Matlab Guide

Version: *3.3.0*
Date: *23.03.2023*
Status: *Valid*
Author: *A. Hueni & D. Kuekenbrink, Remote Sensing Laboratories, University of Zurich*
File: *\SPECCHIO_Matlab_Guide.docx*
Pages: *19*

Classification:
Distribution: *SPECCHIO Users*

History

Version	Date	Author	Remark
0.1.0	18.05.2012	A. Hueni & D. Kueken-brink	First version
0.2.0	20.05.2012	A. Hueni & D. Kueken-brink	Added static classpath definition and a metadata extraction example.
0.3.0	26.07.2012	A. Hueni	Enhanced SPECCHIO API (sampling geometry data extraction) and according examples
3.1.0	06.07.2014	A. Hueni	Updates and complete revamp for V3
3.1.1	09.11.2014	A. Hueni	Note regarding static classpath
3.3.0	17.09.2019	A. Hueni	Additional method for static classpath definition. Note on problems with dynamic classpath. Updates related to account configuration management.
3.3.0	1.07.2022	A. Hueni	Added more info on the errors encountered under Windows installations related to the classpath settings
3.3.0	27.3.2023	A. Hueni	Updated the metadata extraction code (getMetaparameterValues) to the new SpatialPosition class.

Table of Contents

1 Introduction

The Matlab environment is a well-established tool in engineering, research and science. Matlab includes a Java Virtual Machine and thus allows the use of Java classes within Matlab code. Starting from version 2.1.2, SPECCHIO includes new methods that allow the easy integration of SPECCHIO classes, giving easy access to the spectral data and metadata stored in SPECCHIO databases.
This guide is targeted at SPECCHIO Version 3.1 or higher.

2 Installation and Configuration

2.1 Setup

2.1.1 Definition of the Java Classpath

The java classpath of Matlab needs to be updated to include the SPECCHIO classes. This can be done in a static or dynamic fashion. The non-static definition may be the only option in case Matlab is run from a central server where users cannot change the static classpath.

2.1.1.1 Static Definition of the Java Classpath

In the Matlab prompt type: edit classpath.txt
Add the following line to your classpath.txt file (change the path to reflect your installation location).

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

```
/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-client.jar
```

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

```
<installation-folder>/specchio-client.jar
```

The editing of the classpath.txt may not work on some machines due to missing administrator rights. In this case, the following code[1] can be used to create a javaclasspath.txt file. specchio_pathname is the path pointing to the location of the SPECCHIO jar files.

```
if ~exist(fullfile(prefdir, 'javaclasspath.txt'),'file')
   try
      fid = fopen(fullfile(prefdir, 'javaclasspath.txt'), 'wt');
      fprintf(fid, fullfile(specchio_pathname, 'specchio-client.jar'));  % add jar file
      fclose(fid);
      system(['./' mfilename ' &']);
      uiwait(msgbox('Please restart Matlab for Java path configuration to take effect', 'Information', 'warn'));
   catch err
      msg = sprintf('Could not create %s/javaclasspath.txt - error was: %s', pwd, err.message);
      uiwait(msgbox(msg, 'Error', 'warn'));  % need uiwait since otherwise exit below will delete the msgbox
   end
   exit
end
```

Notes:

- Restart Matlab for the changes to take effect
- On some computers the static definition may not work, e.g. in one instance when using Matlab 2012 release. The error will be that SPECCHIO java classes cannot be found when trying to create them. In such cases it appears that the dynamic definition of the classpath does solve the problem. If so, do remove the SPECCHIO classpath entries from the classpath.txt file before defining the dynamic classpath.

2.1.1.2 Non-static Definition of the Java Classpath

Alternatively you can you can define your Java-classpath in the following way:
Create a new m-file in Matlab that you intend to use for your Matlab code with direct SPECCHIO access. First you need to declare the path of the classes of the SPECCHIO distribution. For that, type in `javaaddpath ({})`
In between the curly brackets, you need to declare the path of the specchio-client.jar file.

[1] https://undocumentedmatlab.com/blog/static-java-classpath-hacks

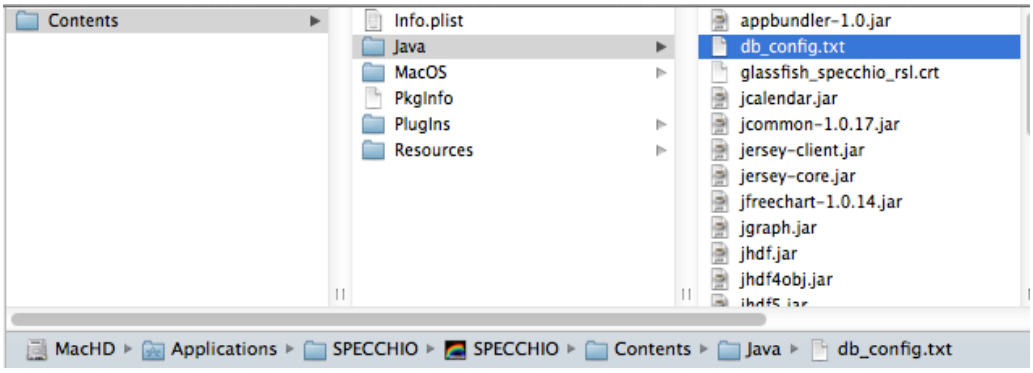
The code should look like this:

```
javaaddpath ({'/Applications/SPECCHIO/SPECCHIO.app/Contents/Java/specchio-
client.jar '})
```

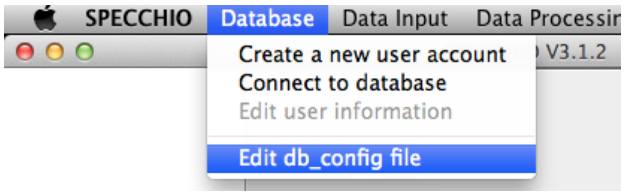
Attention: depending on your Matlab version, the non-static definition will not work and lead to 'class not found' errors. In this case, please use one of the static classpath definition versions. Recent tests show that the dynamic classpath definition does NOT work on Windows machines. For more info see also the Troubleshooting section (8.1) at the end of this guide.

2.1.2 Account Configuration

Using SPECCHIO from Matlab 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, but must be explicitly created by selecting 'Enable editing of db_config file' in the SPECCHIO preferences.
db_config file under MacOSX:



To easily 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 in Section 3.

2.1.2.1 Editing the db_config.txt File

Matlab release R2011a (and maybe others as well; 2018 and 2019 work perfectly well with HTTPS) includes a bug that prevents the use of https connections via keystore entries[2]. To circumvent this, use an unencrypted data connection to SPECCHIO by connecting to the 8080 http port. To achieve this, open the db_config.txt file in a text editor and copy and change the existing https entries you want to use to http and port 8080, e.g.:

```
# Database connections
# Attention: be careful to keep this file local if you specify your password!
#
# Order of fields: protocol, server, port, path, user_name, password
# Note: apart from the server name, any of the fields can be left blank
#
https, v473.vanager.de, 443, /specchio_service, AHueni, XXXXXXXX, jdbc/specchio_test
```

should read after editing:

```
# Database connections
# Attention: be careful to keep this file local if you specify your password!
#
# Order of fields: protocol, server, port, path, user_name, password
# Note: apart from the server name, any of the fields can be left blank
#
https, v473.vanager.de, 443, /specchio_service, AHueni, XXXXXXXX, jdbc/specchio_test
http, v473.vanager.de, 8080, /specchio_service, AHueni, XXXXXXXX, jdbc/specchio_test
```

2.1.3 Importing the SPECCHIO Packages

You need to import several packages, so that MATLAB knows all the Java classes and methods needed to run the SPECCHIO application. These imports should appear in your Matlab code before any SPECCHIO classes are instantiated.

```
import ch.specchio.client.*;
import ch.specchio.queries.*;
```

3 Connecting to the SPECCHIO Database

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

```
% connect to server
cf = SPECCHIOClientFactory.getInstance();
```

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

```
descriptor_list = 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):

[2] http://stackoverflow.com/questions/8740871/matlab-java-interface-java-io-ioexception-the-issuer-can-not-be-found-in-the-t

```
% connect to first connection description
specchio_client = cf.createClient(descriptor_list.get(0));
```

The `SPECCHIOClient` class is using 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:

```
descriptor_list.size()
```

To list a specific entry, use:

```
K>> descriptor_list.get(0)

ans =

http://AHueni@v473.vanager.de:8080/specchio_service@jdbc/specchio_test
```

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 Matlab.
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 Matlab-ready query to clipboard’.

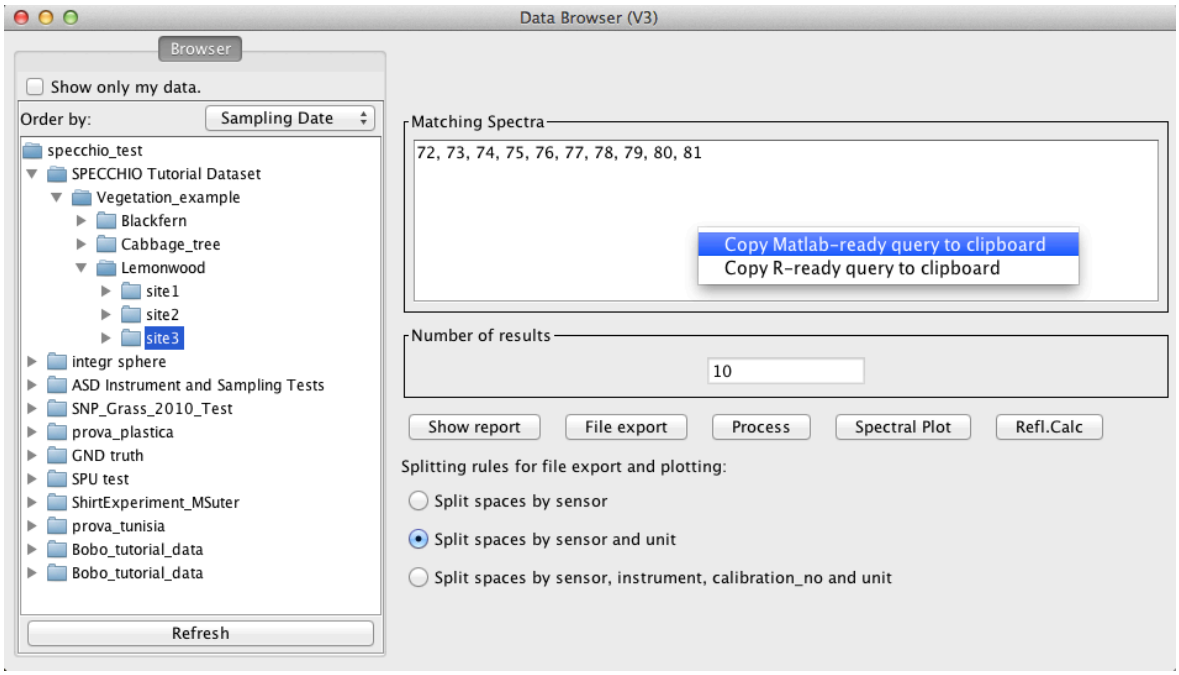


Figure 1: Matlab query pop-up menu in the Data Browser

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 Matlab-ready query to clipboard'.

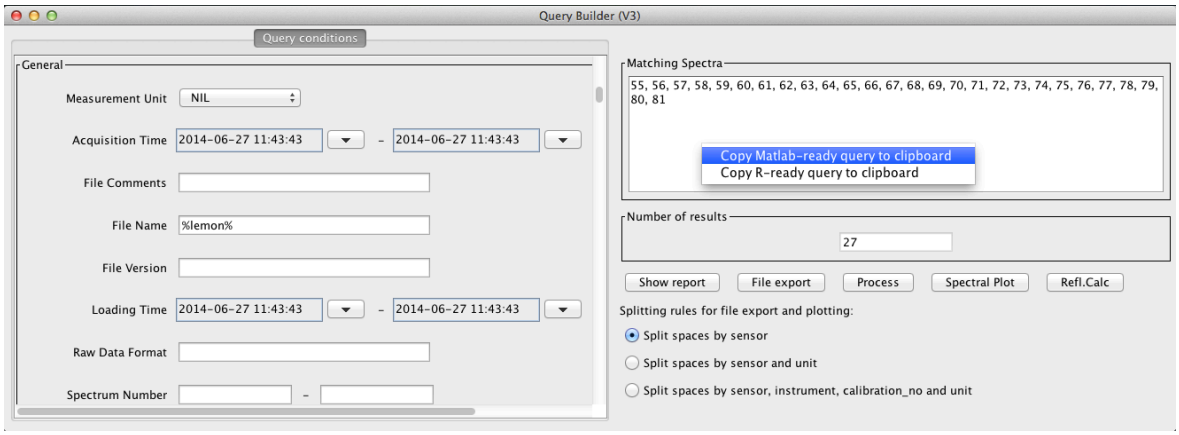


Figure 2: Matlab query pop-up menu in the Query Builder

4.2 Retrieving and Plotting Spectral Data

Data is retrieved from the database by the following steps:

Paste the query conditions (see 4 on how to create queries) in your Matlab file.

This will be similar to:

```
query = Query('spectrum');
query.setQueryType(Query.SELECT_QUERY);

query.addColumn('spectrum_id')

cond = QueryConditionObject('spectrum', 'spectrum_id');
id_array = [72,73,74,75,76,77,78,79,80,81];
ids_list = java.util.ArrayList();
for i=1:size(id_array,2) ids_list.add(id_array(i)); end;
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, 1, 0, '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

ans =

         1
```

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

```
space = spaces(1);
```

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);
```

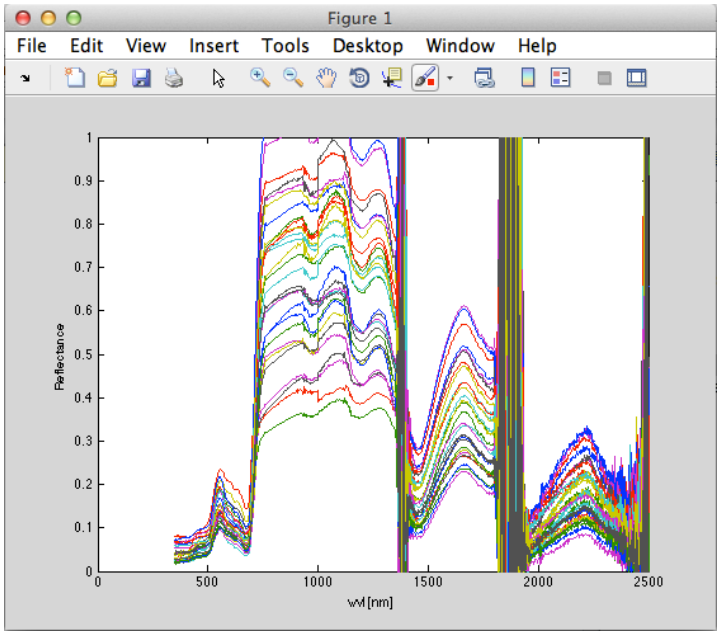
Once a space is loaded use the method `getVectorsAsArray` to get the spectra held by the space as Matlab array and the method `getAverageWavelengths` to get the centre wavelengths (i.e. average of the spectral response function per sensor band):

```
vectors = space.getVectorsAsArray();
wvl = space.getAverageWavelengths();
```

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

```
figure;
```

```
plot(wvl, vectors);
ylim([0 1]);
xlabel('wvl [nm]')
ylabel(char(space.getMeasurementUnit().getUnitName()));
```



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.:

```
altitudes = specchio_client.getMetaparameterValues(ids, 'Altitude');
```

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 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 dark current spectrum.

```
% 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 function is the complete example of code introduced in sections 2.1.3 - 4.2.

```
function SPECCHIO_V3_test()

    import ch.specchio.client.*;
    import ch.specchio.queries.*;

    % connect to server
    cf = SPECCHIOClientFactory.getInstance();
    descriptor_list = cf.getAllServerDescriptors();
    specchio_client = cf.createClient(descriptor_list.get(0)); % connect to first
connection description



    % Query pasted from SPECCHIO Query builder:
    % >>>>>>>>>>>>>>>>>>>>>>
    query = Query('spectrum');
    query.setQueryType(Query.SELECT_QUERY);

    query.addColumn('spectrum_id')

    cond = SpectrumQueryCondition('spectrum', 'measurement_unit_id');
    cond.setValue('1');
    cond.setOperator('=');
    query.add_condition(cond);

    cond = EAVQueryConditionObject('eav', 'spectrum_x_eav', 'Spectrum Number',
'int_val');
    cond.setValue('1');
    cond.setOperator('>=');
    query.add_condition(cond);

    cond = EAVQueryConditionObject('eav', 'spectrum_x_eav', 'Spectrum Number',
'int_val');
    cond.setValue('5');
    cond.setOperator('<=');
    query.add_condition(cond);
```

```
    cond = SpectrumQueryCondition('spectrum', 'sensor_id');
    cond.setValue('0');
    cond.setOperator('=');
    query.add_condition(cond);

    cond = SpectrumQueryCondition('spectrum', 'instrument_id');
    cond.setValue('1');
    cond.setOperator('=');
    query.add_condition(cond);

    ids = specchio_client.getSpectrumIdsMatchingQuery(query);
    %<<<<<<<<<<<<<<<<<<<<<<<<<<<

    spaces = specchio_client.getSpaces(ids, 1, 0, 'Acquisition Time');
    space = spaces(1);
    space = specchio_client.loadSpace(space);



    vectors = space.getVectorsAsArray();
    wvl = space.getAverageWavelengths();

    figure;

    plot(wvl, vectors);
    ylim([0 1]);
    xlabel('wvl [nm]')
    ylabel(char(space.getMeasurementUnit().getUnitName()));

end
```

6.2 Get Metadata of Spectra

This example 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.

```
function get_metadata()

    import ch.specchio.client.*;
    import ch.specchio.queries.*;

    % connect to server
    cf = SPECCHIOClientFactory.getInstance();
    descriptor_list = cf.getAllServerDescriptors();
    specchio_client = cf.createClient(descriptor_list.get(0)); % connect to first
connection description


    % Query pasted from SPECCHIO Query builder and manually edited to just set
conditions for the File Name attribute:
    % >>>>>>>>>>>>>>>>>>>>
    query = Query('spectrum');
    query.setQueryType(Query.SELECT_QUERY);
```

```
    query.addColumn('spectrum_id')

    cond = 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, 1, 0, 'Acquisition Time');
    space = spaces(1);
    space = specchio_client.loadSpace(space);


    ids = space.getSpectrumIds(); % get the ids from the space again; they are
now sorted by 'Acquisition Time'


    % get lat and lon of the Blackfern spectra

    positions =  specchio_client.getMetaparameters(ids, 'Spatial Posi-
tion', java.lang.Boolean(false));

    lat = zeros(positions.size(),1);
    lon = zeros(positions.size(),1);

    for i=0:positions.size() - 1
        lat(i+1) = positions.get(i).getPoint2D().getY();
        lon(i+1) = positions.get(i).getPoint2D().getX();
    end

   % Obsolete calls (used in earlier SPECCHIO versions)
   % latitudes = specchio_client.getMetaparameterValues(ids, 'Latitude');
   % longitudes = specchio_client.getMetaparameterValues(ids, 'Longi-
tude');



    % plot metadata
    figure
    plot(lon, lat, '*');


end
```

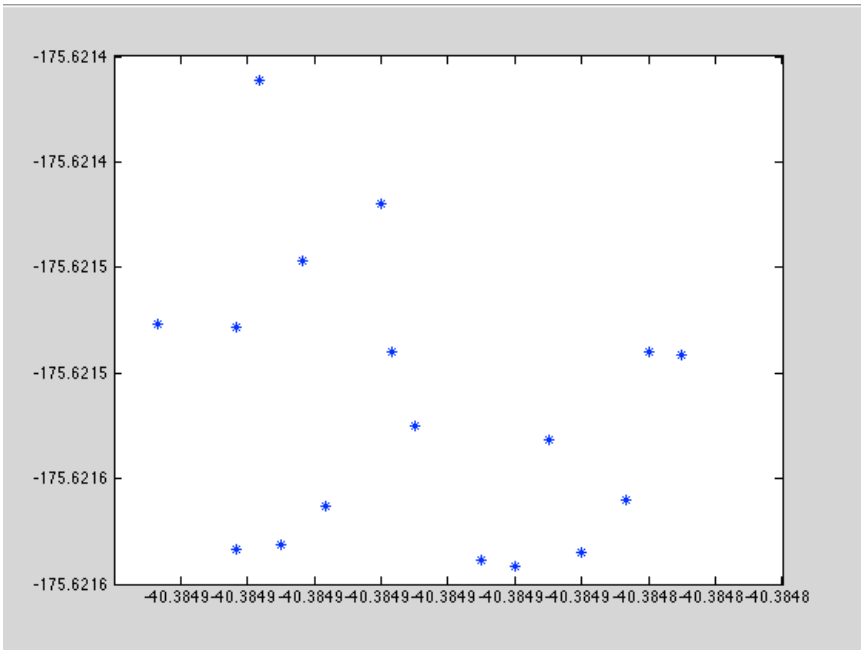


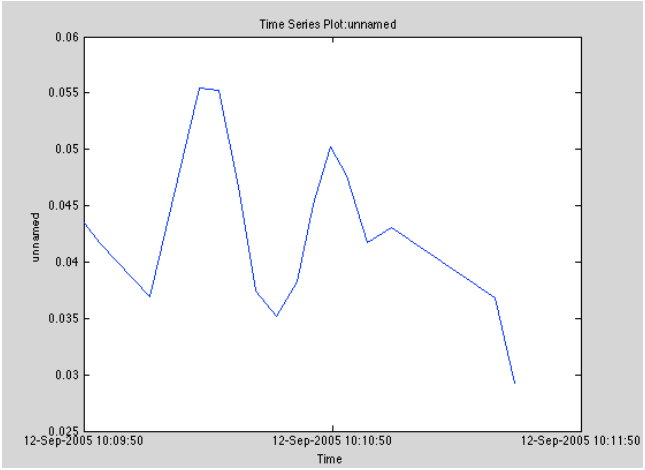
Figure 3: lat-lon plot

6.3 Generating Time Series

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

```
function timeseries_example()

    import ch.specchio.client.*;
    import ch.specchio.queries.*;

    % connect to server
    cf = SPECCHIOClientFactory.getInstance();
    descriptor_list = cf.getAllServerDescriptors();
    specchio_client = cf.createClient(descriptor_list.get(0)); % connect to first
connection description


    % Query pasted from SPECCHIO Query builder and manually edited to just set
conditions for the File Name attribute:
    % >>>>>>>>>>>>>>>>>>>>>>
    query = Query('spectrum');
    query.setQueryType(Query.SELECT_QUERY);

    query.addColumn('spectrum_id')

    cond = 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, 1, 0, 'Acquisition Time');
    space = spaces(1);
    space = specchio_client.loadSpace(space);


    ids = space.getSpectrumIds(); % get the ids from the space again; they are
now sorted by 'Acquisition Time'


    % get acquisition times of the Blackfern spectra

    acq_times = specchio_client.getMetaparameterValues(ids, 'Acquisition Time');


    % convert acquisition times to Matlab time series recognised format

    capture_times = cell(ids.size(),1);

    formatter = java.text.SimpleDateFormat('MMM.dd,yyyy hh:mm:ss');

    for i=1:ids.size()

        tmp = formatter.format(acq_times.get(i-1));
        capture_times{i} = char(tmp);
    end


    % get spectral data
    vectors = space.getVectorsAsArray();
    wvl = space.getAverageWavelengths();


    % build time series for wavelength 500nm
    wvl_ind = wvl == 500;

    band_500nm_data = vectors(:, wvl_ind);

    refl_at_500nm_time_series = timeseries(band_500nm_data, capture_times);


    % plot timeseries
    figure
    plot(refl_at_500nm_time_series);


end
```



7 Building Graphical User Interfaces

Undocumented Matlab features can be used to generate user interfaces that use Java and Matlab GUI components in a seamless fashion. This functionality makes use of the jcontrol library (Lidierth 2009).
This allows the building of interactive database access GUIs via e.g. including a Spectral Data Browser object (Figure 4)[3]:

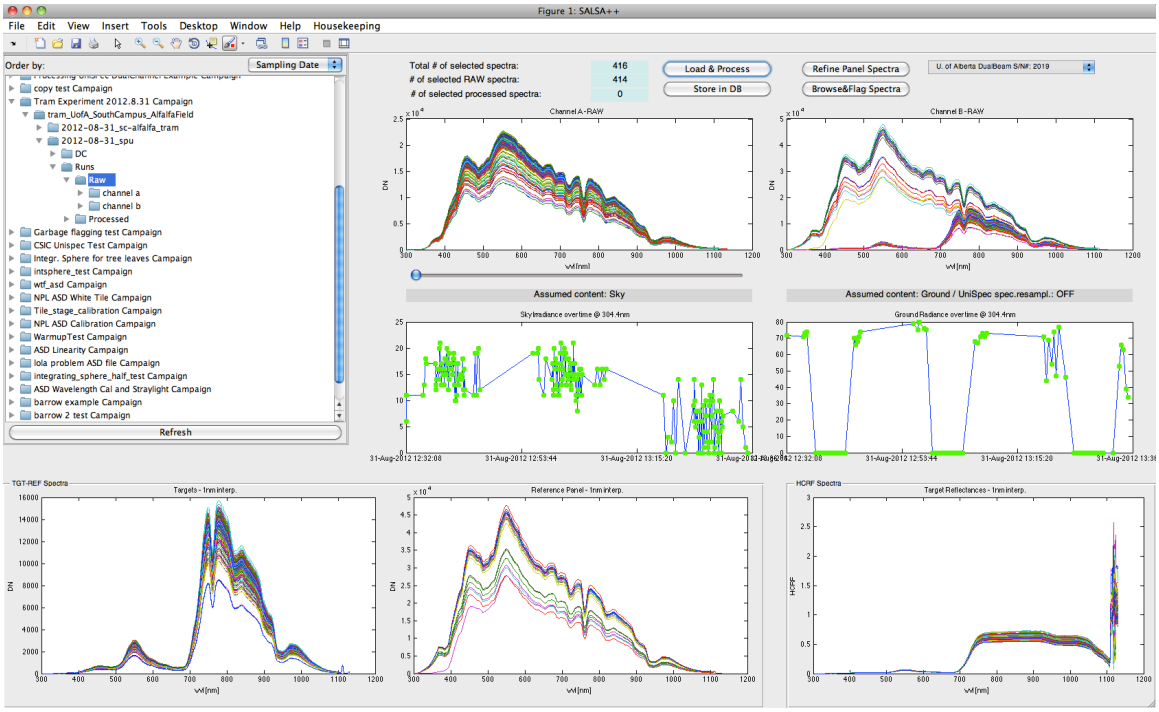


Figure 4: SALSA++ Matlab GUI with a Spectral Data Browser component on the top-left

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

If you require more information on how to build such GUIs please contact the author of this document by email: ahueni@geo.uzh.ch.

8 Troubleshooting

8.1 java.lang.NoClassDefFoundError

A typical error when trying to call Java code from Matlab is the throwing of java.lang.NoClassDefFoundError errors.
Typically, the thrown errors take the form shown below:

```
Error using ConnectSpecchio (line 17)
Java exception occurred:
java.lang.NoClassDefFoundError: javax/ws/rs/ext/MessageBodyReader

      at java.lang.ClassLoader.defineClass1(Native Method)

      at java.lang.ClassLoader.defineClass(ClassLoader.java:763)

      at ja-
va.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)

      at java.net.URLClassLoader.defineClass(URLClassLoader.java:468)
```

The reason for these errors is that Matlab cannot find the classes within the supplied specchio-client.jar file. Most often, this error appears on Windows machines, while the same version of Matlab works fine on systems like MacOS.
The solution for this appears to be the definition of the classpath: if this error is thrown, then try to use a static definition of the classpath as documented in 2.1.1.1.

9 References

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

Lidierth, M. (2009). "sigTOOL: A MATLAB-based environment for sharing laboratory-developed software to analyze biological signals." Journal of Neuroscience Methods **178**(1): 188–196.