The *IdMappingRetrieval* package in Bioconductor: Collecting and caching identifier mappings from online sources.

Alex Lisovich †, Roger S. Day †‡

October 30, 2017

†Department of Biomedical Informatics, †‡ Department of Biostatistics,
University of Pittsburgh

1 Overview

Research which integrates data from multiple data platforms must of course merge on samples processed in parallel on the platforms. However, exploiting the full biological significance of the data depends on merging on the respective features as well. The features have platform-specific biological identifiers, so identifier mapping is critical to this merging. The IdMappingRetrieval package allows initial acquisition of biological identifier mappings (ID maps) from online bioinformatics services, with caching in localdata repositories for subsequent fast retrieval. An ID map is a one-to-many map from one ID type (called the primary key) to another (called the secondary key). The services currently supported are NetAffx and Ensembl. The package employs a unified interface for accessing these services, so that the local repositories will be easy to create, update, and use. Aside from identifier maps themselves, the the service's complete annotation data sets are also accessible through the same mechanism. . Therefore, although ID mapping is the primary goal, secondarily the package performs as a generic annotation data collection tool if desired.

The objects produced by this package are specifically suited for use by the package *IdMappingAnalysis*, currently in preparation for Bioconductor. The purpose of *IdMappingAnalysis* is to characterize and compare two or more ID maps.

Retrieval is typically subsetted by Affymetrix array.

2 The package architecture

2.1 Two types of services: query-based and file-based

The IdMappingRetrieval package retrieves data using service objects. The package supports two types of services, query-based and file-based. The query-based services support automated retrieval either through a standard service API, through third-party R packages, or through simulated interaction with service web pages. The file-based services assist the manual download of raw data, followed by extraction of ID maps and other information. For diverse reasons they generally require more active user participation.

1. Query-based services

   (a) Affymetrix annotation file repository [1],
   Data source: http://www.bioconductor.org/packages/2.2/bioc/html/AffyCompatible.html.
   Service class name: AnnotationAffx.
   Access method: *AffyCompatible* Bioconductor package.
   Service class name: AnnotationEnsembl.
   Access method: *biomaRt* Bioconductor package.

(b) EnVision online query system [4], accessed using the SOAP-based Web Services.
Data source: **LINK NEEDED**.
Service class name: AnnotationEnvision
Access method: *ENVISIONQuery* Bioconductor package.

2. File-based services

(a) Affymetrix NetAffx batch system supporting data download in chunks [1].
Data source: http://www.affymetrix.com/analysis/index.affx.
Service class name: AnnotationNetxAffx.
Access method: See Appendix A.

(b) Ensembl download service [3,4].
Data source: http://uswest.ensembl.org.
Service class name: AnnotationEnsemblCsv..
Access method: See Appendix C.

2.2 User tasks, services, classes and objects

The package utilizes the R.oo framework for S3 class based development ([5], http://cran.r-project.org/web/packages/R.oo/index.html), which we found to be well suited for the task. As seen in the table above, different subclasses of the Annotation class support access to each specific ID map retrieval service.

1. For each service of interest:

(a) If necessary, set user credentials for login.

(b) Instantiate the service object as an instance of the Annotations subclass suitable for the service of interest,Specify the query parameters.

(c) Invoke a data collection method (either getIdList or getDataFrame) on the service object. or a set of objects each of which represents the particular service..

2. For a list of services:

(a) Loop over the services as above.

(b) Alternatively, for convenience, instantiate the ServiceManager and use corresponding methods.

Some parameters are common for all service object types.

- cacheFolderName - a subdirectory name where data for given service object will be cashed ("EnVision")
- primaryColumn and secondaryColumn -the column names which a service object uses to extract the Id Map from the raw data frame. In case primaryColumn/secondaryColumn contain multiple column names, the service object merges the corresponding raw data frame columns into one (service specific)
- swap - a boolean. If true, the primary/secondary columns will be swapped at the end of the data retrieval process. Swapping reverses the 'one to many' identifier relation
- species - the species the collected data should be subsetted on

The user can override default values during the service object's creation. The default values are typically suitable. For the file-based services, the constructor has an additional parameter:

- the location of the file(s) or the directory containing the input data.

Note that micro array type is not an attribute of a service object and is supplied as an argument to the data retrieval method, which allows to reuse the same service object for multiple micro array types data collection. It also allows to define micro array type interactively in case array type argument is equal to 'menu' (see help pages for details).

Class	Data Source	Query versus File-based
AnnotationAffx (based on AffyCompatible)	Affymetrix annotation file repository	Query
AnnotationNetAffx	Affymetrix NetAffx batch query system	File (Appendix A)
AnnotationEnsembl (based on biomaRt)	Ensembl	Query
AnnotationEnsemblCsv	Ensembl front end	File (Appendix C)
AnnotationEnvision (based on ENVISIONQuery)	EnVision Web Services	Query

2.3 The caching subsystem

Due to the large amount of data available for high density micro arrays, the process of online data retrieval takes significant time ranging from a few minutes for a "fast" service to up to an hour (as in case of EnVision Web Services). It would be impractical to retrieve the same data again and again at the beginning of each analysis session given the fact that the data in online repository typically get updated few times a year. The IdMappingRetrieval caching mechanism allows the system to store the retrieved data locally allowing to reuse them in subsequent processing sessions significantly (hundreds of times) increasing the retrieval speed. When request for data retrieval is received, the system checks if the data requested already stored in caching directory and initiates the online retrieval process only when data of interest is not available. The system can be forced to retrieve the data online even if present in caching subsystem thus allowing to update the cached data if necessary. The caching subsystem is implemented as a hierarchical set of folders within a single root directory which is created if necessary during the session setup (see Getting Started).

Under the root directory, there is a set of folders each corresponding to the data collected by a particular service. The service folder name is assigned during the particular service object creation and takes a default name if not specified ("Ensembl" for AnnotationEnsembl object etc.). It is possible to create multiple folders for a given service object type in case the folder name specified explicitly thus allowing to cache multiple instances of data from the same online resource. This feature allows to store data available from the same service at different moments allowing to investigate how the given data source was evolving over time (see Use Case 2).

For a given service folder, there is a set of subfolders each of which corresponds to a particular micro array name ("HG-U133_Plus_2" etc.). Each micro array type related subfolder contains a set of files storing the R data objects (.RData files). There are two types of objects which are stored under the micro array folder. The first is a (single) data frame containing the raw data retrieved from a particular service, possibly containing multiple columns each for a particular data attribute. The second is a set of data frames representing the mapping between the particular attribute pairs (Uniprot Accession to probeset ID mapping as an example) extracted from a raw data frame during the last step of the ID mapping information retrieval process. The "two tier" caching allows to further speed up the data retrieval process for a particular attribute pair eliminating the need to read the entire raw data frame, whose size can easily exceed few dozen MB.

2.4 The output data formats

2.4.1 Identifier mapping data format

The identifier mapping related information (ID Map) is represented by a data frame consisting of two columns in character format. The first (primary) column contains a unique set of identifiers of a

particular type (Uniprot accessions being an example) which represent the identifier mapping source. The second (secondary) column contains the results of mapping of a given (primary) identifier to the target identifier type (Affymetrix probeset ID as an example) obtained as a result of a query to the particular online service. As a single source identifier can map to multiple target identifiers, the secondary column elements are represented by comma separated strings.

The column names can be set during the service object initialization or can be deduced from the raw data if this information is not provided. In later case, the ID Map column names will depend on the particular service in use. If a further comparison of ID mapping performance for various online services is desirable, it is recommended to set the unified column names during the service objects initialization so to the nature of ID Map data ("Uniprot" and "Affy" for Uniprot accessions to Affymetrix probeset ID mapping as an example). In further writing, we will refer to the names of a first and second columns as primary and secondary keys correspondingly.

2.4.2 Complete raw data set format

The system always returns the complete raw data set as a data frame. As each particular online service uses it's own data format, the data frame format will be different for each particular service and the query type. If the detailed information on the data frame content is desirable, we refer user to the documentation from a particular online service provider.

3 Examples

3.1 Session initialization and setup

The data retrieval session always starts with the call to the package initialization function which tells the system which directory should be used for data caching. If specified directory does not exist yet, it will be created during the initialization process. As some online services require credentials (user ID and password) to grant user an access to the data, the next step of initialization process for a newly created caching directory should be setting the credentials for a service which requires them. At the moment, the only service directly requiring credentials is an Affymetrix query-based service (AnnotationNetAffx), however the rest of implemented services also utilizes some of the AnnotationAffx functionality (micro array name and/or probeset ID list retrieval) and therefore requires setting the same credentials during initialization process. Note that setting credentials is mandatory only for newly created caching directory and can be omitted in subsequent data retrieval sessions. The final step in initialization process is setting the verbosity level, defining the micro array type, mnemonic names for source and target identifier types (primary and secondary key values) as well as location of files and/or folders for file-based services. A typical initialization script for a data retrieval session is given below:

```
> library(IdMappingRetrieval)
> # setup verbosity level,array type and DB keys
> arrayType <- "HG-U133_Plus_2";
> primaryKey <- "Uniprot";
> secondaryKey <- "Affy";
> #initialize caching directory structure
> Annotation$init(directory="./annotationData",verbose=TRUE);
> #set Affymetrix credentials
> AnnotationAffx$setCredentials(
+    user="alex.lisovich@gmail.com",password="125438",verbose=TRUE);
```

3.2 Service object initialization

A few examples of different service objects initialization are given below.

```
> #create Affymetrix service object
> annAffx_Q <- tryCatch(
+   AnnotationAffx(cacheFolderName="Affymetrix",
+                  primaryColumn="Probe.Set.ID",
+                  secondaryColumn="SwissProt",
+                  swap=TRUE) );
> #create Ensembl file-based service object
> annEnsemblCsv <- tryCatch(
+   AnnotationEnsemblCsv(
+     cacheFolderName="EnsemblCsv",
+     primaryColumn=c("UniProt.SwissProt.Accession",
+                     "UniProt.TrEMBL.Accession"),
+     secondaryColumn=NA,
+     swap=FALSE, full.merge=TRUE,
+     df_filename="ENSEMBL_biomart_export.txt") );
> #create Envision service object
> annEnvision <- tryCatch(
+   AnnotationEnvision(
+     cacheFolderName="EnVision",
+     primaryColumn=c("UniProt.SwissProt.Accession",
+                     "UniProt.TrEMBL.Accession"),
+     secondaryColumn=NA,
+     swap=TRUE, species="Homo sapiens", full.merge=TRUE) );
> errorSummary = sapply(c('annAffx_Q', 'annEnsemblCsv', ''), function(service)
+   inherits(service, "error") )
> errorMesg = ifelse(any(errorSummary),
+       paste("(In this vignette build, the service(s) ",
+         names(which(errorSummary)), "timed out.)"),
+       errorMesg = "(All services were successfully retrieved.)"
+   )
```

3.3 Collecting ID maps from multiple services

Typically, the IdMappingRetrieval package is used in conjunction with the IdMappingAnalysis package which accepts the set of collected identifier mapping data in the form of IdMap object list . The steps involved in the process include:

- Defining array type and primary/secondary key values for a resulting IdMap object set *.
- Defining the location of a file or directory containing preloaded data for file-based services
- Creating a set of service retrieval objects, one per each particular service
- Retrieving the Id Map through the getIdMap() method or a complete raw data frame through the getDataFrame() method.

*The need to define the primary/secondary column names for resulting IdMap object set on top of primaryColumn/secondaryColumn arguments used during the service object creation arises from the fact that there is no standard naming convention for the raw data column names among the services. Therefore for a subsequent analysis of identifier mapping performance among the services we provide the arguments primaryKey and secondaryKey to getIdMap() and getIdMapList() methods for Annotation and ServiceManager objects respectively.

The script below illustrates the process of collecting such data from all services implemented within the IdMappingRetrieval package.

```
> # create service objects
> services <- list();
```

```
> services$NetAffx_F <- AnnotationAffx("Affymetrix");
> #collect Id Map data
> idMapList <- lapply(services, getIdMap,
+               arrayType=arrayType,
+               primaryKey=primaryKey,
+               secondaryKey=secondaryKey,
+               force=FALSE, verbose=TRUE);
>
```

This initial retrieval takes some time. But repeating the retrieval is fast due to caching.

```
> idMapList <- lapply(services, getIdMap,
+                  arrayType=arrayType,
+                  primaryKey=primaryKey,
+                  secondaryKey=secondaryKey,
+                  force=FALSE, verbose=TRUE);
> names(idMapList);
> idMapList$NetAffx_F[100:110,];
```

Instead of map objects, the results can be stored as data frames.

```
> #collect complete data frames
> dfList <- lapply(services, getDataFrame,
+                  arrayType=arrayType,
+                  force=FALSE, verbose=TRUE);
> names(dfList);
> names(dfList$NetAffx_F);
```

3.4 Collecting data from various services using service manager

An alternative way of achieving the same results is to use the *ServiceManager* object, a container for service objects. The *ServiceManager* class generates the full set of default services automatically. The following script repeats the work of the previous code, more compactly.

```
> #create service manager object containing set of default services
> svm <- ServiceManager(ServiceManager$getDefaultServices());
> names(getServices(svm));
> idMapList <- getIdMapList(svm,
+                         arrayType=arrayType,
+                         selection=c("NetAffx_Q"),
+                         primaryKey=primaryKey,
+                         secondaryKey=secondaryKey, verbose=TRUE);
> #collect complete data frames
> dfList <- getDataFrameList(svm,
+                         arrayType=arrayType,
+                         selection=c("NetAffx_Q"), verbose=TRUE);
```

3.5 Collecting data from various services interactively using service manager

If micro array type and/or the paths for file-based services are not defined (set to NULL), ServiceManager present user with dialog(s) allowing selecting it interactively. Note that in a code snippet below the set of default services is generated internally by invoking a static version of getIdMapList() and getDataFrameList().

```
> #collect complete IdMap data interactively
> #selecting array type and services to collect data from
> idMapList <- ServiceManager$getIdMapList(
+                    primaryKey= primaryKey,
+                    secondaryKey=secondaryKey,verbose=TRUE);
> #collect complete data frames interactively
> #selecting array type and services to collect data from
> dfList <- ServiceManager$getDataFrameList(
+                    arrayType="menu",
+                    selection="menu", verbose=TRUE);
```

4 Session information

This version of IdMappingRetrieval has been developed with R 2.11.0.

R session information:

```
> toLatex(sessionInfo())
```

- R version 3.4.2 (2017-09-28), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Loaded via a namespace (and not attached): compiler 3.4.2, tools 3.4.2

5 Acknowledgments

We would like to thank Rafael Jimenez from the Enfin EnCore team for an enjoyable discussion and the great effort he has put into expanding the ID mapping related EnCore web services to better suite our needs. We also would like to thank Himanshu Grover, a graduate student in Biomedical Informatics at the University of Pittsburgh, for the great effort and useful suggestions he has made during the package preparation and testing.

6 References

[1] Liu, G., NetAffx: Affymetrix probesets and annotations. Nucleic Acids Res. 2003, 31(1):82-6.

[2] Xos? M. Fern?ndez-Su?rez and Michael K. Schuster Using the Ensembl Genome Server to Browse Genomic Sequence Data. UNIT 1.15 in Current Protocols in Bioinformatics, Jun 2010. www.ncbi.nlm.nih.gov/pubmed/20521244

[3] Giulietta M Spudich and Xos? M Fern?ndez-Su?rez Touring Ensembl: A practical guide to genome browsing BMC Genomics 2010, 11:295 (11 May 2010)

[4] Florian Reisinger, Manuel Corpas, John Hancock, Henning Hermjakob, Ewan Birney and Pascal Kahlem. ENFIN - An Integrative Structure for Systems Biology. Data Integration in the Life Sciences Lecture Notes in Computer Science, 2008, Volume 5109/2008, 132-143, DOI: 10.1007/978-3-540-69828-9_13

[5] Henrik Bengtsson, The R.oo package - Object-oriented programming with references using standard R code, Talk at the DSC 2003 conference, Wienna, March 22, 2003.

Appendix A

Obtaining the Affymetrix ID Mapping data files using the NetAffx batch query system and HG-U133_Plus_2 array as an example.

1. Create a set of the files for upload containing the set of Affymetrix Probe Set IDs for a given array ensuring the number of IDs for each file does not exceed 10000 (NetAffx batch query limit):

   AnnotationNetAffx$createSubmission(
   chunk=10000,folder="./NetAffx_submission",verbose=TRUE);

2. Register on the Affymetrix Web site (www.affymetrix.com) if you do not have an account yet.

3. Select "NetAffx" item in the menu at the top of the page and then select "Batch Query" item in " 3' IVT Expression" tab on the NetAffx page. The "Batch Query" page will popup.

4. Create a custom annotation view which should include Probe Set ID as primary (first) identifier and SwissProt as a secondary if not done yet as follows: - Select "Custom Annotation Views in the leftmost panel of the page (NetAffx Analysis Center). The "Create/Edit Custom Views" page will popup. - Enter the custom view name ("Affy2Uniprot" for example) and select Probe Set ID and SwissProt items in "Select Fields for your view" list. Click Submit. - Go back to the "Batch Query" page. Select "Human Genome U133 Plus 2.0 array" and "Probe Set ID" in "Select a GeneChip Array" and "Select the search type" list boxes correspondingly.

5. Select the first upload file from the set created at step 1 using "Browse" button and click "Submit". The "Show Results" page will popup.

6. Click on "Export Results" item. The "Export Center" page will popup. Select "Tab Separated Value" and "Affy2Uniprot" in "Select File Format" and "Select a View" list boxes correspondingly. Click "Submit" and then save the exported file when prompted.

7. Repeat steps 3, 5 and 6 for files created at step 1. Copy the set of resulting .tsv files into directory of your choice and use them when working with AnnotationNetAffx service retrieval object.

8. After downloading the file, the ID Map data can be retrieved by creating the AnnotationNetAffx object with df_filename slot containing the path to the downloaded file directory (or the character vector of paths to file set) and then calling Annotation.getIdMap() on this object. If df_filename is not specified (NULL by default) the File Open dialog will popup during the data retrieval allowing to specify the file set interactively.

Appendix B

Obtaining the Ensembl ID Mapping data file using the HG-U133_Plus_2 array as an example.

1. Open the Ensembl main page (www.ensembl.org) and then select "BioMart" item from the menu at the top.

2. Choose “Ensembl Genes 60” as a database and select “Homo sapience genes (GRCh37.p2)” as a data set.

3. Define the attributes by clicking on the “Attributes” item in the left panel and then check the following items in the right panel: - “GENE/ Ensembl Gene ID”, - “External References/ UniProt/TrEMBL Accession” , - “External References/ UniProt/SwissProt Accession” and - “Microarray/ Affy HG U133-PLUS-2” Click on “Results” button above the left panel. The Results will be displayed in the right panel.

4. In the Results panel, set “Export all results to” as “File” and “CSV” and check the box “Unique Results only”. Press “GO” and then save the file when prompted.

5. After downloading the file, the ID Map data can be retrieved by creating the AnnotationEnsembleCsv object with df_filename slot containing the path to the downloaded file and then calling Annotation.getIdMap() on this object. If df_filename is not specified (NULL by default) the File Open dialog will pop up during the data retrieval allowing to specify the file location interactively.