SETLyze Documentation

Release 1.0

GiMaRIS

December 17, 2010

CONTENTS

CHAPTER

ONE

ABOUT SETLYZE

The purpose of SETLyze is to provide the people involved with the SETL project (mostly biologists) an easy and fast way of getting useful information from the data stored in the SETL database. The database was created so certain analysis could be done which would provide more insight in a set of biological questions. The application should be capable of giving answer to those questions by applying specific analysis to the data and providing the user with an understandable overview of the results.

SETLyze is be capable of performing the following analysis:

Analysis 1 "Spot Preference" Determine a species' preference for a specific location on a SETL plate.

Analysis 2.1 "Attraction of Species (intra-specific)" Determine if a specie attracts or repels individuals of its own kind.

Analysis 2.2 "Attraction of Species (inter-specific)" Determine if two different species attract or repel each other.

The following analysis have not been implemented yet:

Analysis 3 "Relation between Species" Determine if there is a relation between two (groups of) species on SETL plates in a location. Plates per location are compared. And instead of looking at different plate spots, only the presence or absence of a specie on a plate is taken into account.

CHAPTER

TWO

DOCUMENTATIONS

2.1 SETLyze User Manual

Welcome to the user manual for SETLyze. This manual is intended for the end user and explains how to use SETLyze.

2.1.1 Introduction to SETLyze

SETLyze is a part of the SETL-project, a fouling community study focussing on marine invasive species. This project is being lead by Arjan Gittenberger. The website describes the SETL-project as follows:

> "Over the last ten years, marine invaders have had a dramatically increasing impact on temperate water ecosystems around the world. Substantial ecological and economical damage has been caused by the introduction of diseases, parasites, predators, invaders outcompeting native species, and species that are a nuisance for public health, tourism, aquaculture or in any other way. In the SETL-project standardized PVC-plates are used to detect these invasive species and other fouling community organisms. The material and methods of the SETL-project were developed by the ANEMOON foundation in cooperation with the Smithsonian Marine Invasions Laboratory of Smithsonian Environmental Research Centre. In this project 14x14 cm PVC-plates are hung 1 meter below the water surface, and refreshed and checked for species at least every three months." — ANEMOON foundation - SETL

Data collected from these SETL plates are being collected in the SETL database. This database contains over 25000 records containing information of over 200 species in different localities throughout the Netherlands. SETLyze is an application which is capable of performing a set of analysis on the data from the SETL database. SETLyze is be capable of performing the following analysis:

Analysis 1 "Spot Preference" Determine a species' preference for a specific location on a SETL plate.

Analysis 2.1 "Attraction of Species (intra-specific)" Determine if a specie attracts or repels individuals of its own kind.

Analysis 2.2 "Attraction of Species (inter-specific)" Determine if two different species attract or repel each other.

The following analysis have not been implemented yet:

Analysis 3 "Relation between Species" Determine if there is a relation between two (groups of) species on SETL plates in a location. Plates per location are compared. And instead of looking at different plate spots, only the presence or absence of a specie on a plate is taken into account.

Data Collection

First let's have a look at how the data for the SETL-project is being collected. When the SETL plates are checked, each plate is first carefully pulled out of the water and then photographed. All this is done following a standard procedure

described on the ANEMOON foundation's website. First an overview photograph is taken of each plate. Then some more detailed photographs are taken of the species that grow on each plate. Indivdual plates are recognized by their tags. The pictures are then carefully analyzed. For each plate the SETL-monitoring form is filled in. For each species the absence or presence, abundance and area cover are filled in. For this, a 5x5 grid is digitally applied over the photograph. For each specie the presence or absence on each of the 25 plate surfaces are filled in and saved to the database.

Figure 2.1: Figure 1. SETL-plate with digitally applied grid

Each record in the database contains a specie ID, a plate ID, and the 25 plate surfaces. The specie ID links to the specie that was found on the plate. The plate ID links to the plate on which that specie was found. The plate ID is also linked to the location where this plate was deployed. The 25 plate surfaces ("spots") are stored in each record as booleans (meaning they can have a value of True or False). The value 1 (True) for a spot means that the specie in question was present on that spot of the plate. The value 0 (False) means that the specie was absent from that spot.

With 25 x 2500 = 625000+ booleans for the presence/absence of species, automatic methods of analyzing this data are required. Hence SETLyze was developed, a tool for analyzing the settlement of species on SETL plates.

Using SETLyze

SETLyze comes with a graphical user interface (GUI). The GUI consists of dialogs which all have a specific task. These dialogs will guide you in performing the set of analysis it provides. Most of SETLyze's dialogs have a Help button. Clicking this Help button should point you to the corresponding dialog description on this page. All dialog descriptions can be found in the *SETLyze dialogs* section of this manual.

Before SETLyze can perform an analysis, it needs access to a data source containing SETL data. Currently just one data source is supported: manually exported CSV files from the Microsoft Access SETL database. This means that the user must first export the tables of the SETL database from Microsoft Access to CSV files. This would result in four CSV files, one for each table. The user is then required to load these files into SETLyze. First follow *the steps to export the SETL data to CSV files*.

You can perform an analysis once you have the four CSV files containing the SETL data. First run SETLyze by (double) clicking the file named `setlyze.pyw`. You should be presented with the *analysis selection dialog*. Select the analysis you want to perform and press OK to begin. A new dialog will be displayed, most likely the *locations selection dialog*.

If this is your first time running SETLyze, the locations selection dialog will show an empty locations list. The list is empty because the data source has not been set yet. To set the data source and load the SETL data, click on the *Change Data Source* button to open the *change data source dialog*. This dialog allows you to load the data from the CSV files you've just created.

Once the data has been loaded, the locations selection dialog will automatically update the list of locations. From here on it's just a matter of following the instruction one the dialogs. Should you need more help, scroll down to the

SETLyze dialogs section for a more extensive description of each dialog. The dialog descriptions are also accessible from SETLyze's dialogs itself by clicking the Help button on a dialog.

2.1.2 Definition List

This part of the user manual describes some terminology often used throughout the application and this manual.

Spot To analyze SETL plates, photographs of the plates are taken. The photographs are then analyzed on the computer by applying a 5x5 grid to the photographs. This divides the SETL plate into 25 equal surface areas (see *figure 1*). Each of the 25 surface areas are called "spots". Species are scored for presence/absence for each of the 25 spots on each SETL plate, and the data is stored in the SETL database in the form of records. So each SETL record in the database contains presence/absence data of one specie for all 25 spots on a SETL plate.

Positive spot Each record in the SETL database contains data for each of the 25 spots on a SETL plate. The spots are stored as booleans, meaning they can have two values; 1 (True) means that the specie was present on that spot, 0 (False) means that the species was absent on that spot. A spot is "positive" if the spot value is 1 or True. Each record can thus have up to 25 positive spots.

2.1.3 SETLyze dialogs

SETLyze comes with a graphical user interface consisting of separate dialogs. The dialogs are described in this section.

Analysis Selection dialog

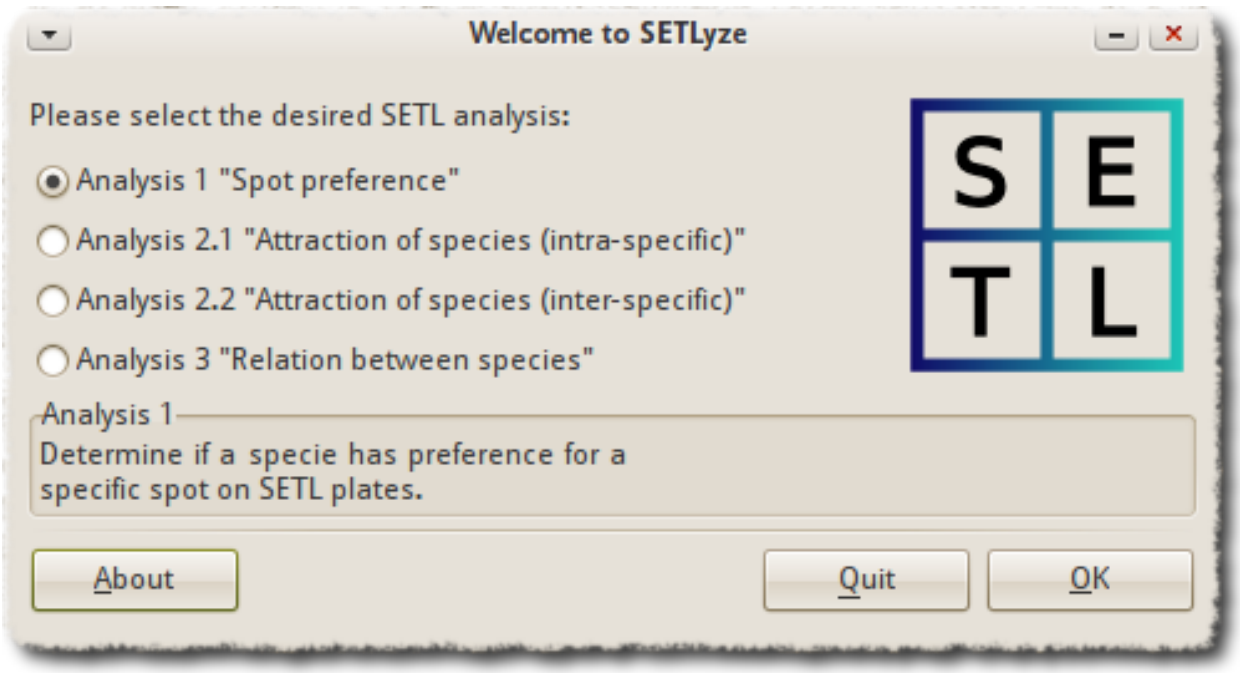


Figure 2.2: Figure 2. Analysis Selection dialog

The analysis selection dialog is the first dialog you see when SETLyze is started. It allows the user to select an analysis to perform on SETL data. The user can select one of the analysis in the list and click on the OK button to start the analysis. Clicking the Quit button closes the application.

After pressing the OK button, two things can happen. If no SETL data was found on the user's computer, SETLyze automatically tries to load SETL localities and species data from the remote SETL database. This requires a direct connection with the SETL database server. A progress dialog is shown while the data is being loaded.

If SETL data is found on the user's computer, a message dialog is shown presenting the user with two options. Option one is to use the SETL data that was previously saved to the user's computer. Option two is to discard the saved data and load data from the remote SETL database.

Clicking the About button shows SETLyze's About dialog. The About dialog shows basic information about SETLyze; its version number, license information, a link to the GiMaRIS website, the application developers, and contact information.

Locations Selection dialog

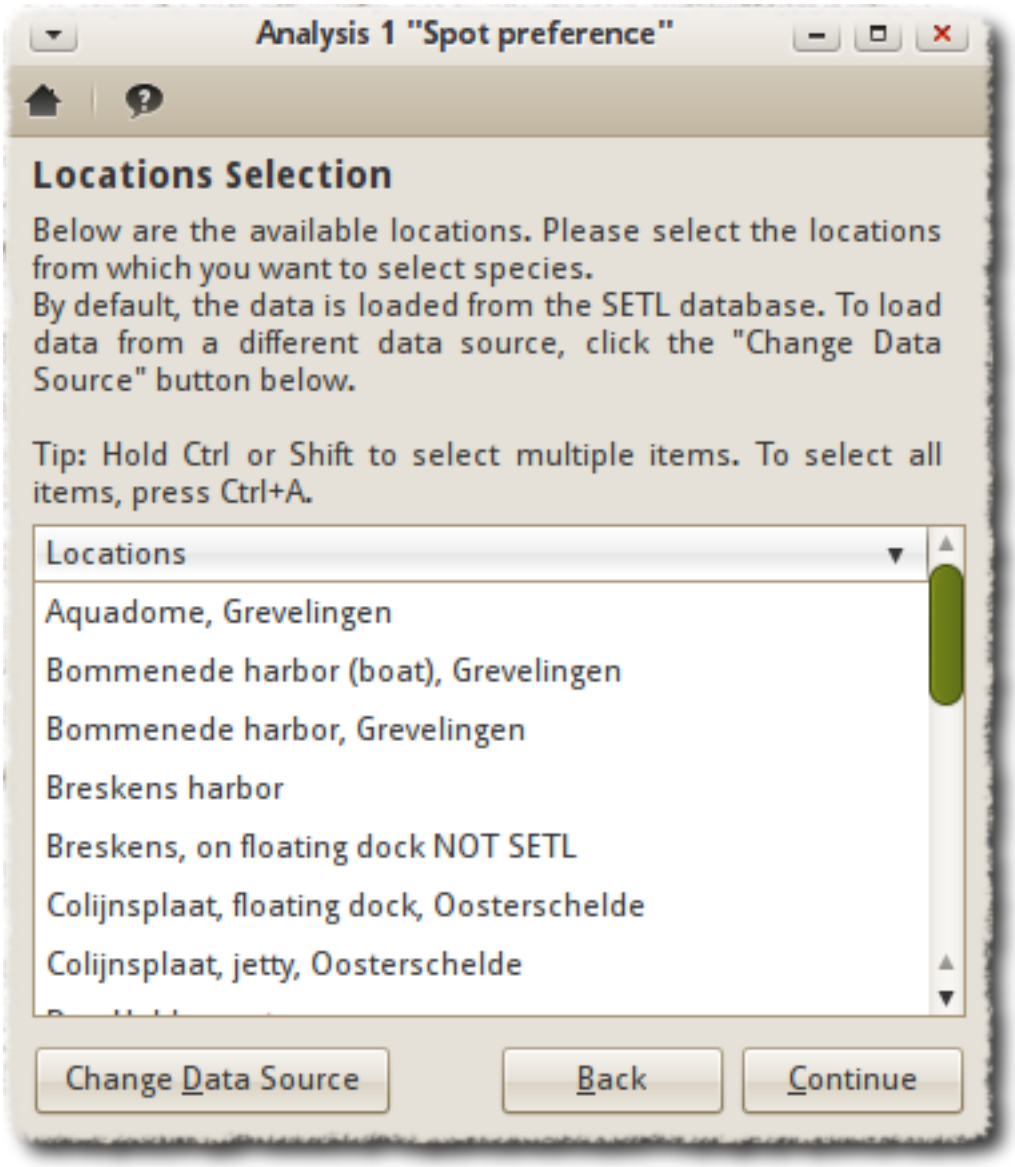


Figure 2.3: Figure 3. Locations Selection dialog

The locations selection dialog shows a list of all SETL localities. This dialog allows you to select locations from which you want to select species. The *species selection dialog* (displayed after clicking the Continue button) will only display the species that were recorded in the selected locations. Subsequently this means that only the SETL records that match both the locations and species selection will be used for the analysis, as each SETL record is bound to a specie and a SETL plate from a specific location.

The *Change Data Source* button opens the *change data source dialog*. This dialog allows you to switch to a different data source. After doing so, the locations selection dialog is automatically updated with the new data.

The Back button allows you to go back to the previous dialog. This can be useful when you want to correct a choice you made in a previous dialog.

The Continue button saves the choices you made in that dialog, closes the dialog, and shows the next dialog.

Making a selection

Just click on one of the locations to select it. To select multiple locations, hold Ctrl or Shift while selecting. To select all locations at once, click on a location and press Ctrl+A.

Species Selection dialog

The species selection dialog shows a list of all SETL species that were found in the selected SETL localities. This dialog allows you to select the species to be included in the analysis. Only the SETL records that match both the the locations and species selection will be used for the analysis.

It is possible to select more than one specie (see *Making a selection*). Selecting more than one specie in a single species selection dialog means that the selected species are threated as one specie for the analysis. However, if the selected analysis requires two or more separate specie selections (i.e. two species are compared), it will display the selection dialog multiple times. In this case, the header of the selection dialog will say "First Species Selection", "Second Species Selection", etc.

The Back button allows you to go back to the previous dialog. This can be useful when you want to correct a choice you made in a previous dialog.

The Continue button saves the choices you made in that dialog, closes the dialog, and shows the next dialog.

Making a selection

Just click on one of the species to select it. To select multiple species, hold Ctrl or Shift while selecting. To select all species at once, click on a specie and press Ctrl+A.

Change Data Source dialog

The change data source dialog allows you to switch to a different data source. Two data sources are possible:

- CSV files exported from the Microsoft Access SETL database. The CSV files need to be exported by Microsoft Access, one file for each of the four tables: SETL_localities, SETL_plates, SETL_records, and SETL_species. The section *Exporting CSV Files from the MS Access database* describes how to export these files.

  After selecting all four CSV files, press the OK button to load all SETL data from these files. A progress dialog is shown while the data is being loaded. Once the data has been loaded, the *locations selection dialog* will be updated with the new data.

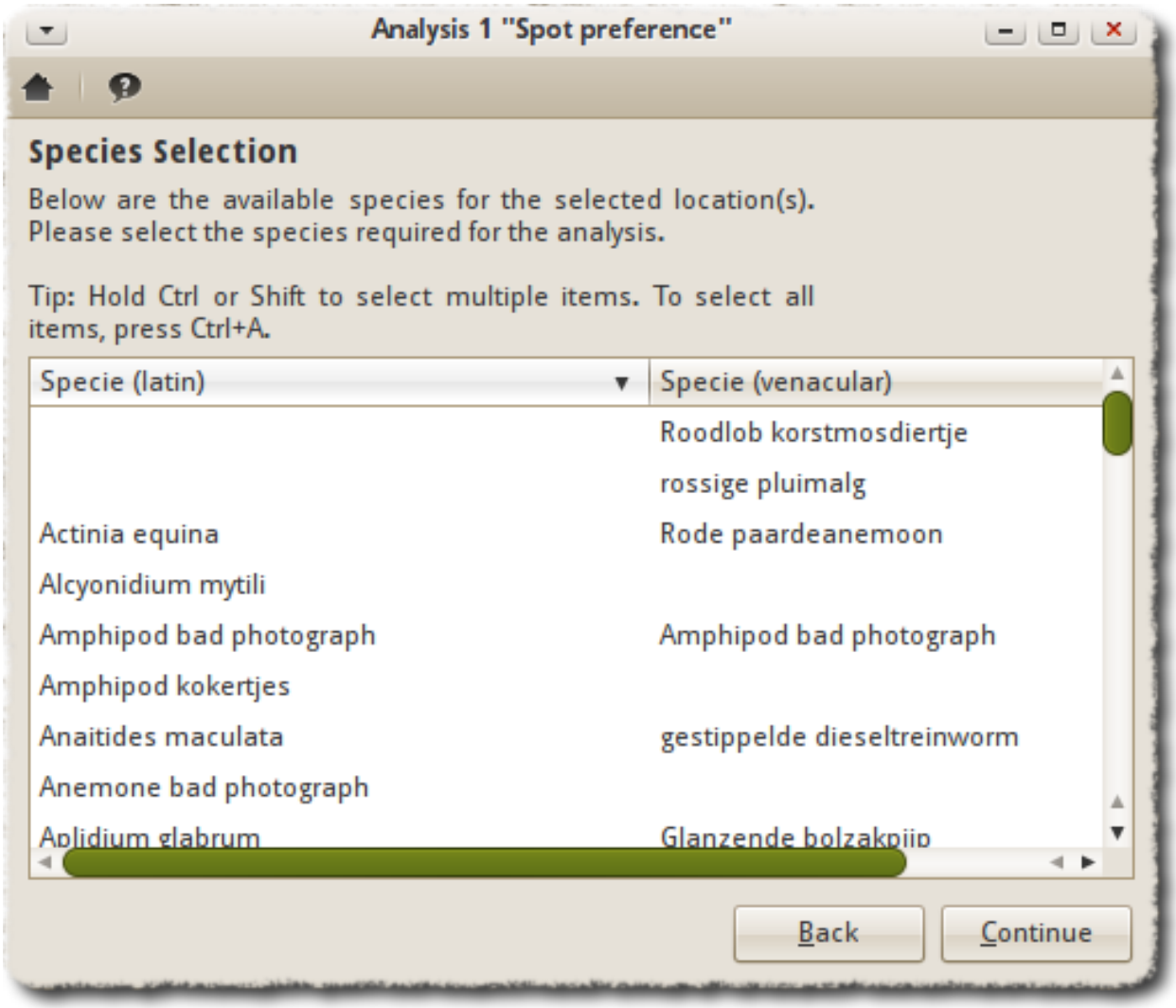


Figure 2.4: Figure 4. Species Selection dialog

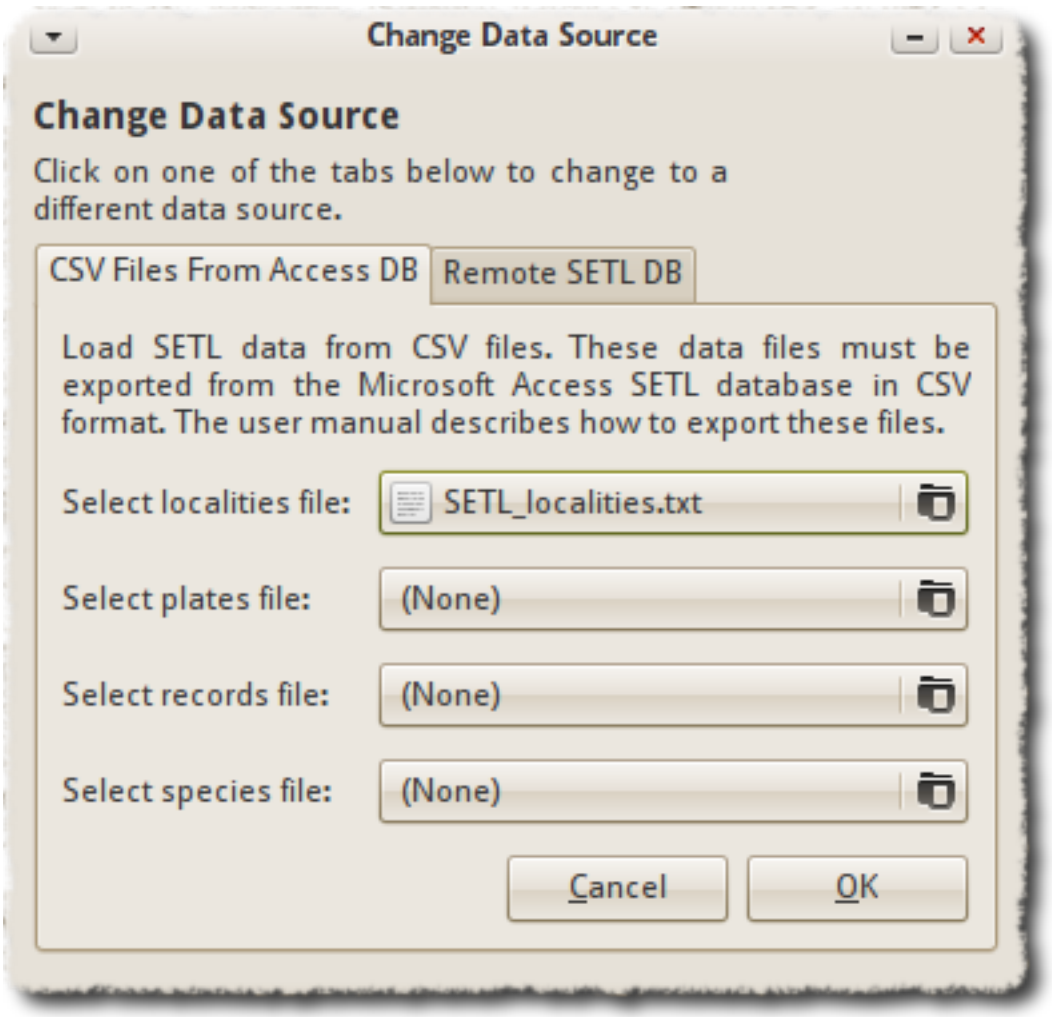


Figure 2.5: Figure 5. Change Data Source dialog

- The remote SETL database. The remote SETL database has not been created yet, so this functionality is not implemented yet. The idea is to move the data from the Microsoft Access database to a PostgreSQL database.

  This dialog should allow you to enter the information needed to connect to the remote database (i.e. the server address and a port number), Pressing the OK button should load the localities and species data. A progress dialog is shown while the data is being loaded. Once the data has been loaded, the *locations selection dialog* will be updated with the new data.

  The plates and records data will not be loaded directly (in contrast to loading data from CSV files). The plates and record data will be loaded when required by the analysis.

Define Plate Areas dialog

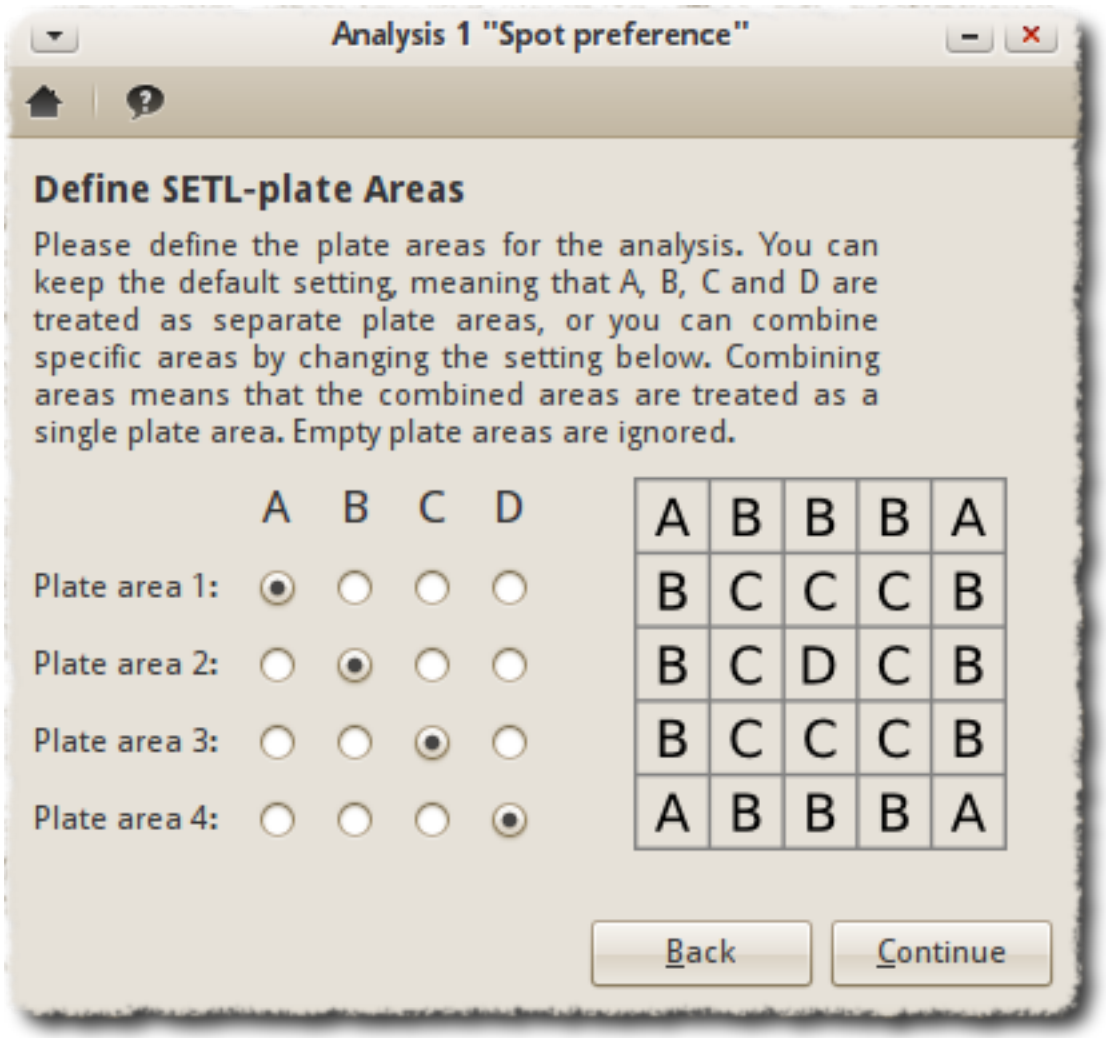


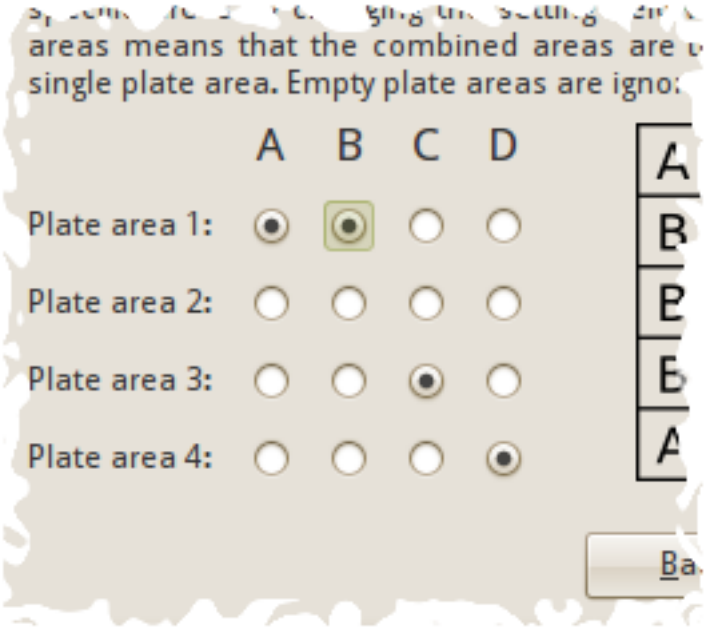
Figure 2.6: Figure 6. Define Plate Areas dialog

This dialog allows you to define the plate areas for analysis 1 “spot preference”. By default, the SETL plate is devided in four plate areas: A, B, C and D. This dialog allows you to combine specific areas by changing the areas selection in the dialog. Combining areas means that the combined areas are treated as a single plate area.

Below is a schematic SETL-plate with a grid. By default the plate is divided in four plate areas (A, B, C and D),

A	B	B	B	A
B	C	C	C	B
B	C	D	C	B
B	C	C	C	B
A	B	B	B	A

But sometimes it's useful to combine plate areas. So if the user decides to combine area A and B, the areas selection would be set like this,



And the resulting plate areas definition would look something like this,

This would result in three plate areas. Analysis 1 would then determine if the selected specie has a preference for either of the three plate areas.

The names of the plate areas (area 1, area 2, ...) do not have a special meaning. It is simply used internally by the application to distinguish between plate areas. These area names are also used in the analysis report to distinguish between the plate areas.

The Back button allows you to go back to the previous dialog. This can be useful when you want to correct a choice you made in a previous dialog.

The Continue button saves the choices you made in the dialog, closes the dialog, and shows the next dialog.

A	A	A	A	A
A	C	C	C	A
A	C	D	C	A
A	C	C	C	A
A	A	A	A	A

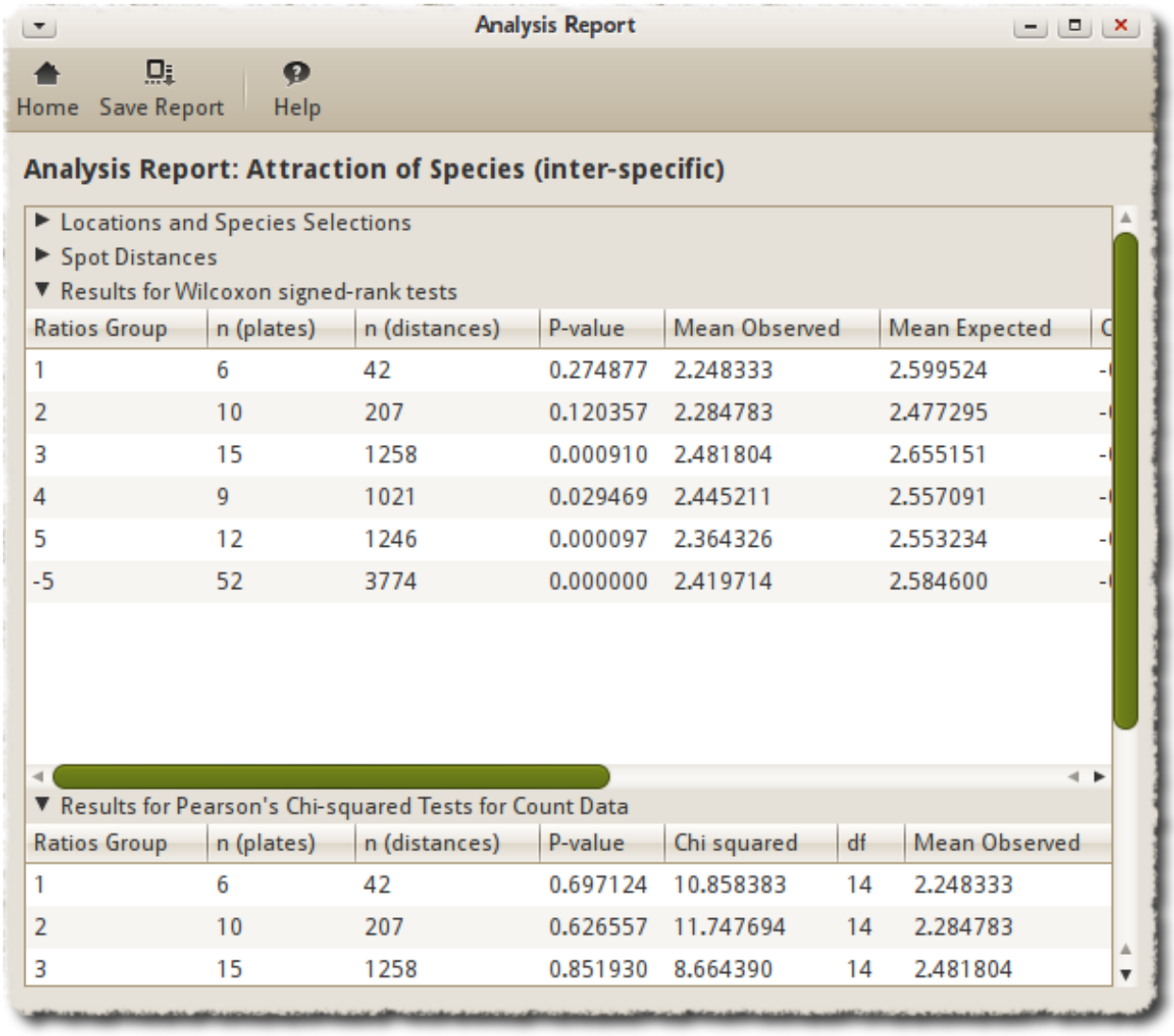


Figure 2.7: Figure 10. Analysis Report dialog

Analysis Report dialog

The analysis report dialog shows the results for the anaylysis. The report is divided into sub sections. Each sub section is described below.

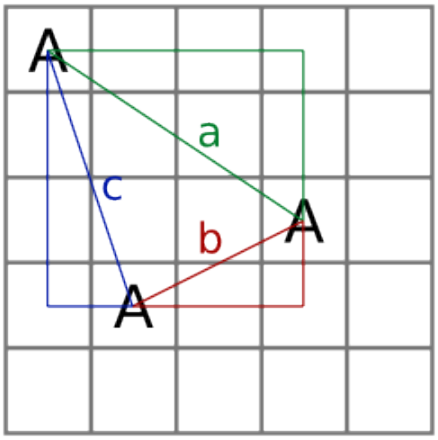
Locations and Species Selections

Displays the locations and species selections. If multiple selections were made, each element is suffixed by a number. For example "Species selection (2)" stands for the second species selection.

Spot Distances

Displays the observed and expected spot distances. The observed spot distances are calculated as follows:

- In the case of analysis 2.1, intra-specific: All possible distances between the spots on each plate are calculated using the Pythagorean theorem. Consider the case of specie A and the following plate:

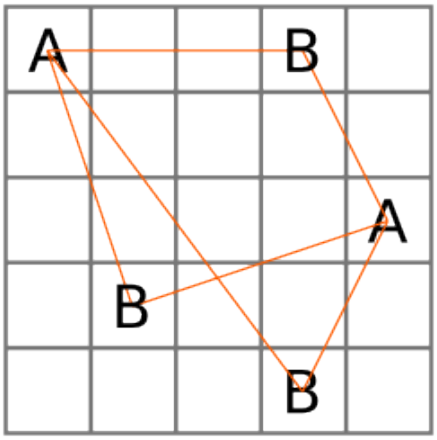


  As you can see from the figure, three positive spots results in three spot distances (a, b and c). This distance from one spot to the next by moving horizontally or vertically defined 1. So for example, the distance for a is calculated by applying Pythagoras' theorem on the green triangle. The distance for a would be $\sqrt{3^2 + 2^2} = 3.61$. This is done for all possible spot distances on each plate. This results in a list of observed spot distances.

- In the case of analysis 2.2 inter-specific: First the plate records are collected that contain both of the selected species. Then all possible spot distances are calculated between the two species. The following figure shows an example with positive spots for two species (A and B) and all possible spot distnaces.

  The distances are calculated the same way as for analysis 2.1.

The expected spot distances are calculated by generating a copy of each plate record matching the species selection. Each copy has the same number of positive spots as its original, except the positive spots are placed randomly at the plates. Then the spot distances are calculates the same way as for the observed spot distances. This mean that the resulting list of expected spot distances has the same length as the observed spot distances.



Results for Wilcoxon signed-rank test

Shows the results for the Wilcoxon signed-rank test.

> "The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test for the case of two related samples or repeated measurements on a single sample. It can be used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed." — Wikipedia - Wilcoxon signed-rank test

Tests showed that spot distances on a SETL plate are not normally distributed (see *figure 13* and *14*), hence the Wilcoxon test was chosen to test if the observed and expected spot distances differ significantly.

Depending on the analysis, the records matching the species selection are first grouped by positive spots number (analysis 2.1) or by ratios group (analysis 2.2).

- Records grouped by number of positive spots: This results in a maximum of 25 groups. Group 1 contains records with just 1 positive spot, group 2 contains records with 2 positive spots, et cetera. however, group 1 and 25 are skipped. Group 1 is excluded because it is not possible to calculate spot distances for records with just one positive spot. Group 25 is excluded because a test on these records will always result in a p-value of 1. This is because the observed spot distances will always be the same as the expected distances. The test is also performed on a group with positive spots number -24. This actually means "up to that number of positive spots". So this test is also performed on records with up to 24 positive spots. Note that records with 1 positive spot will still be excluded.

  So the test is performed 24 times, and each test result is presented on a different row. Each row contains the result of the test for a group.

- Records grouped by ratios groups: In analysis 2.2 we're dealing with two species. For this analysis plate records are matched that contain both species. This means we can get a ratio of positive spots for each SETL plate. Consider figure *fig_spot-distances-inter* which shows positive spots of species A and B. There are two positive spots of one specie, and three positive spots of the other. That makes the ratio for this plate 2:3. The order of the species doesn't matter, so ratio A:B is considered the same as ratio B:A. All records are grouped based on this ratio. We've defined five ratios groups:

  Note:

  comb() A function for generating two-item combinations with replacement from a sequence of digits.

  seq() A function for creating a sequence. Example: $seq(1, 6) = 1, 2, 3, 4, 5$

  Ratios group 1: $comb(seq(1, 6))$ = (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 2), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5), (4, 4), (4, 5), (5, 5)

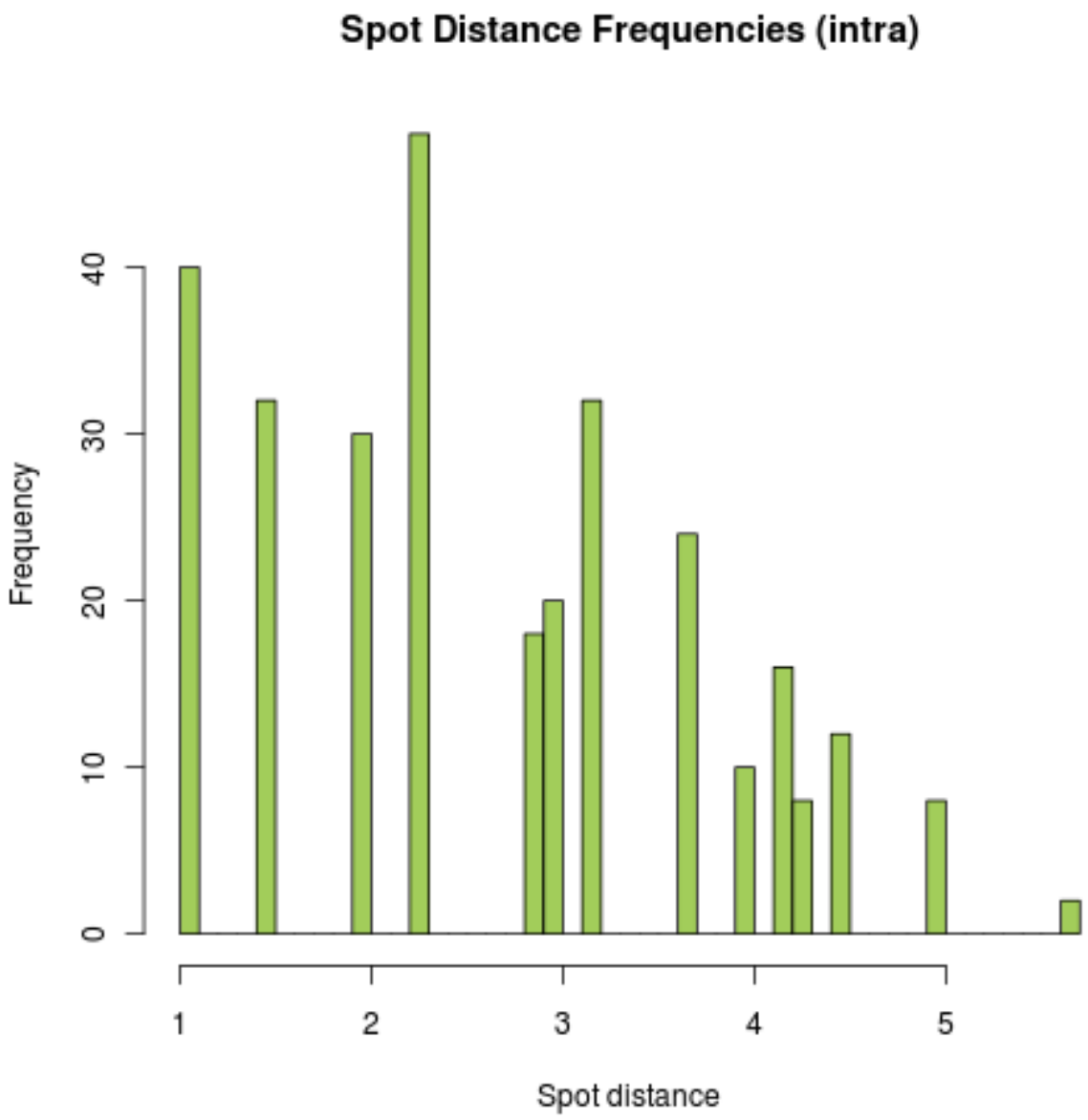


Figure 2.8: Figure 13. Distribution for intra-specific spot distances. The frequencies were obtained by calculating all possible distances between two spots if all 25 spots are covered. The same test was done with different numbers of positive spots randomly placed on a plate with 100.000 repeats. All resulting distributions are very similar to this figure.

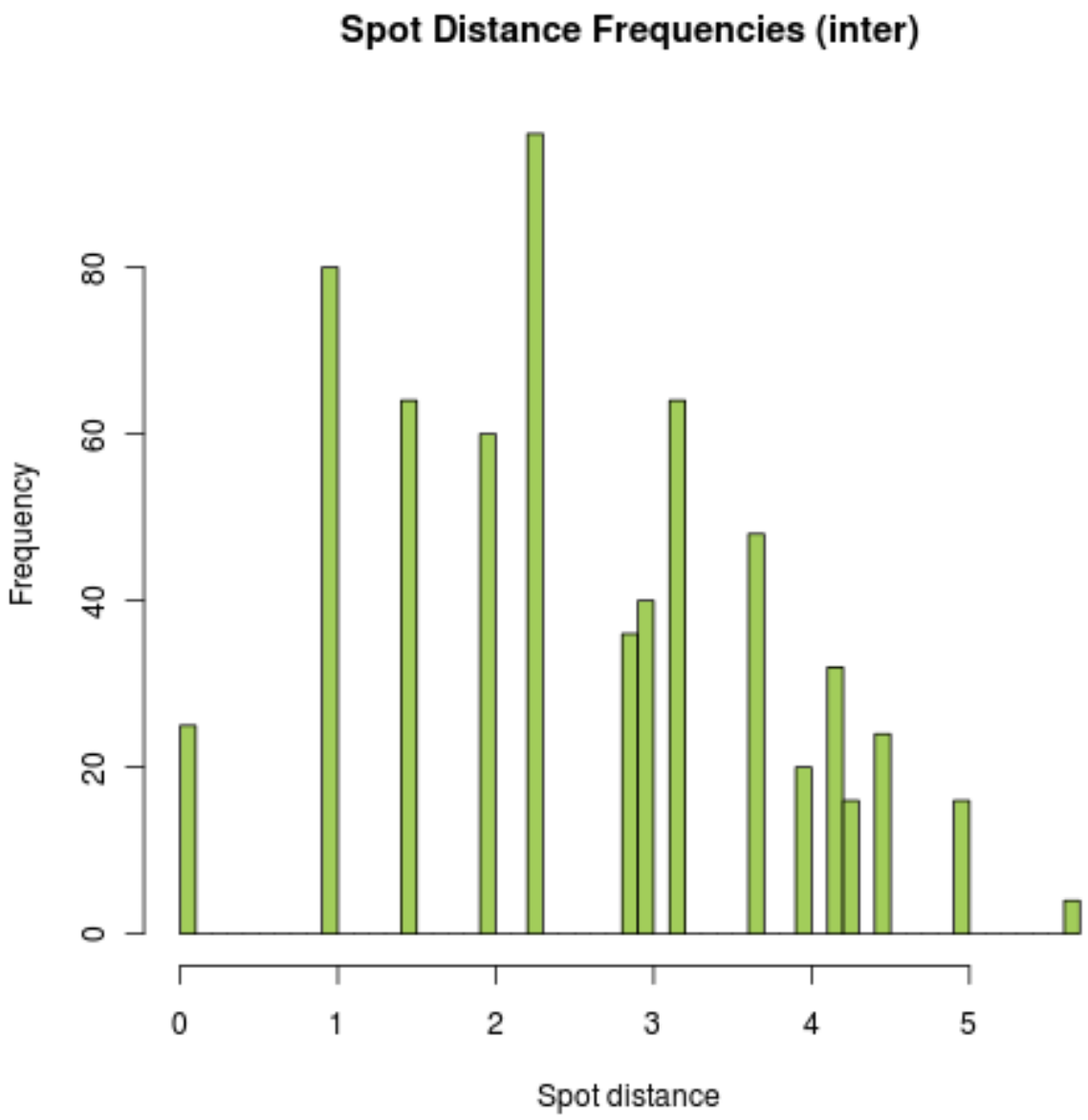


Figure 2.9: Figure 14. Distribution for inter-specific spot distances. The frequencies were obtained by calculating all possible distances between two spots with ratio 25:25 (specie A and B have all 25 spots covered). The same test was done with different positive spots ratios (spots randomly placed on a plate, 100.000 repeats). All resulting distributions are very similar to this figure.

Ratios group 2: $comb(seq(1, 11)) - comb(seq(1, 6))$ = (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 6), (2, 7), (2, 8), (2, 9), (2, 10), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (4, 6), (4, 7), (4, 8), (4, 9), (4, 10), (5, 6), (5, 7), (5, 8), (5, 9), (5, 10), (6, 6), (6, 7), (6, 8), (6, 9), (6, 10), (7, 7), (7, 8), (7, 9), (7, 10), (8, 8), (8, 9), (8, 10), (9, 9), (9, 10), (10, 10)

Ratios group 3: $comb(seq(1, 16)) - comb(seq(1, 11))$ = (1, 11), (1, 12), (1, 13), (1, 14), (1, 15), (2, 11), (2, 12), (2, 13), (2, 14), (2, 15), (3, 11), (3, 12), (3, 13), (3, 14), (3, 15), (4, 11), (4, 12), (4, 13), (4, 14), (4, 15), (5, 11), (5, 12), (5, 13), (5, 14), (5, 15), (6, 11), (6, 12), (6, 13), (6, 14), (6, 15), (7, 11), (7, 12), (7, 13), (7, 14), (7, 15), (8, 11), (8, 12), (8, 13), (8, 14), (8, 15), (9, 11), (9, 12), (9, 13), (9, 14), (9, 15), (10, 11), (10, 12), (10, 13), (10, 14), (10, 15), (11, 11), (11, 12), (11, 13), (11, 14), (11, 15), (12, 12), (12, 13), (12, 14), (12, 15), (13, 13), (13, 14), (13, 15), (14, 14), (14, 15), (15, 15)

Ratios group 4: $comb(seq(1, 21)) - comb(seq(1, 16))$ = (1, 16), (1, 17), (1, 18), (1, 19), (1, 20), (2, 16), (2, 17), (2, 18), (2, 19), (2, 20), (3, 16), (3, 17), (3, 18), (3, 19), (3, 20), (4, 16), (4, 17), (4, 18), (4, 19), (4, 20), (5, 16), (5, 17), (5, 18), (5, 19), (5, 20), (6, 16), (6, 17), (6, 18), (6, 19), (6, 20), (7, 16), (7, 17), (7, 18), (7, 19), (7, 20), (8, 16), (8, 17), (8, 18), (8, 19), (8, 20), (9, 16), (9, 17), (9, 18), (9, 19), (9, 20), (10, 16), (10, 17), (10, 18), (10, 19), (10, 20), (11, 16), (11, 17), (11, 18), (11, 19), (11, 20), (12, 16), (12, 17), (12, 18), (12, 19), (12, 20), (13, 16), (13, 17), (13, 18), (13, 19), (13, 20), (14, 16), (14, 17), (14, 18), (14, 19), (14, 20), (15, 16), (15, 17), (15, 18), (15, 19), (15, 20), (16, 16), (16, 17), (16, 18), (16, 19), (16, 20), (17, 17), (17, 18), (17, 19), (17, 20), (18, 18), (18, 19), (18, 20), (19, 19), (19, 20), (20, 20)

Ratios group 5: $comb(seq(1, 26)) - comb(seq(1, 21)) - comb(25)$ = (1, 21), (1, 22), (1, 23), (1, 24), (1, 25), (2, 21), (2, 22), (2, 23), (2, 24), (2, 25), (3, 21), (3, 22), (3, 23), (3, 24), (3, 25), (4, 21), (4, 22), (4, 23), (4, 24), (4, 25), (5, 21), (5, 22), (5, 23), (5, 24), (5, 25), (6, 21), (6, 22), (6, 23), (6, 24), (6, 25), (7, 21), (7, 22), (7, 23), (7, 24), (7, 25), (8, 21), (8, 22), (8, 23), (8, 24), (8, 25), (9, 21), (9, 22), (9, 23), (9, 24), (9, 25), (10, 21), (10, 22), (10, 23), (10, 24), (10, 25), (11, 21), (11, 22), (11, 23), (11, 24), (11, 25), (12, 21), (12, 22), (12, 23), (12, 24), (12, 25), (13, 21), (13, 22), (13, 23), (13, 24), (13, 25), (14, 21), (14, 22), (14, 23), (14, 24), (14, 25), (15, 21), (15, 22), (15, 23), (15, 24), (15, 25), (16, 21), (16, 22), (16, 23), (16, 24), (16, 25), (17, 21), (17, 22), (17, 23), (17, 24), (17, 25), (18, 21), (18, 22), (18, 23), (18, 24), (18, 25), (19, 21), (19, 22), (19, 23), (19, 24), (19, 25), (20, 21), (20, 22), (20, 23), (20, 24), (20, 25), (21, 21), (21, 22), (21, 23), (21, 24), (21, 25), (22, 22), (22, 23), (22, 24), (22, 25), (23, 23), (23, 24), (23, 25), (24, 24), (24, 25)

Ratio 25:25 is removed from this group because the p-value for records with that ratio would always be 1.

To get back to our example, a records from a plate with ratio 2:3 would be grouped in ratios group 1. The Wilcoxon test is also performed on ratios group "-5". This group includes ratios from all 5 groups (still excluding ratio 25:25).

Each row for the results of the Wicoxon test contains the results of a single test on a spots/ratios group. Each row can have the following elements:

Positive Spots A number representing the number of positive spots. For this test only records matching that number of positive spots were used.

Ratios Group A number representing the ratios group. For this test only records grouped in that ratios group were used.

n (plates) The number of plates that match the number of positive spots.

n (distances) The number of spot distances derived from the records matching the positive spots number.

P-value The P-value for the test.

Mean Observed The mean of the observed spot distances.

Mean Expected The mean of the expected spot distances.

Conf. interval start The start of the confidence interval for the test.

Conf. interval end The end of the confidence interval for the test.

Remarks A summary of the results. Shows whether the p-value is significant, and if so, how significant and decides based on the means if the species attract or repel. Attraction: observed mean < expected mean. Repulsion: observed mean > expected mean.

Results for Pearson's Chi-squared Test for Count Data

Shows the results for Pearson's Chi-squared Test for Count Data.

> "Pearson's chi-square ($\chi 2$) test is the best-known of several chi-square tests. It tests a null hypothesis stating that the frequency distribution of certain events observed in a sample is consistent with a particular theoretical distribution." — Wikipedia - Pearson's Chi-squared Test

The observed values are the frequencies of the observed spot distances. The expected values are calculated with the formula $e(d) = N * p$ where *N* is the total number of observed distances and *p* is the probability for spot distance *d*. The probability *d* has been calculated for each spot distance. The probabilities for intra-specific spot distances are from the model of *figure 13* and the probabilities for inter-specific distances are from the model of *figure 14*. The probabilities have been hard coded into the application:

```
# The probability for each spot distance on a 5x5 SETL plate
# (intra-specific).
# Format of the dictionary: {distance: probability, ...}
SPOT_DIST_TO_PROB_INTRA = {
    1: 40/300.0,
    1.41: 32/300.0,
    2: 30/300.0,
    2.24: 48/300.0,
    2.83: 18/300.0,
    3: 20/300.0,
    3.16: 32/300.0,
    3.61: 24/300.0,
    4: 10/300.0,
    4.12: 16/300.0,
    4.24: 8/300.0,
    4.47: 12/300.0,
    5: 8/300.0,
    5.66: 2/300.0,
    }

# The probability for each spot distance on a 5x5 SETL plate
# (inter-specific).
# Format of the dictionary: {distance: probability, ...}
SPOT_DIST_TO_PROB_INTER = {
    0: 25/625.0,
    1: 80/625.0,
    1.41: 64/625.0,
    2: 60/625.0,
    2.24: 96/625.0,
    2.83: 36/625.0,
    3: 40/625.0,
    3.16: 64/625.0,
    3.61: 48/625.0,
    4: 20/625.0,
    4.12: 32/625.0,
    4.24: 16/625.0,
    4.47: 24/625.0,
    5: 16/625.0,
    5.66: 4/625.0,
```

}

Each row for the results of the Chi-squared tests contains the results of a single test on a spots/ratios group. Each row can have the following elements:

Positive Spots A number representing the number of positive spots. For this test only records matching that number of positive spots were used.

Ratios Group A number representing the ratios group. For this test only records grouped in that ratios group were used.

n (plates) The number of plates that match the number of positive spots.

n (distances) The number of spot distances derived from the records matching the positive spots number.

P-value The P-value for the test.

Chi squared The value the chi-squared test statistic.

df The degrees of freedom of the approximate chi-squared distribution of the test statistic.

Mean Observed The mean of the observed spot distances.

Mean Expected The mean of the expected spot distances.

Remarks A summary of the results. Shows whether the p-value is significant, and if so, how significant and decides based on the means if the species attract or repel. Attraction: observed mean < expected mean. Repulsion: observed mean > expected mean.

Plate Areas Definition

Describes the definition of the plate areas set with the *define plate areas dialog*. Read the description for that dialog to get the meaning of the letters A, B, C and D.

Species Total per Plate Area

Observed Totals How many times the selected specie was found present in each of the plate areas.

Expected Totals The expected totals for the selected specie.

Exporting SETL data to CSV files

This section describes how to export the SETL data from the Microsoft Access database to CSV files.

1. Open the SETL database file (*.mdb) in Microsoft Access. You'll see four tables in the left column: SETL_localities, SETL_plates, SETL_records and SETL_species.
2. To export a table, right-click on it to open the drop menu. From the menu select Export > Text file. Then give the filename of the output file. Make sure to include the table name in the filename (e.g. setl_localities.csv for the "SETL_localities" table). Uncheck all other options and press OK.
3. In the next dialog that appears select the option that separates fields with a character. The separator character must be a semicolon (";"). If it's not, change it by clicking the Advanced button. Then click Finish to export the data to a CSV file.
4. Repeat steps 2 and 3 for all tables.

2.2 SETLyze Developer Guide

Welcome to the Developer Guide for SETLyze. This document describes the SETLyze internals. It's meant for people who are involved in the development process of SETLyze. It should be easy for a new developer to pick up where the last SETLyze developer left off. The purpose of this guide is to give the new developer full understanding of SETLyze's internals, its programming style, what's unfinished, et cetera.

2.2.1 Getting Started

Navigating the SETLyze folder

Some of the key files in SETLyze's root folder are:

doc This folder contains the documentation for SETLyze. This includes the User Manual and the Developer Guide.

setlyze This is the main code base for SETLyze. This package folder contains all the modules for SETLyze. This is the folder where you'll be editing most Python source files for SETLyze.

COPYING This text file contains the license for SETLyze. SETLyze is released under the GNU General Public License version 3.

INSTALL Text file with installation instrutions for SETLyze.

setlyze.pyw This is the executable for SETLyze. This is what you'll run to start SETLyze.

setup.py Installs SETLyze system-wide or to your home directory. This script is used to install SETLyze on your machine. For installation instructions, read the INSTALL file.

Technical Design

SETLyze comes with a Technical Design; a visual representation of SETLyze's design parts (functions/classes/GUI's) interconnected by arrows representing the application's functions and work flow. All design parts have a number. The same numbers can be found in the application's source-code. This means that the different design parts of the Technical Design can be easily linked to the corresponding source-code.

The Technical Design provides an easy to understand overview of the application, but is also of great value to developers. It is much easier to understand how the application works by looking at its Technical Design. If the developer is interested in a specific part of the source-code, he or she can easily navigate to that part of the source-code by the reference numbers used in the Technical Design.

This documentation allows for easy navigation through SETLyze's code base. To start, use the links below that will guide you to the different design parts present in the Technical Design.

Design Parts

SETLyze Design Parts

Design Part #	Reference
1.0	`__main__`
1.1	`__main__.main()`
1.2	`setlyze.database.MakeLocalDB`
1.3	`setlyze.analysis.spot_preference`

Continued on next page

Table 2.1 – continued from previous page

1.3.1	setlyze.analysis.spot_preference.Begin
1.3.2	setlyze.analysis.spot_preference.Start
1.4	setlyze.analysis.attraction_intra
1.4.1	setlyze.analysis.attraction_intra.Begin
1.4.2	setlyze.analysis.attraction_intra.Start
1.5	setlyze.analysis.attraction_inter
1.5.1	setlyze.analysis.attraction_inter.Begin
1.5.2	setlyze.analysis.attraction_inter.Start
1.6	setlyze.analysis.relations
1.6.1	setlyze.analysis.relations.Begin
1.6.2	setlyze.analysis.relations.Start
1.7	setlyze.gui.SelectLocations.save_selection()
1.8	setlyze.gui.SelectSpecies.save_selection()
1.11	setlyze.gui.SelectionWindow.on_select_data_files()
1.12	setlyze.std.ReportGenerator
1.13	setlyze.analysis.spot_preference.Start.generate_report()
1.14	setlyze.analysis.attraction_intra.Start.generate_report()
1.15	setlyze.analysis.attraction_inter.Start.generate_report()
1.16	setlyze.analysis.relations.Start.generate_report()
1.17	setlyze.std.ReportReader.save_report()
1.19.1	setlyze.database.AccessLocalDB.set_species_spots()
1.19.2	setlyze.database.AccessRemoteDB.set_species_spots()
1.20	setlyze.database.AccessDBGeneric.make_plates_unique()
1.21	setlyze.database.AccessDBGeneric.remove_single_spot_plates()
1.22	setlyze.analysis.attraction_intra.Start.calculate_distances_intra()
1.23	setlyze.analysis.attraction_intra.Start.calculate_distances_intra_expected()
1.24	setlyze.analysis.attraction_intra.Start.calculate_significance()
1.27	setlyze.analysis.attraction_inter.Start.calculate_distances_inter()
1.28	setlyze.database.AccessLocalDB
1.29	setlyze.database.AccessRemoteDB
1.30	setlyze.std.ReportGenerator
1.31	setlyze.database.MakeLocalDB.run()
1.32	setlyze.database.MakeLocalDB.insert_from_csv()
1.33	setlyze.database.MakeLocalDB.insert_from_db()
1.34	setlyze.database.MakeLocalDB.insert_localities_from_csv()
1.35	setlyze.database.MakeLocalDB.insert_species_from_csv()
1.36	setlyze.database.MakeLocalDB.insert_plates_from_csv()
1.37	setlyze.database.MakeLocalDB.insert_records_from_csv()
1.38	setlyze.database.MakeLocalDB.create_new_db()
1.39	setlyze.gui.SelectionWindow.update_tree()
1.41.1	setlyze.database.AccessLocalDB.get_record_ids()
1.41.2	setlyze.database.AccessRemoteDB.get_record_ids()
1.42	setlyze.gui.SelectLocations.create_model()
1.43	setlyze.gui.SelectSpecies.create_model()
1.44	setlyze.gui.SelectionWindow.on_continue()
1.45	setlyze.gui.SelectLocations.on_back()
1.46	setlyze.gui.SelectSpecies.on_back()
1.47	setlyze.database.MakeLocalDB.fill_distance_table()
1.48	setlyze.std.ReportGenerator
1.49	setlyze.std.ReportReader
1.50	setlyze.std.ReportGenerator.set_location_selections()
1.51	setlyze.std.ReportGenerator.set_specie_selections()

Continued on next page

Table 2.1 – continued from previous page

1.52	setlyze.std.ReportGenerator.set_spot_distances_observed()
1.53	setlyze.std.ReportGenerator.set_spot_distances_expected()
1.54	setlyze.std.ReportGenerator.set_spot_areas_definition()
1.55	setlyze.std.ReportGenerator.set_area_totals_observed()
1.56	setlyze.std.ReportGenerator.set_area_totals_expected()
1.57	setlyze.config.ConfigManager
1.58	setlyze.analysis.spot_preference.Start.run()
1.59	setlyze.analysis.attraction_intra.Start.run()
1.60	setlyze.analysis.attraction_inter.Start.run()
1.61	setlyze.analysis.relations.Start.run()
1.62	setlyze.analysis.spot_preference.Start.get_areas_totals_observed()
1.63	setlyze.analysis.spot_preference.Start.get_areas_totals_expected()
1.64	setlyze.analysis.spot_preference.Start.chi_square_tester()
1.66	setlyze.database.MakeLocalDB.insert_localities_from_db()
1.67	setlyze.database.MakeLocalDB.insert_species_from_db()
1.68.1	setlyze.analysis.spot_preference.Begin.on_display_report()
1.68.2	setlyze.analysis.attraction_intra.Begin.on_display_report()
1.68.3	setlyze.analysis.attraction_inter.Begin.on_display_report()
1.68.4	setlyze.analysis.attraction.Begin.on_display_report()
1.69	setlyze.analysis.attraction_inter.Start.calculate_distances_inter_expected()
1.70	setlyze.std.ReportGenerator.set_statistics_normality()
1.71	setlyze.std.ReportGenerator.set_statistics_significance()
1.72	setlyze.std.ReportGenerator.set_analysis()
1.73	setlyze.database.AccessDBGeneric.fill_plate_spot_totals_table()
1.74	setlyze.analysis.attraction_inter.Start.calculate_significance()
1.75	setlyze.database.MakeLocalDB.create_table_info()
1.76	setlyze.database.MakeLocalDB.create_table_localities()
1.77	setlyze.database.MakeLocalDB.create_table_species()
1.78	setlyze.database.MakeLocalDB.create_table_plates()
1.79	setlyze.database.MakeLocalDB.create_table_records()
1.80	setlyze.database.MakeLocalDB.create_table_species_spots_1()
1.81	setlyze.database.MakeLocalDB.create_table_species_spots_2()
1.82	setlyze.database.MakeLocalDB.create_table_spot_distances()
1.83	setlyze.database.MakeLocalDB.create_table_spot_distances_observed()
1.84	setlyze.database.MakeLocalDB.create_table_spot_distances_expected()
1.85	setlyze.database.MakeLocalDB.create_table_plate_spot_totals()
1.86	SelectAnalysis
1.87	setlyze.gui.SelectLocations
1.88	setlyze.gui.SelectSpecies
1.89	setlyze.gui.DisplayReport
1.90	setlyze.gui.ChangeDataSource
1.91	setlyze.gui.DefinePlateAreas
1.92	setlyze.gui.ProgressDialog

1.x Modules, Classes & Functions

2.x Data Storage Places

Design Parts: Data The design parts in this overview describes all technical design parts representing data used in SETLyze. This includes database tables, application variables, and data files.

2.x Data Storage Places

2.0 Table `setl_records` in the SETL database. The SETL database can be either the MS Access database or the PostgreSQL database. This table contains the SETL records.

PostgreSQL query:

```
CREATE TABLE setl_records
(
    rec_id              SERIAL,
    rec_pla_id          INTEGER NOT NULL,
    rec_spe_id          INTEGER NOT NULL,
    rec_unknown         BOOLEAN,
    rec_o               BOOLEAN,
    rec_r               BOOLEAN,
    rec_c               BOOLEAN,
    rec_a               BOOLEAN,
    rec_e               BOOLEAN,
    rec_sur_unknown     BOOLEAN,
    rec_sur1            BOOLEAN,
    rec_sur2            BOOLEAN,
    rec_sur3            BOOLEAN,
    rec_sur4            BOOLEAN,
    rec_sur5            BOOLEAN,
    rec_sur6            BOOLEAN,
    rec_sur7            BOOLEAN,
    rec_sur8            BOOLEAN,
    rec_sur9            BOOLEAN,
    rec_sur10           BOOLEAN,
    rec_sur11           BOOLEAN,
    rec_sur12           BOOLEAN,
    rec_sur13           BOOLEAN,
    rec_sur14           BOOLEAN,
    rec_sur15           BOOLEAN,
    rec_sur16           BOOLEAN,
    rec_sur17           BOOLEAN,
    rec_sur18           BOOLEAN,
    rec_sur19           BOOLEAN,
    rec_sur20           BOOLEAN,
    rec_sur21           BOOLEAN,
    rec_sur22           BOOLEAN,
    rec_sur23           BOOLEAN,
    rec_sur24           BOOLEAN,
    rec_sur25           BOOLEAN,
    rec_1st             BOOLEAN,
    rec_2nd             BOOLEAN,
    rec_v               BOOLEAN,
    rec_photo_nrs       VARCHAR(100),
    rec_remarks         VARCHAR(100),

    CONSTRAINT rec_id_pk PRIMARY KEY (rec_id),
    CONSTRAINT rec_pla_id_fk FOREIGN KEY (rec_pla_id)
        REFERENCES setl_plates (pla_id)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT rec_spe_id_fk FOREIGN KEY (rec_spe_id)
        REFERENCES setl_species (spe_id)
        ON DELETE NO ACTION
```

```
        ON UPDATE NO ACTION
);
```

2.1 Table `setl_species` in the SETL database. The SETL database can be either the MS Access database or the PostgreSQL database. This table contains the SETL specie records.

PostgreSQL query:

```
CREATE TABLE setl_species
(
    spe_id                  SERIAL,
    spe_name_venacular      VARCHAR(100) UNIQUE,
    spe_name_latin          VARCHAR(100) NOT NULL UNIQUE,
    spe_invasive_in_nl      BOOLEAN,
    spe_description         VARCHAR(300),
    spe_remarks             VARCHAR(160),
    spe_picture             OID,

    CONSTRAINT spe_id_pk PRIMARY KEY (spe_id)
);
```

2.2 Table `setl_localities` in the SETL database. The SETL database can be either the MS Access database or the PostgreSQL database. This table contains the SETL locality records.

PostgreSQL query:

```
CREATE TABLE setl_localities
(
    loc_id              SERIAL,
    loc_name            VARCHAR(100) NOT NULL UNIQUE,
    loc_nr              INTEGER,
    loc_coordinates     VARCHAR(100),
    loc_description     VARCHAR(300),

    CONSTRAINT loc_id_pk PRIMARY KEY (loc_id)
);
```

2.3 Table `species` in the local SQLite database. This table is automatically filled from *2.1* when the user starts a SETLyze analysis.

2.3.1 Same as *2.3*, but filled from *2.1*.

2.3.2 Same as *2.3*, but filled from *2.19*.

SQLite query:

```
CREATE TABLE species
(
    spe_id INTEGER PRIMARY KEY,
    spe_name_venacular VARCHAR,
    spe_name_latin VARCHAR,
    spe_invasive_in_nl INTEGER,
    spe_description VARCHAR,
    spe_remarks VARCHAR
);
```

2.4 Table `localities` in the local SQLite database. This table is automatically filled from *2.2* when the user starts a SETLyze analysis.

SQLite query:

```
CREATE TABLE localities
(
    loc_id INTEGER PRIMARY KEY,
    loc_name VARCHAR,
    loc_nr VARCHAR,
    loc_coordinates VARCHAR,
    loc_description VARCHAR
);
```

2.4.1 Same as *2.4*, but filled from *2.2*.

2.4.2 Same as *2.4*, but filled from *2.18*.

2.5 Table `records` in the local SQLite database. This table is only filled if the user selected CSV files to import SETL data from. By default this table is empty, and the records data from *2.0* is used.

SQLite query:

```
CREATE TABLE records
(
    rec_id INTEGER PRIMARY KEY,
    rec_pla_id INTEGER,
    rec_spe_id INTEGER,
    rec_unknown INTEGER,
    rec_o INTEGER,
    rec_r INTEGER,
    rec_c INTEGER,
    rec_a INTEGER,
    rec_e INTEGER,
    rec_sur_unknown INTEGER,
    rec_sur1 INTEGER,
    rec_sur2 INTEGER,
    rec_sur3 INTEGER,
    rec_sur4 INTEGER,
    rec_sur5 INTEGER,
    rec_sur6 INTEGER,
    rec_sur7 INTEGER,
    rec_sur8 INTEGER,
    rec_sur9 INTEGER,
    rec_sur10 INTEGER,
    rec_sur11 INTEGER,
    rec_sur12 INTEGER,
    rec_sur13 INTEGER,
    rec_sur14 INTEGER,
    rec_sur15 INTEGER,
    rec_sur16 INTEGER,
    rec_sur17 INTEGER,
    rec_sur18 INTEGER,
    rec_sur19 INTEGER,
    rec_sur20 INTEGER,
    rec_sur21 INTEGER,
    rec_sur22 INTEGER,
```

```
    rec_sur23 INTEGER,
    rec_sur24 INTEGER,
    rec_sur25 INTEGER,
    rec_1st INTEGER,
    rec_2nd INTEGER,
    rec_v INTEGER
);
```

2.6 A list `[<selection-1>,<selection-2>]` for storing a maximum of two location selections. `<selection-1>` and `<selection-2>` are lists of integers representing location IDs. These IDs are the same as the IDs in column `loc_id` in *2.2* and *2.4*.

If no location selections are made yet, this variable has the value `[None,None]`.

Get the value with `setlyze.config.ConfigManager.get()`

```
setlyze.config.cfg.get('locations-selection', slot=int)
```

Set the value with `setlyze.config.ConfigManager.set()`

```
setlyze.config.cfg.set('locations-selection', list, slot=int)
```

2.7 A list `[<selection-1>,<selection-2>]` for storing a maximum of two species selections. `<selection-1>` and `<selection-2>` are lists of integers representing specie IDs. These IDs are the same as the IDs in column `spe_id` in *2.1* and *2.3*.

Get the value with `setlyze.config.ConfigManager.get()`

```
setlyze.config.cfg.get('species-selection', slot=int)
```

Set the value with `setlyze.config.ConfigManager.set()`

```
setlyze.config.cfg.set('species-selection', list, slot=int)
```

2.9 Table `species_spots_1` in the local database containing the SETL records for the *first* selection of species and locations.

This table does not contain the complete records, but just the plate ID and the 25 record surfaces.

SQLite query:

```
CREATE TABLE species_spots_1
(
    id INTEGER PRIMARY KEY,
    rec_pla_id INTEGER,
    rec_sur1 INTEGER,
    rec_sur2 INTEGER,
    rec_sur3 INTEGER,
    rec_sur4 INTEGER,
    rec_sur5 INTEGER,
    rec_sur6 INTEGER,
    rec_sur7 INTEGER,
    rec_sur8 INTEGER,
    rec_sur9 INTEGER,
    rec_sur10 INTEGER,
    rec_sur11 INTEGER,
    rec_sur12 INTEGER,
```

```
    rec_sur13 INTEGER,
    rec_sur14 INTEGER,
    rec_sur15 INTEGER,
    rec_sur16 INTEGER,
    rec_sur17 INTEGER,
    rec_sur18 INTEGER,
    rec_sur19 INTEGER,
    rec_sur20 INTEGER,
    rec_sur21 INTEGER,
    rec_sur22 INTEGER,
    rec_sur23 INTEGER,
    rec_sur24 INTEGER,
    rec_sur25 INTEGER
);
```

2.9.1 Same as *2.9*, but with unique plates.

2.9.2 Same as *2.9*, but with plates with just one spot removed.

2.10 Table `species_spots_2` in the local database containing the SETL records for the *second* selection of species and locations.

This table does not contain the complete records, but just the plate ID and the 25 record surfaces.

SQLite query:

```
CREATE TABLE species_spots_2
(
    id INTEGER PRIMARY KEY,
    rec_pla_id INTEGER,
    rec_sur1 INTEGER,
    rec_sur2 INTEGER,
    rec_sur3 INTEGER,
    rec_sur4 INTEGER,
    rec_sur5 INTEGER,
    rec_sur6 INTEGER,
    rec_sur7 INTEGER,
    rec_sur8 INTEGER,
    rec_sur9 INTEGER,
    rec_sur10 INTEGER,
    rec_sur11 INTEGER,
    rec_sur12 INTEGER,
    rec_sur13 INTEGER,
    rec_sur14 INTEGER,
    rec_sur15 INTEGER,
    rec_sur16 INTEGER,
    rec_sur17 INTEGER,
    rec_sur18 INTEGER,
    rec_sur19 INTEGER,
    rec_sur20 INTEGER,
    rec_sur21 INTEGER,
    rec_sur22 INTEGER,
    rec_sur23 INTEGER,
    rec_sur24 INTEGER,
    rec_sur25 INTEGER
);
```

2.10.1 Same as *2.10*, but with unique plates.

2.10.2 Same as *2.10*, but with plates with just one spot removed.

2.12 Table `spot_distances_observed` in the local database containing the observed spot distances.

Contains the spot distances for the records in *2.9* if created by `calculate_distances_intra()`.

If the table is created by `calculate_distances_inter()`, the table contains the distances between spots in *2.9* and *2.10*.

SQLite query:

```
CREATE TABLE spot_distances_observed
(
    id INTEGER PRIMARY KEY,
    rec_pla_id INTEGER,
    distance REAL
);
```

2.13 Table `spot_distances_expected` in the local database. Has the same design as *2.12*, but contains random generated spot distances instead. These random generated spot distances will serve as the expected spot distances.

SQLite query:

```
CREATE TABLE spot_distances_expected
(
    id INTEGER PRIMARY KEY,
    rec_pla_id INTEGER,
    distance REAL
);
```

2.14 Table `info` in the local SQLite database for storing basic information about the local database.

SQLite query:

```
CREATE TABLE info
(
    id INTEGER PRIMARY KEY,
    name VARCHAR,
    value VARCHAR
);
```

This information includes its creation date, the data source, and a version number. The data source is a string which has the same design as *2.22*. You can insert the data source with the following SQLite query

```
cursor.execute( "INSERT INTO info VALUES (null, 'source', ?)", [setlyze.config.cfg.get('data-source'
```

Giving a version number to the local database could be useful in the future. We can then notify the user if the local database is too old, followed by creating a new local database. This would only work if the version for the database is incremented each time you change the design of the local database. To do this, edit the version number in `create_table_info()`. The version number can be inserted with

```
cursor.execute("INSERT INTO info VALUES (null, 'version', ?)", [db_version])
```

The creation date and data source is inserted by the methods `insert_from_csv()` and `insert_from_db()`. The date can be inserted with

```
cursor.execute( "INSERT INTO info VALUES (null, 'date', date('now'))" )
```

2.15 Table `setl_plates` in the SETL database. The SETL database can be either the MS Access database or the PostgreSQL database. This table contains the SETL plate records.

PostgreSQL query:

```
CREATE TABLE setl_plates
(
    pla_id                  SERIAL,
    pla_loc_id              INTEGER NOT NULL,
    pla_setl_coordinator    VARCHAR(100),
    pla_nr                  VARCHAR(100),
    pla_deployment_date     TIMESTAMP,
    pla_retrieval_date      TIMESTAMP,
    pla_water_temperature   VARCHAR(100),
    pla_salinity            VARCHAR(100),
    pla_visibility          VARCHAR(100),
    pla_remarks             VARCHAR(300),

    CONSTRAINT pla_id_pk PRIMARY KEY (pla_id),
    CONSTRAINT pla_loc_id_fk FOREIGN KEY (pla_loc_id)
        REFERENCES setl_localities (loc_id)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

2.16 Table `plates` in the local SQLite database. This table is only filled if the user selected CSV files to import SETL data from. By default this table is empty, and the plates data from *2.15* is used.

SQLite query:

```
CREATE TABLE plates
(
    pla_id INTEGER PRIMARY KEY,
    pla_loc_id INTEGER,
    pla_setl_coordinator VARCHAR,
    pla_nr VARCHAR,
    pla_deployment_date TEXT,
    pla_retrieval_date TEXT,
    pla_water_temperature VARCHAR,
    pla_salinity VARCHAR,
    pla_visibility VARCHAR,
    pla_remarks VARCHAR
);
```

2.17 Links to an instance of `xml.dom.minidom.Document`. It's a XML DOM (Document Object Model) object containing the analysis settings and results. This XML DOM object is generated by `setlyze.std.ReportGenerator`.

Get the value with `setlyze.config.ConfigManager.get()`

```
setlyze.config.cfg.get('analysis-report')
```

Set the value with `setlyze.config.ConfigManager.set()`

```
setlyze.config.cfg.set('analysis-report', value)
```

2.18 CSV file containing the locality records exported from the MS Access SETL database.

If exported from the MS Access SETL database, the CSV file must have the format

```
LOC_id;LOC_name;LOC_nr;LOC_coordinates;LOC_description
```

2.19 CSV file containing the specie records exported from the MS Access SETL database.

If exported from the MS Access SETL database, the CSV file must have the format

```
SPE_id;SPE_name_venacular;SPE_name_latin;SPE_invasive_in_NL;SPE_description;SPE_remarks;SPE_picture
```

2.20 CSV file containing the plate records exported from the MS Access SETL database.

If exported from the MS Access SETL database, the CSV file must have the format

```
PLA_id;PLA_LOC_id;PLA_SETL_coordinator;PLA_nr;PLA_deployment_date;PLA_retrieval_date;PLA_water_tempe
```

2.21 CSV file containing the SETL records exported from the MS Access SETL database.

If exported from the MS Access SETL database, the CSV file must have the format

```
REC_id;REC_PLA_id;REC_SPE_id;REC_?;REC_O;REC_R;REC_C;REC_A;REC_E;REC_sur?;REC_sur1;REC_sur2;REC_sur3
```

2.22 A string variable representing the current data source.

Can be either `setl-database` or `csv-msaccess`. Several application functions check this variable to figure out where to obtain data from. The first means the PostgreSQL SETL database, and the second from user selected CSV files exported from the MS Access SETL database.

This variable should be set whenever the data source has changed.

Get the value with `setlyze.config.ConfigManager.get()`

```
setlyze.config.cfg.get('data-source')
```

Set the value with `setlyze.config.ConfigManager.set()`

```
setlyze.config.cfg.set('data-source', value)
```

2.23 Table `spot_distances` in the local database containing all possible pre-calculated spot distances.

SQLite query:

```
CREATE TABLE spot_distances
(
    id INTEGER PRIMARY KEY,
    delta_x INTEGER,
    delta_y INTEGER,
    distance REAL
);
```

Each distance in this table is coupled to a horizontal and a vertical spot difference. The distances are pre-calculated by `setlyze.std.distance()`. In other words, if we have two spots, and we know the horizontal difference (Δx) and the vertical difference (Δy), we can look up the corresponding distance in the `spot_distances` table.

2.24 Variable of type `dict` containing the plate areas definition for `analysis 1`.

The dictionary has the format

```
{
'area1': list,
'area2': list,
'area3': list,
'area4': list
}
```

Where `list` is a list of strings. The possible strings are `A`, `B`, `C` and `D`. Each letter represents a surface on a SETL plate. For a clearer picture, refer to *figure-setl-plate*.

The default value for the plate areas definition is

```
{
'area1': ['A'],
'area2': ['B'],
'area3': ['C'],
'area4': ['D']
}
```

Using `setlyze.gui.DefinePlateAreas`, the user can change this definition. The user could for example combine the surfaces `A` and `B`, meaning the value for this variable becomes

```
{
'area1': ['A', 'B'],
'area3': ['C'],
'area4': ['D']
}
```

Keep in mind that the dictionary keys (area1, area2, ..) don't have any meaning. They just make it possible to destinct between the plate areas.

Get the value with `setlyze.config.ConfigManager.get()`

```
setlyze.config.cfg.get('plate-areas-definition')
```

Set the value with `setlyze.config.ConfigManager.set()`

```
setlyze.config.cfg.set('plate-areas-definition', value)
```

2.25 An application variable that contains the actual species totals for each plate area. Keep in mind that this is not the number of individual organisms found on the plate area, as the records just tell the presence of a specie. So it tells how many times the presence of a specie was found on each plate area.

This is what the value can look like

```
{
'area4': 52,
'area1': 276,
'area2': 751,
'area3': 457
}
```

Namespace: `setlyze.analysis.spot_preference.areas_totals_observed`

2.26 An application variable that contains the expected species totals for each plate area. Keep in mind that this not the number of individuals found on the plate area, as the records just tell the presence of a specie.

This is what the value can look like

```
{
'area4': 61.439999999999998,
'area1': 245.75999999999999,
'area2': 737.27999999999997,
'area3': 491.51999999999998
}
```

Namespace: `setlyze.analysis.spot_preference.areas_totals_expected`

2.27 The element `location_selections` in the XML DOM report that contains the user selected locations.

2.28 The element `specie_selections` in the XML DOM report that contains the user selected species.

2.29 The element `spot_distances_observed` in the XML DOM report that contains the actual spot distances.

2.30 The element `spot_distances_expected` in the XML DOM report that contains the expected spot distances.

2.31 The element `plate_areas_definition` in the XML DOM report that contains the user defined plate areas definition.

2.32 The element `area_totals_observed` in the XML DOM report that contains the actual species totals per plate area.

2.33 The element `area_totals_expected` in the XML DOM report that contains the expected species totals per plate area.

2.34 The element `statistics_normality` in the XML DOM report that contains the statistic results for the normality tests.

2.35 The element `statistics_significance` in the XML DOM report that contains the statistic results for the significance tests.

2.36 Analysis variable that contains the statistic results for the normality tests.

Namespace: `setlyze.analysis.attraction_intra.Begin.statistics['normality']`

2.37 Analysis variable that contains the statistic results for the significance tests.

Namespace: `setlyze.analysis.attraction_intra.Begin.statistics['significance']`

2.38 The element `analysis` in the XML DOM report that contains the name of the analysis.

2.39 Table `plate_spot_totals` in the local database for the number of positive spots for each plate ID in the tables *2.9* and/or *2.10*.

Column `n_spots_a` is for the spots in *2.9*, and column `n_spots_b` for the spots in *2.10*.

SQLite query:

```
CREATE TABLE plate_spot_totals
(
    pla_id INTEGER PRIMARY KEY,
    n_spots_a INTEGER,
    n_spots_b INTEGER
);
```

2.40 A XML file containing all data elements from *2.17*.

3.x Graphical User Interfaces

Design Part #	Reference
3.0	*Select Analysis*
3.1	*Select Locations*
3.2	*Select Species*
3.3	*Analysis Report*
3.4	*Change Data Source*
3.5	*Define Plate Areas*
3.6	*Progress Dialog*

4.x Documents

Design Parts: Documents The design parts in this overview describes all technical design parts representing documents created by SETLyze.

4.x Documents

4.0 The analysis report. This document can be exported in different formats. To be supported formats are plain text and LaTeX.

A LaTeX documents can be converted to PDF using third party tools (e.g. `pdflatex` on UNIX systems).

Navigating the SETLyze Code Base

SETLyze's many functions and classes are stored in different modules. Classes and functions with similar functions are placed in the same module.

Below is an overview of all modules for SETLyze. You can click on a module to get a description of that module and all its elements. You can even view the source-code for a specific function or class by clicking the *[source]* link on the right side of the description.

SETLyze modules

SETLyze Standard Modules This reference manual describes the modules that are part of SETLyze.

`setlyze.config` — Configuration manager

Author Serrano Pereira

Release 1.0

Date December 16, 2010

Module Contents This modules provides application-wide access to SETLyze's configuration and data variables.

This module provides an object `cfg` for handling a fixed set of configuration and data variables for SETLyze. The big advantage is that this makes the variables available across all modules. Here is a small usage example,

```
>>> import setlyze.config
>>> setlyze.config.cfg.set('significance-alpha', 0.01)
>>> setlyze.config.cfg.set('species-selection', [11, 12, 13, 14], slot=0)
>>> setlyze.config.cfg.set('species-selection', [15, 16, 17], slot=1)
>>> print "The alpha level for the t-test and Wilcoxon test is set to", setlyze.config.cfg.get('signi
The alpha level for the t-test and Wilcoxon test is set to 0.01
>>> print "The first species selection is", setlyze.config.cfg.get('species-selection', slot=0)
The first species selection is [11, 12, 13, 14]
>>> print "The second species selection is", setlyze.config.cfg.get('species-selection', slot=1)
The second species selection is [15, 16, 17]
```

Importing this module in a different module gives access to the same `cfg` object, and thus its variables can be obtained or manipulated using its get() and set() methods.

class `setlyze.config.`**ConfigManager**

Class for managing SETLyze's data and configuration variables.

An instance of this class provides access to a fixed set of variables that need to be accessable across SETLyze's modules. By importing this module, one instance of this class is created. Subsequent imports in other modules provides access to that same instance.

The method set() is used to change the value of variables. The method get() is used to get the value of a variable.

All variables and their default values can be found in the variable `DEFAULT_CONFIG` of this module.

Design Part: 1.57

get(*key*, ***kwargs*)

Return the value for the configuration with name *key*.

Some configurations have extra keyword arguments. These arguments are handled by *kwargs*. The configurations that have extra arguments are as follows:

locations-selection, species-selection If *slot* is set to 0 (default), the value of the first selection is returned. If set to 1,the second selection is returned.

set(*key*, *value*, ***kwargs*)

Set the configuration with name *key* to *value*.

Some configurations have extra keyword arguments. These arguments are handled by *kwargs*. The configurations that have extra arguments are as follows:

locations-selection, species-selection If *slot* is set to 0 (default), the value is saved as the first selection. If set to 1, the value is saved as the second selection.

set_data_source(*source*)
Set the configuration with name `data-source` to *source*.

Possible values for *source* are `setl-database` and `csv-msaccess`. The value of this configuration tells the application where to look for SETL data. This is especially used by the database module.

If an unknown data source is given, an error is printed.

setlyze.database — Database access

Author Serrano Pereira

Release 1.0

Date December 13, 2010

Module Contents Facilitate access to the local SQLite database and the remote SETL database.

The local SQLite database is created by `setlyze.database.MakeLocalDB.create_new_db()`. It is created using the SQLite Python module. This database is used internally by SETLyze for local data storage and allows for the execution of SQL queries. The database is a single file created in the user's home directory in a subfolder called `.setlyze`.

Here we refer to the PostgreSQL SETL database as the remote SETL database.

class `setlyze.database.`**AccessDBGeneric**
Super class for AccessLocalDB and AccessRemoteDB.

This class contains methods that are generic for both sub-classes. It provides both sub classes with methods for data that is always present in the local database.

fill_plate_spot_totals_table(*spots_table1*, *spots_table2=None*)
Fill the `plate_spot_totals` table with the number of positive spots for each plate.

The information is obtained from the tables *spots_table1* and *spots_table2* (optional).

The `plate_spot_totals` table is used in several situations:

- Calculating the expected distances, where the positive spot number serves as a template for the random spots generator (see *~setlyze.std.get_random_for_plate*).
- Significance calculators, where the tests are applied to plates with a specific number of positive spots (see *get_distances_matching_spots_total*).

If just *spots_table1* is provided, only the column `n_spots_a` is filled with positive spot numbers. If both *spots_table1* and *spots_table2* are provided, `n_spots_a` is filled from *spots_table1*, and `n_spots_b` filled from *spots_table2*.

Design Part: 1.73

get_distances_matching_ratios(*cursor*, *distance_table*, *ratios*)
Get the spot distances from distance table *distance_table* where positive spots numbers between species A and B have ratio matching the list of ratios *ratios*.

The ratio A:B is considered the same as B:A.

cursor must be a SQLite cursor object.

get_distances_matching_spots_total(*cursor*, *distance_table*, *spots_n*)
Get the distances from distance table *distance_table* that are from plates having *spots_n* positive spot numbers.

cursor must be a SQLite cursor object.

get_locations()
Return a list of all locations from the local database.

Returns a list of tuples `(loc_id, 'loc_name')`.

make_plates_unique(*slot*)
Combine the records with the same plate ID in a spots table. The value for *slot* defines which spots table is used. The possible values of *slot* are `0` for table `species_spots_1` and `1` for `species_spots_2`.

We're doing this so we can threat multiple species selected by the user as a single specie.

Design Part: 1.20

remove_single_spot_plates(*table*)
Remove records that have just one positive spot. Intra-specific distance can't be calculated for those.

Note: Use of this function is discouraged. You can easily check for a minimum number of spots in your functions. Also the function `get_spot_combinations_from_record()` will return an empty list if no combinations are possibe (e.g. the record contains less than 2 positive spots).

Design Part: 1.21

class `setlyze.database.`**AccessLocalDB**
Provide standard methods for accessing data in the local SQLite database. These methods are only used when the data source is set to `csv-msaccess`.

Inherits from `AccessDBGeneric` which provides this class with methods that are not data source specific.

Design Part: 1.28

get_record_ids(*loc_ids*, *spe_ids*)
Return a list of record IDs that match the localities IDs in the list *loc_ids* and the species IDs in the list *spe_ids*.

Design Part: 1.41

get_species(*loc_slot=0*)
Return a list of species tuples in the format `(spe_id, 'spe_name_venacular', 'spe_name_latin')` from the local database that match the localities selection.

The values for *loc_slot* are `0` for the first localities selection and `1` for the second localities selection.

get_spots(*rec_ids*)
Return all 25 spot booleans for the records with IDs matching the record IDs in the list *rec_ids*.

set_species_spots(*rec_ids*, *slot*)
Create a table in the local database containing the spots information for SETL records matching *rec_ids*. Two spots tables can be created. The values for *slot* can be `0` for table `species_spots_1` and `1` for `species_spots_2`.

Each record in the spots table consists of the plate ID followed by the 25 spot booleans.

Design Part: 1.19.1

class `setlyze.database.`**AccessRemoteDB**
Provide standard methods for accessing data in the remote PostgreSQL SETL database. These methods are only used when the data source is set to `setl-database`.

Inherits from `AccessDBGeneric` which provides this class with methods that are not data source specific.

Design Part: 1.29

class `setlyze.database.`**MakeLocalDB**
Create a local SQLite database with default tables and fill some tables based on the data source.

This class can load data from two data sources. One data source is user supplied CSV files containing SETL data. The CSV files must be exported by the MS Access SETL database. The second source is the PostgreSQL SETL database. This function requires a direct connection with the SETL database server.

Because the import of SETL data into the local database can take a while, and instance of this class must run in a separate thread. An instance of this class is therefor a thread object.

Once a thread object is created, its activity must be started by calling the thread's start() method. This invokes the run() method in a separate thread of control.

Design Part: 1.2

create_new_db()
Create a new local database and then calls the methods that create the necessary tables.

This deletes the current local database if present in the user's home folder.

Design Part: 1.38

create_table_info()
Create a table for storing basic information about the local database.

This information includes its creation date and the data source. The data source is either the SETL database (MS Access/PostgreSQL) or CSV files containing SETL data. A version number for the database is also saved. This could be handy in the future, for example we can notify the user if the local database is too old, followed by creating a new local database.

Design Part: 1.75

create_table_localities()
Create a table for the SETL localities.

Because the data from this table is accessed frequently, the localities records are automatically saved to this table when an analyze is started.

Design Part: 1.76

create_table_plate_spot_totals()
Create a table for the total of spots per plate in the distance tables.

Design Part: 1.85

create_table_plates()
Create a table for the SETL plates.

This table is only filled if the user selected CSV files to import SETL data from. If the remote SETL database is used, the plate records are obtained directly via queries.

Design Part: 1.78

create_table_records()
Create a table for the SETL records.

This table is only filled if the user selected CSV files to import SETL data from. If the remote SETL database is used, the records are obtained directly via queries.

Design Part: 1.79

create_table_species()
Create a table for the SETL species.

Because the data from this table is accessed frequently, the species records are automatically saved to this table when an analyze is started.

Design Part: 1.77

create_table_species_spots_1()
Create a table that will contain the SETL records for the first species selection.

Because the user can select multiple species, the plate IDs in column `rec_pla_id` don't have to be unique, so we're creating a separate column `id` as the primary key.

Design Part: 1.80

create_table_species_spots_2()
Create a table that will contain the SETL records for the second species selection.

Because the user can select multiple species, the plate IDs in column `rec_pla_id` don't have to be unique, so we're creating a separate column `id` as the primary key.

Design Part: 1.81

create_table_spot_distances()
Create a table that will contain all the pre-calculated spot distances that are possible on a SETL plate.

Design Part: 1.82

create_table_spot_distances_expected()
Create a table for the expected spot distances.

Design Part: 1.84

create_table_spot_distances_observed()
Create a table for the observed spot distances.

Design Part: 1.83

fill_distance_table()
Calculate all possible spot distances for a SETL plate and put them in a table in the local database.

Design Part: 1.47

insert_from_csv()
Create a new local database and load all SETL data from the user selected CSV files into the local database.

The SETL data is loaded from four separate CSV files:

- localities_file, containing the SETL locations.
- species_file, containing the SETL species.
- records_file, containing the SETL records.
- plates_file, containing the SETL plates.

These files must be exported from the MS Access SETL database and contain all fields. The CSV file must be in Excel format, which means delimited by semicolons (;) and double quotes (") as the quote character for fields with special characters.

Design Part: 1.32

insert_from_db()
Create a new local database and load localities and species data from the remote SETL database into the local database.

This method requires a direct connection with the SETL database server.

The reason why we don't load all SETL data into the local database is because we can execute queries on the remote database. So there's no need to load large amounts of data onto the user's computer. Because the localities and species data is accessed often, loading this into the local database increases speed of the application.

Design Part: 1.33

insert_localities_from_csv(*delimiter=';'*, *quotechar=""*)
Insert the SETL localities from a CSV file into the local database.

delimiter is a one-character string used to separate fields in the CSV file. *quotechar* is a one-character string used to quote fields containing special characters in the CSV file. The default values for these two arguments are suited for CSV files in Excel format.

For a description of the format for the localities file, refer to *2.18*.

Design Part: 1.34

insert_plates_from_csv(*delimiter=';'*, *quotechar=""*)
Insert the plates from a CSV file into the local database.

delimiter is a one-character string used to separate fields in the CSV file. *quotechar* is a one-character string used to quote fields containing special characters in the CSV file. The default values for these two arguments are suited for CSV files in Excel format.

For a description of the format for the localities file, refer to *2.20*.

Design Part: 1.36

insert_records_from_csv(*delimiter=';'*, *quotechar=""*)
Insert the records from a CSV file into the local database.

delimiter is a one-character string used to separate fields in the CSV file. *quotechar* is a one-character string used to quote fields containing special characters in the CSV file. The default values for these two arguments are suited for CSV files in Excel format.

For a description of the format for the localities file, refer to *2.21*.

Design Part: 1.37

insert_species_from_csv(*delimiter=';'*, *quotechar=""*)
Insert the species from a CSV file into the local database.

delimiter is a one-character string used to separate fields in the CSV file. *quotechar* is a one-character string used to quote fields containing special characters in the CSV file. The default values for these two arguments are suited for CSV files in Excel format.

For a description of the format for the localities file, refer to *2.19*.

Design Part: 1.35

remove_db_file(*tries=0*)
Remove the database file.

This method does 3 tries if for some reason the database file could not be deleted (e.g. the file is in use by a different process.

run()
Decide based on the configuration variables which functions should be called.

This method first checks if SETLyze is configured to create a new local database. This is done by checking the value of `setlyze.config.cfg.get('make-new-db')`. If this is set to `False`, the method ends and the thread is destroyed. If set to `True`, a new local database is created based on the data source configuration.

The data source configuration is checked by calling `setlyze.config.cfg.get('data-source')`. If this is set to `setl-database`, the method `insert_from_db` is called and some SETL data from the remote SETL database is loaded into the local database. If set to `csv-msaccess`, the method

`insert_from_csv` is called which loads all SETL data from the supplied CSV files into the local database.

Design Part: 1.31

`setlyze.database.`**`get_database_accessor`**`()`
Return an object that facilitates access to either the local or the remote database.

Based on the data source configuration, obtained with `setlyze.config.cfg.get('data-source')`, this function wil either return an instance of AccessLocalDB or AccessRemoteDB. This instance provides methods that are specific to the data source currently in use.

`setlyze.database.`**`get_plates_total_matching_spots_total`**`(`*n_spots*, *slot=0*`)`
Return an integer representing the number of plates that match the provided number of positive spots *n_spots*.

Possible values for *slot* are 0 for the first species selection, and 1 for the second species selection.

`setlyze.gui` — Graphical interfaces

Author Serrano Pereira

Release 1.0

Date December 16, 2010

Module Contents This module provides the graphical user interfaces ("GUIs") for SETLyze.

We use GTK+ for the GUI creation. GTK+ is a highly usable, feature rich toolkit for creating graphical user interfaces which boasts cross platform compatibility and an easy to use API.

GTK+ is an event driven toolkit, which means it will sleep in gtk.main() until an event occurs and control is passed to the appropriate function. To understand the code, it's important that you gain basic understanding of signals and callbacks, as these are used throughout the application (not just this module). The following tutorial is a good place to start: Theory of Signals and Callbacks

Each class in this module represents a graphical dialog or window. Displaying one of these dialogs is as easy as instantiating the class. You can easily test this from the interactive Python shell:

```
>>> import setlyze.gui
>>> g = setlyze.gui.SelectLocations()
```

The above would display a nice localities selection dialog. You could then do the following to get a list of the selected locations.

```
>>> import setlyze.config
>>> setlyze.config.cfg.get('locations-selection')
```

class `setlyze.gui.`**`ChangeDataSource`**
Display a dialog that allows the user to change to a different data source. The following data sources are supported:

- CSV files with SETL data exported from the MS Access SETL database.
- TODO: The remote SETL database. This requires a direct connection with the SETL database server.

Design Part: 1.90

`create_layout``()`
Construct the layout for the dialog.

`create_page_csv``()`
Return a notebook page for switching to SETL data from CSV files.

create_page_db()
Return a notebook page for switching to SETL data from the remote SETL database.

on_cancel(*widget*, *data=None*)
Close the dialog.

on_csv_ok(*widget*, *data=None*)
Save the paths to the CSV files, set the new value for the data source configuration, load the SETL data from the CSV file into the local database, and close the dialog.

update_working_folder(*chooser*, *data=None*)
Set the working folder for the file choosers to the folder where the first data file was selected from.

This way the user doesn't have to navigate to the same folder multiple times.

class setlyze.gui.**DefinePlateAreas**(*title='Define SETL-plate Areas'*)
Display a dialog that allows the user to define the areas on a SETL plate.

Below is a SETL-plate with a grid. By default there are 4 surface areas defined on a SETL plate (A, B, C and D). Sometimes it's useful to combine plate areas.

A	B	B	B	A
B	C	C	C	B
B	C	D	C	B
B	C	C	C	B
A	B	B	B	A

So if the user decides to combine A and B, the plate areas definition looks like this,

A	A	A	A	A
A	C	C	C	A
A	C	D	C	A
A	C	C	C	A
A	A	A	A	A

Design Part: 1.91

create_definition_table()
Construct the form for defining the plate areas.

create_layout()
Construct the layout for the dialog.

get_selection()
Return the plate areas as defined by the user.

iscorrect(*definition*)
Check if the user defined plate areas are correct.

Return True if the definition is correct, and False if it's not. If the definition is False, display a message dialog describing the problem.

normalize(*definition*)
Return a normalized and simplified version of *definition*.

This means that an area selection that looks like [True, False, True, False] will be converted to ['A','C']. Empty areas are removed from the normalized spot areas definition.

For example, if *definition* equals

```
{
'area1': [True, False, False, False],
'area2': [False, True, False, False],
```

```
'area3': [False, False, True, True],
'area4': [False, False, False, False]
}
```

This method will return

```
{
'area1': ['A'],
'area2': ['B'],
'area3': ['C', 'D']
}
```

on_back(*widget*, *data=None*)
Destroy the dialog and send the `define-areas-dialog-back` signal.

This function is called when the user presses the Back button.

on_close_dialog(*widget=None*, *data=None*)
Close the dialog and send the `define-areas-dialog-closed` signal.

This method should is called when the user presses the X (close) button of the selection dialog. The signal can then be handled in the analysis class.

on_continue(*widget*, *data=None*)
Check if the user made a correct definition. If yes, normalize the definition and save it.

save(*definition*)
Save the plate areas definition and emit the `plate-areas-defined` signal.

class `setlyze.gui.`**DisplayReport**(*report=None*)
Display a dialog visualizing the elements in the XML DOM analysis data object.

This class uses `setlyze.std.ReportReader` to read the data from the XML DOM analysis data object.

Design Part: 1.89

add_area_totals()
Add the species totals per plate area to the report dialog.

add_distances()
Add the spot distances (observed + expected) to the report dialog.

add_locations_selections()
Add the locations selection(s) to the report dialog.

add_plate_areas_definition()
Add the plate areas definition to the report dialog.

add_report_elements()
Add the report elements present in the XML DOM object to the report dialog.

add_selections()
Add the location + species selections to the report dialog.

add_species_selections()
Add the species selection(s) to the report dialog.

add_statistics_chisq_ratios()
Add the statistic results to the report dialog.

add_statistics_chisq_spots()
Add the statistic results to the report dialog.

add_statistics_normality()
Add the statistic results for normality to the report dialog.

add_statistics_wilcoxon_ratios()
Add the statistic results to the report dialog.

add_statistics_wilcoxon_spots()
Add the statistic results to the report dialog.

add_title_header()
Add a header text to the report dialog.

The header contains the name of the analysis.

create_layout()
Construct the layout for the dialog.

on_close(*obj*, *data=None*)
Close the dialog and emit the *report-dialog-closed* signal.

on_save(*obj*, *data=None*)
Display a dialog that allows the user to save the report to a file.

set_report_reader(*report*)
Create a report reader and pass the XML DOM report data object *report* to the reader. *report* can also be the path to a report data XML file.

class setlyze.gui.**ProgressDialog**(*title*, *description*)
Display a progress dialog.

This progress dialog is useful if you have a process that could take a long time to run. This class allows you to display a progress dialog which shows the progress for a long process (called the worker process). This worker process needs to run in a separate thread for this to work.

Follow these steps to get the progress dialog working:

1. Create an instance of this class,

```
pd = setlyze.gui.ProgressDialog(title="Analyzing",
    description="Performing heavy calculations, please wait...")
```

2. Register the progress dialog using the `config` module,

```
setlyze.config.cfg.set('progress-dialog', pd)
```

3. Edit the worker process to automatically update the progress dialog. This is as easy as calling the custom update method between the lines of your code,

```
setlyze.std.update_progress_dialog(0.0, "Calculating this...")
...
setlyze.std.update_progress_dialog(0.5, "Calculating that...")
...
setlyze.std.update_progress_dialog(1.0, "Finished!")
```

4. Then start your worker process in a separate thread (if you're new to threading, start with the threading documentation)

```
t = MyClass()
t.start()
```

Then run your application and watch the progress bar grow.

Design Part: 1.92

create_layout()
Construct the layout for the dialog.

destroy_silent()
Destroy the dialog without sending the `progress-dialog-closed` signal.

on_close(*widget=None*)
Destroy the dialog and send the `progress-dialog-closed` signal.

class `setlyze.gui.`**SelectExportElements**(*reader*)
Display a dialog for allowing the user to select which report elements to export.

create_layout()
Add widgets to the dialog.

get_selected_elements()
Return a list with the names of the selected report elements.

set_report_reader(*reader*)
Set the report reader.

class `setlyze.gui.`**SelectLocations**(*title='Locations Selection'*, *description='Select the locations:'*, *width=-1*, *slot=0*)
Display a selection dialog that allows the user to make a selection from the SETL locations in the local database.

Design Part: 1.87

create_columns(*treeview*)
Create the columns for the tree view.

create_model()
Create a model for the tree view from the location IDs and names.

Design Part: 1.42

on_back(*button*)
Destroy the selection dialog and send the `locations-dialog-back` signal.

The `locations-dialog-back` signal is sent with the save slot as an attribute. The save slot can have a value of `0` for the first selection and `1` for the second selection.

This function is called when the user presses the Back button in a location selection window.

Design Part: 1.45

save_selection()
Save the locations selection and send the `locations-selection-saved` signal.

The `locations-selection-saved` signal is sent with the save slot as an attribute. The save slot can have a value of `0` for the first selection and `1` for the second selection.

Design Part: 1.7

class `setlyze.gui.`**SelectSpecies**(*title='Species Selection'*, *description='Select the species:'*, *width=-1*, *slot=0*)
Display a selection dialog that allows the user to make a selection from the SETL species in the local database.

Design Part: 1.88

create_columns(*treeview*)
Create columns for the tree view.

create_model()
Create a model for the tree view from the specie IDs and names.

Design Part: 1.43

on_back(*widget*, *data=None*)
Destroy the selection dialog and send the `species-dialog-back` signal.

The `species-dialog-back` signal is sent with the save slot as an attribute. The save slot can have a value of `0` for the first selection and `1` for the second selection.

This function is called when the user presses the Back button in a species selection window.

Design Part: 1.46

save_selection()
Save the species selection and send the `species-selection-saved` signal.

The `species-selection-saved` signal is sent with the save slot as an attribute. The save slot can have a value of `0` for the first selection and `1` for the second selection.

Design Part: 1.12.2

class `setlyze.gui.`**SelectionWindow**(*title*, *description*, *width*, *slot*)
Super class for `SelectLocations` and `SelectSpecies`.

create_layout()
Construct the layout for the selection dialog.

destroy_handler_connections()
Disconnect all signal handlers created by this class.

on_changed(*treeselection*)
Save the locations selection to a temporary variable.

This method is called whenever the selection changes.

on_close_dialog(*widget=None*, *data=None*)
Close the dialog and send the `selection-dialog-closed` signal.

This method should is called when the user pressed the X (close) button of the selection dialog. The signal can then be handled in the analysis class.

on_continue(*button*)
Before saving the localities/species selection, check if anything was selected. If not, display a message dialog. If yes, call method self.save_selection to save the selection and then close the selection dialog.

Design Part: 1.44

on_select_data_files(*button*)
Display the ChangeDataSource window.

Design Part: 1.11

set_description(*description*)
Set the description text to *description*.

set_header(*header*)
Set the header text to *header*.

set_save_slot(*slot*)
Set the localities/species selection save slot to *slot*.

The selection variable has two slots available for saving selections (in analysis 2.2, two selections need to be saved, hence two slots were created).

The possible values of *slot* are `0` for the first selection, and `1` for the second selection.

update_tree(*widget=None*)
Load the localities/species data from the local database into the tree view.

This function should be called whenever the localities/species data is updated. For example after new localities/species data was imported into the local database. This function is also called when the selection dialog is created.

Design Part: 1.39

`setlyze.gui.`**`markup_header`**(*text*)
Apply Pango markup to *text* to make it look like a header.

`setlyze.gui.`**`on_help`**(*button*, *section*)
Display the help contents for *section* in the system's default application for displaying HTML files (usually a web browser).

`setlyze.gui.`**`on_quit`**(*button*, *data=None*)
Quit the application.

`setlyze.locale` — English text retrieval

Author Serrano Pereira

Release 1.0

Date December 14, 2010

Module Contents This module is for storing frequently used English lines used throughout the source-code. The purpose is to have a standard place for storing English sentences. This was basically meant for convenience so the developer doesn't have to browse through code just to change a sentence.

This module wasn't created for adding multi-language support, though it can be easily expanded to do so.

`setlyze.locale.`**`text`**(*key*, **args*)
Return the text string from the `ENGLISH` dictionary where key is *key*.

A simple example:

```
>>> import setlyze.locale
>>> setlyze.locale.text('analysis-running')
'Please stand by while the analysis is running. This may take a while...'
```

Substitution is also supported:

```
>>> import setlyze.locale
>>> setlyze.locale.text('dummy', "windy with a slight chance of rain")
"And tomorrow's forecast is, windy with a slight chance of rain"
```

`setlyze.std` — Standard functions and classes

Author Serrano Pereira

Release 1.0

Date December 13, 2010

Module Contents This module provides standard functions and classes. All functions and classes that don't belong in any of the other modules are placed here.

class `setlyze.std.`**`ExportLatexReport`**(*reader=None*)
Generate an analysis report in LaTeX format.

Design Part:

set_report_reader(*reader*)
Set the report reader.

class setlyze.std.**ExportTextReport**(*reader=None*)
Generate an analysis report in text format.

Design Part:

chapter(*header*)
Return *header* marked up as a header for parts in reStructuredText format.

part(*header*)
Return *header* marked up as a header for parts in reStructuredText format.

section(*header*)
Return *header* marked up as a header for sections in reStructuredText format.

set_report_reader(*reader*)
Set the report reader.

subsection(*header*)
Return *header* marked up as a header for subsections in reStructuredText format.

subsubsection(*header*)
Return *header* marked up as a header for subsubsections in reStructuredText format.

table(*headers*)
Return a table header with column names from *headers* in reStructuredText format.

headers is a list/tuple containing two-item tuples `(<column name>, <column width>)`. The column name is a string, and the column width is an integer defining the character width of the column. A value of -1 for column with means that the column will have the same width as the column name.

table_rule(*col_widths*)
Return a rule for a table with column widths *col_widths*.

col_widths is a list/tuple containing column widths (integers).

class setlyze.std.**ReportGenerator**
Create a XML DOM (Document Object Model) object of the analysis settings, data and results. The DOM can then be exported to an XML file containing all data for the analysis.

Using XML DOM objects for storing analysis data has great advantages. Because the object can contain all analysis data, it's easy to use Python's XML parser to generate analysis reports. We can allow the user to choose which elements of the XML DOM object to export to say a LaTeX document. Also, `xml.dom.minidom` provides methods for exporting this object to an XML file. This file by default contains all analysis data. This file can be easily loaded in SETLyze so we can display a dialog showing the analysis data and results present in that XML file. The XML file can be used as a backup file of the analysis data.

So too the class `ReportReader` uses this XML DOM object to access the analysis data.

Design Part: 1.48

create_element(*parent*, *name*, *child_elements={}*, *attributes={}*, *text=None*)
Add a new child element with name *name* to the element *parent* for the XML DOM object.

Usually you'll start by adding child elements to the root element `self.report`. In this case *parent* would be `self.report`. It's then possible to add child elements for for those by setting *parent* to the newly created child elements.

Optionally you can easily add child elements by setting *child_elements* to a dictionary. The dictionary keys will be the names of the child elements, and the corresponding dictionary values will be the values for the elements.

This method also allows you to easily add attributes. To add attributes, set *attributes* to a dictionary. The dictionary keys will be the names of attributes, and the corresponding dictionary values will be the values for the attributes.

The *text* argument gives the element a value. The value of *text* can be anything.

This method returns the newly created element. This allows you to set the returned element object as the parent element for other elements.

We shall clarify with some usage examples. First we add an empty child element to the root element:

```
location_selections_element = self.create_element(
    parent=self.report,
    name="location_selections"
    )
```

Then we add some child elements to the just created element:

```
self.create_element(
    parent=location_selections_element,
    name="location",
    child_elements={'nr':1, 'name':"Aquadome, Grevelingen"},
    attributes={'id':1}
    )

self.create_element(
    parent=location_selections_element,
    name="location",
    child_elements={'nr':2, 'name':"Colijnsplaat, floating dock, Oosterschelde"},
    attributes={'id':2}
    )
```

Would this be exported to an XML file, the contents of the file would look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<setlyze:report xmlns:setlyze="http://www.gimaris.com/setlyze/">
    <location_selections>
        <location id="1">
            <nr>
                1
            </nr>
            <name>
                Aquadome, Grevelingen
            </name>
        </location>
        <location id="2">
            <nr>
                2
            </nr>
            <name>
                Colijnsplaat, floating dock, Oosterschelde
            </name>
        </location>
    </location_selections>
</setlyze:report>
```

export_xml (*filename*)
Export the XML source of the XML DOM report to a file.

get_element (*parent*, *name*)
Return the element object with name *name* from a parent element *parent*.

get_report()
Return the XML DOM report object.

set_analysis(*name*)
Add the element `analysis` with value *name* to the XML DOM report.

This element describes to which analysis this report belongs. So *name* is just a string with the title of an analysis. However, if the value of *name* exists as a key in the `analysis_names` dictionary, the corresponding value from that dictionary will be used as the value for the element instead.

Design Part: 1.72

set_area_totals_expected(*totals_observed*)
Add the element `area_totals_expected` to the XML DOM report.

This element will be filled with the expected species totals per plate area.

The XML representation looks like this:

```
<area_totals_expected>
    <area id="area1">
        24.64
    </area>
    <area id="area2">
        73.92
    </area>
    <area id="area3">
        55.44
    </area>
</area_totals_expected>
```

Design Part: 1.56

set_area_totals_observed(*totals_observed*)
Add the element `area_totals_observed` to the XML DOM report.

This element will be filled with the observed species totals per plate area.

The XML representation looks like this:

```
<area_totals_observed>
    <area id="area1">
        27
    </area>
    <area id="area2">
        75
    </area>
    <area id="area3">
        52
    </area>
</area_totals_observed>
```

Design Part: 1.55

set_location_selections()
Add the element `location_selections` to the XML DOM report.

This element will be filled with the locations selections. If two locations selections were made, both will be added to the element.

The XML representation looks like this:

```
<location_selections>
    <selection slot="0">
        <location id="1">
            <nr>
                1
            </nr>
            <name>
                Aquadome, Grevelingen
            </name>
        </location>
    </selection>
    <selection slot="1">
        <location id="2">
            <nr>
                2
            </nr>
            <name>
                Colijnsplaat, floating dock, Oosterschelde
            </name>
        </location>
    </selection>
</location_selections>
```

Design Part: 1.50

set_plate_areas_definition()

Add the element `plate_areas_definition` to the XML DOM report.

This element will be filled with the user defined spot areas definition.

The XML representation looks like this:

```
<plate_areas_definition>
    <area id="area1">
        <spot>
            A
        </spot>
    </area>
    <area id="area2">
        <spot>
            B
        </spot>
    </area>
    <area id="area3">
        <spot>
            C
        </spot>
        <spot>
            D
        </spot>
    </area>
</plate_areas_definition>
```

Design Part: 1.54

set_specie_selections()

Add the element `specie_selections` to the XML DOM report.

This element will be filled with the species selections. If two species selections were made, both will be added to the element.

The XML representation looks like this:

```
<specie_selections>
    <selection slot="0">
        <specie id="2">
            <name_latin>
                Ectopleura larynx
            </name_latin>
            <name_venacular>
                Gorgelpijp
            </name_venacular>
        </specie>
    </selection>
    <selection slot="1">
        <specie id="6">
            <name_latin>
                Metridium senile
            </name_latin>
            <name_venacular>
                Zeeanjelier
            </name_venacular>
        </specie>
    </selection>
</specie_selections>
```

Design Part: 1.51

set_spot_distances_expected()

Add the element `spot_distances_expected` to the XML DOM report.

This element will be filled with the expected spot distances.

The XML representation looks like this:

```
<spot_distances_expected>
    <distance plate_id="62">
        1.0
    </distance>
    <distance plate_id="62">
        3.16
    </distance>
    <distance plate_id="228">
        4.47
    </distance>
</spot_distances_expected>
```

Design Part: 1.53

set_spot_distances_observed()

Add the element `spot_distances_observed` to the XML DOM report.

This element will be filled with the observed spot distances.

The XML representation looks like this:

```
<spot_distances_observed>
    <distance plate_id="63">
        1.0
    </distance>
    <distance plate_id="63">
        2.0
```

```
        </distance>
        <distance plate_id="229">
            3.16
        </distance>
    </spot_distances_observed>
```

Design Part: 1.52

set_statistics(*name*, *data*)
Add the element `statistics` with child element `name` to the XML DOM report.

This element will be filled with the results of the performed statistical tests. The results must be supplied with the *data* argument. The *data* argument is a list containing dictionaries in the format {'attr': {'<name>': <value>, ...}, 'results': {'<name>': <value>, ...}} where the value for 'attr' is a dictionary with the attributes and 'results' is a dictionary with child elements for the `statistics` element.

An XML representation:

```
<statistics>
    <wilcoxon alternative="two.sided" conf_level="0.95" method="Wilcoxon rank sum test w
        <conf_int_start>
            -2.16
        </conf_int_start>
        <p_value>
            0.353678517318
        </p_value>
        <mean_expected>
            2.13333333333
        </mean_expected>
        <conf_int_end>
            1.0
        </conf_int_end>
        <mean_observed>
            1.33333333333
        </mean_observed>
    </wilcoxon>
</statistics>
```

Design Part: 1.71

class setlyze.std.**ReportReader**(*report=None*)
Provide standard methods for extracting data from the XML DOM object containing analysis data.

This class can also export the XML DOM object to an XML document.

Design Part: 1.49

export_xml(*filename*)
Export the XML source of the XML DOM report to a file.

get_analysis_name()
Return the value of the *analysis* element. This element contains the name of the analysis.

get_area_totals_expected()
Return the expected specie totals per area from the XML DOM report.

This method returns a dictionary. For example:

```
{
'area1': 23.12,
'area2': 60.10,
```

```
'area3': 40.44
}
```

get_area_totals_observed()
Return the observed specie totals per plate area from the XML DOM report.

This method returns a dictionary. For example:

```
{
'area1': 24.64,
'area2': 73.92,
'area3': 55.44
}
```

get_child_names(*parent=None*)
Return a list with all child element names from the XML DOM object. The child elements for the report elements are excluded.

Use this as a quick way to find out which elements are present in a XML DOM report object.

get_element(*parent*, *name*)
Return the element object with name *name* from a parent element *parent*.

get_locations_selection(*slot=0*)
Return the locations selection from the selection slot with number *slot* from the XML DOM report. Possible values for *slot* are `0` and `1`.

This is a generator, meaning that this function returns an iterator. This iterator returns dictionaries in the format `{'nr': <value>, 'name': <value>}`.

Usage example:

```
locations_selection = reader.get_locations_selection(slot=0)
for loc in locations_selection:
    print loc['nr'], loc['name']
```

get_plate_areas_definition()
Return the spot areas definition from the XML DOM report. This method returns a dictionary. For example:

```
{
'area1': ['A'],
'area2': ['B'],
'area3': ['C', 'D']
}
```

or:

```
{
'area1': ['A'],
'area2': ['B'],
'area3': ['C'],
'area4': ['D']
}
```

get_species_selection(*slot=0*)
Return the species selection from the selection slot with number *slot* from the XML DOM report. Possible values for *slot* are `0` and `1`.

This is a generator, meaning that this function returns an iterator. This iterator returns dictionaries in the format `{'name_latin': <value>, 'name_venacular': <value>}`.

Usage example:

```
species_selection = reader.get_species_selection(slot=0)
for spe in species_selection:
    print spe['name_latin'], spe['name_venacular']
```

get_spot_distances_expected()
Return the expected spot distances from the XML DOM report.

This is a generator, meaning that this function returns an iterator. The iterator returns the distances.

Usage example:

```
expected_distances = reader.get_spot_distances_expected()
for dist in expected_distances:
    print dist
```

get_spot_distances_observed()
Return the observed spot distances from the XML DOM report.

This is a generator, meaning that this function returns an iterator. The iterator returns the distances.

Usage example:

```
observed_distances = reader.get_spot_distances_observed()
for dist in observed_distances:
    print dist
```

get_statistics(*name*)
Return the statistics elements with name *name* from the XML DOM report.

This is a generator, meaning that this function returns an iterator. This iterator returns tuples in the format `({'<name>': <value>, ...}, {'<name>': <value>, ...})` where the first dictionary contains the attributes and the second the results.

get_xml()
Return the XML source for the XML DOM report.

set_report(*report*)
Set the XML DOM report object *report* generated by `ReportGenerator`. *report* can also be the path to an XML file containing an analysis report.

class setlyze.std.**Sender**
Custom GObject for emitting SETLyze specific application signals.

This module creates a single instance of this class. Subsequent imports of this module gives access to the same instance. Thus only one instance is created for each run.

The `__gsignals__` class attribute is a dictionary containing all custom signals an instance of this class can emit. To emit a signal, use the `emit()` method. To signal that an analysis has started for example, use:

```
setlyze.std.sender.emit('analysis-finished')
```

If you want to emit a signal from a separate thread, you must use `gobject.idle_add()` as only the main thread is allowed to touch the GUI. Emitting a signal from a separate thread looks like this:

```
gobject.idle_add(setlyze.std.sender.emit, 'analysis-finished')
```

Anywhere in your application you can add a function to be called when this signal is emitted. This function is called a callback method. To add a callback method for a specific signal, use the `connect()` method:

```
self.handler_id = setlyze.std.sender.connect('analysis-finished',
    self.on_analysis_finished)
```

When you are done using that handler, be sure to destroy it as the handler will continue to exist if the callback function does not return `False`. To destroy a signal handler, use the `disconnect()` method:

```
setlyze.std.sender.disconnect(self.handler_id)
```

> **Warning:** Remember to use `gobject.idle_add()` if you decide to emit signals from separate threads. If you don't do this, the application becomes unstable resulting in crashes.

See Also:

Theory of Signals and Callbacks It's recommended to study this subject of the PyGTK documentation to get a better understanding of signals and callbacks.

Advanced Event and Signal Handling It's recommended to study this subject of the PyGTK documentation to get a better understanding of event and signal handling.

gobject.idle_add PyGTK documentation for `gobject.idle_add()`.

`setlyze.std.`**`chisq_test`**(*x*, *y=None*, *correct=True*, *p=None*, *rescale_p=False*, *simulate_p_value=False*, *b=2000*)

Performs chi-squared contingency table tests and goodness-of-fit tests.

This is a wrapper function for the `chisq.test` function from R. It depends on R and RPy. The latter provides an interface to the R Programming Language.

This function returns a dictionary containing the results. Below is the format of the dictionary with example results

```
{
'null.value': {'difference in means': 0},
'method': 'Welch Two Sample t-test',
'p.value': 0.97053139295765201,
'statistic': {'t': -0.037113583386291726},
'estimate': {'mean of y': 2.552142857142857, 'mean of x': 2.5417857142857141},
'conf.int': [-0.56985924418141154, 0.54914495846712563],
'parameter': {'df': 53.965197921982607},
'alternative': 'two.sided'
}
```

See Also:

R Documentation for Pearson's Chi-squared Test for Count Data The R Documentation gives a more extensive documentation of this function, its arguments, usage, etc. To view the documentation, type `help(chisq.test)` from the R prompt.

`setlyze.std.`**`combinations_with_replacement`**(*iterable*, *r*)

Return r length subsequences of elements from the input iterable allowing individual elements to be repeated more than once.

Combinations are emitted in lexicographic sort order. So, if the input iterable is sorted, the combination tuples will be produced in sorted order.

Elements are treated as unique based on their position, not on their value. So if the input elements are unique, the generated combinations will also be unique.

This function was taken from the Python documentation for `itertools`.

A simple example:

```
>>> i = setlyze.std.combinations_with_replacement('ABCD', 2)
>>> [x for x in i]
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'B'), ('B', 'C'), ('B', 'D'), ('C', 'C'
```

setlyze.std.**combine_records**(*records*)

Return a combined SETL records from a list of multiple SETL records with the same plate ID.

records is a list containing multiple SETL records where each record is a sequence with the plate ID as the first item followed by 25 spot booleans. As this function is used for combining records from different species found on the same plate, the plate ID of all records must be equal.

A basic usage example:

```
>>> import setlyze.std
>>> rec1 = (4567,1,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)
>>> rec2 = (4567,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1)
>>> rec3 = (4567,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)
>>> records = [rec1,rec2,rec3]
>>> setlyze.std.combine_records(records)
[4567, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

setlyze.std.**distance**(*p1*, *p2*)

Use the Pythagorean theorem to calculate the distance between two spots.

setlyze.std.**export_report**(*reader*, *path*, *type*, *elements=None*)

Save the data from the XML DOM report to a data file or an analysis report document. The file is saved to *path* in a format specified by *type*. Possible values for *type* are `xml`, `txt` or `latex`.

If *type* is `xml`, all data from the DOM object is saved to an XML file. If *type* is `txt` or `latex`, the report elements specified by the user will be saved to a human readable document.

Design Part: 1.17

setlyze.std.**get_random_for_plate**(*n*)

Return a *n* length list of random integers with a range from 1 to 25. So naturally *n* can have a value from 0 to 25. The list of integers returned represents random selected spots from a 25 spots SETL plate.

We make use of the `random.sample()` function from the Python standard library. As described in the Python documentation, this function is bound to an instance of `random.Random` which uses the `random.random()` method. In turn this method uses the Mersenne Twister (*M. Matsumoto and T. Nishimura*) as the core generator. The Mersenne Twister is one of the most extensively tested random number generators in existence.

See Also:

Module `random` Documentation of the `random` standard module.

setlyze.std.**get_spot_combinations_from_record**(*record1*, *record2=None*)

Return all possible positive spot combinations for *record1* or if both are provided, between *record1* and *record2*. Each record must be a sequence of 25 spot booleans.

This function returns an iterable object, which returns the combinations as tuples with two items. Each item in the tuple is the spot number of a positive spot.

If just *record1* was provided, return all possible positive spot combinations within this record. If both *record1* and *record2* are given, return all possible positive spot combinations between the the two records. If no combinations are possible (i.e. not enough positive spots), the iterable object returns nothing.

An example with one record

```
>>> import setlyze.std
>>> record = (4567,1,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)
>>> combos = setlyze.std.get_spot_combinations_from_record(record[1:])
>>> for spot1,spot2 in combos:
...     print spot1,spot2
...
1 2
1 5
1 15
2 5
2 15
5 15
```

An example with two records

```
>>> import setlyze.std
>>> record1 = (4567,1,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0)
>>> record2 = (4538,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1)
>>> combos = setlyze.std.get_spot_combinations_from_record(record1[1:], record2[1:])
>>> for spot1,spot2 in combos:
...     print spot1,spot2
...
1 3
1 25
2 3
2 25
5 3
5 25
15 3
15 25
```

`setlyze.std.`**`get_spot_coordinate`**`(`*spot_num*`)`

Return a tuple `(row,col)` representing on which row and column a spot with number *spot_num* is located on a 5x5 SETL plate. The possible values for *spot_num* are integers from 1 to 25.

If this is not clear, picture this 5x5 SETL plate and compare it with the examples below:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Some examples:

```
>>> import setlyze.std
>>> setlyze.std.get_spot_coordinate(1)
(1, 1)
>>> setlyze.std.get_spot_coordinate(5)
(1, 5)
>>> setlyze.std.get_spot_coordinate(14)
(3, 4)
>>> setlyze.std.get_spot_coordinate(24)
(5, 4)
```

`setlyze.std.`**`get_spot_position_difference`**`(`*s1*, *s2*`)`

Return a tuple `(h,v)` containing the horizontal and vertical difference (delta x and y) between spots *s1* and *s2*. *s1* and *s2* are spot numbers with possible values from 1 to 25.

Picture a 5x5 grid with spots numbered from 1 to 25:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

If you got two spot numbers that are right next to eachother (say 1 and 2), the horizontal difference would be 1, and the vertical difference 0. A few more examples:

```
>>> print setlyze.std.get_spot_position_difference(3,3)
(0, 0)
>>> print setlyze.std.get_spot_position_difference(1,2)
(1, 0)
>>> print setlyze.std.get_spot_position_difference(3,5)
(2, 0)
>>> print setlyze.std.get_spot_position_difference(6,11)
(0, 1)
>>> print setlyze.std.get_spot_position_difference(9,25)
(1, 3)
>>> print setlyze.std.get_spot_position_difference(1,25)
(4, 4)
```

setlyze.std.**get_spots_from_record**(*record*)

Return a list containing all spot numbers of the positive spots from *record*, a sequence of 25 spot booleans.

A simple usage example

```
>>> import setlyze.std
>>> record = (1,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0)
>>> print setlyze.std.get_spots_from_record(record)
[1, 2, 5, 15]
```

setlyze.std.**make_remarks**(*results*, *attributes*)

Return a remarks string that contains a summary of the results and attributes of a statistical test.

setlyze.std.**mean**(*x*)

Return the arithmetic mean of a sequence of numbers *x*.

A simple example:

```
>>> import setlyze.std
>>> x = [5.91, 1, 10, 19, 22.1, 16, 3.3, 25, 12, 8, 18.5, 17, 23, 2, 7]
>>> setlyze.std.mean(x)
12.654
```

setlyze.std.**median**(*values*)

Return the median of a series of numbers.

setlyze.std.**on_close_progress_dialog**(*delay=0*)

Close the progress dialog. Optionally set a delay of *delay* seconds before it's being closed.

There's no need to call this function manually, as it is called by `on_update_progress_dialog()` when it's needed.

setlyze.std.**on_update_progress_dialog**(*fraction*, *action=None*, *autoclose=True*)

Set the progress dialog's progressbar fraction to `fraction`. The value of *fraction* should be between 0.0 and 1.0. Optionally set the current action to *action*, a short string explaining the current action. Optionally set *autoclose* to automatically close the progress dialog if *fraction* equals `1.0`.

Don't call this function manually; use `update_progress_dialog()` instead.

setlyze.std.**remove_items_from_list**(*a*, *b*)
Remove the items in list *b* from list *a*.

setlyze.std.**shapiro_test**(*x*)
Performs the Shapiro-Wilk test of normality.

This is a wrapper function for the `shapiro.test` function from R. It depends on R and RPy. The latter provides an interface to the R Programming Language.

The data sequence *x* passed to the `shapiro.test` function must contain between 3 and 5000 numberic values. If the length of *x* is below 3, a ValueError is raised. If the length of *x* is above 5000, `random.sample()` is used to get 5000 random values from *x*.

This function returns a dictionary containing the results. Below is the format of the dictionary with example results

```
{
'method': 'Shapiro-Wilk normality test',
'p.value': 6.862712394148655e-08,
'statistic': {'W': 0.75000003111895985}
}
```

See Also:

R Documentation for Shapiro-Wilk Normality Test The R Documentation gives a more extensive documentation of this function, its arguments, usage, etc. To view the documentation, type `help(shapiro.test)` from the R prompt.

setlyze.std.**t_test**(*x*, *y=None*, *alternative='two.sided'*, *mu=0*, *paired=False*, *var_equal=False*, *conf_level=0.94999999999999996*)
Performs one and two sample t-tests on sequences of data.

This is a wrapper function for the `t.test` function from R. It depends on R and RPy. The latter provides an interface to the R Programming Language.

This function returns a dictionary containing the results. Below is the format of the dictionary with example results

```
{
'null.value': {'difference in means': 0},
'method': 'Welch Two Sample t-test',
'p.value': 0.97053139295765201,
'statistic': {'t': -0.037113583386291726},
'estimate': {'mean of y': 2.552142857142857, 'mean of x': 2.5417857142857141},
'conf.int': [-0.56985924418141154, 0.54914495846712563],
'parameter': {'df': 53.965197921982607},
'alternative': 'two.sided'
}
```

See Also:

R Documentation for Student's t-Test The R Documentation gives a more extensive documentation of this function, its arguments, usage, etc. To view the documentation, type `help(t.test)` from the R prompt.

setlyze.std.**uniqify**(*seq*)
Remove all duplicates from a list.

setlyze.std.**update_progress_dialog**(*fraction*, *action=None*, *autoclose=True*)
Set the progress dialog's progressbar fraction to `fraction`. The value of *fraction* should be between 0.0 and

1.0. Optionally set the current action to *action*, a short string explaining the current action. Optionally set *autoclose* to automatically close the progress dialog if *fraction* equals `1.0`.

The `progress-dialog` configuration must be set to an instance of gtk.ProgressBar for this to work. If no progress dialog is set, nothing will happen.

`setlyze.std.`**`wilcox_test`**(*x*, *y=None*, *alternative='two.sided'*, *mu=0*, *paired=False*, *exact=None*, *correct=True*, *conf_int=False*, *conf_level=0.94999999999999996*)

Performs one and two sample Wilcoxon tests on sequences of data; the latter is also known as 'Mann-Whitney' test.

This is a wrapper function for the `wilcox.test` function from R. It depends on R and RPy. The latter provides an interface to the R Programming Language.

This function returns a dictionary containing the results. Below is the format of the dictionary with example results

```
{
'estimate': {'difference in location': -2.0000005809455006},
'null.value': {'location shift': 0},
'p.value': 0.000810583642587086,
'statistic': {'W': 1.0},
'alternative': 'two.sided',
'conf.int': [-3.1200287512799796, -1.2399735289828238],
'parameter': None,
'method': 'Wilcoxon rank sum test with continuity correction'
}
```

See Also:

R Documentation for Wilcoxon Rank Sum and Signed Rank Tests The R Documentation gives a more extensive documentation of this function, its arguments, usage, etc. To view the documentation, type `help(wilcox.test)` from the R prompt.

Analysis Modules The modules described in this chapter all perform on of SETLyze's analysis.

`setlyze.analysis.attraction_inter` — Analysis 2.2, Attraction of species (inter-specific)

Author Serrano Pereira

Release 1.0

Date December 15, 2010

Module Contents

class `setlyze.analysis.attraction_inter.`**`Begin`**

Make all the preparations for analysis 2.2:

- Let the user make the first locations selection.
- Let the user make the first species selection.
- Let the user make the second locations selection.
- Let the user make the second species selection.

When done, start the analysis.

Design Part: 1.5.1

destroy_handler_connections()
Disconnect all signal connections with signal handlers created by this analysis.

handle_application_signals()
Respond to signals emitted by the application.

on_analysis_closed(*obj=None*)
Show the main window and destroy the handler connections.

on_display_report(*sender*)
Display the report in a window.

Design Part: 1.68

on_select_locations(*sender=None*, *data=None*)
Display the window for selecting the locations.

on_select_species(*sender=None*, *data=None*)
Display the window for selecting the species.

on_start_analysis(*sender=None*)
Start the analysis.

class setlyze.analysis.attraction_inter.**Start**
Perform all the calculations for analysis 2.2.

Design Part: 1.5.2

calculate_distances_inter()
Calculate the inter-specific distances for each plate (present in two tables) and save the distances to local_database.spot_distances_observed.

Design Part: 1.27

calculate_distances_inter_expected()
Calculate the expected distances based on the observed inter-specific distances (2.12) and save these to a table in the local database (2.13).

Design Part: 1.69

calculate_significance()
Perform statistical tests to check if the differences between the means of the two sets of distances are statistically significant.

We perform an unpaired Wilcoxon signed-rank test. We use unpaired because the two sets of distances are unrelated. In other words, a distance n in 'observed' is unrelated to distance n in 'expected' (where n is an item number in the lists).

Null hypothesis: The means of the two sets of distances are equal. The specie in question doesn't attract or repel itself.

Alternative hypothesis: The means of the two sets of distances are not equal. The specie in question attracts (mean observed < mean expected) or repels (mean observed > mean expected) itself.

The decision is based on the P-value calculated by the test: P >= alpha level: Null hypothesis. P < alpha level: Alternative hypothesis.

The default value for the alpha level is 0.05 (5%). The default value for the confidence level is 0.95 (95%).

A high number of positive spots on a plate will of course lead to a high P-value. These plates will negatively affect the result of statistical test. To account for this, the tests are performed multiple times. Instead of doing one test on all plates, we group the plates based on the number of positive spots they contain. This results in 24 groups (2 to 25 spots). And we perform the test on each of these groups.

References: 1. N. Millar, Biology statistics made simple using Excel, School Science Review, December 2001, 83(303). 2. P. Dalgaard, Introductory Statistics with R, DOI: 10.1007 / 978-0-387-79054-1_1.

Design Part: 1.24

generate_report()
Generate the analysis report and display the report in a dialog.

Design Part: 1.15

generate_spot_ratio_groups()
Return an iterator that returns the ratio groups.

Each returned group is a list of ratios in the form of two-item tuples.

run()

Call the necessary methods for the analysis in the right order:

- **For the first species selection:**
  - get_record_ids() or get_record_ids()
  - set_species_spots() or set_species_spots()
  - make_plates_unique()
- **For the second species selection:**
  - get_record_ids() or get_record_ids()
  - set_species_spots() or set_species_spots()
  - make_plates_unique()
- calculate_distances_inter()
- calculate_distances_inter_expected()
- calculate_significance()
- generate_report()

Design Part: 1.60

setlyze.analysis.attraction_intra — Analysis 2.1, Attraction of species (intra-specific)

Author Serrano Pereira

Release 1.0

Date December 13, 2010

Module Contents

class setlyze.analysis.attraction_intra.**Begin**

Make all the preparations for analysis 2.1:

- Let the user select the locations.
- Let the user select the species.

When done, start the analysis.

Design Part: 1.4.1

destroy_handler_connections()
Disconnect all signal connections with signal handlers created by this analysis.

handle_application_signals()
Respond to signals emitted by the application.

on_analysis_closed(*obj=None*, *data=None*)
Show the main window and destroy the handler connections.

on_display_report(*sender*)
Display the report in a window.

Design Part: 1.68

on_select_locations(*obj=None*, *data=None*)
Display the window for selecting the locations.

on_select_species(*obj=None*, *data=None*)
Display the window for selecting the species.

start_analysis(*obj=None*, *data=None*)
Start the analysis.

class setlyze.analysis.attraction_intra.**Start**
Perform all the calculations for analysis 2.1.

Design Part: 1.4.2

calculate_distances_intra()
Calculate the intra-specific distances for each plate in the species_spots table and save the distances to a table in the local database.

Design Part: 1.22

calculate_distances_intra_expected()
Calculate the expected distances based on the observed distances and save these to a table in the local database.

Design Part: 1.23

calculate_significance()
Perform statistical tests to check if the differences between the means of the two sets of distances are statistically significant.

We perform an unpaired Wilcoxon signed-rank test. We use unpaired because the two sets of distances are unrelated. In other words, a distance n in 'observed' is unrelated to distance n in 'expected' (where n is an item number in the lists).

Null hypothesis: The means of the two sets of distances are equal. The specie in question doesn't attract or repel itself.

Alternative hypothesis: The means of the two sets of distances are not equal. The specie in question attracts (mean observed < mean expected) or repels (mean observed > mean expected) itself.

The decision is based on the P-value calculated by the test: P >= alpha level: Null hypothesis. P < alpha level: Alternative hypothesis.

The default value for the alpha level is 0.05 (5%). The default value for the confidence level is 0.95 (95%).

A high number of positive spots on a plate will of course lead to a high P-value. These plates will negatively affect the result of statistical test. To account for this, the tests are performed multiple times. Instead of doing one test on all plates, we group the plates based on the number of positive spots they contain. This results in 24 groups (2 to 25 spots). And we perform the test on each of these groups.

References: 1. N. Millar, Biology statistics made simple using Excel, School Science Review, December 2001, 83(303). 2. P. Dalgaard, Introductory Statistics with R, DOI: 10.1007 / 978-0-387-79054-1_1.

Design Part: 1.24

generate_report()
Generate the analysis report and display the report in a dialog.

Design Part: 1.14

run()

Call the necessary methods for the analysis in the right order:

- get_record_ids() or get_record_ids()
- set_species_spots() or set_species_spots()
- make_plates_unique()
- calculate_distances_intra()
- calculate_distances_intra_expected()
- calculate_significance()
- generate_report()

Design Part: 1.59

setlyze.analysis.relations — Analysis 3, Relations between species

Author Serrano Pereira

Release 1.0

Date December 13, 2010

Module Contents

setlyze.analysis.spot_preference — Analysis 1, Spot preference

Author Serrano Pereira, Jonathan den Boer

Release 1.0

Date December 03, 2010

class setlyze.analysis.spot_preference.**Begin**

Make all the preparations for analysis 1:

- Let the user select the locations.
- Let the user select the species.
- Let the user define the plate areas.

When done, start the analysis.

Design Part: 1.3.1

destroy_handler_connections()
Disconnect all signal connections with signal handlers created by this analysis.

handle_application_signals()
Respond to signals emitted by the application.

on_define_plate_areas(*sender=None*, *data=None*)
Display the window for defining the plate areas.

on_display_report(*sender*)
Display the report in a window.

Design Part: 1.68

on_select_locations(*sender=None*, *data=None*)
Display the window for selecting the locations.

on_select_species(*sender=None*, *data=None*)
Display the window for selecting the species.

on_start_analysis(*sender=None*, *data=None*)
Start the analysis.

on_window_closed(*sender=None*, *data=None*)
Show the main window and destroy the handler connections.

class setlyze.analysis.spot_preference.**Start**
Perform the calculations for analysis 1 "Spot Preference".

Design Part: 1.3.2

chi_square_tester(*areas_totals_observed*, *areas_totals_expected*)
Perform the Chi-square test on the observed and expected values.

Design Part: 1.64

generate_report()
Generate the analysis report and display the report in a dialog.

Design Part: 1.13

get_areas_totals_expected()
Return the expected totals for a specie for each plate area.

Design Part: 1.63

get_areas_totals_observed()
Return the observed totals for a specie for each plate area.

Design Part: 1.62

run()
Call the necessary methods for the analysis in the right order and do some data checks:

- get_areas_totals_observed()
- Check if all plate areas totals are zero. If so, abort.
- get_areas_totals_expected()
- chi_square_tester()
- generate_report()

Design Part: 1.58

2.2.2 Coding Style Guidelines

Code layout

Please write PEP-8 compliant code.

One often-missed requirement is that the first line of docstrings should be a self-contained one-sentence summary.

We use 4 space indents for blocks, and never use tab characters.

Trailing white space should be avoided, but is allowed. If possible, configure your text editor to automatically remove trailing spaces and tabs upon saving.

Unix style newlines (LF) are used.

Each file must have a newline at the end of it.

Lines should be no more than 79 characters if at all possible. Use a text editor that has some kind of long line marker indicating the 79 characters boundary. Lines that continue a long statement may be indented in either of two ways:

within the parenthesis or other character that opens the block, e.g.:

```
my_long_method(arg1,
               arg2,
               arg3)
```

or indented by four spaces:

```
my_long_method(arg1,
    arg2,
    arg3)
```

The first is considered clearer by some people; however it can be a bit harder to maintain (e.g. when the method name changes), and it does not work well if the relevant parenthesis is already far to the right. Avoid this:

```
self.legbone.kneebone.shinbone.toebone.shake_it(one,
                                                two,
                                                three)
```

but rather

```
self.legbone.kneebone.shinbone.toebone.shake_it(one,
    two,
    three)
```

or

```
self.legbone.kneebone.shinbone.toebone.shake_it(
    one, two, three)
```

For long lists, we like to add a trailing comma and put the closing character on the following line. This makes it easier to add new items in the future:

```
from setlyze.std import (
    uniqify,
    median,
    distance,
    )
```

There should be spaces between function parameters, but not between the keyword name and the value:

```
call(1, 3, cheese=quark)
```

Module Imports

- Imports should be done at the top-level of the file, unless there is a strong reason to have them lazily loaded when a particular function runs. Import statements have a cost, so try to make sure they don't run inside hot functions.
- Preserved namespaces when importing modules, e.g.:
  - Yes: `import setlyze.config`
  - No: `import setlyze.config as config`

Naming

Functions, methods or members that are relatively private are given a leading underscore prefix.

We prefer class names to be concatenated capital words (`TestCase`) and variables, methods and functions to be lowercase words joined by underscores (`revision_id`, `get_revision`).

For the purposes of naming some names are treated as single compound words: "filename", "revno".

Consider naming classes as nouns and functions/methods as verbs.

Try to avoid using abbreviations in names, because there can be inconsistency if other people use the full name.

Standard Names

`revision_id` not `rev_id` or `revid`

Functions that transform one thing to another should be named `x_to_y` (not `x2y` as occurs in some old code.)

License Statement

SETLyze is released under the GNU General Public License version 3. Each file that's part of SETLyze must have the copyright notice and copying permission statement included at the top of the file after the encoding declaration. So the top of each file should look like this:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  Copyright 2010, GiMaRIS <info@gimaris.com>
#
#  This file is part of SETLyze - A tool for analyzing SETL data.
#
#  SETLyze is free software: you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation, either version 3 of the License, or
#  (at your option) any later version.
#
#  SETLyze is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
```

```
#  You should have received a copy of the GNU General Public License
#  along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

2.3 References

All references used in the documentation are listed here.

2.3.1 Reference List

1. M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator", ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 1998.
2. N. Millar, Biology statistics made simple using Excel, School Science Review, December 2001, 83(303).
3. P. Dalgaard, Introductory Statistics with R, DOI: 10.1007 / 978-0-387-79054-1_1.

2.4 Legal Information

2.4.1 Copyright

Documentation

SETLyze

2.4.2 Links to other websites

This documentation contains links to other websites and resources. The links are provided for convenience only and GiMaRIS is not responsible for the content of any linked websites. The inclusion of any link to a website does not imply endorsement by GiMaRIS of the website or their entities, products or services.

2.4.3 Disclaimer

This documentation was created using Sphinx which is property of their authors.

SETLyze is written in the Python programming language and thus needs the Python interpreter to operate. SETLyze might come in packages bundled with Python and other software tools it requires. The third party software tools bundled with SETLyze are property of their individual authors and are governed by their individual applicable licence. Below is a list of the key third party software tools that SETLyze depends on:

- Python
- GTK+
- PyGTK
- PyCairo
- PyGObject
- setuptools
- R
- RPy
- Python Win32 Extensions
- and probably others I forgot to mention...

2.4.4 Credits

- This legal information is based on Canonical's legal information.
- The *Developer Guide* is based on the Developer Guide for Bazaar.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S

INDEX

D

E

F

G

H

I

M

N

O

P

R

S

T

U

W