ScheldeMonitor Manual:
Using the RStudio environment

Written, maintained and updated by VLIZ - Version 1 (15/12/2020)

1 ABOUT

This document was written as a manual on the use of the online RStudio environment of the ScheldeMonitor information and data portal. This environment has been accessible through the ScheldeMonitor website since 2021. It provides accredited researchers and the partners of ScheldeMonitor with a centralized RStudio hub to do analysis and build up scripts directly based on the data and information that is held within the portal and the underlying database.

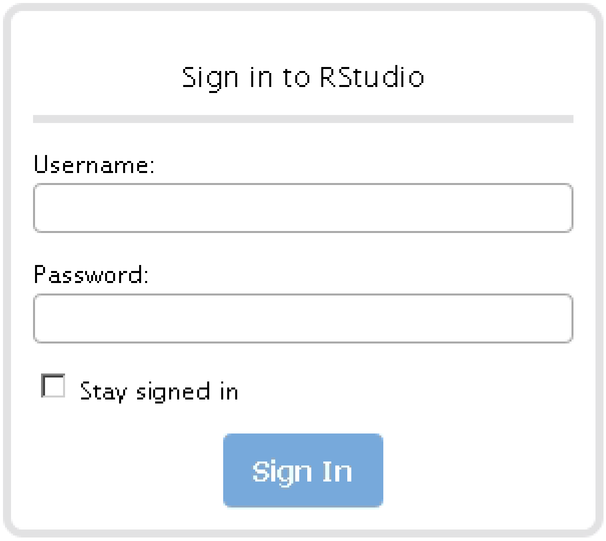
Within the manual, guidelines are provided for new users or users that are inexperienced in the use of RStudio, as well as some overall recommendations on how to keep work in RStudio structurized and comprehensible. The last chapter of this manual also provides a step-by-step breakdown on how to load data from ScheldeMonitor into the RStudio workspace.

The RStudio environment can be accessed from the website, using the following link. To access, credentials are required. These credentials can be requested here, or by sending a mail to info@scheldemonitor.org with a statement on the reason why use of the RStudio environment is required.

It is also possible to link the RStudio environment with an existing project from the ScheldeMonitor GitHUB organization. How to work with GitHUB in relation with RStudio, is discussed in an additional RStudio manual.

2 Connecting to the RStudio environment of ScheldeMonitor

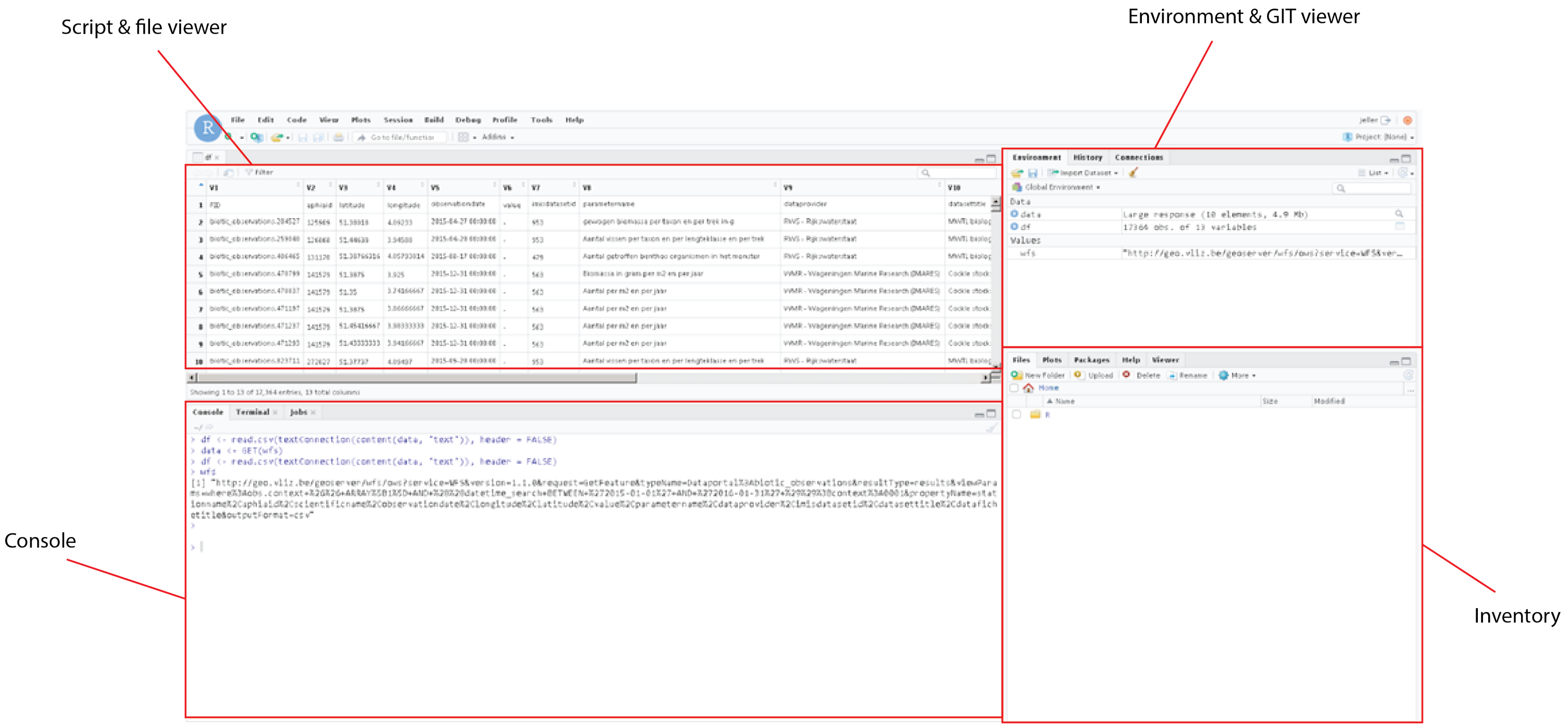
The ScheldeMonitor environment can be accessed at (https://rstudio.scheldemonitor.org/auth-sign-in) using your received credentials:



These credentials are similar to those used for all other tools within the ScheldeMonitor platform, such as the data download toolbox and the E-room. To receive your personal credentials, contact the helpdesk of ScheldeMonitor stating a reason to use the RStudio environment.

Once logged in, you will see your personal workspace in RStudio. This workspace can either be cleaned (for instance if you are a new user) or show the structure and content that was worked on during your previous session if you saved the workspace image on the last closure.

A personal workspace is standardly composed of four windows, showing scripts or dataframes, your environment, the console, and your project or personal file structure. It also indicates which user is logged in and which project is linked to your workspace:



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	FID	aphiaid	latitude	longitude	observationdate	value	imisdatasetid	parametername	dataprovider	datasettitle
2	biotic_observations.284527	125869	51.38818	4.09255	2015-04-27 00:00:00	.	953	gewogen biomassa per taxon en per trek in g	RWS - Rijkswaterstaat	MWTL biolog
3	biotic_observations.259848	126868	51.44638	3.94588	2015-04-28 00:00:00	.	953	Aantal vissen per taxon en per lengteklasse en per trek	RWS - Rijkswaterstaat	MWTL biolog
4	biotic_observations.486465	131128	51.38766356	4.05793814	2015-08-17 00:00:00	.	479	Aantal getroffen benthos organismen in het monster	RWS - Rijkswaterstaat	MWTL biolog
5	biotic_observations.470789	141529	51.3875	3.925	2015-12-31 00:00:00	.	563	Biomassa in gram per m2 en per jaar	WMR - Wageningen Marine Research (IMARES)	Cockle stock
6	biotic_observations.470837	141575	51.35	3.74166667	2015-12-31 00:00:00	.	563	Aantal per m2 en per jaar	WMR - Wageningen Marine Research (IMARES)	Cockle stock
7	biotic_observations.471197	141529	51.3875	3.86666667	2015-12-31 00:00:00	.	563	Aantal per m2 en per jaar	WMR - Wageningen Marine Research (IMARES)	Cockle stock
8	biotic_observations.471237	141575	51.45416667	3.98333333	2015-12-31 00:00:00	.	563	Aantal per m2 en per jaar	WMR - Wageningen Marine Research (IMARES)	Cockle stock
9	biotic_observations.471293	141529	51.43333333	3.94166667	2015-12-31 00:00:00	.	563	Aantal per m2 en per jaar	WMR - Wageningen Marine Research (IMARES)	Cockle stock
10	biotic_observations.825711	272627	51.37737	4.09497	2015-09-28 00:00:00	.	953	Aantal vissen per taxon en per lengteklasse en per trek	RWS - Rijkswaterstaat	MWTL biolog

Showing 1 to 13 of 17,364 entries, 13 total columns

```
Data
  data    Large response (10 elements, 4.9 Mb)
  df      17364 obs. of 13 variables
Values
  wfs     "http://geo.vliz.be/geoserver/wfs/ows?service=WFS&ver…
```

```
> df <- read.csv(textConnection(content(data, "text")), header = FALSE)
> data <- GET(wfs)
> df <- read.csv(textConnection(content(data, "text")), header = FALSE)
> wfs
[1] "http://geo.vliz.be/geoserver/wfs/ows?service=WFS&version=1.1.0&request=GetFeature&typeName=Dataportal%3Abiotic_observations&resultType=results&viewPara
ms=where%3Aobs.context+%26%26+ARRAY%5B1%5D+AND+%28%28datetime_search+BETWEEN+%272015-01-01%27+AND+%272016-01-31%27+%29%29%3Bcontext%3A0001&propertyName=stat
ionname%2Caphiaid%2Cscientificname%2Cobservationdate%2Clongitude%2Clatitude%2Cvalue%2Cparametername%2Cdataprovider%2Cimisdatasetid%2Cdatasettitle%2Cdatafich
etitle&outputFormat=csv"
>

> |
```

3 Working with the RStudio environment of ScheldeMonitor

3.1 Guidelines for workspace

The default behaviour of R for the handling of .RData files and workspaces encourages and facilitates a model of breaking work contexts into distinct working directories. This implies that the user can select a certain folder in his local directory to use as the location where files, handled through RStudio, are saved. This local directory, or workspace, can be altered at any given moment by the user.

In version v0.95 of RStudio, a new 'Projects' feature was introduced to make managing multiple working directories more straightforward. It is recommended to use this feature, however this chapter also explains how to handle your workspaces in the default manner.

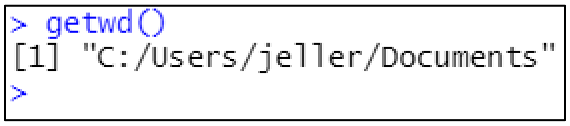
As with a local RStudio installation, the online RStudio environment of ScheldeMonitor uses the local user's home directory as workspace by default. This workspace is typically referenced using ~ in R. When RStudio starts up it does the following:

- Executes the .Rprofile (if any) from the default working directory.
- Loads the .RData file (if any) from the default working directory into the workspace.
- Performs the other actions described in R Startup.

When RStudio exits and changes to the workspace have been made, a dialog box asks whether these changes should be saved to the .RData file in the current working directory. Clicking "Save" will ensure that your changes are stored and will appear as they were the next time you login to the RStudio environment.

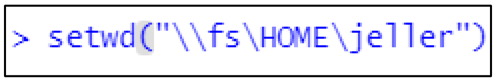
Set a workspace

RStudio displays the current working directory within the title region of the Console. To check your current working directory, you can run the command getwd() in the RStudio console:



```
> getwd()
[1] "C:/Users/jeller/Documents"
>
```

To change the working directory, you can run the command *setwd()* in the RStudio console with the new directory inserted as a string:



```
> setwd("\\fs\HOME\jeller")
```

You can also change the working directory by selecting the “Tools” menu and “Change Working Directory”. For users with Mac, this is found in the “Session” menu and the “Set Working Directory”.

Be careful to consider the side effects of changing your working directory:

- Relative file references in your code (for data sets, source files, etc.) will become invalid when you change working directories.
- The location where .RData is saved at exit will be changed to the new directory.

Because these side effects can cause confusion and errors, it is usually best to start within the working directory associated with your project and remain there for the duration of your session.

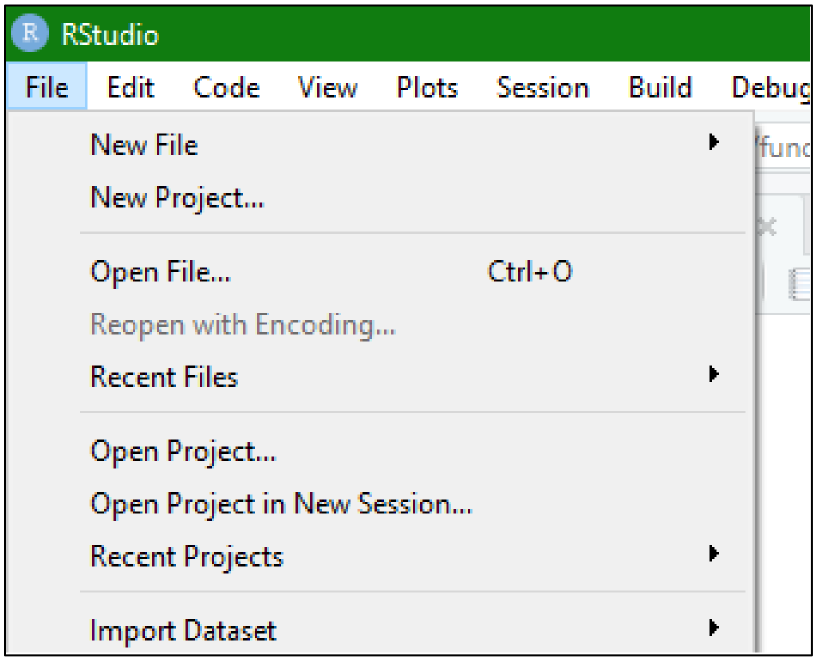
The best practice, however, is to connect the RStudio environment to a certain ‘project’. This allows for a better oversight on the working directories and different cases you work on within the same RStudio environment. The next segments describe how such projects are instigated.

Start a local project

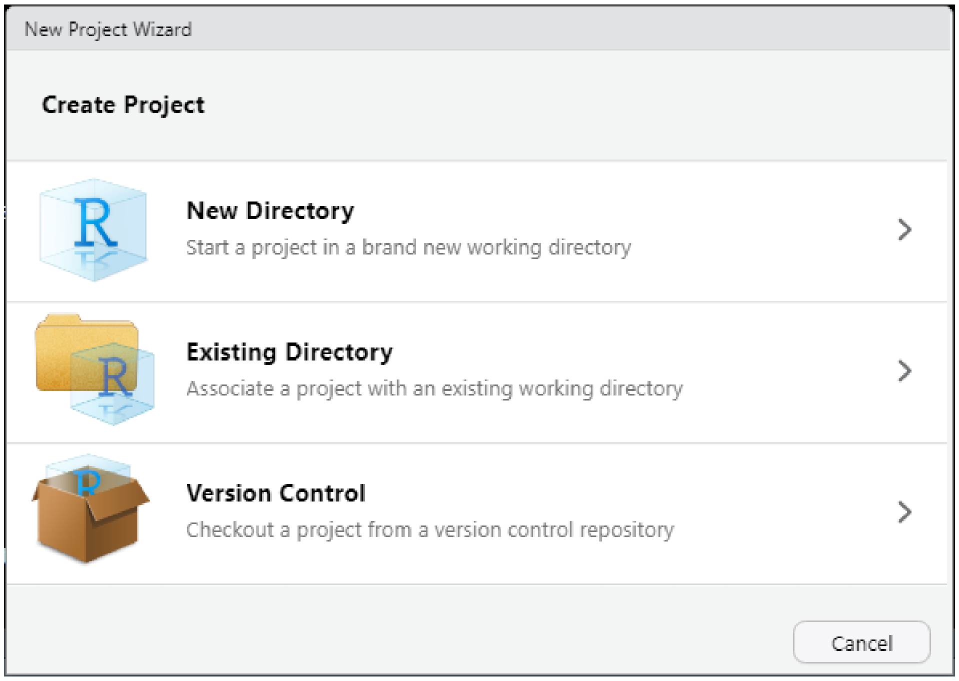
Any approved user can utilize the RStudio environment to commence or continue his or her personal project with the data of ScheldeMonitor. Doing so, users can either start a new local project, download an existing project from their own GitHUB, or connect their work to the GitHUB organization of ScheldeMonitor.

Starting a local project is the easiest way to commence your work. This project is saved on a local drive of the user’s hardware, and can only be restarted by accessing that drive. To initiate such a local project, users need to follow the following steps:

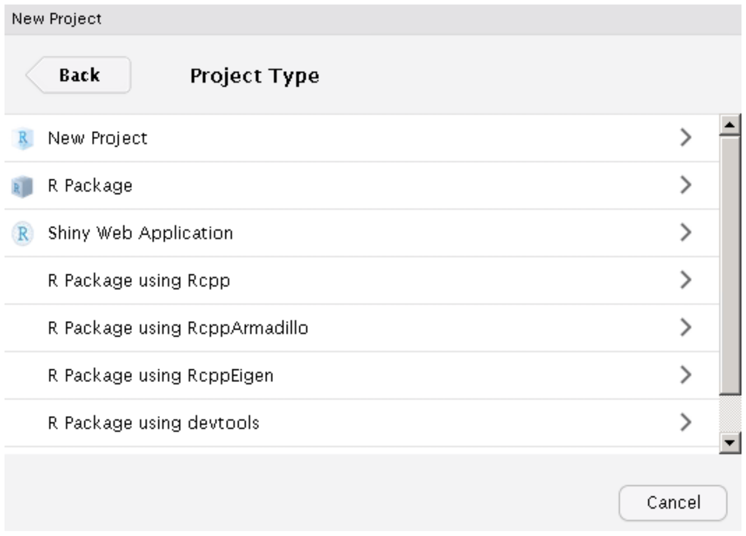
✓ Open the “File” menu and select “New Project”.



✓ In the New Project Wizard, select "New Directory". This will start a new project that will be saved on the local drive of the user's hardware.

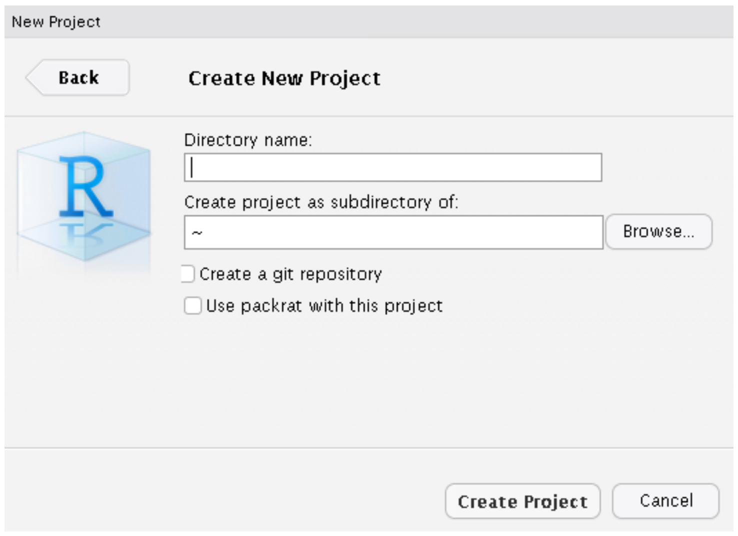

New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

✓ The user can now choose which type of project needs to be started.



- New Project: Basic R project where all kind of scripts can be set up.
- R Package: Project where users can make and publish dedicated R packages for other users.
- Shiny Web Application: R project where all scripts are premade for users to create and run Shiny web applications.

✓ Lastly, the user can name the project and select the directory in which the project will create a subdirectory. Selecting the option "create a git repository" will allow the user to use locally installed version control. This option is not needed when working with an online GitHUB space.

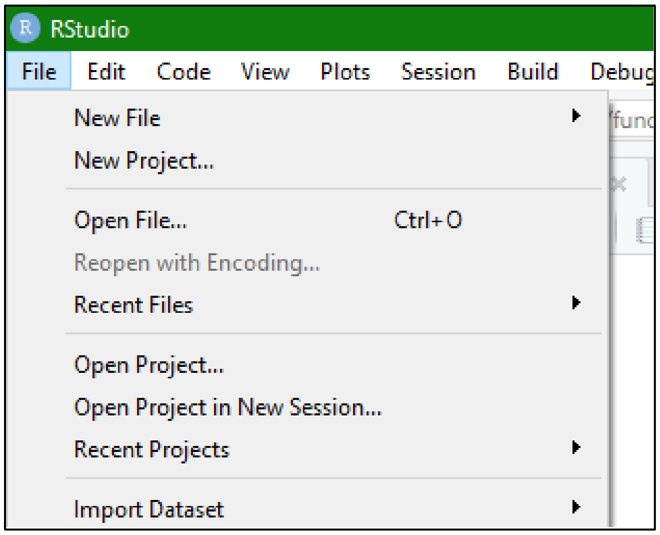


Set up an existing GitHUB project

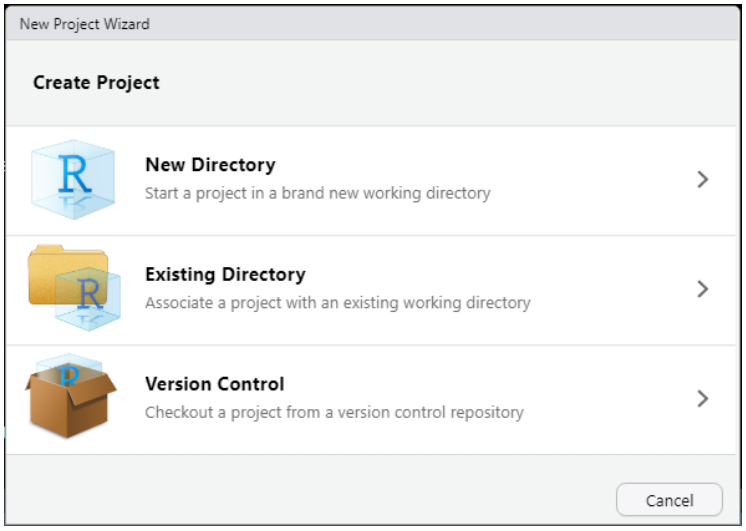
If a project needs to be user-controlled and accessible by multiple users, it is best practice to create a repository on GitHUB. Setting up a personal GitHUB repository is beyond the scope of this manual. However, as ScheldeMonitor has created a GitHUB organization, a dedicated manual on the use of GitHUB is available on the website.

To connect your personal space of the RStudio environment on ScheldeMonitor to a known GitHUB repository, the following steps need to be taken:

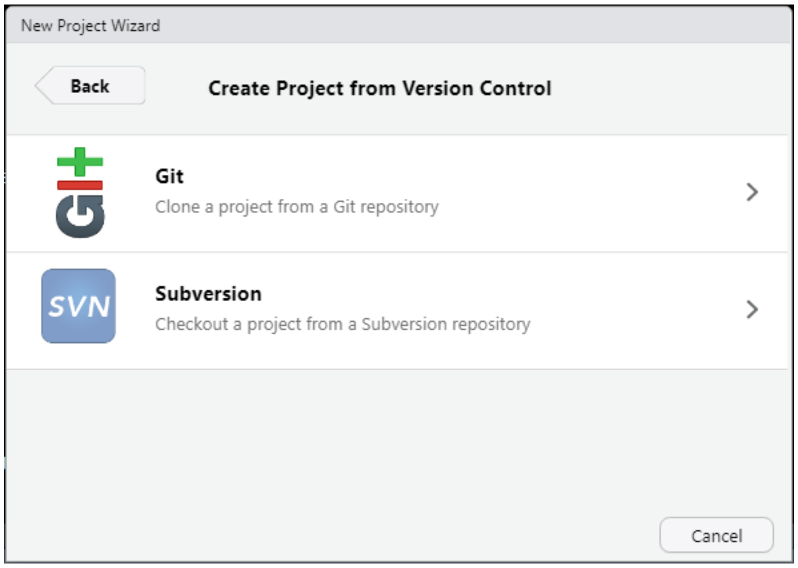
- ✓ Open the "File" menu and select "New Project".



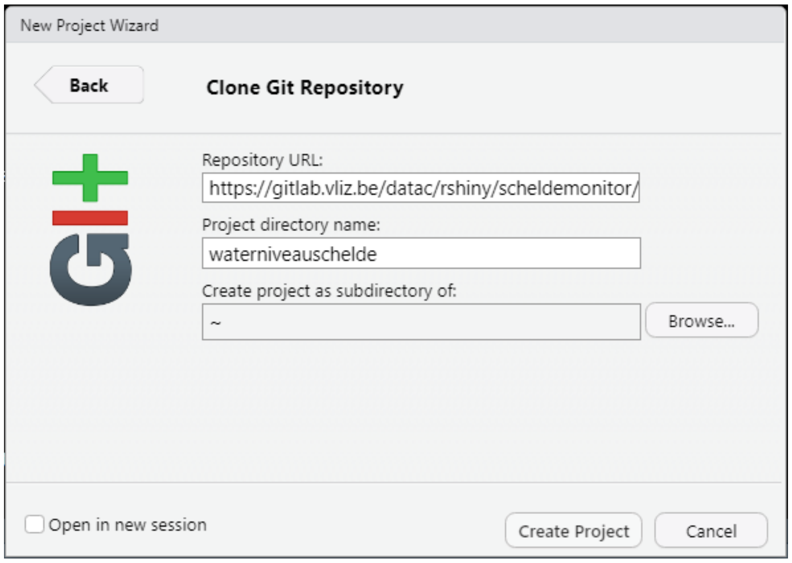
- ✓ In the New Project Wizard, select the option "Version Control".


New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

✓ Next, select "Git" as this is the version controlled methodology used within the context of ScheldeMonitor.


New Project Wizard
Back
Create Project from Version Control
Git
Clone a project from a Git repository
Subversion
Checkout a project from a Subversion repository
Cancel

✓ To establish a connection between your personal RStudio environment and the GitHUB repository, the system needs to know from which online space the repository can be downloaded. This URL matches the online repository on https://github.com/. Additionally, the system needs a local directory to store the downloaded files, or to hold up the files that have not been uploaded back to the repository yet.



Repository URL: https://gitlab.vliz.be/datac/rshiny/scheldemonitor/

Project directory name: waterniveauschelde

Create project as subdirectory of: ~

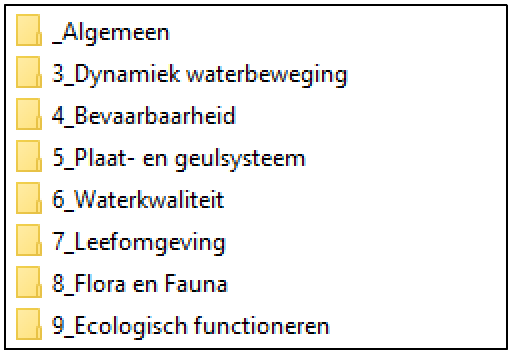
3.2 Guidelines for scripts

A working directory or project in RStudio can hold a large number of scripts and files to work with. In order to keep the work organized, as well as reproductive over time, it's important to structure these scripts both in the directory as well as internally. The segments below suggest guidelines that might aid researchers in keeping their work transparent for themselves and other users.

Directory structure

A working directory or project is similar to any other folder on the local drive of your hardware. This implies that such a directory can consist of folders and subfolders. It is, however, imperative that folders are created following a certain structure or idea, to make the scripts and underlying data findable for yourself and other users. There are multiple levels on which a directory can be structured.

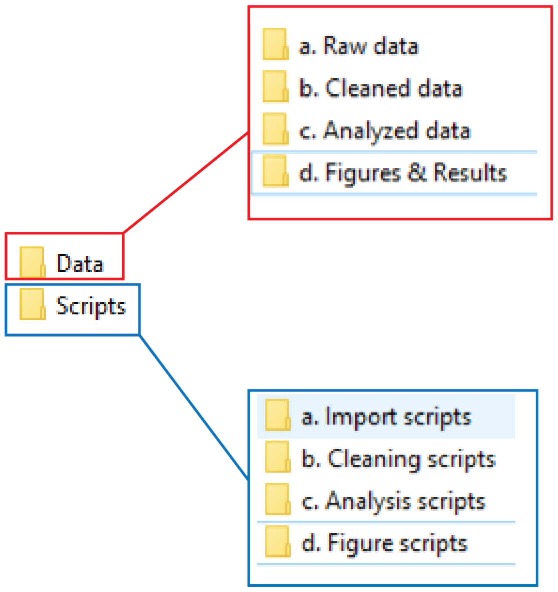
Firstly, if your work in RStudio is linked to a certain publication or report, your directory structure should mimic the same structure as the headings of the report. Here is an example from the T2015 report on the Scheldt, for which the project directory was structured conform the titles and subtitles within the published report:



_Algemeen
3_Dynamiek waterbeweging
4_Bevaarbaarheid
5_Plaat- en geulsysteem
6_Waterkwaliteit
7_Leefomgeving
8_Flora en Fauna
9_Ecologisch functioneren

Yet, it is even more important to have a uniform structure at the lowest level of the working directory, where all files are stored. Especially for projects that are not linked to a fixed report, and for which the above-mentioned structure is not applicable.

Typically, data files and scripts should be saved in separate folders. Although it might seem more convenient to keep those files together, the general overview benefits from the two-folder structure. Scripts and data files often do not have a 1:1 relationship, as a single script can use multiple data files while these data files are run through multiple different scripts. However, the structure of each folder should be the same, with a folder for every phase of the project:



Using this structure, a uniform workflow can be established within the project directory. This workflow follows four steps, that are explained using the following table:

	Using data from:	Using scripts or functions from:	Saving new data or results in:
Step 1 – Import data (if necessary)	n/a	'a. Import scripts'	' a. Raw data'
Step 2 – Clean data	'a. Raw data'	'b. Cleaning scripts'	'b. Cleaned data'
Step 3 – Analyze data	'b. Cleaned data'	'c. Analysis scripts'	'c. Analyzed data'
Step 4 – Create figures or results	'c. Analyzed data'	'd. Figure scripts'	'd. Figures & Results'

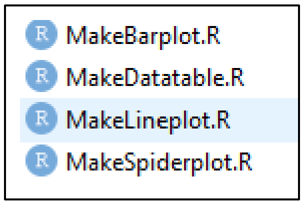
It is possible that users rather run a single script to go through all these steps, especially in smaller projects. In this case, a 'Main.R' script can be saved alongside the 'Data' and 'Scripts' folders. This main script can then run through all these steps on its own, while sourcing different data files and functions from the underlying folder structure. The latter is especially important in larger projects, to ensure that the length and readability of the main scripts is optimal. When doing so, it is very important that the main script is well structured and annotated, as will be further explained in Script structure and Script annotations below.

In any case, only one 'main.R' file should be present as to not create confusion.

Script naming

Scripts should be named in such a way that users can easily derive its purpose, in order to not have to open all scripts in an RStudio environment to know what they are used for. This is especially important when working with a main script that sources functions from other scripts throughout the different phases.

For example, when using different scripts for different kind of graphs, the nomenclature should clearly indicate which plot is made using the script:



MakeBarplot.R
MakeDatatable.R
MakeLineplot.R
MakeSpiderplot.R

Additionally, if the work in the RStudio environment is linked to a certain report or publication, the figure number from the publication could be inserted in the file name:

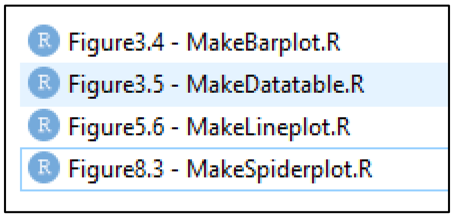


Figure3.4 - MakeBarplot.R
Figure3.5 - MakeDatatable.R
Figure5.6 - MakeLineplot.R
Figure8.3 - MakeSpiderplot.R

It is also possible that multiple scripts are used for the same figure, for instance if users want to be able to show both the original and the new plot on a later date. Still, the nomenclature needs to clearly indicate the discrepancies in the different scripts:

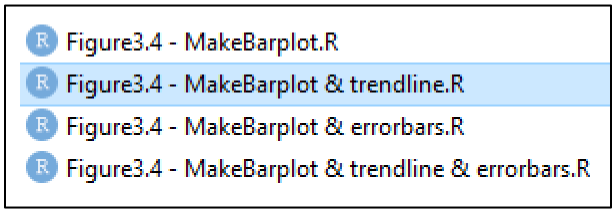


Figure3.4 - MakeBarplot.R
Figure3.4 - MakeBarplot & trendline.R
Figure3.4 - MakeBarplot & errorbars.R
Figure3.4 - MakeBarplot & trendline & errorbars.R

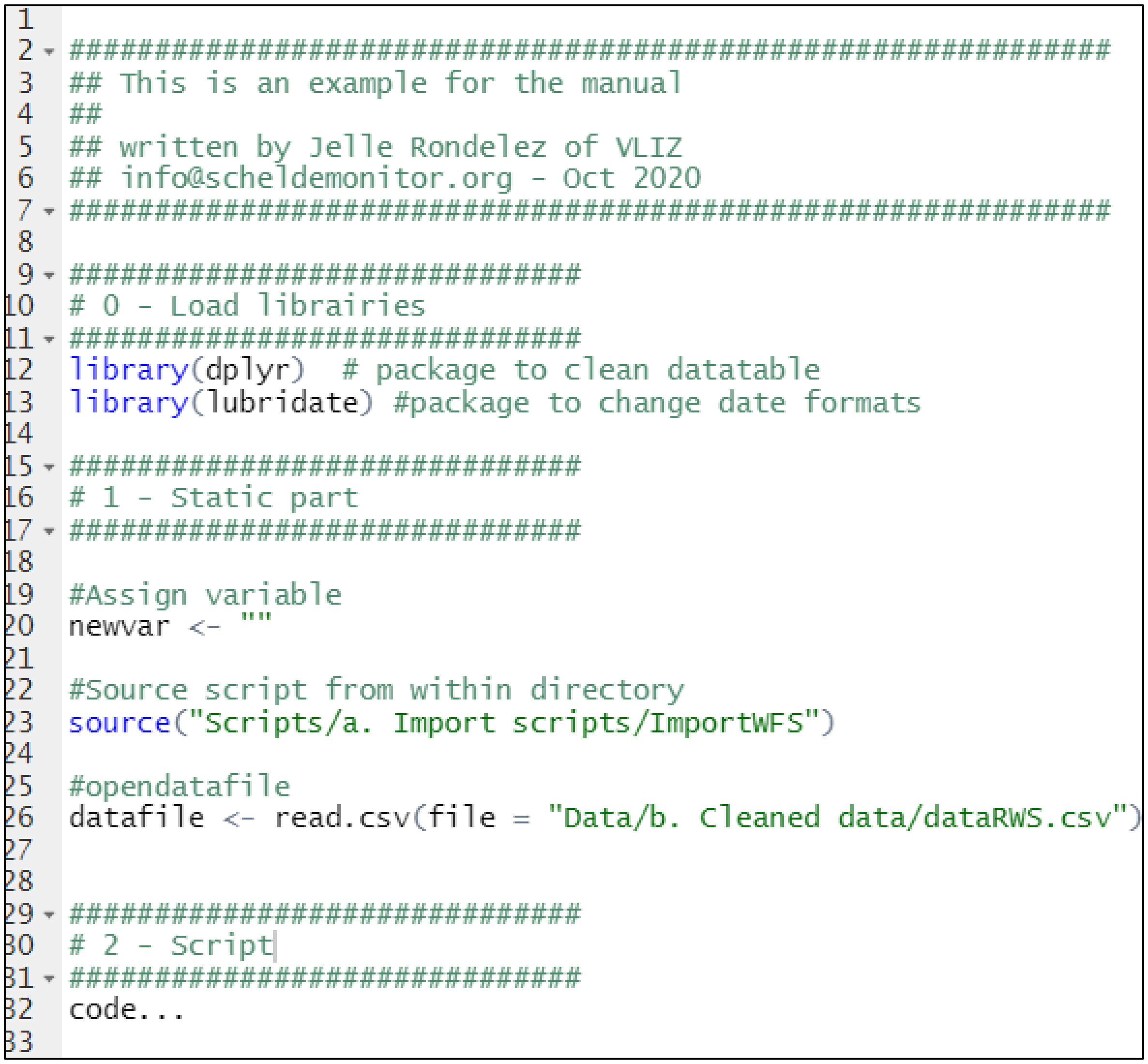
Nevertheless, whatever nomenclature is chosen, it should consist of a fixed and uniform naming convention. There are several options to choose from, similar to the ones available for code nomenclature as explained in Naming conventions:

- alllowercase: e.g. makebarplot
- period.separated: e.g. make.barplot
- underscore_separated: e.g. make_barplot
- lowerCamelCase: e.g. makeBarPlot
- UpperCamelCase: e.g. MakeBarPlot

Script structure

Similar to a directory, an individual script can greatly benefit from a fixed and uniform structure. This structure should clearly delineate the different sections in a script, which gives the reader a quick overview on the content, but also ensures the user that all actions and functions are run in a fixed order. Script structure can be accomplished almost immediately by using headings in the code. These are inserted in the same way as annotations are done. Ideally, all scripts should have the same headings to start with:

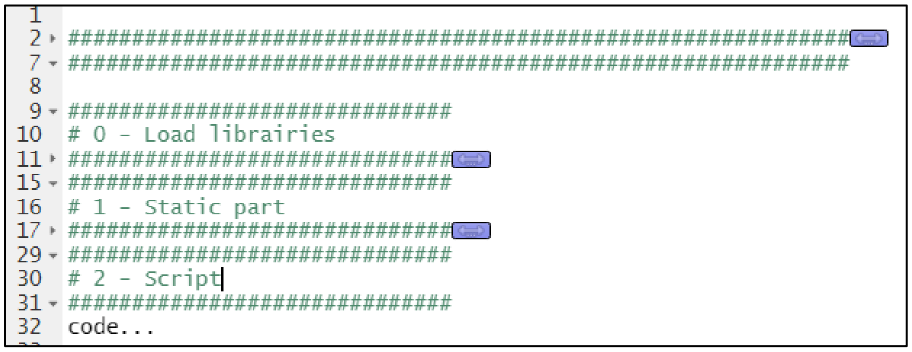
- Who, when, what and how: This is a large heading that should start every script in your project, stating who wrote the script, when it was written, how to contact the writer and what its purpose is.
- 0 – Load libraries: In this section all libraries are listed that need to be loaded before running the whole script. This section can also give some further explanation on the use of those libraries.
- 1 – Static part: In this part, all static actions are taken such as loading in data files, preparing those data files for analysis, sourcing other scripts and functions or naming arguments that will be used later on in the script.
- 2 – Script: This section contains the actual code that makes the script fulfill its purpose.



```r
#############################################################
## This is an example for the manual
##
## written by Jelle Rondelez of VLIZ
## info@scheldemonitor.org - Oct 2020
#############################################################

##############################
# 0 - Load librairies
##############################
library(dplyr)  # package to clean datatable
library(lubridate) #package to change date formats

##############################
# 1 - Static part
##############################

#Assign variable
newvar <- ""

#Source script from within directory
source("Scripts/a. Import scripts/ImportWFS")

#opendatafile
datafile <- read.csv(file = "Data/b. Cleaned data/dataRWS.csv")


##############################
# 2 - Script
##############################
code...

```

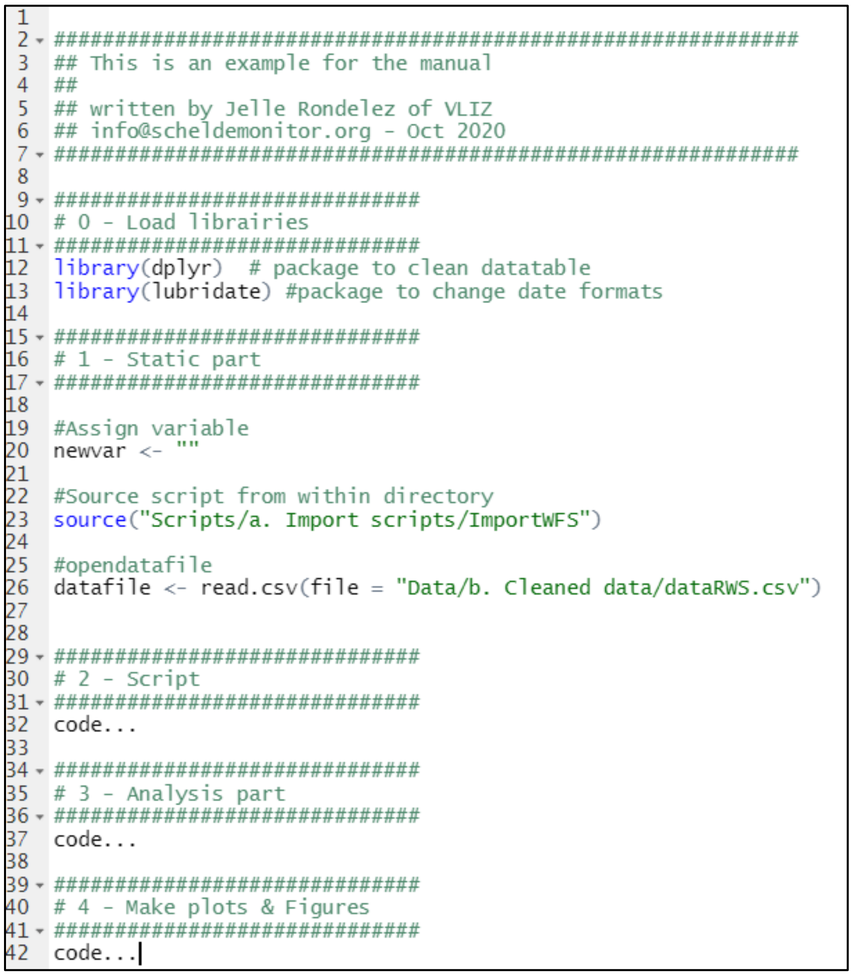
Note that the sourced files in the example above are using the directory structure as described in Directory structure.

These headings not only give a fixed structure and order to all scripts in the project, it also has the added advantage that sections can be collapsed or expanded if needed. Especially for longer scripts, in which certain sections of the code are not of interest to the user, this can greatly increase the readability of the script:



```
###############################################################
###############################################################

##############################
# 0 - Load libraries
##############################
##############################
# 1 - Static part
##############################
##############################
# 2 - Script
##############################
code...
```

Larger scripts can benefit more from an expanded structure with additional headings. This is especially true for 'main.R' scripts that run through all phases of the project within a single script, as discussed in Directory structure. Those type of scripts typically source and use a multitude of different functions and files. An extended structure can make these scripts more readable and can make it easier to search for a specific function or action:

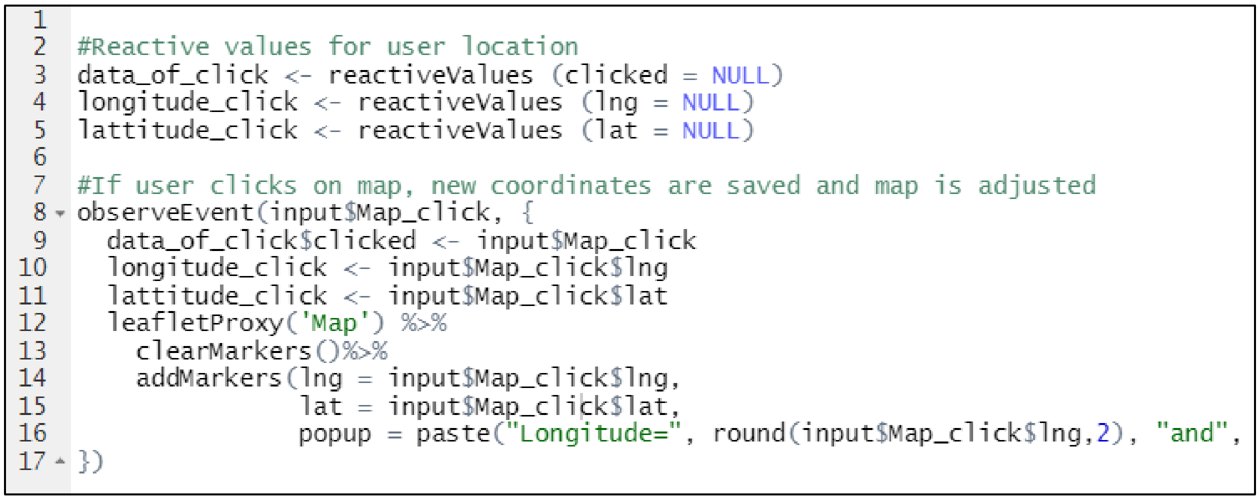


```

###############################################################
## This is an example for the manual
##
## written by Jelle Rondelez of VLIZ
## info@scheldemonitor.org - Oct 2020
###############################################################

##############################
# 0 - Load libraries
##############################
library(dplyr)  # package to clean datatable
library(lubridate) #package to change date formats

##############################
# 1 - Static part
##############################

#Assign variable
newvar <- ""

#Source script from within directory
source("Scripts/a. Import scripts/ImportWFS")

#opendatafile
datafile <- read.csv(file = "Data/b. Cleaned data/dataRWS.csv")


##############################
# 2 - Script
##############################
code...

##############################
# 3 - Analysis part
##############################
code...

##############################
# 4 - Make plots & Figures
##############################
code...
```

Script annotations

Annotating code is important for a number of reasons. The main reason is for the user personally when looking back on what was coded. It helps to explain in detail what a line, chunk or even section of code is trying to accomplish. This is also helpful for other people who read the code. Explaining what a line of code is doing can be useful for others who are looking to adapt work to their own, or when someone is checking or evaluating a chunk of code.

Annotating code is done with the symbol # (hashtag). Typically annotating can be done above a whole chunk of code, like when explaining the purpose of a certain function.



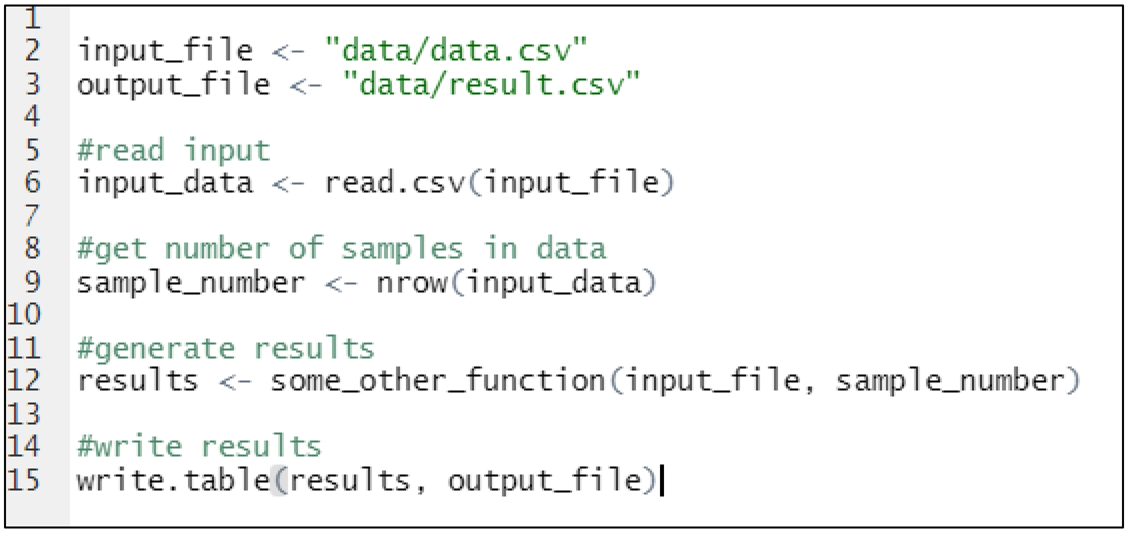
```r

#Reactive values for user location
data_of_click <- reactiveValues (clicked = NULL)
longitude_click <- reactiveValues (lng = NULL)
lattitude_click <- reactiveValues (lat = NULL)

#If user clicks on map, new coordinates are saved and map is adjusted
observeEvent(input$Map_click, {
  data_of_click$clicked <- input$Map_click
  longitude_click <- input$Map_click$lng
  lattitude_click <- input$Map_click$lat
  leafletProxy('Map') %>%
    clearMarkers()%>%
    addMarkers(lng = input$Map_click$lng,
               lat = input$Map_click$lat,
               popup = paste("Longitude=", round(input$Map_click$lng,2), "and",
})
```

3.3 GUIDELINES FOR CODE

Unfortunately, unlike other programming languages, R has no widely accepted coding best practices. Instead there have been various attempts to put together a few sets of rules. This chapter tries to fill the gap by summarizing what was found relevant in those various attempts.

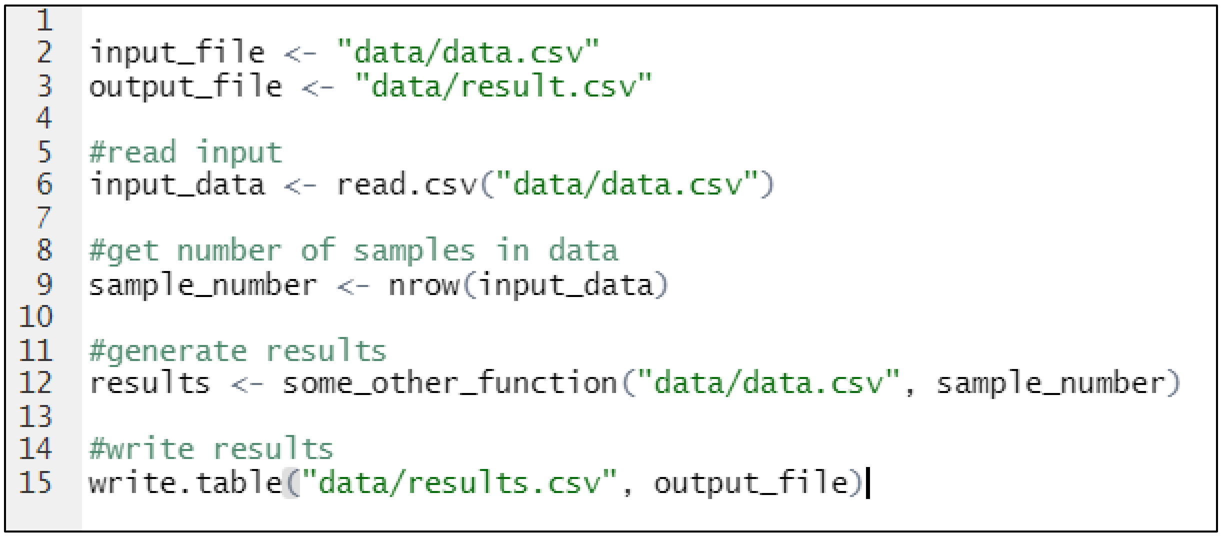
Hardcoding

Calling to a file or folder from within a script is mostly done through 'hardcoding', e.g. giving the location of the file as a string. However, users are strongly recommended to keep the amount of hardcoding minimal, as it requires less effort to change a script when a directory location changes if less harcoding is used. To do so, if your code will read in data from a file, define a variable early in the code that stores the path to that file. By doing so, the following example:



```r

input_file <- "data/data.csv"
output_file <- "data/result.csv"

#read input
input_data <- read.csv(input_file)

#get number of samples in data
sample_number <- nrow(input_data)

#generate results
results <- some_other_function(input_file, sample_number)

#write results
write.table(results, output_file)
```

is preferable to:



```r

input_file <- "data/data.csv"
output_file <- "data/result.csv"

#read input
input_data <- read.csv("data/data.csv")

#get number of samples in data
sample_number <- nrow(input_data)

#generate results
results <- some_other_function("data/data.csv", sample_number)

#write results
write.table("data/results.csv", output_file)
```

Naming conventions

R has no naming conventions for variables and functions that are generally agreed upon. As a newcomer to R it is useful to decide which naming convention to adopt. Generally, there are five naming conventions to choose from. It is important to pick one convention and stick to it for the remainder of your project:

- alllowercase: e.g. adjustcolor
- period.separated: e.g. plot.new
- underscore_separated: e.g. numeric_version
- lowerCamelCase: e.g. addTaskCallback
- UpperCamelCase: e.g. SignatureMethod

Above else, and besides the chosen naming convention, it is important to choose variable and function names that are concise and meaningful.

Spacing

As with naming conventions, there are no syntax conventions when it comes to writing code in R. However, large scripts benefit greatly from the use of a clear and consistent syntax, as it makes the code more open and readable. Using correct spacing in your code makes an invaluable difference in the syntax. It can be implemented by following these rules:

- Always put a space after a comma, never before, just like in regular English.

```
# Good
x[, 1]

# Bad
x[,1]
x[ ,1]
x[ , 1]
```

- Do not put spaces inside or outside parentheses for regular function calls.

```
# Good
mean(x, na.rm = TRUE)

# Bad
mean (x, na.rm = TRUE)
mean( x, na.rm = TRUE )
```

- Place a space before and after () when used with if, for and while.

```
# Good
if (debug) {
  show(x)
}

# Bad
if(debug){
  show(x)
}
```

- Place a space after () used for function arguments:

```
# Good
function(x) {}

# Bad
function (x) {}
function(x){}
```

- Most infix operators (==, +, -, <-, etc.) should always be surrounded by spaces:

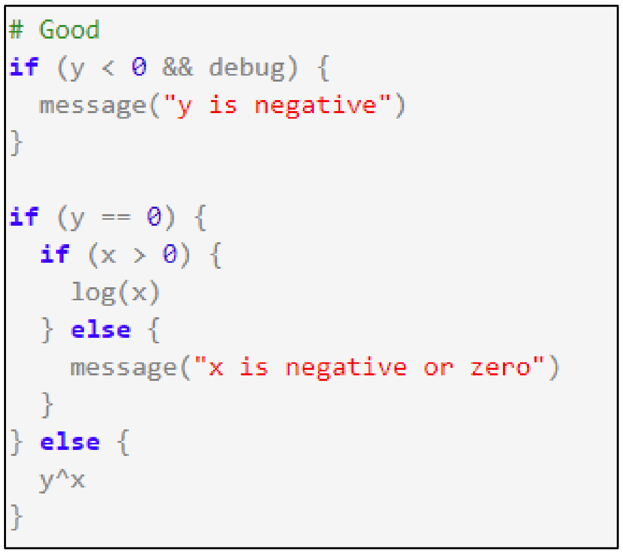
```
# Good
height <- (feet * 12) + inches
mean(x, na.rm = 10)

# Bad
height<-feet*12+inches
mean(x, na.rm=10)
```

However, it is important to not overdo spacing as well. Adding extra space can help, but only if it improves the alignment of = or <-. Do not add extra spaces to places where space is not helpful.

Code blocks

Just as when talking about the overall structure of a script, hierarchy is equally important within the code itself. To define the most important hierarchies, curly braces are used. However, to keep the hierarchy transparent for yourself and other users, a consistent syntax is needed when using curly braces. This syntax is based on three rules:

- '{' should be the last character on the line. Related code (e.g. an if clause, a function declaration, a trailing comma, ...) must be on the same line as the opening brace.
- The contents should be indented by two spaces.
- '}' should be the first character on the line.



```
# Good
if (y < 0 && debug) {
  message("y is negative")
}

if (y == 0) {
  if (x > 0) {
    log(x)
  } else {
    message("x is negative or zero")
  }
} else {
  y^x
}
```

Long lines of code

Users are recommended to always strive to limit the code to 80 characters per line. To do so, using a concise and efficient naming convention might already be an important step. If a function call is too long to fit on a single line, use one line each for the function name, each argument, and the closing bracket. This makes the code easier to read and to change later:

```
# Good
do_something_very_complicated(
  something = "that",
  requires = many,
  arguments = "some of which may be long"
)

# Bad
do_something_very_complicated("that", requires, many, arguments,
                              "some of which may be long"
                              )
```

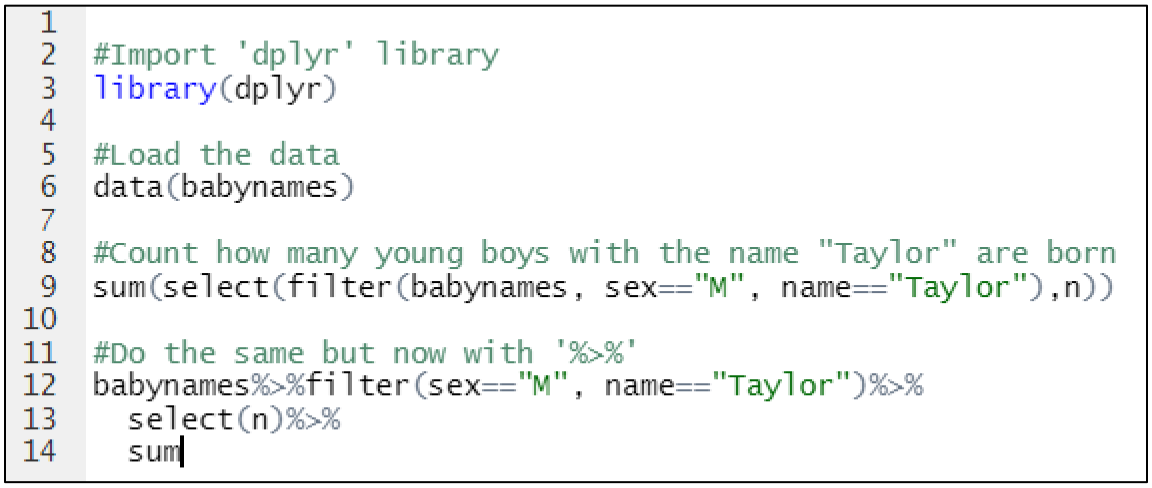
Pipes

Even when using correct spacing and adequate structuring of code blocks, a script can remain quite difficult to understand. This is especially true for scripts where a lot of different operations and functions are being used. When code is formed by a lot of functional language, it comes with a large number of parentheses and arguments per function. This can make code extremely complex and hard to understand.

To overcome this problem, users are recommended to using ‘piping’ for multiple actions on the same argument. Piping uses the ‘%>%’ operator and can be used by installing the ‘magrittr’ or ‘dplyr’ library. It is best explained through three simple rules:

- f(x) can be rewritten as x %>% f
- f(x, y) can be rewritten as x %>% f(y)
- h(g(f(x))) can be rewritten as x %>% f %>% g %>% h

When following these rules, it results in the following real-life R example:



```

#Import 'dplyr' library
library(dplyr)

#Load the data
data(babynames)

#Count how many young boys with the name "Taylor" are born
sum(select(filter(babynames, sex=="M", name=="Taylor"),n))

#Do the same but now with '%>%'
babynames%>%filter(sex=="M", name=="Taylor")%>%
  select(n)%>%
  sum
```

Tidyverse style guide & add-ons

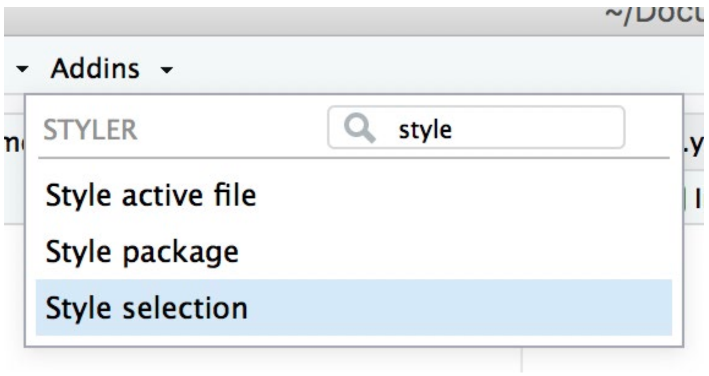
The R-community has multiple guides on how to style and manage your code in order to make it readable and clean. All these style guides are however fundamentally opinionated. Some decisions genuinely do make code easier to use, but many decisions are arbitrary. The most important thing about a style guide is that it provides consistency, making code easier to write because you need to make fewer decisions.

Users of the RStudio environment of ScheldeMonitor are recommended to use the tidyverse style guide, as it is one of the most commonly used guides. The rules mentioned above in this manual are also part of the tidyverse style guide.

There are two tools that can be installed by users that make it easier to implement this style guide, the 'styler' and 'lintr' packages. These packages can be installed with the following R code:

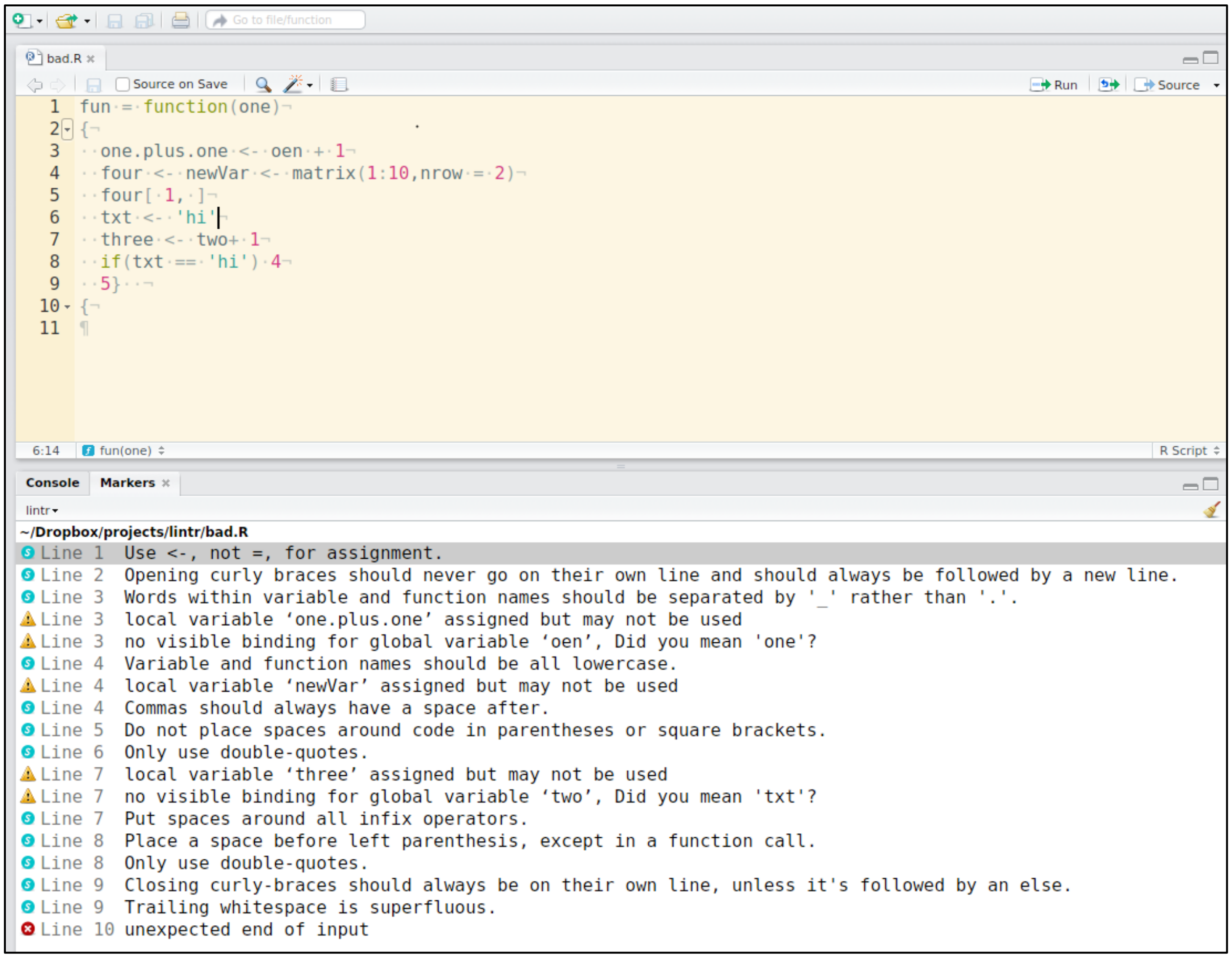
Install.packages()

- The 'styler' package allows to interactively restyle selected text, files or entire projects. It includes an RStudio add-in, the easiest way to restyle the existing code.



- The 'lintr' package can perform automated checks to confirm that code is conform the style guide. This check is automatically displayed in the RStudio 'Markers pane'. To show this pane, go the "Tools" Menu and select "Global Options…". A window with title "Options" will pop up. In that window: Select "Code" on the left; Select "Diagnostics" tab; Check "Show diagnostics for R".

The following window will now be visible:



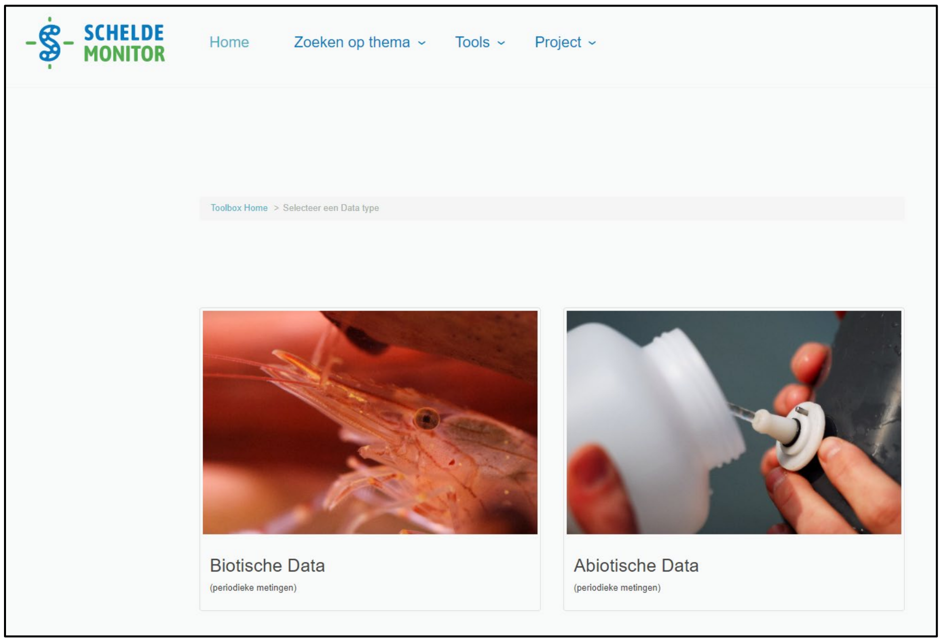
```
fun = function(one)
{
  one.plus.one <- oen + 1
  four <- newVar <- matrix(1:10,nrow = 2)
  four[ 1, ]
  txt <- 'hi'
  three <- two+ 1
  if(txt == 'hi') 4
  5}  
{

```

```
~/Dropbox/projects/lintr/bad.R
Line 1  Use <-, not =, for assignment.
Line 2  Opening curly braces should never go on their own line and should always be followed by a new line.
Line 3  Words within variable and function names should be separated by '_' rather than '.'.
Line 3  local variable ‘one.plus.one’ assigned but may not be used
Line 3  no visible binding for global variable ‘oen’, Did you mean 'one'?
Line 4  Variable and function names should be all lowercase.
Line 4  local variable ‘newVar’ assigned but may not be used
Line 4  Commas should always have a space after.
Line 5  Do not place spaces around code in parentheses or square brackets.
Line 6  Only use double-quotes.
Line 7  local variable ‘three’ assigned but may not be used
Line 7  no visible binding for global variable ‘two’, Did you mean 'txt'?
Line 7  Put spaces around all infix operators.
Line 8  Place a space before left parenthesis, except in a function call.
Line 8  Only use double-quotes.
Line 9  Closing curly-braces should always be on their own line, unless it's followed by an else.
Line 9  Trailing whitespace is superfluous.
Line 10 unexpected end of input
```

4 USING DATA FROM SCHELDEMONITOR IN RSTUDIO

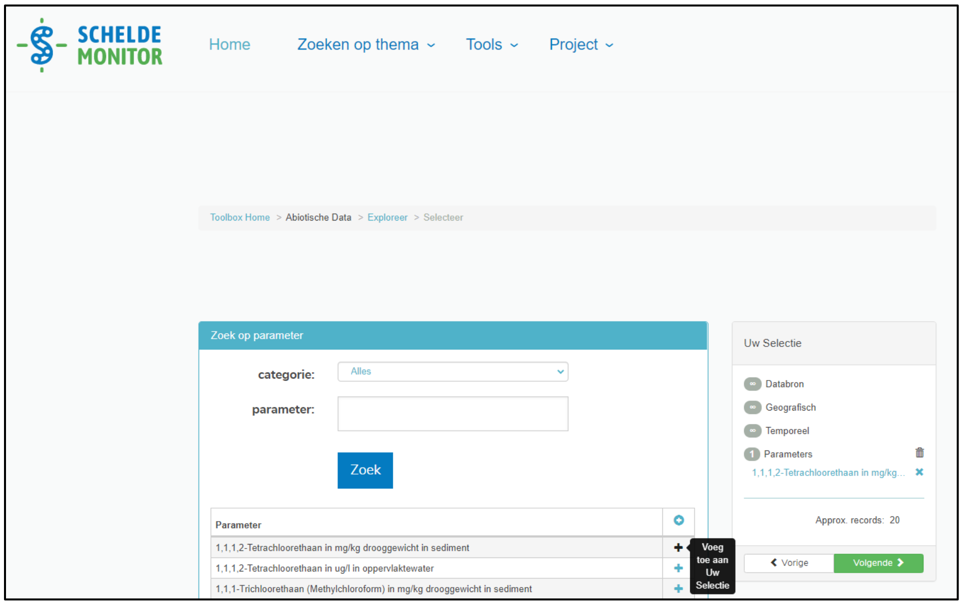
Most of the data in ScheldeMonitor can be used freely, and users are encouraged to use the RStudio environment of ScheldeMonitor to further analyse and validate our data collection. To do so, the data needs to be loaded into the RStudio environment first. This can be done either by loading downloaded data files such as CSV or TXT, or by using the generic webservices of ScheldeMonitor. Both methods involve accessing the Data Download Toolbox of ScheldeMonitor, which can be done using the following steps:

- ✓ Go to the home screen of the toolbox and choose between biotic and abiotic data.



- ✓ The toolbox offers several criteria to filter the database of ScheldeMonitor. These criteria differ for biotic and abiotic data, yet are not mandatory to be selected. When criteria are selected, the counter on the right side of the screen shows the remaining number of records that match the chosen criteria.

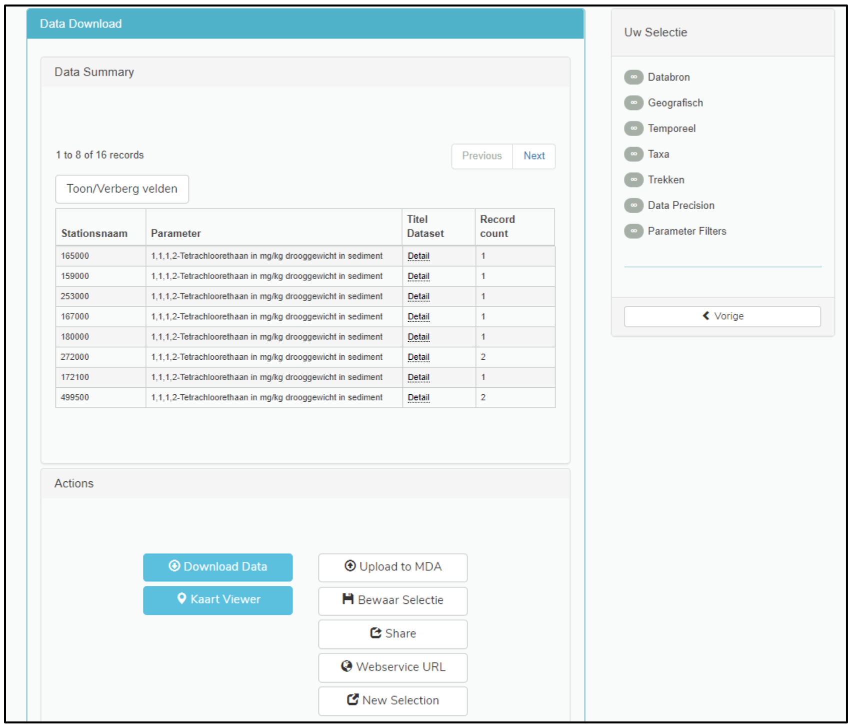
- ✓ Once all desired criteria are selected, select the green “Next” button to view a data summary of your data in the toolbox.



Parameter
1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment
1,1,1,2-Tetrachloorethaan in ug/l in oppervlaktewater
1,1,1-Trichloorethaan (Methylchloroform) in mg/kg drooggewicht in sediment

✓ The toolbox shows a summary of the chosen data set, along with several options to download or visualize the data. The following actions can be taken in the toolbox:

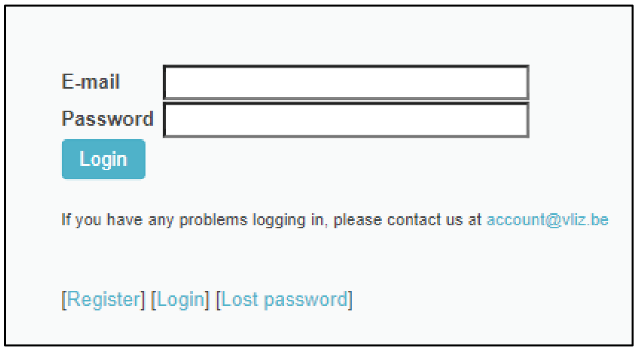
- *Download Data*: a data file in csv-format will be downloaded. More detailed information is available in the segment 'A: Using data from download data files'.
- *View on Map*: visualizes the data in a dynamic map viewer.
- *Upload to MDA*: saves your specific data selection to the Marine Data Archive, so that this selection becomes reusable on a later date.
- *Save selection*: saves a JSON file describing your specific data selection.
- *Share*: creates a URL link of your selection
- *Webservice URL*: generates a WFS url (Web Feature Service) that can be used to automatically load the data in a script or medium. More detailed information is available in the segment 'B: Using data from generic webservices'.
- *Load selection*: loads in a previously saved data selection using a JSON file.



1 to 8 of 16 records

Stationsnaam	Parameter	Titel Dataset	Record count
165000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
159000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
253000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
167000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
180000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
272000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	2
172100	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
499500	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	2

- ✓ Most, but not all data in ScheldeMonitor is public. Some data are only visible for users with appropriate credentials. For these data sets, no values will be given when downloading the data using both data files and webservices. Therefore, the toolbox provides a “Login” button in the upper right corner. This button will take users to a login screen where credentials can be entered or requested. After successful login, return to the toolbox. All values will now be visible upon downloading the data set.



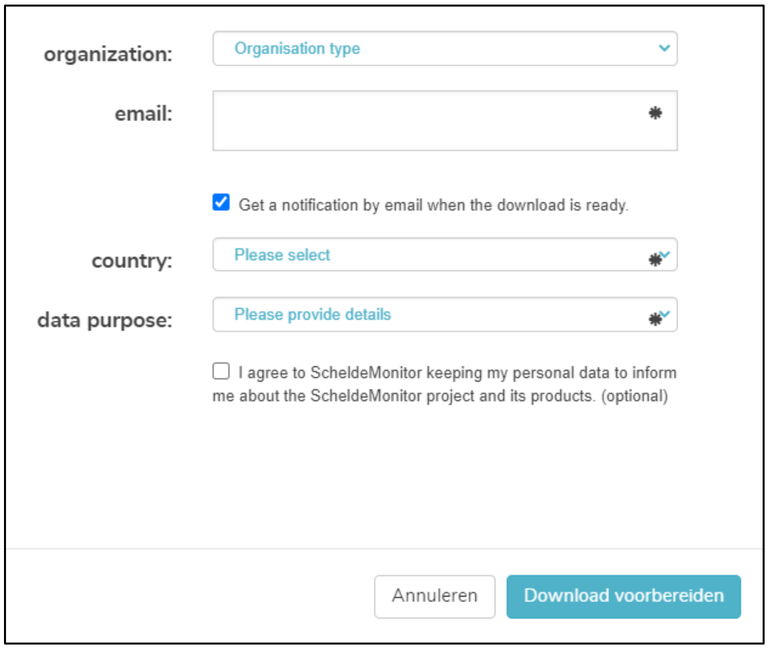
If you have any problems logging in, please contact us at account@vliz.be

[Register] [Login] [Lost password]

4.1 USING DATA FROM DOWNLOAD DATA FILES

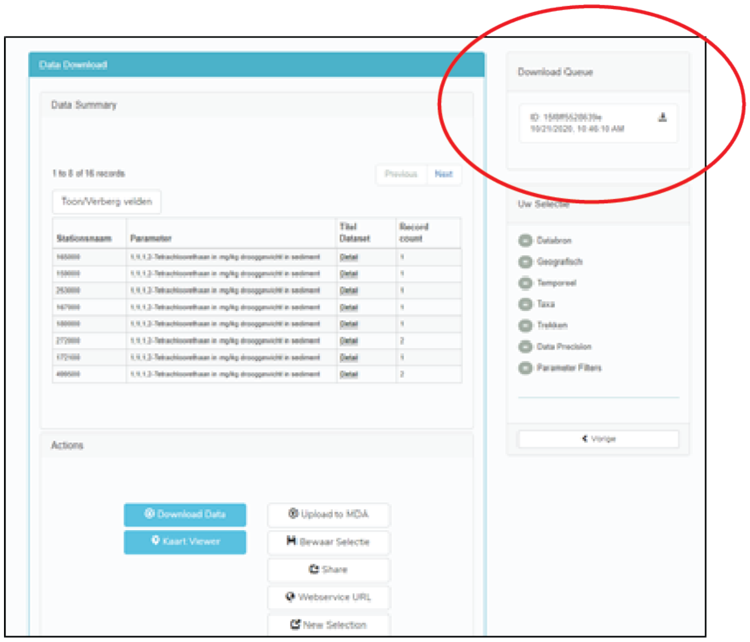
Users can now choose to download the data from the ScheldeMonitor toolbox as data files in a CSV file format. To do so, and to use them in the RStudio environment, the user can perform the following steps:

- ✓ The user selects the “Download Data” button and submits all necessary information to commence his/her download:
  - Organization: Select the type of organization you work for. This is not mandatory.
  - Email: Provide the toolbox with an email address to which a notification can be send on the readiness of your download.
  - Country: Select the country from which the download is done.
  - Data purpose: Select for which purpose the download is done.



organization:	Organisation type
email:	*
	✓ Get a notification by email when the download is ready.
country:	Please select *
data purpose:	Please provide details *
	☐ I agree to ScheldeMonitor keeping my personal data to inform me about the ScheldeMonitor project and its products. (optional)

- ✓ After the necessary information has been submitted, your data will be prepared for download. This preparation can be followed in the upper right corner of the screen. After the preparation is done, a button will be provided by which the download can begin. For large data files, a mail can be sent to a given address to notify a user that the download is fully prepared.


Data Download

Data Summary

1 to 8 of 16 records

Stationsnaam	Parameter	Titel Dataset	Record count
165000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
159000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
263000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
167000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
180000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
272000	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	2
172100	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	1
499500	1,1,1,2-Tetrachloorethaan in mg/kg drooggewicht in sediment	Detail	2

Download Queue

ID: 15f8ff5528639e
10/21/2020, 10:46:10 AM

- ✓ Once the data file is saved on the local drive, the user can load it into the RStudio environment to start working with the data. This can be done by using the basic package of R, by running the following function:

  data = read.csv(“path/file.csv”, stringsAsFactors = FALSE)

  For example:

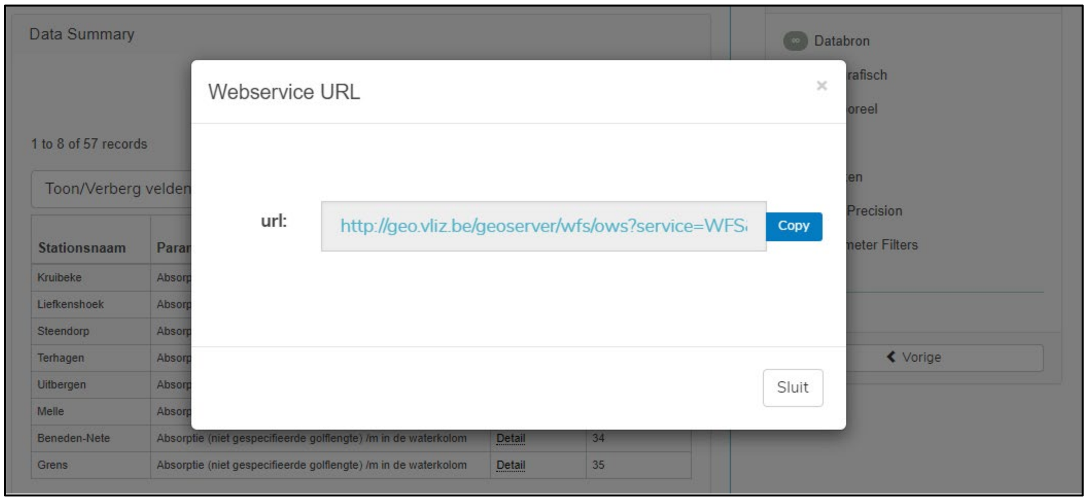
```
data_waterstand = read.csv("Data/hoogwater_combined.csv", stringsAsFactors = FALSE)
```

- ✓ CSV is the only format in which the data files can be downloaded. This format does however have a limit of 1.000.000 records. Larger files will lose records when a user wants to open them in MS Excel before loading them in R. Therefore, users are recommended to open these larger data files as a TXT file, in programs like Notepad++.

4.2 Using data from generic webservices

However, users of the RStudio environment of ScheldeMonitor are urged to make use of the generic webservices that are available in the data download toolbox of ScheldeMonitor. These webservices are a URL format that automatically queries the ScheldeMonitor database without human intervention. The composition of this URL is automatically generated, based on the selection made by the user in the criteria of the data download toolbox. Using webservices has the added advantage that no data files are needed to load in the data set in R, and that the most recent version of the database is queried. The latter implies that when new data is added in the database to an already downloaded data set, the same webservice URL will be able to automatically load in the newly added data. To use the webservices in the RStudio environment:

- ✓ Select the “Webservice URL” option in the data download toolbox, which will give you the URL that is to be used to acquire the selected data set.



Data Summary

1 to 8 of 57 records

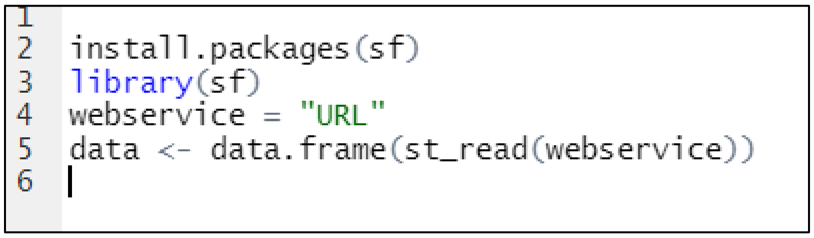
Toon/Verberg velden

Stationsnaam	Param		
Kruibeke	Absorp		
Liefkenshoek	Absorp		
Steendorp	Absorp		
Terhagen	Absorp		
Uitbergen	Absorp		
Melle	Absorp		
Beneden-Nete	Absorptie (niet gespecifieerde golflengte) /m in de waterkolom	Detail	34
Grens	Absorptie (niet gespecifieerde golflengte) /m in de waterkolom	Detail	35

Webservice URL ×

url: http://geo.vliz.be/geoserver/wfs/ows?service=WFS&

- ✓ Once you copied the entire URL, you can use it to load your data into the RStudio environment. Therefore you can use a function in the R-library ‘sf’ and the following lines of code:



```r

install.packages(sf)
library(sf)
webservice = "URL"
data <- data.frame(st_read(webservice))

```

- ✓ Depending on the size of the requested data set, loading the data in R can take up to a minute. Nevertheless, the data set will be available in the environment of the RStudio. The limit of the webservice is capped at around 1.000.000 records per request. Therefore, it is recommended that users generate multiple separate URL’s in the toolbox if they want to analyze more than a million records, and merge the data set in R itself.

5 HELPDESK

VLIZ is responsible to keep the RStudio environment of ScheldeMonitor up and running. Besides foreseeing the necessary server and memory capacity, VLIZ will thus also make sure that all necessary R libraries and packages are installed on the RStudio server. If new libraries and packages need to be installed, users can contact VLIZ to do so.

To accommodate these and other needs of users and contributors, VLIZ will have a permanent helpdesk. This helpdesk can be contacted through the general address of the ScheldeMonitor:

Helpdesk ScheldeMonitor
Data Centre - Local Services & Projects

Vlaams Instituut voor de Zee vzw
Flanders Marine Institute
InnovOcean site, Wandelaarkaai 7
8400 Oostende, Belgium

T +32 (0)59340172
info@scheldemonitor.org
www.vliz.be

For urgent matters or questions, or if users and contributors want to discuss the use of the RStudio environment for certain projects, the project manager of ScheldeMonitor should be contacted:

Jelle Rondelez

Project Manager
Data Centre - Local Services & Projects

Vlaams Instituut voor de Zee vzw
Flanders Marine Institute
InnovOcean site, Wandelaarkaai 7
8400 Oostende, Belgium

M +32 (0)473510828
jelle.rondelez@vliz.be
www.vliz.be