This document contains specific workflow instructions on generating new NARCIS-websiteplots. It is meant as supplementary to “Reharvest en updaten websitegrafieken algemene opzet\_RB+EB” (in Dutch), read that one first.

## Requirements

This workflow was written for execution in a mac OSX environment, using R-scripts.

You can download R [here](https://cran.rstudio.com/).

Working with R is easiest in RStudio (Rstudio.org for information, or download [here](https://www.rstudio.com/products/rstudio/download/#download)). It was written using R version 3.4.2, but no problems are foreseen using another R version.

The files described in this document can be found on Emil’s Github account, <https://github.com/EmilBode/NARCIS-analysis>

Note that there are multiple branches here. I’ll try to keep the branch ‘slimdcplot’ for this workflow, it’s a bit trimmed down, and used parameters will be more likely to be what you need.

When starting the workflow, it is recommended to look around the needed files to see what’s in them, to get a feel for them.

Not all files are needed, I’ll list the ones used here, and mark them. H means it is needed for a new harvest, P for plotting and generating websitecode, and HP means needed for both.

* SetLocal.R (HP) is a general file which specifies where files can be found.  
  SetLocal template.R is a template that can be used.
* Help.R (HP) is a file declaring helper-functions. The following functions with signatures are used:
  + libinstandload(..., order=F, quiet=T) (HP)
  + readNARCIScla(FilePath=paste0(Paths$Params,'/classification\_en.pdf')) (H)
  + ReadForAnalysisFromTotal (Summarize=FALSE, FilePath='Auto', set='auto', ForceReload=F, silent=F, DropCols=NULL, KeepCols=NULL, KeepSets=NULL, KeepMulti=NULL)
* VSNU-pdf.R reads some pdf containing counts by the VSNU, needed to calculate totals (P)
* OAI Harvest New Records.R (H) is the main script for harvesting files (step 3 in general setup)
* Plotjes.R (P) is the script for turning a dataset into plots.
* Parameter files in a folder:
  + Classification\_en.pdf (H) contains the NARCIS classification scheme. It’s mostly experimental, but some classification is done when keywords match with the classification. This is good as it is.
  + VSNU.rds is a dataset with counts/official statistics by the VSNU. It is a somewhat transformed version of this: <http://www.vsnu.nl/files/documenten/Feiten_en_Cijfers/Inzet_en_output_2015.xlsx>
  + WaardeOmzet.csv (H) contains mappings for adding information to the harvest.   
    They all work in the same way: If in “FieldFrom” a value in “ValFrom” is encountered, in “FieldTo” the value in “ValTo” is filled in.  
    So if in originURL in the harvest the value ‘https://pure.amc.nl/ws/oai’ is seen, it means this record is from the UvA, so in Bron, UvA is fiiled in.

This file needs updates once in a while, because new fieldvalues are used. Mostly because new repositories are added, but also because new publication types are introduced.  
Current fields are:

* + - harvestURL to Bron and BronType. What we get from NARCIS are urls the NARCIS-harvester uses, this needs to be mapped to an institution/university name. Needs to be updated once in a while, because repo’s move. I leave the old values there, but this is not necessary.
    - Type: something formatted like “info:eu-repo....” to a shorter, more readable label.
    - Languagelabel according to RFC 1766/ISO 639 to a readable label (for printing)
  + TypeCats.csv (P) is for classify more broadly in publication types. It is used for the “Open access voor verschillende publicatietypen” plot.
  + PubWeigh (P) is used for the calculation of number of articles. The idea is that publications are always part of “scientific output” as used by the VSNU, never, or partially.   
    If you don’t know what to do with it, leave it. Or, if new publication types are added, fill them in here and add the same number as the publication type it mmost looks like.
  + UniNaamOmzet.csv (P) is a mapping of the university names as we use, to those used by the VSNU.
  + GglLabels.csv (P) specifies the labels and texts to use with the plots. The plotnr is the number as used by the script, where the current plots are 7, 2, 22 and 14 (in that order), and number 99 is reserved for a general text at the bottom. The column “Omschr” is only used for comments, but if you want to include more comments, fill in an unknown value in language. Also note that for the extra text (nr 99), the header is ignored.
  + PlotKleuren.csv (P) is a relic, it specifies colors for some cases. Not used a lot.
* Besides these files from github, you need some more input:
  + Html-files from the website (P), to merge this file with the new graphs, saved locally (in the folder your specified in SetLocal.R, in Paths$input), called “GoogleChartsCode Origen.htm” en “GoogleChartsCode Orignl.htm”. This is simply the source code of <https://www.narcis.nl/metrics/Language/en> and <https://www.narcis.nl/metrics/Language/nl>
    - No files with the name equal to the output files. Otherwise these are used as a template.
  + (H) If you want to speed up the harvest process: 2 older files, a list of a previous harvest, along with a list of all old identifiers from a previous harvest, called Total dc (…).rds and IDlist dc (…).rds.
    - If you don’t have this: no problem, it just takes some more time.
  + Files with counts from the VSNU. From a coding point of view, they are more like parameters, but they are NOT public. I can distribute them in DANS, but not on GitHub. They are called *Narcis versus KUOZ 1.pdf*  and Narcis versus *KUOZ 2 uitgebreid.pdf*, and should be put in the *Paths$input* folder (along with the html-files)
* Regarding folder structure: originally I was working with 3 locations, but you can merge some of it if you want. If you want to move folder locations, adjust this is SetLocal.R:
  + One folder with code, and a subfolder with parameter documents. You can just git this folder. The path to this folder doesn’t need to be specified in SetLocal.R: the script looks at its own call to determine where it is running.
  + One folder to store the files. They can grow large, so best not use a shared folder (Git, OwnCloud, DropBox, etc) for this. There are a few subfolders:
    - InputForNew is to place to store older harvests. If you don’t have older harvests, this folder is automatically created, at the start of the harvest, with two stub files. And in the end the result is copied here, to be used for the next harvest (rather: you are prompted if you want to copy).
    - A folder with the name of your harvest (automatically created). There all files from the harvest are stored. The final result is in subfolder ‘dc’
    - WerkSet is the place to store the dataset for which you want to create plots. After the harvest is complete, the script asks if you want to copy here. If there are multiple files here, the newest one (with the right filename) is taken.

# Instructions: a new harvest

* Make sure all needed files are stored locally, adjust any paths in SetLocal.R.
* Open OAI Harvest New Records.R.
  + If you want to make the code more readable: in Rstudio, just to the right of the line numbers, there are small triangles. Click on them to collapse or expand chunks of code. I copied my comments just after these blocks, so you can collapse them and see what a chunk does.The code contains some parameters inline, switches that decide what should be run. The ones here under should be checked/adapted, they are all onder the ‘General settings’ header
  + Params$harv should read ‘dc’, to just harvest the dc dataset
  + Params$NameOfHarv tells R where to store the results, best to use the date (as a string, but don’t use special characters)
  + Step should read ‘JustGo’.
    - This means the script first checks how much work has been done before (so that you can interrupt the script), and starts anew if nothing has been found.
  + Params$debug should read F to just run the script.
  + Params$WayBack is only used if Params$debug is T. In that case, files from after the specified date/time are ignored. Useful for debugging, to see how the script is running from another point in time.
  + Params$Summarize tells whether to output a summary of results. I usually leave it at just c(‘IDs’). This means a summary of IDs is printed if new IDs have been harvested (i.e. if you interrupt and continue at another point in time, no new summary is shown), and a summary of harvested records is shown also if new records have been harvested (idem)
  + The fields Params$didlbeperkt and Params$mongoColl are not used when doing a dublin core harvest (as we are doing now). For the trimmed down version of the script, these are moved.
  + To just run the script, they should read:
    - Params$harv <- 'dc'
    - Params$NameOfHarv <- '*yyyy-mm-dd*'
    - Step <- 'JustGo'
    - Params$debug <- F
    - Params$WayBack <- as.POSIXct('2099-12-31 23:59:59', tz='UTC')
    - Params$Summarize <- c('IDs')
    - Params$oldfdate <- as.POSIXct(‘2018-01-01’, tz=’UTC’)
* Make sure the older files are located in the InputForNew folder, under a name with ‘Total’+‘dc’ and ‘IDList’+’dc’. Or make sure there are no matching files to do a completely new reharvest
* One problem is that the script needs a lot of ‘stack memory’. This is ordinary RAM, but is usually limited by the system (standard on OSX is 8 MB per process). RStudio can’t adjust this limit, if you run the script from RStudio it simply exits with an error at some point. If you need to debug you can use RStudio, but if it throws an error like ‘close to stack limit’ or ‘stack limit exceeded’, this is the cause.   
  So for just running it entirely we need to run it separately:
  + Open a terminal
  + Type ulimit -s 32768. This sets the stack memory limit to 32768 KB (Not sure how much is needed, but 8192 is too little, this works, 65532 is maximum)
  + “R” opens R
  + Run the script (source(‘*Path*/*File*.R’))
  + Ctrl+c to break
  + When finished type q(), you don’t need to save your workspace.
  + I encountered it a few times that it breaks without throwing an error, especially when it’s harvesting. It’s not reproducible, but then you just end up with a bash prompt, without seeing anything. Just restart R (“R”), and run the script (source(‘*Path*/*File*.R’) again). It will pick up approximately where it left.
  + During the harvesting, if you run with default options, it also shows 2 graphs. You can ignore these if you want, but they show the differences between the old harvest and the new one. The labels are not made pretty, not a problem.
    - The first graph shows the “new” records. They are graphed according to their timestamp, and color-coded to show if they were there before:
      * Never: Probably not any, although could be some on the left-hand side. Records that have been marked ‘deleted’ here, but have also been marked ‘deleted’ in the previous harvest.
      * Deleted: They were there before, but now the system tells us that they are deleted.
      * Updated: We’ve seen them before, but something has changed
      * Unchanged: They were there before. This will only be on the left of the plot, and most of them will be left of the margin (so not visible)
      * New: they weren’t there before
    - The second plot show it looking from the old records, what happened to them. The color coding is the same, although:
      * New is of course not in this plot: there is no old records to compare with
      * Disappeared may be present. This means a record was there before, but now it has vanished, without the system telling us. This is not compliant with specifications, but it happens.
    - After the plots are displayed, there is also some textual output, telling how many have which status, split between datasets (dataset=TRUE) and normal publications (dataset=FALSE). Only for new records this split can’t be made (we need to harvest them first), so they are put together (dataset NA means Unknown)
* Running the script On a MacBook Air with DANS wifi takes approx. 7 hours if half of all records have been renewed.
  + The more records are renewed, the longer it takes of course, but it’s less than linear (I guess a full harvest takes 10 hours). The script tries to tell you what it is doing.
    - Except when harvesting it is reliant on another process which doesn’t tell its progress. You can watch in Finder how many files are written there, or break and restart: Press Ctrl+C, then arrow-up and Enter (to start the script again). It then also tells you have much has been done before you interrupted.
  + The first part of this is almost entirely downloading files, which means you need an active network connection, but can otherwise use your laptop perfectly.
  + The second half is more a calculation part, meaning you don’t need a network connection, but a lag is noticeable (CPU usage is high). You could run it at night. When I did this, I installed a ‘noSleep’ app, which disables your laptop going to sleep mode if you close the lid. The screen does turn off, but the script keeps on running.
  + The results are stored on disk, in a folder with the name of the harvest (Params$NameOfHarv which you specified at the startis used), subfolder ‘dc’,
    - “NewTotal” is the result we’ll be working with most. At the end of the script, it is copied to other folders to be used as basis for the plots and as basis for an new harvest if you answer ‘y’ when prompted.
    - IDlist is a summary, with IDs of records that have been harvested. This one is also copied.
    - Total\_N\_Del is a list of all records that have been marked as ‘deleted’. We don’t use them here.
    - ErrorsUpToStepMergeFinal contains a list of all irregularities that have been found, see the next step.
    - Other files are partial results, they are useful for debugging, but can be removed.
  + If you want to use this dataset as basis for plots, answer ‘y’ when prompted if you want to copy files. (if you say no and change your mind, run the script again, it will look what has been done and ask if you want to copy.)
* Open the file ErrorsUpToStepMergeFinal (*timestamp*).rds (in the folder of your harvest, subfolder ‘dc') in RStudio (Drag it to Rstudio or rightclick, Open With,.. ) and specify a name (here I’m calling it “Errors”).   
  This file contains a list of all irregularities found by the script. A lot can be ignored, but some checking is necessary.   
  If you encounter something unexpected, please check the document “Harvest Errorfile”.
  + The file is split in 12 parts. We first need to see how much is in each of the parts: in the RStudio-console, enter   
    *sapply(Errors, function(x) {if(class(x)=='data.frame') nrow(x) else length(x)})* and press <Enter>. You get a list with a count of every part.
    - “Count” and “UpToStep” should be 1
    - “longdelays” “RecParse”, “SummaryOfRecParse”, “skippedFiles” and “ColumnsSkipped” should all be 0. If one of them is not, we need to investigate first.
    - If MultiIDs is 0: fine. If not, execute (in the console) “min(Errors$MultiIDs$LastUpdate)”. The resulting date should be after the start of this workflow. If it is not, investigate.
    - Disappeared means records that were there before, but have now vanished without leaving a trace. That is not compliant with OAI-PMH, but it happens. If this number is too large (>500), talk to Chris. You can see which ones have disappeared with  
      *Errors$Disappeared$ID* (if there are too many you can append this command with “[*From*:*To*]” to get a subset)
    - RecSumm and RecMerge can be any number, the script provides solutions to these problems. They typically are quite large (1K up to 100K (or more). The reason they are still here is just for archiving purposes.
    - If RecMergeFinal is 0: fine. If not, take a look at it by printing it, just type Errors$RecMergeFinal in the console, and it shows a list of strings, specifying what went wrong. It’s mostly about waardeomzet: the values that are “translated”, based on your waardeomzet.csv parameter document. In some cases, it can be ignored (e.g. if there is a language specified by a repository which is not in ISO 639), but this means the “translation” will be *NA* which means “Not available”.
* If you are happy about the final result, you can delete temporary files (i.e. all with (temp) in their filename). Furthermore, it’s sometimes handy to keep the reference files in the folders ‘Chunkdc’, ‘dcXML’ and ‘dcIDs’, but not necessary. You can delete them if you’d like more hd-space.

# Instructions: generating plots

* Make sure the right dataset is stored in the ‘WerkSet’ folder. To see if it is the right one, there is a timestamp embedded in the filename, in the form *yyyymmddHHmmss.*
* Open ‘Plotjes.R’.
  + If you want to make the code more readable: in Rstudio, just to the right of the line numbers, there are small triangles. Click on them to collapse or expand chunks of code. I copied my comments just after these blocks, so you can collapse them and see what a chunk does. If you collapse line 1, and the line just *after* the start of the for-loop, you can see all you need.
  + There are some parameters in the code, switches to decide which parts should be run. multiPlot and multiPlotParams are a list of what to run, and in the for-loop parameters parameters are reset every time a new plot is produced. I’ll go along the parameters you might want to set here, or a summary at the last point
    - multiPlot tells which plots should be produced. The script is able to produce all kinds of plots (although some are just stubs, or older broken versions), which all have their own number. The ones used for the website are 7, 2, 22 and 14, repeated four times.
    - multiPlotParams tells how the graphs should be produced: in English and dutch, and to output htmlfiles marked ‘global’ and ‘local’. The ‘global’ ones are htmlfiles that can be put online, but if you open such a htmlfile when it its stored locally, it looks ugly and broken. So the ‘local version’ adjusts links to stylesheets etc. so that you can use it as a preview.
    - For plotting all four graphs in dutch and English, in local and global version, this should read:  
      part <- T  
      multiPlot <- lapply(rep(c(7, 2, 22, 14), times=4), function(x) {x})[part]  
      multiPlotParams <- list(GglPlotGlobLinks=rep(c(T,T,F,F),each=4)[part],  
       LabelsLang=rep(c('nl','en','en','nl'), each=4)[part])
    - PP$PrintPlot specifies (as T(RUE) or F(ALSE)) whether the plot should be shown as it is produced. It can be helpful to see errors as the code is running, but doesn’t really add anything in a production environment. And it can, however, lead to errors. If you get the ‘polygon edge not found’ error, you can try to run again, but if it keeps on popping up, disable this option.
    - PP$PausePlot tells the script to wait after each plot is produced (it waits until the user presses a key). Set to FALSE for production, but useful for debugging.
    - PP$Saveplot tells whether to store the graphs as separate images as well.
    - PP$Zoomfactor tells the script how big textlabels etc. should be. This option is only used for the imagefiles and the plot shown during drawing.
    - PP$SaveGglPlotStd specifies if the htmlfiles should be generated. Leave to T(RUE) for this workflow
    - TL; DR: To just generate websiteplots as standard, you need this code:
      * Outside the for-loop:

*part <- T  
multiPlot <- lapply(rep(c(7, 2, 22, 14), times=4), function(x) {x})[part]  
multiPlotParams <- list(GglPlotGlobLinks=rep(c(T,T,F,F),each=4)[part],*

*LabelsLang=rep(c('nl','en','en','nl'), each=4)[part])*

* + - * Inside the for-loop:  
        *PP$PrintPlot <- FALSE  
        PP$PausePlot <- FALSE*

*PP$SavePlot <- TRUE*

*PP$Zoomfactor <- .5*

*PP$SaveGglPlotStd <- TRUE*

* Run it. Some warnings are produced that can be ignored:
  + “Warning: Ignoring unknown aesthetics: width”
  + “Removed *n* rows containing missing values (…)”
* If you run it in RSTudio, and want to preview the plots, make sure to have the ‘plots’ tab selected in the bottom right pane
* The script sometimes crashes with an error “polygon edge not found”. This is a bug in a package in R (ggplot2). If it happens, re-run the script, and/or disable the preview printing of plots (Set parameter PP$PrintPlot <- FALSE).
* If something else goes wrong, which you can debug, it’s important to remove temporary result-html-files. The script works by looking if there is a html-file present, and then adjusting this file, for each graph, by either replacing some code or inserting some extra code. If however, something goes wrong, this can mean traces of code are still present, which can cause all kinds of unexpected behaviour, sometimes not immediately visible.
  + You can leave the ‘GoogleChartsCode Orig…’ files, these are not modified.
* It takes approx. 5 minutes, after which in the output folder you’ll find the plot-images in the plots-folder (if you set this parameter), and 4 html-files. The ones for local viewing are to check if the results are all right, the ‘website’ can go to Wilko to upload to the website.
* Wilko will compile the result to upload and go live