RNCBI Manual

Martin Schumann

June 22, 2010

Contents

1 Introduction

This document is going describe some examples, which use the ***RNCBI*** package (0.9). Please visit the project page for information, discussions etc.

If you would like to learn more about the java part of the package, please take a look at the JavaDoc or the source code. To make a long story short, the java part uses the Axis2 libraries from the apache project `http://ws.apache.org/axis2/` to send requests to the NCBI webservice and to retrieve the results respectively.

It is important to mention, that this package was built using the **S4** class system to provide object-oriented programming. This means that mostly every function called will return a modified object of the one that was provided as argument, because the arguments are passed by value. The examples below will highlight this fact.

2 Installation

The following packages are required to install ***RNCBI***:

rJava 0.8-3 This library provides the access to the java part of this package. See `http://www.rforge.net/rJava/` for installation files and infos.

XML 2.8-1 The XML library for R. See `http://cran.r-project.org/web/packages/XML/index.html` for installation files and infos.

3 Usage

The first step is to initialize the interface to the java part. In this step you can provide extra information to the interface, like a proxy server or whether to debug or not. In case of the EFetch operations for each database, the results will contain a lot of empty entries. By default these entries will be removed, but this can be disabled by providing `tidy=FALSE` as argument.

```
> library(RNCBI)
> ncbi <- NCBI()
```

If you have set the proxy settings in your R environment, then this call to the `NCBI()` function is sufficient. The program will use the `Sys.getenv("http_proxy")` to get your settings. If there are any problems with that or you additionally have to provide user and password to your proxy, then these settings can be set manually.

```
> library(RNCBI)
> ncbi <- NCBI(proxy = "http://host:port", debug = FALSE,
+     tidy = TRUE)
> setProxyUser(user = "user", password = "password")
```

This will set the proxy to "http://host:port", debug to `false` and tidy to `true`. Which means, that the program should use the given proxy, it should not debug and it should remove empty entries. With the last step the user and the password for the proxy will be set. The password can be left empty, if no password is required.

All these steps have to be done to use the NCBI webservice. After this, each operation of the webservice can be run. The package provides some useful functions to get an overview of the available operations of the NCBI webservice. Because each operation from the webservice expects different parameters, another function to get a list with these parameters exists. All these functions expect the `ncbi` object as first argument, because this contains the information to connect to the java interface.

```
> getOperations(ncbi, printNames = FALSE)
```

```
     [,1]
[1,] "run_eGquery"
[2,] "run_eInfo"
[3,] "run_eSearch"
[4,] "run_eSummary"
[5,] "run_eLink"
[6,] "run_eSpell"
[7,] "run_ePost"
[8,] "run_eFetch"
```

```
> getEFetchDatabases(ncbi)
```

```
[1] "1 gene"
[1] "2 journals"
[1] "3 nlmc"
[1] "4 omim"
[1] "5 pubmed"
[1] "6 sequence"
[1] "7 snp"
[1] "8 taxon"
```

```
> getRequestParameter(ncbi, "egquery")
```

```
[1] "term"  "tool"  "email"
```

- The `getOperations` function will return the names of the **EUtils** operations by default. If provided with the parameter `printNames=FALSE` a vector will be returned.
- The `getEFetchDatabases` function will return the names of the **EFetch** databases which are available by default. It will also return a vector, if `printNames=FALSE` is given as argument.
- The `getRequestParameter` function will return the names of the parameter, which have to be set, to send a request to the NCBI webservice. The second argument is the name of the operation, for which the request parameter should be returned.

3.1 EUtils operations

After the initialization step, the operations can be called. In case of the **EUtils** the user has to create a new object for each operation.

```
> egquery <- EGQuery(ncbi)
```

With this `egquery` object the user now can set the request parameter. There are two ways to do this:

1. With the help of the `setRequestParameter` function. This takes the initiated object of the webservice operation, the parameter name to set and the value for the parameter.
2. Get the list of request parameter from the operation object and set the parameter manually.

```
> str(egquery@request)
```

```
List of 3
 $ term : NULL
 $ tool : NULL
 $ email: NULL
```

```
> egquery <- setRequestParameter(egquery, "term", "mouse")
> egquery@request$term <- "mouse"
> str(egquery@request)
```

```
List of 3
 $ term : chr "mouse"
 $ tool : NULL
 $ email: NULL
```

The `str` function is only used to make the output a little bit more consolidated. After setting the request parameter, the request can be sent to the NCBI webservice. To do this, every operation has a request method like `requestEGQuery`, which takes the operation object as argument and returns this object with the results from the webservice.

```
> egquery <- requestEGQuery(egquery)
```

If this doesn't produce any errors or warnings, then the results are in the "results" list of the operation object. This list can be accessed directly through the operation object or it can be get using the `getResults` method, which takes an operation object as argument and returns its results list.

```
> results <- egquery@results
> results <- getResults(egquery)
> names(results)
```

```
[1] "term"       "resultitem"
```

The results list is in most cases a very complex list of lists. To get an overview of the returned parameters and their values, the `str` function is a very helpfully tool.

```
> str(results)
```

In this case it is a long list of result items, which contains information about the appearance of the term "mouse" in every database of the NCBI webservice.

```
> str(results$resultitem$resultitem[[1]])
```

```
List of 4
 $ count : chr "1062582"
 $ status: chr "Ok"
 $ menu  : chr "PubMed"
 $ dbname: chr "pubmed"
```

The first "resultitem" indicates the parameter name in the result from the NCBI webservice. The second indicates, that this is an array of items of the type "resultitem". Each item in the array is a "resultitem", which in turn consists of four further parameter "count", "status", "menu" and "dbname". To search through this I would suggest a simple for loop over the elements of the "resultitem" array.

```
> for (i in 1:length(results$resultitem$resultitem)) {
+     tmp <- results$resultitem$resultitem[[i]]
+     if (tmp$count < 15) {
+         print(tmp$dbname)
+     }
+ }
```

```
[1] "pubmed"
[1] "journals"
[1] "taxonomy"
[1] "snp"
[1] "cancerchromosomes"
[1] "pcsubstance"
[1] "gap"
```

This example loop shows the database names, that contains the search term less than 15 times. You can modify this loop to get other results. Another possibilty is to convert the "resultitem" array into a dataframe, which provides better access to each element.

```
> x <- do.call(rbind, lapply(results$resultitem$resultitem,
+     unlist))
> resultitem <- data.frame(x, stringsAsFactors = FALSE,
+     row.names = NULL)
> head(resultitem)
```

```
    count status     menu   dbname
1 1062582     Ok   PubMed   pubmed
2  308823     Ok      PMC      pmc
3       1     Ok Journals journals
4    8625     Ok     MeSH     mesh
5    4574     Ok    Books    books
6    8562     Ok     OMIM     omim
```

The dataframe now contains the information from all the resultitems.

```
> resultitem[resultitem$status != "Ok", ]
```

```
[1] count  status menu   dbname
<0 rows> (or 0-length row.names)
```

This example checks whether the term or a database was not found.

3.2 EFetch operations

As there are distinct services for the EFetch operation depending on the database, the initialization step is a little bit different from the EUtils. To get any information about the EFetch operation, the operation object has to be initiated.

```
> efetch <- EFetch(ncbi, "pubmed")
> str(getRequestParameter(ncbi, "efetch"))
```

```
 chr [1:8] "id" "retstart" "webenv" "query_key" "email" ...
```

The "ncbi" object is the same we initiated before (see 3). If you call the `getRequestParameter` function before initiating the EFetch object, then this will return an error with the description, that an EFetch database has to be set first. After the EFetch object was initiated, the request parameters are availabe and can be set like before (see 3.1).

```
> efetch <- setRequestParameter(efetch, "id", c(12091962,
+     9997))
> efetch <- requestEFetch(efetch)
> results <- getResults(efetch)
```

Every EFetch operation can take mulltiple IDs as arguments. The NCBI webservice handles multiple IDs in most cases as a simple string, that contains the IDs separated by comma. But the EUtils operation "ELink" expects an array of IDs in the request. Because of this difference, multiple IDs have to be provided as vector to the interface. The java part will take care whether to generate an array or a simple string.

The results of this EFetch request will be a very complex list, but again you can use the `str` function to get an overview of the content.

3.3 Further possibilties

This package should give the user the opportunity to use the results at their own will as much as possible. Furthermore the results are very dynamic, because they will come from a webservice. Which means that the result depends on the query. To realize this, all the results are returned as a list, because lists are the most flexible data structure in R. Furthermore it is possible to retrieve the xml document, which was passed from the java interface to R, but only for the EFetch operations.

```
> xml <- efetch@xmldoc
```

The xml document is located in the "xmldoc" slot of the EFetch operation object. It is an XMLNodeList and can be parsed like any other xml document. The structure is very simple and only indicates arrays and objects. An array is a collection of the some objects. And an object contains either another object or an entry. All the xml tags hold a name attribute and the array tags additionally holds a length attribute.

4 Examples from the NCBI website

The following examples are taken from the ncbi website Examples. For further information about each operation please take a look at Chapter 4 of this book.

The ncbi object, which was created at the beginning (see 3) is still required at this point. The object only has to be created once in a session and can be used to call all operations from the NCBI webservice.

4.1 Call EGQuery

As seen in the Usage section (see 3).

4.2 Call EInfo

This example simply calls the EInfo operation with no parameters to get a list of database names, but first we have to create an `einfo` object.

```
> einfo <- EInfo(ncbi)
> einfo <- requestEInfo(einfo)
> str(getResults(einfo))
```

```
List of 1
 $ DBList: chr [1:43] "pubmed" "protein" "nucleotide" "nuccore" ...
```

Now the `einfo` object can be used to set a database name to the request slot and to do a new request to the webservice. The operation object can be reused for this second task, because there were no request parameter set.

```
> einfo <- setRequestParameter(einfo, "db", "pubmed")
> einfo <- requestEInfo(einfo)
> results <- getResults(einfo)
> names(results)
```

```
[1] "lastupdate"  "fieldlist"   "description" "dbname"
[5] "linklist"    "menuname"
```

The results provide the statistics for the "pubmed" database, including a list of indexing fields ("fieldlist") and available link names ("linklist"). These results can be inspected more by converting the "linklist" and the "fieldlist" into a data frame.

```
> results$description

[1] "PubMed bibliographic record"

> results$lastupdate

[1] "2010/06/22 04:57"

> results$dbname

[1] "pubmed"

> results$menuname

[1] "PubMed"

> x <- do.call(cbind, lapply(results$linklist, unlist))
> linklist <- data.frame(x, stringsAsFactors = FALSE)
> linklist[linklist$dbto == "gene", 2:4]

                   menu                    name dbto
8            Gene Links             pubmed_gene gene
9     Gene (OMIM) Links pubmed_gene_citedinomim gene
10 Gene (GeneRIF) Links         pubmed_gene_rif gene

> x <- do.call(cbind, lapply(results$fieldlist, unlist))
> fieldlist <- data.frame(x, stringsAsFactors = FALSE)
> head(fieldlist[1:5, c(3, 4, 7, 8)])

                          description name termcount   fullname
1  All terms from all searchable fields  ALL  88415905 All Fields
2 Unique number assigned to publication  UID         0        UID
3                   Limits the records FILT      3852     Filter
4         Words in title of publication TITL  12360244      Title
5 Free text associated with publication WORD  38878957  Text Word
```

The output is reduced, as you can see. These are all the entries, which link to the gene database and the first six entries from the "fieldlist".

4.3 Call ELink

This example retrieves IDs from Nucleotide for GI 48819, 7140345 to Protein. First create an `elink` object and set the request parameter.

```
> elink <- ELink(ncbi)
> elink <- setRequestParameter(elink, "db", "protein")
> elink <- setRequestParameter(elink, "dbfrom", "nuccore")
> elink <- setRequestParameter(elink, "id", c(48819, 7140345))
> elink <- requestELink(elink)
> results <- getResults(elink)
> names(results$linkset$linkset[1]$linkset)
```

```
[1] "dbfrom"           "webenv"           "idchecklist"
[4] "linksetdbhistory" "linksetdb"        "idlist"
[7] "urlidlist"
```

After setting the request parameter and sending the request to the webservice, the results can be stored. The results will contain all elements the NCBI webservice provides, even if the element is not set. In this case, the element contains "empty". Again the result contains an array of two "linkset" items, as we asked for two IDs. Each of these "linkset" items contain the further subelements for each ID. Altough there are two "linkset" items, only the second item contains the two "linksetdb" items, we are looking for. These two "linksetdb" items contain a list of IDs and they can be retrieved with the following steps.

```
> lset <- results$linkset$linkset[2]$linkset
> firstLSetDb <- lset$linksetdb[1]$linksetdb
> secondLSetDb <- lset$linksetdb[2]$linksetdb
> x <- do.call(rbind, lapply(firstLSetDb$link, unlist))
> firstLink <- data.frame(x, stringsAsFactors = FALSE,
+     row.names = NULL)
> firstLSetDb$linkname
```

```
[1] "nuccore_protein"
```

```
> head(firstLink)
```

```
         id score
1 297307291 empty
```

```
2 297307290 empty
3  77157738 empty
4  77157737 empty
5  38638803 empty
6  37515183 empty
```

```
> x <- do.call(rbind, lapply(secondLSetDb$link, unlist))
> secondLink <- data.frame(x, stringsAsFactors = FALSE,
+     row.names = NULL)
> secondLSetDb$linkname
```

```
[1] "nuccore_protein_cds"
```

```
> head(secondLink)
```

```
        id score
1 16950486 empty
2 16950485 empty
3 15145457 empty
4 15145456 empty
5 15145455 empty
6  7331953 empty
```

These are the IDs we were looking for. The results of this operation depends heavily on the request sent. For all the other tasks, the ELink operation can be used, please take a look at the results with the `str` function first to get an overview.

4.4 Call EPost

This operation puts a list of IDs to the history for later use.

```
> epost <- EPost(ncbi)
> epost <- setRequestParameter(epost, "db", "pubmed")
> epost <- setRequestParameter(epost, "id", c(123, 456,
+     37281, 983621))
> epost <- requestEPost(epost)
> epostResults <- getResults(epost)
> epostResults$webenv
```

```
[1] "NCID_1_11705055_130.14.22.28_9001_1277218547_121407080"
```

```
> epostResults$querykey
```

```
[1] "1"
```

This "webenv" and "querykey" can be used to request information from the ELink operation for example.

```
> elink <- ELink(ncbi)
> elink <- setRequestParameter(elink, "query_key",
+                 epostResults$querykey)
> elink <- setRequestParameter(elink, "webenv",
+                 epostResults$webenv)
> elink <- requestELink(elink)
> elinkResults <- getResults(elink)
```

We have to create a new `elink` object, because the old request parameters are still there. This result has the correct structure and beside the fact that it contains about 400 IDs, it can be converted into a dataframe like this:

```
> linkset <- elinkResults$linkset$linkset$linkset
> linksetdb <- linkset$linksetdb$linksetdb
> length(linksetdb$link)
```

```
[1] 420
```

```
> x <- do.call(rbind, lapply(linksetdb$link, unlist))
> link <- data.frame(x,stringsAsFactors=FALSE, row.names=NULL)
> head(link)
```

```
       id score
1  983621 empty
2   37281 empty
3     456 empty
4     123 empty
5 6400388 empty
6 1140622 empty
```

4.5 Call ESearch

This examples searches in the PubMed Central database for stem cells in free fulltext articles.

```
> esearch <- ESearch(ncbi)
> esearch <- setRequestParameter(esearch, "db", "pmc")
```

```
> esearch <- setRequestParameter(esearch, "term",
+                 "stem+cells+AND+free+fulltext[filter]")
> esearch <- setRequestParameter(esearch, "retmax", 15)
> esearch <- requestESearch(esearch)
> results <- getResults(esearch)
> results$count
```

```
[1] "40859"
```

```
> x <- do.call(cbind, lapply(results$idlist, unlist))
> ids <- data.frame(x, stringsAsFactors=FALSE, row.names=NULL)
> head(ids)
```

```
       id
1 2693088
2 2692479
3 2628721
4 2745366
5 2730033
6 2575525
```

Again we convert the "idlist" into a dataframe for better access. The results list contains several more information about the "term" that was sent to the webservice. Detailed informations can be found in the "`results$translationstack`" sublist.

4.6 Call ESpell

This example retrieves spelling suggestions.

```
> espell <- ESpell(ncbi)
> espell <- setRequestParameter(espell, "db", "pubmed")
> espell <- setRequestParameter(espell, "term", "mouss")
> espell <- requestESpell(espell)
> results <- getResults(espell)
> results$query
```

```
[1] "mouss"
```

```
> results$correctedquery
```

```
[1] "mouse"
```

4.7 Call ESummary

This example retrieves document summaries by a list of primary IDs.

```
> esummary <- ESummary(ncbi)
> esummary <- setRequestParameter(esummary, "db", "nucleotide")
> esummary <- setRequestParameter(esummary, "id", c(28864546,
+     28800981))
> esummary <- requestESummary(esummary)
> results <- getResults(esummary)
```

The results contain two "docsums" and each of them contain a list of "items". This result can be converted to a dataframe again.

```
> firstDocSum <- results$docsum$docsum[1]$docsum
> x <- do.call(rbind, lapply(firstDocSum$item, unlist))
> items <- data.frame(x, stringsAsFactors = FALSE, row.names = NULL)
> firstDocSum$id
```

```
[1] "28864546"
```

```
> items[c(1, 3:5), ]
```

```
        name                            itemcontent  item    type
1    Caption                               AY207443 empty  String
3      Extra gi|28864546|gb|AY207443.1|[28864546] empty  String
4         Gi                               28864546 empty Integer
5 CreateDate                            2003/03/05 empty  String
```

This shows only a selected part of the result. The second "docsum" can be retrieved the same way.

4.8 Call EFetch

This example fetches a record from the taxonomy database. The EFetch operation is a little bit different from the other EUtils operations.

```
> getEFetchDatabases(ncbi)
```

```
[1] "1 gene"
[1] "2 journals"
```

```
[1] "3 nlmc"
[1] "4 omim"
[1] "5 pubmed"
[1] "6 sequence"
[1] "7 snp"
[1] "8 taxon"
```

```
> efetch <- EFetch(ncbi, "taxon")
> getRequestParameter(ncbi, "efetch")
```

```
[1] "id"        "webenv"    "query_key" "email"     "tool"
[6] "report"
```

First we have to find the name for the database. The taxonomy database is available as "taxon" in the interface. So we initiate the `efetch` object with the database name. Now we are able to get the request parameter, if necessary. After that we can set the request parameter.

```
> efetch <- setRequestParameter(efetch, "id", c(9685, 522328))
> efetch <- requestEFetch(efetch)
> results <- getResults(efetch)
```

After setting the request parameter, we make a request to the NCBI webservice. The results contains a "taxaset", which in turn contains two "taxon" elements. We only want to inspect the first "taxon" item a little more.

```
> firstTaxon <- results$taxaset$taxaset$taxon[1]$taxon
```

First we create a dataframe for the "othernames" items.

```
> other <- firstTaxon$othernames
> x <- do.call(rbind, lapply(other$othernamestypechoice_type0,
+     unlist))
> othernames <- data.frame(x, stringsAsFactors = FALSE,
+     row.names = NULL)
> othernames
```

```
                synonym
1 Felis silvestris catus
2       Felis domesticus
3                    cat
4                   cats
5             Korat cats
```

```
> firstTaxon$othernames$genbankcommonname
```

```
[1] "domestic cat"
```

```
> firstTaxon$scientificname
```

```
[1] "Felis catus"
```

We use the some method like many times before to create the dataframe. Now it is time to get a list of the "lineageex". Again we create a dataframe for this.

```
> lineage <- firstTaxon$lineageex$taxon
> x <- do.call(rbind, lapply(lineage, unlist))
> lin <- data.frame(x, stringsAsFactors = FALSE, row.names = NULL)
> head(lin)
```

```
          rank      scientificname  taxid
1      no rank  cellular organisms 131567
2 superkingdom           Eukaryota   2759
3      no rank Fungi/Metazoa group  33154
4      kingdom             Metazoa  33208
5      no rank           Eumetazoa   6072
6      no rank           Bilateria  33213
```

The second "taxon" item can be handled the same way. For further information take a look at the names of each taxon.

```
> names(firstTaxon)
```

```
 [1] "pubdate"         "lineage"         "createdate"
 [4] "parenttaxid"     "rank"            "division"
 [7] "geneticcode"     "lineageex"       "updatedate"
[10] "mitogeneticcode" "scientificname"  "othernames"
[13] "taxid"
```

4.9 Search, Link & Fetch example

This is an example which uses three operations from the NCBI webservice to get a result. The first step is to search in the PubMed database for "cat".

```
> esearch <- ESearch(ncbi)
> esearch <- setRequestParameter(esearch, "db", "pubmed")
> esearch <- setRequestParameter(esearch, "term",
+                 "cat+AND+pubmed_nuccore[sb]")
> esearch <- setRequestParameter(esearch, "sort", "PublicationDate")
> esearch <- setRequestParameter(esearch, "retmax", 5)
> esearch <- requestESearch(esearch)
> esearchRes <- esearch@results
> x <- do.call(cbind, lapply(esearchRes$idlist, unlist))
> idlist <- data.frame(x, stringsAsFactors=FALSE, row.names=NULL)
> idlist
```

```
        id
1 19760058
2 19784554
3 20406687
4 20304455
5 20172041
```

```
> esearchRes$count
```

```
[1] "2558"
```

Like before (see 4.5), we create a dataframe from the "idlist". Now we can proceed to the second step. The second step retrieves links from the nucleotide database for the IDs we previously fetched.

```
> elink <- ELink(ncbi)
> elink <- setRequestParameter(elink, "db", "nuccore")
> elink <- setRequestParameter(elink, "dbfrom", "pubmed")
> elink <- setRequestParameter(elink, "id", idlist$id)
> elink <- requestELink(elink)
> elinkRes <- getResults(elink)
> linksetAr <- elinkRes$linkset$linkset
```

As all the operations accept a vector for the IDs, we simply pass the "id" column from the "idlist". Now we have to get the UIDs from the result. This can be done with a simple for loop.

```
> UIDVec <- NULL
> for (i in 1:length(linksetAr)) {
+     lsetdb <- linksetAr[i]$linkset$linksetdb$linksetdb
+     idAr <- lsetdb$link
+     x <- do.call(rbind, lapply(idAr, unlist))
+     UIDList <- data.frame(x, stringsAsFactors = FALSE,
```

```
+         row.names = NULL)
+     UIDVec <- c(UIDVec, UIDList$id)
+ }
```

In the loop every first element of the ”linksetdb” item from each ”linkset” is used, like in the example. Now we got a vector with the UIDs. With this we proceed to the third step and fetch the records from the nuccore database. As we know from the NCBI website, the nuccore database is contained in the sequence database, so we select this one and set the nuccore database as request parameter.

```
> getEFetchDatabases(ncbi)
```

```
[1] "1 gene"
[1] "2 journals"
[1] "3 nlmc"
[1] "4 omim"
[1] "5 pubmed"
[1] "6 sequence"
[1] "7 snp"
[1] "8 taxon"
```

```
> efetch <- EFetch(ncbi, "sequence")
> getRequestParameter(ncbi, "efetch")
```

```
 [1] "db"         "webenv"     "complexity" "tool"       "strand"
 [6] "seq_stop"   "rettype"    "id"         "retstart"   "seq_start"
[11] "email"      "query_key"  "retmax"     "report"
```

```
> efetch <- setRequestParameter(efetch, "db", "nuccore")
> efetch <- setRequestParameter(efetch, "id", UIDVec)
> efetch <- requestEFetch(efetch)
> efetchRes <- getResults(efetch)
> gbset <- efetchRes$gbset$gbset
> gbsetsequence <- gbset$gbsetsequence
> org <- NULL
> loc <- NULL
> def <- NULL
> for (i in 1:length(gbsetsequence)) {
+     gbseq <- gbsetsequence[i]$gbsetsequence$gbseq
+     org <- c(org, gbseq$gbseq_organism)
+     loc <- c(loc, gbseq$gbseq_locus)
+     def <- c(def, gbseq$gbseq_definition)
+ }
> resultDF <- data.frame(org, loc, def, stringsAsFactors = FALSE,
+     row.names = NULL)
> head(resultDF[, 1:2])
```

```
                       org      loc
1      Emericella nidulans EU734183
2        Aspergillus niger FJ979866
3      synthetic construct GU441535
4      Felid herpesvirus 1 FJ478159
5 Acanthopagrus schlegelii GU799605
6 Acanthopagrus schlegelii GU370345
```

I would suggest to not print the efetchRes object. Instead we proceed with it. We simply loop over "gbseq" and append each entry to the corresponding vector. After that, we put those vectors into a dataframe. The output only shows the first two columns, as the third won't fit on the page. Certainly there are more efficient ways to retrieve the data.

4.10 Using WebEnv & QueryKey example

This example uses WebEnv and QueryKey to retrieve information from the EFetch Pubmed database. First we search for "cat" with the ESearch operation.

```
> esearch <- ESearch(ncbi)
> esearch <- setRequestParameter(esearch, "db", "pubmed")
> esearch <- setRequestParameter(esearch, "term", "cat")
> esearch <- setRequestParameter(esearch, "sort", "PublicationDate")
> esearch <- setRequestParameter(esearch, "usehistory",
+     "y")
> esearch <- requestESearch(esearch)
> esearchRes <- getResults(esearch)
> webenv <- esearchRes$webenv
> querykey <- esearchRes$querykey
```

Now as we have the wanted information, we proceed to the second step and use the "webenv" and "querykey" to fetch five records from the EFetch Pubmed database starting with record ten.

```
> efetch <- EFetch(ncbi, "pubmed")
> getRequestParameter(ncbi, "efetch")
```

```
[1] "id"        "retstart"  "webenv"    "query_key" "email"
[6] "tool"      "retmax"    "rettype"
```

```
> efetch <- setRequestParameter(efetch, "webenv", webenv)
> efetch <- setRequestParameter(efetch, "query_key", querykey)
> efetch <- setRequestParameter(efetch, "retstart", 10)
> efetch <- setRequestParameter(efetch, "retmax", 5)
```

```
> efetch <- requestEFetch(efetch)
> efetchRes <- getResults(efetch)
```

Now this result can be further inspected. We use the same method as before and write a loop, put the information into a vector and create a dataframe out of these vectors.

```
> artSet <- efetchRes$pubmedarticleset
> pubmedArticleAr <- artSet$pubmedarticleset$pubmedarticle
> pmid <- NULL
> title <- NULL
> abstract <- NULL
> for (i in 1:length(pubmedArticleAr)) {
+     medLCit <- pubmedArticleAr[i]$pubmedarticle$medlinecitation
+     pmid <- c(pmid, medLCit$pmid)
+     title <- c(title, medLCit$article$articletitle)
+     abstract <- c(abstract, medLCit$article$abstract$abstracttext)
+ }
> resultDF <- data.frame(pmid, title, abstract, stringsAsFactors = FALSE,
+     row.names = NULL)
> resultDF[, 1]
```

```
[1] "20452121" "20447766" "20363123" "20236822" "20398794"
```

The columns ”title” and ”abstract” of the dataframe are way to long to print them here, so feel free to reproduce this example.