Spectra Thumbnails

# Overview

One of the pieces of functionality that was required by the SPECCHIO application was the ability to view thumbnails for any given spectra before the user commits to opening a report for the given spectra. This data had to be inserted into an existing part of the graphical user interface within the Query Builder frame with all backend logic being dealt with in a separate class.

Topics to be covered

* Overview of the design choices and class layout
* General implementation of the thumbnails
* Implementation issues
* Error catching

# Design Decisions including class layout decisions

In the current implementation there was no way to look at a spectra in any form other than its identification number. A choice had to be made on which piece of information would be shown, there were a range of possibilities that were all text based that take from the MATLAB’s implementation of the getMetaParameters. This method had the possibility to pull back any of the information about a given spectra. Ultimately this was deemed to be only useful if the correct piece of information was picked for the user. The chosen method of interpreting information that was decided upon was to return a spectral graph. This would give the user the ability to have a graphical representation of individual spectra that gives a perfect overview of the given spectra.

The design choice was to have a separate class to handle given identification numbers in order to plot the spectral graph and return a JPanel that would display the given plot. The information would be gathered from the file browser on the left hand side of the current implementations pane and would be implemented in a thread in order to allow long processes to be undertaken without causing any hangs in the application.

A major concern throughout the project was the ability to push these changes to a live implementation of the SPECCCHIO application. This in turn had to shape the design of how the spectral thumbnails would be implemented. In order to reduce processing time when taking multiple reads from the database it was deemed necessary to only plot a single spectra at a time. On development machines where the database is stored locally for testing purposes reading multiple spectra was almost instant as no external connections had to be established, but once these changes hit live every spectra would have to be plotted from an individual read from the database and having multiple spectra being plotted at once theoretically would cause huge wait periods for the user.

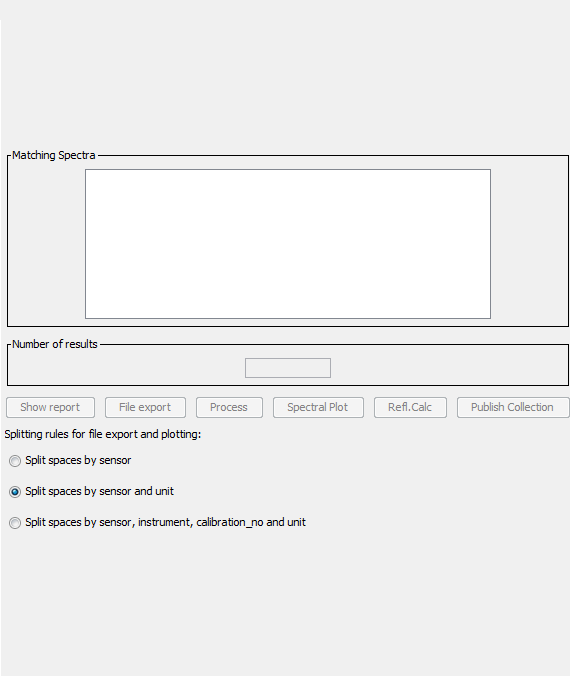


Figure 1 – old specchio query broswer mplementation

The graph had to be plotted somewhere within the current application and within the initial implementation of the query browser there was a substantial amount of space that could be used if the elements were pushed down [Figure 1], this gave the ability to add a small spectral plot to the query browser.

The new implementation used the available free space in order to create a new spectral graph which would be updated as the user clicks on the spinner. This gives the user a quick overview of the spectra within a given set of ranges and allows them to make a more informed decision on what data they wish to take from the database. [Figure 2]  
  
The live application should be free of bugs and this influenced the design decision to have a separate class for handling the data input from the query builder. This way any bugs within the code were quickly fleshed out and error caught and handled.

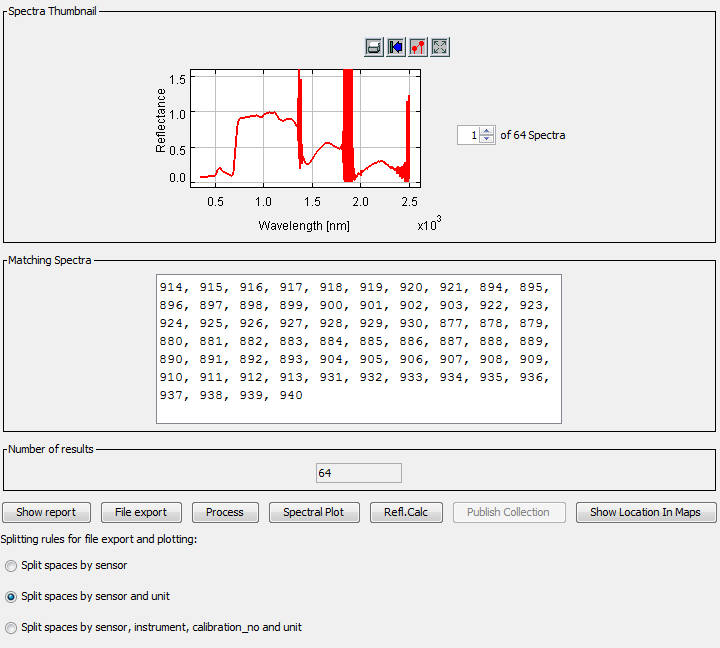


Figure 2 - new implementation of spectral thumbnail

# Implementation

The implementation began by passing in the selected spectra’s from the query builder frame into a thread that is called by a tree listener within a file browser that was already part of the programs core functionality. It was tasked that this tree listener be updated to pass the unsorted selected spectra id’s into a thread, this thread was created to handle four pass variables.

1. ArrayList of spectra id’s
2. Boolean of by sensor
3. Boolean of by sensor and unit
4. String of current order

The variables are then passed into a class that’s sole purpose is to create a spectral plot for given id’s. Within this class a method was created that checks the id’s against the database’s listings for the given id. It would then pass the information to a series of ArrayLists and Hashtables that were created by the applications creators (add reference) which sort the ids and handle the meta data for each associated id.

A separate function set displayed index is called and checks against a JSpinner with a change listener for the currently selected id and creates a new spectral line plot from MATLAB’s library with a defined width and height set for correct viewing within the GUI.

The selected spectral plot is then drawn to a new JPanel and redrawn and revalidated to ensure correct showing within the GUI. This JPanel is then passed back out to the Query Builder and updated and shown.

# 

# Implementation Issues

When creating the thumbnails a number of issues when dealing with the collection and representation of correct spectral id’s and plots had to be dealt with, along with this Java swing also had limitations with handling the panel redrawing and revalidation when adding the spectral plots as well as the spinner having a correct change listener implemented.

A list of known implementation issues are as follows.

1. Swing and layered panels
2. Swing override of panel sizes
3. Single drawing of plots as to reduce program load
4. Implementing a spinner number model to stop index out of bounds exceptions

### Swing and layered panels

When creating new instances of spectral plots they had to be cast to a panel which was being passed from the query builder. These plots were then being drawn to a panel called spectra\_thumbnail\_panel but after the first had been drawn an issue was found that any subsequent panel would be drawn behind the original panel thus causing unusable plots when multiple spectra had been selected.

A new check had to be implemented when the tree’s action listener was called which would remove any current instance of the spectra\_thumbnail\_panel and redraw and revalidate a new panel when the spinner had been given a new value either typed or clicked on. This change also relied heavily on the design decision to only draw one panel and one spectra at a time in order to reduce load on the system when a huge array of spectra had been selected.

### Swing override of panel sizes

Within the swing framework any child panel will inherit the sizes of the parent even when specifically setting a size for the child panel. This meant that when the panel had no current listing for a spectra thumbnail the panel would be a single pixel wide on initial creation and would not update. It was later found that setting a preferred size for a child node would automatically update the parent node to allow for the child’s preferred size.

### Single drawing of plots to reduce program load

The choice to only draw a single spectra at a time was decided upon after a 200 spectra test showed that rendering every drawing at once would cause small hangs in the application even with the database being run on the local development machine.

Due to this issue the spectra spinner was altered to pass a single value to the spectral plotting method. This substantially reduced the load and reduced the hang in the system to a fraction of a second. With the later addition of a thread to handle solely the spectral thumbnail plots the system has no visible slowdown.

### Implementing a spinner number model to stop index out of bounds exceptions

The JSpinner created had the ability to have any number set by the user and due to the change of the spinner directly influencing the drawing of a spectral plot this would create index out of bounds exceptions. A spinner number model was implemented that had an upper limit of the maximum amount of spectra which is determined by checking the ids ArrayList that is passed into the AddSpectralPlot class. The size of this ArrayList was then checked for its size and then added to the spinner number model. This spinner number model also doesn’t allow for input outside of the models value range which would have also caused an index out of bounds exception.

# Class Diagram

Shown below is the class diagram for the query builder to add spectral plot interactions. Listed are all the variables and method calls required for each.

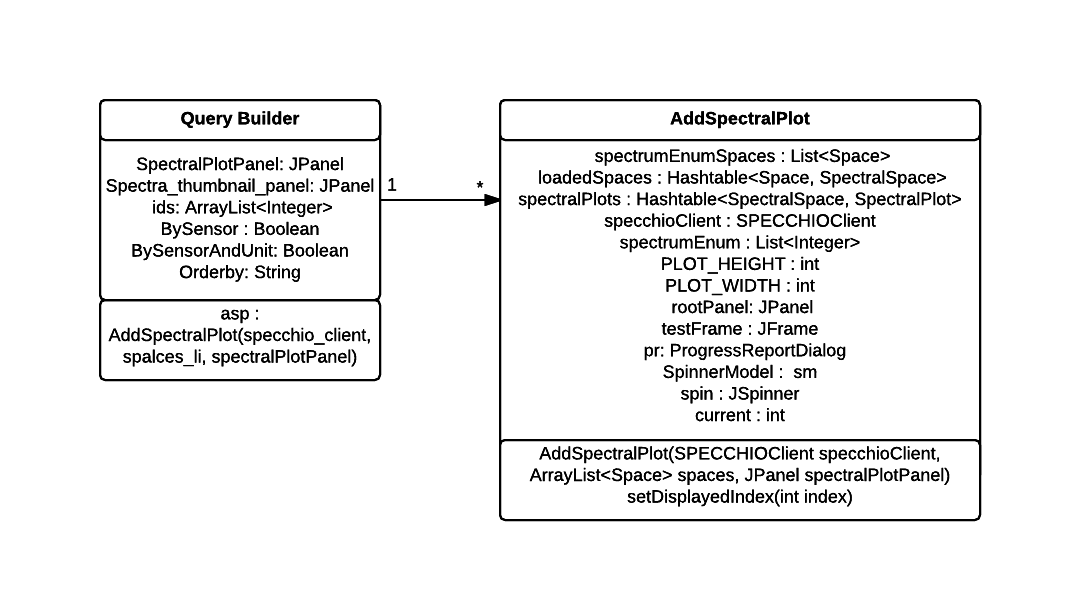


Figure 3 – Query Builder Class Diagram