Package 'AnnotationForge'

March 25, 2013

Title Code for Building Annotation Database Packages

Description Provides code for generating Annotation packages and
their databases. Packages produced are intended to be used with AnnotationDbi.

Version 1.0.3

Author Marc Carlson, Herve Pages

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Depends R (>= 2.7.0), methods, utils, BiocGenerics (>= 0.1.13),Biobase (>= 1.17.0), AnnotationDbi (>= 1.19.15), org.Hs.eg.db

Imports methods, utils, DBI, RSQLite, BiocGenerics, Biobase

Suggests DBI (>= 0.2-4), RSQLite (>= 0.6-4), XML, RCurl, hgu95av2.db,human.db0, affy, Homo.sapiens, hom.Hs.inp.db, GO.db

Collate AnnDbPkg-maker.R sqlForge_utils.R sqlForge_baseMapBuilder.R sqlForge_schemaGen.R sqlForge_tableBuilder.R sqlForge_makeAnnPkgs.R sqlForge_wrapBaseDBPkgs.R sqlForge_seqnames.R makeProbePackage.R NCBI_ftp.R NCBI_getters.R

License Artistic-2.0

biocViews Annotation, Infrastructure

R topics documented:

AnnDbPkg-maker *Creates an SQLite-based annotation package*

Description

Creates an SQLite-based annotation package from an SQLite file.

Usage

```
makeAnnDbPkg(x, dbfile, dest_dir=".", no.man=FALSE, ...)
loadAnnDbPkgIndex(file)
```

Arguments

x	A AnnDbPkgSeed object, a list, a string or a regular expression.
dbfile	The path to the SQLite containing the annotation data for the package to build.
dest_dir	The directory where the package will be created.
file	The path to a DCF file containing the list of annotation packages to build.
no.man	If TRUE then no man page is included in the package.
...	Extra args used for extra filtering.

See Also

AnnDbPkg-checker

Examples

```
## With a "AnnDbPkgSeed" object:
seed <- new("AnnDbPkgSeed",
    Package="hgu133a2.db",
    Version="0.0.99",
    PkgTemplate="HUMANCHIP.DB",
    AnnObjPrefix="hgu133a2"
)
if (FALSE)
    makeAnnDbPkg(seed, "path/to/hgu133a2.sqlite")

## With package names:
## (Note that in this case makeAnnDbPkg() will use the package descriptions
## found in the master index file ANNDBPKG-INDEX.TXT located in the
## AnnotationDbi package.)
if (FALSE)
    makeAnnDbPkg(c("hgu95av2.db", "hgu133a2.db"))

## A character vector of length 1 is treated as a regular expression:
if (FALSE)
    makeAnnDbPkg("hgu.*")
## To make all the packages described in the master index:
if (FALSE)
    makeAnnDbPkg("")
## Extra args can be used to narrow down the roaster of packages to make:
if (FALSE) {
```

```
    makeAnnDbPkg("", PkgTemplate="HUMANCHIP.DB", manufacturer="Affymetrix")
    makeAnnDbPkg(".*[3k]\\.db", species=c("Mouse", "Rat"))
}

## The master index file ANNDBPKG-INDEX.TXT can be loaded with:
loadAnnDbPkgIndex()
```

---

available.db0pkgs *available.db0pkgs*

Description

Get the list of intermediate annotation data packages (.db0 data packages) that are currently available on the Bioconductor repositories for your version of R/Bioconductor.

Or get a list of schemas supported by AnnotationDbi.

Usage

```
available.db0pkgs()
available.dbschemas()
available.chipdbschemas()
```

Details

The SQLForge code uses a series of intermediate database packages that are necessary to build updated custom annotation packages. These packages must be installed or updated if you want to make a custom annotation package for a particular organism. These special intermediate packages contain the latest freeze of the data needed to build custom annotation data packages and are easily identified by the fact that they end with the special ".db0" suffix. This function will list all such packages that are available for a specific version of bioconductor.

The available.dbschemas() and available.chipdbschemas() functions allow you to get a list of the schema names that are available similar to how you can list the available ".db0" packages by using available.db0pkgs(). This list of shemas is useful (for example) when you want to build a new package and need to know the name of the schema you want to use.

Value

A character vector containing the names of the available ".db0" data packages. Or a a character vector listing the names of the available schemas.

Author(s)

H. Pages and Marc Carlson

Examples

```
# Get the list of BSgenome data packages currently available:
available.db0pkgs()

## Not run:
  # Make your choice and install like this:
  source("http://bioconductor.org/biocLite.R")
```

```
  biocLite("human.db0")

## End(Not run)

  # Get the list of chip DB schemas:
  available.chipdbschemas()

  # Get the list of ALL DB schemas:
  available.dbschemas()
```

---

createSimpleBimap *Creates a simple Bimap from a SQLite database in an situation that is external to AnnotationDbi*

Description

This function allows users to easily make a simple Bimap object for extra tables etc that they may wish to add to their annotation packages. For most Bimaps, their definition is stored inside of AnnotationDbi. The addition of this function is to help ensure that this does not become a limitation, by allowing simple extra Bimaps to easily be defined external to AnnotationDbi. Usually, this will be done in the zzz.R source file of a package so that these extra mappings can be seemlessly integrated with the rest of the package. For now, this function assumes that users will want to use data from just one table.

Usage

```
createSimpleBimap(tablename, Lcolname, Rcolname, datacache, objName,
objTarget)
```

Arguments

tablename	The name of the database table to grab the mapping information from.
Lcolname	The field name from the database table. These will become the Lkeys in the final mapping.
Rcolname	The field name from the database table. These will become the Rkeys in the final mapping.
datacache	The datacache object should already exist for every standard Annotation package. It is not exported though, so you will have to access it with ::: . It is needed to provide the connection information to the function.
objName	This is the name of the mapping.
objTarget	This is the name of the thing the mapping goes with. For most uses, this will mean the package name that the mapping belongs with.

Examples

```
##You simply have to call this function to create a new mapping.  For
##example, you could have created a mapping between the gene_name and
##the symbols fields from the gene_info table contained in the hgu95av2
##package by doing this:
library(hgu95av2.db)
```

```
hgu95av2NAMESYMBOL <- createSimpleBimap("gene_info",
                                        "gene_name",
                                        "symbol",
                                        hgu95av2.db:::datacache,
                                        "NAMESYMBOL",
                                        "hgu95av2.db")
```

---

generateSeqnames.db *Generates the seqnames.db package and database*

Description

This function is used to generate the seqnames.db package and it's database from the csv files contained in the template for this package within AnnotationForge. The csv files are converted into database tables, and the DB is packaged into a new seqnames.db package.

Usage

```
generateSeqnames.db(version, outdir=".")
```

Arguments

version	Character. Version number for the final package.
outdir	Character. Path to output directory where the package is to be placed. By default the current working directory will be used.

Details

The generateSeqnames.db function allows users to regenerate the seqnames.db package from csv sources contained in the currently installed AnnotationForge package. It is expected that the typical user will not need to use this at all, but in case they do, we have made it available. We expect that the more common use case is someone who wants to make some new chromosome conventions available for the world. It is expected that this person will more typicalyl be charitable and want to share their conventions, so they could share their .csv files with us and we would add them to AnnotationForge, install the updated package and then run this function to make a new package.

The .csv files need to be formatted the same as the ones that are currently in the template in AnnotationForge. Examples of these .csv files can be found in AnnotationForge in the "inst/seqnames-template/inst/extdata/dataFiles/" directory. Each file must be named after it's corresponding genus and species with an underscore separator and a .csv file extension. The 1st line of each file defines columns that are the names of the corresponding naming conventions. And the chromosome names are then listed below this header line such that the equivalent names for the different styles share the same row.

So for example the 1st four rows of Mus_musculus look like this (but with only one newline at the end of each row):

UCSC,NCBI,ensembl

chr1,1,1

chr2,2,2

chr3,3,3

etc.

Once you have your file ready your only need to place it in the same dir in AnnoationDbi (with the other files), install AnnotationForge, and then run this function to generate a new seqnames.db package. Of course, if you have a useful set of conventions or species to contribute, it would be best if you gave your .csv files to the Bioconductor core team so that we can add these files to the official version of AnnotationForge and so that they can occur in the official seqnames.db package.

Value

A new seqnames.db package, complete with all the latest data stored in the dataFiles subdirectory

Author(s)

Marc Carlson

Examples

```
## Not run:
 generateSeqnames.db(version="1.0.0")

## End(Not run)
```

---

getProbeDataAffy *Read a data file describing the probe sequences on an Affymetrix genechip*

Description

Read a data file describing the probe sequences on an Affymetrix genechip

Usage

```
getProbeDataAffy(arraytype, datafile, pkgname = NULL, comparewithcdf = FALSE)
```

Arguments

arraytype	Character. Array type (e.g. 'HG-U133A')
datafile	Character with the filename of the input data file, or a connection (see example). If omitted a default name is constructed from arraytype (for details you will need to consult this function's source code).
pkgname	Character. Package name. If NULL the name is derived from arraytype.
comparewithcdf	Logical. If TRUE, run a consistency check against a CDF package of the same name (what used to be Laurent's "extraparanoia".)

Details

This function serves as an interface between the (1) representation of array probe information data in the packages that are generated by makeProbePackage and (2) the vendor- and possibly version-specific way the data are represented in datafile.

datafile is a tabulator-separated file with one row per probe, and column names 'Probe X', 'Probe Y', 'Probe Sequence', and 'Probe.Set.Name'. See the vignette for an example.

Value

A list with three components

dataEnv	an environment which contains the data frame with the probe sequences and the other probe data.
symVal	a named list of symbol value substitutions which can be used to customize the man pages. See createPackage.
pkgname	a character with the package name; will be the same as the function parameter pkgname if it was specified; otherwise, the name is constructed from the parameter arraytype.

See Also

makeProbePackage

Examples

```
## Please refer to the vignette
```

---

getProbeData_1lq *Read a 1lq file for an Affymetrix genechip*

Description

Read a 1lq file for an Affymetrix genechip

Usage

```
getProbeData_1lq(arraytype, datafile, pkgname = NULL)
```

Arguments

arraytype	Character. Array type (e.g. 'Scerevisiaetiling)
datafile	Character. The filename of the input data file. If omitted a default name is constructed from arraytype (see this function's source code).
pkgname	Character. Package name. If NULL the name is derived from arraytype.

Details

This function serves as an interface between the (1) representation of array probe information data in the packages that are generated by makeProbePackage and (2) the vendor- and possibly version-specific way the data are represented in datafile.

Value

A list with three components

dataEnv	an environment which contains the data frame with the probe sequences and the other probe data.
symVal	a named list of symbol value substitutions which can be used to customize the man pages. See createPackage.
pkgname	a character with the package name; will be the same as the function parameter pkgname if it was specified; otherwise, the name is constructed from the parameter arraytype.

See Also

makeProbePackage

Examples

```
## makeProbePackage(
##    arraytype = "Scerevisiaetiling",
##    maintainer= "Wolfgang Huber <huber@ebi.ac.uk>",
##    version   = "1.1.0",
##    datafile  = "S.cerevisiae_tiling.1lq",
##    importfun = "getProbeData_1lq")
```

---

makeDBPackage *Creates a sqlite database, and then makes an annotation package with it*

Description

This function 1st creates a SQLite file useful for making a SQLite based annotation package by using the correct popXXXCHIP_DB function. Next, this function produces an annotation package featuring the sqlite database produced. All makeXXXXChip_DB functions REQUIRE that you previously have installed the appropriate XXXX.db0 package. Call the function available.db0pkgs() to see what your options are, and then install the appropriate package with biocLite().

Usage

```
makeDBPackage(schema, ...)

# usage case with required arguments
# makeDBPackage(schema, affy, prefix, fileName, baseMapType, version)

# usage case with all arguments
# makeDBPackage(schema, affy, prefix, fileName, otherSrc, chipMapSrc,
# chipSrc, baseMapType, outputDir, version, manufacturer, chipName,
# manufacturerUrl, author, maintainer)
```

Arguments

schema	String listing the schema that you want to use to make the DB. You can list schemas with available.dbschemas()
affy	Boolean to indicate if this is starting from an affy csv file or not. If it is, then that will be parsed to make the sqlite file, if not, then you can feed a tab delimited file with IDs as was done before with AnnBuilder.
prefix	prefix is the first part of the eventual desired package name. (ie. "prefix.db")
fileName	The path and filename for the file to be parsed. This can either be an affy csv file or it can be a more classic file type.
otherSrc	The path and filenames to any other lists of IDs which might add information about how a probe will map.
chipMapSrc	The path and filename to the intermediate database containing the mapping data for allowed ID types and how these IDs relate to each other. If not provided, then the appropriate source DB from the most current .db0 package will be used instead.
chipSrc	The path and filename to the intermediate database containing the annotation data for the sqlite to build. If not provided, then the appropriate source DB from the most current .db0 package will be used instead.
baseMapType	The type of ID that is used for the initial base mapping. If using a classic base mapping file, this should be the ID type present in the fileName. This can be any of the following values: "gb" = for genbank IDs "ug" = unigene IDs "eg" = Entrez Gene IDs "refseq" = refseq IDs "gbNRef" = mixture of genbank and refseq IDs
outputDir	Where you would like the output files to be placed.
version	What is the version number for the desired package.
manufacturer	Who made the chip being described.
chipName	What is the name of the chip.
manufacturerUrl	URL for manufacturers website.
author	List of authors involved in making the package.
maintainer	List of package maintainers with email addresses for contact purposes.
...	Just used so we can have a wrapper function. Ignore this argument.

Examples

```
## Not run:
##Build the hgu95av2.db package
makeDBPackage("HUMANCHIP_DB",
              affy = TRUE,
              prefix = "hgu95av2",
          fileName = "/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/srcFiles/hgu95av2/HG_U95Av2_annot.csv.0
              otherSrc = c(
           EA="/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/srcFiles/hgu95av2/hgu95av2.EA.txt",
           UMICH="/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/srcFiles/hgu95av2/hgu95av2_UMICH.txt"),
              baseMapType = "gbNRef",
              version = "1.0.0",
              manufacturer = "Affymetrix",
              chipName = "hgu95av2",
              manufacturerUrl = "http://www.affymetrix.com")
```

```
## End(Not run)
```

---

makeOrgPackageFromNCBI

Making an organism package from annotations available from NCBI.

Description

The makeOrgPackageFromNCBI function allows the user to make an organism package from NCBI annotations available from the NCBI.

Usage

```
makeOrgPackageFromNCBI(
  version=,
  maintainer,
  author,
  outputDir=".",
  tax_id,
  genus,
  species)
```

Arguments

version	What is the version number for this package?
maintainer	Who is the package maintainer? (must include email to be valid)
author	Who is the creator of this package?
outputDir	A path where the package source should be assembled.
tax_id	The Taxonomy ID that represents your organism. (NCBI has a nice online browser for finding the one you need)
genus	Single string indicating the genus.
species	Single string indicating the species.

Value

Nothing returned to the R session. Just creates an organism annotation package.

Author(s)

M. Carlson

Examples

```
## Not run:
## Makes an organism package for Zebra Finch from NCBI:

makeOrgPackageFromNCBI(version = "0.1",
                       author = "Some One <so@someplace.org>",
                       maintainer = "Some One <so@someplace.org>",
                       outputDir = ".",
                       tax_id = "59729",
                       genus = "Taeniopygia",
                       species = "guttata")


## End(Not run)
```

---

makeProbePackage *Make a package with probe sequence related data for microarrays*

Description

Make a package with probe sequence related data for microarrays

Usage

```
makeProbePackage(arraytype,
   importfun = "getProbeDataAffy",
   maintainer,
   version,
   species,
   pkgname = NULL,
   outdir  = ".",
   force = FALSE, quiet = FALSE,
   check = TRUE, build = TRUE, unlink = TRUE, ...)
```

Arguments

arraytype	Character. Name of array type (typically a vendor's name like "HG-U133A").
importfun	Character. Name of a function that can read the probe sequence data e.g. from a file. See getProbeDataAffy for an example.
maintainer	Character. Name and email address of the maintainer.
version	Character. Version number for the package.
species	Character. Species name in the format Genus_species (e.g., Homo_sapiens)
pkgname	Character. Name of the package. If missing, a name is created from arraytype.
outdir	Character. Path where the package is to be written.
force	Logical. If TRUE overrides possible warnings
quiet	Logical. If TRUE do not print statements on progress on the console
check	Logical. If TRUE call R CMD check on the package

build	Logical. If TRUE call R CMD build on the package
unlink	Logical. If TRUE unlink (remove) the check directory (only relevant if check=TRUE)
...	Further arguments that get passed along to importfun

Details

See vignette.

Important note for *Windows* users: Building and checking packages requires some tools outside of R (e.g. a Perl interpreter). While these tools are standard with practically every Unix, they do not come with MS-Windows and need to be installed separately on your computer. See http://www.murdoch-sutherland.com/Rtools. If you just want to build probe packages, you will not need the compilers, and the "Windows help" stuff is optional.

Examples

```
filename <- system.file("extdata", "HG-U95Av2_probe_tab.gz",
    package="AnnotationDbi")
outdir   <- tempdir()
me       <- "Wolfgang Huber <huber@ebi.ac.uk>"
makeProbePackage("HG-U95Av2",
                 datafile   = gzfile(filename, open="r"),
                 outdir     = outdir,
                 maintainer = me,
                 version    = "0.0.1",
                 species    = "Homo_sapiens",
 check      = FALSE,
                 force      = TRUE)
dir(outdir)
```

---

populateDB *Populates an SQLite DB with and produces a schema definition*

Description

Creates SQLite file useful for making a SQLite based annotation package. Also produces the schema file which details the schema for the database produced.

Usage

```
populateDB(schema, ...)

#  usage case with required arguments
#  populateDB(schema, prefix, chipSrc, metaDataSrc)

#  usage case with all possible arguments
#  populateDB(schema, affy, prefix, fileName, chipMapSrc, chipSrc,
#  metaDataSrc, otherSrc, baseMapType, outputDir, printSchema)
```

Arguments

schema	String listing the schema that you want to use to make the DB. You can list schemas with available.dbschemas()
affy	Boolean to indicate if this is starting from an affy csv file or not. If it is, then that will be parsed to make the sqlite file, if not, then you can feed a tab delimited file with IDs as was done before with AnnBuilder.
prefix	prefix is the first part of the eventual desired package name. (ie. "prefix.sqlite")
fileName	The path and filename for the mapping file to be parsed. This can either be an affy csv file or it can be a more classic file type. This is only needed when making chip packages.
chipMapSrc	The path and filename to the intermediate database containing the mapping data for allowed ID types and how these IDs relate to each other. If not provided, then the appropriate source DB from the most current .db0 package will be used instead.
chipSrc	The path and filename to the intermediate database containing the annotation data for the sqlite to build. If not provided, then the appropriate source DB from the most current .db0 package will be used instead.
metaDataSrc	Either a named character vector containing pertinent information about the metadata OR the path and filename to the intermediate database containing the metadata information for the package. If this is a custom package, then it must be a named vector with the following fields: metaDataSrc <- c(DBSCHEMA="the DB schema", ORGANISM="the organism", SPECIES="the species", MANUFACTURER="the manufacturer", CHIPNAME="the chipName", MANUFACTURERURL="the manufacturerUrl")
otherSrc	The path and filenames to any other lists of IDs which might add information about how a probe will map.
baseMapType	The type of ID that is used for the initial base mapping. If using a classic base mapping file, this should be the ID type present in the fileName. This can be any of the following values: "gb" = for genbank IDs "ug" = unigene IDs "eg" = Entrez Gene IDs "refseq" = refseq IDs "gbNRef" = mixture of genbank and refseq IDs
outputDir	Where you would like the output files to be placed.
printSchema	Boolean to indicate whether or not to produce an output of the schema (default is FALSE).
...	Just used so we can have a wrapper function. Ignore this argument.

Examples

```
## Not run:
 ##Set up the metadata
 my_metaDataSrc <- c( DBSCHEMA="the DB schema",
                ORGANISM="the organism",
                SPECIES="the species",
                MANUFACTURER="the manufacturer",
                CHIPNAME="the chipName",
                MANUFACTURERURL="the manufacturerUrl")

 ##Builds the org.Hs.eg sqlite:
```

```
populateDB("HUMAN_DB",
           prefix="org.Hs.eg",
       chipSrc = "/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/annosrc/db/chipsrc_human.sqlite",
           metaDataSrc = my_metaDataSrc,
           printSchema=TRUE)


##Builds hgu95av2.sqlite:
populateDB("HUMANCHIP_DB",
           affy=TRUE,
           prefix="hgu95av2",
       fileName="/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/srcFiles/hgu95av2/HG_U95Av2.na27.annot.csv
           metaDataSrc=my_metaDataSrc,
           baseMapType="gbNRef")


##Builds the ag.sqlite:
populateDB("ARABIDOPSISCHIP_DB",
           affy=TRUE,
           prefix="ag",
           metaDataSrc=my_metaDataSrc)


##Builds yeast2.sqlite:
populateDB("YEASTCHIP_DB",
            affy=TRUE,
            prefix="yeast2",
       fileName="/mnt/cpb_anno/mcarlson/proj/mcarlson/sqliteGen/srcFiles/yeast2/Yeast_2.na27.annot.csv",
            metaDataSrc=metaDataSrc)


## End(Not run)
```

---

wrapBaseDBPackages *Wrap up all the Base Databases into Packages for distribution*

Description

Creates extremely simple packages from the base database files for distribution. This is a convenience function for wrapping up these packages in a consistent way each time.

Usage

```
wrapBaseDBPackages(dbPath, destDir, version)
```

Arguments

dbPath	dbPath is just the path to the location of the latest intermediate sqlite source files. These files are then used to make base DB packages.
destDir	destination path for the newly minted packages.
version	version number to stamp onto these newly minted packages.

Examples

```
## Not run:
  ##Make all of the intermediate DBs and place the new packages right here.
  wrapBaseDBPackages(dbPath = "/mnt/cpb_anno/mcarlson/proj/sqliteGen/nli/annosrc/db/",
                     destDir = ".")

## End(Not run)
```

Index