Contents

[Overview 1](#_Toc436136720)

[Flow 1](#_Toc436136721)

[Javascript Contract 2](#_Toc436136722)

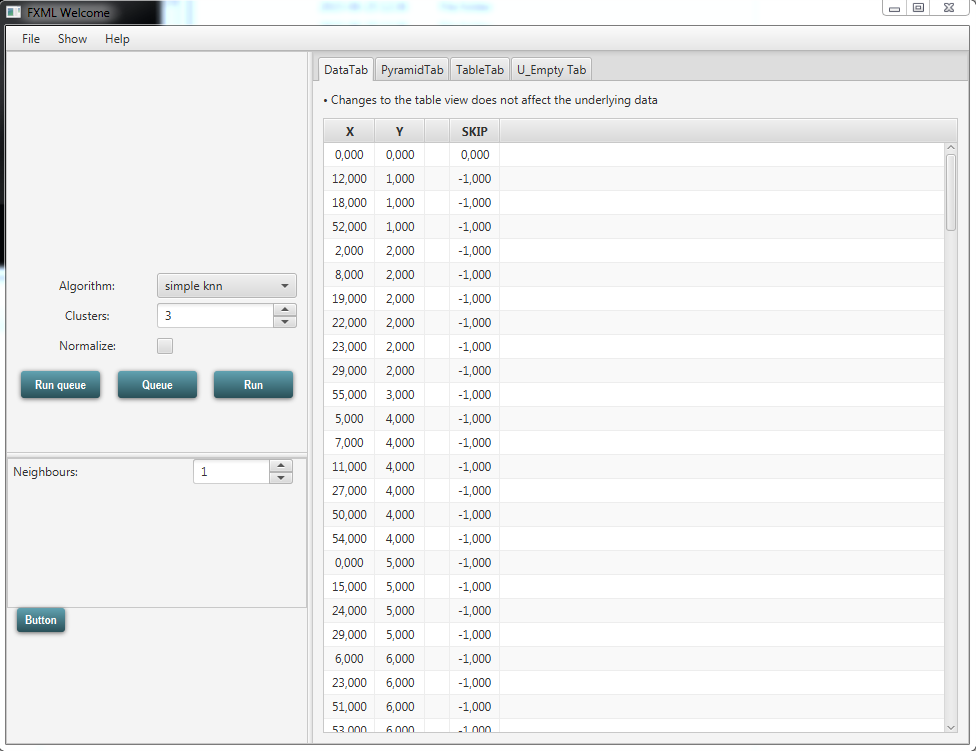
[JFX\_API 3](#_Toc436136723)

[API 3](#_Toc436136724)

## Overview

CluIt (name not registered nor determined, I just needed a name and it was short enough) is a program intended to cluster multidimensional data points and visualize the resulting clusters.

The program is controlled by the user via the main window. The left region controls different variables, such as number of clusters, which javascript to load the clustering method from and also the user-defined variable view. The right region has a number of tabs. The Data-tab shows the currently imported data, the pyramid-tab and table-tab shows results data after clustering, and the U\_Empty-tab is just placeholder.  
The data-tab has a small quirk to it, the space you can see in the image bellow between Y and SKIP indicates which data is reference data and which is clustering data. Data to the right if the gap is only reference data and will not be used in clustering.



## Flow

The program flow is best described by the image in the Overview.pdf but here is short explanation:

The main window updates the Variable Singleton whenever one of the visible variables are changed. When the Import Dialogue imports data from an excel document, a Data object is created and stored in the Variable singleton.

The central part of the program is the Cluster Engine. This class launches the Javascript and supplies it with the necessary variable data from the main window. When the user then presses RUN, the main window calls the Cluster Engine thread and tells it to collect the necessary variables and the data, normalizes it (if the user wishes so) and sends it to the Javascript engine. When the Javascript is finished, it will inform the Clustering Engine that it is done, and the Clustering Engine will create a Result-object from the clustering results.

The Variable Singleton is used to pass different values between different parts of the program. Some variables are observable, and classes can add listeners to these variables. The variable Results is an example. When the Result-variable is changed in the singleton, pyramid-tab and table-tab is notified that result has changed and can thereby update their respective views.

Some communication in the program occurs via an invocation-system, handled by the class MethodMapper. A method can be registered in the MethodMapper by a certain key, the method can later be called by passing said key (and optionally some parameters) to the MethodMapper. This reduces some coupling but does increase the complexity of the program somewhat.

Lastly, a lot of the UI code is quite tricky to explain and understand. If there are any uncertainties or questions, please contact me and I will assist you.

## Javascript Contract

A Javascript that is intended to be read by CluIt has to follow a strict contract, otherwise the program won't work. Instructions for how to write a proper Javascript will follow in this document, as well as explanations of the various API methods that CluIt implements.

First, the Javascript-file has to implement two functions. CluIt will check that these two functions are present in the Javascript. If not, the program will throw an error.

var fields = function() { ... }

var calculate = function() { ... }

The fields() function is called when the script is selected in the scripts-combo box in CluIt's main window. This method should tell CluIt which custom fields this scripts would like to add to the control panel. To add fields, consult the JFX\_API section bellow.

The calculate() function is called when CluIt is asked to perform a clustering calculation using the selected script. The last call in this method \_MUST\_ be API.finish();

Further, two names are reserved in the global namespace: API and JFX\_API. Those two names may not be used by scripts as neither variable names nor function names.

## JFX\_API

Methods in the JFX\_API object should only be called in the fields() function of the Javascript, or in functions called from the fields function.

Methods under this API create custom fields in the main window's control panel. These fields are identified by their name, and the value associated with a certain field can be accessed from a Javascript by calling the API method getFieldValue(String name)

Methods are called by accessing the JFX\_API object. Ex: JFX\_API.createField\_CheckBox("NewBox", true);

\*\*\*Creates an Integer Spinner in the control pane in the main window, associated with the given name\*\*\*

void createField\_IntegerSpinner(String name, int min, int max, int default, int step)

String name Reference name for the Spinner

int min The minimum value for the spinner

int max The maximum value for the spinner

int default The default value for the spinner

int step How many digits does a single click increment/decrement?

\*\*\*Creates a Check Box in the control pane in the main window, associated with the given name\*\*\*

void createField\_CheckBox(String name, boolean default)

String name Reference name for the CheckBox

boolean default The default value of the check box

## API

Methods in the API-object can only be accessed after the calculate() method's been entered. The API object is created just before control is given over to the Javascript.

Methods in this API are access to perform commands in the program from the Javascript. Instead of having the Javascript handle the Java objects, most interaction with Java objects are handled via the API instead. Most methods in the API are quite self-explanatory.

Methods are called by accessing the API object. Ex: API.getAllEntries();

\*\*Creates a new, empty cluster at the parameter coordinates. The return value is the ID of the newly created cluster\*\*

int addCluster(double ... coordinates)

\*\*Adds an entry to a cluster. The return value is -1 if an error occured, or clusterID if successful\*\*

int addEntryToCluster(Entry e, int clusterID)

int API.getTargetNumberOfClusters()

int API.getCurrentNumberOfClusters()

int getNumberOfEntries()

Entry[] getAllEntries()

\*\*The return value is an array of all entries in the space that are not yet added to a cluster\*\*

Entry[] getFreeEntries()

\*\*The return value is an array of all entries in the space that are added to a cluster\*\*

Entry[] getClusteredEntries()

bool isClustered(Entry e)

\*\*The return value is the ID of the cluster this entry is clustered into, or -1 if the entry is free\*\*

int getEntryMembership(Entry e)

\*\*Calculates the Euclidian distance between the two entries\*\*

double getDistance(Entry e1, Entry e2)

\*\*Calculates the Euclidian distance between the two clusters\*\*

API.getDistance(int c1, int c2)

\*\*Shuffles an array of any type\*\*

T[] shuffleArray(T[] t)

\*\*Creates a descending (highest to lowest) Key-PriorityQueue of entries (check CluIt's source)\*\*

KeyPriorityQueue\_Max<Entry> createEntryPriorityQueue\_Max()

\*\*Creates a ascending (lowest to highest) Key-PriorityQueue of entries (check CluIt's source)\*\*

KeyPriorityQueue\_Min<Entry> createEntryPriorityQueue\_Min()

\*\*Fetches the value from a user-defined field. Example: If you created a check box "NewBox" in the fields function, you can fetch the value of that check box by passing the string "NewBox" to this method\*\*

Object getFieldValue(String name)

\*\*Calculates the sum of squared errors for the cluster\*\*

double calcSquaredError(int cluster)

\*\*Misc data is the name of String-Double pairs of data that will be stored in the result-object, and which can later be accessed\*\*

void addMiscData(String ID, double value)