Zeus

1.01

Generated by Doxygen 1.9.6

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Backend Class Reference

Class handling the backend functions of the entire programme. Public variables are expanded on within the code.

```
#include <Backend.h>
```

Public Member Functions

- int saveToFile (String^ name, bool selectedOnly, bool lowestPoint)
- int saveToFileCalibration (String^ name, bool lowestPoint)
- int loadFiles (array< String^>^ fileNames, float cutoff, int whichMode)
- int initializeSets (int length)
- int addSetToSets (float concentration, int index, float cutoff)
- int getRequestedSpectraStandardMode (int option, float range, bool doLowerRange, float lowerRange)
- int getRequestedSpectraCalibrationMode (int option, float range, bool doLowerRange, float lowerRange)
- int getAveragedSpectra ()
- int addWavelength (float wavelength)
- int removeWavelength (float wavelength)
- float getRSquared ()

Public Attributes

- String ^ **directory**
- String ^ **nameOfFile**
- List< String^> ^ **filesToExtract**
- List< Dictionary< float, float >^> ^ **listOfDictionaries**
- Dictionary< float, float > ^ **result**
- List< float > ^ **indexedKeys**
- Dictionary< float, float > ^ **presentToUserResult**
- List< float > ^ **selectedWavelengths**
- List< float > ^ **userSelectionsToKeys**
- List< int > ^ **userSelectionsIndexes**
- List< Dictionary< float, float >^> ^ **listOfResultsForFiles**
- List< float > ^ **listOfAveragedIndividualResults**
- List< String^> ^ **filesToExtract_B**

- List< Dictionary< float, float >^> ^ **listOfDictionaries_B**
- List< String^> ^ **metadata**
- List< List< Dictionary< float, float >^>^> ^ **listOfSets**
- List< float > ^ **listOfConcentrations**
- List< float > ^ **listOfAverages**
- List< List< Dictionary< float, float >^>^> ^ **listOfProcessedSets**
- List< List< float >^> ^ **listOfAveragedIndividualResults_sets**
- float **global_r2**

Private Member Functions

- int processFileIntoDictionary (Dictionary< float, float >^ dict, String^ filename, float cutoff)
- int initializeMemoryFiles (float cutoff, int whichMode)
- int sumDictionaries ()
- int findKeys ()
- int findRequestedValues (int option, float range, bool doLowerRange, float lowerRange)
- int findRequestedValuesCalibration (int option, float range, bool doLowerRange, float lowerRange)
- int numberOfValidSets ()
- System::Tuple< float, float > ^ findHighestKeyValuePair (int i, float key, int option, float rangeEachWay, bool whichDir, Dictionary< float, float >^ inputDict)
- List< float > ^ averageIndividualKeyValuePairs (List< Dictionary< float, float >^>^ LOD)
- float returnDivisionFromTwoFirst (List< float >^ givenList)

4.1.1 Detailed Description

Class handling the backend functions of the entire programme. Public variables are expanded on within the code.

The class has the following main functions:
-opening files after UI has provided this class a list of filenames.
-converting said files into a data format processable by the programme.
-storing the selected peaks, as well as any database present in memory.
-saving information to a file.

4.1.2 Member Function Documentation

4.1.2.1 addSetToSets()

```
int Backend::addSetToSets (
             float concentration,
             int index,
             float cutoff )  [inline]
```

Function adding loaded files to a set at a given index.

Parameters

concentration	User-supplied for a given set.
index	of the current set selected. Provided by which set user currently has selected in the GUI.
cutoff	Values of light intensities below which the intensity will be set to 0 when initializing data structures. Default -199.

Returns

Returns 1 on success and 0 on failure.

4.1.2.2 addWavelength()

```
int Backend::addWavelength (
             float wavelength )  [inline]
```

Function adding an user-supplied wavelength to an internal data structure. Checks if wavelength is correct as well.

Parameters

wavelength	Wavelength to be added.

Returns

Returns 1 if wavelength is valid; 0 otherwise.

4.1.2.3 averageIndividualKeyValuePairs()

```
List< float > ^ Backend::averageIndividualKeyValuePairs (
             List< Dictionary< float, float >^>^ LOD )  [inline], [private]
```

Private function that averages the values of keys in the same user-supplied keys. However, the keys don't necessarily have to be the same - this enables calculations for slightly different keys (for example, the highest keys) that had the same user-supplied key value.

Parameters

LOD	List of Dictionaries - passed by reference to find a result in a set of dictionaries.

Returns

Returns a smart pointer to a list that holds the results for a given list of dictionaries (files).

4.1.2.4 findHighestKeyValuePair()

```
System::Tuple< float, float > ^ Backend::findHighestKeyValuePair (
             int i,
             float key,
             int option,
             float rangeEachWay,
             bool whichDir,
             Dictionary< float, float >^ inputDict )  [inline], [private]
```

Private function for finding the highest key in a given set.

Parameters

i	The index of the key in the indexedKeys array.
key	The value of the key for the dictionary.
option	Option for how to process the data. 1 - find the highest peak within range, 2- sum all datapoints within range. If whichDir == false, this parameter is discarded.
rangeEachWay	The range to check, each way away from the key.
whichDir	boolean determining whether the function finds the highest or lowest value in range.
inputDict	Dictionary (i. e. file) on which the search is performed; passed by reference.

Returns

Returns a tuple of values <highestKey, highestValue> or <lowestKey, lowestValue>, depending on whichDir.

4.1.2.5 findKeys()

```
int Backend::findKeys ( )  [inline], [private]
```

Private function that finds the appropriate keys for the values that user supplies. It does so by looking for the closest key to the value supplied. Heavily amortized O(n) time complexity for this search. Fills the self-contained dictionary.

Returns

Always returns 1.

4.1.2.6 findRequestedValues()

```
int Backend::findRequestedValues (
             int option,
             float range,
             bool doLowerRange,
             float lowerRange )  [inline], [private]
```

Private function finding requested values of intensities for previously provided wavelengths in range for standard LIBS mode.

Parameters

option	Option for how to process the data. 1 - find the highest peak within range, 2- sum all datapoints within range.
range	Range in which to find the highest peak value.
doLowerRange	Does the user want to find lowest value in range?
lowerRange	If yes, what is that range?

Returns

Always returns 1.

4.1.2.7 findRequestedValuesCalibration()

```
int Backend::findRequestedValuesCalibration (
             int option,
             float range,
             bool doLowerRange,
             float lowerRange )  [inline], [private]
```

Private function finding requested values of intensities for previously provided wavelengths in range for calibration LIBS mode.

Parameters

option	Option for how to process the data. 1 - find the highest peak within range, 2- sum all datapoints within range.
range	Range in which to find the highest peak value.
doLowerRange	Does the user want to find lowest value in range?
lowerRange	If yes, what is that range?

Returns

Always returns 1.

4.1.2.8 getAveragedSpectra()

```
int Backend::getAveragedSpectra ( )  [inline]
```

Function averaging all spectra out in a set. Serves as an intermediate step between UI and a private function.

Returns

Returns 1 on success, 0 otherwise.

4.1.2.9 getRequestedSpectraCalibrationMode()

```
int Backend::getRequestedSpectraCalibrationMode (
             int option,
             float range,
             bool doLowerRange,
             float lowerRange )  [inline]
```

Function called by the UI frontend, processing loaded data in accordance to the selected wavelengths in calibration mode. Serves as an intermediate step between other functions.

Parameters

option	Option for how to process the data. 1 - find the highest peak within range, 2- sum all datapoints within range.
range	Float specifying how many datapoints are looked at. User provided values for wavelengths assumed to be in them middle of this range.
doLowerRange	- boolean specifying whether the user has requested to find lowest values in some range as well.
lowerRange	If the user requests that lowest point is found - analogous to parameter 'range'.

Returns

Returns 1 on success, 0 on failure.

4.1.2.10 getRequestedSpectraStandardMode()

```
int Backend::getRequestedSpectraStandardMode (
             int option,
             float range,
             bool doLowerRange,
             float lowerRange )  [inline]
```

Function called by the UI frontend, processing loaded data in accordance to the selected wavelengths in single mode. Serves as an intermediate step between other functions.

Parameters

option	Option for how to process the data. 1 - find the highest peak within range, 2- sum all datapoints within range.
range	Float specifying how many datapoints are looked at. User provided values for wavelengths assumed to be in them middle of this range.
doLowerRange	boolean specifying whether the user has requested to find lowest values in some range as well.
lowerRange	If the user requests that lowest point is found - analogous to parameter 'range'.

Returns

Always returns 1 - programme will fail before reaching the call of this function.

4.1.2.11 getRSquared()

```
float Backend::getRSquared ( )  [inline]
```

Function calculating R^2 value; called after files in calibration mode have been initialized.

Returns

Returns the value of R^2. ALso initializes the global value of R^2 in the prgramme.

4.1.2.12 initializeMemoryFiles()

```
int Backend::initializeMemoryFiles (
            float cutoff,
            int whichMode )  [inline], [private]
```

Private function to extract data from all selected files (as per internally initialized list of files) and load them into RAM. Goes one by one for each filename provided.

Parameters

cutoff	Value of cutoff, below which the intensity of a given datapoint will be brought down to 0. Default -199.
whichMode	Integer describing which mode the programme is operating in. 1 - standard LIBS mode, 2 - calibration LIBS mode.

Returns

Returns 1 on success and 0 on failure.

4.1.2.13 initializeSets()

```
int Backend::initializeSets (
            int length )  [inline]
```

Function initializing data structures in memory. These are intialized in order to be index-addressable in other functions.

Parameters

length	Number of sets of files that should be initialized.

Returns

Always returns 1 - function cannot fail execution.

4.1.2.14 loadFiles()

```
int Backend::loadFiles (
            array< String^>^ fileNames,
            float cutoff,
            int whichMode )  [inline]
```

Function reading files from disc. Admittedly could be simplified together with GUI as it doesn't need two file selection buttons.

Parameters

fileNames	Array of filenames. This list of files should be selected by the user in the GUI; using standard Windows libraries.
cutoff	Values of light intensities below which the intensity will be set to 0 when initializing data structures. Default -199.
whichMode	Integer describing which mode the programme is operating in. 1 - standard LIBS mode, 2 - calibration LIBS mode.

Returns

Returns an integer indicating status. 0 - loading of files failed. 1 - load successful.

4.1.2.15 numberOfValidSets()

```
int Backend::numberOfValidSets ( )  [inline], [private]
```

Function for counting how many non-null elements there are in the sets. Example: user supplies 4 values of wavelengths. However, user also claims there are 3 sets to process - but user only submitted to set 1 and 3. This code works around that.

Returns

Returns the number of valid sets.

4.1.2.16 processFileIntoDictionary()

```
int Backend::processFileIntoDictionary (
            Dictionary< float, float >^ dict,
            String^ filename,
            float cutoff )  [inline], [private]
```

Private function converting a raw file to a dictionary data structure.

Parameters

dict	Pass by reference - dictionary into which a file should be parsed.
filename	Name of file to be processed into a datastructure.
cutoff	Value of cutoff, below which the intensity of a given datapoint will be brought down to 0. Default -199.

Returns

Returns a status flag. 1 - file successfully processed. 0 - file read failed.

4.1.2.17 removeWavelength()

```
int Backend::removeWavelength (
             float wavelength )  [inline]
```

Function removing an user-supplied wavelength from an internal data structure.

Parameters

wavelength	Wavelength to be removed.

Returns

1 if removal successful, 0 otherwise. 0 should never be reached and this should be ensured in the Window.h file.

4.1.2.18 returnDivisionFromTwoFirst()

```
float Backend::returnDivisionFromTwoFirst (
             List< float >^ givenList )  [inline], [private]
```

Function that divides first two items of a list. For programmer's convenience.

Parameters

givenList	List that the division is performed on.

Returns

Returns the result of the division.

4.1.2.19 saveToFile()

```
int Backend::saveToFile (
             String^ name,
             bool selectedOnly,
             bool lowestPoint )  [inline]
```

Save a processed data structure to a file in standard LIBS mode.

Parameters

name	Name of file to save to.
selectedOnly	Save all wavelengths to a file; or only the ones selected by the user.
lowestPoint	Option describing whether the user is also interested in saving the lowest values in range to file.

Returns

Returns an integer describing success/fail of function. 1 - file saved; 0 - file locked by OS; -1 - file not saved because of user/programmer error.

4.1.2.20 saveToFileCalibration()

```
int Backend::saveToFileCalibration (
            String^ name,
            bool lowestPoint )  [inline]
```

Save a processed data structure to a file in calibration LIBS mode.

Parameters

name	Name of file to save to. Does not have to have a correct extension.
lowestPoint	Option describing whether the user is also interested in saving the lowest values in range to file.

Returns

Returns an integer describing success/fail of function. 1 - file saved; 0 - file locked by OS; -1 - file not saved because of user/programmer error.

4.1.2.21 sumDictionaries()

```
int Backend::sumDictionaries ( )  [inline], [private]
```

Private function summing all dictionaries into one resulting dictionaries, for when the strongest signal wants to be seen across n files. This function also initializes indexed keys, which in itself is crucial for O(1) operation - however, it would be good practice to have them be initialized in a separate function and decouple the code a bit.

Returns

Always returns 1, as function will be always successful if the programme execution reaches this point.

The documentation for this class was generated from the following file:

- C:/MEng/LIBSProcessing/LIBSProcessing/Backend.h

4.2 LIBSProcessing::Window Class Reference

Class handling the display of the user interface (later referred to as GUI or UI). Primarily has functions responsible for recording button presses.

```
#include <Window.h>
```

Public Member Functions

- Window (void)

Protected Member Functions

- ∼**Window** ()

  Clean up any resources being used.

Protected Attributes

- Backend **b**

Private Member Functions

- void InitializeComponent (void)

  Automatically generated code for the use of Windows Forms Designer. Should not be modified directly by the user.
- System::Void **elemSubmit_Click** (System::Object^ sender, System::EventArgs^ e)
- System::Void waveSubmit_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void saveFolderSelect_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void removeWave_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void preview_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void saveToFile_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void fileSelect_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void fileSelect_setB_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void Window_Load (System::Object^ sender, System::EventArgs^ e)
- System::Void standardToolStripMenuItem_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void calibrationToolStripMenuItem_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void howManySubmit_Click (System::Object^ sender, System::EventArgs^ e)
- System::Void addSetButton_Click (System::Object^ sender, System::EventArgs^ e)
- void handleSelection (int selectionWindow)

  Helper functions, not to clutter the main UI code - mainly with a single switch.
- void setCalibrationGroup (bool value)
- void setSetAddedGroup (bool value)

Private Attributes

- System::Windows::Forms::CheckBox ^ **saveSelectedBox**
- System::Windows::Forms::RadioButton ^ **highestCheckbox**
- System::Windows::Forms::RadioButton ^ **sumCheckbox**
- System::Windows::Forms::GroupBox ^ **groupBox1**
- System::Windows::Forms::TextBox ^ **noiseCutoff**
- System::Windows::Forms::Label ^ **label8**
- System::Windows::Forms::Label ^ **label9**
- System::Windows::Forms::Label ^ **cutoffLabel**
- System::Windows::Forms::MenuStrip ^ **menuStrip1**
- System::Windows::Forms::ToolStripMenuItem ^ **modeToolStripMenuItem**
- System::Windows::Forms::ToolStripMenuItem ^ **standardToolStripMenuItem**
- System::Windows::Forms::ToolStripMenuItem ^ **calibrationToolStripMenuItem**
- System::Windows::Forms::Label ^ **setAlabel**

- System::Windows::Forms::Label ^ **setBlabel**
- System::Windows::Forms::Label ^ **selectFilesLabel_setB**
- System::Windows::Forms::Button ^ **fileSelect_setB**
- System::Windows::Forms::Label ^ **analyteLabel_setB**
- System::Windows::Forms::TextBox ^ **analyteBox_setB**
- System::ComponentModel::BackgroundWorker ^ **backgroundWorker1**
- System::Windows::Forms::Label ^ **setNumbersLabel**
- System::Windows::Forms::Label ^ **howManyLabel**
- System::Windows::Forms::Button ^ **howManySubmit**
- System::Windows::Forms::TextBox ^ **howManySets**
- System::Windows::Forms::ComboBox ^ **setsOfData**
- System::Windows::Forms::Label ^ **setsOfData_label**
- System::Windows::Forms::Button ^ **addSetButton**
- System::Windows::Forms::Label ^ **Rscore**
- System::Windows::Forms::TextBox ^ **rangeLowerInput**
- System::Windows::Forms::Label ^ **label1**
- System::Windows::Forms::GroupBox ^ **groupBox2**
- System::Windows::Forms::RadioButton ^ **lowerRangeYes**
- System::Windows::Forms::RadioButton ^ **radioButton2**
- System::Windows::Forms::TextBox ^ **waveEdit**
- System::Windows::Forms::Button ^ **waveSubmit**
- System::Windows::Forms::Label ^ **label2**
- System::Windows::Forms::TextBox ^ **rangeInput**
- System::Windows::Forms::Label ^ **label3**
- System::Windows::Forms::ToolTip ^ **toolTip1**
- System::Windows::Forms::ComboBox ^ **allWavelenghts**
- System::Windows::Forms::Label ^ **label4**
- System::Windows::Forms::Button ^ **removeWave**
- System::Windows::Forms::Button ^ **saveFolderSelect**
- System::Windows::Forms::FolderBrowserDialog ^ **folderBrowser**
- System::Windows::Forms::Label ^ **label5**
- System::Windows::Forms::TextBox ^ **savePath**
- System::Windows::Forms::TextBox ^ **nameOfFile**
- System::Windows::Forms::Label ^ **label6**
- System::Windows::Forms::Button ^ **saveToFile**
- System::Windows::Forms::Button ^ **fileSelect**
- System::Windows::Forms::Label ^ **label7**
- System::Windows::Forms::Button ^ **preview**
- System::Windows::Forms::Label ^ **noOfFiles**
- System::Windows::Forms::OpenFileDialog ^ **fileOpener**
- System::ComponentModel::IContainer ^ **components**

4.2.1 Detailed Description

Class handling the display of the user interface (later referred to as GUI or UI). Primarily has functions responsible for recording button presses.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Window()

```
LIBSProcessing::Window::Window (
             void  )  [inline]
```

Initialization of the GUI.

4.2.3 Member Function Documentation

4.2.3.1 addSetButton_Click()

```
System::Void LIBSProcessing::Window::addSetButton_Click (
             System::Object^ sender,
             System::EventArgs^ e )  [inline], [private]
```

GUI handler - set has been added, initialize it in the backend.

Parameters

sender	NA
e	NA

Returns

4.2.3.2 calibrationToolStripMenuItem_Click()

```
System::Void LIBSProcessing::Window::calibrationToolStripMenuItem_Click (
             System::Object^ sender,
             System::EventArgs^ e )  [inline], [private]
```

GUI handler - select calibration mode in toolbar.

Parameters

sender	
e	

Returns

4.2.3.3 fileSelect_Click()

```
System::Void LIBSProcessing::Window::fileSelect_Click (
             System::Object^ sender,
             System::EventArgs^ e )  [inline], [private]
```

GUI handler - select files.

Parameters

sender	NA
e	NA

Returns

Void.

4.2.3.4 fileSelect_setB_Click()

```
System::Void LIBSProcessing::Window::fileSelect_setB_Click (
             System::Object^ sender,
             System::EventArgs^ e )  [inline], [private]
```

GUI handler - select files - calibration mode.

Parameters

sender	
e	

Returns

Void.

4.2.3.5 handleSelection()

```
void LIBSProcessing::Window::handleSelection (
             int selectionWindow )  [inline], [private]
```

Helper functions, not to clutter the main UI code - mainly with a single switch.

GUI helper function - verify whether files selected by the user are valid & check the cutoff value.

Parameters

selectionWindow	

4.2.3.6 howManySubmit_Click()

```
System::Void LIBSProcessing::Window::howManySubmit_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - see how many sets the user is trying to initialize & then do so.

Parameters

sender	NA
e	NA

Returns

Void.

4.2.3.7 InitializeComponent()

```
void LIBSProcessing::Window::InitializeComponent (
            void )  [inline], [private]
```

Automatically generated code for the use of Windows Forms Designer. Should not be modified directly by the user.

Required method for Designer support - do not modify the contents of this method with the code editor.
Functions will be calling the backend internally.

4.2.3.8 preview_Click()

```
System::Void LIBSProcessing::Window::preview_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - process the submitted data.

Parameters

sender	NA
e	NA

Returns

Void.

4.2.3.9 removeWave_Click()

```
System::Void LIBSProcessing::Window::removeWave_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - remove wavelength from analysis.

Parameters

sender	N/A
e	N/A

Returns

Void.

4.2.3.10 saveFolderSelect_Click()

```
System::Void LIBSProcessing::Window::saveFolderSelect_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - save the folder path for user results..

Parameters

sender	N/A
e	N/A

Returns

Void.

4.2.3.11 saveToFile_Click()

```
System::Void LIBSProcessing::Window::saveToFile_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - save to a file.

Parameters

sender	NA
e	NA

Returns

Void.

4.2.3.12 setCalibrationGroup()

```
void LIBSProcessing::Window::setCalibrationGroup (
            bool value )  [inline], [private]
```

GUI helper function - set buttons to Calibration mode.

Parameters

value	true if enable calibration group, false otherwise.

4.2.3.13 setSetAddedGroup()

```
void LIBSProcessing::Window::setSetAddedGroup (
            bool value )  [inline], [private]
```

GUI helper function - set buttons to active/inactive after number of sets have been added.

Parameters

value	True if enable buttons in set, false if disable.

4.2.3.14 standardToolStripMenuItem_Click()

```
System::Void LIBSProcessing::Window::standardToolStripMenuItem_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - select standard mode in toolbar.

Parameters

sender	NA
e	NA

Returns

Void.

4.2.3.15 waveSubmit_Click()

```
System::Void LIBSProcessing::Window::waveSubmit_Click (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - submitting custom wavelengths to the list.

Parameters

sender	N/A
e	N/A

Returns

Void.

4.2.3.16 Window_Load()

```
System::Void LIBSProcessing::Window::Window_Load (
            System::Object^ sender,
            System::EventArgs^ e )  [inline], [private]
```

GUI handler - load the window. Currently empty.

Parameters

sender	NA
e	NA

Returns

Void.

The documentation for this class was generated from the following file:

- C:/MEng/LIBSProcessing/LIBSProcessing/Window.h

Chapter 5

File Documentation

5.1 C:/MEng/LIBSProcessing/LIBSProcessing/Backend.h File Reference

File handling all the backend functions, invisible to the user.

Classes

- class Backend

  Class handling the backend functions of the entire programme. Public variables are expanded on within the code.

Macros

- #define **DATASIZE** 26607
- #define **LINESTOSKIP** 105

5.1.1 Detailed Description

File handling all the backend functions, invisible to the user.

Author

PR

Date

April 2023

5.2 Backend.h

Go to the documentation of this file.

```
/********************************************************************/
#pragma once


using namespace System;
using namespace System::IO;
using namespace System::Windows::Forms;
using namespace System::Collections::Generic;


//how many datapoints are there?
#define DATASIZE 26607
#define LINESTOSKIP 105


public ref class Backend {



public:
    String^ directory;
    String^ nameOfFile;
    List<String^>^ filesToExtract;
    List<Dictionary<float, float>^>^ listOfDictionaries;        //dictionary has O(1) access time when passed the key. Holds all files.
    Dictionary<float, float>^ result;                           //summed dictionary of provided files.
    List<float>^ indexedKeys;                                   //Indexing the keys in the dictionary for easy sequential access.
    Dictionary<float, float>^ presentToUserResult;              //Final-final result that is saved to the computer.
    List<float>^ selectedWavelengths;                           //A list of floats that the user selected for processing.
    List<float>^ userSelectionsToKeys;                          //A list which holds the user selections, but as keys that can be used with dictionaries.
    List<int>^ userSelectionsIndexes;                           //As dictionaries are unsorted, we need to keep track of the indexes as well.

    List<Dictionary<float, float>^>^ listOfResultsForFiles;     //List holding results for selected wavelengths for each file.
    List<float>^ listOfAveragedIndividualResults;               //List holding the individual results.

    //Calibration mode variables.
    List<String^>^ filesToExtract_B;                            //Temporarily selected files & temporarily initialized dictionary when user is selecting.
    List<Dictionary<float, float>^>^ listOfDictionaries_B;

    //multi-set processing.
    List<String^>^ metadata;                                    //Metadata about a set to be displayed about a given set.
    List< List<Dictionary<float, float>^>^>^ listOfSets;        //Processed files as sets inside of a list.
    List<float>^ listOfConcentrations;                          //Concentrations as supplied by the user.
    List<float>^ listOfAverages;                                //result of average division of first wavelength/second wavelength for each set of files.

    List< List<Dictionary<float, float>^>^>^ listOfProcessedSets;   //List holding results for selected wavlengths for each file, for each set.
    List<List<float>^>^ listOfAveragedIndividualResults_sets;       //List holding individual results for each file for each set - average from division.

    float global_r2;
    Backend() {
        directory = Application::StartupPath;
        //for now.
        nameOfFile = "default.csv";
        //initialize the (now empty) dynamically allocated array.
        selectedWavelengths = gcnew List<float>();
    }
    int saveToFile(String^ name, bool selectedOnly, bool lowestPoint) {
        //see if user put in any input; if not, do a default
        if (name) {
            nameOfFile = "\\"+name;
        }
        else {
            nameOfFile = "\\"+System::DateTime::Now.ToString("dd_MM_hhmm");
        }
        if (!name->EndsWith(".csv")) {
            nameOfFile = nameOfFile + ".csv";
        }

        Dictionary<float, float>^ whatToSave = selectedOnly ? presentToUserResult : result;
        StreamWriter^ sw;
        try {
            sw = gcnew StreamWriter(directory + nameOfFile);
        }
        catch (...) {
            return 0;
        }
        //loop through all the keyss
        try {
            sw->Write("AVERAGE OF SUM OF FILES,\n");
            sw->Write("Highest key in range, result\n");
            for each (float key in whatToSave->Keys) {
                sw->Write(Convert::ToString(key));
                sw->Write(',');
                sw->Write(Convert::ToString(result[key]));
                sw->Write('\n');
            }
            sw->Write("AVERAGE OF INDIVIDUAL HIGHEST WITHIN RANGE\n");
            int i = 0, j = 0;
            sw->Write("Supp. wavelength,Result\n");
            for each (float result in listOfAveragedIndividualResults) {
                if (i % 2 != 0 && lowestPoint) {
                    i++;
                    continue;
                }

sw->Write(Convert::ToString(userSelectionsToKeys[j])+","+Convert::ToString(result)+",");
                i++;
                j++;
            }
            sw->Write("\n");
            sw->Write("Division of averaged first intensity over the
other:„„,"+Convert::ToString(returnDivisionFromTwoFirst(listOfAveragedIndividualResults))+"\n\n");
            sw->Write("Individual results\n");
            if (lowestPoint) {
                sw->Write("„„(optional),(optional)\n");
            }

            if (lowestPoint) {
                sw->Write("File no., wavelength, intensity,Lowest key in range, result,peak?\n");
            }
            else {
                sw->Write("File no., wavelength,intensity,\n");
            }

            i = 1;
            for each (Dictionary<float, float> ^ fileAsDictionary in listOfResultsForFiles) {
                sw->Write("File " + (Convert::ToString(i)) + ",");
                //this is quite bad, as no calculations should be done inside this function. However,
I am too tired to do it otherwise right now.
                j = 0;
                float tempHighest, tempLowest;

                for each (float key in fileAsDictionary->Keys) {
                    if (lowestPoint) {
                        if (j %2 == 0) {
                            tempHighest = fileAsDictionary[key];
                        }
                        if (j % 2 != 0) {
                            tempLowest = fileAsDictionary[key];
                        }
                    }
                    sw->Write(Convert::ToString(key) + "," +
Convert::ToString(fileAsDictionary[key])+",");
                    j++;
                    if (j % 2 == 0 && j > 0 && lowestPoint) {
                        if (tempHighest > 3 * tempLowest) {
                            sw->Write("yes„");
                        }
                        else {
                            sw->Write("no„");
                        }
                    }
                }
                sw->Write("\n");
                i++;
            }
        }
        catch (...) {
            //data structure not initialized
            sw->Close();
            return -1;
        }
        sw->Close();
        return 1;
    }
    int saveToFileCalibration(String^ name, bool lowestPoint) {
        if (name) {
            nameOfFile = "\\" + name;
        }
        else {
            nameOfFile = "\\" + System::DateTime::Now.ToString("dd_MM_hhmm");
        }
        if (!name->EndsWith(".csv")) {
            nameOfFile = nameOfFile + ".csv";
        }
        StreamWriter^ sw;
        try {
            sw = gcnew StreamWriter(directory + nameOfFile);
        }
        catch (...) {
            return 0;
        }
        //write the header of the excel file.
        try {
            int elems = numberOfValidSets(); int whichKey = 0;
            bool differentDivisors = false;
            if (userSelectionsToKeys->Count / 2 == elems) {
                differentDivisors = true;
            }
            //write headers
            sw->Write(",Average (sum first intensity then sum second intensity then divide one by the other),");
            sw->Write("dividend,");
            sw->Write("divisor,");
            sw->Write("Concentration\n");
            for (int i = 0; i < metadata->Count; i++) {
                //coded for "no set input"
                sw->Write("S" + Convert::ToString(i + 1) + ",");
                if (listOfAverages[i] == -1) {
                    sw->Write("„,");
                    continue;
                }
                sw->Write(Convert::ToString(listOfAverages[i]) + ",");
                if (differentDivisors) {
                    sw->Write(Convert::ToString(userSelectionsToKeys[whichKey]) + ",");
                    sw->Write(Convert::ToString(userSelectionsToKeys[whichKey + 1]) + ",");
                    whichKey += 2;
                }
                else {
                    sw->Write(Convert::ToString(userSelectionsToKeys[0]) + ",");
                    sw->Write(Convert::ToString(userSelectionsToKeys[1]) + ",");
                }
                sw->Write(Convert::ToString(listOfConcentrations[i]) + "\n");
            }
            sw->Write("R2 score," + Convert::ToString(global_r2) + "\n");
            sw->Write("Individual results\n");
            if (lowestPoint) {
                sw->Write(",HIGHEST->„„LOWEST->„,\n");
            }
            sw->Write("File no., divid. wavelength, intensity, divisor wavelength, intensity,");
            if (lowestPoint) {
                sw->Write("„„,1.peak?,2.peak?,");
            }
            for (int i = 0; i < metadata->Count; i++) {
                sw->Write("\nS" + Convert::ToString(i + 1) + "\n");
                if (listOfAverages[i] == -1) {
                    sw->Write("„,");
                    continue;
                }
                int j = 1;
                for each (Dictionary<float, float> ^ fileAsDictionary in listOfProcessedSets[i]) {
                    sw->Write("File " + (Convert::ToString(j)) + ",");
                    int k = 0;
                    float firstHighest, firstLowest, secondHighest, secondLowest;
                    for each (float key in fileAsDictionary->Keys) {
                        if (k % 4 == 0) {
                            firstHighest = fileAsDictionary[key];
                        }
                        else if (k % 4 == 1) {
                            secondHighest = fileAsDictionary[key];
                        }
                        else if (k % 4 == 2) {
                            firstLowest = fileAsDictionary[key];
                        }
                        else if (k % 4 == 3) {
                            secondLowest = fileAsDictionary[key];
                        }
                        sw->Write(Convert::ToString(key) + "," + Convert::ToString(fileAsDictionary[key]) + ",");
                        k++;
                        if (k % 4 == 0 && k > 0 && lowestPoint) {
                            if (firstHighest > 3 * firstLowest) {
                                sw->Write(",yes,");
                            }
                            else {
                                sw->Write(",no,");
                            }
                            if (secondHighest > 3 * secondLowest) {
                                sw->Write("yes,");
                            }
                            else {
                                sw->Write("no,");
                            }
                        }
                    }
                    j++;
                    sw->Write("\n");
                }
            }
        }
        catch (...) {
            //Data structures not initialized yet.
            sw->Close();
            return -1;
        }
        sw->Close();


        return 1;
    }
    int loadFiles(array<String^>^ fileNames, float cutoff, int whichMode) {
        List<String^>^ files;
        if (whichMode == 1) {
            //reinitialize the arrays each time.
            filesToExtract = gcnew List<String^>();
            //point to the relevant array.
            files = filesToExtract;
        }
        else if(whichMode == 2){
            filesToExtract_B = gcnew List<String^>();
            files = filesToExtract_B;
        }
        else {
            //this should never be reached.
            files = gcnew List<String^>();
            return 0;
        }
        for each (String^ filename in fileNames) {
            if (filename->EndsWith(".asc")) {
                files->Add(filename);
            }
            else {
                return 0;
            }
        }
        return initializeMemoryFiles(cutoff, whichMode);

    }
    int initializeSets(int length) {
        metadata = gcnew List <String^>(length);
        listOfSets = gcnew List< List<Dictionary<float, float>^>^>(length);
        listOfConcentrations = gcnew List<float>(length);
        listOfAverages = gcnew List<float>(length);

        for (int i = 0; i < length; i++) {
            metadata->Add(Convert::ToString(i+1)+ ". THIS SET IS EMPTY.");
            //I'd like these structures to be index-addressable after they have been initialized.
            listOfSets->Add(nullptr);
            listOfConcentrations->Add(-1);
            listOfAverages->Add(-1);

        }

        return 1;

    }
    int addSetToSets(float concentration, int index, float cutoff) {
        try {
            listOfConcentrations[index] = concentration;
            listOfSets[index] = gcnew List<Dictionary<float, float>^>();
            listOfSets[index]->AddRange(listOfDictionaries_B);
            metadata[index] = Convert::ToString(index + 1) + ".SET: " + listOfDictionaries_B->Count +
" FILES, CUTOFF " +
                cutoff + ", CONT." + concentration;
            return 1;
        }
        catch (...) {
            return 0;
        }
    }


    int getRequestedSpectraStandardMode(int option, float range, bool doLowerRange, float lowerRange)
{
        //first, keys CLOSEST to the value that the user input must be found.
        findKeys();
        findRequestedValues(option, range, doLowerRange, lowerRange);
        return 1;
    }
    int getRequestedSpectraCalibrationMode(int option, float range, bool doLowerRange, float
lowerRange) {
        //if we have less than 1 item, discard. We will check for whether each set has it’s own
wavelengths later.
        if (selectedWavelengths->Count < 1) {
            return 0;
        }
        //first, keys CLOSEST to the value that the user input must be found.
        findKeys();
        findRequestedValuesCalibration(option, range, doLowerRange, lowerRange);
        return 1;
    }
    int getAveragedSpectra() {
        return sumDictionaries();
    }
    int addWavelength(float wavelength) {
        if (wavelength < 200.93 || wavelength > 1031.86) {
            return 0;
        }
        selectedWavelengths->Add(wavelength);
        return 1;
    }
    int removeWavelength(float wavelength) {
        if (!selectedWavelengths->Contains(wavelength)) {
            return 0;
        }
        selectedWavelengths->Remove(wavelength);
        return 1;
    }

    float getRSquared() {
        //calculate average intensity & average concentration; all concentrations.
        float runningSumInt = 0, runningSumCon = 0;
        int goodEntries = 0;

        for (int i = 0; i < listOfConcentrations->Count; i++) {
            if (listOfConcentrations[i] == -1) {
                continue;
            }
            goodEntries++;
            runningSumInt += listOfAverages[i];
            runningSumCon += listOfConcentrations[i];
        }
        float averageConcentration = runningSumCon / goodEntries;
        float averageIntensity = runningSumInt / goodEntries;
        //now, we have everything we need. Plug it into the R score formula.
        //1. Dividend - sum(Ii-Iaavg)(Ci-Cavg)
        float div = 0;
        for (int i = 0; i < listOfConcentrations->Count; i++) {
            if (listOfConcentrations[i] == -1) {
                continue;
            }
            div += (listOfConcentrations[i] - averageConcentration)*(listOfAverages[i] -
averageConcentration);
        }
        //2. Divisor - sqrt(sum(Ii-Iavg)^2)*sqrt(sum(Ci-Cavg)^2)
        float divisor;
        float sumI = 0, sumC = 0;
        for (int i = 0; i < listOfConcentrations->Count; i++) {
            if (listOfConcentrations[i] == -1) {
                continue;
            }
            sumC += (listOfConcentrations[i] - averageConcentration)*(listOfConcentrations[i] -
averageConcentration);
            sumI += (listOfAverages[i] - averageIntensity) * (listOfAverages[i] - averageIntensity);
        }
        sumC = Convert::ToSingle(Math::Sqrt(sumC));
        sumI = Convert::ToSingle(Math::Sqrt(sumI));
        divisor = sumC * sumI;
        //3. Finally, caluclate the R score.
        float R = div / divisor;
        global_r2 = R * R;
        return R * R;

    }



```

```

private:
    int processFileIntoDictionary(Dictionary<float, float>^ dict, String^ filename, float cutoff) {
        //file might be opened by another process
        StreamReader^ sr;
        try {
            sr = gcnew StreamReader(filename);
        }
        catch (...) {
            //0 - file used by another process
            return 0;
        }


        String^ line;
        array<String^>^ thisKeyValArray;
        int i = 0;

        //read file line by line.
        while (line = sr->ReadLine()){
            if (i < LINESTOSKIP) {
                i++;
                continue;
            }
            //split line into two strings
            thisKeyValArray = line->Split(',', 2);
            //and add to the dictionary.
            if (Convert::ToSingle(thisKeyValArray[1]) < cutoff) {
                dict->Add(Convert::ToSingle(thisKeyValArray[0]), 0);
            }
            else {
                dict->Add(Convert::ToSingle(thisKeyValArray[0]),
Convert::ToSingle(thisKeyValArray[1]));
            }

        }
        return 1;
    }
    int initializeMemoryFiles(float cutoff, int whichMode) {
        List<String^>^ files;
        int lengthOfListOfMaps;
        List<Dictionary<float, float>^>^ currentListOfDictionaries;
        if (whichMode == 1) {
            lengthOfListOfMaps = filesToExtract->Count;
            listOfDictionaries = gcnew List<Dictionary<float, float>^>(lengthOfListOfMaps);
            //point to the relevant array.
            currentListOfDictionaries = listOfDictionaries;
            files = filesToExtract;

        }
        else if (whichMode == 2) {
            lengthOfListOfMaps = filesToExtract_B->Count;
            listOfDictionaries_B = gcnew List<Dictionary<float, float>^>(lengthOfListOfMaps);
            currentListOfDictionaries = listOfDictionaries_B;
            files = filesToExtract_B;

        }
        else {
            //this should never be reached.
            currentListOfDictionaries = gcnew List<Dictionary<float, float>^>();
            return 0;
        }
        //create and populate a dictionary of data for each of the files.
        for (int i = 0; i < lengthOfListOfMaps; i++) {
            currentListOfDictionaries->Add(gcnew Dictionary<float, float>(DATASIZE - LINESTOSKIP));
            processFileIntoDictionary(currentListOfDictionaries[i], files[i], cutoff);
        }
        return 1;
    }

    int sumDictionaries() {
        //Even if there was a dictionary before, garbage collect it and create a new one.
        result = gcnew Dictionary<float, float>(DATASIZE - LINESTOSKIP);
        indexedKeys = gcnew List<float>(DATASIZE - LINESTOSKIP);
        if (listOfDictionaries == nullptr) {
            //in case user is operating in mode B.
            listOfDictionaries = listOfDictionaries_B;
        }

        for each (float key in listOfDictionaries[0]->Keys) {
            float tempValue = 0;
            Dictionary<float, float>^ temp;
            for (int j = 0; j < listOfDictionaries->Count; j++) {
                //it's upset when I'm doing double dereferencing. It's a good thing temp is a pointer
so there is not a lot
                //of overhead; still the reassignments are taking up processor cycles
                temp = listOfDictionaries[j];
                tempValue += temp[key];
            }
            //average of n spectra.
            result->Add(key, tempValue / listOfDictionaries->Count);
            indexedKeys->Add(key);
        }
        return 1;


    }

    int findKeys() {
        //if (userSelectionsToKeys == nullptr) {
        //every time we process again; we reset the list - easier to handle
        userSelectionsToKeys = gcnew List<float>();
        userSelectionsIndexes = gcnew List<int>();
        //}

        for each (float wavelength in selectedWavelengths) {
            //guesstimate the index based the distances of the datapoints.
            //A line of best fit that maps index to wavelength is y = 201.96*e^(6*10^-5*x), as per
Excel.
            //I have modified it slightly after manually inspecting the mappings, and the formula for
loosely recovering
            //the index is presented below.
            int assumedIndex = Convert::ToInt32(Math::Floor(Math::Pow(10, 5) / 6.21 *
Math::Log(wavelength / 200.9381)));
            //now, see and compare. This is heavily amortized computationally, and will not go over
more than 200 iterations for each point.
            //1.Rarest scenario. We have estimated exactly the correct key
            float tempDiff, newDiff;
            if (indexedKeys[assumedIndex] == wavelength) {
                userSelectionsToKeys->Add(wavelength);
                userSelectionsIndexes->Add(assumedIndex);
                continue;
            }
            //2. We have estmimated the key to be too large. Try a lower key one by one.
            else if (indexedKeys[assumedIndex] >= wavelength) {
                while (indexedKeys[assumedIndex] >= wavelength) {
                    tempDiff = Math::Abs(indexedKeys[assumedIndex] - wavelength);
                    assumedIndex -= 1;
                    newDiff = Math::Abs(indexedKeys[assumedIndex] - wavelength);
                }
                //now, we are pretty much spot on. Just check whether to select assumedIndex or
assumedIndex+1.
                if (tempDiff > newDiff) {
                    userSelectionsToKeys->Add(indexedKeys[assumedIndex]);
                    userSelectionsIndexes->Add(assumedIndex);
                }
                else {
                    userSelectionsToKeys->Add(indexedKeys[assumedIndex + 1]);
                    userSelectionsIndexes->Add(assumedIndex+1);
                }
                continue;
            }
            //3. We have estimated the key to be too small. Try a higher key.
            else {
                while (indexedKeys[assumedIndex] <= wavelength) {
                    tempDiff = Math::Abs(indexedKeys[assumedIndex] - wavelength);
                    assumedIndex += 1;
                    newDiff = Math::Abs(indexedKeys[assumedIndex] - wavelength);
                }
                if (tempDiff > newDiff) {
                    userSelectionsToKeys->Add(indexedKeys[assumedIndex]);
                    userSelectionsIndexes->Add(assumedIndex);
                }
                else {
                    userSelectionsToKeys->Add(indexedKeys[assumedIndex -1]);
                    userSelectionsIndexes->Add(assumedIndex-1);
                }
                continue;
            }

        }
        //success.
        return 1;
    }
    int findRequestedValues(int option, float range, bool doLowerRange, float lowerRange) {
        //reset the "presentToUser" dictionary.
        presentToUserResult = gcnew Dictionary<float, float>();
        float rangeEachWay = range / 2;
        //i must be kept track of - it indicates which key we're currently looking at
        int i = 0;
        for each (float key in userSelectionsToKeys) {
            //first, find it for the averaged dictionaries
            Tuple<float, float>^ retVal = findHighestKeyValuePair(i, key, option, rangeEachWay, true,
result);
```

```
            //now we have found the highest value. Add to the proper result dictionary.
            presentToUserResult->Add(retVal->Item1, retVal->Item2);
            i++;
        }

        //now, find the results for individual files
        int j = 0;      //keeping track of which file we're in right now
        listOfResultsForFiles = gcnew List<Dictionary<float, float>^>(listOfDictionaries->Count);
        for each (Dictionary<float, float> ^ fileAsDictionary in listOfDictionaries) {
            //TODO - have a size predetermined for this dictionary
            listOfResultsForFiles->Add(gcnew Dictionary<float, float>());
            i = 0;
            for each (float key in userSelectionsToKeys) {
                Tuple<float, float>^ retVal = findHighestKeyValuePair(i, key, option, rangeEachWay, true, fileAsDictionary);
                Tuple<float, float>^ retValLower;
                if (doLowerRange) {
                    retValLower = findHighestKeyValuePair(i, key, option, lowerRange/2, false, fileAsDictionary);
                }
                listOfResultsForFiles[j]->Add(retVal->Item1, retVal->Item2);
                if (doLowerRange) {
                    listOfResultsForFiles[j]->Add(retValLower->Item1, retValLower->Item2);
                }
                i++;
            }
            j++;
        }
        //finally, average out the individual elements
        listOfAveragedIndividualResults = averageIndividualKeyValuePairs(listOfResultsForFiles);
        return 1;
    }
    int findRequestedValuesCalibration(int option, float range, bool doLowerRange, float lowerRange) {
        float rangeEachWay = range / 2;
        //first, we must measure what kind of values have been selected in the "Currently selected wavelengths" box...
        bool differentDivisors = false;
        int elems = numberOfValidSets();
        //check if we have as many selected pairs of wavelengths as sets
        if (userSelectionsToKeys->Count / 2 == elems) {
            differentDivisors = true;
        }

        int whichKey = 0, curr = 0;
        float divisor, dividend;
        bool singleMode = false;

        //case of no division - code still runs, but it divides by 1 (i. e. returns intensity)
        if (userSelectionsToKeys->Count == 1) {
            dividend = userSelectionsToKeys[0];
            divisor = 1;
            //makeshift solution - it's quite bad practice since this key does not exist;
            //but it works out for saving a file
            userSelectionsToKeys->Add(1);
            userSelectionsIndexes->Add(-1);
            singleMode = true;
        }

        else if (!differentDivisors) {
            dividend = userSelectionsToKeys[0];
            divisor = userSelectionsToKeys[1];
        }
        //...We have now measured what kind of values have been selected in the "Currently selected wavelengths" box.
        //outer loop - run through all the sets of files.
        listOfProcessedSets = gcnew List<List<Dictionary<float, float>^>^>();
        int i, j = 0;
        for each (List<Dictionary<float, float>^> ^ set in listOfSets) {
            j = 0;
            listOfProcessedSets->Add(gcnew List <Dictionary<float, float>^>());
            //first, handle the first dictionary.
            if (set == nullptr) {
                i++;
                continue;

            }
            if (differentDivisors) {
                dividend = userSelectionsToKeys[whichKey];
                divisor = userSelectionsToKeys[whichKey + 1];
            }
            for each (Dictionary<float, float> ^ fileAsDictionary in set) {
                listOfProcessedSets[i]->Add(gcnew Dictionary<float, float> ());

                //We now definitely have the correct key for dividend and divisor - now, find the highest key/value pair within range
                Tuple<float, float>^ retValDividend = findHighestKeyValuePair(whichKey, dividend, option, rangeEachWay, true, fileAsDictionary);
                Tuple<float, float>^ retValDividendLower;
                Tuple<float, float>^ retValDividerLower;
                if (doLowerRange) {
                    retValDividendLower = findHighestKeyValuePair(whichKey, dividend, option, lowerRange/2, false, fileAsDictionary);
                }
                Tuple<float, float>^ retValDivisor;
                if (!singleMode) {
                    retValDivisor = findHighestKeyValuePair(whichKey+1, divisor, option, rangeEachWay, true, fileAsDictionary);
                    if (doLowerRange) {
                         retValDividerLower = findHighestKeyValuePair(whichKey+1, divisor, option, lowerRange / 2, false, fileAsDictionary);
                    }
                }
                //res = dict[dividend] / (singleMode ? 1 : dict[divisor]);
                //runningSum += res;
                //pointer - will retain information
                List<Dictionary<float, float>^>^ temp = listOfProcessedSets[i];
                temp[j]->Add(retValDividend->Item1, retValDividend->Item2);
                if (!singleMode) {
                    temp[j]->Add(retValDivisor->Item1, retValDivisor->Item2);
                }
                else{
                    //adding a value of 1 with a key of 1 to divide by 1 for result. Key of 1 works as a flag
                    temp[j]->Add(1, 1);
                }
                if (doLowerRange) {
                    //third and fourth column will be reserved for the smallest value within the given range.
                    temp[j]->Add(retValDividendLower->Item1, retValDividendLower->Item2);
                    if (!singleMode) {
                        temp[j]->Add(retValDividerLower->Item1, retValDividerLower->Item2);
                    }
                }

                j++;
            }
            if (differentDivisors) {
                whichKey += 2;
            }
            i++;
        }
        //we have now found the wavelength of biggest intensity in  range. We may proceed to calculating the averages for each set.
        listOfAveragedIndividualResults_sets = gcnew  List<List<float>^>();
        i = 0;
        for each (List<Dictionary<float, float>^> ^ set  in listOfProcessedSets) {
            listOfAveragedIndividualResults_sets->Add(averageIndividualKeyValuePairs(set));
            //there will actually be only two pieces of information in each list anyway, making this perfect
            listOfAverages[i] = returnDivisionFromTwoFirst(listOfAveragedIndividualResults_sets[i]);
            i++;
        }
        return 1;
    }

    int numberOfValidSets() {
        int elems = 0;
        for each (List<Dictionary<float, float>^> ^ set in listOfSets) {
            if (set != nullptr) { elems++; }
        }
        return elems;
    }

    System::Tuple<float,float>^ findHighestKeyValuePair(int i, float key, int option, float rangeEachWay, bool whichDir, Dictionary<float,float>^ inputDict) {
        int index = userSelectionsIndexes[i];
        float currKey = indexedKeys[index];
        float tempKey, tempResult;
        if (!whichDir) {
            tempResult = 99999; tempKey = -1;
        }
        else if (option == 1) { tempResult = -9999; tempKey = -1; }
        else if (option == 2) { tempResult = 0; tempKey = key;}
        while (Math::Abs(key - currKey) < rangeEachWay) {
            //if we're finding the lowest value, we ignore what option was selected (by design)
             if (!whichDir) {
                tempResult = inputDict[currKey] < tempResult ? inputDict[currKey] : tempResult;
                tempKey = inputDict[currKey] == tempResult ? currKey : tempKey;
            }
            else if (option == 1) {
                tempResult = inputDict[currKey] > tempResult ? inputDict[currKey] : tempResult;
                tempKey = inputDict[currKey] == tempResult ? currKey : tempKey;
            }
            else {
                tempResult += inputDict[currKey];
            }
            if (index == 0) { break; }
            index--;

            currKey = indexedKeys[index];
        }
        //escaped the while loop. Now do it again, but the other way. Admittedly there is redundancy in this code.
        index = userSelectionsIndexes[i];
        currKey = indexedKeys[index + 1];
        while (Math::Abs(key - currKey) < rangeEachWay) {
            if (option == 1 && whichDir) {
                tempResult = inputDict[currKey] > tempResult ? inputDict[currKey] : tempResult;
                tempKey = inputDict[currKey] == tempResult ? currKey : tempKey;

            }
            else if (!whichDir) {
                tempResult = inputDict[currKey] < tempResult ? inputDict[currKey] : tempResult;
                tempKey = inputDict[currKey] == tempResult ? currKey : tempKey;
            }

            else {
                tempResult += inputDict[currKey];
            }
            if (index == DATASIZE - LINESTOSKIP) { break; }
            index++;
            currKey = indexedKeys[index];
        }
        //finally, found the highest key & the value corresponding. Return
        System::Tuple<float, float>^ retVal = gcnew Tuple<float, float>(tempKey, tempResult);
        return retVal;
    }

    List<float>^ averageIndividualKeyValuePairs(List<Dictionary<float, float>^>^ LOD) {
        int i;
        int howManyWavelengths = LOD[0]->Count;
        List<float>^ pointerToResult = gcnew List<float>(howManyWavelengths);
        for (int i = 0; i < howManyWavelengths; i++) {
            pointerToResult->Add(0);
        }
        for each(Dictionary<float,float>^ dict in LOD) {
            i = 0;
            for each (float key in dict->Keys) {
                pointerToResult[i] += dict[key];
                i++;
            }
        }
        //now we have our list; divide each item by times called
        for (int i = 0; i < howManyWavelengths; i++) {
            pointerToResult[i] = pointerToResult[i] / LOD->Count;
        }

        //success
        return pointerToResult;
    }

    //function to return a division of two first values from list
    float returnDivisionFromTwoFirst(List<float>^ givenList) {
        if (givenList->Count < 2) {
            return 0;
        }
        if (givenList[0] > givenList[1]) {
            return givenList[0] / givenList[1];
        }
        else {
            return givenList[1] / givenList[0];
        }
    }
```

```
};
```

5.3 C:/MEng/LIBSProcessing/LIBSProcessing/Window.cpp File Reference

File running the programme.

```
#include "Window.h"
```

Functions

- void **main** ()

5.3.1 Detailed Description

File running the programme.

Author

PR

Date

April 2023

5.4 C:/MEng/LIBSProcessing/LIBSProcessing/Window.h File Reference

File handling the UI of the programme.

```
#include "Backend.h"
```

Classes

- class LIBSProcessing::Window

  Class handling the display of the user interface (later referred to as GUI or UI). Primarily has functions responsible for recording button presses.

5.4.1 Detailed Description

File handling the UI of the programme.

Author

PR

Date

April 2023

5.5 Window.h

Go to the documentation of this file.

```
/*********************************************************************/
#pragma once
#include "Backend.h"

namespace LIBSProcessing {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    public ref class Window : public System::Windows::Forms::Form
    {
    public:
        Window(void)
        {
            InitializeComponent();
            nameOfFile->Text = System::DateTime::Now.ToString("dd_MM_hhmm")+".csv";
            //Initializing the backend here.


        }
    private: System::Windows::Forms::CheckBox^ saveSelectedBox;
    private: System::Windows::Forms::RadioButton^ highestCheckbox;
    private: System::Windows::Forms::RadioButton^ sumCheckbox;
    private: System::Windows::Forms::GroupBox^ groupBox1;
    private: System::Windows::Forms::TextBox^ noiseCutoff;
    private: System::Windows::Forms::Label^ label8;
    private: System::Windows::Forms::Label^ label9;
    private: System::Windows::Forms::Label^ cutoffLabel;
    private: System::Windows::Forms::MenuStrip^ menuStrip1;
    private: System::Windows::Forms::ToolStripMenuItem^ modeToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^ standardToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^ calibrationToolStripMenuItem;
    private: System::Windows::Forms::Label^ setAlabel;
    private: System::Windows::Forms::Label^ setBlabel;



    private: System::Windows::Forms::Label^ selectFilesLabel_setB;

    private: System::Windows::Forms::Button^ fileSelect_setB;


    private: System::Windows::Forms::Label^ analyteLabel_setB;


    private: System::Windows::Forms::TextBox^ analyteBox_setB;
    private: System::ComponentModel::BackgroundWorker^ backgroundWorker1;
    private: System::Windows::Forms::Label^ setNumbersLabel;
    private: System::Windows::Forms::Label^ howManyLabel;
    private: System::Windows::Forms::Button^ howManySubmit;


    private: System::Windows::Forms::TextBox^ howManySets;
    private: System::Windows::Forms::ComboBox^ setsOfData;
    private: System::Windows::Forms::Label^ setsOfData_label;
    private: System::Windows::Forms::Button^ addSetButton;
    private: System::Windows::Forms::Label^ Rscore;
    private: System::Windows::Forms::TextBox^ rangeLowerInput;
    private: System::Windows::Forms::Label^ label1;
    private: System::Windows::Forms::GroupBox^ groupBox2;
    private: System::Windows::Forms::RadioButton^ lowerRangeYes;

    private: System::Windows::Forms::RadioButton^ radioButton2;







    public:

    protected:
        Backend b;
        ~Window()
        {
            if (components)
            {
                delete components;
            }
        }

    protected:

    private: System::Windows::Forms::TextBox^ waveEdit;

    private: System::Windows::Forms::Button^ waveSubmit;

    private: System::Windows::Forms::Label^ label2;


    private: System::Windows::Forms::TextBox^ rangeInput;
    private: System::Windows::Forms::Label^ label3;

    private: System::Windows::Forms::ToolTip^ toolTip1;
    private: System::Windows::Forms::ComboBox^ allWavelenghts;
    private: System::Windows::Forms::Label^ label4;
    private: System::Windows::Forms::Button^ removeWave;
    private: System::Windows::Forms::Button^ saveFolderSelect;
    private: System::Windows::Forms::FolderBrowserDialog^ folderBrowser;

    private: System::Windows::Forms::Label^ label5;
    private: System::Windows::Forms::TextBox^ savePath;
    private: System::Windows::Forms::TextBox^ nameOfFile;

    private: System::Windows::Forms::Label^ label6;
    private: System::Windows::Forms::Button^ saveToFile;
    private: System::Windows::Forms::Button^ fileSelect;
    private: System::Windows::Forms::Label^ label7;
    private: System::Windows::Forms::Button^ preview;
    private: System::Windows::Forms::Label^ noOfFiles;
    private: System::Windows::Forms::OpenFileDialog^ fileOpener;




    private: System::ComponentModel::IContainer^ components;

    private:


#pragma region Windows Form Designer generated code
        void InitializeComponent(void)
        {
            this->components = (gcnew System::ComponentModel::Container());
            System::ComponentModel::ComponentResourceManager^ resources = (gcnew
System::ComponentModel::ComponentResourceManager(Window::typeid));
            this->waveEdit = (gcnew System::Windows::Forms::TextBox());
            this->waveSubmit = (gcnew System::Windows::Forms::Button());
            this->label2 = (gcnew System::Windows::Forms::Label());
            this->rangeInput = (gcnew System::Windows::Forms::TextBox());
            this->label3 = (gcnew System::Windows::Forms::Label());
            this->toolTip1 = (gcnew System::Windows::Forms::ToolTip(this->components));
            this->noiseCutoff = (gcnew System::Windows::Forms::TextBox());
            this->analyteBox_setB = (gcnew System::Windows::Forms::TextBox());
            this->setNumbersLabel = (gcnew System::Windows::Forms::Label());
            this->allWavelenghts = (gcnew System::Windows::Forms::ComboBox());
            this->label4 = (gcnew System::Windows::Forms::Label());
            this->removeWave = (gcnew System::Windows::Forms::Button());
            this->saveFolderSelect = (gcnew System::Windows::Forms::Button());
            this->folderBrowser = (gcnew System::Windows::Forms::FolderBrowserDialog());
            this->label5 = (gcnew System::Windows::Forms::Label());
            this->savePath = (gcnew System::Windows::Forms::TextBox());
            this->nameOfFile = (gcnew System::Windows::Forms::TextBox());
            this->label6 = (gcnew System::Windows::Forms::Label());
            this->saveToFile = (gcnew System::Windows::Forms::Button());
            this->fileSelect = (gcnew System::Windows::Forms::Button());
            this->label7 = (gcnew System::Windows::Forms::Label());
            this->preview = (gcnew System::Windows::Forms::Button());
            this->noOfFiles = (gcnew System::Windows::Forms::Label());
            this->fileOpener = (gcnew System::Windows::Forms::OpenFileDialog());
            this->saveSelectedBox = (gcnew System::Windows::Forms::CheckBox());
            this->highestCheckbox = (gcnew System::Windows::Forms::RadioButton());
            this->sumCheckbox = (gcnew System::Windows::Forms::RadioButton());
            this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
            this->label8 = (gcnew System::Windows::Forms::Label());
            this->label9 = (gcnew System::Windows::Forms::Label());
            this->cutoffLabel = (gcnew System::Windows::Forms::Label());
            this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
            this->modeToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
            this->standardToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
            this->calibrationToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());
            this->setAlabel = (gcnew System::Windows::Forms::Label());
            this->setBlabel = (gcnew System::Windows::Forms::Label());
            this->selectFilesLabel_setB = (gcnew System::Windows::Forms::Label());
            this->fileSelect_setB = (gcnew System::Windows::Forms::Button());
            this->analyteLabel_setB = (gcnew System::Windows::Forms::Label());
            this->backgroundWorker1 = (gcnew System::ComponentModel::BackgroundWorker());
            this->howManyLabel = (gcnew System::Windows::Forms::Label());
            this->howManySubmit = (gcnew System::Windows::Forms::Button());
            this->howManySets = (gcnew System::Windows::Forms::TextBox());
            this->setsOfData = (gcnew System::Windows::Forms::ComboBox());
            this->setsOfData_label = (gcnew System::Windows::Forms::Label());
            this->addSetButton = (gcnew System::Windows::Forms::Button());
            this->Rscore = (gcnew System::Windows::Forms::Label());
            this->rangeLowerInput = (gcnew System::Windows::Forms::TextBox());
            this->label1 = (gcnew System::Windows::Forms::Label());
            this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
            this->lowerRangeYes = (gcnew System::Windows::Forms::RadioButton());
            this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());
            this->groupBox1->SuspendLayout();
            this->menuStrip1->SuspendLayout();
            this->groupBox2->SuspendLayout();
            this->SuspendLayout();
            //
            // waveEdit
            //
            this->waveEdit->Location = System::Drawing::Point(12, 46);
            this->waveEdit->Name = L"waveEdit";
            this->waveEdit->Size = System::Drawing::Size(100, 20);
            this->waveEdit->TabIndex = 2;
            //
            // waveSubmit
            //
            this->waveSubmit->Location = System::Drawing::Point(133, 43);
            this->waveSubmit->Name = L"waveSubmit";
            this->waveSubmit->Size = System::Drawing::Size(75, 23);
            this->waveSubmit->TabIndex = 3;
            this->waveSubmit->Text = L"Submit";
            this->waveSubmit->UseVisualStyleBackColor = true;
            this->waveSubmit->Click += gcnew System::EventHandler(this, &Window::waveSubmit_Click);
            //
            // label2
            //
            this->label2->AutoSize = true;
            this->label2->Location = System::Drawing::Point(9, 27);
            this->label2->Name = L"label2";
            this->label2->Size = System::Drawing::Size(89, 13);
            this->label2->TabIndex = 4;
            this->label2->Text = L"Add wavelengths";
            //
            // rangeInput
            //
            this->rangeInput->Location = System::Drawing::Point(12, 88);
            this->rangeInput->Name = L"rangeInput";
            this->rangeInput->Size = System::Drawing::Size(100, 20);
            this->rangeInput->TabIndex = 6;
            this->rangeInput->Text = L"0.07";
            //
            // label3
            //
            this->label3->AutoSize = true;
            this->label3->Location = System::Drawing::Point(9, 72);
            this->label3->Name = L"label3";
            this->label3->Size = System::Drawing::Size(118, 13);
            this->label3->TabIndex = 7;
            this->label3->Text = L"Range (default 0.07nm)";
            //
            // noiseCutoff
            //
            this->noiseCutoff->Location = System::Drawing::Point(244, 43);
            this->noiseCutoff->Name = L"noiseCutoff";
            this->noiseCutoff->Size = System::Drawing::Size(100, 20);
            this->noiseCutoff->TabIndex = 29;
            this->toolTip1->SetToolTip(this->noiseCutoff, L"Values below this threshold will be dropped to 0.");
            //
            // analyteBox_setB
            //
            this->analyteBox_setB->Enabled = false;
            this->analyteBox_setB->Location = System::Drawing::Point(244, 258);
            this->analyteBox_setB->Name = L"analyteBox_setB";
            this->analyteBox_setB->Size = System::Drawing::Size(100, 20);
            this->analyteBox_setB->TabIndex = 42;
            this->toolTip1->SetToolTip(this->analyteBox_setB, L"Values below this threshold will be dropped to 0.");
            //
            // setNumbersLabel
            //
            this->setNumbersLabel->AutoSize = true;
            this->setNumbersLabel->Enabled = false;
            this->setNumbersLabel->Location = System::Drawing::Point(9, 201);
            this->setNumbersLabel->Name = L"setNumbersLabel";
            this->setNumbersLabel->Size = System::Drawing::Size(161, 26);
            this->setNumbersLabel->TabIndex = 44;
            this->setNumbersLabel->Text = L"Set 1: 1st/2nd; Set 2: 3rd/4th... \r\n Hover over for more information.";
            this->toolTip1->SetToolTip(this->setNumbersLabel, resources->GetString(L"setNumbersLabel.ToolTip"));
            //
            // allWavelenghts
            //
            this->allWavelenghts->FormattingEnabled = true;
            this->allWavelenghts->Location = System::Drawing::Point(12, 175);
            this->allWavelenghts->MaxDropDownItems = 100;
            this->allWavelenghts->Name = L"allWavelenghts";
            this->allWavelenghts->Size = System::Drawing::Size(100, 21);
            this->allWavelenghts->TabIndex = 9;
            //
            // label4
            //
            this->label4->AutoSize = true;
            this->label4->Location = System::Drawing::Point(9, 159);
            this->label4->Name = L"label4";
            this->label4->Size = System::Drawing::Size(154, 13);
            this->label4->TabIndex = 10;
            this->label4->Text = L"Currently selected wavelenghts";
            //
            // removeWave
            //
            this->removeWave->Location = System::Drawing::Point(133, 175);
            this->removeWave->Name = L"removeWave";
            this->removeWave->Size = System::Drawing::Size(75, 23);
            this->removeWave->TabIndex = 11;
            this->removeWave->Text = L"Remove";
            this->removeWave->UseVisualStyleBackColor = true;
            this->removeWave->Click += gcnew System::EventHandler(this, &Window::removeWave_Click);
            //
            // saveFolderSelect
            //
            this->saveFolderSelect->Location = System::Drawing::Point(170, 450);
            this->saveFolderSelect->Name = L"saveFolderSelect";
            this->saveFolderSelect->Size = System::Drawing::Size(75, 23);
            this->saveFolderSelect->TabIndex = 12;
            this->saveFolderSelect->Text = L"Browse...";
            this->saveFolderSelect->UseVisualStyleBackColor = true;
            this->saveFolderSelect->Click += gcnew System::EventHandler(this, &Window::saveFolderSelect_Click);
            //
            // label5
            //
            this->label5->AutoSize = true;
            this->label5->Location = System::Drawing::Point(17, 435);
            this->label5->Name = L"label5";
            this->label5->Size = System::Drawing::Size(72, 13);
            this->label5->TabIndex = 13;
            this->label5->Text = L"Save location";
            //
            // savePath
            //
            this->savePath->Location = System::Drawing::Point(20, 452);
            this->savePath->Name = L"savePath";
            this->savePath->ReadOnly = true;
            this->savePath->Size = System::Drawing::Size(143, 20);
            this->savePath->TabIndex = 14;
            //
            // nameOfFile
            //
            this->nameOfFile->Location = System::Drawing::Point(277, 453);
            this->nameOfFile->Name = L"nameOfFile";
            this->nameOfFile->Size = System::Drawing::Size(143, 20);
            this->nameOfFile->TabIndex = 15;
            this->nameOfFile->Text = L"TEMP.csv";
            //
            // label6
            //
            this->label6->AutoSize = true;
            this->label6->Location = System::Drawing::Point(273, 435);
            this->label6->Name = L"label6";
            this->label6->Size = System::Drawing::Size(104, 13);
            this->label6->TabIndex = 16;
            this->label6->Text = L"Name of file to save:";
            //
            // saveToFile
            //
            this->saveToFile->Location = System::Drawing::Point(426, 452);
            this->saveToFile->Name = L"saveToFile";
            this->saveToFile->Size = System::Drawing::Size(75, 23);
            this->saveToFile->TabIndex = 17;
            this->saveToFile->Text = L"Save";
            this->saveToFile->UseVisualStyleBackColor = true;
            this->saveToFile->Click += gcnew System::EventHandler(this, &Window::saveToFile_Click);
            //
            // fileSelect
            //
            this->fileSelect->Location = System::Drawing::Point(425, 93);
            this->fileSelect->Name = L"fileSelect";
            this->fileSelect->Size = System::Drawing::Size(75, 23);
            this->fileSelect->TabIndex = 18;
            this->fileSelect->Text = L"Browse...";
            this->fileSelect->UseVisualStyleBackColor = true;
            this->fileSelect->Click += gcnew System::EventHandler(this, &Window::fileSelect_Click);
            //
            // label7
            //
            this->label7->AutoSize = true;
            this->label7->Location = System::Drawing::Point(241, 95);
            this->label7->Name = L"label7";
            this->label7->Size = System::Drawing::Size(113, 13);
            this->label7->TabIndex = 19;
            this->label7->Text = L"Select files to process:";
            //
            // preview
            //
            this->preview->Location = System::Drawing::Point(204, 389);
            this->preview->Name = L"preview";
            this->preview->Size = System::Drawing::Size(108, 21);
            this->preview->TabIndex = 20;
            this->preview->Text = L"Process results";
            this->preview->UseVisualStyleBackColor = true;
            this->preview->Click += gcnew System::EventHandler(this, &Window::preview_Click);
            //
            // noOfFiles
            //
            this->noOfFiles->AutoSize = true;
            this->noOfFiles->Location = System::Drawing::Point(422, 119);
            this->noOfFiles->Name = L"noOfFiles";
            this->noOfFiles->Size = System::Drawing::Size(80, 13);
            this->noOfFiles->TabIndex = 21;
            this->noOfFiles->Text = L"files selected: 0";
            //
            // fileOpener
            //
            this->fileOpener->FileName = L"Select your files...";
            this->fileOpener->Filter = L"ASC files (*.asc)|*.asc";
            this->fileOpener->Multiselect = true;
            //
            // saveSelectedBox
            //
            this->saveSelectedBox->AutoSize = true;
            this->saveSelectedBox->Checked = true;
            this->saveSelectedBox->CheckState = System::Windows::Forms::CheckState::Checked;
            this->saveSelectedBox->Location = System::Drawing::Point(12, 237);
            this->saveSelectedBox->Name = L"saveSelectedBox";
            this->saveSelectedBox->Size = System::Drawing::Size(179, 17);
            this->saveSelectedBox->TabIndex = 23;
            this->saveSelectedBox->Text = L"Save selected wavelengths only";
            this->saveSelectedBox->UseVisualStyleBackColor = true;
            //
            // highestCheckbox
            //
            this->highestCheckbox->AutoSize = true;
            this->highestCheckbox->Checked = true;
            this->highestCheckbox->Location = System::Drawing::Point(6, 10);
            this->highestCheckbox->Name = L"highestCheckbox";
            this->highestCheckbox->Size = System::Drawing::Size(102, 17);
            this->highestCheckbox->TabIndex = 26;
            this->highestCheckbox->TabStop = true;
            this->highestCheckbox->Text = L"Highest in range";
            this->highestCheckbox->UseVisualStyleBackColor = true;
            //
            // sumCheckbox
            //
            this->sumCheckbox->AutoSize = true;
            this->sumCheckbox->Location = System::Drawing::Point(6, 26);
            this->sumCheckbox->Name = L"sumCheckbox";
            this->sumCheckbox->Size = System::Drawing::Size(87, 17);
            this->sumCheckbox->TabIndex = 27;
            this->sumCheckbox->Text = L"Sum in range";
            this->sumCheckbox->UseVisualStyleBackColor = true;
            //
            // groupBox1
            //
            this->groupBox1->Controls->Add(this->highestCheckbox);
            this->groupBox1->Controls->Add(this->sumCheckbox);
            this->groupBox1->Location = System::Drawing::Point(132, 67);
            this->groupBox1->Name = L"groupBox1";
            this->groupBox1->Size = System::Drawing::Size(103, 43);
            this->groupBox1->TabIndex = 28;
            this->groupBox1->TabStop = false;
            //
            // label8
            //
            this->label8->AutoSize = true;
            this->label8->Location = System::Drawing::Point(241, 24);
            this->label8->Name = L"label8";
            this->label8->Size = System::Drawing::Size(192, 13);
            this->label8->TabIndex = 30;
            this->label8->Text = L"Noise cutoff (select before loading files)";
            //
            // label9
            //
            this->label9->AutoSize = true;
            this->label9->Location = System::Drawing::Point(364, 46);
            this->label9->Name = L"label9";
            this->label9->Size = System::Drawing::Size(126, 13);
            this->label9->TabIndex = 31;
            this->label9->Text = L"Leave blank for no cutoff";
            //
            // cutoffLabel
            //
            this->cutoffLabel->AutoSize = true;
            this->cutoffLabel->Location = System::Drawing::Point(422, 132);
            this->cutoffLabel->Name = L"cutoffLabel";
            this->cutoffLabel->Size = System::Drawing::Size(64, 13);
            this->cutoffLabel->TabIndex = 32;
            this->cutoffLabel->Text = L"at no cutoff.";
            //
            // menuStrip1
            //
            this->menuStrip1->BackColor = System::Drawing::SystemColors::ButtonShadow;
            this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^
      >(1) { this->modeToolStripMenuItem });
            this->menuStrip1->Location = System::Drawing::Point(0, 0);
            this->menuStrip1->Name = L"menuStrip1";
            this->menuStrip1->Size = System::Drawing::Size(519, 24);
            this->menuStrip1->TabIndex = 33;
            this->menuStrip1->Text = L"menuStrip1";
            //
            // modeToolStripMenuItem
            //
            this->modeToolStripMenuItem->BackColor = System::Drawing::SystemColors::Control;
            this->modeToolStripMenuItem->BackgroundImageLayout =
      System::Windows::Forms::ImageLayout::Center;
            this->modeToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
      System::Windows::Forms::ToolStripItem^  >(2) {
                this->standardToolStripMenuItem,
                    this->calibrationToolStripMenuItem
            });
            this->modeToolStripMenuItem->Name = L"modeToolStripMenuItem";
            this->modeToolStripMenuItem->Size = System::Drawing::Size(50, 20);
            this->modeToolStripMenuItem->Text = L"Mode";
            //
            // standardToolStripMenuItem
            //
            this->standardToolStripMenuItem->Checked = true;
            this->standardToolStripMenuItem->CheckState = System::Windows::Forms::CheckState::Checked;
            this->standardToolStripMenuItem->Name = L"standardToolStripMenuItem";
            this->standardToolStripMenuItem->Size = System::Drawing::Size(132, 22);
            this->standardToolStripMenuItem->Text = L"Standard";
            this->standardToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Window::standardToolStripMenuItem_Click);
            //
            // calibrationToolStripMenuItem
            //
            this->calibrationToolStripMenuItem->Name = L"calibrationToolStripMenuItem";
            this->calibrationToolStripMenuItem->Size = System::Drawing::Size(132, 22);
            this->calibrationToolStripMenuItem->Text = L"Calibration";
            this->calibrationToolStripMenuItem->Click += gcnew System::EventHandler(this,
&Window::calibrationToolStripMenuItem_Click);
            //
            // setAlabel
            //
            this->setAlabel->AutoSize = true;
            this->setAlabel->Location = System::Drawing::Point(241, 77);
            this->setAlabel->Name = L"setAlabel";
            this->setAlabel->Size = System::Drawing::Size(110, 13);
            this->setAlabel->TabIndex = 34;
            this->setAlabel->Text = L"Single-set processing:";
            //
            // setBlabel
            //
            this->setBlabel->AutoSize = true;
            this->setBlabel->Enabled = false;
            this->setBlabel->Location = System::Drawing::Point(241, 141);
            this->setBlabel->Name = L"setBlabel";
            this->setBlabel->Size = System::Drawing::Size(103, 13);
            this->setBlabel->TabIndex = 39;
            this->setBlabel->Text = L"Multi-set processing:";
            //
            // selectFilesLabel_setB
            //
            this->selectFilesLabel_setB->AutoSize = true;
            this->selectFilesLabel_setB->Enabled = false;
            this->selectFilesLabel_setB->Location = System::Drawing::Point(241, 221);
            this->selectFilesLabel_setB->Name = L"selectFilesLabel_setB";
            this->selectFilesLabel_setB->Size = System::Drawing::Size(113, 13);
            this->selectFilesLabel_setB->TabIndex = 36;
            this->selectFilesLabel_setB->Text = L"Select files to process:";
            //
            // fileSelect_setB
            //
            this->fileSelect_setB->Enabled = false;
            this->fileSelect_setB->Location = System::Drawing::Point(425, 219);
            this->fileSelect_setB->Name = L"fileSelect_setB";
            this->fileSelect_setB->Size = System::Drawing::Size(75, 23);
            this->fileSelect_setB->TabIndex = 35;
            this->fileSelect_setB->Text = L"Browse...";
            this->fileSelect_setB->UseVisualStyleBackColor = true;
            this->fileSelect_setB->Click += gcnew System::EventHandler(this,
&Window::fileSelect_setB_Click);
            //
            // analyteLabel_setB
            //
            this->analyteLabel_setB->AutoSize = true;
            this->analyteLabel_setB->Enabled = false;
            this->analyteLabel_setB->Location = System::Drawing::Point(241, 244);
            this->analyteLabel_setB->Name = L"analyteLabel_setB";
            this->analyteLabel_setB->Size = System::Drawing::Size(174, 13);
            this->analyteLabel_setB->TabIndex = 43;
            this->analyteLabel_setB->Text = L"Analyte concentration (ppm) for given set:";
            //
            // howManyLabel
            //
            this->howManyLabel->AutoSize = true;
            this->howManyLabel->Enabled = false;
            this->howManyLabel->Location = System::Drawing::Point(241, 157);
            this->howManyLabel->Name = L"howManyLabel";
            this->howManyLabel->Size = System::Drawing::Size(131, 13);
            this->howManyLabel->TabIndex = 47;
            this->howManyLabel->Text = L"How may sets to process\?";
            //
            // howManySubmit
            //
            this->howManySubmit->Enabled = false;
            this->howManySubmit->Location = System::Drawing::Point(425, 176);
            this->howManySubmit->Name = L"howManySubmit";
            this->howManySubmit->Size = System::Drawing::Size(75, 23);
            this->howManySubmit->TabIndex = 46;
            this->howManySubmit->Text = L"Submit";
            this->howManySubmit->UseVisualStyleBackColor = true;
            this->howManySubmit->Click += gcnew System::EventHandler(this,
&Window::howManySubmit_Click);
            //
            // howManySets
            //
            this->howManySets->Enabled = false;
            this->howManySets->Location = System::Drawing::Point(244, 176);
            this->howManySets->Name = L"howManySets";
            this->howManySets->Size = System::Drawing::Size(100, 20);
            this->howManySets->TabIndex = 45;
            //
            // setsOfData
            //
            this->setsOfData->Enabled = false;
            this->setsOfData->FormattingEnabled = true;
            this->setsOfData->Location = System::Drawing::Point(244, 311);
            this->setsOfData->MaxDropDownItems = 20;
            this->setsOfData->Name = L"setsOfData";
            this->setsOfData->Size = System::Drawing::Size(256, 21);
            this->setsOfData->TabIndex = 48;
            //
            // setsOfData_label
            //
            this->setsOfData_label->AutoSize = true;
            this->setsOfData_label->Enabled = false;
            this->setsOfData_label->Location = System::Drawing::Point(241, 295);
            this->setsOfData_label->Name = L"setsOfData_label";
            this->setsOfData_label->Size = System::Drawing::Size(118, 13);
            this->setsOfData_label->TabIndex = 49;
            this->setsOfData_label->Text = L"Sets of data information";
            //
            // addSetButton
            //
            this->addSetButton->Enabled = false;
            this->addSetButton->Location = System::Drawing::Point(425, 258);
            this->addSetButton->Name = L"addSetButton";
            this->addSetButton->Size = System::Drawing::Size(75, 23);
            this->addSetButton->TabIndex = 50;
            this->addSetButton->Text = L"Add set";
            this->addSetButton->UseVisualStyleBackColor = true;
            this->addSetButton->Click += gcnew System::EventHandler(this,
      &Window::addSetButton_Click);
            //
            // Rscore
            //
            this->Rscore->AutoSize = true;
            this->Rscore->Location = System::Drawing::Point(201, 413);
            this->Rscore->Name = L"Rscore";
            this->Rscore->Size = System::Drawing::Size(0, 13);
            this->Rscore->TabIndex = 51;
            //
            // rangeLowerInput
            //
            this->rangeLowerInput->Location = System::Drawing::Point(12, 129);
            this->rangeLowerInput->Name = L"rangeLowerInput";
            this->rangeLowerInput->Size = System::Drawing::Size(100, 20);
            this->rangeLowerInput->TabIndex = 52;
            this->rangeLowerInput->Text = L"0.2";
            //
            // label1
            //
            this->label1->AutoSize = true;
            this->label1->Location = System::Drawing::Point(9, 113);
            this->label1->Name = L"label1";
            this->label1->Size = System::Drawing::Size(147, 13);
            this->label1->TabIndex = 53;
            this->label1->Text = L"Range for finding lowest point";
            //
            // groupBox2
            //
            this->groupBox2->Controls->Add(this->lowerRangeYes);
            this->groupBox2->Controls->Add(this->radioButton2);
            this->groupBox2->Location = System::Drawing::Point(176, 113);
            this->groupBox2->Name = L"groupBox2";
            this->groupBox2->Size = System::Drawing::Size(59, 49);
            this->groupBox2->TabIndex = 29;
            this->groupBox2->TabStop = false;
            //
            // lowerRangeYes
            //
            this->lowerRangeYes->AutoSize = true;
            this->lowerRangeYes->Checked = true;
            this->lowerRangeYes->Location = System::Drawing::Point(6, 10);
            this->lowerRangeYes->Name = L"lowerRangeYes";
            this->lowerRangeYes->Size = System::Drawing::Size(43, 17);
            this->lowerRangeYes->TabIndex = 26;
            this->lowerRangeYes->TabStop = true;
            this->lowerRangeYes->Text = L"Yes";
            this->lowerRangeYes->UseVisualStyleBackColor = true;
            //
            // radioButton2
            //
            this->radioButton2->AutoSize = true;
            this->radioButton2->Location = System::Drawing::Point(6, 26);
            this->radioButton2->Name = L"radioButton2";
            this->radioButton2->Size = System::Drawing::Size(46, 17);
            this->radioButton2->TabIndex = 27;
            this->radioButton2->Text = L"Skip";
            this->radioButton2->UseVisualStyleBackColor = true;
            //
            // Window
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(519, 495);
            this->Controls->Add(this->groupBox2);
            this->Controls->Add(this->label1);
            this->Controls->Add(this->rangeLowerInput);
            this->Controls->Add(this->Rscore);
            this->Controls->Add(this->addSetButton);
            this->Controls->Add(this->setsOfData_label);
            this->Controls->Add(this->setsOfData);
            this->Controls->Add(this->howManyLabel);
            this->Controls->Add(this->howManySubmit);
            this->Controls->Add(this->howManySets);
            this->Controls->Add(this->setNumbersLabel);
            this->Controls->Add(this->analyteLabel_setB);
            this->Controls->Add(this->analyteBox_setB);
            this->Controls->Add(this->setBlabel);
            this->Controls->Add(this->selectFilesLabel_setB);
            this->Controls->Add(this->fileSelect_setB);
            this->Controls->Add(this->setAlabel);
            this->Controls->Add(this->cutoffLabel);
            this->Controls->Add(this->label9);
            this->Controls->Add(this->label8);
            this->Controls->Add(this->noiseCutoff);
            this->Controls->Add(this->groupBox1);
            this->Controls->Add(this->saveSelectedBox);
            this->Controls->Add(this->noOfFiles);
            this->Controls->Add(this->preview);
            this->Controls->Add(this->label7);
            this->Controls->Add(this->fileSelect);
            this->Controls->Add(this->saveToFile);
            this->Controls->Add(this->label6);
            this->Controls->Add(this->nameOfFile);
            this->Controls->Add(this->savePath);
            this->Controls->Add(this->label5);
            this->Controls->Add(this->saveFolderSelect);
            this->Controls->Add(this->removeWave);
            this->Controls->Add(this->label4);
            this->Controls->Add(this->allWavelenghts);
            this->Controls->Add(this->label3);
            this->Controls->Add(this->rangeInput);
            this->Controls->Add(this->label2);
            this->Controls->Add(this->waveSubmit);
            this->Controls->Add(this->waveEdit);
            this->Controls->Add(this->menuStrip1);
            this->MainMenuStrip = this->menuStrip1;
            this->Name = L"Window";
            this->Text = L"Zeus";
            this->Load += gcnew System::EventHandler(this, &Window::Window_Load);
            this->groupBox1->ResumeLayout(false);
            this->groupBox1->PerformLayout();
            this->menuStrip1->ResumeLayout(false);
            this->menuStrip1->PerformLayout();
            this->groupBox2->ResumeLayout(false);
            this->groupBox2->PerformLayout();
            this->ResumeLayout(false);
            this->PerformLayout();

        }
#pragma endregion


        //CODE HANDLING THE UI & calls to the "backend".

    //GUI handler - submit an ELEMENT'S wavelengths to the list
    private: System::Void elemSubmit_Click(System::Object^ sender, System::EventArgs^ e) {

        //Currently empty

    }
    private: System::Void waveSubmit_Click(System::Object^ sender, System::EventArgs^ e) {
        float attemptConversion;
        //try converting - if failed, show a message to the user
        try {
            attemptConversion = Convert::ToSingle(waveEdit->Text);
        }
        catch (...) {
            MessageBox::Show("Error - please input a float value");
```

```
                return;
            }

            if (!b.addWavelength(attemptConversion)) {
                MessageBox::Show("Error - please input a value between 200.93 and 1031.86");
            }
            //for some reason, it order to update the list in the GUI it needs to be fully reset
            allWavelenghts->DataSource = nullptr;
            allWavelenghts->DataSource = b.selectedWavelengths;
            waveEdit->Text = "";

        }
        private: System::Void saveFolderSelect_Click(System::Object^ sender, System::EventArgs^ e) {
            if (folderBrowser->ShowDialog() == System::Windows::Forms::DialogResult::OK)
            {
                String^ folderName = folderBrowser->SelectedPath;
                savePath->Text = folderName;
                b.directory = folderName;       //set the directory in the backend
            }
        }
        private: System::Void removeWave_Click(System::Object^ sender, System::EventArgs^ e) {
            float waveToRemove = Convert::ToSingle(allWavelenghts->Text);
            b.removeWavelength(waveToRemove);
            allWavelenghts->DataSource = nullptr;
            allWavelenghts->DataSource = b.selectedWavelengths;
        }

        //GUI handler - preview all options - actually processes the data for now as well
        private: System::Void preview_Click(System::Object^ sender, System::EventArgs^ e) {
            if (b.getAveragedSpectra()) {}
            else { MessageBox::Show("Error - no files loaded"); return; }
            int option;
            //option 1 - highest point in range, skip range for finding lowest point
            //option 2 - sum in range, skip range for finding lowest point
            //option 3 - highest point in range, yes for range for finding lowest point
            //option 4 - sum in range, yes for range for finding lowest point
            if (highestCheckbox->Checked) { option = 1; }

            float range;
            float lowerRange;
            try {
                range = Convert::ToSingle(rangeInput->Text);
                lowerRange = Convert::ToSingle(rangeLowerInput->Text);
                if (range < 0) { range = -1 * range; };
            }
            catch (...) {
                MessageBox::Show("Error - range must be a float");
                return;
            }
            //Perform operations to retrieve division information
            if (calibrationToolStripMenuItem->Checked) {
                b.getRequestedSpectraCalibrationMode(option, range, lowerRangeYes->Checked, lowerRange);
                Rscore->Text = "R^2 score: " + Convert::ToString(b.getRSquared());
            }
            //standard mode operation
            else {
                b.getRequestedSpectraStandardMode(option, range, lowerRangeYes->Checked, lowerRange);

            }
        }
        private: System::Void saveToFile_Click(System::Object^ sender, System::EventArgs^ e) {
            if (calibrationToolStripMenuItem->Checked) {
                int success = b.saveToFileCalibration(nameOfFile->Text, lowerRangeYes->Checked);
                if(success == 0){
                    MessageBox::Show("Error - file was unable to be saved with name " + b.nameOfFile);

                }
                else if (success == 1) {
                    MessageBox::Show("File saved at " + b.directory + b.nameOfFile);
                }
                else{
                    MessageBox::Show("Error - one of the data structures has not been initialized. This
      most often happens if 'Process' has not been clicked. ");
                }
            }
            //standard mode saving.
            else {
                int success = b.saveToFile(nameOfFile->Text, saveSelectedBox->Checked,
      lowerRangeYes->Checked);
                if (success == 0) {
                    MessageBox::Show("Error - file was unable to be saved with name " + b.nameOfFile);
                }

                else if (success == 1) {
                    MessageBox::Show("File saved at " + b.directory + b.nameOfFile);
                }
                else{
```

```
                MessageBox::Show("Error - one of the data structures has not been initialized. This most often happens if 'Process' has not been clicked. ");
            }
        }


    }

    private: System::Void fileSelect_Click(System::Object^ sender, System::EventArgs^ e) {
        handleSelection(1);
    }
    private: System::Void fileSelect_setB_Click(System::Object^ sender, System::EventArgs^ e) {
         handleSelection(2);
         selectFilesLabel_setB->Text = "Files selected.";
    }

    private: System::Void Window_Load(System::Object^ sender, System::EventArgs^ e) {
    }

    private: System::Void standardToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) {
        setCalibrationGroup(true);
        setSetAddedGroup(false);

    }
    private: System::Void calibrationToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) {
        setCalibrationGroup(false);

    }
    private: System::Void howManySubmit_Click(System::Object^ sender, System::EventArgs^ e) {
        int attemptConversion;
        try {
            attemptConversion = Convert::ToInt32(howManySets->Text);
        }
        catch (...) {
            MessageBox::Show("Error - please input an integer value");
            return;
        }
        b.initializeSets(attemptConversion);
        setsOfData->DataSource = nullptr;
        setsOfData->DataSource = b.metadata;
        setSetAddedGroup(true);
        howManySets->Text = "";


    }
    private: System::Void addSetButton_Click(System::Object^ sender, System::EventArgs^ e) {
        if (b.filesToExtract_B == nullptr || b.filesToExtract_B->Count == 0) {
            MessageBox::Show("Error - no files selected");
            return;
        }
        if (b.metadata == nullptr || b.metadata->Count == 0) {
            MessageBox::Show("Error - no empty sets initialized");
            return;
        }
        float concentration;
        try {
            concentration = Convert::ToSingle(analyteBox_setB->Text);
        }
        catch (...) {
            MessageBox::Show("Error - please input a float value for the concentration");
            return;
        }
        int i = setsOfData->SelectedIndex;
        float cutoff;
        try {
            cutoff = Convert::ToSingle(noiseCutoff->Text);
        }
        catch (...) {
            cutoff = -199;
        }

        b.addSetToSets(concentration, i, cutoff);
        setsOfData->DataSource = nullptr;
        setsOfData->DataSource = b.metadata;
        selectFilesLabel_setB->Text = "Select files to process:";
        analyteBox_setB->Text = "";
        setsOfData->SelectedIndex = i >= b.metadata->Count-1 ? i : i + 1;
        return;
    }


    private: void handleSelection(int selectionWindow) {
        float cutoff;
        //if loaded files successfully,
        if (fileOpener->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
            //try converting noiseCutoff to double,
            try {
                if (noiseCutoff->Text == "") {
                    cutoff = -199;
                }
                else {
                    cutoff = Convert::ToSingle(noiseCutoff->Text);
                }
            }
            //if unsuccessful, inform user and continue operation.
            catch (...) {
                MessageBox::Show("Cutoff value is not a float, setting to no cutoff.");
                cutoff = -199;
            }
            //and process the files into the memory.
            if (b.loadFiles(fileOpener->FileNames, cutoff, selectionWindow)) {
                if (selectionWindow == 1) {
                    noOfFiles->Text = "files selected: " + fileOpener->FileNames->Length;
                    if (cutoff == -199) {
                        cutoffLabel->Text = "at no cutoff. ";
                    }
                    else {
                        cutoffLabel->Text = "at cutoff: " + cutoff;
                    }
                }
                else if (selectionWindow == 2) {
                    //noOfFiles_setB->Text = "files selected: " + fileOpener->FileNames->Length;
                    if (cutoff == -199) {
                        //cutoffLabel_setB->Text = "at no cutoff. ";
                    }
                    else {
                        //cutoffLabel_setB->Text = "at cutoff: " + cutoff;
                    }

                }

            }
            //very basic handling for now.
            else {
                noOfFiles->Text = "Error: one of the files is not an .asc file.";
            }
        }
    }
    private: void setCalibrationGroup(bool value) {
        standardToolStripMenuItem->Checked = value;
        calibrationToolStripMenuItem->Checked = !value;
        //single set processing bit
        setAlabel->Enabled = value;
        label7 -> Enabled = value;
        fileSelect->Enabled = value;
        noOfFiles->Enabled = value;
        cutoffLabel->Enabled = value;
        //dual set processing bit
        setBlabel->Enabled = !value;
        howManyLabel->Enabled = !value;
        howManySets->Enabled = !value;
        howManySubmit->Enabled = !value;
        selectFilesLabel_setB->Enabled = !value;
        analyteLabel_setB->Enabled = !value;
        analyteBox_setB->Enabled = !value;
        //fileSelect_setB->Enabled = !value;
        selectFilesLabel_setB->Enabled = !value;
        //addSetButton->Enabled = !value;
        //information
        setsOfData->Enabled = !value;
        setsOfData_label->Enabled = !value;
        //left hand side
        label3->Enabled = value;
        //rangeInput->Enabled = value;
        setNumbersLabel->Enabled = !value;
        saveSelectedBox->Enabled = value;
    }
    private: void setSetAddedGroup(bool value) {
        fileSelect_setB->Enabled = value;
        addSetButton->Enabled = value;

    }
};
}
```

Index