

Tutorial to the RObsDat-package

Dominik Reusser

October 23, 2013

1 Introduction

Research data is a valuable asset. As Gold [2007] put it:

“[...] data is the currency of science, even if publications are still the currency of tenure. To be able to exchange data, communicate it, mine it, reuse it, and review it is essential to scientific productivity, collaboration, and to discovery itself.”

This is also true for hydrological data. However, making data exchangeable and communication requires a well organized data management, for which funding is often not or only to a limited extend available. Efficient methods to manage hydrological data for small and medium scale projects has been missing up to now. What is required is a system that

- stores data in a structured (adhering to standards) and efficient way
- allows centralized or decentralized data management
- is simple to set up with limited or intermediate database skills
- avoids date time errors due to wrong conversion
- provides flexible and efficient ways of data import and data export
- supports and documents data cleaning steps
- provides tools for data analysis
- runs independent of the operating system
- allows inspection of the code for debugging and further development
- provides documentation

The RObsDat package is designed to fill this gap in conjunction with the R environment and is currently in a testing phase where code is made more efficient and more stable. In order to use RObsDat, you need to understand the format used for data storage. This will be presented in section 2. The role sharing between the R environment and the package is briefly touched upon in section 3. Different database engines are supported by the package and basics are presented in section 4. You might decide to initially skip sections 3 and 4 and use the simple example presented in section 5 to learn the practical usage of RObsDat. We conclude (section 6) with a short outlook and an invitation to contribute.

2 Observations Data Model

Different database designs may be used to manage hydrological (and other) data. RObsDat is designed to store point observations which come with the following properties. To store data with a stronger spatial component (such as satellite images and derivatives there off), a (spatio-temporal) GIS system is recommended.

In general, hydrological observations come as a triple including time, variables and space. This triple is usually extended by information about data quality and the data source/collection method. CUASHI has presented a specification for the Observations Data Model (ODM) which is currently available as version 1.1 [Tarboton et al., 2008]. This data model is implemented by RObsDat, however the package has been designed to be potentially compatible with other data models. In addition, CUASHI recommends to use a controlled vocabulary (CV) for the most important attributes to enhance semantic consistency and provides these through a web service. RObsDat is able to import these CV.

Before you can store data with RObsDat, you need to supply meta-data details for the location, the variable, the quality and the source using the commands `addSite`, `addVariable`, `addQualityControlLevel`, and `addSource`. All of these meta-data sets is stored in a separate data table with additional links to specific CV-tables. See the example in section 5 to get started. The full data model is presented in appendix A.

3 Role sharing between RObsDat, R and other packages

A part of the power of managing your data with RObsDat is the embedding environment R. It provides great flexibility for data import, scripting of standard processes, platform independence. RObsDat relies on these advantages of R and attempts to do the management of your data in database well. When you receive data from the database, use the power of R to do subsequent (statistical) analyses and to create plots. Important packages RObsDat depends on are related to the management of time-objects and space-time objects:

3.1 Date formats, zoo/xts

When adding data values, RObsDat expects you to provide time information in the `POSIXct` format. This is the interface between R and the POSIX standard for date/time handling. You may also store your data as time series data in form of a `zoo` or `xts` object. See the corresponding package documentations. You will need a basic understanding of these to work with RObsDat. Import from `spacetime` classes is planned.

3.2 spacetime

The next version of the package will return data as `spacetime` objects, which is a first step to provide possibilities for efficient handling and analysis of spatio-temporal data in R. See the introductory journal article to the package for more detail [Pebesma, 2012].

4 Database engines

RObsDat has been tested and made compatible with three different, open source database engines: SQLite, MySQL and PostgreSQL. The most simple set-up is by using SQLite, which stores all the information in a single file with the extension db. The default is a file in the current working directory and the name RODM.db. You can share the db-file, such that other users can also access your database. Free graphical user interfaces to SQLite databases exist for most operating systems and can be found on the web.

In case you don't know what to do, `getDefaultDB()` will install a preconfigured database file shipped with the package. This command also work if your db file is called RODM.db and is situated in your working directory.

```
> getDefaultDB()
```

To connect to a different SQLite database, you may use the following set of commands:

```
> require("RObsDat")
> require("RSQLite")
> m <- dbDriver("SQLite")
> dbname = "database.db"
> con <- dbConnect(m, dbname = dbname)
> sqhandler <- new("odm1_1Ver", con=con)
> options(odm.handler=sqhandler)
```

However, concurrent data management by multiple users is not advisable with such a set-up. Running a database server is recommended in such a case. Contact your system administrator if you need to set-up a database server. To connect to a running server, you may use the following commands.

```
> #connect to postgresQL database
> require("RObsDat")
> require("RPostgreSQL")
> m <- dbDriver("PostgreSQL")
> con <- dbConnect(m, user="a_user", password="secret", dbname="obsdat")
> sqhandler <- new("odm1_1Ver", con=con)
> options(odm.handler=sqhandler)
> #connect to MySQL database
> require("RObsDat")
> require("RMySQL")
> m <- dbDriver("MySQL")
> con <- dbConnect(m, user="a_user", password="secret", dbname="obsdat")
> sqhandler <- new("odm1_1Ver", con=con)
> options(odm.handler=sqhandler)
```

5 An example session

We start by obtaining a preconfigured database, shipped with the package as RODM.db into our current working directory. All modification will be stored in this file. If you again use `getDefaultDB()` in the same working directory, a

connection will be made to the existing database file and you will have access to all the former modifications. If you want to start with a blank database, make sure to delete the file. However, RObsDat is designed to be smart about repeated execution of commands and will avoid duplication of records. Let's get started:

```
> getDefaultDB()
```

Before adding data values, we need to set up the meta-data. We will check what the correct names of the meta-data are:

```
> #Store metadata in database
```

```
> getMetadata("SpatialReference", SRSName="WGS84", exact=TRUE)
```

	ID	SRSID	SRSName	IsGeographic	Notes
1	3	4326	WGS84	TRUE	

```
> addSite(Code="testLocation", Name="Virtual test site", x=-5, y=46,
+         LatLongDatum="WGS84", Elevation=1500, State="Germany")
> getMetadata("Units", Abbreviation="cm")
```

	ID	Name
1	9	arcminute
2	14	square centimeter
3	32	watts per square centimeter
4	47	centimeter
5	86	centimeter of mercury
6	87	centimeter of water
7	109	centimeters per hour
8	110	centimeters per second
9	136	calories per square centimeter per day
10	137	calories per square centimeter per minute
11	138	milliliters per square centimeter per day
12	149	absorbance per centimeter
13	187	microsiemens per centimeter
14	190	cubic centimeters per gram
15	193	grams per cubic centimeter
16	226	the ratio between UV absorbance at 254 nm and DOC level
17	253	absorbance per centimeter per mg/L of given acid
18	263	millisiemens per centimeter
19	281	cubic centimeters per second
20	292	decisiemens per centimeter
21	294	Joules per square centimeter
22	318	hits per square centimeter
23	319	hits per square centimeter per hour

	Type	Abbreviation
1	Angle	arcmin
2	Area	cm ²
3	Energy Flux	W/cm ²
4	Length	cm
5	Pressure/Stress	cm Hg

```

6      Pressure/Stress      cm H2O
7      Velocity             cm/hr
8      Velocity             cm/s
9      Energy Flux          cal/cm^2 d
10     Energy Flux          cal/cm^2 min
11     Hyporheic flux       ml/cm^2 d
12     Extinction/Absorbance A/cm
13     Electrical Conductivity uS/cm
14     Specific Volume      cm^3/g
15     Concentration        g/cm^3
16     Specific UV Absorbance L/(mg DOC/cm)
17     Absorbance           100/cm mg/L
18     Electrical Conductivity mS/cm
19     Flow                  cm^3/sec
20     Electrical Conductivity dS/cm
21     Energy per Area      J/cm^2
22     Precipitation        hits/cm^2
23     Velocity             hits/cm^2 hr

```

```
> getMetadata("VariableName", Term="Distance")
```

```
Term
```

```
1 Distance
```

```
1 Distance measured from a sensor to a target object such as the surface of a water body o
```

```

> addVariable(Name="Distance", Unit="cm", ValueType="Field Observation",
+   GeneralCategory="Instrumentation", Code="test_dist")
> addQualityControlLevel(ID=6, Code="ok", Definition="The default")
> addISOMetadata(TopicCategory="Unknown", Title="Testdata",
+   Abstract="This data is created to test the functions of RObsDat")
> addSource(Organization="Your Org", SourceDescription="Madeup data",
+   SourceLink="RObsDat Documentation", ContactName="Yourself",
+   Metadata="Testdata")

```

With this set of commands, we are done preparing meta-data for a first Variable and Location. Note that the information about the source is stored in two tables, one including the Dublin Core information and the other remaining meta-data. From the example, it should be simple to add meta-data for other locations or other variables. For the tutorial, we are content with one site and one variable.

As a next step, we can start to add data. Let's create a tutorial data set and store it a xts. We introduce two data errors that we want to correct in a subsequent test.

```

> library(xts)
> example.data <- xts(1:366, seq(as.POSIXct("2010-01-01", tz="UTC"),
+   as.POSIXct("2011-01-01", tz="UTC"), length.out=366))
> example.data[50] <- 100
> example.data[200] <- 40

```

With loggers, we often obtain multiple files with partially redundant data (the later dataset containing some of the information from a previous dataset). RObsDat is able to detect redundant data. As long as no conflicts arise, the package is silent about redundant entries. If present, existing conflicts are presented and the user is asked about the desired action. We simulate this by first importing the first 100 data sets, modifying our data and then importing the remaining data set.

```
> addDataValues(example.data[1:100], Site="testLocation", Variable="test_dist",
+               Source="Madeup", QualityControlLevel="ok")
```

Importing column 1 out of 1

```
> #Avoid duplicates autmatically
> example.data[75] <- 100
> addDataValues(example.data, Site="testLocation", Variable="test_dist",
+               Source="Madeup", QualityControlLevel="ok")
```

Importing column 1 out of 1

Data to import is not matching data in database for 1 values (See plot)

What shall I do?

- 1) Dischard data to import and import remaining, missing data
- 2) Overwrite values in database with new values
- 0) Stop and let you modify the data to import before another attempt

Enter a number (0-2) for your choise.

Note the "split responsibility" between the R environment and RObsDat (section 3). You will use the power and flexibility of R to get the data into the environment and convert dates into a POSIX-format, while the package provides support mechanisms to get the data in a consistent form into the database. Note that the import method is quite smart about data configurations and automatically detects whether meta-data is valid for columns or rows or is provided as full table (check this out yourself and report unexpected behaviour to the maintainer). Also, typos and multiple versions for site and variable names are interactively caught and the database remembers synonyms for data sets. Data in the database can be accessed and filtered in an intuitive way:

```
> inDB <- getDataValues(Site="test")
> plot(inDB)
```

Raw data often needs to be cleaned before it can be used. RObsDat supports this and allows to reconstruct data modification operations. The following lines demonstrate this:

```
> #Version management
> to.correct <- which(inDB@values < 100 &
+                   index(inDB@values) > as.POSIXct("2010-06-01"))
> inDB@values[to.correct] <- 200
> inDB@values[50] <- 50
> updateDataValues(inDB, "Correction of wrong value")
> ver2 <- inDB
```

```

> ver2@values[50:60] <- 90
> updateDataValues(ver2, "Changing more data")
> ver3 <- inDB
> ver3@values[50:60] <- 190
> updateDataValues(ver3, "Ups, I used 90 instead of 190 by mistake")
> deleteDataValues(inDB[250], "And finally remove a value")

NULL

> getDataVersions()

  VersionID      ValidUntil      VersionComment
1         1 2013-10-23 09:35:34      Initial Version
2         2 2013-10-23 09:35:34      Correction of wrong value
3         3 2013-10-23 09:35:35      Changing more data
4         4 2013-10-23 09:35:36 Ups, I used 90 instead of 190 by mistake
5         5                <NA>      And finally remove a value

> versionQuery <- getDataValues(Site=1, VersionID=1)
> plot(versionQuery)
> versionQuery <- getDataValues(Site=1, VersionID=2)
> plot(versionQuery)

```

6 Conclusion

Ideally RObsDat will be a valuable tool for you to manage your data. It is currently still under development. You can contribute by testing the package and reporting what works and what doesn't, by improving the documentation, by making some funding available for further development...

A The full data model

References

- Anna Gold. Cyberinfrastructure, Data, and Libraries, Part 1: A Cyberinfrastructure Primer for Librarians. *D-Lib Magazine*, 13(9/10):september20september-gold-pt1, 2007. doi: 10.1045/september20september-gold-pt1. URL <http://www.dlib.org/dlib/september07/gold/09gold-pt1.html>.
- Edzer Pebesma. spacetime: Spatio-Temporal Data in R. *Journal of Statistical Software*, 51(7):1–26, 2012.
- David G Tarboton, Jeffery S Horsburgh, and David R Maidment. CUAHSI Community Observations Data Model (ODM) Version 1.1 Design Specifications, 2008.

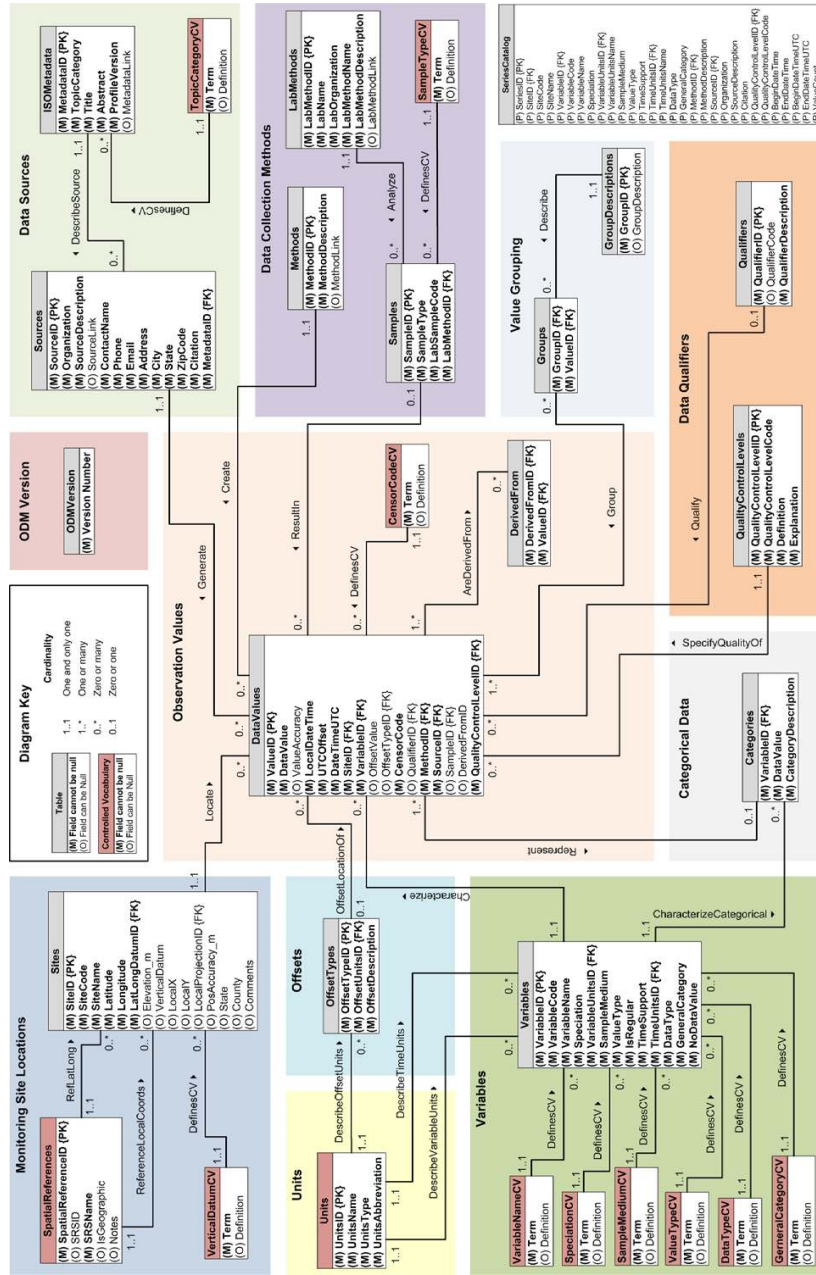


Figure 1: Observations Data Model as specified in Tarboton et al. [2008]. Image source: Tarboton et al. [2008]