R Guide

Version: *3.3.0*
Date: *17.09.2019*
Status: *Valid*
Author: *A. Hueni, Remote Sensing Laboratories, University of Zurich*
File: *\SPECCHIO_R_Guide.docx*
Pages: *16*

Classification:
Distribution: *SPECCHIO Users*

History

Version	Date	Author	Remark
3.1.0	04.07.2014	A. Hueni	First version for SPECCHIO V3.1
3.3.0	17.09.2019	A. Hueni	Updates for account configuration

Table of Contents

1 Introduction

The R environment is a popular tool in research and science. This integration of SPECCHIO with R bases on the rJava library[1], allowing the use of Java classes within R code.
The code introduced in this guide has been tested with version rJava_0.9-4.
The integration has been tested using RStudio; using other R environments are at the end-users own risk.
This guide is targeted at SPECCHIO Version 3.1 or higher.

[1] http://cran.r-project.org/web/packages/rJava/index.html

2 Installation and Configuration

2.1 Setup

2.1.1 Definition of the Java Classpath

The java classpath of the rJava package needs to be updated to include the SPECCHIO classes. This can be done dynamically within each R script using the SPECCHIO functionality by specifying the path to the specchio_client.jar file.

Under MacOSX for the installation compiled as MacOS application, the default path would be:

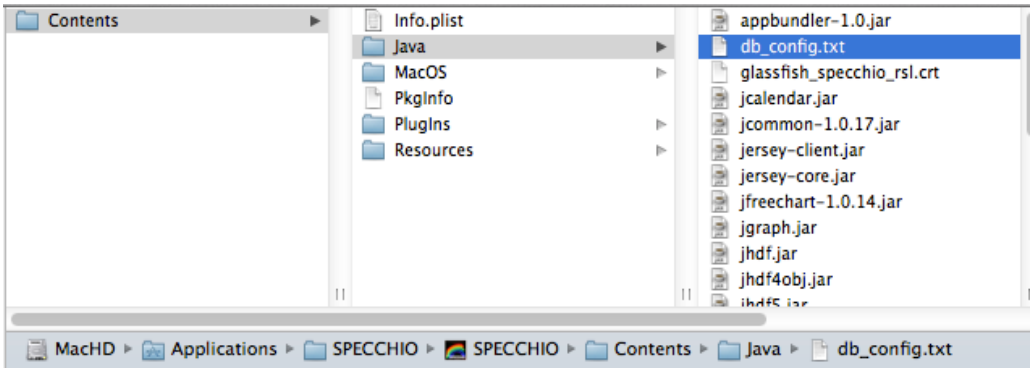
```
.jaddClassPath(path="/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-client.jar")
```

For the cross-platform installation, the path would depend on your installation path:

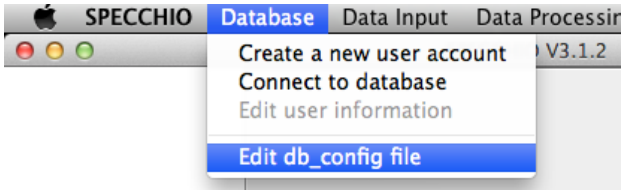
```
.jaddClassPath(path="<installation-folder>/specchio-client.jar")
```

2.1.2 Account Configuration

Using SPECCHIO from R automatically uses your existing account configurations, either stored in the Java Preferences on the operating system, or stored within a file named db_config.txt.
The db_config.txt file is found in your installation path of SPECCHIO in the same directory as the specchio-client.jar file.
E.g. under MacOSX:



To see the content of the db_config.txt start the SPECCHIO Java application and select the *Database* and *Edit db_config file* menu item:



This will open the db_config.txt file in a text editor.

The order of the connections within this configuration file will be the identical to their order within a list of SPECCHIOServerDescriptor objects. The server descriptors are returned as a Java ArrayList and are therefore indexed from zero onwards.
The code to select a connection from the ones stored in the configuration file and connect to SPECCHIO is covered below (see Section 3).

3 Connecting to the SPECCHIO Database

Use the `SPECCHIOClientFactory` class to open a connection to a SPECCHIO database instance:

```
# connect to server
specchio_cf <- J("ch/specchio/client/SPECCHIOClientFactory")$getInstance()
```

This call reads the known database connections from the SPECCHIO db_config.txt file. All known server connections can be returned as an ArrayList object by calling the `getAllServerDescriptors` method;

```
descList <- specchio_cf$getAllServerDescriptors()
```

Connect to any entry in the descriptor list by calling the `createClient` method, parameterized with an entry from the descriptor list. E.g. to connect to the first entry in the list, get the zero-th entry (Java lists are indexed from zero onwards):

```
# connect using first connection description
desc <- descList$get(as.integer(0))
specchio_client <- specchio_cf$createClient(desc)
```

The `SPECCHIOClient` class uses the Singleton pattern (Gamma, Helm et al. 1997). This implies that once the connection is set, all SPECCHIO objects connecting to the database automatically utilise this connection.
The specchio_client object provides all methods to interact with the SPECCHIO database.

To get the number of database connections stored in the db_config.txt file use:

```
descList$size()
```

To list a specific entry, use:

```
descList$get(as.integer(2))
```

```
[1] "Java-Object{https://AHueni@v473.vanager.de:443/specchio_service@jdbc/specchio_prod}"
```

4 Interaction with SPECCHIO

4.1 Preparing Queries

Queries conditions are used to select spectral data from the database. These can be defined manually, but the easiest way is to generate queries using the SPECCHIO Data Browser or Query Builder and use these in R.

These automatically created queries are a good starting point to programmatically create query conditions, e.g. select data within a loop with changing query conditions.

4.1.1.1 Using the Data Browser to form Query Conditions

Start the regular SPECCHIO Java Application, open the Data Browser and select the required data. Then click the menu button over the 'Matching Spectra' text area and select 'Copy R-ready query to clipboard'.

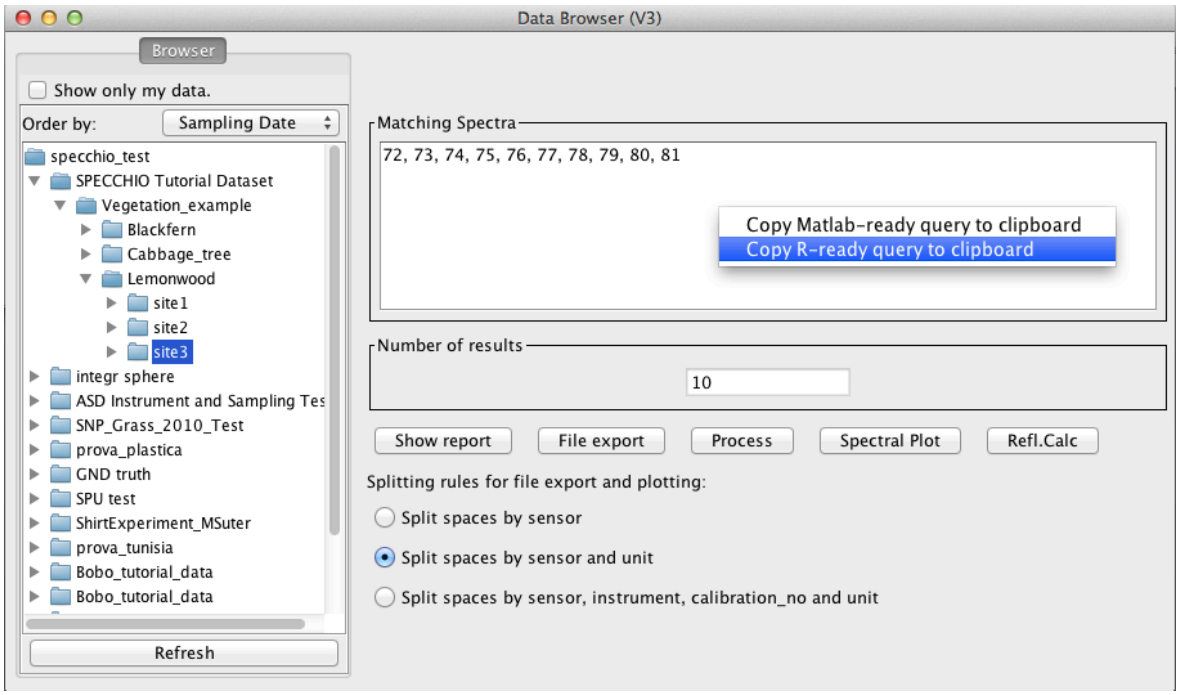


Figure 1: R query pop-up menu in the Data Browser

4.1.1.2 Using the Query Builder to form Query Conditions

Start the regular SPECCHIO Java Application, open the Query Builder and select the required data. Then click the menu button over the 'Matching Spectra' text area and select 'Copy R-ready query to clipboard'.

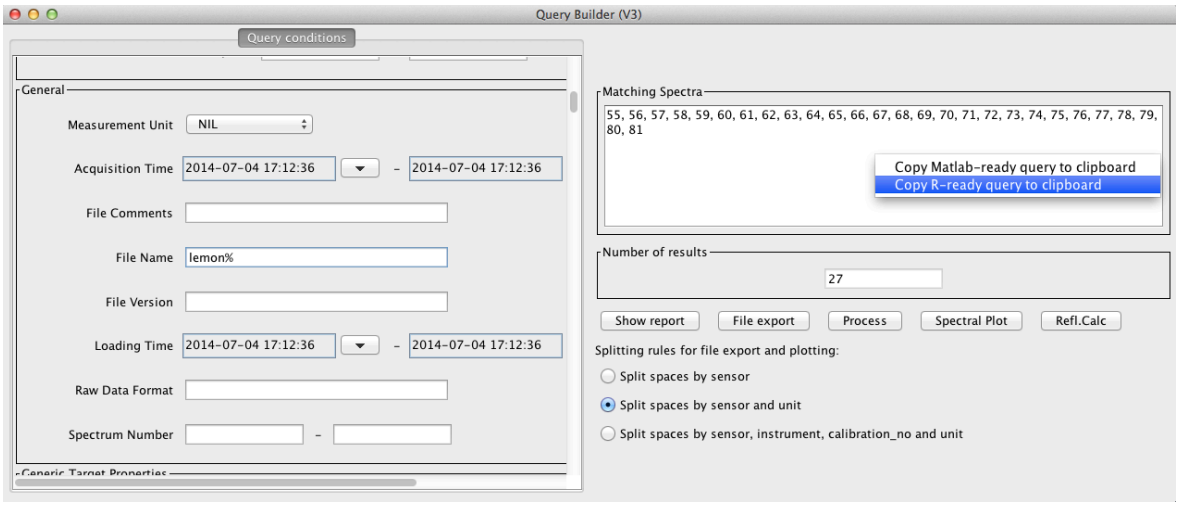


Figure 2: R query pop-up menu in the Query Builder

4.2 Retrieving and Plotting Spectral Data

Data is retrieved from the database by the following steps:

Paste the query conditions (see 4.1 on how to create queries) in your R file.
This will be similar to:

```
query <- .jnew("ch/specchio/queries/Query", "spectrum")
query$setQueryType(query$SELECT_QUERY)

query$addColumn("spectrum_id")

cond <- .jnew("ch/specchio/queries/QueryConditionObject", "spectrum", "spec-
trum_id");
id_array <- c(72,73,74,75,76,77,78,79,80,81)
ids_list <- .jnew("java.util.ArrayList")
for (i in 1:length(id_array)) { ids_list$add(.jnew("java.lang.Integer",
as.integer(id_array[i]))) }
cond$setValue(ids_list);
cond$setOperator("in");
query$add_condition(cond);

ids <- specchio_client$getSpectrumIdsMatchingQuery(query);
```

This code returns the spectrum ids that identify the spectra matching the query conditions.

Call the `getSpaces` method (set to split spaces by sensor only and order the matching entries by their acquisition time) (see 5.1.1):

```
spaces <- specchio_client$getSpaces(ids, TRUE, FALSE, "Acquisition Time")
```

The spaces object is an array of spaces created by the Spectral Space Factory, a service offered by the SPECCHIO server and essentially creating containers to hold spectra of different wavelength dimensions.
To figure out how many spaces were returned, use:

```
spaces$length

[1] 1
```

To get a particular space, use the standard R array syntax, e.g. get the first element of the space array:

```
space <- spaces[[1]] # first element of returned array
```

Spaces are just pre-configured containers to hold spectral vectors. To actually populate a space, use the `loadSpace` method, supplying the space required loading as an argument:

```
space <- specchio_client$loadSpace(space);
```

Once a space is loaded use the method `getVectorsAsArray` to get the spectra held by the space as array and the method `getAverageWavelengths` to get the centre wavelengths (i.e. average of the spectral response function per sensor band):

```
# get spectra and wvls
vectors <- space$getVectorsAsArray()
wvls <- space$getAverageWavelengths()
```

To convert the Java array holding the vectors as R matrix, use

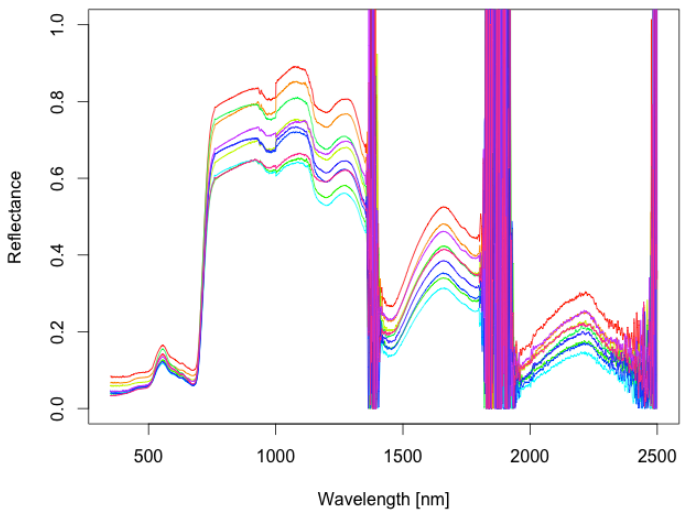
```
# convert java array to R matrix
r_matrix <- .jevalArray(vectors, simplify=TRUE)
```

The space holds information about the measurement unit as well, hence, all spectra can be plotted and the plot annotated as follows:

```
plot(r_matrix[1,]~wvls,type="n", ylim=c(0, 1), xlab="Wavelength [nm]" ,
ylab=space$getMeasurementUnit()$getUnitName()) # create empty plot

col = rainbow(space$getNumberOfDataPoints())


for (i in 1:space$getNumberOfDataPoints() ) {
  lines(r_matrix[i,]~wvls, col=col[i])
}
```



4.3 Retrieving Metadata

Metadata are easily retrieved for a selection of ids by calling the getMetaparameterValues method of the SPECCHIOClient (5.1.2) class, e.g.:

```
latitudes <- specchio_client$getMetaparameterValues(ids, "Latitude")
```

If data are numerical, they are returned as a special list object of the class MatlabAdaptedArrayList, allowing an easy conversion of the data held by the returned list into a Matlab or R array via the method get_as_double_array().
Otherwise, e.g. for string data types, values are returned in a Java ArrayList.

For an example on how to read metadata from SPECCHIO see the example provided in section 6.2.

4.4 Updating Metadata

Existing metadata can be updated programmatically by using the updateEavMetadata method of the SPECCHIOClient () class.

4.5 Inserting new Metadata

New metadata can be inserted programmatically by using the updateEavMetadata method of the SPECCHIOClient () class.
The following example inserts a new Boolean metaparameter to flag a selection of dark current spectra.

```
% get attribute object for the metadata attribute 'DC Flag'
flag_attribute = specchio_client.getAttributesNameHash().get('DC Flag');

% insert flag by creating a new metaparameter, setting the value and updating the
metadata of selected spectra
e = MetaParameter.newInstance(flag_attribute);
e.setValue(1);

specchio_client.updateEavMetadata(e, ids);
```

5 SPECCHIO API

5.1 SPECCHIOClient

5.1.1 getSpaces

Get the space objects for a set of spectrum identifiers.

```
* @param ids                          the spectrum identifiers
* @param split_spaces_by_sensor       boolean
* @param split_spaces_by_sensor_and_unit     boolean
```

```
     * @param order_by                          the field to order by
```

The Boolean flags allow to configure the space building as in the SPECCHIO Data Browser or Query Builder Interfaces.

5.1.2 getMetaparameterValues

Get values for spectrum ids and EAV attribute.

```
     *
     *
     * @param ids       spectrum ids
     * @param attribute attribute name
     *
     * @return list of values, or null if the field does not exist
```

Note: There is currently no check implemented to ascertain that all spectra defined by the ids do have such an attribute defined; it can thus happen that the number of values returned is lower than the number of spectra in the list!

5.1.3 updateEavMetadata

Updates or inserts a metaparameter for a specified list of spectra.

```
* @param mp               the meta-parameter to update
* @param spectrum_ids     the identifiers for which to update the parameter
*
* @return the identifier of the inserted metadata

updateEavMetadata(MetaParameter mp, ArrayList<Integer> spectrum_ids)


* @param mp               the meta-parameter to update
* @param spectrum_ids     the identifiers for which to update the parameter
* @param mp_old           the old meta-parameter
*
* @return the identifier of the inserted metadata
*/
public int updateEavMetadata(MetaParameter mp, ArrayList<Integer> spectrum_ids,
MetaParameter mp_old)
```

6 Examples

6.1 Function to get Spectral Data from SPECCHIO

This script is the complete example of code introduced in sections **Error! Reference source not found.** - 4.2.

```
library(rJava)

.jinit()
```

```
.jaddClassPath(path="/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-
client.jar")

specchio_cf <- J("ch/specchio/client/SPECCHIOClientFactory")$getInstance()

desc <- specchio_cf$getAllServerDescriptors()

desc1 <- desc$get(as.integer(2))

specchio_client <- specchio_cf$createClient(desc1)


# Query pasted from SPECCHIO Query builder:
# >>>>>>>>>>>>>>>>>>>>>>>>
query <- .jnew("ch/specchio/queries/Query", "spectrum")
query$setQueryType(query$SELECT_QUERY)

query$addColumn("spectrum_id")

cond <- .jnew("ch/specchio/queries/QueryConditionObject", "spectrum", "spec-
trum_id");
id_array <- c(72,73,74,75,76,77,78,79,80,81)
ids_list <- .jnew("java.util.ArrayList")
for (i in 1:length(id_array)) { ids_list$add(.jnew("java.lang.Integer",
as.integer(id_array[i]))) }
cond$setValue(ids_list);
cond$setOperator("in");
query$add_condition(cond);

ids <- specchio_client$getSpectrumIdsMatchingQuery(query);

#<<<<<<<<<<<<<<<<<<<<<<<<

spaces <- specchio_client$getSpaces(ids, TRUE, FALSE, "Acquisition Time")

space <- spaces[[1]] # first element of returned array

ids <- space$getSpectrumIds() # get properly ordered ids !
space <- specchio_client$loadSpace(space);


# get spectra and wvls
vectors <- space$getVectorsAsArray()
wvls <- space$getAverageWavelengths()

# convert java array to R matrix
r_matrix <- .jevalArray(vectors, simplify=TRUE)

plot(r_matrix[1,]~wvls,type="n", ylim=c(0, 1), xlab="Wavelength [nm]" ,
ylab=space$getMeasurementUnit()$getUnitName()) # create empty plot
col = rainbow(space$getNumberOfDataPoints())


for (i in 1:space$getNumberOfDataPoints() ) {
  lines(r_matrix[i,]~wvls, col=col[i])
}
```

6.2 Get Metadata of Spectra

This script selects data held by the SPECCHIO online test database (server: `v473.vanager.de`, datasource: `jdbc/specchio_test`) and retrieves their spatial metadata.

First, all Blackfern spectra are selected from the database via a File Name restriction. Then meta-parameters ('Latitude'/'Longitude') are retrieved for the selected spectra and plotted as a scatter plot.

```
library(rJava)

.jinit()
.jaddClassPath(path="/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-client.jar")


specchio_cf <- J("ch/specchio/client/SPECCHIOClientFactory")$getInstance()

desc <- specchio_cf$getAllServerDescriptors()

desc1 <- desc$get(as.integer(2))

specchio_client <- specchio_cf$createClient(desc1)


# Query pasted from SPECCHIO Query builder and manually edited to just set conditions for the File Name attribute:
# >>>>>>>>>>>>>>>>>>>>>>
query <- .jnew("ch/specchio/queries/Query", "spectrum")
query$setQueryType(query$SELECT_QUERY)

query$addColumn("spectrum_id")

cond <- .jnew("ch/specchio/queries/EAVQueryConditionObject", "eav", "spectrum_x_eav", "File Name", "string_val");
cond$setValue("%%bfern%%");
cond$setOperator("like");
query$add_condition(cond);

ids <- specchio_client$getSpectrumIdsMatchingQuery(query);

#<<<<<<<<<<<<<<<<<<<<<<<<

spaces <- specchio_client$getSpaces(ids, TRUE, FALSE, "Acquisition Time")

space <- spaces[[1]] # first element of returned array

ids <- space$getSpectrumIds() # get properly ordered ids !
space <- specchio_client$loadSpace(space);

latitudes <- specchio_client$getMetaparameterValues(ids, "Latitude");
longitudes <- specchio_client$getMetaparameterValues(ids, "Longitude");



plot(latitudes$get_as_double_array()~longitudes$get_as_double_array(),
xlab="Longitudes", ylab="Latitudes")
```

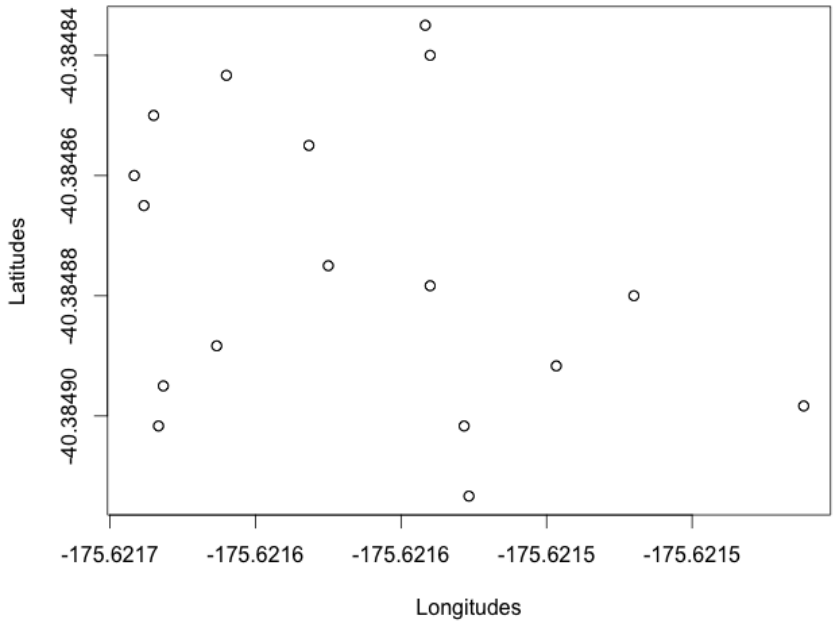


Figure 3: lat-lon plot

6.3 Generating Time Series

~~Example of how to construct a R timeseries object from acquisition time stamps and plot the value of one sensor band over time.~~

ATTENTION: Surely someone regularly using R would figure out how to build irregular timeseries from the datetime strings extracted from SPECCHIO! Be my guest and let me know …

```
library(rJava)

.jinit()
.jaddClassPath(path="/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-
client.jar")


specchio_cf <- J("ch/specchio/client/SPECCHIOClientFactory")$getInstance()

desc <- specchio_cf$getAllServerDescriptors()

desc1 <- desc$get(as.integer(2))

specchio_client <- specchio_cf$createClient(desc1)


# Query pasted from SPECCHIO Query builder and manually edited to just set condi-
tions for the File Name attribute:
# >>>>>>>>>>>>>>>>>>>>>>
query <- .jnew("ch/specchio/queries/Query", "spectrum")
query$setQueryType(query$SELECT_QUERY)

query$addColumn("spectrum_id")

cond <- .jnew("ch/specchio/queries/QueryConditionObject", "spectrum", "spec-
trum_id");
id_array <- c(18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34)
```

```
ids_list <- .jnew("java.util.ArrayList")
for (i in 1:length(id_array)) { ids_list$add(.jnew("java.lang.Integer",
as.integer(id_array[i]))) }
cond$setValue(ids_list);
cond$setOperator("in");
query$add_condition(cond);

ids <- specchio_client$getSpectrumIdsMatchingQuery(query);


#<<<<<<<<<<<<<<<<<<<<<<<<<<<

spaces <- specchio_client$getSpaces(ids, TRUE, FALSE, "Acquisition Time")

space <- spaces[[1]] # first element of returned array

ids <- space$getSpectrumIds() # get properly ordered ids !
space <- specchio_client$loadSpace(space);

vectors <- space$getVectorsAsArray()
wvls <- space$getAverageWavelengths()

# convert java array to R matrix
r_matrix <- .jevalArray(vectors, simplify=TRUE)


acq_times <- specchio_client$getMetaparameterValues(ids, "Acquisition Time");

formatter <- .jnew("java.text.SimpleDateFormat", "dd.mm.yyyy hh:mm:ss")


date_strs <- character(space$getNumberOfDataPoints())

r_time <- numeric()
r_time <- c(r_time, space$getNumberOfDataPoints())

for (i in 0:(space$getNumberOfDataPoints()-1) ) {
  date_strs[i+1] <- formatter$format(acq_times$get(as.integer(i)))
}

# TO BE DONE: convert these date strings into some form that allows irregualr
time series in R ....

r_times <- strptime(date_strs, "%d.%m.%Y %H:%M:%S")

# plot(r_matrix[,400]~r_times, xlab="Time", ylab="R at ASD band 400") # this is
not how you plot a time series ...
```

7 Building Graphical User Interfaces

The building of interactive graphical user interfaces using SPECCHIO components has already been demonstrated with Matlab[2]:
It appears feasible to achieve the same with R with tools like FGUI or Deducer. If anyone has such experience and would like to contribute please contact the author of this document by email: ahueni@geo.uzh.ch.

[2] For more information please visit http://www.geo.uzh.ch/en/units/rsl/news/2013/130218

8 References

Gamma, E., R. Helm, et al. (1997). Design Patterns, Elements of Reusable Object-Oriented Software. Reading, MA, Addison Wesley Longman Inc.