data package that contains mappings between Entrez Gene IDs and their locations on chromosomes. For each chromosome, there assumed to be mappings for the start and end locations of genes represented by Entrez Gene IDs. The data package needs to be built using chrLocPkgBuilder of AnnBuilder
llID	llID a character string for the Entrez Gene ID representing a gene whose neighbors are sought. llID can be missing
chromosome	chromosome a character string for the number of the chromosome of interest. chromosome is only required for locating genes within a range on the chromosome
upBase	upBase a numeric or character string for the number of base pairs that defines the upper limit of the range to locate genes. If neighbors of a given gene is sought, the value will be the distance in number of base pairs from the target gene upstream, to which search for genes will be conducted. Otherwise, the value will be the upper limit in number of base pairs from the p arm, to which search for genes will be conducted
downBase	downBase a numeric or character string for the number of base pairs that defines the lower limit of the range to locate gene. If neighbors of a given gene is sought, the value will be the distance in number of base pairs from the target gene downstream, to which search for genes will be conducted. Otherwise, the value will be the lower limit in number of base pairs from the p arm, to which search for genes will be conducted
organism	organism a character string for the name of the organism of interest
chrEnv	chrEnv an environment object with keys for Entrez Gene IDs and values for the chromosomes where genes reside
loc	loc a numeric or character string for the chromosomal location of gene of interest
base	base either a downBase or upBase
lower	lower a boolean indicating whether the lower or upper boundary of search limit is sought
mergeOrNot	mergeOrNot a boolean to indicate whether gene found up and down streams will be merged (TRUE)
foundLLs	foundLLs a vector of character strings for Entrez Gene IDs

**Details**

A chrLoc data package can be created using function chrLocPkgBuilder of AnnBuilder, in which Entrez Gene IDs are mapped to location data on individual chromosomes.

Genes are considered to be neighbors to a given target gene or within a given range when the transcription of genes start and end within the given range.

`findNeighbors`, `checkArgs`, `findChr4LL`, `getValidChr`, and `getBoundary` are accessory functions called by `findNeighbors` and may not have real values outside.

### Value

The function returns a list of named vectors. The length of the list is one when genes in a given region are sought but varies depending on whether a given gene can be mapped to one or more chromosomes when neighboring genes of a target gene are sought. Names of vector can be "Confident" when a gene can be confidently placed on a chromosome or "Unconfident" when a gene can be placed on a chromosome but its exact location can not be determined with great confidence.

### Author(s)

Jianhua Zhang

### References

<http://www.genome.ucsc.edu/goldenPath/>

### Examples

```
if(require("humanCHRLOC")){
  findNeighbors("humanCHRLOC", "51806", 10, upBase = 600000, downBase = 600000)
}else{
  print("Can not find neighbors without the required data package")
}
```

---

genbank

*A function to open the browser to Genbank with the selected gene.*

---

### Description

Given a vector of Genbank accession numbers or NCBI UIDs, the user can either have a browser display a URL showing a Genbank query for those identifiers, or a XMLdoc object with the same data.

### Usage

```
genbank(..., disp=c("data", "browser"), type=c("accession", "uid"),
        pmaddress=.efetch("Nucleotide", disp, type))
```

### Arguments

<code>...</code>	Vectorized set of Genbank accession numbers or NCBI UIDs
<code>disp</code>	Either "Data" or "Browser" (default is data). Data returns a XMLDoc, while Browser will display information in the user's browser.
<code>type</code>	Denotes whether the arguments are accession numbers or UIDs. Defaults to accession values.
<code>pmaddress</code>	Specific path to the pubmed efetch engine from the NCBI website.

## Details

A simple function to retrieve Genbank data given a specific ID, either through XML or through a web browser. This function will accept either Genbank accession numbers or NCBI UIDs (defined as a Pubmed ID or a Medline ID) - although the types must not be mixed in a single call.

WARNING: The powers that be at NCBI have been known to ban the IP addresses of users who abuse their servers (currently defined as less than 2 seconds between queries). Do NOT put this function in a tight loop or you may find your access revoked.

## Value

If the option "data" is used, an object of type XMLDoc is returned, unless there was an error with the query in which case an object of type try-error is returned.

If the option "browser" is used, nothing is returned.

## Author(s)

R. Gentleman

## See Also

[pubmed](#), [xmlTreeParse](#)

## Examples

```
## Use UIDs to get data in both browser & data forms

if ( interactive() ) {
  disp <- c("data","browser")
} else {
  disp <- "data"
}

for (dp in disp)
  genbank("12345", "9997", disp=dp, type="uid")

## Use accession numbers to retrieve browser info
if ( interactive() )
  genbank("U03397", "AF030427", disp="browser")
```

---

genelocator

*A function to identify genes by their LocusLink (or other id).*

---

## Description

This function uses `locator` to provide some interaction with an image plot of genetic data. It is currently not implemented except in skeleton form.

## Usage

```
genelocator(x)
```

**Arguments**

x                      Undetermined

**Value**

This function is executed mainly for its side effect. When an image plot is active then `genelocator` can be called. When the image is clicked on, using the left mouse button, the users web browser is activated and a web page (determined by one of the arguments to `genelocator`) will be displayed.

**Author(s)**

R. Gentleman

**See Also**

[locator](#)

**Examples**

```
## This is an interactive function so the examples won't work!
```

---

getAnnMap

*Get annotation map*

---

**Description**

This function retrieves a map object from an annotation data package. It is intended to serve as a common interface for obtaining map objects from both SQLite-based and environment-based annotation data packages.

**Usage**

```
getAnnMap(map, chip, load = TRUE, type = c("db", "env"))
```

**Arguments**

map	a string specifying the name of the map to retrieve. For example, "ENTREZID" or "GO"
chip	a string describing the chip or genome
load	a logical value. When TRUE, <code>getAnnMap</code> will try to load the annotation data package if it is not already attached.
type	a character vector of one or more annotation data package types. The currently supported types are "db" and "env". If <code>load</code> is TRUE, you can specify both "db" and "env" and the order will determine which type is tried first. This provides a fall-back mechanism when the preferred annotation data package type is not available. If <code>type</code> is missing, then the first matching annotation package found in the search path will be used, and then the default value of <code>type</code> takes over.



**Details**

getAnnMap uses the search path (see `search`) to find an appropriate annotation data package; when called with `chip="hgu95av2"`, the function will use the first hgu95av2 package on the search path whether it be db or environment-based. If `load=TRUE` and no suitable package is found on the search path, then the function will attempt to load an appropriate package. The `type` argument is used to determine which type of package (db or env) is loaded first.

**Value**

If `type` is "db", an S4 object representing the requested map. If `type` is "env", an R environment object representing the requested map.

**Author(s)**

Seth Falcon

**Examples**

```
map <- getAnnMap("ENTREZID", "hgu95av2", load=TRUE, type=c("env", "db"))
class(map)
```

---

getEvidence

*Get the Evidence codes for a set of GO terms.*

---

**Description**

For each mapping of a gene to a GO term there are a set of evidence codes that are used. Genes can be mapped using one, or more evidence codes and this function obtains the evidence codes for all genes provided in the input list.

**Usage**

```
getEvidence(inlist)
```

**Arguments**

`inlist` A list of GO identifiers.

**Value**

A list of the same length as the input list, each element is a vector of evidence codes.

**Author(s)**

R. Gentleman

**See Also**

[getOntology](#), [dropECode](#)

## Examples

```
library("hgu95av2.db")
bb <- hgu95av2GO[["39613_at"]]
getEvidence(bb)
```

---

getGOTerm

*Functions to Access GO data.*

---

## Description

These functions provide access to data in the GO package. The data are assembled from publically available data from the Gene Ontology Consortium (GO), [www.go.org](http://www.go.org). Given a list of GO identifiers they access the children (more specific terms), the parents (less specific terms) and the terms themselves.

## Usage

```
getGOTerm(x)
getGOParents(x)
getGOChildren(x)
getGOOntology(x)
```

## Arguments

`x` A character vector of valid GO identifiers.

## Details

GO consists of three (soon to be more) specific hierarchies: Molecular Function (MF), Biological Process (BP) and Cellular Component (CC). For more details consult the GO website. For each GO identifier each of these three hierarchies is searched and depending on the function called the appropriate values are obtained and returned.

It is possible for a GO identifier to have no children or for it to have no parents. However, it must have a term associated with it.

## Value

A list of the same length as `x`. The list contains one entry for each element of `x`. That entry is itself a list. With one component named `Ontology` which has as its value one of MF, BP or CC. The second component has the same name as the suffix of the call, i.e. Children, Parents, or Term. If there was no match in any of the ontologies then a length zero list is returned for that element of `x`.

For `getGOOntology` a vector of categories (the names of which are the original GO term names). Elements of this list that are `NA` indicate term names for which there is no category (and hence they are not really term names).

## Author(s)

R. Gentleman

## References

The Gene Ontology Consortium

**Examples**

```
library("GO.db")

sG <- c("GO:0005515", "GO:0000123", "GO:0000124", "GO:0000125",
        "GO:0000126", "GO:0020033", "GO:0006830",
        "GO:0015916")

gT <- getGOTerm(sG)
gP <- getGOParents(sG)
gC <- getGOChildren(sG)
gcat <- getGOOntology(sG)
```

---

getOntology	<i>Get GO terms for a specified ontology</i>
-------------	--

---

**Description**

Find the subset of GO terms for the specified ontology, for each element of the supplied list of associations. The input list is typically from one of the chip-specific meta-data files.

**Usage**

```
getOntology(inlist, ontology=c("MF", "BP", "CC"))
```

**Arguments**

<code>inlist</code>	A list of GO associations
<code>ontology</code>	The name of the ontology you want returned.

**Details**

The input list should be a list of lists, each element of `inlist` is itself a list containing the information that maps from a specified ID (usually `LocusLink`) to GO information. Each element of the inner list is a list with elements `GOID`, `Ontology` and `Evidence`.

**Value**

A list of the same length as the input list. Each element of this list will contain a vector of GOIDs for those terms that match the requested ontology.

**Author(s)**

R. Gentleman

**See Also**

[getEvidence](#), [dropECode](#)

**Examples**

```
library("hgu95av2.db")
bb <- hgu95av2GO[["39613_at"]]
getOntology(bb)
sapply(bb, function(x) x$Ontology)
```

---

getPMInfo	<i>extract publication details and abstract from annotate::pubmed function output</i>
-----------	---

---

**Description**

extract publication details and abstract from annotate::pubmed function output

**Usage**

```
getPMInfo(x)
```

**Arguments**

`x` an object of class `xmlDocument`; assumed to be result of a `pubmed()` call

**Details**

uses `xmlDOMApply` to extract and structure key features of the XML tree returned by `annotate::pubmed()`

**Value**

a list with one element per pubmed id processed by `pubmed`. Each element of the list is in turn a list with elements for author list, title, journal info, and abstract text.

**Note**

this should be turned into a method returning an instance of a formal class representing articles.

**Author(s)**

Vince Carey <stvjc@channing.harvard.edu>

**Examples**

```
demo <- pubmed("11780146",
              "11886385", "11884611")
getPMInfo(demo)
```

## Description

The functions documented here are intended to make it easier to map from a set of manufacturers identifiers (such as you will get from the chips etc) to other identifiers.

## Usage

```
getSYMBOL(x, data)
getLL(x, data)
getEG(x, data)
getGO(x, data)
getPMID(x, data)
getGODesc(x, which)
lookUp(x, data, what, load = FALSE)
getUniqAnnItem()
```

## Arguments

x	The identifiers to be mapped (usually manufacturer)
data	The basename of the meta-data package to be used.
what	what a character string for the name of an annotation element of an annotation data package
which	which a character string in the form of MF, BP, CC, or ANY to indicated the GO categories of interest
load	A logical value indicating whether to attempt to load the required annotation data package if it isn't already loaded.

## Details

Users must supply the basename of the meta-data package that they want to use to provide the mappings. The name of the meta-data package is the same as the basename.

Appropriate translations are done. In some cases such as `getEG` and `getSYMBOL` there will only be one match and a vector is returned. In other cases such as `getPMID` and `getGO` there may be multiple matches and a list is returned.

For `getGODesc` `x` contains GO identifiers (not manufacturer identifiers) and the output is a list of `GOTerms` objects, if `which` specifies some subset of the ontologies (MF, BP or CC) then only terms for that ontology are retained.

`lookUp` is a general function that can be used to look up matches. All other translation functions use `lookUp`

A BioC annotation data package contains annotation data environments whose names are package name (e. g. `hgu95av2`) + element name (e. g. `PMID`). `what` must be one of the element names for the given data package.

`getUniqAnnItem` keeps track of the annotation elements that have one to one mappings.

**Value**

Either a vector or a list depending on whether multiple values per input are possible.

**Author(s)**

R. Gentleman

**See Also**

[mget](#)

**Examples**

```
library("hgu95av2.db")
library("GO.db")

data(sample.ExpressionSet)
gN <- featureNames(sample.ExpressionSet)[100:105]
lookUp(gN, "hgu95av2", "SYMBOL")

# Same as lookUp for SYMBOL except the return is a vector
getSYMBOL(gN, "hgu95av2" )
gg <- getGO(gN, "hgu95av2")
lookUp(gg[[2]][[1]][["GOID"]], "GO", "TERM")

# Same as lookUp for TERM
getGODesc(gg[[2]][[1]][["GOID"]], "ANY")

# For BP only
getGODesc(gg[[2]][[1]][["GOID"]], "BP")
getEG(gN, "hgu95av2")
getPMID(gN, "hgu95av2")
```

---

getSEQ

*Queries the NCBI database to obtain the sequence for a given GenBank Accession number*

---

**Description**

Given a GenBank Accession number, getSEQ queries the NCBI database for the nucleotide sequence.

**Usage**

```
getGI(accNum)
getSEQ(gi)
```

**Arguments**

`accNum` `accNum` a character string for a GenBank Accession number (i.e. M22490)  
`gi` `gi` a character string or numeric numbers for a GenBank accession number or gi number. A gi number is a series of digits that are assigned consecutively to each sequence record processed by NCBI

**Details**

The NCBI database is queried for the given GenBank Accession number to obtain the nucleotide sequence in FASTA format. The leading identification line of the sequence data is then dropped to return only the nucleotide sequence.

getGI returns the gi number corresponding to a given GenBank accession number.

**Value**

getSEQ returns a character string of nucleotide sequence

**Author(s)**

Jianhua Zhang

**References**

<http://www.ncbi.nlm.nih.gov/entrez/query.fcgi>

**Examples**

```
getSEQ("M22490")
```

---

getQueryLink	<i>Functions to create hypertext links that can be placed in a table cell of a HTML file</i>
--------------	--

---

**Description**

Given a vector of ids, the functions will create a vector of hypertext links to a defined public repositories such as LocusLink, UniGene .... The linkages can be placed in a html file constructed by [htmlpage](#).

**Usage**

```
getQueryLink(ids, repository = "ug", ...)
getTDRRows(ids, repository = "ug", ...)
getCells(ids, repository = "ug", ...)
getQuery4LL(ids, ...)
getQuery4UG(ids, ...)
getQuery4SP(ids, ...)
getQuery4GB(ids, ...)
getQuery4OMIM(ids, ...)
getQuery4Affy(ids, ...)
getQuery4FB(ids, ...)
getQuery4EN(ids, ...)
getQuery4TR(ids, ...)
getQuery4ENSEMBL(ids, ...)
```

**Arguments**

<code>ids</code>	A character vector of ids, or alternatively, a list containing character vectors of ids. These will be used to construct hypertext links. A list should be used in cases where there are multiple ids per gene.
<code>repository</code>	A character string for the name of a public repository. Valid values include "ll", "ug", "gb", "sp", "omim", "affy", "en", and "fb". See the details section for more information.
<code>...</code>	Allows end user to pass additional arguments. See details for <a href="#">getQuery4ENSEMBL</a> for more information.

**Details**

[getQuery4LL](#) constructs hypertext links to LocusLink using the provided ids.

[getQuery4GB](#) constructs hypertext links to GenBank using the provided ids.

[getQuery4UG](#) constructs hypertext links to UniGene using the provided ids.

[getQuery4Affy](#) constructs hypertext links to Affymetrix using the provided ids.

[getQuery4SP](#) constructs hypertext links to SwissProt using the provided ids.

[getQuery4OMIM](#) constructs hypertext links to OMIM using the provided ids.

[getQuery4FB](#) constructs hypertext links to FlyBase using the provided ids.

[getQuery4EN](#) constructs hypertext links to EntrezGene using the provided ids. This supercedes [getQuery4LL](#), as LocusLink has been deprecated by NCBI.

[getQuery4TR](#) constructs hypertext links to TAIR using the provided ids.

[getQuery4ENSEMBL](#) constructs hypertext links to Ensembl using the provided ids. An additional 'species' argument must be passed to this function via the `...` argument to `htmlpage`. The form of the argument must be e.g., `species="Homo_sapiens"` for human. Note the capitalized genus and underscore (`_`) separator.

[getQueryLink](#) directs calls to construct hypertext links using the provided ids.

[getQueryTDRows](#) constructs each row of the resulting table.

[getQueryCells](#) constructs each cell of the resulting table.

Note that some of these functions ([getQuery4OMIM](#), [getQuery4UG](#), [getQuery4LL](#)), [getQuery4FB](#) attempt to return empty cells for ids that don't make sense, rather than broken links. For the other [getQuery4XX](#) functions, the end user must replace all nonsense ids with "&nbsp;" in order to have an empty cell.

Also note that creating additional links is quite simple. First, define a new '[getQuery4XX\(\)](#)' function modeled on the existing functions, then add this function to the [getQueryLink](#) function.

**Value**

Returns a vector of character strings representing the hypertext links.

**Author(s)**

Jianhua Zhang <jzhang@jimmy.harvard.edu> with further modifications by James W. MacDonald <jmacdon@med.umich.edu>



---

hasGOannotate	<i>Check for GO annotation</i>
---------------	--------------------------------

---

### Description

Given a GO term, or a vector of GO terms and an ontology this function determines which of the terms have GO annotation in the specified ontology.

### Usage

```
hasGOannotate(x, which="MF")
```

### Arguments

x	A character vector, an instance of the <code>GOTerms</code> class or a list of <code>GOTerms</code> .
which	One of "MF", "BP" or "CC"

### Details

The available GO annotation is searched and a determination of whether a specific GO identifier has a value in the specified ontology is made.

### Value

A logical vector of the same length as `x`.

### Author(s)

R. Gentleman

### See Also

[get](#)

### Examples

```
library("GO.db")
t1 <- "GO:0003680"
hasGOannotate(t1)
hasGOannotate(t1, "BP")
```

---

hgByChroms	<i>A dataset to show the human genome base pair locations per chromosome.</i>
------------	---

---

**Description**

The data is described above.

**Usage**

```
data(hgByChroms)
```

**Format**

A list, with the names consisting of the names of the chromosomes in the human genome (thus 24 elements). Each element consists of a named vector of +/- values - where each value represents the location of a base pair (the numeric value is the location, while the +/- denotes the strand value), with the name providing the name of the base pair.

**Source**

Cheng Li of the Dana-Farber Cancer Institute.

**Examples**

```
data(hgByChroms)
```

---

hgCLengths	<i>A dataset which contains the lengths (in base pairs) of the human chromosomes.</i>
------------	---

---

**Description**

The data is described above.

**Usage**

```
data(hgCLengths)
```

**Format**

A vector containing 24 values, each corresponding to the total chromosome length.

**Source**

UCSC Human Genome Project

**Examples**

```
data(hgCLengths)
```

---

hgu95AProbLocs	<i>chromLocation instance hgu95AProbLocs, an example of a chromLocation object</i>
----------------	--

---

**Description**

gives chromosome locations for Affy U95 probes

**Slots**

species: Object of class character, value: 'Human'  
 datSource: Object of class character, value  
 nChrom: Object of class numeric, value: 24  
 chromNames: Object of class character, value: 1:22, X,Y  
 chromLocs: Object of class list, value: long: sense and antisense locations associated with affy identifiers  
 chromLengths: Object of class numeric,  
 geneToChrom: Object of class environment  
 class: Object of class character, value: 'chromLocation'

---

hgu95Achroloc	<i>Annotation data for the Affymetrix HGU95A GeneChip</i>
---------------	---

---

**Description**

Data, in the form of environments for the Affymetrix U95A chip.

**Usage**

```
data(hgu95Achroloc)
```

**Format**

These data sets provide environments with mappings from the Affymetrix identifiers to chromosomal location, in bases. The environments function like hashtables and can be accessed using `mget`. If the returned value is NA then the current mapping was unable to identify this. Mappings and data sources are constantly evolving so updating often is recommended.

**Source**

The `AnnBuilder` package.

**Examples**

```
data(hgu95Achroloc)
data(sample.ExpressionSet)
mget(featureNames(sample.ExpressionSet)[330:340], env=hgu95Achroloc,
      ifnotfound=NA)
```

---

`hgu95Achrom`*Annotation data for the Affymetrix HGU95A GeneChip*

---

**Description**

Data, in the form of environments for the Affymetrix U95A chip.

**Usage**

```
data(hgu95Achrom)
```

**Format**

This data set provides an environment (treat as a hashtable) with mappings from the Affymetrix identifiers to chromosome number/name. The environment functions like a hashtable and can be accessed using `mget`. If the returned value is `NA` then the current mapping was unable to identify this. Mappings and data sources are constantly evolving so updating often is recommended.

**Source**

The `AnnBuilder` package.

**Examples**

```
data(hgu95Achrom)
data(sample.ExpressionSet)
mget(featureNames(sample.ExpressionSet)[330:340], env=hgu95Achrom, ifnotfound=NA)
```

---

`hgu95All`*Annotation data for the Affymetrix HGU95A GeneChip*

---

**Description**

Data, in the form of environments for the Affymetrix U95A chip.

**Usage**

```
data(hgu95All)
```

**Format**

These data sets provide environments with mappings from the Affymetrix identifiers to Entrez Gene identifiers. The environment functions like a hashtable and can be accessed using `mget`. If the returned value is `NA` then the current mapping was unable to identify this. Mappings and data sources are constantly evolving so updating often is recommended.

**Source**

The `AnnBuilder` package.

**Examples**

```
data(hgu95A11)
data(sample.ExpressionSet)
mget(featureNames(sample.ExpressionSet)[330:340], env=hgu95A11, ifnotfound=NA)
```

---

hgu95Asym

*Annotation data for the Affymetrix HGU95A GeneChip*


---

**Description**

Data, in the form of environments for the Affymetrix U95A chip.

**Usage**

```
data(hgu95Asym)
```

**Format**

This data set provides an environment with mappings from the Affymetrix identifiers to gene symbol. The environment functions like a hashtables and can be accessed using `mget`. If the returned value is `NA` then the current mapping was unable to identify this. Mappings and data sources are constantly evolving so updating often is recommended.

**Source**

The `AnnBuilder` package.

**Examples**

```
data(hgu95Asym)
data(sample.ExpressionSet)
mget(featureNames(sample.ExpressionSet)[330:340], env=hgu95Asym, ifnotfound=NA)
```

---

homoData-class

*Class "homoData"*


---

**Description**

A class to present data for HomologGene data of a matching sequence

**Objects from the Class**

Objects can be created by calls of the form `new("homoData", ...)`.

**Slots**

**homoOrg:** Object of class "character" the scientific name of the organism of interest  
**homoLL:** Object of class "numeric" the LocusLink id of the gene of interest  
**homoType:** Object of class "character" the type of similarity. Valid values include B - a reciprocal best best between 3 or more organisms, b - a reciprocal best match, and c - a curated homology relationship  
**homoPS:** Object of class "numeric" percent similarity value  
**homoURL:** Object of class "character" the URL for curated homology relationship  
**homoACC:** Object of class "character" the accession number  
**homoHGID:** Object of class "numeric" the internal HomologGeneID

**Methods**

**homoPS** signature(object = "homoData"): the get function for slot homoPS  
**homoLL** signature(object = "homoData"): the get function for slot homoLL  
**homoOrg** signature(object = "homoData"): the get function for slot homoOrg  
**homoType** signature(object = "homoData"): the get function for slot homoType  
**homoURL** signature(object = "homoData"): the get function for slot homoURL  
**homoACC** signature(object = "homoData"): the get function for slot homoACC  
**homoHGID** signature(object = "homoHGID"): the get function for slot homoHGID

**Author(s)**

Jianhua Zhang

**References**

<ftp://ftp.ncbi.nih.gov/pub/HomoloGene/README>

**Examples**

```
new("homoData", homoPS = 82.3, homoLL = 2324853, homoOrg = "Homo sapiens",  
homoType = "B", homoURL = "", homoHGID = 12345)
```

---

htmlpage

*Functions to build HTML pages*

---

**Description**

This function is designed to create an HTML table containing both static information as well as links to various online annotation sources.

**Usage**

```
htmlpage(genelist, filename, title, othernames, table.head,  
table.center = TRUE, repository = list("en"), ...)
```

**Arguments**

<code>genelist</code>	A list or <code>data.frame</code> of character vectors containing ids to be made into hypertext links. See details for more information.
<code>filename</code>	A filename for the resultant HTML table.
<code>title</code>	A title for the table.
<code>othernames</code>	A list or <code>data.frame</code> of other things to add to the table. These will not be hyperlinks. The list of othernames can contain vectors, matrices, <code>data.frames</code> or lists.
<code>table.head</code>	A character vector of column headers for the table.
<code>table.center</code>	Center the table? Defaults to <code>TRUE</code> .
<code>repository</code>	A list of repositories to use for creating the hypertext links. Currently available repositories include 'gb' (GenBank), 'en' (EntrezGene), 'omim' (Online Mendelian Inheritance in Man), 'sp' (SwissProt), 'affy' (Affymetrix), 'ug' (UniGene), 'fb' (FlyBase), 'go' (Gene Ontology), 'ens' (Ensembl). Additional repositories can easily be added. See <code>setRepository</code> for more information.
<code>...</code>	Further arguments to be passed. See details for more information.

**Details**

This function will accept a list or `data.frame` of character vectors, each containing different ids that are to be turned into hyperlinks (e.g., a list containing affy ids, genbank accession numbers, and Entrez Gene ids). For instances where there are more than one id per gene, use a sub-list of character vectors. See the vignette 'HowTo: Get HTML Output' for more information. Othernames should be a list or `data.frame`. Again, if there are multiple entries for a given gene, use a sub-list. This is more easily explained using an example - please see the examples section below and the above mentioned vignette.

In even the simplest case the `genelist`, `othernames` and `repository` have to be lists. A simple character vector will not suffice.

Note that this function now uses `xtable` to create the HTML table, and there is the ability to pass some arguments on to either `xtable` or `print.xtable`. One such argument would be `'append=TRUE'`, which would allow one to put lots of tables in one page, as long as the filename argument remained the same.

Additionally, the Ensembl repository needs a species argument in order to form a usable URI. This argument can be passed in the form of e.g., `species = "Homo\_sapiens"`. Note the capitalization of the genus, and the separation by an underscore (`\_`).

**Value**

This function is used only for the side effect of creating an HTML table.

**Author(s)**

Robert Gentleman <rgentlem@fhcrc.org>, further modifications by James W. MacDonald <jmacdon@med.umich.edu>

**Examples**

```
library("annotate")
## A very simple example. Two columns, one with links, the other without.
```

```

gos <- paste("GO:000000", 1:9, sep="")
notlinks <- LETTERS[1:9]

htmlpage(list(gos), "simple.html", "Two column data", list(notlinks),
          c("GO IDs", "Letters"), repository = list("go"))

if(!interactive())
  file.remove("simple.html")

## A more complex example with multiple links per cell
## first we create data to annotate
unigene <- list("Hs.600536",c("Hs.596913","HS.655491"),"Hs.76704")
refseq <- list(c("NM_001030050", "NM_001030047", "NM_001648",
"NM_001030049"), "NM_000860", c("NM_001011645", "NM_000044"))
entrez <- c("354", "3248", "367")
genelist <- list(unigene, refseq, entrez)

## now some other data

symb <- c("KLK3","HPGD","AR")
desc <- c("Prostate-specific antigen precursor",
"15-hydroxyprostaglandin dehydrogenase",
"Androgen receptor")
t.stat <- c(40.21, -22.14, 21.56)
p.value <- rep(0,3)
fold.change <- c(3.54, -2.35, 3.18)
expression <- matrix(c(11.78, 11.69, 11.62, 8.17, 5.78, 5.58, 5.68,
8.26, 9.08, 9.28, 9.19, 6.05), ncol=4, byrow=TRUE)

otherdata <- list(symb, desc, t.stat, p.value, fold.change, expression)
table.head <- c("UniGene", "RefSeq", "EntrezGene", "Symbol",
"Description", "t-stat", "p-value", "fold change",
paste("Sample", 1:4))

htmlpage(genelist, "test.html", "Some gene expression data", otherdata,
          table.head, repository = list("ug","gb","en"))

if(!interactive())
  file.remove("test.html")

```

---

isValidKey

*Get or verify valid IDs for a package.*


---

## Description

These functions either verify that a list of IDs are primary and valid IDs for a package, or else return all the valid primary IDs from a package

## Usage

```

isValidKey(ids, pkg)
allValidKeys(pkg)

```



**Arguments**

`ids` A character vector containing IDs that you wish to validate.  
`pkg` The package name of the chip for which we wish to validate IDs.

**Details**

Every package has some kind of ID that is central to that package. For chip-based packages this will be some kind of probe, and for the organism based packages it will be something else (usually an entrez gene ID). `isValidKey` takes a list of IDs and tests to see whether or not they are present (valid) in a particular package. `allValidKeys` simply returns all the valid primary IDs for a package.

**Value**

`isValidKey` returns a vector of TRUE or FALSE values corresponding to whether or not the ID is valid.

`allValidKeys` returns a vector of all the valid primary IDs.

**Author(s)**

Marc Carlson

**See Also**

[updateSymbolsToValidKeys](#)

**Examples**

```
## Not run:
## 2 bad IDs and a 3rd that will be valid
ids <- c("15S_rRNA_2", "21S_rRNA_4", "15S_rRNA")
isValidKey(ids, "org.Sc.sgd")

## 2 good IDs and a 3rd that will not be valid
ids <- c("5600", "7531", "altSymbol")
isValidKey(ids, "org.Hs.eg")

## Get all the valid primary id from org.Hs.eg.db
allValidKeys("org.Hs.eg")

## End(Not run)
```

---

makeAnchor

*A Function To Generate HTML Anchors*

---

**Description**

This function will take a set of links and titles and will generate HTML anchor tags out of these values

**Usage**

```
makeAnchor(link, title, toMain = FALSE)
```

**Arguments**

link	A vector of URLs
title	A vector of website names
toMain	Used for frame pages

**Value**

A vector of HTML anchor tags

**Author(s)**

Jeff Gentry

**Examples**

```
makeAnchor("http://www.bioconductor.org", "Bioconductor")
```

---

mapOrgs

*Functions to map to organism IDs used by NCBI homology.*

---

**Description**

These functions help map to organism identifiers used at the NCBI.

**Usage**

```
mapOrgs(toMap, what = c("code", "name"))
getOrgNameNCode()
```

**Arguments**

toMap	vect a vector of character strings
what	what a character string that can either be "code" or "name".

**Details**

mapOrgs converts organism codes to scientific names.

**Value**

mapOrgs returns a vector of character strings.

**Author(s)**

Jianhua Zhang

**References**

<ftp://ftp.ncbi.nih.gov/pub/HomoloGene/README>

---

neighborGeneFinder *A widget for locating genes neighboring a target gene*

---

## Description

Given a set of data (matrix) with entries for Entrez Gene or UniGene ids, the neighboring genes of a gene selected from a list on the interface can be located.

## Usage

```
neighborGeneFinder(geneData, keyName = c("unigene", "locuslink"),
  organism = c("human", "mouse", "rat"))
```

## Arguments

geneData	geneData a matrix with columns named. The name for one of the columns has to be either "locuslink" or "unigene"
keyName	keyName a character string for the name of the key columns of geneData. Has to be either "locuslink" or "unigene"
organism	organism a character string for the name of the organism of interest. Has to be "human", "mouse", or "rat"

## Details

Bioconductor's data package XXXCHRLLOC has to be installed for the widget to work. If keyName is "unigene", XXXLLMappings is required, where XXX is the name of the organism of interest.

## Value

This function returns a list of lists. Elements of the top level list are either Entrez Gene or UniGene ids. A sublist is return a list of lists whose top level elements are chromosome numbers, each of which is a list with an "upstream" and "downstream" elements.

## Author(s)

Jianhua Zhang

## Examples

```
if(interactive()){
  require("annotate", character.only = TRUE) ||
  stop("Package annotate is not available")
  geneData <- cbind(paste("100", 1:16, "_at", sep = ""), c(1, 50,
    10044, 51, 71, 51371, 81, 51426, 188, 293, 360,
    364, 375, 387, 513, 10572))
  colnames(geneData) <- c("Probe", "locuslink")
  neighborGeneFinder(geneData, "locuslink", "human")
}
```

---

organism	<i>Convenience function for getting the organism from an object or package</i>
----------	--

---

**Description**

The most basic organism method just takes a character string (which represents a particular annotation package) and returns the organism that said package is based upon.

**Usage**

```
organism(object)
```

**Arguments**

object            a character string that names a package

**Value**

The name of the organism used for this package or object

**Author(s)**

Marc Carlson

**Examples**

```
require(hgu95av2.db)
## get the organism for this annotation package
organism("hgu95av2")

## get the organism this object refers to
## (for a ChromLocation object)
z <- buildChromLocation("hgu95av2")
organism(z)
```

---

p2LL

---

*A function to map from probes to unique Entrez Gene IDs*


---

**Description**

For any chip, this function computes the map from unique Entrez Gene ID to all probes.

**Usage**

```
p2LL(data)
```

**Arguments**

data            The character string naming the chip.

## Details

This function is deprecated.

This is essentially the computation of the reverse map, we store probe to Entrez gene information in the `ENTREZID` environment. This is used to compute the inverse mapping.

## Value

A list, with length equal to the number of unique Entrez Gene IDs on the chip, the elements correspond to the probes that map to the Gene ID.

## Author(s)

R. Gentleman

## See Also

[getEG](#)

## Examples

```
## Not run:
  library("hgu95av2.db")
  x <- p2LL("hgu95av2")
  table(sapply(x, length))

## End(Not run)
```

---

pm.abstGrep

*An interface to grep for PubMed abstracts.*

---

## Description

A user friendly interface to the functionality provided by `pubmed`.

## Usage

```
pm.abstGrep(pattern, absts, ...)
```

## Arguments

<code>pattern</code>	A pattern for the call to <code>grep</code> .
<code>absts</code>	A list containing abstracts downloaded using <code>pubmed</code> or equivalent.
<code>...</code>	Extra arguments passed to <code>grep</code> .

## Details

The `absts` are a list of PubMed XML objects that have been downloaded and parsed. This function lets the user quickly search the abstracts for any regular expression. The returned value is a logical vector indicating which of the abstracts contain the regular expression.

**Value**

The returned value is a logical vector indicating which of the abstracts contain the regular expression.

**Author(s)**

Robert Gentleman

**See Also**

[pm.getabst](#), [pm.titles](#)

**Examples**

```
library("hgu95av2.db")
hoxa9 <- "37809_at"
absts <- pm.getabst(hoxa9, "hgu95av2")
pm.abstGrep("NUP98", absts[[1]])
pm.abstGrep("apoptosis", absts[[1]])
```

---

pm.getabst

*Obtain the abstracts for a set PubMed list.*

---

**Description**

The data provided by PubMed is reduced to a small set. This set is then suitable for further rendering.

**Usage**

```
pm.getabst(geneids, basename)
```

**Arguments**

geneids	The identifiers used to find Abstracts
basename	The base name of the annotation package to use.

**Details**

We rely on the annotation in the package associated with the `basename` to provide PubMed identifiers for the genes described by the gene identifiers. With these in hand we then use the `pmfetch` utility to download the PubMed abstracts in XML form. These are then translated (transformed) to a shorter version containing a small subset of the data provided by PubMed.

This function has the side effect of creating an environment in `.GlobalEnv` that contains the mapping for the requested data. This is done for efficiency – so we don't continually read in the data when there are many different queries to be performed.

**Value**

A list of lists containing objects of class `pubMedAbst`. There will be one element of the list for each identifier. Each of these elements is a list containing one abstract (of class `pubMedAbst` for each PubMed identifier associated with the gene identifier.

**Author(s)**

Robert Gentleman

**See Also**

[pm.abstGrep](#), [pm.titles](#)

**Examples**

```
library("hgu95av2.db")
hoxa9 <- "37809_at"
absts <- pm.getabst(hoxa9, "hgu95av2")
```

---

pm.titles

*Obtain the titles of the PubMed abstracts.*

---

**Description**

This function returns the titles from a list of PubMed abstracts.

**Usage**

```
pm.titles(absts)
```

**Arguments**

absts            The list of PubMed abstracts.

**Details**

It simply uses `sapply`.

**Value**

A character vector of length equal to the number of abstracts. Each element is the title of the corresponding abstract.

**Author(s)**

Robert Gentleman

**See Also**

[pm.abstGrep](#)

**Examples**

```
library("hgu95av2.db")
hoxa9 <- "37809_at"
absts <- pm.getabst(hoxa9, "hgu95av2")
pm.titles(absts)
```

## Description

This function will take a `pubMedAbst` object, or a list of these objects and generate a web page that will list the titles of the abstracts and link to their full page on PubMed

## Usage

```
pmAbst2HTML(absts, filename, title, frames = FALSE, table.center = TRUE)
```

## Arguments

<code>absts</code>	A list of <code>pubMedAbst</code> (or a single object)
<code>filename</code>	The output filename. If <code>frames</code> is <code>FALSE</code> , this is the name of the single output file and defaults to <code>absts.html</code> . Otherwise, this is taken to be the base of a set of filenames, and the default base is the empty string. See <code>value</code> for more information on output files.
<code>title</code>	Extra title information for your listing
<code>frames</code>	If <code>frames</code> is <code>TRUE</code> , the resulting page will use HTML frames, resulting in a more complex set of output pages.
<code>table.center</code>	If <code>TRUE</code> , will center the listing of abstracts

## Details

This function uses the `Entrez` functionality provided by NCBI to retrieve the abstract URL at the PubMed site. It will then create a tabular webpage which will list the titles of the abstracts provided and have them link to the appropriate PubMed page. If `frames` is `TRUE`, the table of links will be on the left hand side of the page and the right hand will link directly to the appropriate PubMed page.

## Value

If `frames` is `FALSE`, a simple HTML file is created with the name specified by `filename`.

If `frames` is `TRUE`, then there are four HTML files created, of the form `XXXtop.html`, `XXXside.html`, `XXXmain.html` and `XXXindex.html`, where `XXX` is the string provided by `filename`.

## Author(s)

Jeff Gentry

## See Also

`pubMedAbst`



**Examples**

```
x <- pubmed("9695952","8325638","8422497")
a <- xmlRoot(x)
numAbst <- length(xmlChildren(a))
absts <- list()
for (i in 1:numAbst) {
  absts[[i]] <- buildPubMedAbst(a[[i]])
}
## First try it w/o frames - using a temporary
## file for the output
fname <- tempfile()
pmAbst2HTML(absts, filename=fname)

if (interactive())
  browseURL(paste("file://", fname, sep=""))

## Now try it w/ frames, using temporary files again.
fnameBase <- tempfile()
pmAbst2HTML(absts, filename=fnameBase, frames=TRUE)

if (interactive())
  browseURL(paste("file://", fnameBase, "index.html", sep=""))
```

---

pmid2MIAME

*use web to populate MIAME instance with pubmed details*


---

**Description**

use web to populate MIAME instance with pubmed details

**Usage**

```
pmid2MIAME(pmid)
```

**Arguments**

```
pmid          string encoding PMID
```

**Details**

uses XML library to decode parts of the query response and load a MIAME object

**Value**

An instance of class [MIAME](#)

**Author(s)**

Vince Carey <stvjc@channing.harvard.edu>

**Examples**

```
if (interactive()) pmid2MIAME("9843569")
```

---

pmidQuery      *A function to query PubMed*

---

### Description

Given a PMID, will create a URL which can be used to open a browser and retrieve the specified information from PubMed.

### Usage

```
pmidQuery(query)
```

### Arguments

query      The PubMed ID (or IDs)

### Details

Using published details from NCBI we construct an appropriate string for directing a web browser to the information available at the NCBI.

### Value

A character string containing the appropriate URL

### Author(s)

Jeff Gentry

### References

NCBI, <http://www.ncbi.nih.gov/>

### See Also

[UniGeneQuery](#)

### Examples

```
a <- "9695952"
pmidQuery(a)
```

---

probesByLL	<i>A function that does reverse the mappings between probe ids and the corresponding values</i>
------------	---

---

### Description

This function takes the name of a platform specific annotation data package (e. g. "hgu95av2") and a character string for the name of an environment stored in the data directory and returns a list of vectors of character string with values of the environment as the names of the lists of vectors and probe ids corresponding to the values of the environment as values.

### Usage

```
probesByLL(baseName, what = "ENTREZID")
```

### Arguments

baseName	baseName a character string for the name of platform specific annotation data package
what	what a character string for the name of the environment object, which is the last part of an environment name excluding the package name

### Details

Each platform specific annotation data package has mappings between probe ids to other values. This function does reverse mappings by grouping probe ids under unique values of a given environment and returns the mappings as a list.

Values for the environment object for which the reverse mappings are sought must be vectors.

### Value

This function returns a list of vectors of character strings with unique values of an environment as the names and probe ids corresponding to the values as values.

### Author(s)

Jianhua Zhang

### Examples

```
library("hgu95av2.db")
tt <- probesByLL("hgu95av2", "ENTREZID")
tt[1:4]
```

---

pubMedAbst-class     *Class pubMedAbst, a class to handle PubMed abstracts, and methods for processing them.*

---

## Description

This is a class representation for PubMed abstracts.

## Creating Objects

```
new('pubMedAbst',
  authors = ....., # Object of class vector
  pmid = ....., # Object of class character
  abstText = ....., # Object of class character
  articleTitle = ....., # object of class character
  journal = ....., # Object of class character
  pubDate = ....., # Object of class character
)
```

## Slots

**pmid:** Object of class "character" The PubMed ID for this paper.  
**authors:** Object of class "vector" The authors of the paper.  
**abstText:** Object of class "character" The contained text of the abstract.  
**articleTitle:** Object of class "character" The title of the article the abstract pertains to.  
**journal:** Object of class "character" The journal the article was published in.  
**pubDate:** Object of class "character" The date the journal was published.

## Methods

**pmid** signature(object = "pmid"): An accessor function for pmid  
**abstText** signature(object = "pubMedAbst"): An accessor function for abstText  
**articleTitle** signature(object = "pubMedAbst"): An accessor function for articleTitle  
**authors** signature(object = "pubMedAbst"): An accessor function for authors  
**journal** signature(object = "pubMedAbst"): An accessor function for journal  
**pubDate** signature(object = "pubMedAbst"): An accessor function for pubDate

## Author(s)

Jeff Gentry

## See Also

[pubmed](#), [genbank](#)

**Examples**

```
x <- pubmed("9695952", "8325638", "8422497")
a <- xmlRoot(x)
numAbst <- length(xmlChildren(a))
absts <- list()
for (i in 1:numAbst) {
  absts[[i]] <- buildPubMedAbst(a[[i]])
}
```

pubmed

*A function to open the browser to Pubmed with the selected gene.***Description**

Given a vector of Pubmed identifiers or accession numbers, the user can either have a browser display a URL showing a Pubmed query for those identifiers, or a XMLDoc object with the same data.

**Usage**

```
pubmed(..., disp=c("data", "browser"), type=c("uid", "accession"),
        pmaddress=.efetch("PubMed", disp, type))
```

**Arguments**

...	Vectorized set of Pubmed ID's
disp	Either "Data" or "Browser" (default is data). Data returns a XMLDoc, while Browser will display information in the user's browser.
type	Denotes whether the arguments are accession numbers or UIDS. Defaults to uids.
pmaddress	Specific path to the pubmed efetch engine from the NCBI website.

**Details**

A simple function to retrieve Pubmed data given a specific ID, either through XML or through a web browser. This function will accept either pubmed accession numbers or NCBI UIDs (defined as a Pubmed ID or a Medline ID) - although the types must not be mixed in a single call.

**WARNING:** The powers that be at NCBI have been known to ban the IP addresses of users who abuse their servers (currently defined as less then 2 seconds between queries). Do NOT put this function in a type loop or you may find your access revoked.

**Value**

If the option "data" is used, an object of type XMLDoc is returned, unless there was an error with the query in which case an object of type try-error is returned.

If the option "browser" is used, nothing is returned.

**Author(s)**

R. Gentleman

**See Also**

[genbank](#), [xmlTreeParse](#)

**Examples**

```
if( interactive() )
  opts <- c("data","browser") else
  opts <- "data"
for (dp in opts)
  pubmed("11780146","11886385","11884611",disp=dp)
```

---

readGEOAnn

*Function to extract data from the GEO web site*

---

**Description**

Data files that are available at GEO web site are identified by GEO accession numbers. Given the url for the CGI script at GEO and a GEO accession number, the functions extract data from the web site and returns a matrix containing the data.

**Usage**

```
readGEOAnn(GEOAccNum, url = "http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?")
readIDNAcc(GEOAccNum, url = "http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?")
getGPLNames(url = "http://www.ncbi.nlm.nih.gov/geo/query/browse.cgi?")
getSAGEFileInfo(url =
  "http://www.ncbi.nlm.nih.gov/geo/query/browse.cgi?view=pl
getSAGEGPL(organism = "Homo sapiens", enzyme = c("NlaIII", "Sau3A"))
readUrl(url)
```

**Arguments**

url	url the url for the CGI script at GEO
GEOAccNum	GEOAccNum a character string for the GEO accession number of a desired file (e. g. GPL97)
organism	organism a character string for the name of the organism of interests
enzyme	enzyme a character string that can be either NlaII or Sau3A for the enzyme used to create SAGE tags

**Details**

url is the CGI script that processes user's request. [readGEOAnn](#) invokes the CGI by passing a GEO accession number and then processes the data file obtained.

[readIDNAcc](#) calls [readGEOAnn](#) to read the data and the extracts the columns for probe ids and accession numbers. The GEOAccNum has to be the id for an Affymetrix chip.

[getGPLNames](#) parses the html file that lists GEO accession numbers and descriptions of the array represented by the corresponding GEO accession numbers.

**Value**

Both `readGEOAnn` and `readIDNAcc` return a matrix.

`getGPLNames` returns a named vector of the names of commercial arrays. The names of the vector are the corresponding GEO accession number.

**Author(s)**

Jianhua Zhang

**References**

[www.ncbi.nlm.nih.gov/geo](http://www.ncbi.nlm.nih.gov/geo)

**Examples**

```
# Get array names and GEO accession numbers
#geoAccNums <- getGPLNames()
# Read the annotation data file for HG-U133A which is GPL96 based on
# examining geoAccNums
#temp <- readGEOAnn(GEOAccNum = "GPL96")
#temp2 <- readIDNAcc(GEOAccNum = "GPL96")
```

---

serializeEnv

*A Function To Serialize Environment*

---

**Description**

This function will serialize an environment in R to an XML format stored in a compressed file.

**Usage**

```
serializeEnv(env, fname)
serializeDataPkgEnvs(pkgDir)
```

**Arguments**

<code>env</code>	The name of the environment to serialize.
<code>fname</code>	The name of the output file.
<code>pkgDir</code>	The directory where a data package is

**Details**

The environment is converted into an XML format and then outputted to a gzipped file (using `gzfile`). The values in the environment are serialized (using `serialize`) in ASCII format although the keys are stored in plain text.

The format of the XML is very simple, with the primary block being `values`, which contain blocks of `entries`, and each entry having a `key` and a `value`. For instance, if we had an environment with one value in it, the character `c` with a key of `a` (e.g. `assign("a", "c", env=foo)`), this is what the output would look like.

```

<?xml version="1.0"?>
<values xmlns:bt="http://www.bioconductor.org/RGDBM">
  <entry>
    <key>
      a
    </key>
    <value>
      A\n2\n131072\n66560\n1040\n1\n1033\n1\nc\n
    </value>
  </entry>
</values>

```

**Author(s)**

Jeff Gentry

**See Also**

[gzfile](#), [serialize](#)

**Examples**

```

z <- new.env()
assign("a", 1, env=z)
assign("b", 2, env=z)
assign("c", 3, env=z)
serializeEnv(z, tempfile())

```

---

setRepository

*Functions to add arbitrary repositories*

---

**Description**

These functions allow end users to add arbitrary repositories for use with the `htmlpage` function.

**Usage**

```

setRepository(repository, FUN, ..., verbose=TRUE)
getRepositories()
clearRepository(repository, verbose=TRUE)

```

**Arguments**

<code>repository</code>	A character name for the repository.
<code>FUN</code>	A function to build hyperlinks for the repository. See details for more information.
<code>...</code>	Allows one to pass arbitrary code to underlying functions.
<code>verbose</code>	Output warning messages?



## Details

These functions allow end users to add, view, and remove repositories for use with the `htmlpage` function. `getRepositories` will output a vector of names for available repositories. `clearRepository` can be used to remove a repository if so desired. `setRepository` can be used to add a repository. See the examples section for the format of the FUN argument.

Once a new repository has been set, the `htmlpage` function can be called using the name of the new repository as a value in the repository argument (e.g., `htmlpage(<other args>, repository = list("newrepositoryname"))`)

## Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

## Examples

```
## A simple fake URI
repofun <- function(ids, ...)
paste("http://www.afaakeuri.com/", ids, sep = "")

setRepository("simple", repofun)

## More complicated, we want to make sure that
## NAs get converted to empty cells

repofun <- function(ids, ...){
bIDs <- which(is.na(ids))
out <- paste("http://www.afaakeuri.com/", ids, sep = "")
out[bIDs] <- "&nbsp;"
out
}

setRepository("complex", repofun)

## More complicated URI where we need to pass more information
## An example is Ensembl, which requires a species as part of the URI
## Since htmlpage() has an '...' argument, we can pass arbitrary
## arguments to this function that will be passed down to our
## repfun. Here we assume the argument species="Homo_sapiens" has been
## included in the call to htmlpage().

repofun <- function(ids, ...){
if(!is.null(list(...)$species))
  species <- list(...)$species
else
  stop("To make links for Ensembl, you need to pass a 'species' argument.",
       call. = FALSE)
out <- paste("http://www.ensembl.org/", species, "/Search/Summary?species=",
            species, ";idx=q=", ids, sep = "")
out
}

setRepository("species_arg", repofun)
```

```
updateSymbolsToValidKeys
```

*Take a list of symbols and translate them into the best possible ID for a package.*

---

## Description

Given a list of gene symbols and a package, find a valid ID for that package. If there isn't a valid ID, then return the original symbol.

## Usage

```
updateSymbolsToValidKeys(symbols, pkg)
```

## Arguments

<code>symbols</code>	A character vector containing gene symbols that you wish to try and translate into valid IDs.
<code>pkg</code>	The package name of the chip for which we wish to validate IDs.

## Details

This is a convenience function for getting from a possibly varied list of gene symbols mapped onto something that is a nice concrete ID such as an entrez gene ID. When such an ID cannot be found, the original symbol will come back to prevent the loss of any information.

## Value

This function returns a vector of IDs corresponding to the symbols that were input. If the symbols don't have a valid ID, then they come back instead.

## Author(s)

Marc Carlson

## See Also

[isValidKey](#)

## Examples

```
## Not run:
## one "bad" ID, one that can be mapped onto a valid ID, and a 3rd
## which already is a valid ID
syms <- c("15S_rRNA_2", "21S_rRNA_4", "15S_rRNA")
updateSymbolsToValidKeys(syms, "org.Sc.sgd")

## 3 symbols and a 4th that will NOT be valid
syms <- c("MAPK11", "P38B", "FLJ45465", "altSymbol")
updateSymbolsToValidKeys(syms, "org.Hs.eg")

## End(Not run)
```

---

usedChromGenes	<i>A function to select used genes on a chromosome from an ExpressionSet.</i>
----------------	---

---

**Description**

Given an instance of an `ExpressionSet`, a `chromLocation` object and the name of a chromosome this function returns all genes represented in the `ExpressionSet` on the specified chromosome.

**Usage**

```
usedChromGenes(eSet, chrom, specChrom)
```

**Arguments**

eSet	An instance of an <code>ExpressionSet</code> object.
chrom	The name of the chromosome of interest.
specChrom	An instance of a <code>chromLocation</code> object.

**Value**

Returns a vector of gene names that represent the genes from the `ExpressionSet` that are on the specified chromosome.

**Author(s)**

Jeff Gentry

**Examples**

```
data(sample.ExpressionSet)
data(hgu95AProbLocs)
usedChromGenes(sample.ExpressionSet, "1", hgu95AProbLocs)
```

# Index

## \*Topic **classes**

chromLocation-class, 14  
homoData-class, 37  
HTMLPage-class, 4  
pubMedAbst-class, 52

## \*Topic **datasets**

hgByChroms, 34  
hgCLengths, 34  
hgu95Achroloc, 35  
hgu95Achrom, 36  
hgu95All, 36  
hgu95Asym, 37

## \*Topic **data**

chrCats, 13

## \*Topic **interface**

accessionToUID, 9  
entrezGeneByID, 18  
entrezGeneQuery, 19  
genbank, 22  
neighborGeneFinder, 43  
pmidQuery, 50  
pubmed, 53  
UniGeneQuery, 8  
usedChromGenes, 59

## \*Topic **iplot**

genelocator, 23

## \*Topic **manip**

annPkgName, 10  
aqListGOIDs, 10  
dropECode, 17  
filterGOByOntology, 20  
findNeighbors, 20  
getAnnMap, 24  
getEvidence, 25  
getGOTerm, 26  
getOntology, 27  
getQueryLink, 31  
getSYMBOL, 29  
GO2heatmap, 2  
GOMnplot, 3  
hasGOannotate, 33  
htmlpage, 38  
isValidKey, 40

mapOrgs, 42  
organism, 44  
p2LL, 44  
pm.abstGrep, 45  
pm.getabst, 46  
pm.titles, 47  
PMIDAmat, 6  
probesByLL, 51  
PWAmat, 7  
readGEOAnn, 54  
setRepository, 56  
updateSymbolsToValidKeys, 58

## \*Topic **methods**

hgu95AProbLocs, 35

## \*Topic **misc**

ACCNUMStats, 1  
compatibleVersions, 16  
getSEQ, 30  
LL2homology, 5

## \*Topic **models**

getPMInfo, 28  
pmid2MIAME, 49

## \*Topic **utilities**

buildChromLocation, 11  
buildPubMedAbst, 12  
makeAnchor, 41  
pmAbst2HTML, 48  
serializeEnv, 55

abstText (*pubMedAbst-class*), 52  
abstText, pubMedAbst-method  
(*pubMedAbst-class*), 52  
ACC2homology (*LL2homology*), 5  
accessionToUID, 9  
ACCNUMStats, 1  
allValidKeys (*isValidKey*), 40  
annPkgName, 10  
aqListGOIDs, 10  
articleTitle (*pubMedAbst-class*),  
52  
articleTitle, pubMedAbst-method  
(*pubMedAbst-class*), 52  
authors (*pubMedAbst-class*), 52

- authors, *pubMedAbst*-method  
(*pubMedAbst*-class), 52
- buildChromLocation, 11, 15
- buildPubMedAbst, 12
- checkArgs (*findNeighbors*), 20
- chrCats, 13
- chromInfo (*chromLocation*-class),  
14
- chromInfo, *chromLocation*-method  
(*chromLocation*-class), 14
- chromLengths  
(*chromLocation*-class), 14
- chromLengths, *chromLocation*-method  
(*chromLocation*-class), 14
- chromLocation-class, 14
- chromLocs (*chromLocation*-class),  
14
- chromLocs, *chromLocation*-method  
(*chromLocation*-class), 14
- chromNames (*chromLocation*-class),  
14
- chromNames, *chromLocation*-method  
(*chromLocation*-class), 14
- clearRepository (*setRepository*),  
56
- compatibleVersions, 16
- createLLChrCats (*chrCats*), 13
- createMAPIncMat (*chrCats*), 13
- dataSource (*chromLocation*-class),  
14
- dataSource, *chromLocation*-method  
(*chromLocation*-class), 14
- dropECode, 17, 25, 27
- entrezGeneByID, 18
- entrezGeneQuery, 19
- fileName (*HTMLPage*-class), 4
- fileName, *HTMLPage*-method  
(*HTMLPage*-class), 4
- filterGOByOntology, 20
- findChr4LL (*findNeighbors*), 20
- findNeighbors, 20
- FramedHTMLPage (*HTMLPage*-class), 4
- FramedHTMLPage, *HTMLPage*-method  
(*HTMLPage*-class), 4
- FramedHTMLPage-class  
(*HTMLPage*-class), 4
- genbank, 12, 22, 52, 54
- genelocator, 23
- geneSymbols  
(*chromLocation*-class), 14
- geneSymbols, *chromLocation*-method  
(*chromLocation*-class), 14
- get, 33
- getAnnMap, 10, 24
- getBoundary (*findNeighbors*), 20
- getCells, 32
- getCells (*getQueryLink*), 31
- getEG, 45
- getEG (*getSYMBOL*), 29
- getEvidence, 17, 25, 27
- getGI (*getSEQ*), 30
- getGO (*getSYMBOL*), 29
- getGOChildren (*getGOTerm*), 26
- getGODesc (*getSYMBOL*), 29
- getGOOntology (*getGOTerm*), 26
- getGOParents (*getGOTerm*), 26
- getGOTerm, 26
- getGPLNames, 54, 55
- getGPLNames (*readGEOAnn*), 54
- getLL (*getSYMBOL*), 29
- getOntology, 17, 25, 27
- getOrgNameNCode (*mapOrgs*), 42
- getPMID (*getSYMBOL*), 29
- getPMInfo, 28
- getQuery4Affy, 32
- getQuery4Affy (*getQueryLink*), 31
- getQuery4EN, 32
- getQuery4EN (*getQueryLink*), 31
- getQuery4ENSEMBL, 32
- getQuery4ENSEMBL (*getQueryLink*),  
31
- getQuery4FB, 32
- getQuery4FB (*getQueryLink*), 31
- getQuery4GB, 32
- getQuery4GB (*getQueryLink*), 31
- getQuery4LL, 32
- getQuery4LL (*getQueryLink*), 31
- getQuery4OMIM, 32
- getQuery4OMIM (*getQueryLink*), 31
- getQuery4SP, 32
- getQuery4SP (*getQueryLink*), 31
- getQuery4TR, 32
- getQuery4TR (*getQueryLink*), 31
- getQuery4UG, 32
- getQuery4UG (*getQueryLink*), 31
- getQueryLink, 31, 32
- getRepositories (*setRepository*),  
56
- getSAGEFileInfo (*readGEOAnn*), 54
- getSAGEGPL (*readGEOAnn*), 54

- getSEQ, 30
- getSYMBOL, 29
- getTDRows, 32
- getTDRows (*getQueryLink*), 31
- getUniqAnnItem (*getSYMBOL*), 29
- getValidChr (*findNeighbors*), 20
- GO2heatmap, 2
- GOMnplot, 3, 7
- gzfile, 55, 56
  
- hasGOannotate, 33
- heatmap, 2
- hgByChroms, 34
- hgCLengths, 34
- HGID2homology (*LL2homology*), 5
- hgu95Achroloc, 35
- hgu95Achrom, 36
- hgu95All, 36
- hgu95AProbLocs, 35
- hgu95Asym, 37
- homoACC (*homoData-class*), 37
- homoACC, *homoData-method* (*homoData-class*), 37
- homoData (*homoData-class*), 37
- homoData-class, 37
- homoHGID (*homoData-class*), 37
- homoHGID, *homoData-method* (*homoData-class*), 37
- homoLL (*homoData-class*), 37
- homoLL, *homoData-method* (*homoData-class*), 37
- homoOrg (*homoData-class*), 37
- homoOrg, *homoData-method* (*homoData-class*), 37
- homoPS (*homoData-class*), 37
- homoPS, *homoData-method* (*homoData-class*), 37
- homoType (*homoData-class*), 37
- homoType, *homoData-method* (*homoData-class*), 37
- homoURL (*homoData-class*), 37
- homoURL, *homoData-method* (*homoData-class*), 37
- HTMLPage (*HTMLPage-class*), 4
- htmlpage, 31, 38
- HTMLPage, *HTMLPage-method* (*HTMLPage-class*), 4
- HTMLPage-class, 4
  
- initialize, *FramedHTMLPage-method* (*HTMLPage-class*), 4
- isValidKey, 40, 58
  
- journal (*pubMedAbst-class*), 52
- journal, *pubMedAbst-method* (*pubMedAbst-class*), 52
  
- KEGG2heatmap, 3, 7
- KEGG2heatmap (*GO2heatmap*), 2
- KEGG2heatmap, *character, eSet, character-method* (*GO2heatmap*), 2
- KEGG2heatmap, *character, matrix, character-method* (*GO2heatmap*), 2
- KEGGmnplot (*GOMnplot*), 3
- KEGGmnplot, *character, eSet, character-method* (*GOMnplot*), 3
- KEGGmnplot, *character, matrix, character-method* (*GOMnplot*), 3
  
- lapply, 17
- LL2homology, 5
- locator, 24
- lookUp (*getSYMBOL*), 29
  
- mainPage (*HTMLPage-class*), 4
- mainPage, *FramedHTMLPage-method* (*HTMLPage-class*), 4
- mainPage, *HTMLPage-method* (*HTMLPage-class*), 4
- makeAnchor, 41
- mapOrgs, 42
- mget, 30
- MIAME, 49
  
- nChrom (*chromLocation-class*), 14
- nChrom, *chromLocation-method* (*chromLocation-class*), 14
- neighborGeneFinder, 43
  
- organism, 44
- organism, *character-method* (*organism*), 44
- organism, *chromLocation-method* (*chromLocation-class*), 14
- orgNameNCode (*homoData-class*), 37
  
- p2LL, 44
- packageDescription, 16
- pageText (*HTMLPage-class*), 4
- pageText, *HTMLPage-method* (*HTMLPage-class*), 4
- pageTitle (*HTMLPage-class*), 4
- pageTitle, *HTMLPage-method* (*HTMLPage-class*), 4
  
- pm.abstGrep, 45, 47
- pm.getabst, 46, 46
- pm.titles, 46, 47, 47

- pmAbst2HTML, 48
- pmid (*pubMedAbst-class*), 52
- pmid, *pubMedAbst*-method  
(*pubMedAbst-class*), 52
- pmid2MIAME, 49
- PMIDAmat, 6
- pmidQuery, 50
- probesByLL, 51
- probesToChrom  
(*chromLocation-class*), 14
- probesToChrom, *chromLocation*-method  
(*chromLocation-class*), 14
- pubDate (*pubMedAbst-class*), 52
- pubDate, *pubMedAbst*-method  
(*pubMedAbst-class*), 52
- pubmed, 9, 12, 23, 52, 53
- pubMedAbst (*pubMedAbst-class*), 52
- pubMedAbst, *pubMedAbst*-method  
(*pubMedAbst-class*), 52
- pubMedAbst-class, 52
- PWAmat, 7
  
- readGEOAnn, 54, 54, 55
- readIDNacc, 54, 55
- readIDNacc (*readGEOAnn*), 54
- readUrl (*readGEOAnn*), 54
  
- sapply, 17
- serialize, 55, 56
- serializeDataPkgEnvs  
(*serializeEnv*), 55
- serializeEnv, 55
- setRepository, 56
- show, *chromLocation*-method  
(*chromLocation-class*), 14
- show, *FramedHTMLPage*-method  
(*HTMLPage-class*), 4
- show, *homoData*-method  
(*homoData-class*), 37
- show, *HTMLPage*-method  
(*HTMLPage-class*), 4
- show, *pubMedAbst*-method  
(*pubMedAbst-class*), 52
- sidePage (*HTMLPage-class*), 4
- sidePage, *FramedHTMLPage*-method  
(*HTMLPage-class*), 4
- sidePage, *HTMLPage*-method  
(*HTMLPage-class*), 4
  
- toFile (*HTMLPage-class*), 4
- toFile, *FramedHTMLPage*-method  
(*HTMLPage-class*), 4
- toFile, *HTMLPage*-method  
(*HTMLPage-class*), 4
- topPage (*HTMLPage-class*), 4
- topPage, *FramedHTMLPage*-method  
(*HTMLPage-class*), 4
- topPage, *HTMLPage*-method  
(*HTMLPage-class*), 4
  
- UniGeneQuery, 8, 50
- updateSymbolsToValidKeys, 41, 58
- usedChromGenes, 59
  
- weightByConfi (*findNeighbors*), 20
- whatACC (*ACCNUMStats*), 1
  
- xmlTreeParse, 9, 23, 54